Integrated Parallel Simulations and Visualization for Large-scale Weather Applications

A Thesis

Submitted For the Degree of

Doctor of Philosophy

in the Faculty of Engineering

by

Preeti Malakar

Computer Science and Automation

Indian Institute of Science

BANGALORE – 560 012

JULY 2013

To Ma, Dida and in loving memory of Dadu.

Acknowledgements

I thank everyone who has helped me during the last few years. My advisors Prof. Sathish Vadiyar and Prof. Vijay Natarajan have been extremely helpful, supportive and always approachable. Prof. Sathish always ensured that I set clear goals and complete them on time. Prof. Vijay provided crucial feedback on my research. I am inspired by their disciplined and organized style of working. It is because of the several brainstorming sessions with them that this thesis has taken a good shape. I am truly blessed to have such wonderful advisors. Not only did they provide guidance on identifying important research problems and devise effective solutions, but they also gave useful advice on presentation skills. I am thankful to them for their valuable suggestions on how to clearly communicate. I have tried to learn from them how to write effectively and express clearly, and am still learning.

I express my deep gratitude to Prof. Ravi Nanjundiah for valuable discussions on weather and climate research topics. I am grateful to Yogish, Thomas and other colleagues of IBM Research for many useful discussions and suggestions.

I thank system administrator Mr. Pushparaj of CSA for his help. I owe many thanks to Mr. Kiran for helping me with job allocations on IISc's Blue Gene/L on several occasions. I would also like to thank him and Mr. Shankar of SERC for help with the National Knowledge Network between IISc and CDAC. Many thanks to the SERC system administrators for keeping the network up and the clusters running 24x7. I would like to thank Innovative Computing Laboratory of University of Tennessee, Knoxville for providing me access to their cluster. I am also thankful to Teragrid (now XSEDE) and CDAC for letting me run experiments on their supercomputers. I thank Prof. Uday for

his help with job execution on the CSA fist cluster.

I enjoyed two wonderfully-taught courses – Computer Architecture by Prof. Matthew Jacob and Linear Algebra by Prof. R. Vittal Rao. I extend my sincere thanks to Prof. R. Govindarajan and Prof. Y. Narahari, the charimen of SERC and CSA respectively, for creating a wonderful environment for research in these departments. I would also like to thank CSA and SERC staff. Mrs. Lalitha of CSA has been always very helpful. Vaidyaji of CSA deserves thanks for his ever-cheerful disposition. I would also like to thank IISc staff for preserving IISc's serene atmosphere.

I am fortunate to be supported by MHRD scholarship initially and TCS Research Fellowship later. I would also like to take this opportunity to thank my funding sources for travel – Asian Technology Information Program (ATIP), SAP Labs, Microsoft Research, IISc GARP, Tata Consultancy Servies, and IEEE Technical Committee on Parallel Processing.

I thank Meghana and Arnab for their kind help during my initial days at IISc. I enjoyed the several technical and non-technical discussions with several of my labmates – Anurag, Cijo, Dilip, Hari, Nithin, Rajath, Vidya ... I thank GARL and VGL members who gave useful suggestions during my practice presentations. I also thank Pradeesha, Abhijit, Anshuman, Jyotsna, Sujata, Yamini and others for making my stay at IISc enjoyable. Erstwhile Tea-board, Faculty club canteen, CEDT tea outlet and Prakruthi owners deserve thanks for the innumerable cups of coffee. I received constant support from my old friends in Bangalore – Jaya, Kaustav, Ria, Sanjoy, Soumyajit, Trupti ... I thank Pramod for his relentless support and encouraging words and Roshni for being a good friend.

Finally, I would like to thank my family members for their love and blessings. My mother has been my constant source of inspiration in life. I thank her for her love, care and blessings.

Publications

- "An Adaptive Framework for Simulation and Online Remote Visualization of Critical Climate Applications in Resource-constrained Environments", P. Malakar, V. Natarajan, and S. Vadhiyar, in Proceedings of the 2010 ACM/IEEE conference on Supercomputing, November 2010, New Orleans, LA.

- "InSt: An Integrated Steering Framework for Critical Weather Applications", P. Malakar, V. Natarajan, and S. Vadhiyar, in ICCS 2011: Proceedings of the International Conference on Computational Science, June 2011, Singapore.

- "A Divide and Conquer Strategy for Scaling Weather Simulations with Multiple Regions of Interest", P. Malakar, T. George, S. Kumar, R. Mittal, V. Natarajan, Y. Sabharwal, V. Saxena, and S. Vadhiyar, in Proceedings of the 2012 ACM/IEEE conference on Supercomputing, November 2012, Salt Lake City, UT.
  (*Best Student Paper Finalist*)

- "A Diffusion-Based Processor Reallocation Strategy for Tracking Multiple Dynamically Varying Weather Phenomena", P. Malakar, V. Natarajan, S. Vadhiyar and R. Nanjundiah, in Proceedings of the 42^{nd} International Conference on Parallel Processing, October 2013, Lyon, France.

- "End-to-end Adaptive Framework for Efficient Online Visualization of Critical Weather Applications", P. Malakar, V. Natarajan, and S. Vadhiyar.
  To be submitted

- "An Integrated Simulation and Visualization Framework for Tracking Cyclone

Aila", P. Malakar, V. Natarajan, and S. Vadhiyar and R. Nanjundiah, Workshop on HPC in India held in conjunction with 2009 IEEE/ACM conference on Supercomputing, November 2009, Portland OR.

- "An Integrated Simulation and Visualization Framework for Tracking Cyclone Aila", P. Malakar, V. Natarajan, and S. Vadhiyar and R. Nanjundiah, Student Research Symposium, International Conference on High Performance Computing (HiPC 2009), December 2009, Kochi, India.
  (*TCPP Best paper award*).

- "A Coupled Framework for Parallel Simulation and Visualization", P. Malakar, V. Natarajan, and S. Vadhiyar, Grace Hopper Celebration of Women in Computing INDIA (GHC India 2010), December 2010, Bangalore, India.

- "Integrated Parallelization of Computations and Visualization for Large-scale Applications", P. Malakar, V. Natarajan, and S. Vadhiyar, IPDPS 2012 PhD Forum, May 2012, Shanghai.

- "Integrated Parallelization of Computations and Visualization for Large-scale Applications", P. Malakar, V. Natarajan, and S. Vadhiyar, SC Doctoral Dissertation Research Showcase, 2012 ACM/IEEE conference on Supercomputing, November 2012, Salt Lake City, UT.

Abstract

The emergence of the exascale era necessitates development of new techniques to efficiently perform high-performance scientific simulations, online data analysis and on-the-fly visualization. Critical applications like cyclone tracking and earthquake modeling require high-fidelity and high-performance simulations involving large-scale computations and generate huge amounts of data. Faster simulations and simultaneous online data analysis and visualization enable scientists provide real-time guidance to policy makers.

In this thesis, we present a set of techniques for efficient high-fidelity simulations, online data analysis and visualization in environments with varying resource configurations. First, we present a strategy for improving throughput of weather simulations with multiple regions of interest. We propose parallel execution of these nested simulations based on partitioning the 2D process grid into disjoint rectangular regions associated with each subdomain. The process grid partitioning is obtained from a Huffman tree which is constructed from the relative execution times of the subdomains. We propose a novel combination of performance prediction, processor allocation methods and topology-aware mapping of the regions on torus interconnects. We observe up to 33% gain over the default strategy in weather models.

Second, we propose a processor reallocation heuristic that minimizes data redistribution cost while reallocating processors in the case of dynamic regions of interest. This algorithm is based on hierarchical diffusion approach that uses a novel tree reorganization strategy. We have also developed a parallel data analysis algorithm to detect regions of interest within a domain. This helps improve performance of detailed simulations of multiple weather phenomena like depressions and clouds, thereby increasing the lead

time to severe weather phenomena like tornadoes and storm surges. Our method is able to reduce the redistribution time by 25% over a simple partition from scratch method.

We also show that it is important to consider resource constraints like I/O bandwidth, disk space and network bandwidth for continuous simulation and smooth visualization. High simulation rates on modern-day processors combined with high I/O bandwidth can lead to rapid accumulation of data at the simulation site and eventual stalling of simulations. We show that formulating the problem as an optimization problem can determine optimal execution parameters for enabling smooth simulation and visualization. This approach proves beneficial for resource-constrained environments, whereas a naive greedy strategy leads to stalling and disk overflow. Our optimization method provides about 30% higher simulation rate and consumes about 25-50% lesser storage space than a naive greedy approach.

We have then developed an integrated adaptive steering framework, InSt, that analyzes the combined effect of user-driven steering with automatic tuning of application parameters based on resource constraints and the criticality needs of the application to determine the final parameters for the simulations. It is important to allow the climate scientists to steer the ongoing simulation, specially in the case of critical applications. InSt takes into account both the steering inputs of the scientists and the criticality needs of the application.

Finally, we have developed algorithms to minimize the lag between the time when the simulation produces an output frame and the time when the frame is visualized. It is important to reduce the lag so that the scientists can get on-the-fly view of the simulation, and concurrently visualize important events in the simulation. We present most-recent, auto-clustering and adaptive algorithms for reducing lag. The lag-reduction algorithms adapt to the available resource parameters and the number of pending frames to be sent to the visualization site by transferring a representative subset of frames. Our adaptive algorithm reduces lag by 72% and provides 37% larger representativeness than the most-recent for slow networks.

Keywords

Weather simulation; Online visualization; Remote visualization; Adaptivity; Scheduling; Computational steering; Regions of interest; Performance modeling; Processor allocation; Processor reallocation; Redistribution; Topology-aware mapping; Data analysis; Cyclone tracking; Cloud tracking; Representative frame selection

Contents

Acknowledgements i

Publications iii

Abstract v

Keywords vii

1 Introduction 1
- 1.1 Simulation, Data Analysis, and Visualization . . . 1
  - 1.1.1 High-performance Simulation . . . 2
  - 1.1.2 Data Analysis . . . 2
  - 1.1.3 Online Visualization . . . 3
- 1.2 Weather Simulations . . . 3
- 1.3 Motivation . . . 5
- 1.4 Problem Statement . . . 6
- 1.5 Weather Models . . . 8
  - 1.5.1 Weather Research and Forecasting Model . . . 9
  - 1.5.2 Regions of Interest . . . 10
- 1.6 Adaptive Framework . . . 10
- 1.7 Computational Steering . . . 11
- 1.8 Representative Frame Selection . . . 12
- 1.9 Processor Allocation and Reallocation . . . 13
- 1.10 Thesis Outline . . . 14

2 Related Work 16
- 2.1 In-situ Visualization . . . 16
- 2.2 Framework for Visualization . . . 18
- 2.3 Computational Steering . . . 19
- 2.4 Selection of Representative Frames . . . 22
- 2.5 Performance Modeling . . . 24
- 2.6 Mesh Repartitioning and Load Balancing . . . 25
- 2.7 Topology-aware Task Mapping . . . 27
- 2.8 Performance Analysis of WRF . . . 28

3 Improving Throughput of Nested Simulations 30
3.1 Introduction . . . 30
3.1.1 Motivation . . . 31
3.1.2 Problem Statement . . . 32
3.1.3 Results and Contributions . . . 33
3.2 Parallel Execution of Subdomains . . . 34
3.3 Performance Prediction . . . 35
3.4 Processor Allocation . . . 38
3.5 Mapping . . . 40
3.5.1 Topology-oblivious mapping . . . 43
3.5.2 Topology-aware mapping . . . 44
3.6 Resource Allocation and Mapping . . . 47
3.7 Experiments and results . . . 47
3.7.1 Domain Configurations . . . 47
3.7.2 Experimental Setup . . . 49
3.7.3 Improvement in execution time . . . 50
3.7.4 Improvement with topology-aware mapping . . . 54
3.7.5 Effect on high-frequency output simulations . . . 58
3.7.6 Efficiency of our processor allocation and partitioning strategy . . . 59
3.7.7 Scalability and speedup . . . 59
3.8 Summary . . . 61

4 Diffusion-based Repartitioning Strategies 62
4.1 Introduction . . . 62
4.1.1 Challenges . . . 63
4.1.2 Problem Statement . . . 64
4.1.3 Chapter Outline . . . 65
4.2 Tracking Cloud Systems via Parallel Data Analysis . . . 65
4.3 Processor Allocation . . . 70
4.3.1 Partition from scratch . . . 72
4.3.2 Tree-based hierarchical diffusion . . . 74
4.3.3 Dynamic Strategy . . . 80
4.4 Experiments and results . . . 81
4.4.1 Data analysis algorithm . . . 81
4.4.2 Domain Configurations . . . 82
4.4.3 Experimental Setup . . . 83
4.4.4 Improvement in redistribution time . . . 84
4.4.5 Distance between senders and receivers . . . 85
4.4.6 Dynamic Approach . . . 87
4.5 Discussion . . . 88
4.5.1 Use of our Techniques for Other Applications . . . 88
4.5.2 Use of our Techniques for Other Platforms and Interconnects . . . 89
4.6 Summary . . . 90

5 Simultaneous Simulation and Visualization 92
5.1 Introduction . . . 92
5.1.1 Motivation . . . 93
5.1.2 Problem Statement . . . 94
5.1.3 Chapter Outline . . . 96
5.2 Adaptive Integrated Framework . . . 96
5.2.1 Application Manager . . . 96
5.2.2 Job Handler and Simulation Process . . . 98
5.2.3 Frame Sender and Receiver, and Visualization Process . . . 99
5.2.4 Decision Algorithm for the Application Manager . . . 99
5.3 Experiments and Results . . . 108
5.3.1 Weather Application: Tracking Cyclone Aila . . . 109
5.3.2 Framework Implementation . . . 111
5.3.3 Resource Configuration . . . 112
5.3.4 Results . . . 114
5.4 Discussion . . . 130
5.5 Summary . . . 130

6 Integrated Algorithmic and User-driven Steering 132
6.1 Introduction . . . 132
6.1.1 Motivation . . . 133
6.1.2 Problem Statement . . . 133
6.1.3 Chapter Outline . . . 134
6.2 InSt Steering Framework . . . 134
6.2.1 User Interface, SimDaemon and VisDaemon . . . 136
6.2.2 Application Manager . . . 136
6.3 Reconciling User-driven and Algorithmic Steering . . . 137
6.4 Experiments and Results . . . 141
6.4.1 Resource Configuration . . . 141
6.4.2 Weather Model and Cyclone Tracking . . . 142
6.4.3 Framework Implementation . . . 142
6.4.4 Computational Steering Results . . . 143
6.5 Summary . . . 148

7 Reducing Simulation-Visualization Lag on Constrained Networks 150
7.1 Introduction . . . 150
7.1.1 Motivation . . . 150
7.1.2 Problem Statement . . . 151
7.1.3 Chapter Outline . . . 152
7.2 Simulation-Visualization Lag . . . 152
7.3 Reduction of Simulation-Visualization Lag . . . 156
7.3.1 Requirements for Online Visualization . . . 156
7.3.2 Frame Selector . . . 158
7.3.3 Strategies for Selection of Time steps to Reduce the Lag . . . 158

7.4 Experiments and Results 165
7.4.1 Evaluation Strategies 165
7.4.2 High-bandwidth Configuration 167
7.4.3 Medium-bandwidth Configuration 172
7.4.4 Low-bandwidth Configuration 174
7.5 Putting it all together 180
7.6 Summary 181

8 Conclusions and Future Work **182**
8.1 Conclusions 182
8.2 Future Work 184

List of Tables

3.1 Average and maximum improvement in MPI_Wait times on BG/L and BG/P . . . 51
3.2 Sibling configurations for four siblings on BG/L . . . 52
3.3 Sibling configurations and performance improvement for varying nest sizes on up to 8192 BG/P cores. . . . 54
3.4 Execution times (sec) for default, topology-oblivious and topology-aware mappings for various sibling configurations on BG/L. . . . 54
3.5 Execution times (sec) for default, topology-oblivious and topology-aware mappings on 4096 BG/P cores . . . 56

4.1 Processor allocation on 1024 cores . . . 71
4.2 Processor allocation on 1024 cores . . . 73
4.3 Simulation Configurations . . . 83
4.4 Average improvement in redistribution times for synthetic test cases . . . 84

5.1 Illustration of Disk Space Limitation. Weather simulation of grid size 4486×4486 points, 10 km resolution, execution on 16,384 cores with 1.2 seconds of execution time per time step, and I/O bandwidth of about 5 GBps. . . . 95
5.2 Problem Parameters . . . 104
5.3 Resolutions for different Pressure Values . . . 110
5.4 Simulation and Visualization Configurations . . . 113

6.1 Simulation and Visualization Configurations . . . 141

7.1 Simulation and Visualization Configurations . . . 166
7.2 Statistics for rms distance between successive frames for high-bandwidth configuration . . . 169
7.3 Volume between All and Frame Selection Algorithms for high-bandwidth configuration . . . 171
7.4 Statistics for rms distance between successive frames for low-bandwidth configuration . . . 176
7.5 Volume between All and Frame Selection Algorithms for low-bandwidth configuration . . . 178

7.6 Volume and Lag between All and Frame Selection Algorithms for emulated low-bandwidth configuration 179

List of Figures

1.1 Volume rendering of perturbation pressure on 25^{th} May at 20:00 hours (Latitude extents: 10°S - 40°N; Longitude extents: 60°E - 120°E.) 4
1.2 Nest domains within a parent domain. 8

3.1 Visualization of multiple depressions in August 2010 on Pacific Ocean. . 31
3.2 Execution times of nested weather simulations over a 10^7 sq. km. domain on upto 1024 Blue Gene/L cores. 32
3.3 Delaunay Triangulation of points representing known execution times. . . 36
3.4 Partitions of processor space in the ratio of execution times of nested simulations. 39
3.5 Partitions for $k = 3$ when the first partition is along the longer dimension (a) and when it is along the shorter dimension (b). 41
3.6 Communication times in WRF simulations on 1024 Blue Gene/L cores. . 42
3.7 2D to 3D mapping. 43
3.8 Topology-aware mappings. 45
3.9 Multi-level mapping for the 3-sibling configuration in Figure 3.5(a). . . . 46
3.10 Sample domain in South East Asia with four sibling nests at 1.5 km resolution. 48
3.11 Performance improvement of execution time on up to 4096 BG/P cores including and excluding I/O times. 50
3.12 Sibling execution times on 1024 processors on BG/L for four siblings. . . 52
3.13 Sibling execution times on up to 8192 BG/P cores. The legends on the rightmost side show the number of cores. 53
3.14 Percentage improvement in execution times with and without topology-aware mapping on 1024 BG/L cores. 55
3.15 Percentage improvement in MPI_Wait times with and without topology-aware mapping on 1024 BG/L cores. 56
3.16 Percentage reduction in MPI_Wait times with and without topology-aware mapping on 4096 BG/P cores. 57
3.17 Reduction in average number of hops with and without topology-aware mapping on 4096 BG/P cores. 57
3.18 Variation of integration, I/O, and total per iteration times with number of processors on BG/P. 59

3.19 Variation of fraction of integration and I/O times averaged over all the different configurations vs. number of processors on BG/P. 60
3.20 Scalability and speedup of default sequential strategy and our concurrent execution approach. 60

4.1 Tall clouds over the Indian region during the 2005 monsoon season. Image generated from WRF simulation. Darker regions correspond to regions with higher cloud water mixing ratios. 63
4.2 Illustration of processor allocation for nests. 71
4.3 Data redistribution from old to new set of processors assigned to a nest. . 72
4.4 Processor allocation for nests using partition from scratch. 73
4.5 (a) Existing and (b) new processor allocation in the hierarchical diffusion approach. 74
4.6 (a) Existing and (b) new trees in the hierarchical diffusion approach. Predicted execution time ratios of the nests are the weights in the leaf nodes. 75
4.7 Skewed rectangle due to large difference in weights of the two nodes. . . . 76
4.8 Steps of the tree-based hierarchical diffusion algorithm for deleting nests $1, 2, 4$, retaining nests $3, 5$ and adding new nest 6. 79
4.9 Nearest neighbour clustering for our parallel data analysis algorithm. . . 82
4.10 Average hop-bytes for partition from scratch method and tree-based hierarchical approach. X-axis denotes the test case number and Y-axis denotes the hop-bytes. Tree-based hierarchical approach incurs lesser hop-bytes than scratch method. 85
4.11 Percentage overlap between senders and receivers for partition from scratch method and tree-based hierarchical approach. X-axis denotes the test case number and Y-axis denotes the percentage overlap. Tree-based hierarchical approach has more overlap than scratch method. 86
4.12 Execution and redistribution times. 88

5.1 Illustration of simultaneous simulation and remote visualization using stable storage. 93
5.2 Adaptive framework for continuous simulations and online visualization. . 97
5.3 Output Interval and integration time step. 108
5.4 Windspeed visualization in finer resolution nest inside parent domain. . . 110
5.5 Visualization (volume rendering) of Perturbation Pressure at 18:00 hours on 23rd, 24th and 25th May, 2009. 111
5.6 Simulation times with progress in executions for *inter-department* configuration. The graphs show faster rate of simulation for Optimization-based approach. Greedy-Threshold (red) and Optimization-based Approach (blue). 115
5.7 Progress of Greedy-Threshold (red) and Optimization-based Approach (blue) at the visualization end for *inter-department* configuration. Optimization approach shows faster visualization progress. 116

5.8 Free disk space with progress in executions for *inter-department* configuration. The graphs show the decrease in available disk space as simulation progresses in time. Greedy-Threshold (red) and Optimization-based Approach (blue). . . . 117
5.9 Adaptivity of the framework showing variation in number of processors (Left y-axis) and output interval (Right y-axis) for *inter-department* configuration. . . . 118
5.10 Simulation times with progress in executions for *intra-country* configuration. The graphs show faster rate of simulation for Optimization-based approach. Greedy-Threshold (red) and Optimization-based Approach (blue). 119
5.11 Progress of Greedy-Threshold (red) and Optimization-based Approach (blue) at the visualization end for *intra-country* configuration. Optimization approach shows faster visualization progress. . . . 119
5.12 Free disk space with progress in executions for *intra-country* configuration. The graphs show the decrease in available disk space as simulation progresses in time. Greedy-Threshold (red) and Optimization-based Approach (blue). . . . 120
5.13 Adaptivity of the framework showing variation in the number of processors (red, left y-axis) and output interval (green, right y-axis) for *intra-country* configuration. Decision algorithm computes the optimal number of processors and output interval. MPR = 5. . . . 121
5.14 Simulation throughput for *cross-continent moria* configuration. Greedy-Threshold approach leads to stalling. Optimization approach is able to complete simulation without stalling. Greedy-Threshold (red) and Optimization-based Approach (blue). . . . 122
5.15 Progress of Greedy-Threshold (red) and Optimization-based Approach (blue) at the visualization end for *cross-continent moria* configuration. Optimization approach shows faster visualization progress. . . . 123
5.16 Free disk space with progress in executions for *cross-continent moria* configuration. The graphs show the decrease in available disk space as simulation progresses in time. Greedy-Threshold (red) and Optimization-based Approach (blue). . . . 124
5.17 Adaptivity of the framework showing variation in number of processors (Left y-axis) and output interval (Right y-axis) for *cross-continent moria* configuration. . . . 125
5.18 Simulation throughput for *cross-continent Abe* configuration. . . . 126
5.19 Visualization progress for *cross-continent Abe* configuration. . . . 127
5.20 Disk Consumption for *cross-continent Abe* configuration. . . . 127
5.21 Change in number of processors and output interval for *cross-continent Abe* configuration. . . . 128
5.22 Actual rates of simulations for *cross-continent Abe* configuration. The rate constraint of our decision algorithm ensures higher simulation values than the MPR. . . . 129

6.1 InSt: Integrated Steering Framework . . . 135
6.2 Flowchart depicting reconciliation . . . 139
6.3 Simulation (blue) and Visualization (red) progress, and Disk Consumption (green) for *intra-country* configuration with computational steering. Initial WRF resolution = 24 km, MPR = 5. Events $E_1 - E_6$ affect the simulation throughput and the visualization progress as reflected in the graph. . . . 144
6.4 Simulation progress for *inter-country* configuration with computational steering. Initial WRF resolution = 18 km, MPR = 3. Both algorithmic and user-driven steering events ($E_1 - E_4$) affect the simulation throughput as reflected in the graph. . . . 147

7.1 Simulation times (blue) and visualization times (red) showing the simulation-visualization lag. . . . 153
7.2 Simulation and visualization progress. . . . 155
7.3 Frame Selector. . . . 158
7.4 Auto-clustering Strategy. . . . 159
7.5 Simulation and visualization times for high-bandwidth configuration. Visualization curves (except for *all*) are very close to the simulation curve. . 168
7.6 Histogram for *all* for high-bandwidth configuration. . . . 170
7.7 Histogram for the auto-clustering algorithm for high-bandwidth configuration. . . . 170
7.8 Histogram for the most-recent algorithm for high-bandwidth configuration. 170
7.9 Histogram for the adaptive algorithm with lag bound of 30 minutes for high-bandwidth configuration. . . . 171
7.10 Histogram for the adaptive algorithm with lag bound of 20 minutes for high-bandwidth configuration. . . . 171
7.11 Nest position changes for high-bandwidth configuration. . . . 173
7.12 Simulation and visualization times for medium-bandwidth configuration. 174
7.13 Simulation and visualization times for low-bandwidth configuration. . . . 175
7.14 Rms between successively visualized frames. . . . 176
7.15 Histogram for *all* for low-bandwidth configuration for the variable perturbation pressure. . . . 177
7.16 Histogram for the auto-clustering algorithm for low-bandwidth configuration. . . . 177
7.17 Histogram for the most-recent algorithm for the low-bandwidth configuration. . . . 177
7.18 Histogram for the adaptive algorithm with lag bound of 45 minutes for the low-bandwidth configuration. . . . 177
7.19 Nest position changes for low-bandwidth configuration. . . . 180

Chapter 1

Introduction

Computational science complements experimentation and theory, the two traditional modes of scientific discovery. It integrates computing, mathematics and science to solve large and complex real world problems. Computational science helps us tackle fundamental problems in science in areas such as climate modeling, weather forecasting, tsunami prediction, drug discovery, genomic research, aircraft design, automotive design and many others. The complexity of the mathematics and the large number of calculations involved in the modeling of these real world scientific problems necessitate the use of high-performance computing. Increased computational power over the last decade empowers us with unprecedented potential for scientific discovery at unforeseen scale and accelerated throughput.

1.1 Simulation, Data Analysis, and Visualization

Simulation and visualization are the two necessary tools for computational scientists. It is time-consuming and sometimes infeasible to conduct real experiments to validate and verify scientific theories. Furthermore, it is difficult to predict and test the outcome of changing a multitude of experimental parameters. Therefore, computer simulation has emerged as an important tool in understanding scientific phenomena. Visualization is an effective tool to comprehend the simulation output. Online visualization enables on-the-fly view of the simulation and hence enables early comprehension of simulation

output by the scientists. The exponential growth in the amount of data produced by the simulations demands runtime data analysis. On-the-fly data analysis can detect important regions in the simulation domain, which enables automatic refinement of domain for high resolution simulation. This can improve accuracy of the simulation and hence enhance the quality of the simulation. Runtime data analysis can also reduce redundancy in the simulation output by pruning unimportant data. Presenting only significant data to the computational scientists is not only practical but it will also enhance their visual experience. In this thesis we explore the challenges involved in simultaneous simulation, data analysis, and online visualization for weather applications.

1.1.1 High-performance Simulation

Simulation is the numerical evaluation of mathematical models of complex real-world phenomena, often representing a complex and large system. Simulation is an integral part of scientific discovery, and can be applied to various scientific disciplines. For example, molecular dynamics simulations are used to enhance the process of drug discovery [53]. Computational fluid dynamics simulations are used in aerospace applications such as aircraft wing design. Crack propagation simulations are used for prediction of growth and propagation of cracks in structures [143]. In many cases, simulations are required to be done in real time to decide upon the future course of action. For example, air-traffic control system simulates several hours of air traffic to decide in real time the best way to reroute the traffic [67]. Simulations that can give real time predictions of the wake vortices created during the take-off and landing are required in order for the air-traffic control to take appropriate actions [83].

1.1.2 Data Analysis

The increasingly large scale of problems tackled by the present-day simulations generate terabytes of data routinely. Future exascale systems will enable simulations to produce exabytes of data. On-the-fly data analysis can help reduce the data for visualization and provide faster insight [132]. Efficient data analysis techniques can also detect interesting

patterns in the simulation output and recommend high-quality simulations. For example, simulation-time analysis of weather simulations may reveal decrease in atmospheric pressure over certain regions. Early detection of low pressure areas can be beneficial in improving the fidelity of the simulations by increasing simulation resolution over those areas. Furthermore, data analysis can prove useful for selection of important frames[1] for visualization. For example, fewer frames may be sufficient for comprehending the output of weather simulation of a relatively calm day, whereas more number of significant frames will be required for analysis and visualization of turbulent weather conditions. Runtime data analysis can help in adaptive selection of significant frames for visualization.

1.1.3 Online Visualization

Visualization is an effective medium of communicating voluminous information to the human mind [113]. It assists in comprehending the simulation output. Visualization helps in perception of unknown patterns in the data and also facilitates hypothesis formulation. Figure 1.1 shows the visualization of pressure in the Indian subcontinent. The low pressure area (blue-coloured) above the Bay of Bengal prominently conveys the occurrence of a cyclone. Online visualization gives the scientists an on-the-fly view of the simulation. This is very important for real time simulations because the scientists can get real time visualizations of the simulations and hence decision-making process can be expedited.

1.2 Weather Simulations

Weather simulations are important for weather forecasting. Computers have been playing an important role in weather forecasting since the inception of weather modeling. A weather model is a mathematical representation of the atmospheric processes based on physical, biological and chemical principles. The physical processes are described by ordinary and partial differential equations, which are solved numerically using methods

[1] A frame corresponds to the output of a simulation time step.

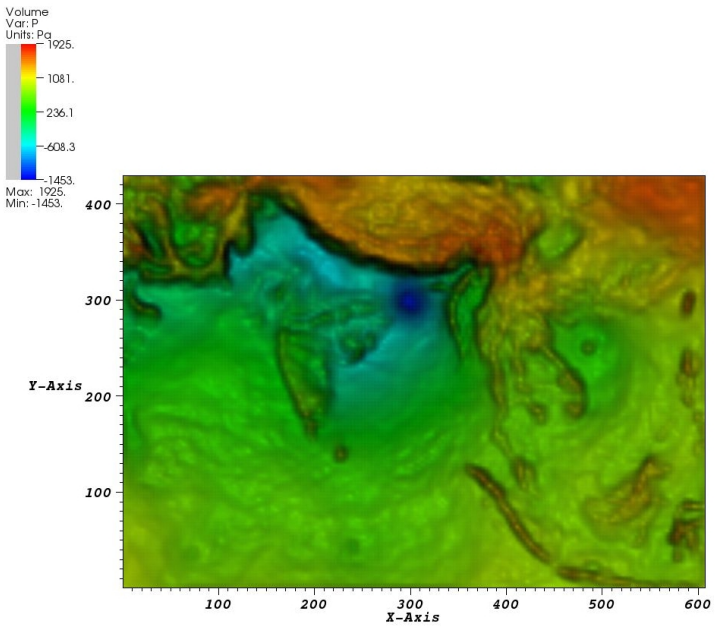


Figure 1.1: Volume rendering of perturbation pressure on 25^{th} May at 20:00 hours (Latitude extents: 10°S - 40°N; Longitude extents: 60°E - 120°E.)

like finite-difference approximations. Various meteorological organizations around the world develop regional-to-global-scale weather models to understand the scientific basis of weather phenomena, potential impacts of climate change and options for mitigation. The Intergovernmental Panel on Climate Change [45] reviews social and economic impact of climate change based on current knowledge of the science of climate change, and recommends possible response strategies. Weather models are important and are being continuously improved for better prediction of extreme events like hurricanes, typhoons, flash floods and tornadoes.

Weather simulations that can predict catastrophic events well in advance can save lives and prevent damages. Rapid variations in seasonal weather patterns, mainly due to change in atmospheric composition, have been recently observed, and are capable of affecting the surface climate of the earth [45]. Drastic changes in the weather can result in unexpected phenomena at unusual times and so it is important to have accurate weather forecasting models and efficient online visualization of the model output. Efficient data analysis techniques can detect abnormal weather patterns and recommend

finer resolution simulations for more accurate weather simulations. Faster output from the weather simulation models are highly desirable because early forecasts of calamities such as hurricanes, cyclones and tornadoes can give sufficient lead time to policy makers so that thousands of lives can be saved.

1.3 Motivation

In this thesis, we address important challenges associated with high performance simulations, data analysis, and online visualization for weather modeling applications. Accurate and timely weather predictions regarding strong cloud cover, heavy rainfall, severe depressions, cyclones, and hurricanes can benefit us tremendously. Weather simulations mainly comprise of solving non-linear higher order partial differential equations numerically. Ongoing research efforts in the climate science and weather community continuously improve the fidelity of weather models by employing higher order numerical methods suitable for solving model equations at high resolution discrete elements. This makes the weather simulations highly compute-intensive.

The sheer amount of simulation output renders traditional methods of data analysis and visualization less useful. As we move forward to the exascale era, challenges in visualization and data analysis of large scale simulation data will increase manifold. New approaches are required for coupling simulation, analysis and visualization of large datasets [25]. Weather simulations can be highly dynamic in nature due to the sudden severe and extreme weather patterns in some areas. The computational requirements of the simulations and the I/O and communication requirements for online visualization of interesting weather patterns evolve dynamically. Hence, an adaptive framework, that can continuously simulate at required resolutions and perform efficient data analysis and online visualization, is of importance. Such a weather simulation and visualization framework should have the following desirable characteristics.

- Perform online visualization so that timely actions can be taken in case of hazards.
- Determine the frequency of visualization and important frames for visualization.

- Analyze simulation output for occurrence of severe events.
- Refine resolution adaptively for greater accuracy.
- Consider resource parameters and application parameters during adaptation.
- Maximize simulation throughput despite many refinement levels to provide high-fidelity simulation.
- Improve performance of simulations executed on diverse supercomputing platforms with different processor interconnection topologies.

The objective of this thesis is to explore some of these challenges. Critical weather applications like cyclone tracking and severe depressions require high-performance simulations and simultaneous online visualization for timely collaborative analysis by a geographically distributed climate science community. An adaptive framework that can alter simulation and resource parameters based on the criticality of the application is necessary for continuous simulation. It is also highly desirable to have efficient online remote visualization, where the output of the simulation is visualized as soon as it is produced. Remote visualization of critical weather events enables joint analysis by geographically distributed climate science community. A steering framework for the weather simulation to enable the climate scientist interact with an ongoing simulation helps reduce the turnaround time for viewing the effect of the interaction. The dynamic nature of weather simulations and the possibility of occurrence of multiple uncertain weather events simultaneously necessitate adaptive refinement of the simulations. Data analysis can highlight possible regions of interest in the simulation, which may require further detailed simulations. It is essential to maximize the throughput of these high-resolution weather simulations so that precautionary actions can be advocated faster.

1.4 Problem Statement

The objective of this thesis is to present a unified framework for simultaneous simulation, data analysis and visualization for weather applications. We have developed techniques

for improving throughput of weather simulations while performing on-the-fly data analysis and visualization. This thesis aims to integrate strategies for high-performance simulations with adaptive refinement and efficient online visualization for weather applications in any simulation-visualization environment.

Firstly, we have developed an adaptive framework for continuous simulation and online remote visualization of critical weather events in resource-constrained environments. The framework recommends optimal simulation and visualization parameters under constraints imposed by the application and resource dynamics like network bandwidth and storage space. The framework attempts to maximize the rate of simulation based on the resource characteristics of the simulation-visualization environment. Secondly, we have developed an integrated user-driven and algorithmic steering framework InSt for critical weather applications that can integrate steering inputs of the scientists, criticality of the application and resource constraints to decide the optimal simulation parameters for steady simulation and online visualization. Such an integrated framework allows the scientists to alter simulation parameters while the framework reconciles between the user inputs and the algorithmic recommendations for continuous and faster simulation and visualization. Thirdly, we devised algorithms for selection of important frames from the simulation output that are representative of the entire output. Representative frame selection enables efficient online visualization of the simulation output. In the next part of the thesis, we propose methodologies for high performance simulation of multiple events. We propose simultaneous simulations of the multiple events, and performance modeling and processor allocation strategies for the same. We also propose novel topology-aware mapping heuristics for the processes executing the multiple simulations. Additionally, we present rescheduling algorithms for dynamic weather events that occur concurrently. The rescheduling algorithms consider the existing processor allocation and recommend new subset of processors for the multiple simultaneous simulations so that the data redistribution cost is minimized.

We have demonstrated these techniques for simulations, data analysis and visualization for critical weather applications like tracking cyclones and cumulonimbus clouds in

the Indian region and multiple depressions over the Pacific and South East Asian regions.

Thus, the thesis comprehensively considers resource characteristics and application dynamics to adaptively refine simulations and continuously perform high-throughput simulations and on-the-fly visualization. The following sections describe some of the key concepts used in this thesis and give an overview of our work.

1.5 Weather Models

Weather simulation models involve a *domain* of simulation which represents the chosen geographical area at a certain horizontal and vertical resolution. The horizontal and vertical dimensions of the domain represent the number of discrete grid cells in the domain. Model resolution is defined as the distance between grid points in the model. The parent domain is generally simulated at a coarser resolution because finer (i.e., higher) resolution implies more number of grid cells and hence more number of computations. A *nest* is a finer resolution model run embedded within a coarser resolution parent domain. Nested simulation is typically spawned over a smaller area representing the region of interest. There can be many nested domains, called *siblings* within a parent domain as shown in Figure 1.2.

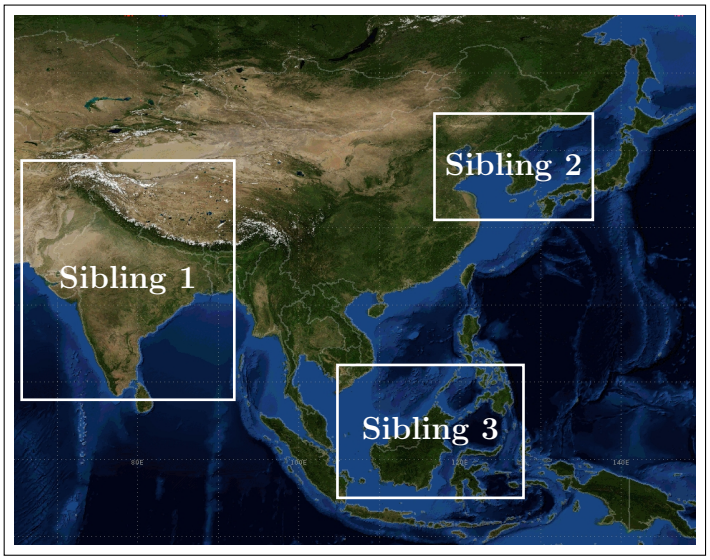


Figure 1.2: Nest domains within a parent domain.

1.5.1 Weather Research and Forecasting Model

In our work, we have used a popular open source weather forecasting tool called Weather Research and Forecast Model (WRF) version 3.3.1 [80, 121]. It is a regional and mesoscale numerical weather prediction model and is used by weather agencies all over the world. The WRF-ARW solves the fully-compressible, non-hydrostatic Euler equations using a finite-difference scheme on an Arakawa C-grid staggering in the horizontal plane and a terrain following, dry hydrostatic pressure vertical coordinate. Integration in time is performed using a time-split method with a 2nd or 3rd-order Runge-Kutta scheme with a smaller time step for acoustic and gravity-wave modes. The model supports static and moving nested grid configurations. Various options like periodic, open, symmetric and specified are available for the lateral boundary conditions [79, 121].

WRF has been designed and written to perform well on massively parallel computers. It is written in Fortran90 and can be built in serial, parallel (MPI) and mixed-mode (OpenMP and MPI) forms, simply by choosing the appropriate option during the configure process. The model code has been analyzed extensively for a number of HPC platforms in the past but the study has been mostly restricted to a simple single domain model configuration [80, 114, 140]. However, for most practical applications, the weather community can immensely benefit from high-resolution nested domains. Nested simulations enable simulations of smaller regions of the parent domain at greater depth and hence higher fidelity. For example, high-resolution nested simulation can be spawned over the lowest pressure region in the domain in the case of hurricanes and cyclones. The data for the finer resolution nested domains are generally interpolated from the coarser domain by a process called *forcing*. In a two-way nest integration, the finer grid solution overwrites the coarser grid solution for the coarse grid points that lie inside the finer grid by a process called *feedback* in WRF [81, 121].

WRF outputs data in the form of NetCDF [100] files. Each NetCDF file contains the values of various meteorological variables for each of the grid points in the domain for a specified number of simulation time steps.

1.5.2 Regions of Interest

Finer resolution run can result in high-fidelity weather simulations. However, higher resolution model run requires smaller time step and more number of grid points and hence increases the amount of computation. Nesting [81] enables finer resolution runs over smaller regions of interest in order to avoid expensive computation across the entire parent domain. There can be multiple nests formed over multiple regions of interest, either dynamically or statically. There can be a single region of interest as in the case of tracking cyclones or hurricanes. In this case, a single nested simulation is spawned over the cyclone or hurricane. This will typically be a moving nest because of the moving cyclone. There can be multiple regions of interest like in the case of depressions or cloud formations. In this case, a nested simulation is spawned for each region of interest. The first part of this thesis deals with single region of interest and in the later part of the thesis, we extend to multiple regions of interest.

1.6 Adaptive Framework

Critical weather applications like cyclone tracking and earthquake modeling require high-performance simulations and online visualization simultaneously performed with the simulations for timely analysis. *In-situ* visualization directly uses the simulation data from physical memory for visualization, without requiring any intermediate storage of the data. However, users often require the entire simulation data for further analysis. In-situ visualization allows limited interaction with the full mesh. So, we transfer the entire simulation data to a remote visualization site. Here remote implies that the simulation and visualization processors do not share physical memory and can be connected by very high to very low bandwidth networks. Remote visualization of critical weather events enables joint analysis by geographically distributed climate science community.

In remote online visualization, resource constraints like limited storage space, low network bandwidth and low I/O bandwidth can limit the effectiveness of online visualization. In our work, we have developed an adaptive framework that simultaneously

performs numerical simulations and online remote visualization of critical weather applications in resource-constrained environments. Considering application dynamics like intensity of the weather phenomenon and resource dynamics like available storage space and network bandwidth, the framework adapts various application parameters like simulation resolution, progress rate of simulation and amount of data for visualization and resource parameters like number of processors to be allocated. Such an adaptive framework is highly essential for optimizing the rate of simulation and the rate of visualization considering the resource characteristics of the simulation-visualization environment. Our framework, that continuously adapts to the application dynamics, is integral to weather simulations, where occurrence of sudden events require spawning of nested simulations and dynamic nature of the events require alteration in application parameters.

We present two algorithms for the decision making process in the framework. These algorithms output the processor allocation for simulation and the frequency of data for visualization based on the resource and application parameters. The first algorithm is based on a naïve greedy approach. It initially tries to maximize the throughput of simulation and the number of frames for visualization. It reactively responds to resource constraints and hence sometimes leads to stalling of the simulation. The second algorithm solves a linear optimization problem to recommend optimal parameters for the simulation and visualization. This is a proactive approach and tries to optimize the execution parameters during the entire simulation. We have experimented with varying network bandwidths between the simulation and visualization sites. Specifically, we have conducted inter-department, intra-country and cross-country simulation-visualization experiments. We show that our optimization method is able to provide about 30% higher simulation rate and consumes about 25-50% lesser storage space than the greedy approach.

1.7 Computational Steering

Computational steering is a process by which the user can interactively explore a simulation during execution based on the visualization of the current results. It is a well-studied

approach that allows the user to give feedback to the simulation based on the visualization [33, 59, 95, 131]. *Algorithmic steering* uses an algorithm to decide application parameters to improve system and application performance [131]. *User-driven steering* takes inputs from the user to steer the simulation.

Steering of weather simulations is essential for effective and timely analysis by climate scientists. Steering enables the climate scientists to refine the ongoing simulation and see the effects of refinement immediately. A weather simulation steering framework can also enable scientists to modify other execution parameters like location of nested simulation and frequency of visualization. However, in remote online visualization, it is also important to consider the resource constraints along with the inputs of the scientists. We have developed an integrated user-driven and automated steering framework InSt for simulations, online remote visualization, and analysis for critical weather applications. InSt provides the user control over various application parameters including region of interest, resolution of simulation, and frequency of data for visualization.

InSt reconciles between the parameters decided by the framework algorithm and the input parameters of the scientists. The scientists may specify high progress rate of simulation, high resolution and high output frequency of visualization. However, depending on the resource characteristics like I/O bandwidth, speed of computation, available storage space and network bandwidth, it may not be feasible to satisfy all requirements of the user. In this scenario, our framework can give options to the user or override some of the user inputs in order to maximize most important simulation parameters like the rate of simulation.

1.8 Representative Frame Selection

In online weather simulations, it is important to visualize the simulation output as soon as it is produced. However, the output interval is decided by our framework depending on resource constraints like storage space and network bandwidth. Very high frequency of output can lead to substantial delay between the simulation time and the visualization time if the simulation-visualization network bandwidth is low. The challenge in

online remote visualization is to minimize the *lag* between the time when the simulation produces an output frame and the time when the frame is visualized. It is important to reduce the lag so that the scientists can get on-the-fly view of the simulation. We have developed algorithms to minimize the simulation-visualization lag and concurrently visualize important events in the simulation.

We present algorithms for online data analysis to decide representative frames to be sent to the visualization site. There may be redundancy in frames when the interval between two consecutive frames output by the simulation is low. For example, if the simulation outputs frames every 3 minutes of simulation, there may not be significant difference between two consecutive frames. In such cases, we transfer the most representative frames for visualization and drop the lesser important frames. This also reduces the simulation-visualization lag.

The number of pending frames at the simulation site depends on the application and resource parameters. Our algorithms transfer a subset of significant frames to the visualization site depending on the lag at the simulation site. The *most-recent* algorithm transfers the most recently simulated frame for maximum reduction in the simulation-visualization lag. The *auto-clustering* frame selection algorithm clusters the pending frames based on modified k-means algorithm for temporal clustering. Additionally, the *adaptive* algorithm dynamically decides the information content in the frames to be transferred, depending on the user-specified lag bound.

1.9 Processor Allocation and Reallocation

Accurate and timely prediction of weather phenomena, such as hurricanes and flash floods, require high-fidelity compute-intensive simulations of multiple finer regions of interest within a coarse simulation domain. The high-resolution nested simulations improve accuracy of the simulation output at the cost of more number of computations. Current weather simulation models execute these nested simulations sequentially using all the available processors. This may lead to sub-optimal performance depending on the size of the nest and the number of available processors, due to sub-linear scalability

of the weather models. We present a strategy for parallel execution of multiple nested domain simulations to improve the overall performance.

Simultaneous executions of nested simulations imply allocating a disjoint subset of the maximum number of processors for each nested simulation. We partition the 2D process grid into disjoint rectangular regions associated with each nested domain. The processor allocation is based on the estimated execution time of the nested simulation. For this, we present our strategy of performance modeling the nested simulations. We also present a rectangular subdivision algorithm for allocating a sub-rectangle of the 2D process grid to a nested simulation. The nested simulations are executed on a subset of the process grid and the parent simulation is executed on the entire process grid. To optimize the communication times for both nested and parent simulations, we propose topology-aware mapping of processes to the processor cores on the network topology. This helps reduce the overall communication times and improves the overall performance.

Many meteorological phenomena occur at different locations simultaneously. These phenomena vary temporally and spatially. Hence nested simulations for these phenomena are dynamically spawned and dissolved. Dynamic variation in the number of these nests require efficient processor reallocation strategies. We have developed strategies for efficient partitioning and repartitioning of the nests among the processors. As a case study, we consider an application of tracking multiple organized cloud clusters in tropical weather systems. We present a parallel data analysis algorithm to detect such clouds. We have developed a novel tree reorganization based hierarchical diffusion algorithm that reallocates processors for the nests such that the redistribution cost is less.

1.10 Thesis Outline

The rest of the thesis is organized as follows. In Chapter 2, we review prior work related to simultaneous simulation and visualization, computational steering, performance modeling, load balancing, repartitioning, and topology-aware mapping. In Chapter 3, we describe techniques for performance modeling, processor allocation and topology-aware mapping for providing high throughput for nested simulations. In Chapter 4, we

introduce diffusion-based repartitioning strategies to reallocate processors for dynamic nests. In Chapter 5, we discuss simultaneous simulation and online remote visualization for critical weather applications in resource-constrained environments. In Chapter 6, we describe an integrated steering framework for weather simulations. In Chapter 7, we discuss algorithms for efficient online visualization. In Chapter 8, we provide some concluding remarks and directions on future work.

Chapter 2

Related Work

In this chapter, we first describe some earlier work related to simultaneous simulation and visualization. Next, we describe related work in computational steering of large-scale simulations and visualizations of scientific data. The next section describes prior work on selection of representative frames. We then review prior work on performance modeling, load balancing and partitioning, and topology-aware mapping. Finally, we present an overview of earlier work on performance analysis of WRF.

2.1 In-situ Visualization

The analysis and study of time-varying output data, obtained from numerical simulations, is integral to scientific discovery. The conventional approach of visualizing simulation output is an *offline* "post-processing" step after the simulation is completed. There have been strategies on offline visualization for earthquake simulations [146]. However, for critical applications like weather forecasting, *online* visualization strategies play an important role.

Simulation-time analysis and visualization using the same resources as the simulation is called in-situ visualization. It has been extensively studied in the recent times [28, 30, 75, 76, 123, 129, 136]. Tu et al. [129] and Ma et al. [75, 76] proposed tightly-coupled execution of the simulation and visualization components where a simulation time step is followed by its visualization on the same set of processors. The

authors use a common data structure for both simulation and visualization. A drawback of this approach is that a single data structure is often not efficient for both simulation and visualization. Additionally, all algorithms in the visualization pipeline may not scale as efficiently as the simulation, which limits the effectiveness of tight coupling [30]. They have considered the simulation of earthquake ground motion. The simulation and visualization cycles alternate executions on the same set of processors using the same shared data, minimizing the cost of communication from the simulation to the visualization component. Due to alternate executions, the simulation component is stalled while the visualization is performed. Since high-fidelity simulations demand more computations and more resources than visualization, it is undesirable to stall the simulation. Stalling simulation while the visualization component runs would cause the subsequent output of simulation to be produced after a considerable delay.

Another approach to in-situ visualization is to use shared physical memory [28, 32] or network [147] by the simulation and visualization phases. This alleviates the time taken to write data onto the secondary memory but requires huge amount of physical memory for accurate simulations. In [28], the authors use 1 TB of shared memory in their experimental setup. They have conducted weather simulation of the 2005 Hurricane season using a global climate model. Though this approach decouples the set of processors on which each component runs, it requires large amount of shared memory. A long-running high-resolution simulation with high output frequency executed on modern-era petaflop supercomputers is highly likely to suffer from unavailability of memory to store the simulation output. Also, the longer the scientist interacts with the current time step data, the lesser will be the time taken to overflow the physical memory.

All the above efforts consider critical weather applications in tightly-coupled environments. An important drawback of all the above approaches is that the visualization expert cannot interact with the full mesh, going back and forth in time. In our model of continuous simulation and remote visualization, we aim to perform online visualization without the requirement of a huge expensive physical memory and giving full flexibility to the scientist to interact with the full mesh, going back and forth in time. Ours

is the first work that adaptively performs simultaneous simulations and online remote visualization for these applications.

Though the transfer time can vary from seconds to minutes depending on the amount of data sent and the network bandwidth, our approach of transferring the simulation output to another site for visualization is a viable option for petascale applications because of limited physical memory at the simulation site. Current supercomputers like Kraken [60] have 16 GB physical memory per node whereas the requirements for a coupled simulation and in-situ visualization for a high-fidelity weather application can exceed the available physical memory per process. Additionally, the simulation time increases with the fidelity of the simulation. The option of stalling the simulation to perform in-situ visualization is not considered in our work.

Another approach is to perform visualization calculations at the site of the simulations and send the visualization results (images) to a different site for visualization by the scientist. The images are typically much smaller than the compressed raw data, and hence result in smaller costs due to data transfers to the visualization site. This can also help in avoid using data reduction techniques discussed in this thesis. However, sending only the images will allow the user to receive only a predecided view of the data and not interact with the full data. If the user wants to modify the visualization pipeline, the request has to be sent to the simulation site, which in turn will increase the interaction time of the scientist.

2.2 Framework for Visualization

Whitlock et al. [136] introduced a simulation library Libsim to enable in-situ analysis and visualization using VisIt. Libsim reads simulation data from physical memory when required by the VisIt server. Once the user creates a plot and VisIt executes the plot's data flow network, the data access functions of the simulation are invoked to retrieve the required data. The simulation code needs to be modified to incorporate these data access functions. Fabian et al. [30] introduces the ParaView coprocessing library to integrate visualization with simulation codes. The temporal frequency at which the

simulation data is passed for visualization and the visualization pipeline configuration like orientation of slice plane is predecided before the current simulation time step output is processed by the library. In our work, the user can perform the full analysis on-the-fly. This is helpful because the analysis scenarios are often not known a priori, especially in critical weather applications.

Wu et al. [141] have proposed a framework for remote visualization to self-adapt according to visualization needs and time-varying network and node conditions. They group the modules that implement visualization and networking subtasks, and map them onto computing nodes with disparate computing capabilities and network connections. Using estimates for communication and processing times of subtasks, they have presented a polynomial-time algorithm to compute a decomposition and mapping to achieve minimum end-to-end delay of the visualization pipeline. Moad et al. [82] use parallel VTK to visualize WRF data sets at an interactive rate from the storage. In our work, we aim to achieve minimum end-to-end delay of the entire simulation-visualization pipeline. Our framework considers both application and resource dynamics to adapt various simulation and visualization parameters like simulation resolution and amount of data for visualization.

2.3 Computational Steering

Computational steering enables scientists to interact with a running simulation and provide feedback or change simulation parameters. Computational steering has been extensively studied over the past several years [33, 36, 37, 47, 71, 92, 94, 95, 102, 103, 116, 131, 150]. A variety of steering systems have emerged like SCIRun [93], CUMULVS [33], Discover [71], VASE [47] etc. A taxonomy of steering systems and tools can be found in [37, 95]. Different kinds of steering have been used. *Exploratory steering* allows the scientists to control the execution of long-running, resource-intensive applications for application exploration. For example, in the work by Shenfield et al. [116], they propose a steering system that allows the user to monitor or alter execution parameters of multi-objective evolutionary algorithm for engineering design. *Performance steering* allows

scientists to change application parameters to improve application performance. *Algorithmic steering* uses an algorithm to decide application parameters to improve system and application performance [131]. For example, the work by Ribler et al. [101] proposes using fuzzy logic to adapt to changing application resource demands and system resource availability.

Computational steering has been applied to different kinds of applications like molecular dynamics simulation, biological applications, astrophysics, atmospheric simulations, computational fluid dynamics etc. [10, 42, 44, 50, 52, 83, 134]. These frameworks were mainly developed for exploratory steering in order to change simulation parameters interactively and thereafter, visualizing the simulation output with the new parameters. Johnson et al. have applied their steering model to cardiac defibrillation simulation and defibrillation electrode design [52]. They allow designing of bioelectric devices in an interactive graphical environment. Beazley et al. [10] have presented a lightweight steering tool based on scripting languages for large-scale molecular dynamics simulations. Jean et al. [50] have used Falcon [36] to develop an integrated approach for online monitoring, steering, visualization of atmospheric simulations. The work in [134] describes an exploratory simulation environment for Smoothed Particle Hydrodynamics simulation of astrophysical phenomena in areas such as star formation and evolution. We have also built an user interface for exploratory steering by climate scientists. The interface allows scientists to input simulation parameters like resolution and output frequency as described in Chapter 6.

There are some steering systems which do performance steering [26, 33, 36, 59, 99, 101, 102, 125]. CUMULVS [33, 59] provides the user with a viewer and steering interface for modifying the application's computational parameters and improving application performance. Falcon [36] is a system for online monitoring and steering of large-scale parallel programs with the goal of improving the application's performance or affect its execution behaviour. Autopilot [101, 102] is about dynamically adapting to changing application resource demands and system resource availability. They use real-time

performance data to steer the system. In [116], the authors show that steering of multi-objective evolutionary algorithm improves quality of the solutions. In [26], the authors have built a tuning framework for automatic reconfiguration of the application through tunable parameters. Their work on performance steering is about automatic mechanisms for adapting parallel programs for better performance. The runtime system is adaptable since it can dynamically reconfigure the tunable parameters. We have worked on automatic tuning for smooth and continuous simulation and visualization. In Chapter 5, we have shown dynamic adaptation of application and system parameters based on the resource constraints for critical weather applications. Various resource constraints including available storage space, network bandwidth and I/O bandwidth are considered to be able to do continuous visualization with maximum simulation rate.

Pablo [99] performance analysis environment is a system for real-time adaptive control system that configures resources based on observed application behavior. This dynamic performance instrumentation of applications is advantageous over the traditional post-analysis of poor performance of applications. Their framework can adapt to temporally varying application resource demands. Active Harmony [125] deals with automated performance tuning. It allows runtime tuning of application parameters like read-ahead parameter, switching of algorithms etc. They have developed runtime tuning algorithms to intelligently set the parameters at runtime to tune the application performance. The most common performance metrics considered are CPU time or memory space used. The main objective of our algorithmic steering framework is to perform on-the-fly remote visualization simultaneously with the simulations adapting to changing resource configurations.

There have been some efforts on remote computational steering [15, 142]. Wu et al. [142] focus on computational steering in distributed environments. They formulate visualization pipeline configuration problems with the objective of maximizing the frame rate. They show that these problems are NP-complete and propose heuristics based on a dynamic programming approach. In the work by Brooke et al. [15], the authors show how geographically distributed teams can view simultaneously a visualization of a

running simulation and can steer the application.

Jean et al. [50] have developed an integrated approach for online monitoring, steering and visualization of atmospheric simulations using Falcon [36]. To our knowledge, this is the only work on steering of climate applications prior to our work. They have done simulation of physical and chemical interactions in the ocean and atmosphere. In order to evaluate different parameter settings by the user, they have built a steering interface to let the user dynamically modify the application execution. We have worked on algorithmic steering and user-driven steering of weather simulations. In Chapter 6 we describe our efforts on reconciliation of algorithmic and user-driven steering. Our work differs from the above efforts because we not only let the user interactively steer the application, but we also let the system override the user decision in order to meet resource constraints and application performance of critical weather applications.

2.4 Selection of Representative Frames

Selection of frames to visualize important events may appear similar to keyframe selection, which has been widely studied for video summarization [72, 73, 149]. However, it is not directly applicable to selection of simulation output for online visualization of critical applications because in this case we want to select the best set of representative frames taking into account the current lag. In real-time video transmission, each frame may be encoded using different bit rate depending on the importance of the frame or on the perceptual quality required for that frame [90]. Varying number of bits are used on different parts of the video sequence to maximize the quality delivered to the end user. Often these videos are pre-analyzed to allocate the right number of bits to each frame, depending on the frame content. In our application, capturing the key events is more preferable unlike video transmission, where continuity of the video clip is a constraint. Frame skipping may lead to perceptual disturbance in on-demand video transmission [91], whereas for our application skipping a frame may be acceptable.

In [115], the authors exploit data coherency between consecutive simulation time steps. They calculate the differential information among a sequence of simulation data.

At each time step the differential information is used to compute the locations of pixels that need updating. They have found that when the percentage of changed elements is over 50%, the performance of their differential volume rendering method is worse than the regular ray casting method. This method is useful when a large percentage of values remain constant from one time step to the next time step. In our case, we found that there are more than 80% changed elements between successive time steps. Hence sending differential information will lead to diminishing returns in our case. Since the differential information is more, it will not result in saving time by sending differential frames and hence will not reduce the lag.

In [135], Wang et al. derive an importance measure for each spatial block in the joint feature-temporal space of the data based on the formulation of conditional entropy. They use this importance measure to identify the most essential aspects of time-varying data. Based on the clustering of the importance curves of all the volume blocks, they suggest effective visualization techniques to reveal the important aspects of time-varying data. To calculate entropy, they form multidimensional histograms from the data block statistics. Their results show that the histogram calculation takes 19 hours for a $960 \times 660 \times 360$ volume for 222 time steps with block size of $48 \times 33 \times 18$. Such a large amount of time for selection of important frames from a running simulation is clearly unsuitable for efficient online visualization.

Patro et al. [96] have presented a method to measure saliency in molecular dynamics simulation data. They use the saliency function to guide the selection of representative and anomalous time steps for summarization of simulations. Their keyframe selection method for molecular dynamics simulations considers the atom positions to determine saliency of the atoms in a time step and then aggregate information to find saliency of the time step. Their saliency function for keyframe selection is specific to molecular dynamic simulations where the changes in atomic positions between consecutive time steps of a simulation are dominated by Brownian motion and interesting and purposeful molecular conformational changes occur only over larger time scales.

Kendall et al. [51, 57] demonstrate optimized analysis techniques for multivariate

pattern discovery from terascale climate data stored on disk in NetCDF format. There are also other works [56, 105] which demonstrate offline approach of analyzing climate data, we perform online weather data analysis in our work.

2.5 Performance Modeling

There has been a lot of research on performance modeling and prediction of application execution times on HPC systems. Allan et al. [5] compare tools for predicting performance on a range of architectures and applications. Dimemas [6, 104] predicts performance of MPI applications on distributed and grid environments. It requires real execution of the application to capture CPU bursts and communication pattern, and the system parameters. PACE [16, 49, 86] is a tool for performance prediction of applications running on parallel systems. Application model is generated using static source code analysis to infer the control flow and communication pattern of the application. PACE takes as input the application model and the hardware configuration of the resource on which the application will run, and predicts the application performance on the given resources.

PHANTOM [148] uses deterministic replay techniques to measure sequential computation time of an application by executing each process one by one on a single node of target architecture. They show results for up to 1024 cores. This approach does not effectively capture the effect of network topology on communication times. It is also not easily scalable to higher number of processors because of replaying every process. Barnes et al. [8] use a second-order polynomial regression model to fit execution time on large number of processors collected from many instrumented runs for different input configurations on fewer processors. This requires many training runs and does not capture the non-linearity in many applications. Dinda et al. [24] describe a system that predicts running time of compute-bound tasks on different hosts to enable real-time scheduling decisions. They use linear time series models for host load predictions. Many existing works [7, 130] develop analytical models of applications to predict their performance.

Barker et al. [7] present an analytical performance model for a hydrodynamics application. They model the computation times and various communication times of the application. Such analytical modeling requires detailed knowledge of the application.

There exists some prior work on weather forecasting applications. Kerbyson et al. [58] describe an analytical performance model parameterized in terms of WRF application inputs (grid size, computation load per grid point, etc.) and system parameters (processor count, network topology, latencies and bandwidth, etc.). This model was developed via a careful manual inspection of the dynamic execution behavior of the WRF application and was subsequently validated using performance measurements on two systems - an AMD Opteron cluster and IBM Blue Gene/L. Delgado et al. [23] (extending their earlier work in [109]) describe a regression-based approach for modeling WRF performance on systems with less than 256 processors, but their primary focus is on capturing the system related factors such as clock speed, network bandwidth, which they do via a multiplicative effects model. Unlike previous efforts, our performance prediction model uses linear interpolation based on the grid size and aspect ratio of the grid using actual simulation times obtained from a small set of profiling WRF runs. Our prediction approach does not explicitly incorporate system related factors. Further, [58] and [23] only focus on predicting performance for single domain configurations whereas we use performance prediction results for partitioning the available processor space into different sizes for concurrent executions of multiple nested domains.

2.6 Mesh Repartitioning and Load Balancing

Many of the existing efforts in mesh adaptation and repartitioning have been in the domain of Adaptive Mesh Refinement (AMR) applications [87, 118], in which repartitioning and dynamic load balancing are critical components. Schloegel et al. present a multilevel diffusion scheme for repartitioning adaptive meshes [112]. A commonly used strategy for mesh repartitioning is to map the problem to graph partitioning and attempt to minimize the edge-cut representing the amount of communication between the partitions [55, 87, 112]. There also exists approaches such as recursive bisection [117]

that cater to both regular and irregular domains. Recent work [84] addresses scenarios involving processors with heterogeneous computational capacity and communication bandwidth that requires partitioning the domain of interest into multiple subdomains in proportion to the computational capacity of the processors. In our case, the partitions are the nests and there is no communication between the nests. Hence the graph partitioning heuristics are not directly applicable in our case. In contrast to earlier work, the application-specific constraints in our work require that a single nested domain is assigned to a rectangular processor grid with constraints on the aspect ratio of the rectangle to achieve optimal performance. We partition the processor space into multiple disjoint rectangular grids that are assigned to the individual nested domains so that the computational capacity is proportional to the domain workload.

Moreover, existing efforts in AMR [65, 87] and irregular mesh applications [112] perform repartitioning of adaptively refined meshes primarily to achieve dynamic load balancing. Diffusion methods have been used earlier [41, 54, 74, 78] to achieve dynamic load balancing. In our work, load balancing is implicitly achieved by partitioning the processor space into multiple disjoint rectangular grids that are assigned to the individual nested domains in proportion to the domain workloads. The primary focus of our work is repartitioning the rectangular processor grid into sub-rectangles while minimizing the data redistribution cost when nests dynamically appear and disappear. In our case, a rectangular process grid needs to be allocated for each nested domain and one processor executes a region of a nested domain. The new processor allocation after the redistribution is also a sub-rectangle of the rectangular process grid for the parent simulation. Thus we require to reform the existing rectangles such that there is maximum overlap between the old and new process grids (sub-rectangles) for the same nest in consecutive adaptation points.

Furthermore, in AMR applications, a grid is refined multiple times and inter-grid operations between the refinement levels are considered while repartitioning using Space Filling Curve (SFC) strategies like Hilbert ordering [110]. In our case the multiple high-resolution nests are formed at different locations in the simulation domain and there is

no communication between these nests. We focus on minimizing the data redistribution and maximizing the overlap between the old and new rectangular process grids for the same nest. Hence SFC based repartitioning is not applicable for our domain.

Sinha et al. [118] presented an adaptive system sensitive partitioning of AMR applications that tune the partitioning parameters to improve overall performance. In these efforts, the virtual 2D process topology is not considered to make redistribution decisions. However, in our work we need to allocate a sub-rectangle of the 2D process grid to each nest. This requires us to consider the virtual 2D process topology in our redistribution algorithm for the selection of sub-rectangles.

2.7 Topology-aware Task Mapping

Application scheduling is integral to performance of applications [11]. There exists lot of work on task scheduling [27, 106, 120, 133], some on communication-aware task scheduling [63, 88, 89, 119]. Past work [12–14, 19, 29, 39] shows various techniques to map parallel applications to communication networks. The techniques vary with the different network topologies. Specifically, mapping optimizations for Blue Gene torus networks [12, 19] take the application communication logs as an input and generate mapfiles for future application runs to optimize the hop-byte metric. Bhatele et al. [12] also show benefits in a single domain WRF run with their techniques. These techniques in literature are oblivious to the exact data flow in the application, though they can be tuned to map certain critical phases of the application. While they may be sufficient for single domain WRF simulations, multiple sibling domains running simultaneously present a harder problem. This is because we need to optimize the communications in the main domain as well as the multiple nested domains. In Chapter 3, we present mapping optimizations for WRF that selectively map each subdomain to a part of the torus, while also keeping the number of hops minimal for the subdomain and the parent domain processes. None of the existing work address this problem of topology-aware mapping for multiple dependant subtasks of an application.

2.8 Performance Analysis of WRF

Porter et al. [97] consider various aspects of WRF performance using three nested domain configuration. They present various compiler optimizations for PathScale and PGI compilers and mixed-mode executions for improving WRF performance. They report performance improvements of up to 10% for WRFv3.1.1 as a result of altering PGI compiler flags. They also present results to show that executing WRF on fewer cores per node in dm-mode reduces the time spent in MPI communication. Additionally, they report improvement in WRF performance upon changing the default x-y processor decomposition.

In [140], Wright et al. examine the scalability of WRF (version 2.1.2) on a single domain across different architectures using Integrated Performance Monitoring (IPM) tool [46] to analyze the performance. They show that for most of the architectures, WRF exhibits a sublinear speedup of both computation and communication times with increasing number of cores. Their results also show that MPI_Wait predominates the MPI communication times in WRF owing to the large number of point-to-point communications. Their experiments demonstrate the absence of significant contention for the on-node network resources between cores on the same node when fewer cores are used per node. They also report that the load imbalance in WRF increases with more number of cores and impacts the communication and computation time scalabilities.

Shainer et al. [114] analyze WRF performance on HPC clusters using a single domain WRF configuration. They show that the performance and scalability of WRF is better on Infiniband than gigabit Ethernet. Their results show that the productivity of WRF increases by executing multiple jobs per node. Using message size distribution profiling, they demonstrate the importance of high interconnect throughput to improve efficiency of WRF with increasing number of cores. From their experiments, they conclude that latency for 0–64B messages and throughput for 16K–64K messages are critical. They observe that MVAPICH and OpenMPI have higher throughput than HP-MPI for 16K–64K messages and hence the performance of WRF degrades when HP-MPI is used.

However, these message sizes are specific to their domain because the size of messages changes with the domain size.

In this thesis, we have worked on improving overall WRF performance for simulations with multiple sibling subdomains.

Chapter 3

Improving Throughput of Nested Simulations

High resolution nested simulations improve accuracy of the simulation output. However higher resolution and smaller time step increases the computation cost of nested simulations. The simulations are more expensive than the visualization, which can be performed fast on present day hardware accelerators. Hence, we focus on improving the throughput of simulations. In this chapter, we present a strategy to improve the performance of nested simulations by parallel execution of multiple nested domain simulations.

3.1 Introduction

Accurate and timely prediction of catastrophic events such as hurricanes, heat waves, and thunderstorms enables policy makers to take quick preventive actions. Such predictions require high-fidelity weather simulations and simultaneous online visualization to comprehend the simulation output on-the-fly. Multiple similar weather phenomena may occur simultaneously in different locations of the parent domain. Simulating and tracking the multiple regions of interest at fine resolutions is important in understanding the interplay between multiple weather phenomena and for comprehensive predictions. For example, Figure 3.1 illustrates the phenomena of two depressions occurring simultaneously in the Pacific Ocean. Here, it is necessary to track both depressions to forecast the possibility of a typhoon or heavy rainfall. In such scenarios, multiple simulations need to be spawned within the main parent simulation to track these phenomena. The

high resolution simulations are generally executed as subtasks within the coarser-level parent simulation.

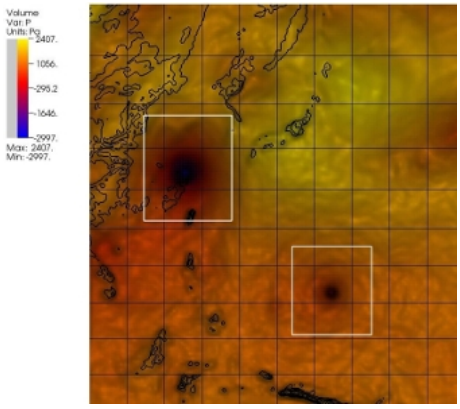


Figure 3.1: Visualization of multiple depressions in August 2010 on Pacific Ocean.

In weather simulations involving multiple regions of interest, the nested child simulations are solved r number of times for each parent integration step, where r is the ratio of the resolution of the parent simulation to the nested simulation. At the beginning of each nested simulation, data for each finer resolution smaller region is interpolated from the overlapping parent region. At the end of r integration steps, data from the finer region is communicated to the parent region. The nested simulations demand large amounts of computation due to their fine resolutions. Hence, optimizing the executions of nested simulations can lead to a significant overall performance gain. Additionally, the need for simultaneous visualization of the fine-grained weather predictions also entails high frequency output of weather forecast, which in turn results in huge I/O costs. Thus, reducing the I/O costs can also improve the overall performance.

3.1.1 Motivation

Existing weather applications employ a default strategy of executing the nested simulations corresponding to a single parent domain sequentially one after the other using all the available processors. However, these applications typically exhibit sub-linear scalability resulting in diminishing returns as the problem size becomes smaller relative to the number of available cores. For example, we observed that WRF is scalable up to large

number of cores [80] when executed without a subdomain, but exhibits poor scalability when executed with subdomains. Figure 3.2 shows the scalability of WRF on 1024 cores of IBM Blue Gene/L. The figure shows the execution times of WRF simulations with different configurations, each involving a parent domain of size 286×307 and a child domain. We experimented with various sizes of subdomains from 415×445 to 166×199 (nest sizes are shown in the figure). Note that the performance of WRF involving a subdomain saturates at about 512 processors. Hence in a WRF simulation with two subdomains executed on a total of 1024 cores, the performance of a subdomain executed on 512 cores will be about the same as when executed on all the 1024 cores. Thus, partitioning the 1024 cores proportionally among the subdomains for simultaneous execution will give better performance than serial execution on all the 1024 cores.

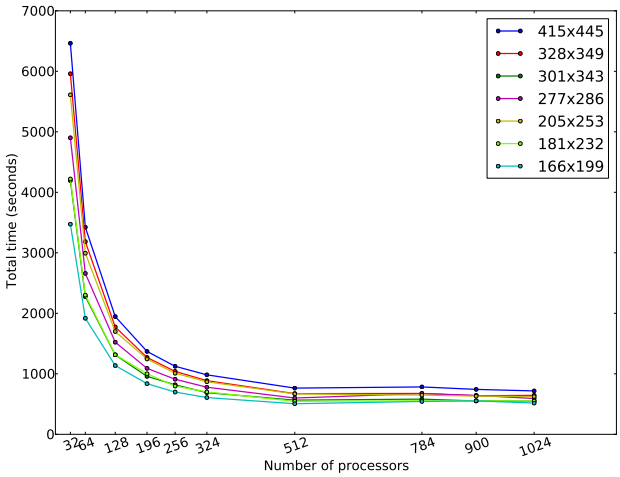


Figure 3.2: Execution times of nested weather simulations over a 10^7 sq. km. domain on upto 1024 Blue Gene/L cores.

3.1.2 Problem Statement

We focus on optimizing the parallel execution of high-resolution nested simulations to improve the overall performance of weather simulations pertaining to multiple regions of interest. The simultaneous execution of independent and non-homogeneous nested simulations, with different subdomain sizes, requires an efficient partitioning of the entire

processor space into multiple disjoint rectangular processor grids that can be assigned to the different nested simulations. This can minimize the parallel execution time if the number of processors are allocated in proportion to the work load associated with each nested simulation. Proportional processor allocation ensures that the time spent in the r integration steps of the different nested simulations is nearly equal, and the nested domains reach the synchronization step with the parent simulation together. There are also additional constraints on the aspect ratio of the sub-communicators that need to be handled in order to ensure that communication costs are minimized. We propose an efficient processor allocation strategy based on recursive bisection that takes into account the above requirements, and also uses estimates of relative execution times of the nests. The relative execution times of nests are estimated using a performance prediction model based on linear interpolation in a 2D domain from a small set of actual simulation times obtained from profiling runs. Our experiments show that our prediction model is highly accurate and exhibits less than 6% prediction error for most configurations.

We also propose topology-aware mapping of the nested subdomains on torus interconnects. Torus networks are widely prevalent in modern supercomputers, with 11 of the top 25 supercomputers in the November 2012 list based on torus network topology [128]. Topology-aware mapping has proved beneficial for applications on architectures with torus networks[12]. In this work, we consider architectures with 3D torus network topology viz. IBM's Blue Gene/L and Blue Gene/P. We have developed heuristics for mapping the 2D virtual process topology involved in the nested simulations to the 3D torus such that the neighbouring processes in the virtual topology are mapped onto neighbouring nodes of the torus. We propose mapping heuristics that minimize the communication for nested simulations and the parent simulation. Our approach for parallelization of multiple nested domain simulations also results in better scalability of the I/O operations.

3.1.3 Results and Contributions

Experiments on IBM Blue Gene systems show that the proposed performance modeling, partitioning and processor allocation strategies can improve simulation performance over

the default strategy of employing the maximum number of processors for all the nested simulations by up to 33% with topology-oblivious mapping and up to an additional 7% with topology-aware mapping. We also achieve up to 66% reduction in MPI_Wait times. Our approach for parallelization of multiple nested simulations also results in better I/O scalability. We explain the following primary contributions in this chapter.

1. A performance model for nested simulations based on linear interpolation that can predict execution times with less than 6% error.

2. Efficient method for processor allocation that result in 8% improvement over a naïve proportional allocation policy.

3. Topology-aware 2D to 3D mapping that result in 7% improvement over topology-oblivious mapping.

Section 3.2 presents our performance model, processor allocation, and mapping strategies. Section 3.6 briefly highlights the significance of our processor allocation and mapping strategies. Section 3.7 presents experimental results to illustrate the performance improvement achieved. We summarize in Section 3.8 and briefly highlight the generality of this work.

3.2 Parallel Execution of Subdomains

In simulations involving nested domains, the simulation of high-resolution nests are compute-intensive. Thus, the performance of these simulations improves with increasing number of cores. However, increasing the number of cores often leads to diminishing returns, especially when applications exhibit sub-linear scalability. The current strategy in many weather models including WRF is to execute these high-resolution nested simulations sequentially, utilizing all the cores to process one nest at a time. We show that performance of the overall simulation can be improved by subdividing the processor space into partitions for simultaneous executions of the nested simulations.

We concurrently execute the multiple nested simulations on disjoint subsets of processors. Estimates of the execution times of the nested simulations are required to decide the number of processors to be allocated for each nested simulation. We use linear interpolation for performance prediction as described in Section 3.3. The performance prediction of the simulation times is then used for partitioning the available processor space for the nested simulations as described in Section 3.4.

High-resolution nested simulations are spawned from the parent domain simulation. As mentioned above, the default WRF strategy is to spawn the nested simulations on the full set of available processors; these simulations use the MPI_COMM_WORLD communicator. In our approach, we create sub-communicators for each nested simulation. Since we use different sub-communicators for different nested simulations, it is beneficial to map the processes within a sub-communicator onto neighbouring nodes in the network topology. Furthermore, since the parent simulation uses the global communicator, a universal mapping scheme benefits both the parent and nested simulations. These topology-aware mapping heuristics are described in Section 3.5.

3.3 Performance Prediction

We propose a performance model that predicts relative execution times of the nested simulations with low prediction errors. A naïve approach is to assume that execution times are proportional to the number of points in the domain. However, our experiments indicate that a simple univariate linear model based on this feature results in more than 19% prediction errors. Our model uses piecewise linear interpolation based on the domain sizes. For a domain having width n_x and height n_y, we use the following two features of the domain for interpolation

1. Total number of points in the domain, given by $n_x \cdot n_y$.

2. Aspect ratio of the domain, given by n_x/n_y.

The naïve approach exhibits higher errors than our model because the total number of points do not capture the X-communication volume and Y-communication volume

separately. Hence the prediction for domain size of $n_{x1} \times n_{y1}$ would be same as the prediction for domain size of $n_{x2} \times n_{y2}$ where $n_{x1} \cdot n_{y1} = n_{x2} \cdot n_{y2}$. The aspect ratio together with the total number of points capture the X- and Y-dimensions and hence give better predictions.

We conducted experiments on a fixed number of processors for a small set (size = 13) of domains with different domain sizes and different aspect ratios. The actual execution times of these 13 domains are used to interpolate the execution times for other domains of varying sizes. Each domain can be represented as a point in the XY plane, where the X-coordinate denotes the aspect ratio and the Y-coordinate denotes the total number of points in the domain. The convex hull of these 13 points is triangulated using Delaunay triangulation [22]. Figure 3.3 shows a snapshot of the triangulation. The vertices of

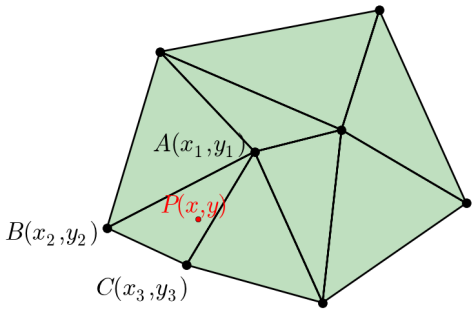


Figure 3.3: Delaunay Triangulation of points representing known execution times.

the triangles represent the known execution times. A domain D, represented as a point $P(x, y)$ inside the convex hull, will fall inside one of the triangles as marked in Figure 3.3. P lies inside $\triangle ABC$ whose vertices are A(x_1, y_1), B(x_2, y_2), and C(x_3, y_3). The barycentric coordinates [21] of P are obtained from A, B, and C using Equations (3.1), (3.2), and (3.3). The predicted execution time $\mathcal{T}_D$ of the nest domain D represented as P can be interpolated from the execution times of the domains represented by the vertices of

$\triangle ABC$ as shown in Equation (3.4).

$$\lambda_1 = \frac{(y_2-y_3)*(x-x_3)+(x_3-x_2)*(y-y_3)}{(y_2-y_3)*(x_1-x_3)+(x_3-x_2)*(y_1-y_3)} \tag{3.1}$$

$$\lambda_2 = \frac{(y_3-y_1)*(x-x_3)+(x_1-x_3)*(y-y_3)}{(y_2-y_3)*(x_1-x_3)+(x_3-x_2)*(y_1-y_3)} \tag{3.2}$$

$$\lambda_3 = 1 - \lambda_1 - \lambda_2 \tag{3.3}$$

$$\mathcal{T}_D = \lambda_1 * \mathcal{T}_{(x_1,y_1)} + \lambda_2 * \mathcal{T}_{(x_2,y_2)} + \lambda_3 * \mathcal{T}_{(x_3,y_3)} \tag{3.4}$$

The 13 domains required for interpolation will have to be carefully chosen for good predictions. To determine this set, we randomly generated a large number of points with domain size ranging from 94×124 to 415×445 and the aspect ratio ranging from 0.5 – 1.5. From this large set, we manually selected a subset of 13 points that nicely cover the rectangular region defined by the diagonal between (minimum domain sizes, minimum aspect ratio) and (maximum domain sizes, maximum aspect ratio). These points were selected in a way that the region formed by them could be triangulated well.

We note that it suffices to estimate the relative execution times for processor allocation to the nests. Hence, prediction on a particular processor size is sufficient to deduce the relative execution times. For larger domains, i.e. for the points lying outside the basis set of 13 points, we scale down to the region of coverage and then interpolate for those points. Though this does not accurately predict the execution times of the larger domains, this approach captures the relative execution times of those larger domains, and hence suffices as a first order estimate. Therefore, these 13 experiments suffice for predictions and it is not necessary to obtain the actual execution times for larger domain sizes. We have tested this approach on test domains with varying sizes and aspect ratios, and our predictions exhibit less than 6% error. The total number of points in these test domains lie in the range of 55,900 – 94,990, and the aspect ratio lie in the range of 0.5 – 1.5. We also tested by scaling up the number of points in each sibling, while retaining the aspect ratio. Section 3.7 shows the results for various nest domain sizes.

Our performance model based on Delaunay triangulation can be useful in modeling applications where the exact interplay of the parameters used for modeling are unknown.

In absence of such analytical model, linear interpolation gives a fairly accurate estimate as shown by our approach. This approach can also be applied for modeling more than two parameters.

3.4 Processor Allocation

We propose a processor allocation strategy in the context of multiple nested simulations that results in near-optimal performance. A simple processor allocation strategy is to equally subdivide the total number of processors among the nested simulations. However, this results in load imbalance because of the varying domain sizes of the nested simulations. We therefore use the relative execution times obtained from the performance prediction model to decide the number of processors to allocate for each nested simulation.

The parent simulation domain is solved on the full set of available processors. The processor space can be considered as a virtual processor grid of size $P_x \cdot P_y$. Consider the parent simulation domain of size $n_x \times n_y$. Initially, this domain is distributed over the processors by assigning rectangular regions of size $n_x/P_x \times n_y/P_y$ to each processor. The sibling domains are assigned processors as follows. The virtual processor grid is partitioned into multiple rectangular subgrids. The number of partitions is equal to the number of nested simulations. The area of a region allocated for a nested simulation is proportional to the predicted execution time of the nested simulation. This is illustrated in Figure 3.4 – it shows the subgrids of the processor space allocated to 4 nested simulations whose predicted execution times (as obtained from our performance prediction model) are in the ratio of $0.15 : 0.3 : 0.35 : 0.2$.

The subdivision of the 2D virtual process topology into k rectangular regions is a variant of the rectangular partitioning problem, which is known to be NP-hard [70]. We develop some heuristics for this problem. The pseudocode for this is shown in Algorithm 3.1. This algorithm divides the processor grid into regions that are as square-like as possible in order to minimize the difference in the communication volume of the X and Y dimensions.

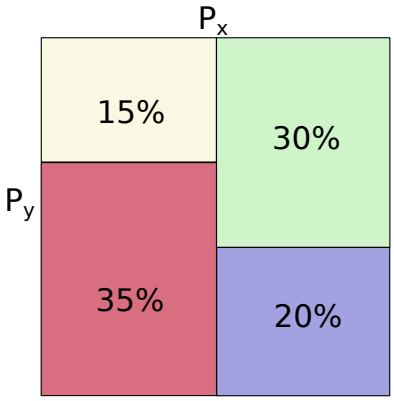


Figure 3.4: Partitions of processor space in the ratio of execution times of nested simulations.

The algorithm works as follows. We start by constructing a Huffman tree [20] using the execution time ratios as the weights, as shown in line 1. This gives us a binary tree such that at every internal node, the left and right children are fairly well-balanced in terms of the sum of the execution time ratios of the nested domains that belong to the two subtrees rooted at these two children. We then use this Huffman tree to construct a balanced split-tree over the virtual processor grid. This is done as follows. Note that all the nested domains lie at the leaves of the Huffman tree. We traverse the internal nodes of the Huffman tree in a breadth first (BFS) order, as shown in line 2. For every internal node, we first determine the longer of the two dimensions in lines 6–10. We then split the current grid along the longest dimension in the ratio of the total execution times of the nested domains in the left and right subtrees, as shown in lines 11–13; we then set the processor grid sizes for the two children (lines 14–18).

The partitioning is always done along the longer dimension to ensure that the rectangles are as square-like as possible. Figure 3.5 shows the difference when the first partitioning is along the longer dimension and when it is along the shorter dimension for $k = 3$. As can be seen, rectangle 3 is more square-like in Figure 3.5(a) than in Figure 3.5(b).

Input: Nested simulation domains $\{D_1, D_2, \cdots, D_k\}$, execution time ratios $R = \{R_1, R_2, \cdots, R_k\}$ of k nested simulations, total number of processors P, virtual processor grid $P_x \times P_y$

Output: Partition $P_x(i), P_y(i)$ for each nested simulation domain D_i for $1 \leq i \leq k$

Construct a Huffman tree, H, over the nested domains with execution time ratios as weights ;

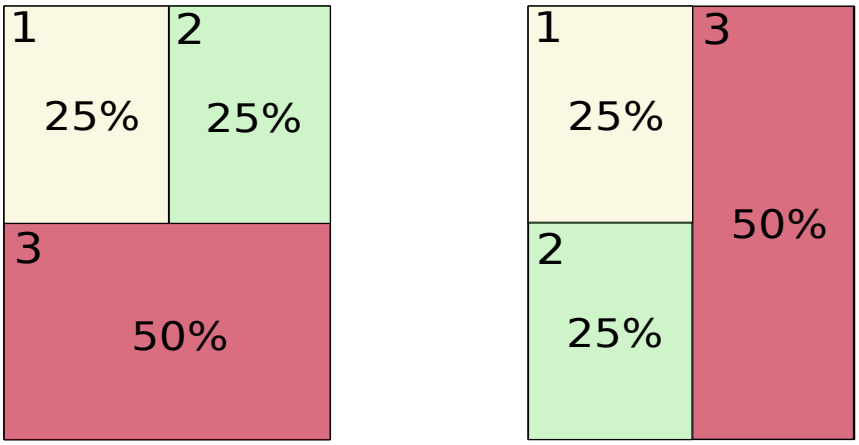
`/* Construct a balanced split-tree using the Huffman tree */`

for *every internal node u of H traversed in BFS order* **do**
    if *(u=root)* **then**
        $(P_x(u), P_y(u)) = (P_x, P_y)$
    if *($P_x(u) \leq P_y(u)$)* **then**
        $P_{ShortDim} = P_x(u)$, $P_{LongDim} = P_y(u)$;
    else
        $P_{ShortDim} = P_y(u)$, $P_{LongDim} = P_x(u)$;
    end
    Let l and r denote the left and right children of u respectively ;
    Let $Subtree_l$ and $Subtree_r$ denote the nested domains in the subtrees rooted at l and r respectively ;
    $W_l = \sum_{j \in Subtree_l} R_j$, $W_r = \sum_{j \in Subtree_r} R_j$;
    Divide $P_{LongDim}$ into P_l & P_r in the ratio of $W_l : W_r$;
    if *($P_x(u) \leq P_y(u)$)* **then**
        Set $(P_x(l), P_y(l)) = (P_{ShortDim}, P_l)$ & $(P_x(r), P_y(r)) = (P_{ShortDim}, P_r)$;
    else
        Set $(P_x(l), P_y(l)) = (P_l, P_{ShortDim})$ & $(P_x(r), P_y(r)) = (P_r, P_{ShortDim})$;
    end
end

Algorithm 3.1: Partitioning algorithm

3.5 Mapping

Weather simulations are communication intensive applications. For example, in WRF, each integration time-step involves 144 message exchanges with four neighbouring processes [80]. IBM's HPCT [18] profiling tools show that about 30–40% of the total execution time in WRF is spent in communication. Figure 3.6 shows the communication times



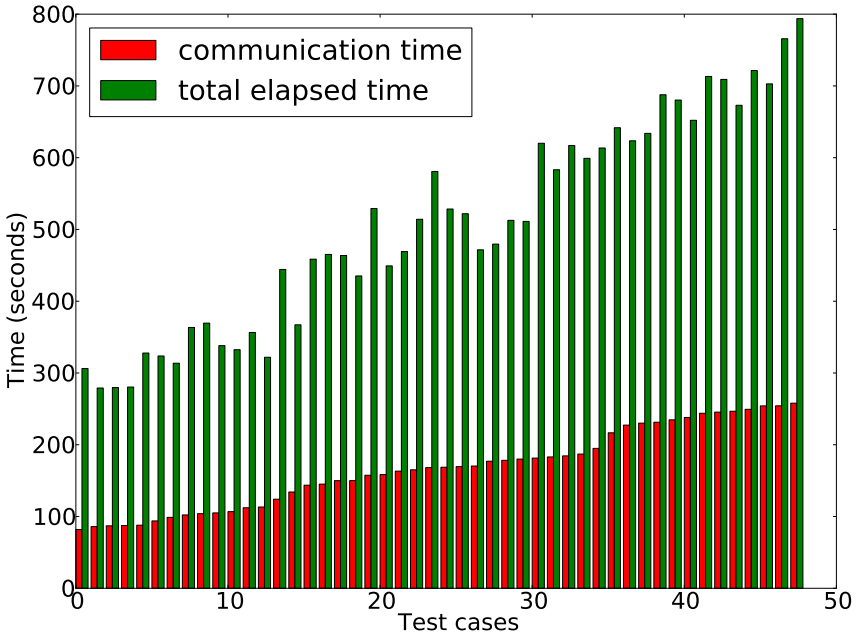
(a) Process grid 3 is as square-like as possible.

(b) Process grid 3 is a skewed rectangle.

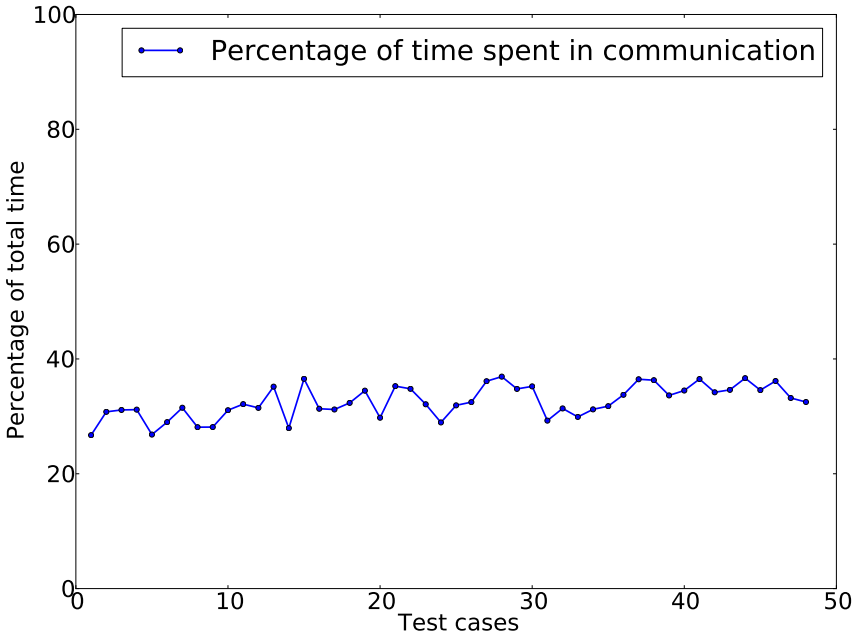
Figure 3.5: Partitions for $k = 3$ when the first partition is along the longer dimension (a) and when it is along the shorter dimension (b).

and the total elapsed times for 30 simulation configurations with 0–4 nest domains on 1024 Blue Gene/L cores. The red bars in Figure 3.6(a) show the communication times and the green bars show the total execution times for 2-hour simulation time. The y-axis shows the time in seconds and the x-axis shows 30 test cases with different number of nests and various nest sizes. The execution times and communication times are higher for simulation configurations with bigger and more number of nests. Figure 3.6(b) shows the percentage of time spent in communication for the test cases of Figure 3.6(a). It can be observed that 30–40% of execution time is spent in communications. Therefore reducing communication times can improve execution times.

Communication times are affected by process-to-processor mapping. Mapping is the placement of processes in the virtual topology onto the physical network topology. In our work we consider supercomputers with 3D torus interconnects and hence we address the problem of 2D to 3D mapping as shown in Figure 3.7. Figure 3.7(a) shows the 2D virtual process topology for a WRF configuration with 2 sibling domains (sibling 1 and sibling 2) of identical sizes. This virtual topology is used by the application for MPI communications. Figure 3.7(b) shows the 3D torus architecture of many modern-day



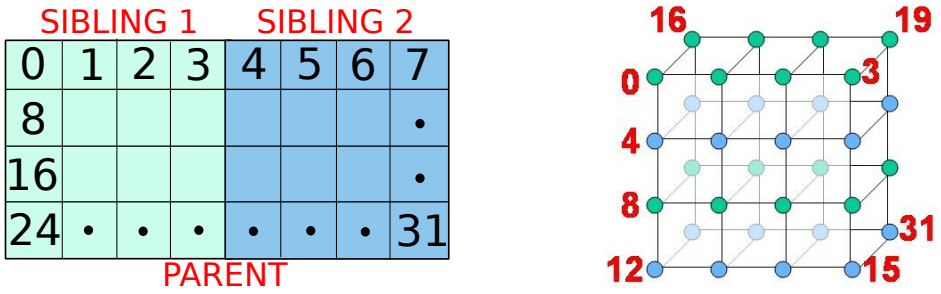
(a) Communication times and total elapsed times for 2-hour simulations.



(b) Percentage of time spent in communication.

Figure 3.6: Communication times in WRF simulations on 1024 Blue Gene/L cores.

supercomputers. Each process in the 2D grid is mapped to one of the nodes in the 3D network topology. This placement affects the number of hops in the network between neighbouring processes in the 2D topology because of the difference in dimensionality. The fewer the hops between the communicating processes in the torus, the lesser will be the time required for communication, thereby improving the overall application performance. We propose mapping heuristics in the context of nested simulations of multiple regions of interest. We describe topology-oblivious and topology-aware mapping heuristics in the following sections.



(a) Virtual process topology for 32 processes.

(b) Topology-oblivious mapping on $4 \times 4 \times 2$ torus.

Figure 3.7: 2D to 3D mapping.

3.5.1 Topology-oblivious mapping

The partitioning scheme subdivides the processor space into rectangular regions for simultaneous executions of the nested simulations. The case of 2 nested simulations is shown in Figure 3.7(a). The processes 0–3, 8–11, 16–19, 24–27 are allocated to one nested simulation and the rest of the processes are allocated to the other nested simulation. The simple mapping scheme is to sequentially map the processes in increasing order of process numbers to the torus nodes in increasing order of x, y and z coordinates. This is shown in Figure 3.7(b). In this example, the simple sequential mapping places processes 0–3 on the topmost row ($y = 0$) of the first plane ($z = 0$) of the torus, followed by processes 4–7 in the second row ($y = 1, z = 0$), 8–11 in the third row ($y = 2, z = 0$)

and so on.

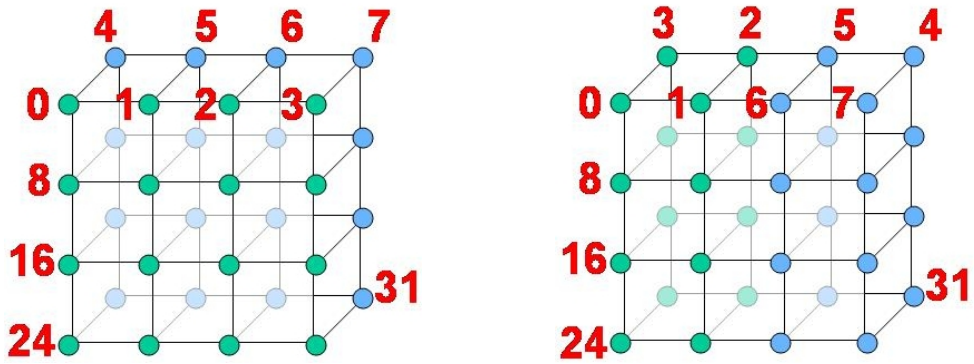
The topology-oblivious mapping in Figure 3.7(b) is sub-optimal for the communications within each of the nested simulations. This is because the neighbouring rows in the virtual topology are more than 2 hops apart in the torus, as shown with the help of green and blue nodes. For example, 0 and 8 are neighbours in the 2D topology whereas they are 2 hops apart in the torus. Similarly, process 8 is 3 hops away from process 16 in the torus. The topology-aware mapping heuristics discussed in the next section addresses this problem.

3.5.2 Topology-aware mapping

The general problem of mapping belongs to NP [12]. We describe below two heuristics for 2D to 3D mapping for multiple nested simulations. The first heuristic maps process partitions for each nest to contiguous nodes in the processor grid to reduce the communication times for halo exchanges in the nested simulations. The second heuristic also considers the neighbouring processes in the parent simulation while mapping processes to processors. The partitions are folded along one dimension so that communicating processes in parent simulation and nested simulations are neighbours.

Partition mapping - In this algorithm, we map each partition onto contiguous nodes of the torus. The partition mapping for the 2D process topology of Figure 3.7(a) is shown in Figure 3.8(a). The neighbouring processes in the virtual topologies of the partitions are neighbours in the torus. For example, processes 0 and 8 are neighbours in the virtual topology as well as in the torus. The first plane ($z = 0$) of the torus retains the 2D topology of the first partition, as shown by green rectangle in Figure 3.7(a) and green nodes in Figure 3.8(a). Similarly, the second plane ($z = 1$) of the torus retains the 2D topology of the second partition, as shown by blue rectangle in Figure 3.7(a) and blue nodes in Figure 3.8(a). This mapping also improves the parent domain communication performance because the neighbouring processes in the smaller rectangles are also neighbouring processes in the bigger rectangle. However, some of the processes in the parent domain are more than 1 hop away. For example, process 3 is 2 hops away from process

4 in Figure 3.8(a). To alleviate this problem, we propose another mapping heuristic in the next section.



(a) Partition mapping. (b) Multi-level mapping.

Figure 3.8: Topology-aware mappings.

Multi-level mapping - This is a modification of the partition mapping such that the neighbouring processes in the nested simulations as well as the neighbouring processes in the parent simulations are neighbours in the torus. In this mapping, we fold the rectangular partition in half and curl it across Z-axis. In the above example, the rectangles are curled across two XY-planes so that half the rectangle is in first plane ($z = 0$) and the other half is in the second plane ($z = 1$). This is illustrated in Figure 3.8(b).

Processes in the first rectangle of Figure 3.7(a) are folded anti-clockwise from the first plane ($z = 0$) to the second plane ($z = 1$). For example, process 0 is mapped to coordinate $(0, 0, 0)$ in the torus, process 1 is mapped to coordinate $(1, 0, 0)$, process 2 is mapped to $(1, 0, 1)$, process 3 is mapped to $(0, 0, 1)$ and so on. This ensures the processes in the first rectangle have 1-hop distant neighbours. Processes in the second rectangle of Figure 3.7(a) are folded anti-clockwise from the second plane ($z = 1$) to the first plane ($z = 0$). For example, process 4 is mapped to coordinate $(3, 0, 1)$ in the torus, process 5 is mapped to coordinate $(2, 0, 1)$, process 6 is mapped to $(2, 0, 0)$, process 7 is mapped to $(3, 0, 0)$ and so on. This ensures that the processes in the second rectangle have 1-hop distant neighbours. Thus, this improves performance of nested simulations. This mapping also ensures that the processes in the parent domain are 1 hop apart. For

example, processes 3 and 4 are 1 hop apart and so on[1]. Thus this universal mapping scheme benefits both the nested simulations and the parent simulation.

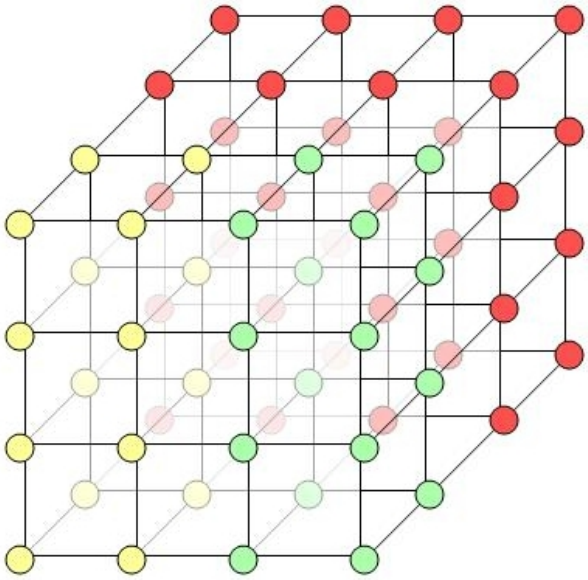


Figure 3.9: Multi-level mapping for the 3-sibling configuration in Figure 3.5(a).

Figure 3.9 shows multi-level mapping for the 3-sibling configuration that was shown in Figure 3.5(a). Here, the nests 1 (coloured yellow) and 2 (coloured green) are curled around the Z-axis and mapped from top to bottom, i.e. from row $x = 0$ to $x = 3$ as explained in the previous example. The third nest is mapped onto the planes $z = 2$ and $z = 3$, folded [12] from $z = 2$ to $z = 3$. The topmost row of the nest 3 and the bottom-most rows of nests 1 and 2 are neighbours in parent domain. Therefore in the plane $z = 2$, the processes are mapped from bottom ($x = 3$) to top ($x = 0$); followed by top ($x = 0$) to bottom ($x = 3$) mapping in $z = 3$ plane. This ensures the parent domain processes are neighbours on the torus.

We map nests to sub-rectangles where communication is always among near neighbours. This being an optimal mapping, the processes are placed nearby and so the network contention due to the halo exchanges reduces. Our mapping schemes can be applicable where there is an overlap between the processors allocated to different dependant

[1]The links between first and last nodes in a row/column of the torus have not been shown in the figures.

subtasks of an application.

3.6 Resource Allocation and Mapping

In our work, the finer grid is overlaid on the coarser grid and both coarser resolution parent simulation and finer resolution nest simulations are forwarded in time. The processors allocated for the parent and nest simulations may have partial or total overlap. This is significantly different from space-filling curves which are used for load-balancing adaptive mesh refinement (AMR) applications, where a major step is regridding from either coarser to finer resolution or vice versa [65, 138]. Space-filling curves like Hilbert or Morton ordering are used to generate the processor allocations which may result in non-rectangular partitions. Moreover, the resulting processor mapping from the curves will incur higher average hop-bytes. This is because even though space-filling curves move locally and are spatially clustered, yet it is possible that in many regions, the linear order fails to preserve the neighbourhood between communicating processes in stencil computations. Our processor allocation, reallocation and mapping strategies not only gives the required rectangular partitions but also provides communication-aware mapping for halo exchanges.

3.7 Experiments and results

3.7.1 Domain Configurations

We used WRF for all our experiments. Our WRF simulations involved up to a maximum of 4 sibling domains and resolution of up to 1.5 km. The minimum and maximum nest sizes used in the experiments were 178×202 and 925×820. For empirical evaluation, we chose the following two regions.

South East Asia – This covers South East Asian countries such as Malaysia, Singapore, Thailand, Cambodia, Vietnam, Brunei, and Philippines. The innermost nests were chosen such that the major business centers in this region are well represented. All these locations are affected by the meteorological features that are developed over

South China Sea. Thus, it is desirable to assess the meteorological impact on these key locations within the same modeling framework. Figure 3.10 shows a sample domain configuration that has the parent domain at 4.5 km resolution and the sibling domains at 1.5 km resolution. We experimented with eight different configurations at varying levels of nesting and different number of sibling domains. Three configurations had sibling domains at the second level whereas the remaining ones had siblings at the first level of nesting.

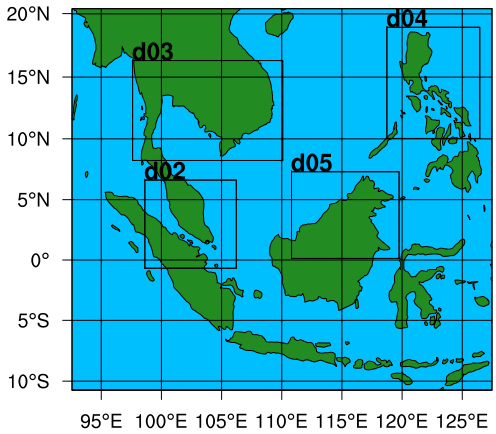


Figure 3.10: Sample domain in South East Asia with four sibling nests at 1.5 km resolution.

Pacific Ocean – The second region extends from 100°E - 180°E and 10°S - 50°N, covering the western Pacific Ocean region, where typhoons occur frequently. We experimented for the July 2010 typhoon season with 85 different configurations of the nest domains. These configurations were randomly generated with domain size ranging from 94×124 to 415×445 and the aspect ratio ranging from $0.5 - 1.5$. We form multiple nests to track multiple depressions over the Pacific region. There can be several depressions forming over the region, which trigger high-resolution nest formation. The parent domain size is 286×307 at 24 km resolution and the nests have 8 km resolution, with up to 4 siblings at the first level of nesting.

3.7.2 Experimental Setup

IBM Blue Gene/L: Blue Gene/L (BG/L) [126] is the first generation of IBM's Blue Gene supercomputers. Each node consists of two 700 MHz PPC 440 processor cores with 1 GB of physical memory. The system supports two execution modes for the applications – coprocessor (CO) mode and virtual node (VN) mode. In the CO mode, one core is dedicated to communication and other for computation. In the VN mode, both cores are used for computation. BG/L nodes are connected with three communication networks: a 3D torus network providing point-to-point communication between compute nodes, a collective network for global broadcast and integer reduction operations and a global interrupt network for fast barrier synchronizations. 3D torus network is the primary communication network in BG/L and handles bulk of all the communication. We have experimented on maximum of 1024 cores on BG/L in VN mode.

IBM Blue Gene/P: Blue Gene/P (BG/P) [127] is the second generation of Blue Gene supercomputers. Each node contains four 850 MHz PPC 450 processor cores with 4 GB of physical memory. BG/P supports three different application execution modes – Symmetric Multi Processing (SMP) mode, Dual mode and the VN mode. SMP mode supports one process per node with up to four threads per process; Dual mode supports two processes per node with up to two threads per process and VN mode supports four single-threaded processes per node. The communication network in BG/P is similar to BG/L. We experimented on up to 8192 cores on BG/P in VN mode.

WRF Runtime Setup: Parallel netCDF (PnetCDF) [69] was used for performing I/O on BG/P. The split I/O option of WRF was used on BG/L, where every process writes its own data onto the disk. WRF was run in the VN mode on BG/P in order to study the scalability issues while using higher number of MPI ranks. In all the simulations, Kain-Fritsch convection parameterization, Thompson microphysics scheme, RRTM long wave radiation, Yonsei University boundary layer scheme, and Noah land surface model were used. We experimented with both low and high output frequencies for parallel I/O on BG/P. The output frequency for BG/L simulations was 1 hour.

3.7.3 Improvement in execution time

In this section, we present the results on the performance improvement on BG/L and BG/P using WRF domains with varying nest sizes and varying number of siblings.

Improvement in per-iteration time

There was an average of 21.14% and maximum of 33.04% percentage reduction in execution times. This is the overall improvement in the simulation performance on 1024 cores (512 nodes in VN mode) on BG/L from 85 configurations with nest sizes varying from 178×202 to 394×418 and number of siblings varying from 2–4. This improvement is due to the parallel execution of sibling domains on different subsets of processors. However, it is important to note that the default strategy of using all the processors for solving a nest, can be beneficial if the application exhibits linear or superlinear speedup.

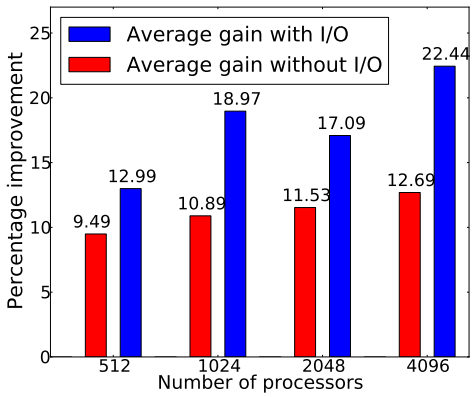


Figure 3.11: Performance improvement of execution time on up to 4096 BG/P cores including and excluding I/O times.

Figure 3.11 shows the percentage improvement in execution time, averaged over 30 different domain configurations. The figure shows that the performance improvement is higher when the I/O times are also considered. This is because parallel NetCDF does not scale well with increasing number of processors. In our approach, fewer number of processors output data for the siblings, thereby the time to output data is lesser than the default approach. It should be noted that for any practical application, generation of

output data is important for visualization and perceiving the simulation output. Hence our approach proves beneficial for practical scenarios.

Improvement in communication time

The average and maximum percentage improvement in MPI_Wait time is shown in Table 3.1. In WRF, the simulations perform halo exchanges with 4 neighbouring processes. One of the reasons for the high wait times observed in the default execution is due to the high average number of hops between neighbouring processes. However, in our case, since the siblings are solved on smaller subset of processors, the average number of hops decreases resulting in lesser load on the network. It also leads to lesser congestion and smaller delay for point-to-point message transfer between neighbouring processes.

Table 3.1: Average and maximum improvement in MPI_Wait times on BG/L and BG/P

Number of cores	Average (%)	Maximum (%)
1024 on BG/L	38.42	66.30
512 on BG/P	30.70	60.92
1024 on BG/P	36.01	60.11
2048 on BG/P	27.02	55.54
4096 on BG/P	28.68	43.86

Improvement in sibling simulation time

WRF solves one parent time step followed by solving r nested time steps. Therefore, improving the performance of nest solve time steps improves the overall performance of the application. In our approach we simultaneously execute all the siblings as compared to sequentially executing them one after the other. We illustrate the benefit of this approach on the sibling integration times with the help of a domain configuration which has 4 siblings at the first level. The sibling configurations and the number of processors allocated to these siblings according to our performance model and partitioning strategy

are shown in Table 3.2. Figure 3.12 compares the per-iteration nest execution times for

Table 3.2: Sibling configurations for four siblings on BG/L

	Sibling 1	Sibling 2	Sibling 3	Sibling 4
Nest size	394 × 418	232 × 202	232 × 256	313 × 337
Number of cores	18 × 24	18 × 8	14 × 12	14 × 20

this configuration. The first stacked bar shows the sibling times for the default serial execution. In this case, the siblings take 0.4, 0.2, 0.2 and 0.3 seconds when executed sequentially on 1024 cores on BG/L. Since the siblings are solved sequentially, their execution times add up resulting in 1.1 seconds. In our parallel strategy, when the siblings are solved on subset of processors, the solve times for the four siblings are 0.7, 0.6, 0.6 and 0.7 seconds. The individual sibling solve times have increased due to using fewer than 1024 processors. However, since these are solved concurrently, the overall time for per iteration of nest solve step for the four siblings is 0.7 seconds in our approach. This results in 36% performance gain in execution times for the sibling domains. Our

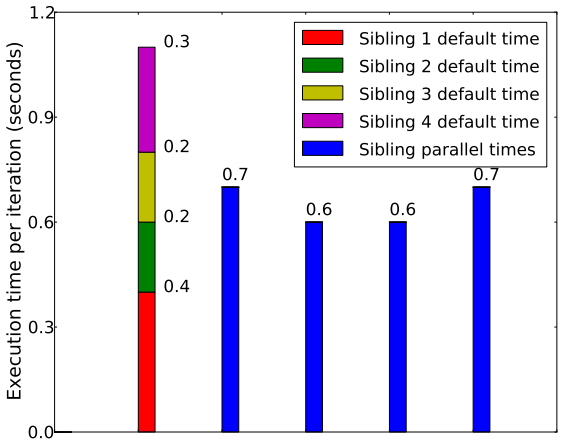


Figure 3.12: Sibling execution times on 1024 processors on BG/L for four siblings.

processor allocation strategy reduces the number of processors per sibling as compared to the default strategy. Hence if the nest sizes are large, the performance improvement by

executing the simulation on fewer number of processors will be low. This is because the scalability of the larger domains reach saturation at higher number of processors. Hence as we increase the number of processors for the simulation, the performance improvement will increase. We illustrate this with the help of a simulation configuration with three large siblings of sizes 586×643, 856×919 and 925×850. The performance improvement for different number of processors and the nest execution times for default sequential strategy and our approach are shown in Figure 3.13. The performance improvement on 1024 processors is only 1.33% because of higher saturation limit for larger nests. As the number of processors is increased for the full simulation, the number of processors allocated to the nests also increase. Moreover, the larger sibling domains reach saturation limit much before 8192 processors. Therefore we observe that performance improvement increases from 4.39% to 15.9% on doubling 2048 cores whereas there is only additional 4.74% improvement on doubling 4096 cores.

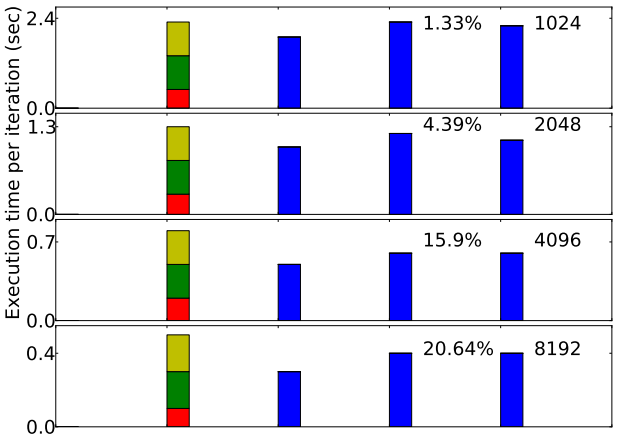


Figure 3.13: Sibling execution times on up to 8192 BG/P cores. The legends on the rightmost side show the number of cores.

Effect on varying sibling configurations

In this section we present results for varying number of sibling domains and varying sizes of sibling domains.

Varying number of siblings – The more the number of siblings, the longer will be the time taken per iteration by the default approach because of sequential execution of the nests. In our approach since we concurrently execute all the siblings, the number of siblings do not affect the time if the maximum number of processors is sufficiently high for the nest sizes. Hence we observe that the average performance improvement for experiments involving two siblings is 19.43% whereas the average improvement in execution time for experiments involving four siblings is 24.22%.

Varying sibling sizes – The larger the nest sizes, the higher will be the number of processors required to improve performance. Hence we observe that with larger nest sizes the performance improvement decreases as shown in Table 3.3.

Table 3.3: Sibling configurations and performance improvement for varying nest sizes on up to 8192 BG/P cores.

Maximum nest size	205×223	394×418	925×820
Percentage improvement	25.62	21.87	10.11

3.7.4 Improvement with topology-aware mapping

Table 3.4: Execution times (sec) for default, topology-oblivious and topology-aware mappings for various sibling configurations on BG/L.

Default	Topology-oblivious	Partition mapping	Multi-level mapping	TXYZ mapping
2.77	2.25	2.10	2.07	2.12
3.69	3.08	2.95	2.92	2.95
3.43	2.89	2.72	2.72	2.83
4.98	3.92	3.72	3.72	3.99
4.75	3.53	3.39	3.33	3.44

In this section, we present the performance improvement achieved by the topology-aware mappings discussed in Section 3.5.2. Table 3.4 shows the execution times per iteration for the default strategy, the topology-oblivious and topology-aware mappings on 1024 BG/L cores. The first three rows correspond to 2-sibling domain configuration, and the fourth and fifth row correspond to 3-sibling and 4-sibling configurations. We observe additional improvement of up to 7% over the topology-oblivious mapping. It can be seen that our mappings outperform the existing TXYZ mapping in Blue Gene.

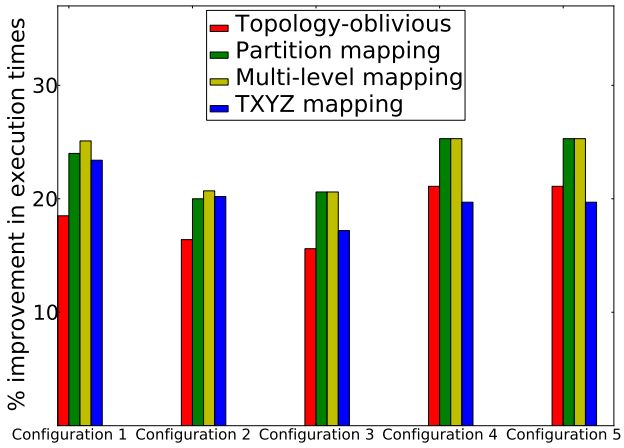


Figure 3.14: Percentage improvement in execution times with and without topology-aware mapping on 1024 BG/L cores.

The percentage improvement in execution times and MPI_Wait times over the default strategy is illustrated in Figures 3.14 and 3.15 respectively. It can be noted that the multi-level mapping is slightly better or almost equal in performance as compared to the partition mapping. This is because even though partition mapping does not optimize the parent simulation, as explained in Section 3.5.2, the overall simulation is not adversely affected because the nested simulations are executed r times more than the parent simulations.

Table 3.5 shows the execution times per iteration for the default strategy, the topology-oblivious and topology-aware mappings for various sibling configurations on BG/P. The

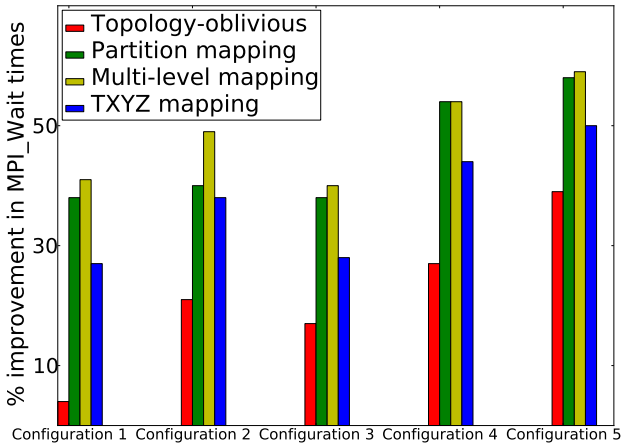


Figure 3.15: Percentage improvement in MPI_Wait times with and without topology-aware mapping on 1024 BG/L cores.

first two rows correspond to 4-sibling domain configuration and the third row corresponds to 3-sibling configuration. The multi-level mapping performs almost similar to the partition mapping. This may be due to load imbalance in WRF.

Table 3.5: Execution times (sec) for default, topology-oblivious and topology-aware mappings on 4096 BG/P cores

Default	Topology-oblivious	Partition mapping	Multi-level mapping
5.43	3.94	3.92	3.93
5.65	4.20	4.1	4.1
5.61	4.39	4.28	4.39

Figure 3.16 illustrates the percentage improvement in the MPI_Wait times for 3 configurations. The higher MPI_Wait times for the default approach is due to increased network congestion caused by more halo communications across multiple hops. The MPI_Wait times decrease by more than 50% on average for the topology-oblivious and topology-aware mappings. The topology-aware mappings further decrease the wait times due to reduction in the average number of hops. This is due to our efficient mapping

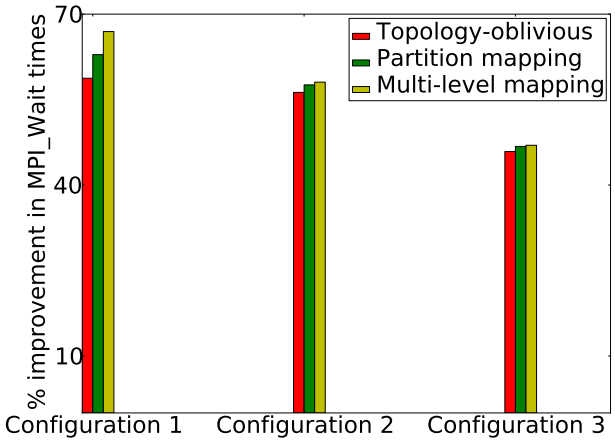


Figure 3.16: Percentage reduction in MPI_Wait times with and without topology-aware mapping on 4096 BG/P cores.

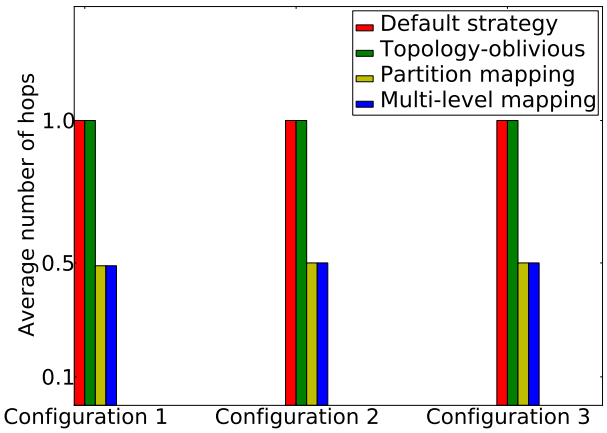


Figure 3.17: Reduction in average number of hops with and without topology-aware mapping on 4096 BG/P cores.

heuristics that map the neighbouring processes in the virtual topology to the neighbouring torus nodes. The maximum reduction in MPI_Wait times is observed for the multi-level topology-aware mapping because this mapping strategy maps communicating processes of both parent simulation and nested simulations onto neighbouring nodes of the torus. Therefore it results in lower MPI_Wait times during the halo exchanges.

Figure 3.17 illustrates the reduction in the average number of hops over the default

strategy. The average number of hops for the topology-oblivious mapping is the same as the default execution because in both the cases, the default mapping in BG/P is used. A 50% reduction in the average number of hops is observed for the topology-aware mappings, which leads to decrease in the wait times. The reduction in the number of hops is because of mapping communicating processes to neighbouring nodes in the processor grid. The neighbouring processes in the virtual topology are a single hop away in the physical processor grid due to the efficient topology-aware mappings.

3.7.5 Effect on high-frequency output simulations

High resolution operational forecasts typically require forecast output very frequently. In order to simulate this scenario, we performed experiments with output generated every 10 minutes of a simulation for all the various regions of interest at the innermost level. We present the results for high-frequency output simulations. Figures 3.18(a-c) show the variation of per time-step times for integration, I/O operations, and the total time. The I/O time consists of time for writing output files and processing the boundary conditions. The per iteration integration time in Figure 3.18(a) shows a steady decreasing trend for both the default sequential and the parallel versions until 4096 processors. The parallel sibling version shows slightly better scaling behavior in the range 4096–8192. However, in the case of I/O performance, the parallel sibling case provides significant reduction in I/O time. For the sequential version, the per iteration I/O time steadily increases with increasing number of processors. The effect of I/O performance on the total times is clearly seen in the relative ratios of integration and I/O times in Figure 3.19. This observation suggests that PnetCDF has scalability issues as the number of MPI ranks increases and could be a real bottleneck in scaling high resolution weather forecasting simulations. In the parallel execution case, only a subset of the MPI ranks take part in writing out a particular output file and thus, this results in better I/O performance. Since the I/O times remain a relatively low fraction of the total time, the parallel execution of sibling nests shows better scalability for the total per iteration time as shown in Figure 3.18(c).

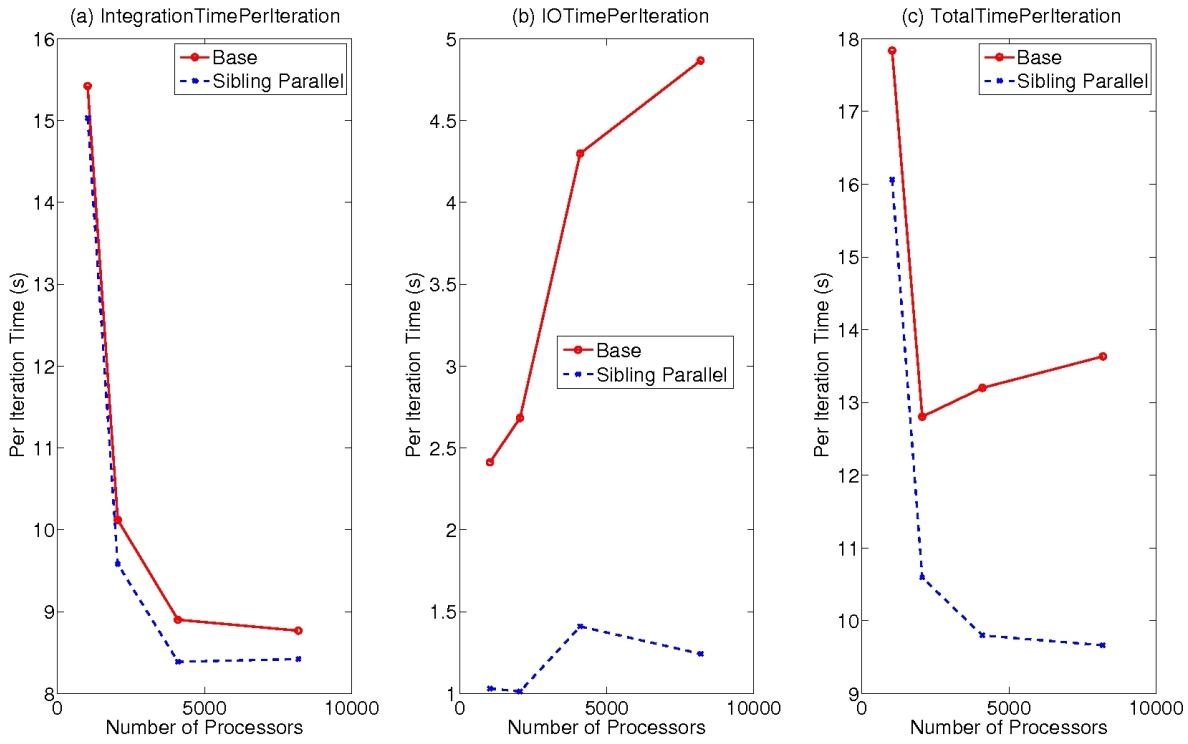


Figure 3.18: Variation of integration, I/O, and total per iteration times with number of processors on BG/P.

3.7.6 Efficiency of our processor allocation and partitioning strategy

Our performance prediction model coupled with the partitioning algorithm improves the performance by 8% as compared to a naïve strategy of subdividing the processor space into consecutive rectangular chunks based on the total number of points in the sibling. We experimented with a 4-sibling domain configuration, whose default execution time is 4.49 seconds per iteration. The naïve strategy decreases the execution time to 4.08 seconds, achieving 9% improvement whereas our algorithm decreases the execution time to 3.72 seconds, thereby obtaining 17% improvement over the default strategy.

3.7.7 Scalability and speedup

We executed a simulation with two sibling nests of 259×229 size for the default sequential approach and our simultaneous execution approach, varying the number of processors

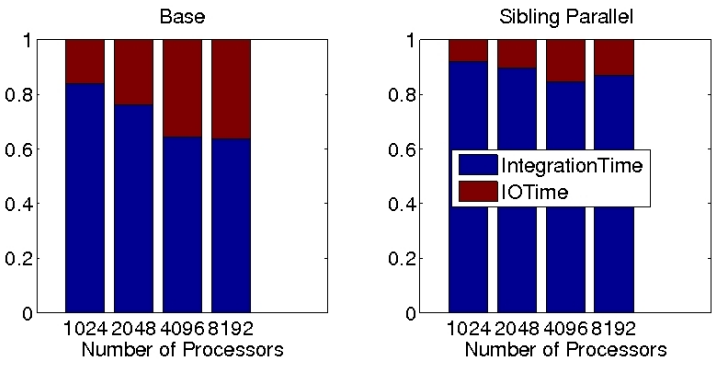


Figure 3.19: Variation of fraction of integration and I/O times averaged over all the different configurations vs. number of processors on BG/P.

from 32 to 1024. Figure 3.20 shows the scalability and speedup curves. Both approaches

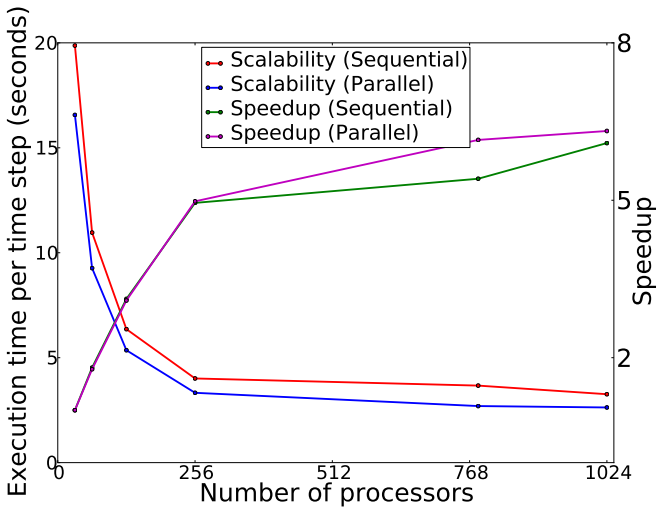


Figure 3.20: Scalability and speedup of default sequential strategy and our concurrent execution approach.

have similar scalability saturation limits. However, our approach exhibits lower execution times for all processor sizes. It can be observed that our strategy of simultaneous executions of siblings shows better speedup than the default sequential strategy at a higher number of processors. This is because the simulation stops scaling beyond 700 processors. Hence, increasing the number of processors for the siblings proves less useful than solving the siblings simultaneously on smaller subset of processors. For lower

number of processors, the speedup for both the approaches is almost the same. This is because the simulation reaches saturation limit at higher number of processors. Hence solving them sequentially on the full set of processors gives equal performance as solving them concurrently on subsets of processors.

3.8 Summary

In this chapter, we presented a comprehensive scheme to optimize weather simulations involving multiple nested regions of interest. We showed that the performance of such weather simulations can be improved by allocating subsets of processors to each region of interest instead of the entire processor space. Though we focussed on weather applications, the algorithms developed in this work can improve the throughput of applications with multiple simultaneous simulations within a main simulation, for example crack propagation in a solid using LAMMPS[64]. Multiple cracks can be simultaneously atomistically simulated within a continuum simulation domain. This methodology can also be applied to nested high-resolution coastal circulation modeling using ROMS[107].

We proposed a linear interpolation based performance prediction model which predicts the execution times with low error. Our processor allocation scheme based on Huffman tree construction and recursive bisection outperforms a naïve proportional allocation by 8% with respect to the total execution time. We developed 2D to 3D mapping heuristics that take into consideration communications in the nested simulations as well as the parent simulation. We achieved up to 33% improvement in performance with up to an additional 7% improvement with our topology-aware mapping heuristics. Our topology-oblivious and topology-aware mappings reduce the communication times by a maximum of 66%. To the best of our knowledge, this is the first work that optimizes parallel execution of weather simulations involving multiple nested regions of interest.

The multiple regions of interest may not be static. For example, the depressions shown in Figure 3.1 may weaken over time, thereby triggering deletion of the corresponding nest. Hence we need to tackle the temporal variation in the regions of interest. We describe such an extension to dynamic nests in the next chapter.

Chapter 4

Diffusion-based Repartitioning Strategies

The regions of interest within a simulation may appear and disappear with time. Therefore the nested simulations spawned over the regions of interest may be also dynamic. In the previous chapter, we have showed how to efficiently partition the processor space for a given static nest configuration. In this chapter, we will describe processor reallocation strategies with minimum data redistribution costs for dynamic nests.

4.1 Introduction

Multiple similar meteorological phenomena may occur at the same time in different regions of a geographical domain. For example, Figure 4.1 illustrates the phenomena of tall clouds occurring at multiple regions simultaneously in the Indian region. Weather simulations need to track these clouds at higher resolutions. Simulating and tracking these multiple regions of interest at high resolutions is important in understanding the weather phenomena and for accurate weather predictions. These phenomena may vary temporally as well as spatially. Some of the regions of interest may disappear in subsequent time steps while new regions of interest may form in the domain. For example, some of the cloud systems[1] shown in Figure 4.1 may produce severe weather conditions such as high winds, intense localized rainfall, floods and storm surges over coastal regions, some clouds may move to different regions and cluster with other clouds, and some

[1]Cloud systems are a hierarchical organization of clouds.

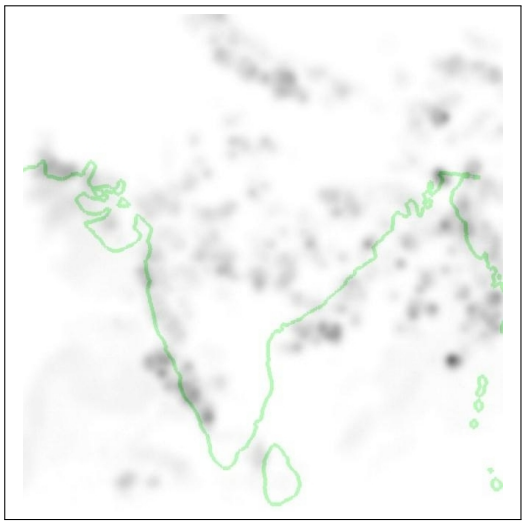


Figure 4.1: Tall clouds over the Indian region during the 2005 monsoon season. Image generated from WRF simulation. Darker regions correspond to regions with higher cloud water mixing ratios.

may disappear with time.

4.1.1 Challenges

Simulation methods have to efficiently handle simultaneous occurrence of meteorological phenomena at different locations. Tracking these multiple phenomena incur new challenges which are more difficult to tackle than the simulation of a single phenomenon. We list some of the challenges below.

- Simultaneously tracking the appearance, disappearance and merging of the phenomena at various locations requires dynamic representations and efficient data analysis.
- Simulating multiple events at high resolutions needs efficient processor allocation schemes for these multiple events.
- The dynamic nature of these events require fast data redistribution strategies.

Nests are spawned within the parent simulation to simultaneously track these multiple regions of interest. Each nested simulation is executed on disjoint subsets of the total

number of processors for high performance, as explained in Chapter 3. Therefore, with dynamic appearance and disappearance of regions of interest, we need to modify the processor allocation for the nests. The processors allocated to the nests in the previous time step of simulation will have to be freed if the nests do not exist in the current time step. Similarly, a subset of processors needs to be allocated to the new nests formed in the current time step. The removal of old nests and addition of new nests may lead to new processor allocation for the old nests which continue to exist from the previous time step.

4.1.2 Problem Statement

Reconfiguration of processor allocation implies data redistribution for the old nests. In this work, we have developed a tree-based hierarchical diffusion algorithm for processor allocation that reduces data movement by considering the old processor allocation. This algorithm results in lower redistribution time as compared to a strategy that does not consider the existing processor allocation.

We have implemented the redistribution algorithm to support resource reconfigurations in an application that detects and tracks organized tropical convective cloud systems which have widespread occurrence of tall *cumulonimbus* clouds as a distinct signature. These clouds are associated with thunderstorms and atmospheric instability, forming from water vapour carried by powerful upward air currents. They can produce heavy rain and flash flooding. Hence, it is important to track these clouds. These clouds may form and disappear with time. We have developed a parallel data analysis algorithm that detects these clouds from simulation output. We spawn nests over these regions of interest within the running simulation, and also dynamically remove nests when the old regions of interest no longer exist.

We performed experiments on Blue Gene/L and Intel Xeon-based clusters using both real data corresponding to Mumbai rainfall of 2005, and synthetic nest formations and deletions for the same period. Our results showed that we were able to reduce the redistribution time by 25% over the scratch method and resulted in 53% lesser hop-bytes

on Blue Gene/L.

While we have used the tracking of tall clouds as a case study, our algorithms for data analysis and processor allocation are generic and applicable to other scenarios that involve multiple dynamically varying nested simulations.

4.1.3 Chapter Outline

Section 4.2 presents our parallel data analysis algorithm. Section 4.3 presents our data redistribution strategies. Experimental results are presented in Section 4.4 to demonstrate the performance improvement achieved. Section 4.5 remarks about the applicability of our work to other scientific applications and on other platforms. Section 4.6 presents some concluding remarks.

4.2 Tracking Cloud Systems via Parallel Data Analysis

In this section, we describe an algorithm for parallel data analysis of simulation output. The algorithm analyzes the cloud water mixing ratio (QCLOUD) and outgoing long wave radiation (OLR) in WRF simulation output to detect tall clouds in tropical weather systems. These clouds are referred to as *cumulonimbus* clouds. These clouds extend vertically from 1 km above the surface to more than 10 km. QCLOUD is the amount of liquid water contained in a cloud. Generally, high values of QCLOUD correspond to tall clouds. OLR is the infrared radiation at the top of the atmosphere. Coherent patterns of low OLR indicate occurrence of organized cloud systems (such as tropical depressions and cyclones) and would contain tall cumulonimbus clouds. A combination of OLR and QCLOUD better identifies such systems and precludes identification of isolated cumulonimbus (as QCLOUD alone would do) [108]. We use 200 as the upper threshold for OLR [35].

Each process running WRF generates output for its subdomain and writes into a split file. These split files are analyzed in parallel as shown in Algorithm 4.1. This algorithm forms contiguous, non-overlapping, and small clusters whose sizes do not grow

Input: Per-process simulation output of one time step from P processes $\{F_1, F_2, \cdots, F_P\}$,
Number of processes for parallel data analysis N

Output: *Rectangles*: Rectangular regions with high cloud water mixing ratio

`/* Divide` P `files among` N `processes */`

1 $k = P/N$;

2 Let S be the set of k files assigned to each of the N processes;

`/* Begin analysis of` QCLOUD `values in the files in` S `by each of the` N `processes */`

3 $count = 0$;

4 **foreach** $file \in S$ **do**

5 Read QCLOUD and OLR from $file$ for each grid point;

6 Aggregate *qcloud* and increment *count* where $OLR[gridpoint] \leq 200 \; \forall \; gridpoint \in file$;

7 Let *area* be the total number of grid points in the file;

8 $olrfraction = count/area$;

9 **end**

`/* End analysis */`

10 $root = 0$; `/* Assume rank 0 is the root rank */`

11 Root collects the *qcloud* and $olrfraction$ information from every process in *qcloudinfo*;

`/* Form rectangular regions in` *root* `process */`

12 **if** *(my rank == root)* **then**

13 Sort *qcloudinfo* in decreasing order of *qcloudinfo.qcloud*;

14 $Clusters$ = NNC(*qcloudinfo*);

15 $Rectangles = \emptyset$;

16 **foreach** $(list \in Clusters)$ **do**

17 Let $item = (minX, maxX, minY, maxY)$ be set of the minimum and maximum of x and y coordinates of elements of *list*;

18 Add *item* to *Rectangles*;

19 **end**

20 **end**

Algorithm 4.1: Parallel Data Analysis (PDA) algorithm

uncontrollably. It is simple and fast and hence suitable for online analysis. Let P be the number of processes running WRF and N be the number of processes which analyze the QCLOUD values in the split files. The algorithm takes as input the split files $\{F_1, F_2, \cdots, F_P\}$. These split files are distributed to the N processes. Each of the N processes analyze k files (lines 1–2). The subset S of files, where $|S| = k$, is chosen as a rectangular subset of (Px, Py), where $Px \cdot Py = P$ is the rectangular process decomposition in WRF. Thus P is divided into N rectangular subsets.

The value of QCLOUD at each grid point in each split file is aggregated if the outgoing long wave radiation OLR $\leq$ 200 (lines 4–9). The fraction of the grid points which satisfy the above criteria, $olrfraction$, is calculated (lines 7–8). The aggregated QCLOUD values, one value per file, are then sent by all the N processes to a root process, rank 0 in our case. Each process will atmost send k values. Note that some of the split files may not have regions with OLR $\leq$ 200, in which case the process owning these split files will send fewer than k values. The root process gathers the aggregated QCLOUD values and the $olrfraction$ values (line 11).

The rest of the algorithm is executed only on the root process. Firstly the aggregated QCLOUD values obtained from the split files are sorted in non-increasing order (line 13). A contiguous region with high cloud cover can span multiple split files processed by multiple processes. To obtain a contiguous region, we perform a variant of nearest neighbour clustering (NNC) (line 14). NNC outputs a set of clusters with each cluster containing a contiguous region of high cloud cover. A rectangle is formed around each cluster (lines 16–19) and these rectangles constitute nests for fine-resolution simulations in WRF.

Nearest Neighbour Clustering: The pseudo code for the NNC algorithm is shown in Algorithm 4.2. It takes as input the sorted list of QCLOUD values, *qcloudinfo*. Each element in *qcloudinfo* is a tuple of aggregated QCLOUD values for a split file and the corresponding fraction of the split file which has OLR $\leq$ 200. The QCLOUD value of each element in the list represent the cloud cover for a subdomain. The spatial location, i.e. the latitude and longitude extents of a subdomain is used in this algorithm to determine

proximity between two subdomains.

The algorithm iterates over each element in the input array *qcloudinfo* (lines 2–20). Line 3 checks whether the aggregate QCLOUD value and the fraction of the subdomain that has OLR $\leq$ 200 are greater than a threshold, which is 0.005 in our case. This avoids analyzing smaller cloud-covered regions with a very low QCLOUD value. Clusters are formed based on proximity of the elements (lines 4–18). Each cluster represents a contiguous region of strong cloud cover. An element is added to a cluster if it is either 1-hop or 2-hop away from an existing cluster. Initially, the list of clusters is empty. First, we check if the current element is at 1-hop distance from any element in an existing cluster (lines 6–9). If this does not hold true, then we check if the element is 2 hops away from any element in an existing cluster (lines 10–13).

In lines 6 and 10, the DISTANCE function is invoked to calculate the proximity. If it returns true, the element is added to *list*. If *element* is within *hop* distance from *member*, then it is added to the cluster *list* iff it does not deviate the mean of the QCLOUD values by more than a threshold (30% in our case) (lines 23–29). This ensures that a cluster of contiguous cloud region has low standard deviation and also helps in controlling the size of an existing cluster.

If *element* is not within 2 hops from any element in any of the existing clusters, then a new cluster *newlist* is formed. *element* is added to *newlist*, which is added to the set of clusters *Clusters* (lines 16–18). NNC outputs *Clusters* which is the set of clusters representing different contiguous regions of cloud cover.

The parallel data analysis algorithm is executed simultaneously on a different set of processors than the processors running the WRF simulation. Hence execution of PDA does not affect WRF execution times. In Algorithm 4.1, the analysis of QCLOUD values in each split file is done in parallel because this is the most time-consuming step. For a maximum of 1024 split files, experiments show that the number of elements gathered at the root process is less than 200 for most of the time steps. The sequential NNC algorithm (Algorithm 4.2) takes less than a second to cluster such few values. In this case, parallel clustering would have been an overkill for online analysis. However, we

```
   Input: Sorted array qcloudinfo
   Output: Clusters: List of elements, clustered by proximity
 1 Clusters = ∅;
 2 LOOP: foreach element ∈ qcloudinfo do
 3     if (element.qcloud ≥ threshold and element.olrfraction ≥ threshold) then
           /* Check if this element is physically close to any member of any list  */
 4         foreach list ∈ Clusters do
 5             foreach member ∈ list do
 6                 if (DISTANCE (element,member,list,1)) then
 7                     Add element to list;
 8                     Continue next iteration of LOOP;
 9                 end
10                 if (DISTANCE (element,member,list,2)) then
11                     Add element to list;
12                     Continue next iteration of LOOP;
13                 end
14             end
15         end
           /* Form a new list                                                  */
16         Initialize newlist;
17         Add element to newlist;
18         Add newlist to Clusters;
19     end
20 end
21 Return Clusters;
22 Begin Function DISTANCE (element, member, list, hop)
23 if (distance between member and element == hop) then
24     OldMean = Mean of QCLOUD values of members of list;
25     NewMean = Mean of QCLOUD values of members of list and element.qcloud;
26     if (NewMean is within 30% of OldMean) then
27         Return True;
28     end
29 end
30 Return False;
31 End Function DISTANCE
```

Algorithm 4.2: Nearest Neighbour Clustering (NNC) algorithm

would like to parallelize the NNC algorithm in future for simulations on larger number of processors.

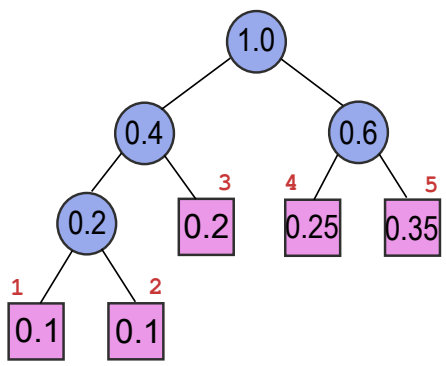
4.3 Processor Allocation

The parallel data analysis (PDA) algorithm computes a set of regions of interest (ROI) in the domain, which in our case are the regions with high cloud cover. Nested simulations are spawned over the regions of interest. We simulate these nests at high resolutions for better accuracy. The resolutions of these nested simulations are thrice that of the parent simulation. We modified the WRF code to spawn nests on-the-fly without stopping the simulation. The initial data for the nested domains are interpolated from the parent domain.

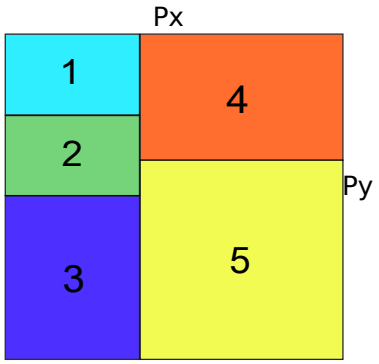
We showed in Chapter 3 that significant performance improvements can be achieved by executing the nests simultaneously on different subsets of the total number of processors, P. We use the performance modeling and Huffman tree based algorithm described in Chapter 3 to determine the size of the subset of processors for a nest and the position of the subset in the processor grid $Px \times Py$ where $Px \cdot Py = P$. The performance model is used to predict the execution times of nests based on the size and aspect ratio of the nests. The Huffman tree based algorithm is used to determine the initial processor allocation for each nested domain.

An example of processor allocation for 5 nests is shown in Figure 4.2. Assume that the ratios of the predicted execution times of the nests are 0.1 : 0.1 : 0.2 : 0.25 : 0.35. These ratios are used as weights in the construction of the Huffman tree, as shown in Figure 4.2(a). The corresponding processor sub-grid for each nest is shown in Figure 4.2(b). The 5 sub-rectangles correspond to the set of processors that execute each of the nests. The start rank i.e. the rank of the processor at the north-west corner of the sub-rectangle and the rectangular dimensions of each processor sub-grid for this example configuration are shown in Table 4.1 for a maximum of 1024 cores.

The regions of interest may persist in time or disappear in subsequent time steps. Our regions of interest are the regions with high cloud cover. Clouds may form and disappear



(a) Huffman tree for 5 nests with execution times in ratios of 0.1 : 0.1 : 0.2 : 0.25 : 0.35



(b) Sub-division of the processor grid $Px \times Py$ for the 5 nests.

Figure 4.2: Illustration of processor allocation for nests.

Table 4.1: Processor allocation on 1024 cores

Nest ID	Start Rank	Processor sub-grid
1	0	13×8
2	256	13×8
3	512	13×16
4	13	19×13
5	429	19×19

over a period of time. The PDA algorithm is invoked periodically (every 2 minutes) to detect regions of interest (ROI) in the output of the current simulation time step. A nest is spawned whenever a new ROI is detected. A nest is deleted when an existing ROI is not output by PDA. A *retained* nest is one which was output by PDA in the previous invocation as well as in the current invocation. The insertion, deletion and retainment of nests cause changes in the Huffman tree structure and hence in the processor allocation. Therefore the newly allocated set of processors (*receivers*) executing a retained nest may not be the same as the previously allocated set of processors (*senders*) for the nest. The *senders* need to distribute the nest domain data to the *receivers*. We modified the

WRF code to execute this redistribution. First the amount of data to be redistributed is calculated based on the nest size, followed by MPI_Alltoallv to redistribute data for each nest. The processors that are neither senders nor receivers for a nest send and receive 0 value during the MPI_Alltoallv for that nest.

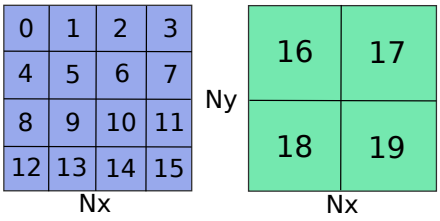


Figure 4.3: Data redistribution from old to new set of processors assigned to a nest.

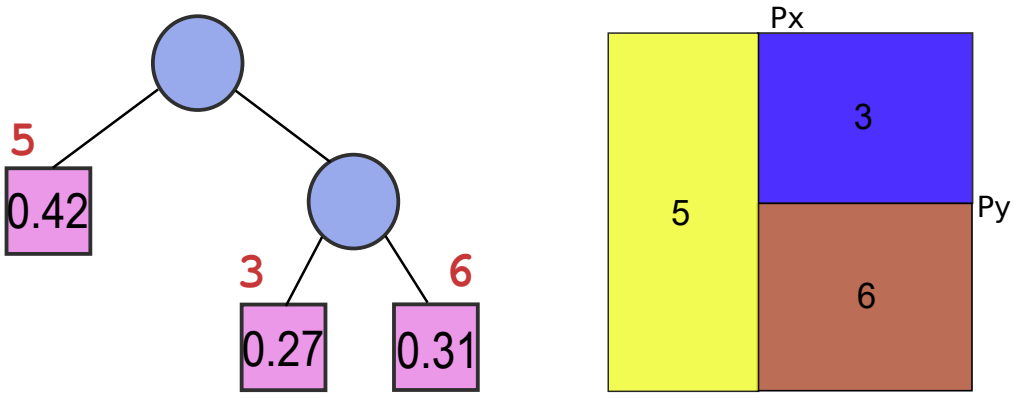
An example is shown in Figure 4.3 for a nest size of $Nx \times Ny$. A nest is equally subdivided among its allocated processors $0-15$ as shown in the left grid. These processors distribute the nest data to the newly allocated processors $16-19$ as shown in the right grid of the figure. It can be observed that the region of the nest domain that processor 16 owns was previously owned by $0, 1, 4, 5$. Hence 16 receives the domain data from $0, 1, 4, 5$. Similarly, the other receivers also receive data from 4 senders in this example.

In the above example, the senders and receivers are non-intersecting sets. The communication cost for the data redistribution between the senders and the receivers can be minimized if the senders and receivers overlap. In torus networks, minimizing the number of hops between the senders and receivers can minimize the redistribution cost. We describe two strategies for data redistribution in the next section.

4.3.1 Partition from scratch

In this approach, we partition the entire process grid $Px \times Py$ for processor allocation based on Huffman tree constructed using the predicted execution times of the nests as weights, as explained in the previous section. The tree construction does not consider the current allocation of processors. Hence this strategy can lead to completely non-overlapping senders and receivers, which will lead to increased redistribution cost.

For example, let us consider the configuration in Figure 4.2. Assume that in the



(a) Huffman tree for nests $3, 5, 6$ with execution times in ratios of $0.27 : 0.42 : 0.31$.

(b) Sub-division of the processor grid $Px \times Py$ for 3 nests.

Figure 4.4: Processor allocation for nests using partition from scratch.

next invocation, PDA outputs the nests $3, 5, 6$ as regions of interest. So the nests $1, 2$, and 4 will be deleted and new nest 6 will be formed. Let the ratios of the predicted execution times of the nests $3, 5, 6$ be $0.27 : 0.42 : 0.31$. The corresponding Huffman tree and the processor partition are shown in Figure 4.4. The start rank and the rectangular dimensions of each processor sub-grid for each nest are given in Table 4.2 for a maximum of 1024 cores. Comparing the previous and the new allocation for nests 3 and 5 from Tables 4.1 and 4.2, we can observe that there is no overlap between senders and receivers. This can increase the redistribution cost.

Table 4.2: Processor allocation on 1024 cores

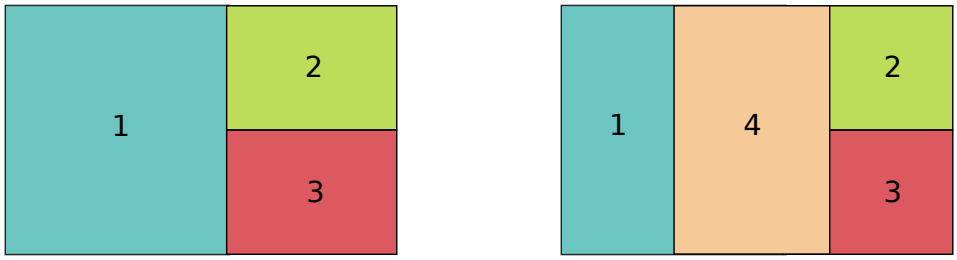
Nest ID	Start Rank	Processor sub-grid
3	13	19×13
5	0	13×32
6	429	19×19

The redistribution cost may be high in some cases in this approach. However, the rectangular partitions based on the Huffman tree are as square-like as possible owing

to the tree construction in the order of increasing weights. The square-like partitions minimize the execution times of the nests.

4.3.2 Tree-based hierarchical diffusion

In this approach, we try to maximize the overlap between senders and receivers of the *retained* nests. The key idea is to shift the boundaries of rectangular partitions for the retained nests so that the distribution of data is among neighbouring processes and the overlap in the nest data between the old and new set of processes is maximized. This minimizes the redistribution cost, especially on torus networks. An example is illustrated in Figure 4.5. Figure 4.5(a) shows the existing processor partitioning for nests $1, 2, 3$.



(a) (b)

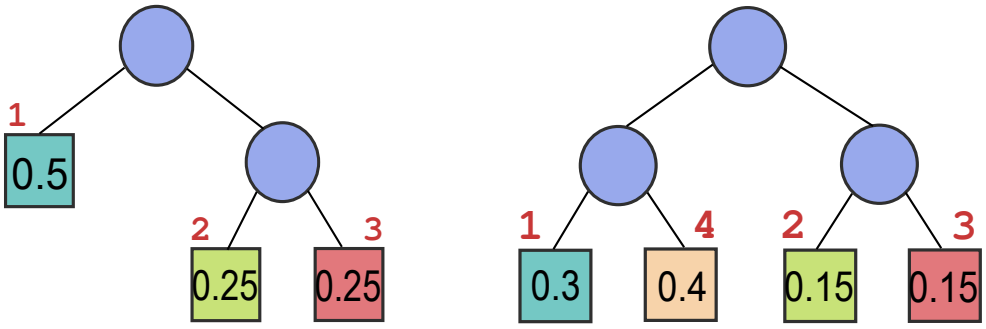
Figure 4.5: (a) Existing and (b) new processor allocation in the hierarchical diffusion approach.

When a new nest is added, the existing partitions are shrunk. In this example, the right boundary of rectangle for nest 1 is shifted to the left and the left boundaries of the nests 2 and 3 are shifted to the right, thereby leaving some processors free for inserting the new nest, as shown in Figure 4.5(b). This also leads to a large overlap between the old and new processor partitions for nests $1, 2, 3$.

This repartitioning method is based on modifying the tree corresponding to the current allocation, rather than building the Huffman tree from scratch. The positions of the nodes corresponding to the retained nests are kept intact in the tree. Note that the weights of the old nodes, i.e. the retained nests, may be modified because the weights represent the ratios of the number of processors that will execute each nest. When new

nests are added and/or old nests are deleted, the processor shares of the existing nests may change.

When there is no deletion, and there is only insertion of new nodes, they are inserted near those existing nodes whose weights are similar to that of the new nodes. By inserting a new node near a node in the Huffman tree with similar weight, we attempt to obtain rectangular partitions for the nests that are more square-like. However, inserting a new node near a node with large difference in weights will lead to skewed rectangles. Square-like partitions lead to smaller executions times for the nests, while skewed rectangular partition increases the execution time of a nest.



(a) (b)

Figure 4.6: (a) Existing and (b) new trees in the hierarchical diffusion approach. Predicted execution time ratios of the nests are the weights in the leaf nodes.

The existing and the new trees corresponding to the processor partitions of Figure 4.5 are shown in Figure 4.6. The new tree in Figure 4.6(b) is constructed by inserting node 4 near node 1. This is because the weight of node 4 is closest to that of the new weight of node 1. The size of a partition that each node gets is proportional to its weight. Thus, the nodes 1 and 4 get $\frac{3}{7}^{th}$ and $\frac{4}{7}^{th}$ of the processors allocated to their parent node. Since the difference in weights of nodes 1 and 4 is less, so the resulting rectangles for 1 and 4 will be as square-like as possible. Note that this would not have been the case if node 4 was inserted near node 2 whose weight is 0.15. This is because the corresponding shares for 4 and 2 would have been $\frac{0.4}{0.55} = \frac{8}{11}$ and $\frac{0.15}{0.55} = \frac{3}{11}$. Thereby the rectangle for node 2 would not have been square-like due to the large difference in weights. This is illustrated

in Figure 4.7. One can note that rectangle 2 is skewed as compared to rectangle 4.

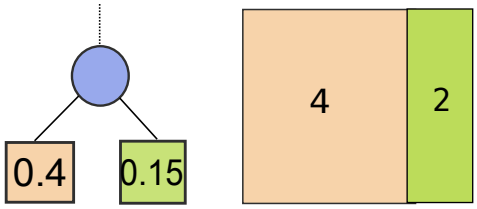


Figure 4.7: Skewed rectangle due to large difference in weights of the two nodes.

When nests are both inserted and deleted, the nodes corresponding to the deleted nests are deleted from the tree. Further, new nodes are inserted in the positions of deleted nests so that the positions of the retained nests remain intact as much as possible. This may enhance the chance of overlapping old and new nest processor allocations for the retained nests. The algorithm for modifying the existing tree for new processor allocation is detailed in Algorithm 4.3. The inputs are the existing tree *oldtree*, the list of deleted nodes *deletednodes*, the modified weights of the retained nests *rweights* and the weights of the new nodes *nweights*. The output is the modified tree *newtree*.

Firstly, nodes from *deletednodes* are marked as free in *oldtree* and added to the set *freenodes* (lines 2–6). The siblings of these nodes are added to the set *siblings* (line 5). These are used later as insertion points. The weights of the retained nodes are modified (lines 7–9). Based on the deletion and modification of weights of retained nodes, the weights of the internal nodes are updated (line 10). The new weights are added in the positions of the deleted nodes (lines 11–17). As explained above, the new nodes should be inserted near the ones who have closest weights. So, we inspect the weights of the sibling nodes of the deleted nodes. Inserting a new node in the place of a deleted node will lead to minimum modification of the existing tree structure. This is shown in line 13. *new_weight* is inserted in the position of *node*, which was marked empty earlier. *node* is selected such that the difference between the weight of its sibling *sibnode* and *new_weight* is minimum. *node* and *sibnode* are deleted from their respective sets (lines 14–15).

Note that the operation in line 13 is done only when there are multiple nodes in the

Input: Existing tree *oldtree*, list of deleted nodes *deletednodes*, new weights of retained nests *rweights*, and weights of new nests *nweights*.

Output: New tree *newtree*

```
1  freenodes = ∅, siblings = ∅;
2  foreach node ∈ deletednodes do
3      Mark node as free in the oldtree;
4      Add node to freenodes;
5      Add sibling of node to siblings;
6  end
7  foreach weight ∈ rweights do
8      Update weight for the corresponding retained node;
9  end
10 Update weights of internal nodes of oldtree;
   /* Insert in the positions of deleted nodes, near to the nodes with closest
      weights                                                                  */
11 foreach new_weight ∈ nweights do
12     if (|freenodes| > 1)) then
13         Add new_weight to the position of node whose sibling's weight is closest to new_weight ;
14         node from freenodes;
15         Delete sibnode from siblings;
16     end
17 end
18 if (|nweights| ≥ |deletednodes|)) then
19     Build Huffman tree for the remaining new weights rooted at node ∈ freenodes;
20 else
21     Delete the remaining nodes in freenodes from oldtree;
22 end
23 Copy oldtree to newtree;
```

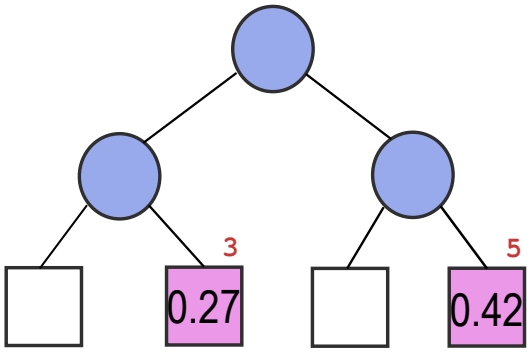
Algorithm 4.3: Tree-based hierarchical diffusion algorithm

set $freenodes$. This is because when the number of deletions is less than the insertions, we build Huffman tree using the remaining unmatched weights in *nweights*, and this subtree is rooted at the position of the last element in $freenodes$. This is shown in lines 18–20. If there are fewer insertions than deletions, we delete the remaining nodes of $freenodes$ (line 21). The updated *oldtree* is output as *newtree*.

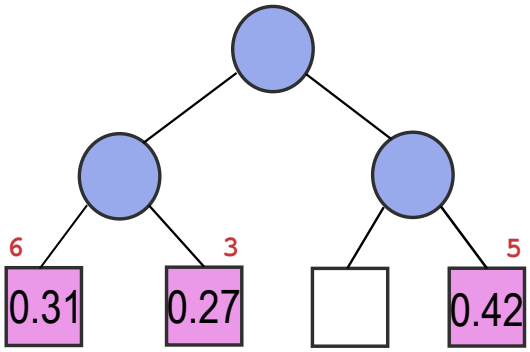
This approach reduces the data movement between the senders and receivers and hence achieves significant reduction in redistribution time as compared to the partition from scratch method. This is because we attempt to allocate receivers such that there is a large overlap between senders and receivers and the receivers are neighbouring processes of the senders.

The processor allocation using tree-based hierarchical diffusion algorithm for the example in Figure 4.2 is shown in Figure 4.8. To compare with the *partition from scratch* approach, let us assume the same output of PDA that was considered in Section 4.3.1 (see Figure 4.4). The nests $1, 2$ and 4 are deleted, nests 3 and 5 are retained and 6 is the new region of interest. Figure 4.8(a) shows the tree after nodes $1, 2$ and 4 are marked as deleted and weights of 3 and 5 are modified. Note that deleted nodes $1, 2$ have been combined as one empty node because the two free rectangles represented by them can be considered as one free rectangle. Hence there are two free slots available for inserting new node 6 - the weight of one sibling node is 0.27 and that of the other is 0.42. Node 6 is inserted in the position of sibling of node 3 because $0.31 - 0.27 < 0.42 - 0.31$ i.e., the weight of node 3 is closer to weight of node 6. The rectangular partitioning based on this tree is shown in Figure 4.8(d). Comparing this with the partitioning obtained from the partitioning from scratch method shown in Figure 4.4(b), we can see that there is considerable overlap between the old and new set of processors for nests 3 and 5, as compared to no overlap in the partition from scratch approach. Also, we observe that the rectangles for 3 and 5 expand to neighbouring processes because we try to keep the positions of retained nests as intact as possible.

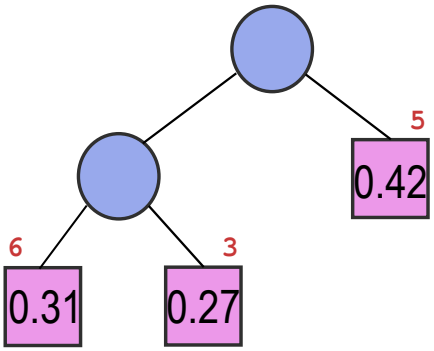
Note that the resulting modified tree may no longer be a Huffman tree in this approach. However, the modifications may lead to some overlap between new and old



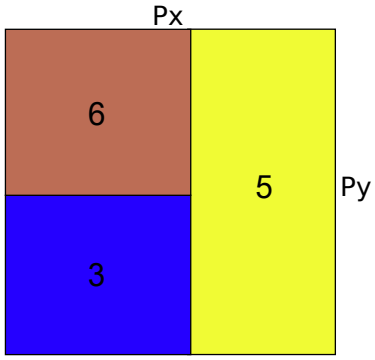
(a) Deleted nodes marked empty and weights of retained nests modified.



(b) Node 6 inserted near node 3



(c) Remaining deleted nodes removed



(d) Sub-division of the processor grid based on the modified tree.

Figure 4.8: Steps of the tree-based hierarchical diffusion algorithm for deleting nests $1, 2, 4$, retaining nests $3, 5$ and adding new nest 6.

processors and redistribution among neighbouring processes. Our techniques are scalable for large number of processors. Also, the maximum number of hops between old and new set of processors is likely to increase for the scratch method with larger total processor count. Therefore the data redistribution time may increase with increase in number of processors for the scratch method. Processor reallocation via Huffman tree construction or reorganization depends on the number of nests and is not affected by increase in processor count.

4.3.3 Dynamic Strategy

The performance differences between the two methods, namely, the partition from scratch method and our diffusion-based method, depend on both the execution times of the resulting partitions and the redistribution costs. The execution time ratios of the nests and hence the percentage of total number of processors allocated for the nests are same in both partition from scratch method and our diffusion-based method. However, due to integral sides of the sub-rectangles, the rectangular grids and the aspect ratios of the rectangles for the same nest configuration may not be exactly the same. For example, one method may allocate 16×18 while the other may allocate 17×17. This can lead to slight difference in execution times of the nests for the two methods.

Similarly, while we expect the redistribution costs for our diffusion-based method to be smaller than the partition from scratch method, there may be cases when the redistribution costs are almost same in both approaches. This is because both approaches are based on tree construction using the ratios of predicted execution times of nests as weights. The relative order of the weights affect the construction of the tree, and hence also affects the resulting rectangular processor grid allocated to the nests. Similar relative order of the weights of those nests that persist between reconfigurations may result in similar trees for both approaches, and hence similar redistribution costs. Therefore we propose a dynamic strategy that selects the approach which requires minimum redistribution time and execution time. For this, we need to predict both these times.

Performance model for redistribution time

The primary component of the redistribution time is MPI_Alltoallv between the processors. We assume direct algorithm for MPI_Alltoallv [61] between the processors in mesh and torus based networks. We predict MPI_Alltoallv time as the maximum communication time between senders and receivers. First, we find the size of the message that a sender will send to its receiver(s), and then find the number of hops between the sender and its receivers. Using this, we find the communication time for every sender-receiver pair. The maximum of these communication times is predicted as the time for

MPI_Alltoallv. For non-mesh networks like switched networks, the times taken for sender to send messages to all receivers can be added to predict the time for MPI_Alltoallv.

Performance model for execution time

We profiled the execution times of a small set (size = 13) of domains with different domain sizes on a few (10 in our case) processor sizes within the maximum number of processors (1024 in our case). The actual execution times of these 13 domains are used to interpolate the execution times of the nests formed in our simulation using Delaunay triangulation. The details of these steps are presented in Chapter 3. Additionally, we predict the execution times of the nests for the 10 processor sizes. Using these times, we perform linear interpolation to predict the execution time on desired number of processors. This gives good prediction accuracies as shown later in Section 4.4. The prediction execution times are used for dynamic selection of methods, and also for determining the weights of the nests needed for processor allocation in the partition from scratch and our tree-based methods.

Using the above predictions for redistribution and execution times for both scratch and tree-based approaches, the dynamic scheme selects the one which has lower sum of these times.

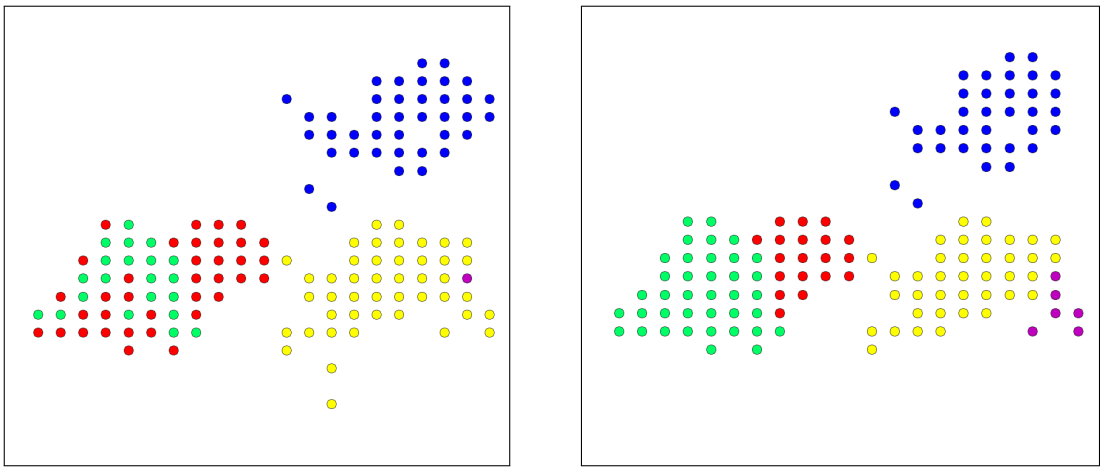
4.4 Experiments and results

4.4.1 Data analysis algorithm

One of the primary components in our work is the data analysis algorithm described in Section 4.2 to identify clouds and form nests. We form clusters of contiguous regions with high cloud cover using QCLOUD values in non-increasing order. A QCLOUD value in this list represents the aggregated QCLOUD over a subdomain, where OLR ≤ 200. The contiguous regions are clustered based on the proximity between the subdomains.

In this section, we compare our nearest neighbour clustering algorithm described with a simple nearest neighbour clustering approach. In Figure 4.9(a), we show the clustering

using only 2 hop distance criteria. This strategy checks whether the list element is within 2 hops from an existing cluster. We can observe there are some overlapping clusters.



(a) Nearest neighbour clustering using 2-hop distance and no mean deviation criteria. Clusters overlap in space.

(b) Nearest neighbour clustering using 1-hop and 2-hop distances and mean deviation threshold of 30%. Clusters do not overlap.

Figure 4.9: Nearest neighbour clustering for our parallel data analysis algorithm.

In Figure 4.9(b), we show the clusters formed by our method. It can be observed that the clusters formed by our method are non-overlapping because we first check for 1 hop and then 2 hop distance. We check for 2 hop distance only if the list element is not within 1 hop from an existing cluster. This ensures that the list element is added to its nearest cluster. We insert into a cluster only if the mean deviation is not more than 30% to ensure that the cluster size does not grow uncontrollably.

4.4.2 Domain Configurations

We modified the WRF source code for dynamic insertion and deletion of nested domains. We simulated over the Indian region from 60°E - 120°E and 5°N - 40°N for the July 2005 Mumbai rainfall event [111]. The period of simulation was from July 24, 2005 18:00 hours – July 27, 2005 18:00 hours. The parent simulation resolution was 12 km and the resolutions of the nested domains were 4 km. We compared our tree-based hierarchical

diffusion approach with the partition from scratch method for both real and synthetic test cases. For the dynamic approach, we experimented with synthetic test cases.

Real: Nests were formed over regions with high cloud cover, which were detected by our parallel data analysis algorithm. The maximum number of nests formed during these runs were 7. The maximum and minimum sizes of the nests formed were 202×349 and 175×175. There were approximately 100 reconfigurations of processor allocations for the nests.

Synthetic: The real traces for our application had fewer configuration changes and fewer $(4 - 5)$ nests on average. We generated some synthetic test cases in order to test our algorithm for higher number of nests in a time step and more number of redistributions per adaptation point. We tested with upto 70 random nest configuration changes, with number of nests varying between $2 - 9$. Nests were randomly inserted and deleted. The maximum and minimum sizes of the nests formed were 361×361 and 181×181.

4.4.3 Experimental Setup

We performed our simulations on two different kinds of systems, a Blue Gene/L system and an Intel Xeon cluster called *fist*. Table 4.3 details our experimental configurations.

Table 4.3: Simulation Configurations

Simulation Configuration	*Maximum Number of Cores*
Blue Gene/L: Dual-core 700 MHz PowerPC 440 processor cores with 1 GB physical memory, 3D torus network	1024
fist: 2 Xeon quad core processors (2.66GHz, 12MB L2 Cache) with 16GB memory, connected by Infiniband switched network	256

For the experiments on Blue Gene/L [126], we developed a folding-based topology-aware mapping [145] that maps the neighbouring processes to neighbouring processors

on the 3D torus. This topology-aware mapping was used for all our experiments so that the processes are one hop away from their neighbours in the process grid. This also benefits the execution times for both the partition from scratch method and diffusion based approach.

For all our experiments, visualization was performed on a graphics workstation in Indian Institute of Science (IISc) with a dual quad-core Intel® Xeon® E5405 and an NVIDIA graphics card GeForce 8800 GTX.

4.4.4 Improvement in redistribution time

Our tree-based hierarchical diffusion method achieved 14% and 12% improvements in redistribution times on 512 and 1024 Blue Gene/L cores respectively over partition from scratch method for the real test cases.

Table 4.4 shows the average percentage improvement in redistribution times for our tree-based hierarchical diffusion method over partition from scratch method for the synthetic test cases. It can be observed that the performance improvement is higher in

Table 4.4: Average improvement in redistribution times for synthetic test cases

Simulation Configuration	*Improvement*
BG/L 1024 cores	15%
BG/L 256 cores	25%
fist 256 cores	10%

the case of Blue Gene/L which has 3D torus network. This is because our tree-based hierarchical approach selects the new processor allocation based on the neighbours in the process grid. For Blue Gene/L the neighbours in the process grid are also neighbours in the processor topology because of our topology-aware mapping. However, in the *fist* cluster, there is no regular mesh/torus topology, hence the gains are lower. However, it is important to note that we still achieve 10% improvement over the scratch method because of the overlap between the senders and receivers in our approach. Maximum overlap ensures less data communication during the redistribution. We also observe

higher improvement for 256 cores. We assume that this may be because of larger per-core data for redistribution in the case of smaller number of cores.

For both real and synthetic test cases, we observed an average of 4% increase in execution times for our approach over the partition from scratch method. This is because in our approach, the Huffman tree is not constructed from scratch and we try to maximize the overlap. Hence the resulting partitions may not always be square-like. However, when the number of adaptation points is high, it is more important to minimize the redistribution cost.

4.4.5 Distance between senders and receivers

Figure 4.10 shows the average hop-bytes during the sender-receiver communication for partition from scratch and our approach for 70 synthetic test cases on 1024 Blue Gene/L cores. The hop-bytes metric is the weighted sum of message sizes where the weights are the number of hops (links) traveled by the respective messages. Higher hop-bytes

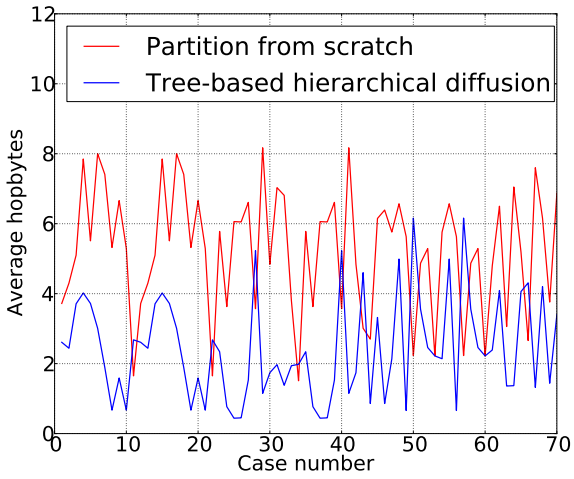


Figure 4.10: Average hop-bytes for partition from scratch method and tree-based hierarchical approach. X-axis denotes the test case number and Y-axis denotes the hop-bytes. Tree-based hierarchical approach incurs lesser hop-bytes than scratch method.

is an indication of higher communication load on the network [12]. It can be seen that the average in the case of partition from scratch is 5.25 whereas in our approach the

average is 2.44. This is because in our strategy the receiver process grid is placed closer to the sender process grid so that the number of hops between a sender-receiver pair is minimized.

Figure 4.11 shows the percentage of overlap of data points between the senders and receivers for partition from scratch and our approach for 70 synthetic test cases on 1024 Blue Gene/L cores. It can be observed that the overlap is higher for our method and hence our approach incurs lesser redistribution time.

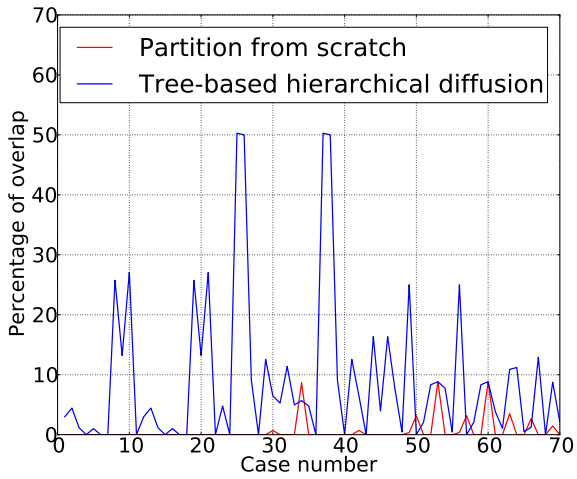


Figure 4.11: Percentage overlap between senders and receivers for partition from scratch method and tree-based hierarchical approach. X-axis denotes the test case number and Y-axis denotes the percentage overlap. Tree-based hierarchical approach has more overlap than scratch method.

In the case of *fist* cluster, we found that there was an overlap of 27% data points between senders and receivers for our tree-based hierarchical approach. For the scratch method, there was 15% overlap. This is because in our method, we try to maximize the overlap between senders and receivers so that there is less data communication during the redistribution.

4.4.6 Dynamic Approach

In this section, we present the results for our dynamic scheme which selects either the scratch or the tree-based approach. We tested 12 reconfigurations for synthetic cases on 1024 BG/L cores for a simulation period of 4 hours. The approach with the minimum sum of predicted execution and redistribution times was selected by the dynamic approach. Since the efficiency of dynamic selection approach depends on the ability to predict the execution times of different nest configurations, we calculated the Pearson's correlation coefficient between the actual and predicted execution times. We found that our prediction method yielded Pearson's correlation coefficient of 0.9. This shows linear relationship between the two and hence also shows that our performance prediction for execution times is nearly accurate.

Out of the 12 reconfiguration cases, scratch method was selected two times and tree-based approach was selected ten times. The dynamic approach made correct decisions in 10 out of the 12 cases. In terms of actual execution times, our tree-based diffusion method gave smaller sum of execution and redistribution times than partition from scratch method in 9 cases, while the partition from scratch method gave smaller sum in the remaining 3 cases.

Figure 4.12 shows the total times including the execution times and redistribution times for tree-based approach, partition from scratch method and dynamic approach. It can be observed that the redistribution time is lowest in our tree-based method, while the execution time is the lowest in the partition from scratch method. The dynamic selection approach combines the advantages of both the methods, with its redistribution time similar to the tree-based approach and its execution time similar to the partition from scratch method. The dynamic scheme resulted in 3% improvement in overall execution time than the next best-performing tree-based approach. It should be noted that more frequent adaptation points seen in our real runs (about 70 adaptation points) will result in higher performance improvement for the dynamic scheme.

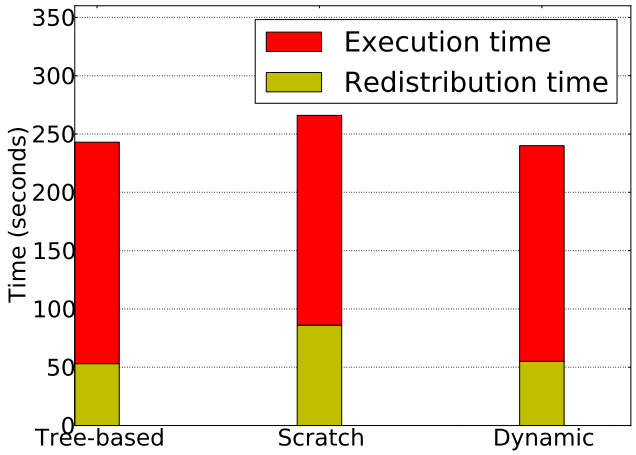


Figure 4.12: Execution and redistribution times.

4.5 Discussion

In this section, we discuss how our techniques can be useful for other applications and can be applicable on other platforms.

4.5.1 Use of our Techniques for Other Applications

There are many applications in various other scientific domains where our methods can be applied and prove useful. Some examples are flow dynamics, and vehicle dynamics. In the case of multi-scale flow simulations, macroscopic flow phenomenon in a large computational flow domain is simulated using continuum approach whereas the more compute-intensive molecular dynamics (MD) simulations track the motion of individual discrete atoms in small spatial regions [122]. Detailed MD simulations are required for atomistic physical phenomena which can vary spatially and temporally. In the case of vehicle dynamics, multibody simulations are used to simulate the entire vehicle whereas the more sophisticated finite element method (FEM) simulations are applied to study structural deformations to tires on rough surfaces. FEM simulations are dynamically spawned based on road conditions [9]. In both these cases, our approach of performance modeling and processor allocations can be used for the detailed simulations. For

performance prediction, the parameters for interpolation need to be determined by the scientist/user. Our processor allocation and reallocation algorithms for detailed simulations can be used to determine the partitioning and redistribution with minimum data movement. Another such application domain is fluid-structure interactions where fluid flow causes deformation of the structure and boundary conditions of the fluid flow are altered [40]. Deformations can dynamically occur, thereby requiring remeshing into finer grids in certain regions of the domain.

Our performance model and resource allocation strategies can also be extended to be used for visualization of cosmological simulation output where the parent domain is dynamically refined into coarser levels [62]. Our methods can be extended to coupled simulations, for e.g., climate simulations like Community Earth System Model [43]. The long running climate simulations incur temporal variability in invoking certain computations in the model components based on atmospheric dynamics, dynamic vegetation, aerosol formation and removal etc. Thus, they require remapping of components in certain time-steps. The processor allocations and reallocations for model components can be determined using our strategies. However, in coupled simulations, since there are few communications between the independent components, an additional constraint of distance between communicating components need to be considered when mapping the sub-tasks onto sub-partitions.

4.5.2 Use of our Techniques for Other Platforms and Interconnects

Although we have shown results with IBM Blue Gene, our techniques can be applied to other systems with torus networks that are widely prevalent. Our data redistribution strategies will be effective on custom interconnects as well as clusters that do not have regular interconnect topologies. The processor allocation and reallocation algorithms are based on Huffman-tree construction using sub-domain sizes and do not assume anything about the underlying network. Hence, they can be directly used on any platform with any interconnect.

Our mapping heuristics are also applicable on other regular custom interconnect

topologies where jobs are allocated in contiguous partitions. The mapping heuristics discussed in Section 3.5, are based on folding of virtual topology on the physical network. For mesh and torus-based topologies, folding can be achieved in a similar way as discussed in this thesis. However, rethinking is required for mapping algorithms for multi-level direct networks [13] and fat-tree topology [68]. The 2D virtual topology of the application cannot be folded on a graph/tree-based topology trivially. Lower communication times can be achieved by mapping a sub-domain on to adjacent groups in the lower levels of these kinds of hierarchical networks.

4.6 Summary

In this chapter, we presented a parallel data analysis algorithm and efficient processor reallocation algorithm to detect and track tall clouds in tropical weather systems. Our data analysis algorithm detects organized cloud systems using a variant of nearest neighbour clustering. We performed nested high-resolution simulations for the regions with high cloud cover. The nested simulations were executed on a disjoint subset of the total number of processors. Due to the dynamic nature of the clouds, the nests may form and disappear with time. We proposed a tree-based efficient processor allocation algorithm that reallocates processors for the persistent nests at a low data redistribution cost.

Our approach considers the existing processor allocation and selects a new subset of processors with maximum overlap with the existing rectangular subset of processors. Results showed that we are able to reduce the redistribution times by upto 25% as compared to partition from scratch method with minimum increase in the execution times. We also developed a dynamic scheme that attempts to select the best of the two approaches, namely, partition from scratch and our approach.

Our detection and tracking algorithms are quite generic. In future, we would like to apply these algorithms for other applications which require simultaneous tracking of multiple dynamic events.

Weather simulations produce large amounts of output data, which needs to be quickly visualized for comprehending the simulation output. On-the-fly visualization becomes

more important in cases of weather conditions like depressions and cloud formations, which can result in severe weather phenomena like cyclones and heavy rainfall. In the next chapter, we describe an adaptive framework for simultaneous simulations and on-the-fly visualization that decides execution parameters based on the resource characteristics like network bandwidth, computation speed and stable storage space.

Chapter 5

Simultaneous Simulation and Visualization

Conventional post-processing of simulation data suffers from shortcomings like delay between simulation and visualization times. Recent efforts focus on approaches for online visualization [75, 132]. In this chapter we discuss our adaptive framework for loosely coupled simulation and online visualization.

5.1 Introduction

Critical weather applications like cyclone or hurricane tracking and earthquake modeling require high-performance and high-fidelity simulations to obtain real-time forecasts and high-resolution visualization by the climate scientists for subsequent analysis. For timely analysis and rapid response, these applications require online/"on-the-fly" visualization simultaneously performed with the simulation. This will enable the scientists to provide real-time guidance to policy and decision makers, and feedback control for refining the simulation. Remote visualization, where the visualization is performed at a location different from the site of simulation, can enable geographically distributed climate scientists to share vital information, perform collaborative analysis, and provide joint guidance on critical weather events, and hence can help harness the expertise of a large climate science community.

Such high performance simulation and simultaneous visualization involve the use of large stable storage or disk for storing the weather data and networks for transfer of data

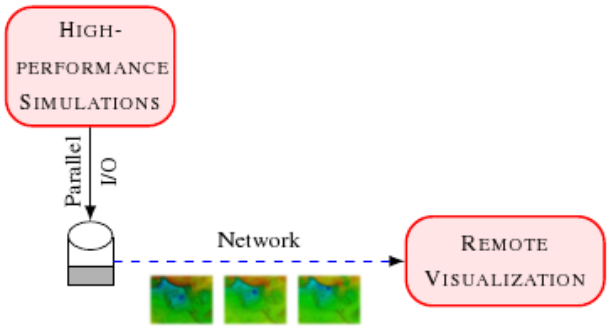


Figure 5.1: Illustration of simultaneous simulation and remote visualization using stable storage.

from the stable storage to the remote visualization site as shown in Figure 5.1. However, constraints on the size and capacity of the stable storage and the network can limit the effectiveness of online and simultaneous remote visualization of critical weather events. In this work, we assume that the data that is transferred to the visualization site is removed from the simulation site thereby increasing the available free disk space at the simulation site.

5.1.1 Motivation

Contemporary weather simulations have demonstrated very high scalability on large number of modern-day processors [80]. Simulations running on thousands of cores take less than a second of execution time per time step [80]. Parallel I/O can enable very high I/O bandwidth of the range of 5 – 20 GBps on large number of cores [66, 144]. A combination of high simulation rate and high I/O bandwidth leads to high rate of generation of gigabytes of weather data as output and hence rapid accumulation of data in the stable storage. This gives rise to the critical problem of storage limitation for long-running weather applications. The network bandwidth between the simulation and visualization sites impacts the rate at which data is moved out from the simulation site and hence determines the amount of remaining disk space available for simulation output. Furthermore, the continuous development of high resolution simulation models for fine-grained analysis leads to an increase in simulation output data volume and hence exacerbates

the problem of storage space limitation for weather simulations. Eventual unavailability of storage for storing simulation output, due to the disk becoming completely full, can result in either stalling of the simulations or visualization of fewer frames, resulting in loss of visualization of critical weather events.

The problem is illustrated in Table 5.1 that shows the estimated time when the stable storage becomes unavailable for weather simulations (last column). The table shows data for a weather simulation of grid size 4486 × 4486 points, 10 km resolution resulting in about 31 GB of output per time step, execution time of 1.2 seconds for a simulation time step on 16,384 processor cores, I/O bandwidth of about 5 GBps, and for various total disk sizes and network bandwidths. The values for simulation grid sizes and resolutions, parallel I/O bandwidths and execution time on about 16,000 cores are reported and projected in recent research efforts [66, 80, 144]. We find that even the presence of large disk spaces, and fast networks can result in the storage becoming unavailable within few minutes to hours of high-resolution weather simulations that are envisaged to execute for few days to weeks on large-scale machines. This in turn hampers effective remote visualization of critical weather events. Hence it is highly essential to adaptively use the processor space and adjust the frequency of output based on the application and resource dynamics. Such a dynamic solution will ensure that the disk space is always available to store the output of the simulation and the climate scientists are able to constantly monitor progress of the simulation.

5.1.2 Problem Statement

In this chapter, we describe our adaptive framework that simultaneously performs numerical simulations and online continuous remote visualization of critical weather applications in resource-constrained environments. The objective of our framework is to enable continuous progress in simulation and maximize temporal resolution in visualization, considering the limitations in storage and network capacities. We define temporal resolution as the frequency at which successive frames are visualized. High temporal resolution would mean that more number of successively produced frames are visualized.

Table 5.1: Illustration of Disk Space Limitation. Weather simulation of grid size 4486×4486 points, 10 km resolution, execution on 16,384 cores with 1.2 seconds of execution time per time step, and I/O bandwidth of about 5 GBps.

Disk Space	Network Bandwidth	Time when storage becomes full
5 TB	1 Gbps	25 minutes
	10 Gbps	36 minutes
100 TB	1 Gbps	8 hours
	10 Gbps	12 hours
300 TB	1 Gbps	24.5 hours
	10 Gbps	36 hours
500 TB	1 Gbps	41 hours
	10 Gbps	60 hours

Our framework considers both application and resource dynamics including the intensity of weather events, available disk space and the network bandwidth to adapt various application and resource parameters including simulation resolutions, rate of simulation, selecting the number of processors for simulation, and the frequency of data output for visualization. We have developed two algorithms for processor allocation and selecting the frequency of data output for visualization. The first algorithm is a greedy strategy that attempts to maximize the simulation rate and frequency of output while the second algorithm is based on linear optimization that attempts to provide steady-state simulation and visualization rates. We have applied our framework for large-scale and long-range tracking of cyclones. We conducted experiments with our framework for three experiment settings corresponding to inter-department, intra-country and cross-continent visualizations. We show that our optimization method is able to provide about 30% higher simulation rate completing the entire simulation for all network configurations, consumes about 25-50% lesser storage space completely avoiding disk overflow problem and the resulting stalling of simulation, and provides higher and more consistent rate of visualization than the greedy approach.

5.1.3 Chapter Outline

Section 5.2 presents our adaptive framework including the components and interactions. Section 5.2.4 explains our adaptive algorithms for deciding processor allocation and output frequency. Section 5.3 presents our experiments involving different network bandwidths and results including simulation rates. Section 5.4 briefly discusses the generality of the framework and we summarize in Section 5.5.

5.2 Adaptive Integrated Framework

Critical weather application simulations require immediate analysis on the occurrence of critical events. Hence executions of these applications require efficient processor allocation and a robust disk-space management strategy because of the sheer amount of data produced by the simulations. The absence of such a middleware can lead to problems including disk overflow, stalling of simulation, and low temporal resolution.

We have developed an adaptive framework that performs efficient processor allocation and robust disk-space management to handle the large amount of data produced by the simulations. Our framework, shown in Figure 5.2, consists of the following components to perform coordinated simulations and continuous online remote visualizations: an *application manager* that determines the application configuration for weather simulations based on resource characteristics, a *job handler* that coordinates the execution of weather simulations, a *simulation process* that performs weather simulations with different application configurations, *frame sender and receiver daemons* that deal with transport of frames from simulation to visualization sites, and a *visualization process* for visualization of the frames. The following subsections describe in detail the components and their interactions.

5.2.1 Application Manager

Application manager is the primary component that makes our framework adaptive to resource configuration changes. It invokes a decision algorithm periodically or at specified

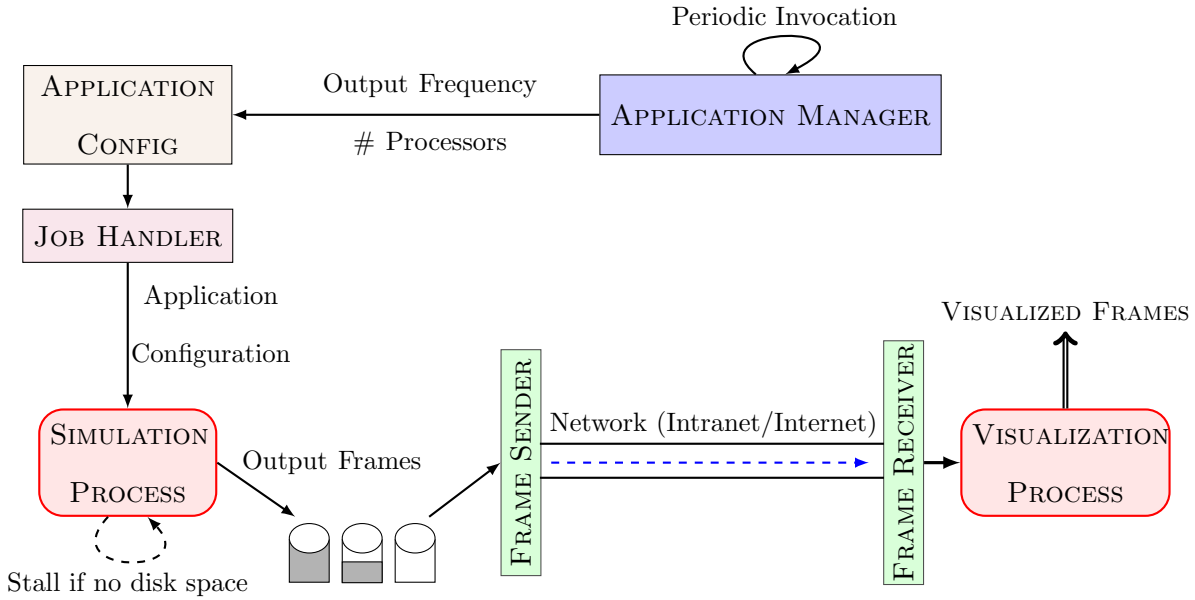


Figure 5.2: Adaptive framework for continuous simulations and online visualization.

times. The decision algorithm considers as input the bandwidth of the network between the weather simulation and visualization sites, the available free disk space, the I/O bandwidth, and the resolution of weather simulation. The application manager also uses average observed bandwidth between the simulation and visualization sites, obtained by using the time taken for sending about 1 GB message across the network. The decision algorithm then determines the number of processors for execution of the simulation and the frequency of output of weather data for continuous visualization. The application manager stores these parameters to an application configuration file. It also notifies other components in the framework if the available free disk space becomes significantly low by setting a *CRITICAL* flag in the application configuration file.

The efficiency of the decision algorithm used in the application manager impacts the rate of simulation and online visualization in our framework. We have developed two decision algorithms for the application manager. These algorithms are described in Section 5.2.4.

5.2.2 Job Handler and Simulation Process

The job handler component is responsible for scheduling the WRF weather simulation application with the application configuration determined by the application manager. The WRF weather simulation process is executed on a certain number of processors with a simulation resolution and frequency of output of the weather data specified in the application configuration. The job handler starts, stops and restarts the simulation process whenever the application configuration changes. The simulation process simulates weather over a specified period of time and produces output for visualization.

The simulation process continuously simulates weather events across time steps and outputs weather data to disks as long as the available disk space is sufficient for accommodating the output. The WRF simulation process also periodically reads the application configuration file written by the application manager. If the available free disk space is significantly low, the application manager sets the *CRITICAL* flag. In this case, the simulation process *stalls* execution, and periodically checks the application configuration file. When the free disk space becomes sufficient again, the application manager resets the *CRITICAL* flag, and WRF continues execution. When the application configuration specified in the configuration file changes from the current configuration used for the WRF execution, the WRF process stops. The job handler then restarts WRF with the new application configuration and continues execution.

In our framework, the WRF simulation process is made to stall if the available free disk space is very low. Continuing the simulation without generating the output will result in large time "gaps" in the visualization of weather events. Therefore, instead of continuing simulation without generating output, we stall the simulation. This is a reasonable strategy since our framework is primarily intended for online and continuous remote visualization.

The modifications to WRF for our application are minimal as explained later in Section 5.3.2. Whenever simulation resolution needs to be changed or configuration change is signalled by the application manager, WRF stops. Job handler reschedules it using a new configuration input from the application manager, using the number of

processors and frequency of output specified in the application configuration file.

5.2.3 Frame Sender and Receiver, and Visualization Process

Critical weather applications require continuous visualization of the simulated output. The *frame sender* daemon continuously checks for the availability of weather data output frames and sends the available frames over the network to the remote visualization site. The *frame receiver* daemon at the remote visualization site receives the frames and invokes the visualization process for visualization of the frames.

5.2.4 Decision Algorithm for the Application Manager

The decision algorithm invoked by the application manager determines

1. the number of processors, and
2. the frequency of output of simulation

for execution of simulations for a given

1. resolution of simulation,
2. the bandwidth of the network connecting the simulation and visualization sites,
3. the available free disk space at the simulation site, and
4. the minimum progress rate (MPR) of simulations desired by the user.

The decision algorithm takes as input the execution times for different number of processors and simulation resolutions. The execution times of a subset of configurations have been experimentally found by running sample WRF runs for simulation time of 1 hour on different discrete number of processors, spanning the available processor space. Nearest neighbour interpolation is used to interpolate the execution times for other number of processors. The algorithm also considers lower bound for output frequency or upper bound for interval between outputs, *upper_output_interval*. This upper bound corresponds to the minimum frequency with which the climate scientist would want to

visualize the weather events. The decision algorithm can be invoked periodically with updated network bandwidth values to consider the scenario of changing network bandwidths and transfer times.

The objective of the decision algorithm is to maximize the rate of simulations and to enable continuous visualization with maximum temporal resolution. However, these objectives are contradictory. Visualization of maximum number of output frames can be achieved by increasing the frequency of output of weather data by simulations but this can increase the time for I/O and hence decrease the rate of simulations. Increasing the frequency of output can also lead to rapid accumulation of output data and hence rapid decrease in the available free disk space, eventually stalling the simulations. On the other hand, decreasing the frequency of output can increase the simulation rate, but will result in visualization of fewer frames.

Faster networks with high bandwidths result in faster transfer of output frames from the simulation to visualization sites and hence results in large-scale freeing of disk space at the simulation site. Also, unlike traditional scheduling, executing the simulations on large number of processors corresponding to the maximum simulation rate may not be the optimum strategy. Sometimes, the simulations may have to be "slowed down" because faster simulations can lead to faster consumption of storage if the network to the visualization site is slow. Thus a decision algorithm has to carefully consider these various dependent impacting factors to achieve a good balance between its contradictory objectives of maximizing simulation rate and temporal resolution of visualization. The decision algorithm considers both application and resource parameters as inputs. The simulation resolution and the minimum progress rate of simulation are application-specific parameters. The resolution of simulation impacts the visualization quality. For example, in the case of cyclone tracking, a climate scientist may want to visualize with coarser resolutions during the initial stages of cyclone formation and with finer resolutions when the cyclone intensifies. The I/O bandwidth, network bandwidth and the free disk space are resource parameters.

We have devised two decision algorithms: a greedy algorithm that uses thresholds for modifying parameters, and an optimization-based approach. The following subsections describe these algorithms.

Greedy-Threshold Algorithm

This algorithm attempts to employ the maximum number of processors for maximum simulation rate and output every simulated time step for maximum temporal resolution. However, since this greedy strategy can result in rapid decrease in available free disk space, the algorithm also uses thresholds for free disk space to dynamically adjust the frequency of output and the number of processors for execution. The algorithm considers two sets of thresholds, *lowdiskspace-thresholdset* when the remaining disk space is low and *highdiskspace-thresholdset* when the remaining disk space is high. For our current work, we set *lowdiskspace-thresholdset* = $\{50, 25\}$, and *highdiskspace-thresholdset* = $\{60\}$. When the remaining disk space is less than an upper bound of *lowdiskspace-thresholdset*, the algorithm first decreases the frequency of output i.e. increases the interval of output, *output_interval*. If the *output_interval* is already equal to its maximum value, *upper_output_interval*, and if the free disk space is still less than the thresholds in *lowdiskspace-thresholdset*, the algorithm "slows down" the simulation or increases the execution time by decreasing the number of processors used for simulation. If the free disk space is less than the lowest threshold in *lowdiskspace-thresholdset*, the algorithm sets the *CRITICAL* flag in the application configuration file, thereby leading to stalling of the simulations.

The observation is that decreasing the rate of simulation and the frequency of output may eventually lead to freeing up of disk space. At some point when the remaining free disk space increases sufficiently, the algorithm follows a reverse process using the thresholds in *highdiskspace-thresholdset*, whereby it increases the simulation rate by increasing the number of processors for execution first. If the maximum simulation rate is achieved and the remaining free disk space is sufficient, then the algorithm decreases the *output_interval*. Thus this algorithm gives more preference to maximizing the simulation

rate than to maximizing the output frequency.

The pseudocode for this is shown in Algorithm 5.1. This algorithm is invoked periodically every 1.5 hours. In the pseudocode, OI refers to the *output_interval*. $oldOI$ and $newOI$ refer to the old and new values of *output_interval*. $minOI$ and $maxOI$ refer to the minimum and maximum values of *output_interval*. $mintime$ and $maxtime$ refer to the minimum and maximum values of execution time per time step of simulation, and correspond to execution with maximum and minimum number of processors respectively.

Input: oldOI, minOI, maxOI, oldtime, mintime, maxtime

1 D ← Remaining free disk space;
2 **if** *(D ≤ 10%)* **then** set CRITICAL flag;
3 **else if** *(D ≤ 50%)* **then**
4 **if** *(D ≥ 25%)* **then**
5 $newOI \leftarrow oldOI + \frac{(50-D)}{25} \cdot (maxOI - oldOI)$;
6 **else**
7 $newtime \leftarrow oldtime + \frac{(25-D)}{15} \cdot (maxtime - oldtime)$;
8 **end**
9 **else if** *(D ≥ 60%)* **then**
10 **if** *(oldtime > mintime)* **then**
11 $newtime \leftarrow oldtime - \frac{(D-60)}{40} \cdot (oldtime - mintime)$;
12 **else if** *(oldOI > minOI)* **then**
13 $newOI \leftarrow oldOI - \frac{(D-60)}{40} \cdot (oldOI - minOI)$;
14 **end**

Output: $newOI$ and corresponding number of processors for $newtime$ to application configuration file

Algorithm 5.1: Greedy-Threshold Algorithm

If the remaining disk space is between 25 and 50%, then OI is decreased (lines 4 – 5) so that the rate at which data is output to the disk is decreased, and hence the disk fills up at a slower rate. If the remaining disk space is lower than 25%, the simulation rate is decreased (lines 6 – 8). If the remaining disk space is more than 60% and the simulation rate is lesser than the maximum possible rate, the simulation rate is increased by increasing the number of processors (lines 10 – 11). If the remaining disk space is

more than 60% and the simulation rate is maximum, then frequency of output of data by the simulation process is increased (lines 12 – 13). The algorithm calculates the new execution time *newtime* for simulation from the previous value *oldtime* in lines 7 and 11 and determines the corresponding number of processors using the benchmark profiling runs with WRF. Thus, the greedy-threshold heuristic takes reactive approach in determining the number of processors for simulation and the output frequency for visualization based on the available disk space thresholds.

Optimization Method

The primary objective of a traditional scheduling problem for parallel simulations is to maximize the rate of simulations. The rate of simulations is typically high for large number of processors and high I/O bandwidth. However faster execution time and higher I/O bandwidth can lead to faster consumption of storage space by the simulations. In addition, if the network bandwidth from the simulation to the visualization end is low, then the disk can overflow soon. It is also interesting to note that the fastest rate of simulation can be achieved and the disk space limitation problem can be avoided in spite of high I/O bandwidth, slow network and small execution time, if the output frequency is 0, i.e. output is not generated by the simulations at all. But for critical weather applications, it is vital to output as frequently as possible in order to perform continuous visualization of the output and monitor the simulation output. However frequent I/O can decrease the simulation rate and also leads to faster accumulation in the storage. Thus we can think of our problem as an optimization problem that primarily attempts to maximize the simulation rate within the constraints related to continuous visualization, acceptable frequency of output, minimum progress rate of simulations (MPR), I/O bandwidth, disk space and network speed.

We formulate our problem as a linear programming problem with constraints to obtain the number of processors and the frequency of output for simulations. The simulation resolution determines the amount of output produced. The I/O bandwidth implies the time to write to disk and the network bandwidth controls the time to transfer data.

Together, the resource constraints control the rate at which the disk is filled up and the rate at which data reaches the visualization site. We consider an important constraint involving the minimum progress rate of simulations, for considering the criticality needs of the application. This constraint is useful for ensuring quality of service to the climate scientist for continuous and fast visualization. We do not consider the actual visualization time in this formulation because visualization is performed in a pipelined fashion. The task is accelerated by the GPU and completes within a second, much before the next frame arrives at the visualization site even in the case of high-bandwidth networks.

Since we want the best possible throughput of the simulation inspite of the resource constraints, we express the objective of our optimization problem as

$$minimize\ t$$

where t is the execution time to solve a time step. The parameters used in the formulation are listed in Table 5.2. Among these parameters, the decision variables involved in the formulation are $\mathcal{S}$, $\mathcal{F}$, $\mathcal{T}$ and t. In the table, a frame is the output of one time step of simulation and corresponds to the smallest unit of simulation output that can be visualized. Interval corresponds to some fixed execution time interval for the simulations. The following sub-sections describe the formulation of the constraints.

Table 5.2: Problem Parameters

t	Time to solve one simulation time step
$\mathcal{S}$	Number of frames solved in an interval
$\mathcal{F}$	Number of frames output in an interval
$\mathcal{T}$	Number of frames transferred in an interval
$\mathcal{O}$	Size of one frame output in one time step
$\mathcal{D}$	Total remaining free disk space
$\mathcal{T}_{IO}$	Time to output one time step
b	Network bandwidth

Time Constraint: Frames should be continuously transferred from the simulation site so that there is minimum stalling at the visualization end. Consider an interval $\mathcal{I}$ when $\mathcal{T}$ frames are transferred, $\mathcal{S}$ frames are solved and $\mathcal{F}$ frames are output. For continuous visualization, the time to produce $\mathcal{F}$ frames should be less than the time to transfer $\mathcal{T}$ frames since the next set of frames should be ready for transfer by the time the current frames are transferred. If the next set of frames are not available, the continuity of the visualization will be affected and the visualization process will incur idling. The time to produce a frame corresponding to a time step includes the time to solve the time step and the time to write the frame onto the disk. Thus, the time to produce $\mathcal{F}$ frames includes the time to solve $\mathcal{S}$ frames and to write $\mathcal{F}$ frames onto the disk. This gives equation (5.1) where *tts* is the time to solve, *tto* is the time to output and *ttt* is the time to transfer. Expanding equation (5.1), we obtain the constraint specified in equation (5.2) which can be rearranged as equation (5.3). The relation between $\mathcal{S}$ and $\mathcal{F}$ is determined by the output frequency for the simulation. For example, if the output frequency is 1 then $\mathcal{S} = \mathcal{F}$, i.e. every frame that is solved is written to the disk.

$$tts + tto \leq ttt \tag{5.1}$$

$$\mathcal{S} \cdot t + \mathcal{F} \cdot \mathcal{T}_{IO} \leq \frac{\mathcal{O}}{b} \cdot \mathcal{T} \tag{5.2}$$

$$t + (\mathcal{F}/\mathcal{S}) \cdot \mathcal{T}_{IO} \leq \frac{\mathcal{O}}{b} \cdot (\mathcal{T}/\mathcal{S}) \tag{5.3}$$

Disk Constraint: If the rate at which the simulation process writes data to the disk is lower than the rate at which data is transferred from the disk, then there is no disk constraint but generally this is not the case. Assuming that the rate of input to the disk from the simulation is greater than the rate at which the simulation output data is transferred to the visualization site, then the time n in which the disk will overflow is given by equation (5.4) where $\mathcal{R}_{in}$ and $\mathcal{R}_{out}$ are the rate of input to the disk and rate of output from the disk respectively. $\mathcal{R}_{in}$ is calculated using the solve time t, the output data size $\mathcal{O}$ and the output interval (inverse of frequency expressed in simulated time units) OI. $\mathcal{R}_{out}$ is calculated using network bandwidth b. From this we derive

equation (5.5) which can be rearranged as equation (5.6).

$$n \leq \frac{\mathcal{D}}{(\mathcal{R}_{in})-(\mathcal{R}_{out})} \tag{5.4}$$

$$\frac{\mathcal{O}\cdot\mathcal{F}}{t\cdot\mathcal{S} + \mathcal{T}_{IO}\cdot\mathcal{F}} - b \leq \frac{\mathcal{D}}{n} \tag{5.5}$$

$$t \geq \left[\frac{\mathcal{O}}{(\frac{\mathcal{D}}{n} + b)} - \mathcal{T}_{IO}\right] \cdot (\mathcal{F}/\mathcal{S}) \tag{5.6}$$

Rate Constraint: A steady rate of progress in simulation is required for critical weather applications in order for scientists to provide advance information based on visualization. For weather applications, the rate of progress is represented by the ratio of the time simulated by the application (simulation time) and the wall-clock time taken by the application for the simulation (wall-clock time). For critical applications like cyclone tracking where advance information is needed by the scientists to provide timely guidance to decision makers, this ratio has to be greater than 1, i.e., the simulation time has to be greater than the wall-clock time taken for the simulation. The rate of simulation is dependent on the computation speed as well as on the output frequency. Higher the frequency of output, higher will be the number of I/O writes to the stable storage and hence lesser will be the simulation rate. Also, lower the I/O bandwidth, more significant will be the impact of high output frequency on the simulation rate. In most cases, the scientists may want to specify a minimum acceptable limit for this ratio. We denote this minimum ratio as MPR (Minimum Progress Rate). Equation (5.7) specifies the constraint related to the rate of simulations. Let ts denote the integration time step associated with the resolution of the simulation. This is the amount of time simulated or solved per time step and depends on the simulation resolution. If S frames are solved in an interval I and F frames are produced in that interval, then simulation time will be $ts \cdot S$ and wall-clock time will be the summation of solve time and time to output F frames i.e. $t \cdot S + \mathcal{T}_{IO} \cdot F$ as specified in equation (5.8). This can be rewritten as

equation (5.9).

$$\frac{Simulation\ time}{Wall{-}clock\ time} \geq MPR \tag{5.7}$$

$$\implies \frac{(ts \cdot S)}{(t \cdot S\ +\ \mathcal{T}_{IO} \cdot F)} \geq MPR \tag{5.8}$$

$$\implies \frac{ts}{(t\ +\ \mathcal{T}_{IO}\ \cdot\ F/S)} \geq MPR \tag{5.9}$$

Bounds: Depending on the maximum number of processors available and the scalability of the application, t has a lower bound $\mathcal{T}_{LB}$, as specified in equation (5.10). We specify an upper bound OI_{UB} for the output interval based on the minimum frequency of visualization of weather events desired by the users/climate scientists. For our experiments, we have specified the upper bound for output frequency to be 30 simulated minutes i.e. the interval between two time steps which are output by the simulation for visualization is maximum of 30 minutes. Output interval also has a lower bound OI_{LB} based on the limitations of the simulation application. For our weather application, the output interval has a lower bound of 1 simulated minute. The bounds for the output interval are specified in equation (5.11).

$$t \geq \mathcal{T}_{LB} \tag{5.10}$$

$$OI_{LB} \leq OI \leq OI_{UB} \tag{5.11}$$

Linearizing the Constraints: To linearize the non-linear constraints in equations (5.3), (5.6) and (5.9), we substitute $\frac{\mathcal{F}}{\mathcal{S}}$ by z and $\frac{\mathcal{T}}{\mathcal{S}}$ by y respectively to obtain the constraints (5.12), (5.13) and (5.14) for our optimization problem.

$$t\ +\ z \cdot \mathcal{T}_{IO} \leq \frac{\mathcal{O}}{b} \cdot y \tag{5.12}$$

$$t \geq \frac{\mathcal{O}}{(\frac{\mathcal{D}}{n}\ +\ b)} - \mathcal{T}_{IO} \cdot z \tag{5.13}$$

$$\frac{ts}{(t\ +\ T_{IO}\ \cdot\ z)} \geq MPR \tag{5.14}$$

OI depends on the ratio between the number of frames solved by the simulations and the

number of frames output to the disk. OI is a multiple of the integration time step ts as shown in Figure 5.3. A frame is solved after every ts simulated time and a frame is output

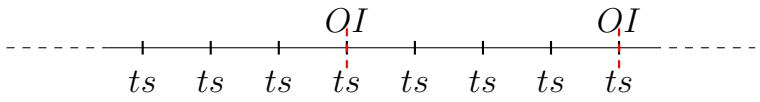


Figure 5.3: Output Interval and integration time step.

to disk after every OI simulated time. Thus the total time simulated in an interval of execution time, where $\mathcal{S}$ frames are solved and $\mathcal{F}$ frames are output to the disk, is given by equation (5.15). Using equation (5.15), the bound constraint of equation (5.11) can be rewritten as equation (5.16).

$$OI \cdot \mathcal{F} = ts \cdot \mathcal{S} \tag{5.15}$$

$$OI_{LB} \leq \tfrac{ts}{z} \leq OI_{UB} \tag{5.16}$$

We used GLPK (GNU Linear Programming Kit) [34] to solve the above linear programming problem and obtain the values for t, z and y. From the value of t, we determine the corresponding number of processors using the benchmark profiling runs with the weather simulations as explained in Section 5.2.4. We obtain the output interval, OI, by substituting for $z = \frac{\mathcal{F}}{\mathcal{S}}$ and ts in equation (5.15). This decision algorithm is invoked every 1.5 hours during the simulation run period. The optimization algorithm takes less than 0.3 seconds per run, thereby incurring negligible overheads. Moreover, the decision algorithm is invoked in a background process, thus not interfering with the run times of the simulations. Given the inputs $\mathcal{D}$, $\mathcal{T}_{IO}$, MPR, b, ts, and $\mathcal{O}$, this algorithm outputs the number of processors corresponding to t and OI to the application configuration file.

5.3 Experiments and Results

In this section we present our experimental setup including the details of the weather application used for simulation and remote visualization, the resource configurations used

for experiments involving networks of different bandwidths, and the results for automatic tuning by our adaptive framework.

5.3.1 Weather Application: Tracking Cyclone Aila

We have applied our framework for large-scale and long-range tracking of cyclones that involves a large amount of parallel computations with large data sets, subsequent data analysis and high-resolution visualization for scientific discovery. Visualization of cyclones is vital for subsequent data analysis and to help scientists comprehend the huge volume of data output. Visualization can be helpful in identifying low pressure regions or the appearance of high vorticity.

In our experiments, we used the framework for tracking a tropical cyclone, *Aila*, in the Indian region. Aila was the second tropical cyclone to form in the Northern Indian Ocean during 2009 [4]. The cyclone was formed on May 23, 2009 about 400 km south of Kolkata, India and dissipated on May 26, 2009 in the Darjeeling hills. There were 330 fatalities, 8,208 reported missing and about $40.7 million estimated damage. We simulated Aila upto a finest resolution of 3.33 km using WRF. To track the lowest pressure region or eye of the cyclone Aila, we employ a finer resolution nest on the region of our interest inside the parent domain as shown in Figure 5.4. We have performed the simulations for an area of approximately 32×10^6 sq. km. from 60°E - 120°E and 10°S - 40°N, comprising of the region of formation and dissipation of Aila over a period of about 3 days. The nesting ratio i.e. the ratio of the resolution of the nest to that of the parent domain, was set to 1:3. The 6-hourly 1-degree FNL analysis [31] GRIB (Gridded Binary) meteorological input data for our model domain was obtained from CISL Research Data Archive[98].

As WRF is a regional model, with each level of refinement, it needs input data at a finer resolution. Before executing WRF, the WRF Preprocessing System (WPS) is executed to interpolate the meteorological data onto the domain of interest. For tracking cyclones, our framework contains mechanisms for identifying the formation of cyclones in addition to the functionalities described in the earlier sections. The framework spawns a

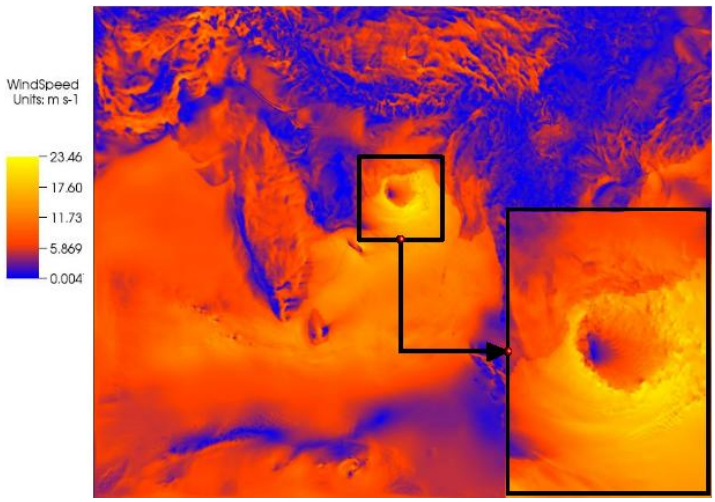


Figure 5.4: Windspeed visualization in finer resolution nest inside parent domain.

nest when the pressure drops below 995 hPa. We also use a configuration file that specifies the different resolutions for simulations for different pressure gradients or intensity of the cyclone. This can be specified by the climate scientists who typically use coarser resolutions for the initial stages of cyclone formation and finer resolutions when the cyclone intensifies. As and when the cyclone intensifies i.e. the pressure decreases, our framework changes the resolution of the nest multiple times to obtain a better simulation result from the model. Table 5.3 shows the pressure values used for different resolutions.

Table 5.3: Resolutions for different Pressure Values

Pressure (hPa)	995	994	992	990	988	986
Resolution (km)	24	21	18	15	12	10

The track of Aila as produced by simulation can be seen in Figure 5.5. It can be observed from the figure that the depression was formed in the central Bay of Bengal region (around 14°N) and traversed north-east upto Darjeeling (27°N).

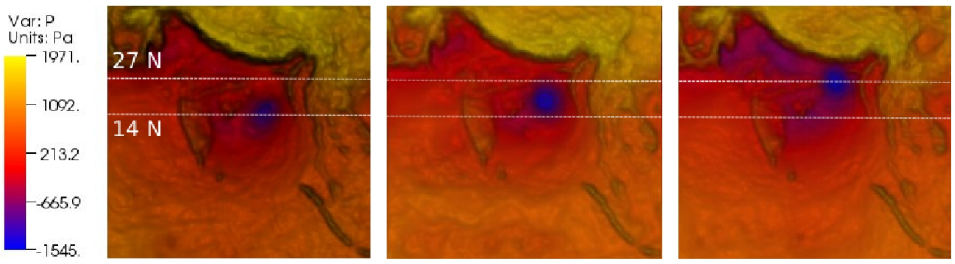


Figure 5.5: Visualization (volume rendering) of Perturbation Pressure at 18:00 hours on 23rd, 24th and 25th May, 2009.

5.3.2 Framework Implementation

The modifications to the WRF weather application for our work are minimal. We modified WRF to include monitoring of lowest pressure in the nest domain or in the parent domain when there is no nest. Our framework forms a nest dynamically based on the lowest pressure value in the domain and monitors the nest movement in the parent domain along the eye of the cyclone. The nest is centered at the location of lowest pressure in the parent domain. WRF is restarted whenever pressure drops below the threshold values specified by the user. When the application configuration file specifies the number of processors and output interval that are different from the current configuration, the simulation process is rescheduled on a different number of processors.

For faster I/O we used WRF's split NetCDF approach, where each processor outputs its own data. For example, simulation of 12 km resolution running on 288 processes results in output of 3 MB per process. The split approach is beneficial, especially for low bandwidth, low latency networks for faster data transfer from the simulation site. It also significantly reduces the I/O time per time step by up to 90% over the default approach of generating output to a single large NetCDF file. We have developed a utility to merge these split NetCDF files at the visualization site. Our plug-in for VisIt[17] enables directly reading the WRF NetCDF output files, eliminating the cost of post-processing before data analysis. We also customized VisIt to automatically render as and when these WRF NetCDF files are merged after arriving at the visualization site. The simulation output has been visualized using GPU-accelerated volume rendering, vector

plots employing oriented glyphs, pseudocolor and contour plots of the VisIt visualization tool.

The application manager periodically (in our work, every 1.5 hours) monitors the available disk space using the UNIX command *df*. The application manager also uses the average observed bandwidth between the simulation and visualization sites, obtained by using the time taken for sending about 1 GB message across the network. The decision algorithm is invoked every 1.5 hours. This frequency was sufficient for our experiment settings where the storage space and network bandwidth did not exhibit high fluctuations. For highly dynamic environments, the decision algorithm will have to be invoked more frequently. Although some of the threshold values used in our framework are specific to our experiment settings and WRF simulations, the underlying principles of our framework are generic and applicable to other applications.

5.3.3 Resource Configuration

For all our experiments, visualization was performed on a graphics workstation in Indian Institute of Science (IISc) with a Intel® Pentium® 4 CPU 3.40 GHz and an NVIDIA graphics card GeForce 7800 GTX. We used hardware acceleration feature of VisIt for faster visualization. We executed the simulations on four different sites resulting in three different remote visualization settings, namely, *inter-department*, *intra-country* and *cross-continent* visualizations. In the *inter-department* configuration, the WRF simulations were executed on a dual-core AMD Opteron 2218 cluster, *fire*, in Indian Institute of Science (IISc). In the *intra-country* configuration, the simulations were performed on quad-core Intel Xeon X5460 cluster, *gg-blr*, in Centre for Development of Advanced Computing (C-DAC) [1], Bangalore, India. The transfer between simulation and visualization site for this *intra-country* configuration was carried out on the National Knowledge Network (NKN)[85] with maximum bandwidth of 1 Gbps. In the *cross-continent* configurations, the simulations were performed on two clusters. One was a dual-core AMD Opteron 265 cluster, *moria*, in Innovative Computing Lab of University of Tennessee, Knoxville, USA. The other was the Abe [2] cluster in the National Centre for

Table 5.4: Simulation and Visualization Configurations

Configuration	*Simulation Configuration*	*Maximum Cores for Simulation*	*Maximum Disk Space Used*	*Average Sim-Vis Bandwidth*
inter-department	*fire*: 12x2 dual-core AMD Opteron 2218 based 2.64 GHz Sun Fire servers, CentOS release 4.3, each with 4 GB RAM, 250 GB Hard Drive, and connected by Gigabit Ethernet	48	182 GB	56 Mbps
intra-country	*gg-blr*: HP Intel Xeon quad-core processor X5460, 40 nodes, 320 3.16 GHz cores, RHEL 5.1 on Rocks 5.0 operating system, each with 16 GB RAM and 500 GB SATA based storage, and connected by Infiniband	90	150 GB	40 Mbps
cross-continent:moria	*moria*: dual AMD Opteron 265 (dual-core) 1.8 GHz cores, RHEL 5, each with 4GB RAM and 180 GB SATA Hard Drive, and connected by Gigabit Ethernet	56	100 GB	60 Kbps
cross-continent:abe	*abe*: Dell PowerEdge 1955 dual-socket quad-core compute blades, 1200 blades, 9600 2.33 GHz cores, RHEL 4 operating system, 1 GB RAM per core, 100 TB Lustre filesystem, and connected by Infiniband	128	700 GB	8 Mbps

Supercomputing Applications at the University of Illinois in USA. Table 5.4 gives detailed specifications of the three resource configurations including the maximum number of cores used for simulations, the maximum disk space used by our adaptive framework for the experiments, and the average available bandwidth between the simulation and visualization sites for each of the configurations.

To obtain the simulation rates for different number of processors, sample WRF runs were executed for different discrete number of processors and interpolated for other number of processors as mentioned in Section 5.2.4. These WRF profiling runs were executed on

1. 12, 16, 24, 30, 36, 42 and 48 processors in *fire* cluster.
2. 16, 24, 32, 48, 56, 64, 80 and 90 processors in *gg-blr* cluster.
3. 12, 16, 24, 30, 36, 42, 48 and 56 processors in *moria* cluster.
4. 32, 48, 64, 80, 96, 112 and 128 processors in *abe* cluster.

WRF simulations have limitations on the number of cores that can be used, depending on the grid size. Specifically, each MPI process should have at least 6×6 parent domain grid points and 9×9 nest domain grid points to process. For our simulations, we used a minimum nest grid size of 100×127 that is appropriate for the region of interest of cyclone Aila. This imposed limitation on the number of cores for simulations.

5.3.4 Results

The initial simulation resolution in all experiments was 24 km. We present results related to simulation throughput, visualization progress, disk consumption and adaptivity for our various experimental configurations in the following sections.

Inter-department Configuration

Figure 5.6 shows the simulation progress for the *inter-department* configuration. On the x-axis, the wall-clock time progression is shown when different frames are simulated

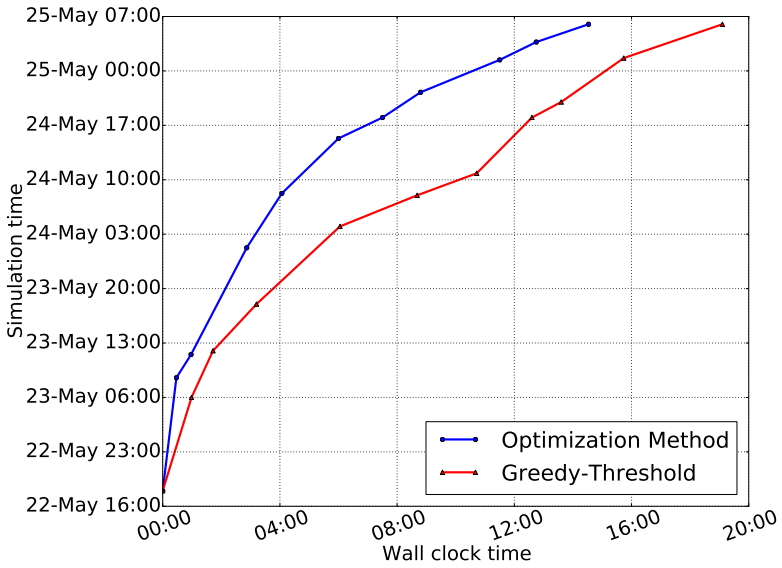


Figure 5.6: Simulation times with progress in executions for *inter-department* configuration. The graphs show faster rate of simulation for Optimization-based approach. Greedy-Threshold (red) and Optimization-based Approach (blue).

and on the y-axis, the corresponding simulation time steps are shown. The graph plots the times simulated with progress in executions. The optimization approach resulted in about five hours of less execution time than the greedy-threshold algorithm for the same total simulated time. The optimization approach maximizes simulation rate within resource constraints, and so it is able to provide a steady state solution for the simulation rate. The greedy-threshold algorithm, due to its reactive behavior, shows inconsistent rates of simulations throughout the simulation period.

Figure 5.7 shows the progress of visualization at the visualization site. On the x-axis, the wall-clock time progression is shown when different frames are visualized and on the y-axis, the corresponding simulation time steps represented by the frames are shown. Any given point in the graphs corresponds to the time when a simulated time step is visualized. For example, at time 15:00 hours corresponding to the x-axis, the frame corresponding to 23^{rd} May 03:00 hours simulation time step is visualized in the optimization-based approach. It can be seen in the figures that the visualization progress

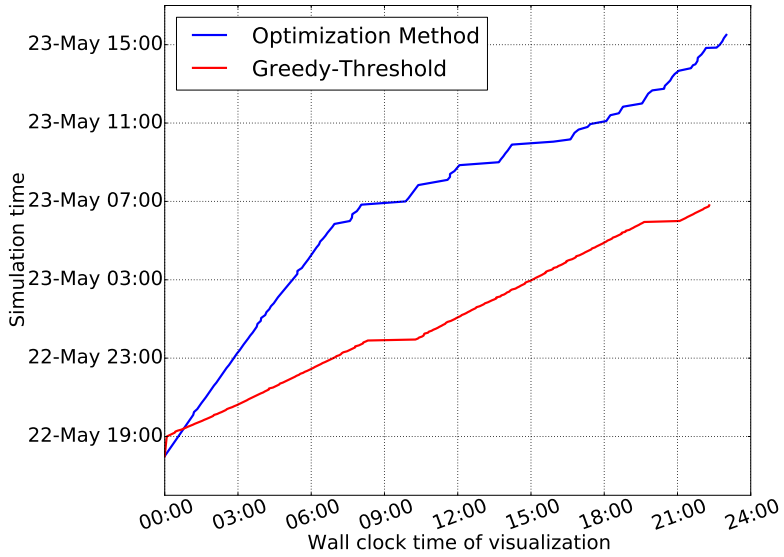


Figure 5.7: Progress of Greedy-Threshold (red) and Optimization-based Approach (blue) at the visualization end for *inter-department* configuration. Optimization approach shows faster visualization progress.

is much faster for the optimization method whereas the greedy heuristic approach lags behind in visualizing frames because it tries to send every time step from the simulation to the visualization site in the initial stages. Since the optimization method outputs frames depending on the resource constraints, reasonable visualization progress is made.

Figure 5.8 shows the rate of disk consumption for the *inter-department* configuration. The graph plots the percentage of available free disk space with progress in simulation. The greedy-threshold algorithm, due to its emphasis on minimizing the execution time and maximizing the frequency of disk output in initial stages, results in high rate of storage space consumption in the initial stages. The available free disk space becomes less than 50% within 4 hours of execution time. The algorithm then tries to take corrective action and consumes less storage by decreasing the simulation rate and the frequency of output to the disk. Nevertheless, the greedy algorithm consumes about 90% of the disk space by the end of the simulations. The optimization method, due to its steady state behavior, is able to determine the appropriate simulation rate and the frequency of

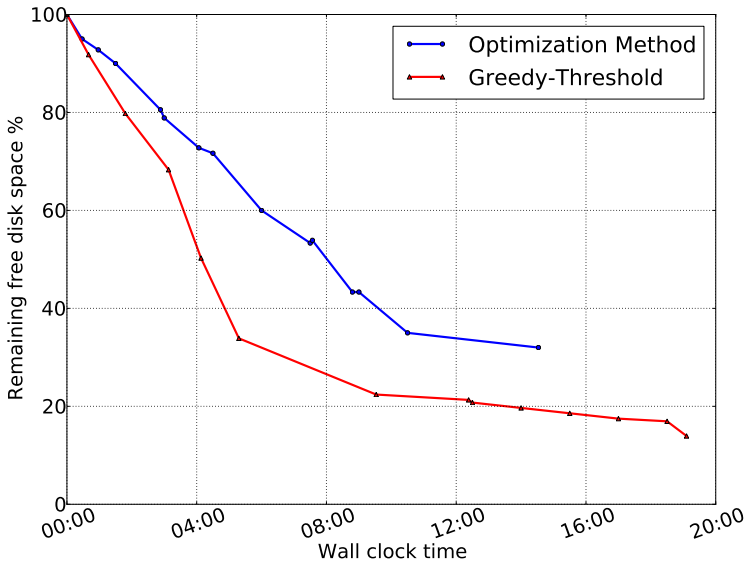


Figure 5.8: Free disk space with progress in executions for *inter-department* configuration. The graphs show the decrease in available disk space as simulation progresses in time. Greedy-Threshold (red) and Optimization-based Approach (blue).

output considering the different resource constraints and arrive at a global solution over a longer period of time. The efficiency of the optimization method results in about 25% less consumption of storage space than the greedy algorithm.

Figure 5.9 shows the adaptation in the number of processors and the output interval of the simulations. The wall-clock time progression for the simulation is plotted on x-axis, left y-axis plots the number of processors used during the simulation period and right y-axis plots the output interval chosen by the framework at various points of the simulation. The greedy method initially recommends maximum number of processors in order to have the highest simulation throughput. It also starts with a lowest output interval of 3 minutes in order to output as many time steps as possible. However with time, as the free disk space decreases, the output interval is increased and the number of processors is gradually decreased. The optimization method adjusts output frequency as and when needed according to the remaining disk space. It adapts the frequency

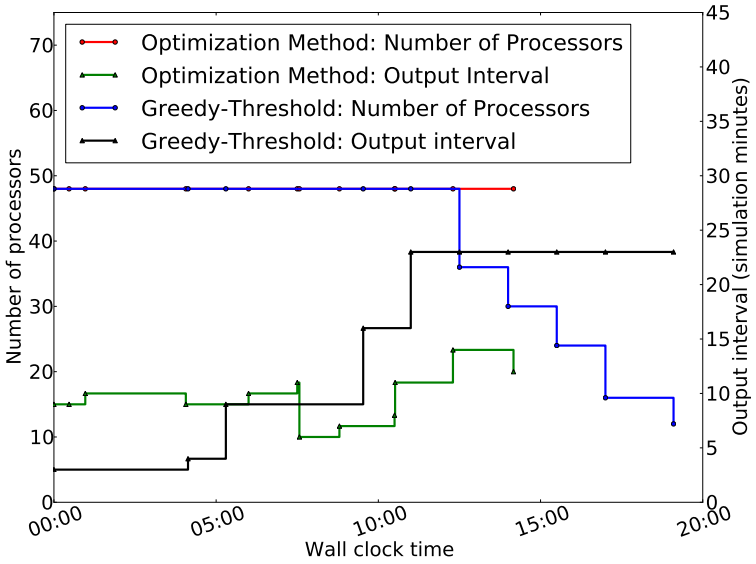


Figure 5.9: Adaptivity of the framework showing variation in number of processors (Left y-axis) and output interval (Right y-axis) for *inter-department* configuration.

of output to the best possible value for the given resource constraints from the beginning of the simulations. Since the primary objective is to minimize the solve time, the optimization method recommends maximum number of processors depending on the resource constraints and when it is able to satisfy the upper bound of output interval. The optimization method is able to sustain maximum simulation throughput throughout the simulation (red curve in the figure). However, though the greedy approach starts with best simulation rate (blue curve in the figure), it has to decrease the rate of simulation eventually due to increased disk consumption because of high initial output frequency.

Intra-country Configuration

Figure 5.10 shows the simulation progress for the *intra-country* configuration. On the x-axis, the wall-clock time progression is shown when different frames are simulated and on the y-axis, the corresponding simulation time steps are shown. We find that the optimization method provides steady and faster rate of simulations than the greedy-threshold algorithm. The optimization approach resulted in about seven hours of less

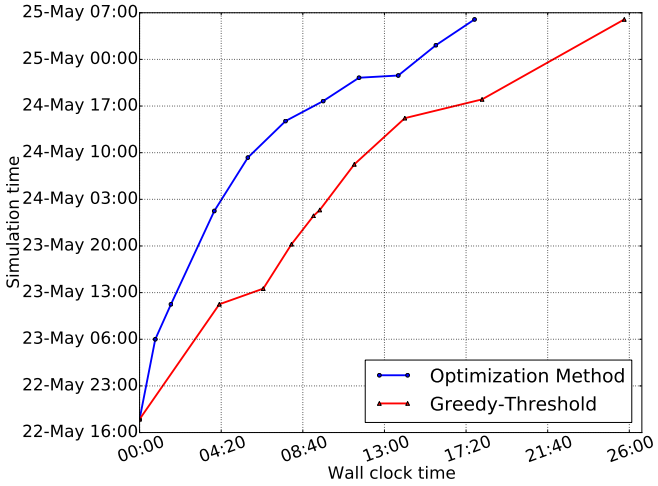


Figure 5.10: Simulation times with progress in executions for *intra-country* configuration. The graphs show faster rate of simulation for Optimization-based approach. Greedy-Threshold (red) and Optimization-based Approach (blue).

execution time than the greedy-threshold algorithm for the same total simulated time.

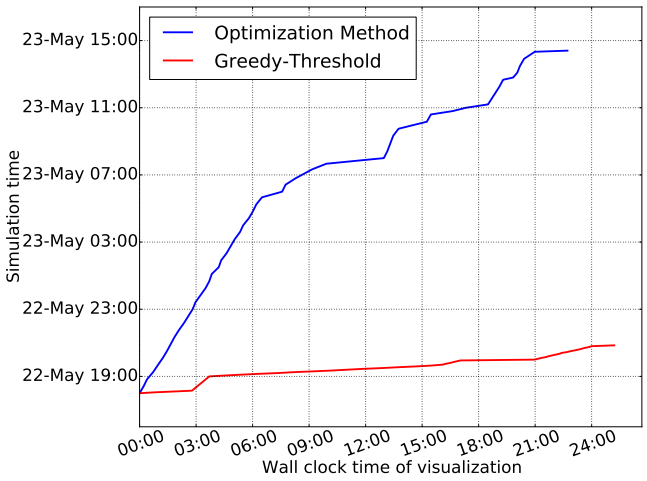


Figure 5.11: Progress of Greedy-Threshold (red) and Optimization-based Approach (blue) at the visualization end for *intra-country* configuration. Optimization approach shows faster visualization progress.

Figure 5.11 shows the progress of visualization at the visualization site for the *intra-country* configuration. On the x-axis, the wall-clock time progression is shown when

different frames are visualized and on the y-axis, the corresponding simulation time steps represented by the frames are shown. The visualization progress is faster for the optimization method whereas the greedy heuristic approach lags behind in visualizing frames because it tries to visualize every time step in the initial stages. The greedy heuristic is able to visualize less than 10 hours of simulation frames even after 21 hours of simulation. This is because of lower network bandwidth. The optimization method is able to visualize about a day of simulation progress in *intra-country* configuration. It can be observed in Figures 5.7 and 5.11 that the visualization progress in the *inter-department* and *intra-country* configurations is similar despite different network bandwidths. This is due to optimal selection of output frequency by the optimization method.

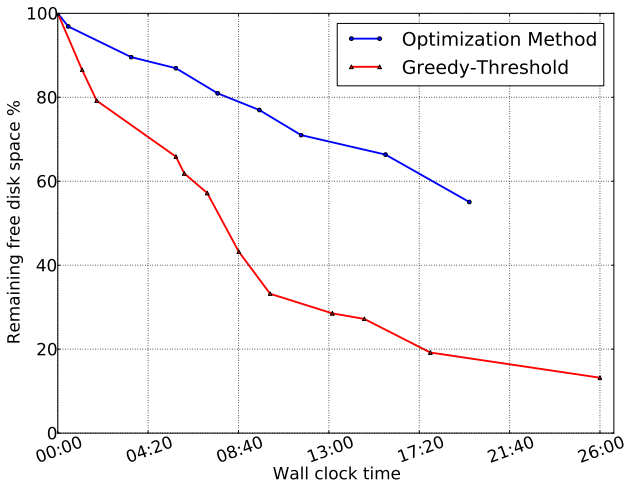


Figure 5.12: Free disk space with progress in executions for *intra-country* configuration. The graphs show the decrease in available disk space as simulation progresses in time. Greedy-Threshold (red) and Optimization-based Approach (blue).

Figure 5.12 shows the rate of disk consumption for the *intra-country* configuration. In the greedy-threshold heuristic, the disk space is quickly consumed and the free disk space falls below 20% because of faster solve time and slower network. In the optimization approach, the free disk space never drops below 50% during the entire run of the simulation because of the consideration of the global solution by this method.

Figure 5.13 shows the number of processors and output interval for the simulations

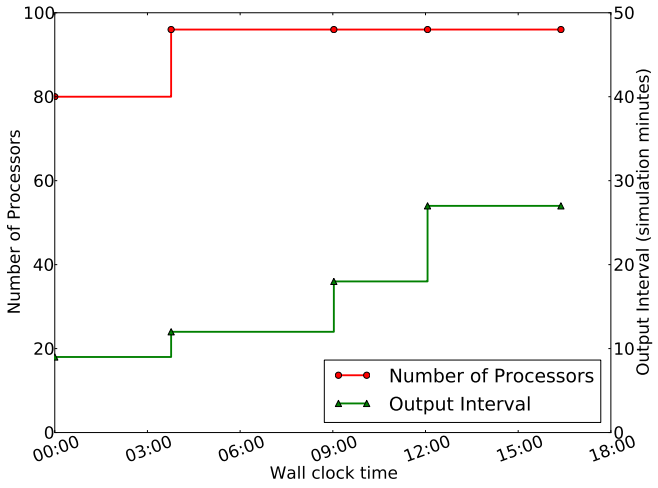


Figure 5.13: Adaptivity of the framework showing variation in the number of processors (red, left y-axis) and output interval (green, right y-axis) for *intra-country* configuration. Decision algorithm computes the optimal number of processors and output interval. MPR = 5.

automatically determined by our framework at different stages of execution for the *intra-country* configuration. During the initial stages, our framework chooses an initial value of 9 and 80 for output interval and the number of processors respectively. During the course of execution, the number of processors and output interval changes a few times. This happens when one or more parameters changes in the constraint equations as explained in Section 5.2.4. For example, when resolution changes from 18 km to 15 km, the time to solve a time step, the output data size per time step and the time to output a time step also increase. So, the decision algorithm re-evaluates the correct number of processors and the best output interval for the current parameters. As the resolution becomes finer, the output size and hence the time to output increases. Therefore, it can be observed that with refinement of simulation, the output interval is increased by the decision algorithm to satisfy the MPR and to prevent disk overflow due to frequent I/O. Thus, our framework adaptively changes the execution and application parameters based on the application and resource configurations and application simulation rates.

Our framework also provides guarantees regarding the rate of simulations (simulation time/wall-clock time). Specifically, the framework attempts to maintain higher

simulation rates than the minimum progress rate (MPR), using the rate constraint in equation 5.9 of the decision algorithm. The minimum progress rate (MPR) of simulations for the rate constraint of our algorithm was chosen as 5 for this setup. We found that the average rate of simulation maintained by our framework during the execution was 5.33. Thus our framework satisfies MPR.

Cross-continent Configuration: moria

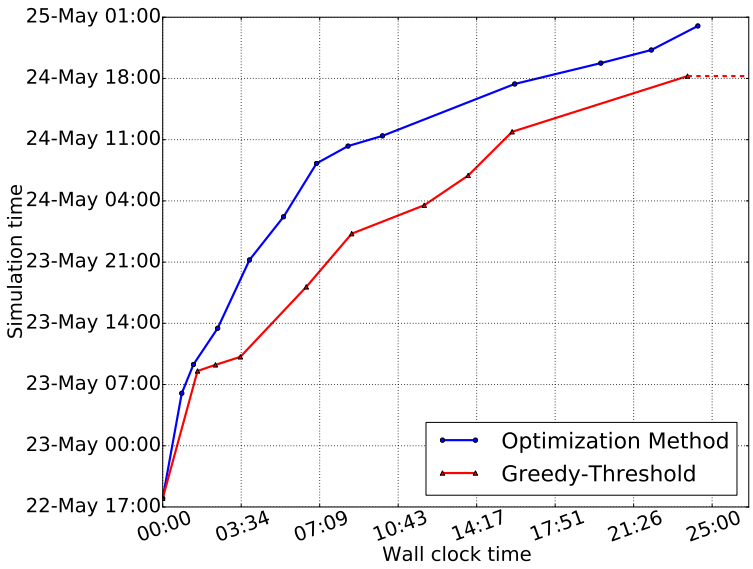


Figure 5.14: Simulation throughput for *cross-continent moria* configuration. Greedy-Threshold approach leads to stalling. Optimization approach is able to complete simulation without stalling. Greedy-Threshold (red) and Optimization-based Approach (blue).

Figure 5.14 shows the simulation progress for the *cross-continent* simulations on *moria* cluster. On the x-axis, the wall-clock time progression is shown when different frames are simulated and on the y-axis, the corresponding simulation time steps are shown. The available disk space gets filled at a faster rate since the bandwidth was only 60 Kbps. Hence in the greedy approach, WRF simulation had to be stalled even before the completion of the simulation time-period. This is shown by means of dotted lines in the graph after the wall clock time value of 24 hours. However, using the optimization

approach, WRF simulation was able to complete without stalling. This is because of selection of near-optimal parameters for execution based on the resource characteristics like network bandwidth and disk space as explained in Section 5.2.4. It is also clear from this result that a non-adaptive solution would result in stalling of the simulation much earlier than in the greedy-threshold algorithm described in Section 5.2.4.

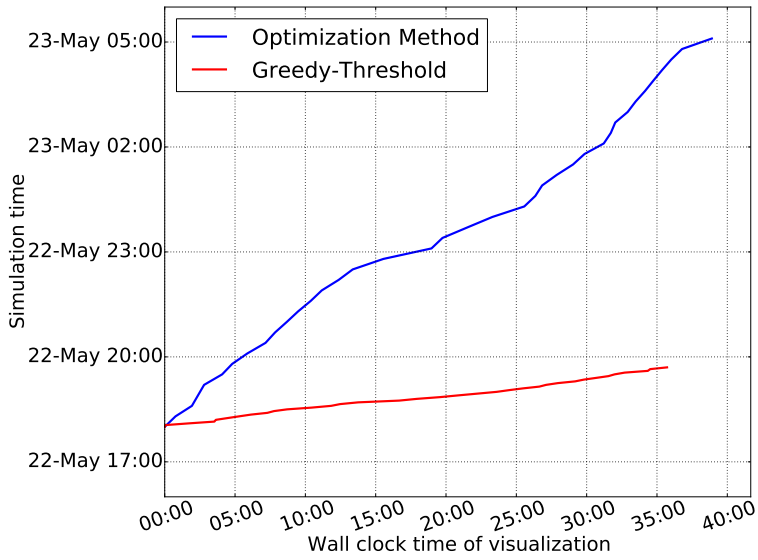


Figure 5.15: Progress of Greedy-Threshold (red) and Optimization-based Approach (blue) at the visualization end for *cross-continent moria* configuration. Optimization approach shows faster visualization progress.

Figure 5.15 shows the progress of visualization. On the x-axis, the wall-clock time progression is shown when different frames are visualized and on the y-axis, the corresponding simulation time steps are shown. The visualization progress is much faster for the optimization method whereas the greedy heuristic approach lags behind in visualizing frames because of higher output frequency initially. The greedy heuristic is able to visualize less than 3 hours of simulation frames even after 30 hours of simulation. This is because the network bandwidth for this configuration is low and hence transferring data corresponding to every time step slows down the visualization progress. The frames arrive at the visualization site at a slower rate because of low network speed.

However, the optimization method is able to provide higher visualization throughput than the greedy-threshold method because the optimization algorithm recommends a higher output interval.

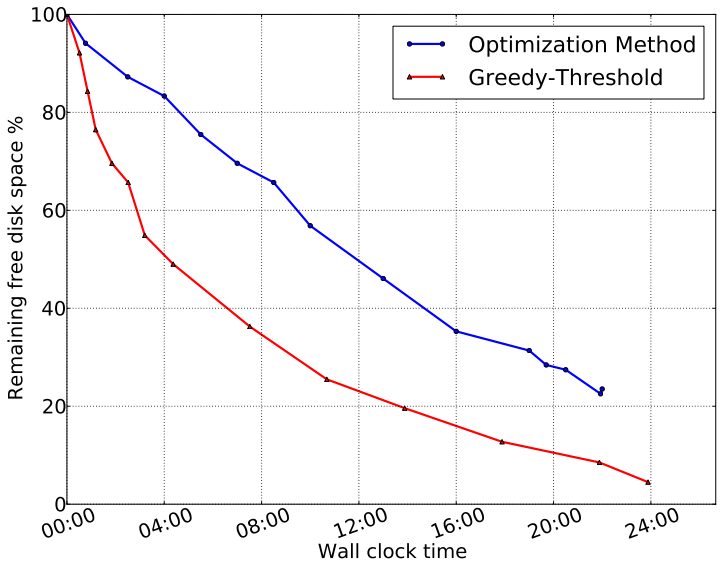


Figure 5.16: Free disk space with progress in executions for *cross-continent moria* configuration. The graphs show the decrease in available disk space as simulation progresses in time. Greedy-Threshold (red) and Optimization-based Approach (blue).

Figure 5.16 shows the rate of disk consumption. The storage space decreases rapidly for the greedy-threshold heuristic because it tries to maximize the number of frames output without considering available disk space in the initial stages. Since the network bandwidth is very low, this approach suffers from disk overflow (less than 5%) before completion of simulation. The optimization method is able to complete simulation with more than 20% of the disk space remaining. This is because though the disk is filled faster due to slower network, the optimization method tries to adjust the output frequency and number of processors from the beginning of the simulation.

Figure 5.17 shows the adaptation in the number of processors and the output interval of the simulations by the framework based on the application and resource configurations for the *cross-continent* configuration on moria cluster. The wall-clock time progression

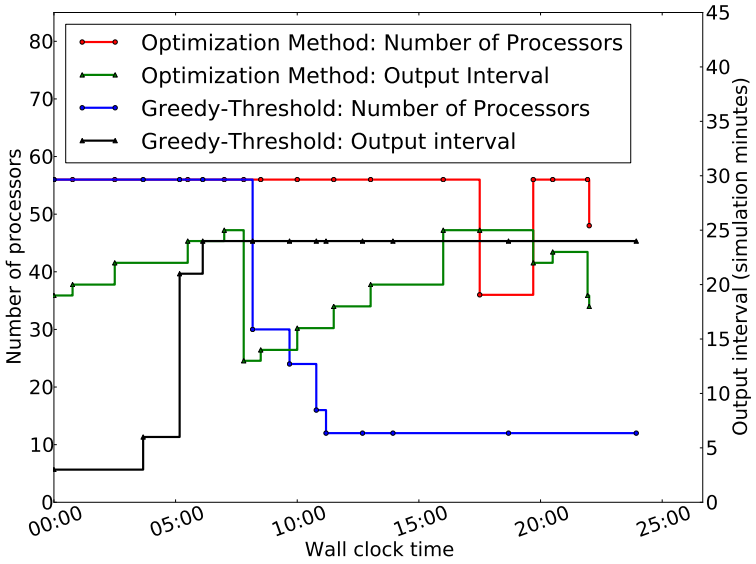


Figure 5.17: Adaptivity of the framework showing variation in number of processors (Left y-axis) and output interval (Right y-axis) for *cross-continent moria* configuration.

for the simulation is plotted on x-axis, left y-axis plots the number of processors used during the simulation period and right y-axis plots the output interval chosen by the framework as the simulation progresses. When the greedy approach cannot further increase output interval or decrease number of processors and the disk space is sufficiently low, the WRF simulations have to stall. Whereas in the optimization approach, it tries to avoid stalling from the beginning by considering the various impact factors for smooth simulation and visualization. It can be observed that the optimization method is able to provide a higher simulation rate during the entire run though the network speed for this configuration is very low. In contrast, the simulation throughput for the greedy method drops after few hours of the run. We also find that the output interval has lesser variation in the optimization method than the greedy-threshold heuristic. Thus, the optimization method is able to provide a consistent "quality-of-service" for visualization by the climate scientists, which is very essential for remote visualization of critical weather applications.

Cross-continent Configuration: Abe

In this section, we show some results of automatic tuning of execution parameters by our framework for the *cross-continent* experiment on *Abe* cluster of NCSA. We performed simulations over a larger domain, approximately from 30°E - 150°E and 10°S - 40°N, and for a period of 3 days 18 hours. The minimum progress rate (MPR) of simulations for the rate constraint of our algorithm was chosen as 3 for this setup. Steady progress in simulation and visualization without stalling is observed.

Figure 5.18 shows the simulation throughput. We observe that the simulation for 3 days and 18 hours was completed in about a day because of the high simulation rate. The simulation curve shows a rapid change in slope for the initial few hours when the cyclone is being formed and the simulation resolution is being refined. After it reaches

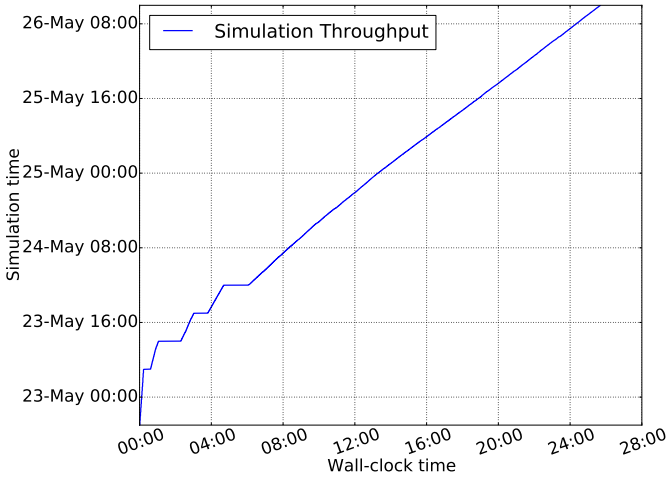


Figure 5.18: Simulation throughput for *cross-continent Abe* configuration.

the finest resolution, it continues at a steady rate and there is not much change in slope. At a coarser resolution, the time steps are solved at a much faster rate because the number of grid points are lesser than in the finer resolution. Hence the slope is steeper in the beginning of the simulation.

Figure 5.19 shows that we are able to continuously visualize despite the slow network bandwidth. It can be seen that there is a change in slope of the curve at a point

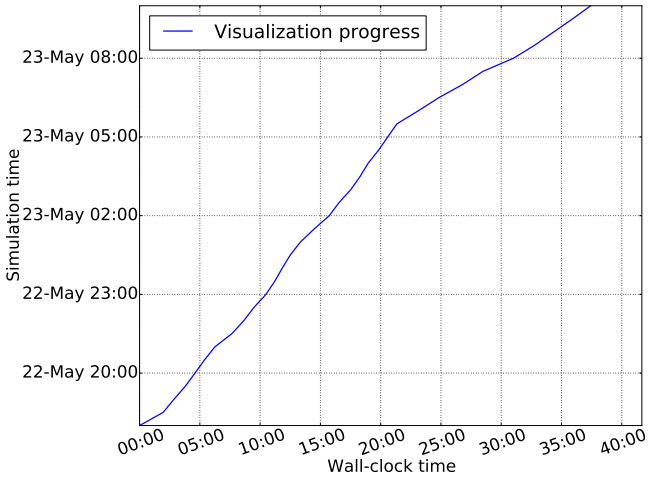


Figure 5.19: Visualization progress for *cross-continent Abe* configuration.

corresponding to 23^{rd} May, 05:00 hours simulation time step. This is because a nest is formed at this time step, and hence the total amount of output data for a single time step increases and hence the time to transfer one time step increases.

Figure 5.20 shows that the simulation completed without overflowing available disk space. This is because of adjusting the number of processors to the correct value so that availability of disk space is not a problem even though the simulation rate is high and the network bandwidth is low.

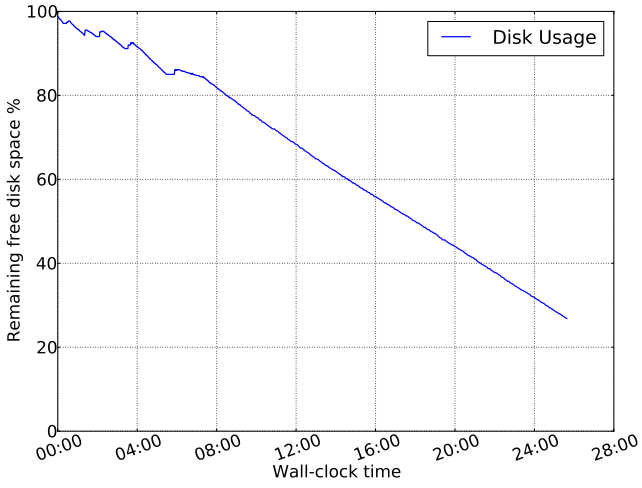


Figure 5.20: Disk Consumption for *cross-continent Abe* configuration.

We find a considerable time lag between the time when a time step was simulated and when the corresponding frame was visualized. This is because of the high simulation rate in Abe cluster, and the slow Internet-based transfer of frames to the visualization site in India. We address this problem of lag between simulation and visualization times in Chapter 7.

The time taken for simulating a time step in the coarsest resolution (24 km) in Abe is approximately 0.6 seconds. Our framework chose the initial number of processors for WRF executions in Abe as 96, and the output interval for the simulations as 30 as can be seen in Figure 5.21. The maximum number of processors, i.e 128 processors were not chosen because of the disk space constraint due to the slow network. If simulation had started on 128 processors, it would have progressed at a much faster rate leading to storage problem later. At the same time, it was also not drastically reduced so that a minimum progress rate of simulation is maintained without overflowing the disk. The output interval was chosen to be 30 minutes because more output frames would mean a higher disk space consumption rate due to slow data transfer from the disk.

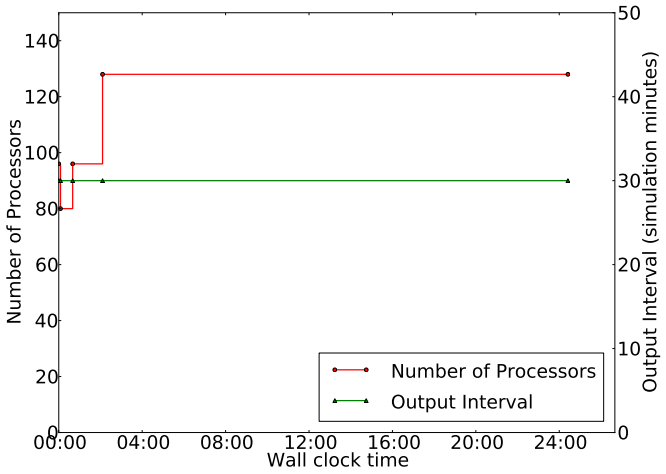


Figure 5.21: Change in number of processors and output interval for *cross-continent Abe* configuration.

It can be seen that the change in the number of processors during the course of execution is not frequent. This is because the decision algorithm selects an optimal value

considering parameters like simulation resolution, I/O bandwidth, network bandwidth, time to solve and free disk space available. When the simulation runs at a given resolution, only the disk space varies. However since the total available disk space is already considered by the algorithm while selecting the optimal value, there is not much variation later during the run. When the resolution changes, some of the parameters like output per time step and time to solve a time step change and hence we can see some variation in the number of processors. At a finer resolution, the time to solve a time step increases. Since the simulation starts with a coarser resolution and then changes to a finer resolution at different points of the experiment, the number of processors is more for the finer resolutions.

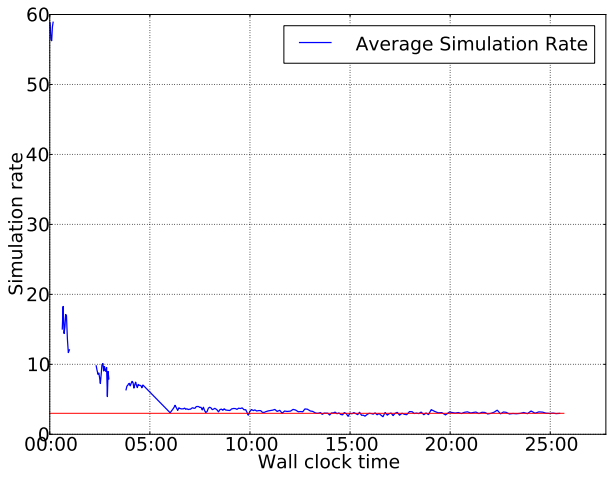


Figure 5.22: Actual rates of simulations for *cross-continent Abe* configuration. The rate constraint of our decision algorithm ensures higher simulation values than the MPR.

Figure 5.22 shows the actual rates of simulations for different stages of WRF execution for the experiment with the *cross-continent Abe* configuration. We set an MPR value of 3 for this experiment and we find that the rates are always maintained higher than the MPR value, thus ensuring minimum guarantees in the speed of simulations, which is very essential for continuous tracking of critical weather events. Even though the simulation is slowed down in the beginning of the experiment when the simulation rate is quite high due to coarser resolution, yet the framework ensures that the MPR is satisfied.

We observe that the simulation rates reduces with time after the formation of nest and further refinement. The nest domain has a smaller time step and is integrated more number of times than the parent, and hence the overall solve time increases. Resolution is continuously refined during the execution for better simulation accuracy. Finer resolution simulations require more number of computations, and hence the decision algorithm increases the number of processors for higher resolutions to satisfy the MPR.

5.4 Discussion

Simulation data continues to grow in size in many applications like cosmology [3] and earthquake simulations [129]. In these applications, online visualization approach will be effective in quick understanding of the data. Our adaptive framework for online visualization is readily applicable to other applications executed on any resource configuration for continuous simultaneous simulation and visualization using optimal parameters.

The framework effectively responds to application and resource dynamics. Dynamic nest formation by the framework affects the simulation rate and output, which results in change in simulation parameters by the framework. Resource usage like shared disk space and network bandwidth may also affect the simulation progress rate and data transfer rate. This is efficiently handled by our framework by periodic invocation of the decision algorithm.

5.5 Summary

In this chapter, we have described an adaptive integrated framework for simultaneous simulation and on-the-fly remote visualization of critical weather applications like cyclones. We show how our framework achieves high simulation rates for loosely coupled simulation and visualization, and adapts simulation parameters according to the resource parameters. The framework recommends the number of processors for simulation and the output interval based on the disk space, I/O bandwidth and network speed constraints. In the process, the framework also considers the dynamics of the changing

resource configurations. As shown in Section 5.3, a simple and intuitive greedy approach may lead to low throughput, stalling of simulation and disk overflow. This shows the importance of considering network bandwidth, execution time and output frequency for a smooth online visualization. Our optimization method is able to provide about 30% higher simulation rate completing the entire simulations for all network configurations, consumes about 25-50% lesser storage space completely avoiding the disk overflow problem and the resulting stalling of simulations, and provides higher and more consistent rate of visualization than the greedy approach. Hence we claim that it is very important to consider the currently available network bandwidth, and disk space to be able to do online visualization with best throughput and best temporal resolution.

The experiments discussed in this chapter involve a single region of interest. However, it is conceptually straight-forward to extend to multiple regions of interest. Specifically, some of the WRF simulation problem parameters in Table 5.2 will be calculated based on the multiple regions of interest instead of a single region of interest. The size of simulation output of one time step will increase depending on the number and dimensions of nests. The time to solve one simulation time step should be estimated from the dimensions and resolutions of the parent and the nested simulations. However, the linear program formulation remains the same.

In weather simulations and online visualization, the climate scientist or the user may like to alter some of the simulation parameters like the region of interest and resolution. We have extended our framework to add steering capabilities for interactive simulation-visualization in order to incorporate the user inputs in the simulation. We describe in the next chapter reconciling user-driven steering with the algorithmic steering described in this chapter.

Chapter 6

Integrated Algorithmic and User-driven Steering

6.1 Introduction

Large-scale simulations for critical weather applications like cyclone tracking and earthquake modeling require simultaneous online/"on-the-fly" visualization simultaneously performed with the simulations. Online visualization enables the scientists to provide real-time feedback in order to steer the simulations for better and more appropriate output suited to the scientific needs. Remote visualization can enable geographically distributed climate scientists to share vital information, perform collaborative analysis, and provide joint guidance on critical weather events. Remote visualization and feedback control using computational steering can thus assist a large climate science community in analyzing large-scale scientific simulations.

In the previous chapter, we described an adaptive framework that automatically tunes various parameters including the frequency of visualization output and number of processors for simulations to enable simultaneous simulations and continuous online remote visualization of critical weather applications in environments with storage and network constraints. In this chapter, we consider computational steering and feedback control of critical weather applications by remote scientists in addition to the automatic tuning of execution parameters.

6.1.1 Motivation

Computational steering is a well-studied approach that allows the user to interactively explore a simulation in time or space by giving feedback to the simulation, based on visualization output. By allowing user input to instantaneously impact the simulation, interactive steering "closes the loop" between simulation and visualization [93, 95]. Unlike existing efforts on computational steering, a steering framework for controlling the high performance simulations of critical weather events needs to take into account both the steering inputs of the scientists and the criticality needs of the application. We use the minimum progress rate (MPR) of the simulations as a parameter to represent the criticality of the application. This is a parameter input by the climate scientist to express the desired quality-of-service of the weather simulations. It denotes the minimum number of climate days that has to be simulated and output in a given wall-clock time by the application. A steering framework for critical weather applications, while allowing the scientist or user to remotely steer various application parameters including the resolution of simulations and the frequency of output for visualization, should also analyze the impact of the user-specified parameters and given resource constraints on the MPR, guide the user on possible alternative options, and possibly override the user-specified values with automatically determined values. For example, the steering framework should override a very high output frequency specified by the user if it determines that the high value can lead to unavailability of storage and severely compromise the MPR or criticality needs of the application.

6.1.2 Problem Statement

In this chapter, we describe an integrated user-driven and automated steering framework, InSt (**In**tegrated **St**eering), for simulations, online remote visualization, and analysis for critical weather applications. InSt allows steering of application parameters including resolution of simulation, rate of simulation and frequency of data for visualization. Further, it allows scientists to specify region of interest and perform finer resolution simulation for the region of interest. However InSt also considers the criticality of the

application, namely, the minimum progress rate needed for the application, and various resource constraints including storage space, network bandwidth, and the number of processors, to decide on the final parameter values for simulation and visualization. Thus the integrated framework tries to find the best possible parameter values based on both user input and resource constraints. In the previous chapter, we described *algorithmic steering* using an optimization-based algorithm to automatically tune application parameters. In this chapter, we describe our framework, InSt, that effectively combines computational steering by the user/scientist with the algorithmic steering performed by the runtime system of the framework. Thus InSt is unique since it considers the reconciliation of both user-driven computational steering and algorithmic steering, unlike existing work that considers only user-driven computational steering [83, 93, 134, 139]. Using our framework, we performed cross-continent steering with the steering and visualization performed in India and simulations conducted on a TeraGrid [137] cluster in NCSA, USA. We show that InSt performs reconciliation between algorithmic and user-driven computational steering for a 3-day weather simulation while providing quality of service with respect to simulation rate and continuous visualization.

6.1.3 Chapter Outline

Section 6.2 presents our integrated steering framework including the components and interactions. Section 6.3 explains the reconciliation between the algorithmic steering and user-driven steering. Section 6.4 presents our experiments involving different network bandwidths and results including simulation rates and we summarize in Section 6.5.

6.2 InSt Steering Framework

Simultaneous and continuous visualization for user-guided simulation of critical weather applications require robust middleware for better application performance and efficient resource management. We have developed an integrated steering framework that performs automatic tuning as well as user-driven steering. Our framework, InSt, shown

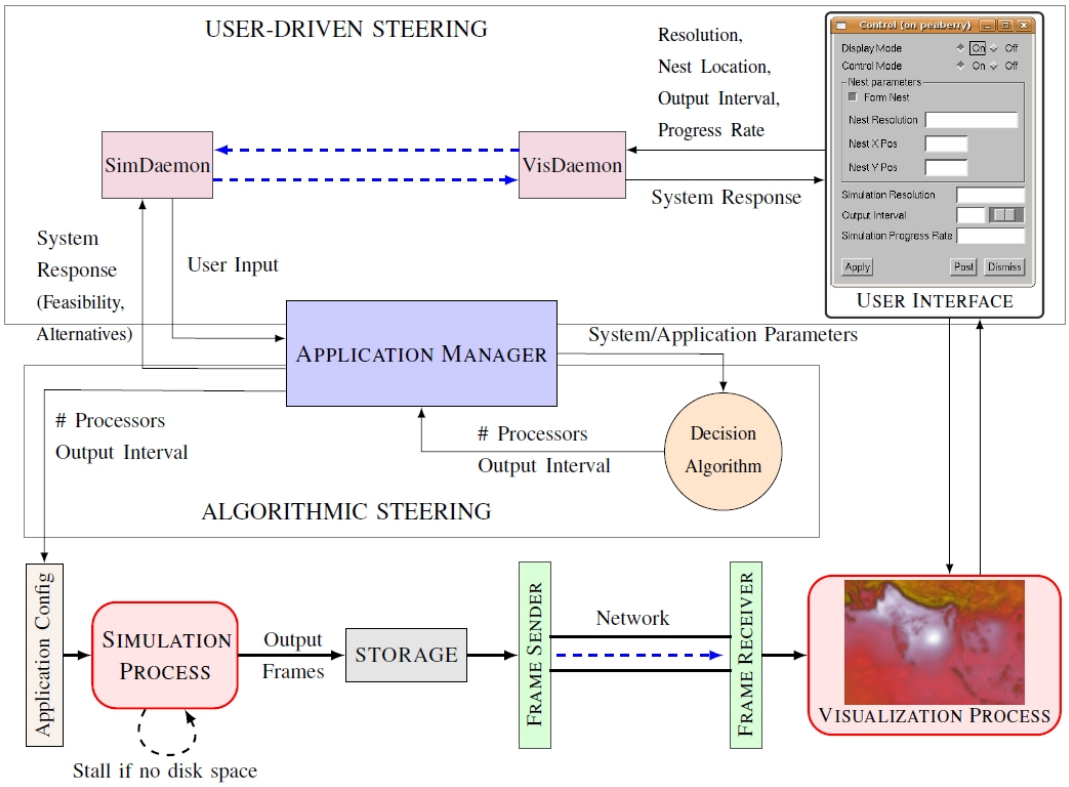


Figure 6.1: INST: Integrated Steering Framework

in Figure 6.1, consists of the following components to perform coordinated simulations, online remote visualizations, and user-driven steering: *an application manager* that determines the application configuration for weather simulations based on resource characteristics and user input, *a simulation process* that performs weather simulations with different application configurations, *a visualization process* for visualization of the frames, *frame sender and receiver daemons* that deal with transfer of frames from simulation to visualization sites, *simdaemon and visdaemon* for communication of user-specified simulation parameters and system response and *user interface* for accepting user input. In our work, we remove the frames from the simulation site once they are transferred to the visualization site. The following subsections describe in detail the primary components for user-driven steering.

6.2.1 User Interface, SimDaemon and VisDaemon

The user gives input through the user interface as shown in Figure 6.1. In particular, user can specify nest location, simulation resolution, bounds for output interval and simulation progress rate through the user interface. The input values from the user are sent to the application manager through the *VisDaemon* and *SimDaemon*. The VisDaemon receives user input from the user interface through the visualization process, and communicates the same to the SimDaemon. The SimDaemon specifies this user input to the application manager. The response from the manager is conveyed back by the daemons to the user through the user interface.

6.2.2 Application Manager

The application manager is the primary component of InSt and acts as the bridge between automatic steering and user-driven steering in our framework. The application manager periodically monitors the resource parameters, namely the free disk space and available network bandwidth. For automatic/algorithmic steering, the application manager periodically invokes the decision algorithm, explained in Section 5.2.4 for obtaining the number of processors for simulations and the frequency of weather data output to be generated by the simulations for continuous visualization, given resource parameters and resolution of simulation. For user-driven steering, the application manager asynchronously receives the user inputs, including the upper bound for frequency of output and simulation resolution, from the visualization site. The manager checks the feasibility of running the simulations with the user inputs, advises the users of alternate options if not feasible, and invokes the decision algorithm with the user inputs and resource parameters if feasible.

The manager writes the simulation parameters output by the decision algorithm to the application configuration file, and starts or stops-and-restarts the simulations with the parameters. More details on the reconciliation of the automatic/algorithmic and user-driven steering are given in Section 6.3.

6.3 Reconciling User-driven and Algorithmic Steering

Initially, before starting the simulations, the application manager specifies default values for simulation resolution and MPR. The application manager then determines the frequency of output using the decision algorithm based on resource constraints for the given resolution of simulation. The simulations are then started with these values. The user at the visualization site can change these simulation parameters during execution through the user interface. The interface also allows the user to specify a location for formation of nest or sub-region in the domain for finer resolutions. When the user requests nest placement or a finer simulation resolution, the simulation process restarts and continues with a new configuration involving the nest and the new resolution.

When a user specifies an output interval (inverse of frequency) and/or MPR value, the InSt framework considers the criticality of the application, proactively checks the feasibility of executing the simulations with these inputs, and guides the user with possible alternative values for ensuring continuous and reasonable progress of simulations and visualization. When a user specifies an output interval and does not specify MPR, the framework checks if the specified interval can be used without violating the rate constraint of equation (5.9), i.e., if the simulations can generate output with the specified interval such that the simulation rate will continue to be greater than the current MPR used by the application manager. This relationship between output interval, OI, and MPR is given by equation (6.1) which is derived using equation (5.9) and the relation $\mathcal{F}/\mathcal{S} = ts/OI$ described in equation (5.15).

$$MPR \leq \frac{ts}{[t + T_{IO} \cdot (ts/OI)]} \tag{6.1}$$

It can be clearly seen that there is a direct relation between OI and MPR. For example, let the resolution be 15 km, the integration time step, ts, be 45 seconds, T_{IO} be 43 seconds and t be 3.95 seconds. Now if the user requests OI of 180 seconds (i.e. solved time steps are output onto the storage every 3 simulated minutes), then from equation (6.1), MPR will be less than or equal to 3.06. But if the MPR currently used by the application

manager is 5, then the system clearly cannot satisfy his request for an OI of 180 seconds. Therefore in such cases, InSt has to override the user requested value for OI.

Also, it can be seen from equation (6.1) that if the OI is too low, it will decrease the simulation rate as well. Hence to continuously simulate and visualize at a steady rate, InSt determines the lower bound of OI from equation (6.1) using the current value of MPR used by the application manager. If the value of OI requested by the user is less than this lower bound, then InSt does not incorporate the user-specified OI. In this case, the InSt framework informs the user of the lowest feasible OI. If the user-specified value of OI is greater than the lower bound, this value forms the upper bound for OI in the decision algorithm, i.e. the decision algorithm in InSt tries to find the best possible OI within the user-specified bound.

If the user specifies a MPR value, then the application manager attempts to change the current MPR value it uses in the decision algorithm with the user-specified value. However, it first checks if the user-specified MPR, $uMPR$ is feasible for the current resolution of simulations by calculating an upper bound, MPR_{max}, feasible for the resolution and comparing the user-specified $uMPR$ with MPR_{max}. For calculating MPR_{max}, the feasible upper bound, we substitute OI with ∞ and t with its lower bound T_{LB} in equation (6.1) and obtain MPR_{max} as the ratio of integration time step ts and T_{LB}. If the user specified MPR, $uMPR$, is greater than this feasible upper bound, MPR_{max}, for the current resolution, InSt proactively tries to find a coarser resolution and checks the feasibility of $uMPR$ for the coarser resolution by calculating MPR_{max} for the coarser resolution. MPR_{max} for the coarser resolution will be higher than the MPR_{max} for the finer resolution because of greater ts. The application manager then provides the user with the options of coarser resolution at which the $uMPR$ is feasible. Thus, InSt proactively tries to find a balance between the algorithmic steering of the decision algorithm and the user-driven steering values considering the criticality, represented by MPR, of the application.

The flowchart in Figure 6.2 depicts the reconciliation or handshaking dialogue between the algorithmic steering and user-driven steering. In the flowchart, MPR is the

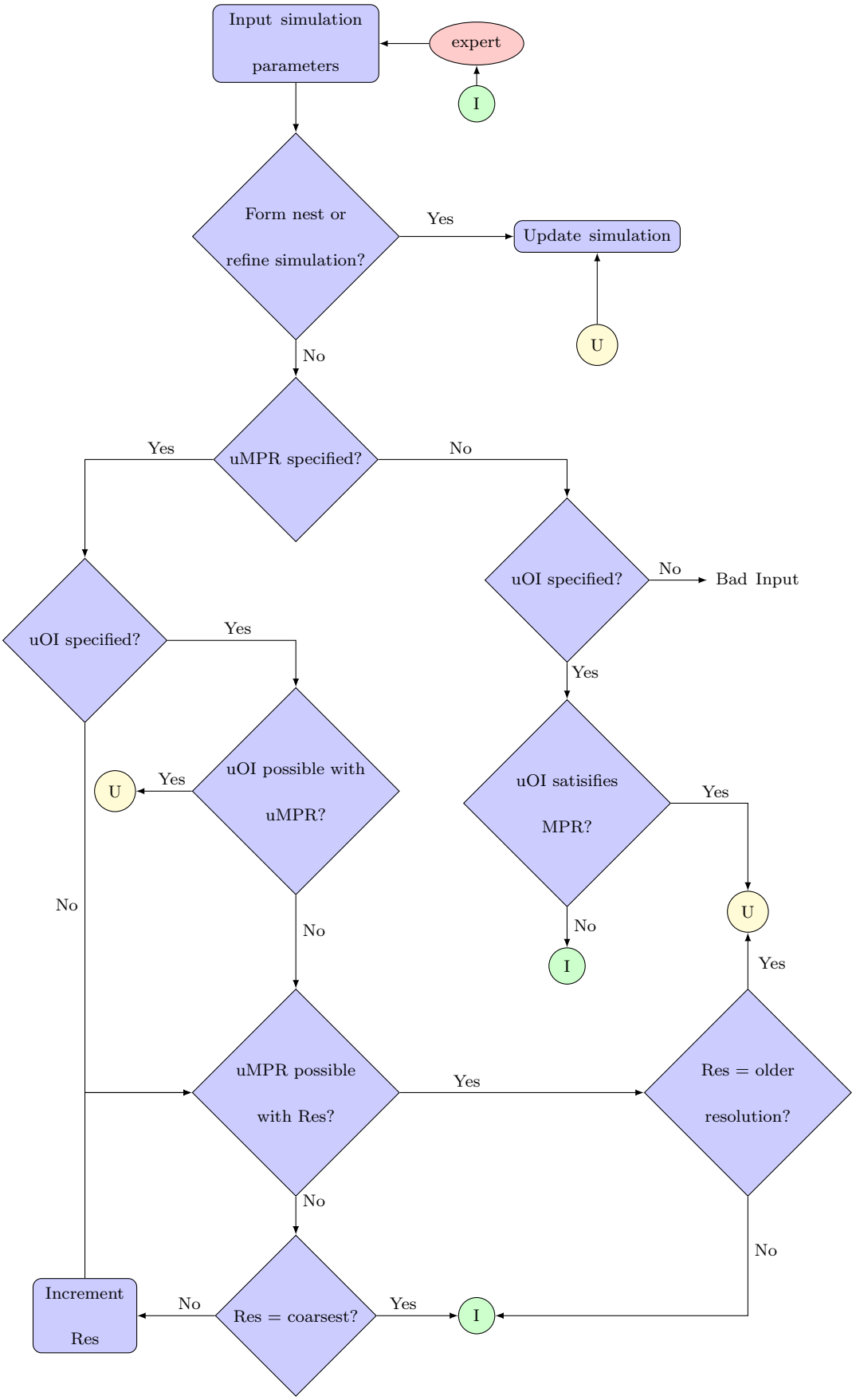


Figure 6.2: Flowchart depicting reconciliation

value currently used by the application manager for simulations, *Res* represents the current resolution of simulation, $uMPR$ and uOI are the user-provided MPR and OI (output interval) values. The flowchart shows the various feasibility analysis done by the application manager before accepting or overriding user request. If the user-specified values are approved by the application manager, the simulation process is updated with the new values. This is denoted in the flowchart using connectors marked 'U'. When the application manager is not able to satisfy user-request due to infeasibility with respect to the current minimum progress rate used in the simulations, then the user is recommended an optimal set of values. This is denoted by connectors marked 'I' in the flowchart.

In summary, InSt updates the simulation parameters based on user input, i.e. there is a path in the flowchart from the start to 'U' in the following cases when the user specifies:

- nest location.
- $uMPR$ and uOI, and both values can be used by InSt either at the current resolution or a lower resolution.
- uOI, and it is feasible with the current MPR used in the execution.

InSt reports infeasible user inputs, i.e. there is a path in the flowchart from the start to 'I' in the following cases when the user specifies:

- $uMPR$ and uOI, and both values cannot be used even at the coarsest resolution.
- uOI, and it is not possible to use uOI with the current MPR used in the execution.
- null input values for all input fields.

Therefore the user drives the simulation process and the automatic tuning framework steers the weather simulation to the favorable state of continuous visualization with minimum stalling and maximum progress rate.

6.4 Experiments and Results

6.4.1 Resource Configuration

For all our experiments, visualization was performed on a graphics workstation in Indian Institute of Science (IISc) with a dual quad-core Intel® Xeon® E5405 and an NVIDIA graphics card GeForce 8800 GTX. We used hardware acceleration feature of VisIt for faster visualization. We executed the simulations on two different sites resulting

Table 6.1: Simulation and Visualization Configurations

Configuration	*Simulation Configuration*	*Maximum Cores for Simulation*	*Maximum Disk Space*	*Average Sim-Vis Bandwidth*
intra-country	*gg-blr*: HP Intel Xeon Quad Core Processor X5460, 40 nodes, 320 3.16 GHz cores, RHEL 5.1 on Rocks 5.0 operating system, each with 16 GB RAM and 500 GB SATA based storage, and connected by Infiniband primary interconnect and Gigabit Ethernet secondary interconnect. For our work, we used the Gigabit network	96	150 GB	40 Mbps
inter-country	*abe*: Dell PowerEdge 1955 dual-socket quad-core compute blades, 1200 blades, 9600 2.33 GHz cores, RHEL 4 operating system, 1 GB RAM per core, 100 TB Lustre filesystem, and connected by Infiniband	128	700 GB	8 Mbps

in two different remote visualization and computational steering settings, namely, *intra-country* and *inter-country* steering. In the *intra-country* configuration, the simulations were performed on a quad-core Intel® Xeon® X5460 cluster, *gg-blr*, in the Centre for Development of Advanced Computing (C-DAC), Bangalore, India. The transfer between simulation and visualization site for this *intra-country* configuration was carried out on the National Knowledge Network (NKN). In the *inter-country* configuration, the WRF simulations were conducted on the dual-socket quad-core Intel 64 (Clovertown) PowerEdge 1955 cluster, *Abe*, in National Center for Supercomputing Applications (NCSA), Illinois, USA. Table 6.1 gives the detailed specifications of the two resource configurations including the maximum cores used for simulations, the maximum disk space used by our adaptive framework for the experiments, and the average available bandwidth between the simulation and the visualization sites for each of the configurations.

6.4.2 Weather Model and Cyclone Tracking

We used our framework for tracking cyclone *Aila*. The details of Aila were specified in Section 5.3.1. For the intra-country experiments, we performed simulations for an area of approximately 32×10^6 sq. km. from 60°E - 120°E and 10°S - 40°N over a period of 2 days. For the inter-country experiments, we performed simulations over a larger area or domain and for a longer period of time, since the *Abe* cluster supports faster rate of simulations (approximately 1.5 time steps per second) and has faster Infiniband interconnect. For these inter-country experiments, the domain was approximately from 30°E - 150°E and 10°S - 40°N and the simulation was done over a period of 3 days 18 hours. These domains correspond to the areas of formation and dissipation of Aila.

6.4.3 Framework Implementation

The modifications to the WRF were mentioned in Section 5.3.2. We specified the upper bound for output frequency (OI_{UB}) to be 30 simulated minutes. For the steering interface, we have developed a GUI inside VisIt using Qt. A snapshot of the GUI can be seen in Figure 6.1.

6.4.4 Computational Steering Results

In this section, we demonstrate the user-driven steering supported by InSt, namely, the various steering capabilities provided to the user, the feedback mechanisms in the framework, and the reconciliation of the user-driven steering and automatic tuning or algorithmic steering by the framework. For these experiments, the simulations are started at a particular resolution. However, unlike in the automatic tuning experiments of the previous section, the resolutions of the ongoing simulations can be changed only by the user. Similarly, the placement of the nest for finer simulations can also be performed only by the user.

Intra-country Steering

Figure 6.3 shows the steering results for the *intra-country* experimental setup using the *gg-blr* cluster. The simulation throughput (blue curve), the visualization progress (red curve) and the remaining free disk space (green curve) are shown in the figure. The graph also shows the various steering events provided by the user. Initially, the simulations were started with a resolution of 24 km, and MPR of 5. User input at various stages of the simulation and visualization resulted in steering events. Below, we list the steering events together with the system response and the effect of these events.

- E_1**:** This event occurs after 3.8 hours of execution. In this event the user decides to form a nest based on the visualization output and also provides the nest location. This causes the decision algorithm to reconsider the various system and application parameter values and compute the optimal number of processors and output interval. The formation of nest decreases the simulation rate (blue curve) and hence the slope decreases after the event. There is also a drop in the available disk space (green curve) after this event occurs. This is because of increased rate of output due to the output corresponding to the new nest along with the output of parent domain data. Increased output per time step also decreases the slope of the visualization progress plot (red curve).

- E_2**:** This event occurs after about 4.8 hours of execution. In this event the user decides to refine the simulation to a finer resolution of 18 km. To start at a finer resolution, WRF has to preprocess input data at that resolution. Hence we observe the flat region after E_2 corresponding to the time required for preprocessing. This time depends on the I/O bandwidth of the system. The framework then starts with the user-provided resolution of 18 km. Finer simulation resolution implies more computation time and more output per time step. Therefore we observe further decrease in slope for simulation throughput and visualization progress plots. We also observe reduction in disk space after E_2.

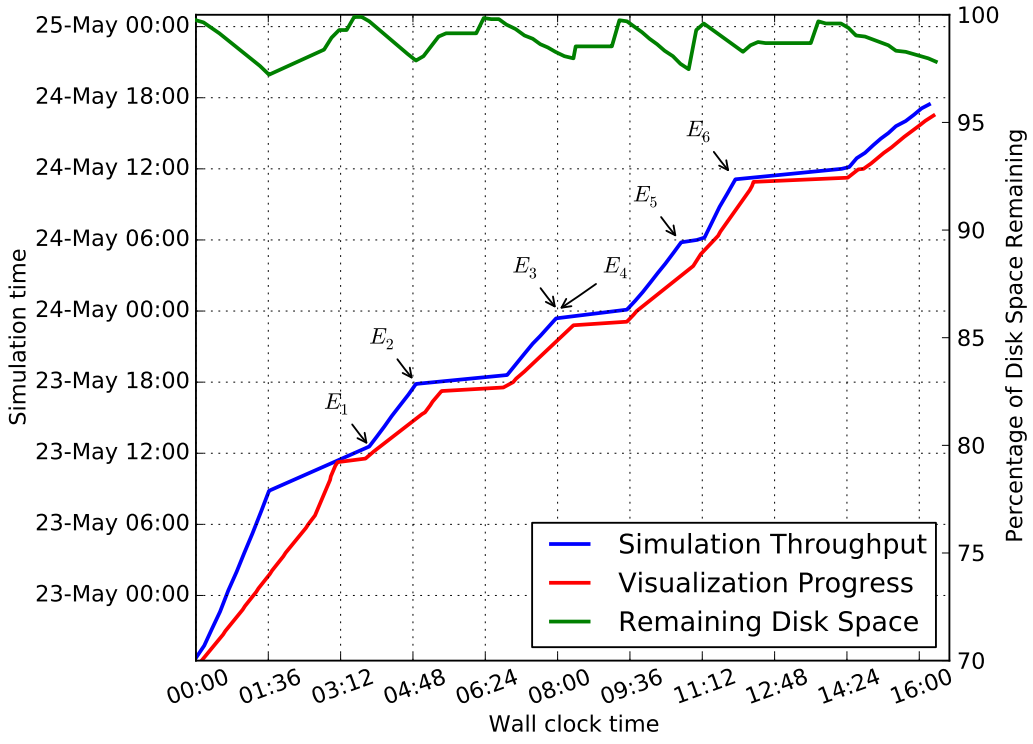


Figure 6.3: Simulation (blue) and Visualization (red) progress, and Disk Consumption (green) for *intra-country* configuration with computational steering. Initial WRF resolution = 24 km, MPR = 5. Events $E_1 - E_6$ affect the simulation throughput and the visualization progress as reflected in the graph.

- **E_3, E_4:** This event occurs after about 8 hours of execution. E_3 denotes the event where the user requests for a simulation rate of 8 but the system denies owing to infeasibility. The maximum rate possible with the upper bound of output interval of 30 minutes at resolution of 18 km is 6 considering the time to solve and time to output in this experimental setup. This value can be estimated using equation (6.1). The system then tries to find a coarser resolution at which the user's desired rate is feasible as explained in Section 6.3. In this case, the system provides an alternate option of making the resolution 21 km. In this way, the system tries to satisfy one requirement of the user, while compromising another based on feasibility analysis. The user can then prioritize resolution over rate or the other way round. E_4 denotes the event where the user asks for a resolution of 21 km, as suggested by the system.

  As these events demonstrate, InSt takes a proactive approach towards user-driven computational steering. While it attempts to steer the simulations based on user inputs, it also analyzes the impact of the user inputs on the criticality of the application, namely, the MPR desired for the simulations, and "advises" the user about possible violations of the "quality-of-service" due to his inputs, and provides him with suitable alternate options. Thus, InSt follows an effective reconciliation approach towards steering executions. Unlike existing work that mainly focuses on user-driven steering, this reconciliation of the user inputs and the criticality needs of the application is very essential for critical weather applications like cyclone tracking.

- **E_5:** This event occurs after about 10.6 hours of execution. In this event the user decides to increase the output interval to 21. As can be seen in the simulation curve, the slope slightly increases signifying an increase in the simulation rate. The simulation rate increases because the increased output interval causes the simulation process to spend lesser amount of time writing files to disk. Increased output interval causes a slight increase in the available disk space after event E_5. This is because the rate of input to the disk decreases whereas the rate of output

from the disk, determined by the network bandwidth, remains almost the same.

- E_6**:** This event occurs after about 11.8 hours of execution. This is where the user decides to refine the resolution to 12 km for better visualization. Since finer resolution implies more time to solve a time step, it can be seen from the graph that the slope of the simulation curve after E_6 is lower compared to before. Finer resolution also increases amount of output produced by the simulation per time step and hence rapid decrease in the available disk space (green curve) can also be observed.

Figure 6.3 shows that the available disk space is always above 95%. This is because of adjusting the number of processors by the decision algorithm to the correct value so that disk space is not a problem even when inputs are given by the user. Fluctuations in the disk curve can be seen at times corresponding to the events E_1 to E_6. Increase in disk space can be seen after the events and before restarting WRF because during this period, the transfer rate remains the same as there is almost no input to the disk.

It can also be observed that the visualization progress closely follows the simulation progress. This is because when the simulation rate is slower due to finer resolutions, the increased amount of output increases the transfer time to the visualization site. Similarly, coarser resolution simulations produce lesser amount of data and thus requires lesser transfer time to the visualization site. For example, simulation at 24 km parent resolution and 8 km nest resolution produces 162 MB data per time step, whereas simulation with parent resolution of 10 km and nest resolution of 3.33 km produces 973 MB data per time step.

Inter-country Steering Results

We also performed *inter-country* steering from the visualization site in India to the simulation site in NCSA, USA. Figure 6.4 shows the results obtained in the *inter-country* configuration with NCSA's Abe cluster in USA for simulations, and the visualization engine in IISc, India. The graph shows the various algorithmic steering events and user-driven steering events. Initially, the simulations were started with a resolution of 18 km,

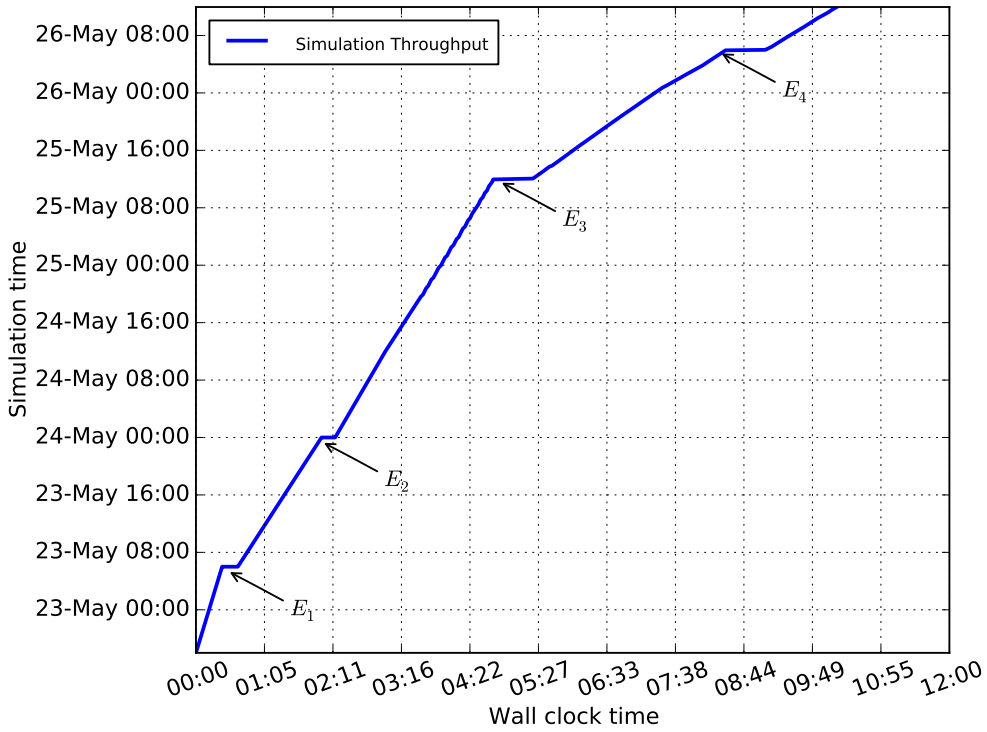


Figure 6.4: Simulation progress for *inter-country* configuration with computational steering. Initial WRF resolution = 18 km, MPR = 3. Both algorithmic and user-driven steering events ($E_1 - E_4$) affect the simulation throughput as reflected in the graph.

and MPR of 3 on 96 processors with an output interval of 30 simulated minutes. Below, we list the algorithmic events (E_1 and E_4) and user-driven steering events (E_2 and E_3) that occurred during the 11 hours of execution. The response to these events in terms of the simulation throughput and visualization progress is similar to the intra-country experiment.

- E_1**:** This event occurs after 30 minutes of execution. In this event, the decision algorithm computes the number of processors as 80. This change from 96 to 80 is because of the rapid disk space consumption due to the high simulation rate at coarser resolution. Decrease in number of processors decreases the rate of simulation as implied by the decrease in slope after E_1.

- E_2**:** This event occurs after 2 hours of execution. In this event, the user requests for change in output interval to 60 simulated minutes. Increase in output interval leads to slight increase in the slope after E_2 because of slight increase in the simulation rate due to fewer disk writes.

- E_3**:** This event occurs after 5 hours of execution. In this event, the user requests for change in resolution from 18 km to 12 km for better simulation output. A significant decrease in slope can be observed after E_3 because of more number of computations per time step.

- E_4**:** This event occurs after 8 hours of execution when finer resolution causes the simulation rate to decrease. The decision algorithm increases the number of processors from 80 to 112 in response to this event and maintains the minimum simulation rate.

6.5 Summary

High-performance simulations, effective "on-the-fly" remote visualizations, and user-driven computational steering of simulations based on feedback to focus on important scientific phenomena are essential for efficient monitoring of critical weather events, and providing timely analysis. In this chapter, we have described our integrated steering framework, InSt, that combines user-driven steering with automatic tuning of application parameters based on resource constraints and the criticality needs of the application to determine the final parameters for simulations. Our steering framework proactively analyzes the impact of user inputs on the criticality of the application, advises the user on violations, guides with alternate options, and arrives at the final agreeable parameters. We have demonstrated the algorithmic and steering aspects of our framework with experiments involving intra and inter country steering. Results of these experiments demonstrate how the framework guarantees a minimum rate of simulation, continuous visualizations, and reconciliation between algorithm and user-driven steering.

In our integrated framework for algorithmic and user-driven steering, there can be

delay in visualization depending on the network bandwidth. The framework may recommend higher output interval depending on resource parameters like higher disk space. This may lead to more delay between simulation and visualization times in medium and low-bandwidth networks. We elaborate on this lag between simulation and visualization times in the next chapter and describe some heuristics for reducing this delay.

Chapter 7

Reducing Simulation-Visualization Lag on Constrained Networks

The delay between the simulation time and the visualization time for a simulation time step should be as minimum as possible in the case of critical weather simulations. In this chapter, we discuss efficient online visualization techniques to reduce the lag between simulation and visualization times.

7.1 Introduction

High-fidelity high resolution numerical weather simulations involve large-scale computations and generate large amount of data. Online visualization of the simulation output enriches the process of scientific discovery and fosters profound and unexpected insights. While the ability to generate data continues to grow in leaps and bounds, the ability to comprehend it all on-the-fly continues to encounter great challenges [38]. This is especially true for high resolution and nested simulations, which can generate gigabytes of data per time step.

7.1.1 Motivation

Efficient online visualization, where the output of the simulation is visualized as soon as it is produced, is highly desirable in critical applications. Though our framework InSt,

as explained in Chapters 5 and 6, adapts to resource constraints and helps in smooth and continuous simulation and visualization of critical applications in resource-constrained environments, it cannot guarantee an upper bound on the *lag* between the time when the simulation produces an output frame and the time when the frame is visualized. It is important to reduce the lag between the simulation and visualization times so that scientists are able to get *on-the-fly* view of the simulation. Also, for critical applications like tsunami prediction, the faster the simulation output reaches the visualization site, the better are the chances of taking active measures on time. In this chapter, we address the critical issue of reducing this lag by adapting to the available resource parameters and the simulation output.

In an environment of continuous simulation and remote visualization, the accumulation of simulated frames in the simulation site will lead to a longer queue of pending frames for visualization, and hence increase the number of frames to be sent to the visualization site before transferring the recently produced frames. The length of this queue will increase with time because simulation will continuously output frames. This, in turn, will increase the lag between when a frame is produced and when it is visualized because of sending all the frames in the queue in order.

7.1.2 Problem Statement

Our work tries to minimize the lag between simulation and visualization times to enable efficient online visualization. A viable solution to this problem can be to discard some frames from the queue. This will reduce the queue size and hence reduce the lag as well. One option can be to send one frame from the queue. But in this case, though the lag will be reduced, the occurrence of important events might go unnoticed. Another option can be to select the most representative set of frames from the queue and discard the rest. This will minimize the queue length to some extent. A different approach can be to increase the output interval so that no frames are discarded. In this case, one should note that increasing output interval is equivalent to decreasing temporal resolution (i.e. the frequency at which successive frames are visualized) without examining the data.

If most of the frames in the queue are redundant, then increasing output interval is a good option, whereas if the nature of data is unknown, i.e., redundancy in the frames is unknown, increasing output interval may lead to missing important events. Hence it is highly important to examine the data before discarding.

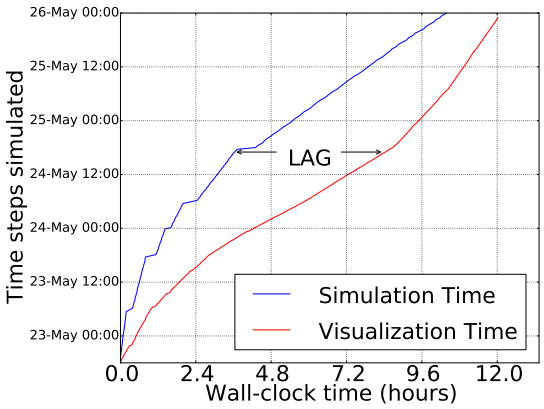
In our work, we transfer a representative subset of pending frames to the visualization site and discard rest of the frames. We have developed three algorithms to reduce the lag - *most-recent*, *auto-clustering* and *adaptive*. The aim is to minimize the lag between simulation and visualization times and also to visualize important events in the simulation. *Most-recent* tries to achieve the best possible lag, *auto-clustering* tries to visualize all important events in the simulation and *adaptive* tries to visualize most of the important events within acceptable lag. Using experiments with different network configurations, we find that in general the adaptive algorithm strikes a good balance in reducing lag and visualization of most representative frames, with up to 72% smaller lag when compared to auto-clustering, and 37% larger representativeness than most-recent for slow networks. We also introduce new measures to evaluate the quality of the visualized output.

7.1.3 Chapter Outline

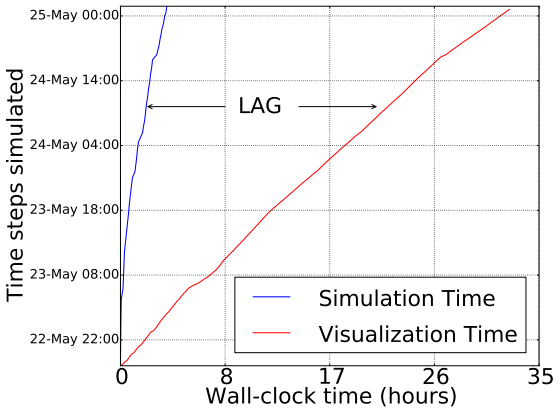
In Section 7.2, we elaborate the problem of simulation-visualization lag. Section 7.3 presents algorithms to reduce simulation-visualization lag. Section 7.4 presents the experimental results on lag reduction on different network bandwidths. Section 7.5 briefly summarizes the contributions. Section 7.6 concludes and enumerates our future efforts.

7.2 Simulation-Visualization Lag

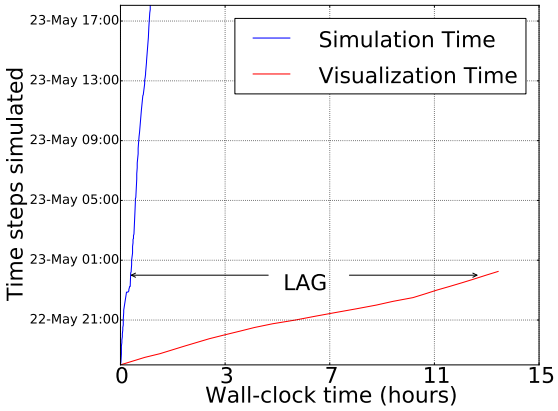
Figure 7.1 shows the lag between the simulation and visualization times for three different network bandwidths between the simulation and visualization sites. We simulate the cyclone Aila (refer Section 5.3.1). During the simulation, a nest is formed over the cyclone and the simulation is refined with growing intensity of the cyclone. The y-axes



(a) High-bandwidth configuration



(b) Medium-bandwidth configuration



(c) Low-bandwidth configuration

Figure 7.1: Simulation times (blue) and visualization times (red) showing the simulation-visualization lag.

in Figures 7.1(a), 7.1(b) and 7.1(c) show the time steps simulated. The x-axes indicate the wall-clock times for simulation and visualization. The blue curve shows the wall-clock time at which a time step was simulated and the red curve shows the wall-clock time when the time step was visualized.

Figure 7.1(a) shows the simulation and visualization times for high-bandwidth (56 Mbps) configuration. It can be seen that slope of the simulation curve keeps decreasing due to gradual refinement of simulation resolution. With finer resolution, the simulation speed decreases and hence the lag between simulation and visualization decreases. Figure 7.1(b) shows the simulation and visualization times for medium-bandwidth (16 Mbps) configuration. The difference between the visualization time and the simulation time for a given time step is more than the high-bandwidth configuration because of lower network speed. In this case, the lag monotonically increases unlike the high-bandwidth case because the initial lag is comparatively higher. Hence, decrease in simulation rate due to increase in simulation resolution does not decrease the lag significantly. The simulation-visualization lag is much more in the low-bandwidth (1.1 Mbps) configuration as shown in Figure 7.1(c). The time taken for a frame to reach the visualization site from the simulation site is more due to the slow network speed between the simulation and visualization sites. Hence a time step is visualized much later than when it was simulated.

In critical applications, it is imperative for simulation output to be visualized as soon as the data is produced. Online/on-the-fly visualization requires that the visualization times closely follow the simulation times. If the network bandwidth is low, then the visualization times might lag behind the simulation times. This lag can increase if the simulation speed is very high and the network bandwidth is very low. This is evident from Figures 7.1(a), 7.1(b) and 7.1(c). The lag for the time step corresponding to 23^{rd} May 00 hours is 25 minutes for high-bandwidth, 3 hours for medium-bandwidth and 12.5 hours for low-bandwidth configurations. A comparison of these figures shows that as the network bandwidth decreases, the lag between the simulation and visualization increases. When the network bandwidth is high, the frames are transferred quicker and

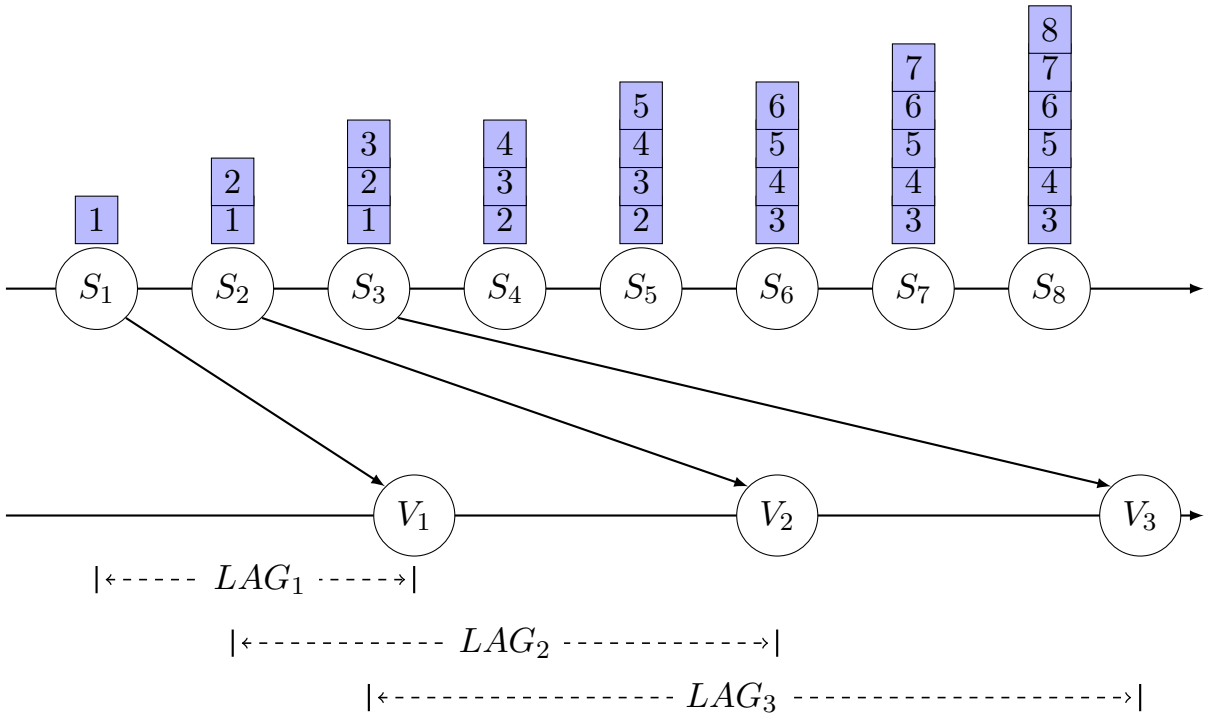


Figure 7.2: Simulation and visualization progress.

hence the number of frames simulated in that time is fewer whereas in the case of low network bandwidth, the frames take longer time to be transferred and hence the number of frames simulated in that time is higher. This leads to lag accumulation as explained below.

An illustration of increasing lag between the simulation and visualization times is shown in Figure 7.2. The two horizontal lines show the simulation and visualization progress. S_i is the simulation time for the i^{th} frame and V_i is the visualization time for the same. LAG_i shows the difference between V_i and S_i, i.e. the time difference between the visualization and simulation of the i^{th} frame. It can be seen from the figure that when the 1^{st} frame reaches the visualization site at V_1, the 2^{nd} and 3^{rd} frames are already produced and are waiting to be sent. When the 2^{nd} frame reaches the visualization site at V_2, the 3^{rd}, 4^{th}, 5^{th} and 6^{th} frames are queued at S_6. Though the i^{th} frame is produced at S_i, it can only be sent after the previously-queued frames are transferred. Therefore the queue size continues to increase as shown by the numbered rectangles in the figure.

Since the number of frames waiting at the simulation site increases, the time between when a frame is produced and when it is visualized also increases. For example, the 8^{th} frame will have to wait in the queue for the 3^{rd}, 4^{th}, 5^{th}, 6^{th} and 7^{th} frames to be sent. In this case, the transfer times of the queued frames will add to the lag for the 8^{th} frame. Thus this leads to cumulative addition of lag for the later frames. Hence LAG_i increases as illustrated in Figure 7.2 for the 1^{st}, 2^{nd} and 3^{rd} frames.

7.3 Reduction of Simulation-Visualization Lag

The scenario shown in Figure 7.2 can be better or worse depending on the network bandwidth between the simulation site and the visualization site. This is evident from the experiments with different network bandwidths (Figure 7.1). If the bandwidth is very low, there will be more number of frames accumulated during the time when a frame is transferred and hence the lag will be more. On the other hand, if the bandwidth is very high, the lag will be less. So it is required for an online remote visualization framework to adapt to the changing network bandwidths and minimize the lag. We have developed strategies in InSt (Section 6.2) for adapting to the network bandwidth and the length of the queue of pending frames, i.e. the frames which are yet to be sent to the visualization site from the simulation site.

7.3.1 Requirements for Online Visualization

As shown in Figure 7.2, the lag between S_3 and V_3 is more than the lag between S_1 and V_1, i.e. the lag for the frames produced later is more than the lag for the earlier frames. This increasing lag is mainly because of two reasons:

1. Sending all the frames.
2. More frames being produced by the simulations in the time when a frame is being transferred.

A simple strategy that will not lead to increasing lag will be to increase the output interval i.e. to not produce excess frames if the network bandwidth is low. Though this

will ensure that the frames are sent for visualization as soon as they are produced, the high output interval may result in missing important events between two consecutive frames. Since the purpose of visualization is to identify important events, this strategy is not desirable. Hence, in our work we do not consider this strategy of increasing the output interval to reduce the lag.

It is easy to observe that with increasing queue of pending frames, the lag for successive frames keeps increasing. This gives the idea of decreasing the queue length to decrease the lag. The approach we follow to decrease the queue length is to drop some frames from the queue. In our work, we choose a subset of frames from the queue and discard the rest so that the queue size decreases, which in turn will reduce the lag. Our framework adapts to different network bandwidths to reduce the lag. The higher the network bandwidths, the lesser the number of frames that will be dropped by our framework.

The criterion to drop frames must adhere to either of the two conflicting goals:
Case 1: The sent frames contain useful information - In this case, the goal is to send good quality frames. The quality of the frames can be based on the amount of non-redundant information contained in the frames that are sent. The objective is to send the most representative frames, i.e. the frames which represent their immediate temporal neighborhood well. The representative frames are chosen such that they are somewhat distinct from each other. However this will not give the best possible lag because there may be many important frames in the queue.
Case 2: Minimal lag is maintained - In this case, the goal is to always maintain the best possible lag irrespective of whether all important information is visualized or not.

Thus we need to either compromise the quality of visualized frames or the lag between the simulation and visualization times. We have extended the *Frame Sender* (Section 5.2.3) component for dynamically deciding whether to send or discard the pending frames. We describe this in detail in the following subsections.

7.3.2 Frame Selector

We reduce the simulation-visualization lag by extending the frame sender in our framework to include a *Frame Selector* component as illustrated in Figure 7.3. The Frame Selector invokes a frame selection algorithm to select frames that will be sent to the visualization site. These algorithms are explained in the next section.

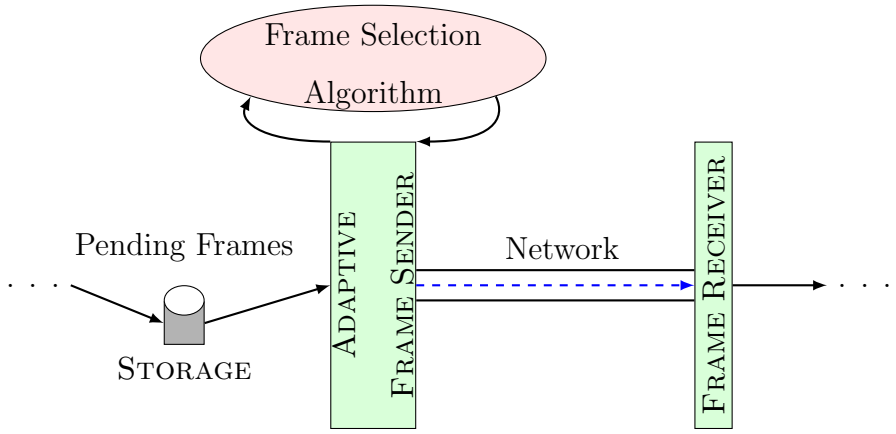


Figure 7.3: Frame Selector.

7.3.3 Strategies for Selection of Time steps to Reduce the Lag

We have developed three frame selection algorithms for the *Frame Selector* component of InSt. These algorithms are described below.

Most-Recent

This is a simple strategy that selects the frame that is most recently generated by the simulation process for sending to the remote visualization site. If t_cur is the current time when a frame is chosen, t_gen is the time when the frame is generated by the simulation process, and t_tran is the time for transferring the chosen frame to the visualization site, the most-recently generated frame results in minimal value of $(t\_cur - t\_gen) + t\_tran$ among all the frames in the queue, since the t_gen for the most recent frame is the highest. Thus this strategy aims to reduce the lag between the visualization and simulation of a frame to the minimal value. However, it is important to note that the most recent frame

may not be the most representative frame of all the frames in the queue. This algorithm has time complexity of $O(1)$.

Auto-clustering

This algorithm reduces the lag as well as sends only useful information to the visualization site. The strategy is to select some frames from the queue of pending frames and to discard some based on the importance of each frame. In our work, we decide the importance of a frame based on how well that frame represents the other frames in its temporal neighborhood. Since sending all the frames can result in large simulation-visualization lag, as discussed in Section 7.3.1, this strategy reduces the lag by discarding some frames. The other important requirement is to visualize the significant temporal phases in the simulation output. *Auto-clustering* achieves this by retaining some representative frames so that useful information is not lost.

In the first part of this algorithm, we examine the current queue of pending frames and form temporal non-overlapping clusters as shown in Figure 7.4. The different colours represent the different clusters. These clusters represent phases in the queue of pending frames. The phases are determined by comparing the root mean square distance of the

Figure 7.4: Auto-clustering Strategy.

values of a varying field between two successive frames. If P_1 and P_2 are two vectors, then the root mean square distance and the normalized root mean square distance are given by Equations (7.1) and (7.2) respectively.

$$RMSD(P_1, P_2) = \sqrt{\frac{\sum_{i=1}^{N}(x_{1,i} - x_{2,i})^2}{N}} \tag{7.1}$$

$$NRMSD = \frac{RMSD}{x_{max} - x_{min}} \tag{7.2}$$

After forming the clusters, a representative frame from each cluster is chosen such that it has the least standard deviation among the frames in that cluster. These are colored

black in Figure 7.4. The pseudocode for this is shown in Algorithm 7.1. The algorithm takes as input the set of pending frames $\mathcal{F} = \{f_1, f_2, ..., f_n\}$, which are queued at the simulation site and outputs the set of representative frames $\mathcal{R} = \{r_1, r_2, ..., r_k\}$, $\mathcal{R} \subseteq \mathcal{F}$, for sending to the visualization site.

In the algorithm, the number of clusters k is determined in the lines 1–11. We calculate the normalized root mean square distance (NRMSD) of pressure variable between every two consecutive frames using Equation (7.2) and find the standard deviation. In our simulations, the value of pressure decreases over time, thereby increasing the function range. Relative values and ranges of the variables affect clustering results [48]. Therefore, we use NRMSD in order to avoid biasing the RMSD by higher function range. The number of clusters is determined by the number of frames having a high standard deviation. In our algorithm we have chosen a threshold of 0.4 standard deviation above the mean to define large distances of frames from the previous frames. From initial observations, we found that 0.4 standard deviation provides a balance between very few and very high number of clusters. Thus, the algorithm uses the principle that if $\mathcal{F}_i$ is distinctly different from $\mathcal{F}_{i+1}$, it may imply a change of phase.

Once the number of clusters is determined, we then find the cluster centres using an iterative method similar to the well-known k-means [77] clustering algorithm. Unlike the traditional k-means approach, we aim to find a temporal clustering, which means that the clusters are sequenced according to the temporal order. The reason for this is to capture distinct temporal phases among the frames in the queue. Each cluster has a common boundary with each of its neighboring cluster to its right side and to its left side as shown in Figure 7.4. Initially, we place the cluster centres at equal distances from each other. In each iteration, each frame is assigned to the closest cluster centre among the two centres to its left and right as shown in lines 13–16 of Algorithm 7.1. After assigning all the frames to one of the cluster centres to its left or right, a new cluster centre is determined for each cluster based on the standard deviation of root mean square distance. In each of the clusters, the one which has the least standard deviation is selected as the new cluster centre (lines 17–19).

Input: The set of pending frames $\mathcal{F}$

foreach $i \in \mathcal{F}$ **do**
    Find $rms(\mathcal{F}_{i-1}, \mathcal{F}_i)$ using Equation (7.2);
    $rms[\mathcal{F}_i] \leftarrow rms(\mathcal{F}_{i-1}, \mathcal{F}_i)$;
end
$avg\_rms \leftarrow$ average of $rms[\mathcal{F}_i] \; \forall \; i \in \mathcal{F}$;
$k \leftarrow 0$; /* k is the number of clusters */
foreach $i \in \mathcal{F}$ **do**
    if *(avg_rms - $rms[\mathcal{F}_i] \geq$ threshold)* **then**
        $k \leftarrow k + 1$;
    end
end
/* Let $\mathcal{C}(\mathcal{G}_1), \mathcal{C}(\mathcal{G}_2), ..., \mathcal{C}(\mathcal{G}_k)$ be the frames that represent the centers of k clusters */
/* Initially, the clusters centers are equally spaced */
/* Refine the cluster centres */
repeat
    /* Form clusters and cluster boundaries using the cluster centers */
    foreach $j \in \mathcal{F}$ **do**
        $\mathcal{G}_p \leftarrow \underset{i=\{left,right\}}{\text{argmin}} (rms(\mathcal{F}_j, \mathcal{C}(\mathcal{G}_i)))$;
        add j to members($\mathcal{G}_p$);
    end
    foreach $i \in \mathcal{G}$ **do**
        $\mathcal{C}(\mathcal{G}_i) \leftarrow \underset{i \in members(\mathcal{G}_i)}{\text{argmin}}$ standard deviation(i);
    end
until *there is no change in the cluster centres* ;
foreach $i \in \mathcal{G}$ **do**
    $\mathcal{R}_i \leftarrow \mathcal{C}(\mathcal{G}_i)$;
end

Output: The set of representative frames $\mathcal{R}$

Algorithm 7.1: Auto-clustering Algorithm

The space requirements for this algorithm are modest because only the data points and centroids are stored, similar to that of k-means [124]. Specifically, the storage required is $O(n + k)$, where n is the number of points and k is the number of clusters. The time required is $O(I*n)$, where I is the number of iterations required for convergence. We have found empirically that this algorithm converges fast and I is very small, therefore this algorithm is linear in the number of data points.

Although auto-clustering does not guarantee minimal lag, it selects representative frames from the queue of pending frames to retain the frames containing significant information. It sends useful information to the visualization site so that important phases in the simulation are visualized.

Adaptive

Adaptive algorithm is a hybrid strategy that combines the characteristics of the *most-recent* and *auto-clustering* methods because it is important to reduce the lag as well as to visualize the important phases of the simulation. In this algorithm, utmost importance is given to visualizing the frames as soon as they are produced by the simulation process. The user/scientist can specify an upper bound LAG_UB to limit the simulation-visualization lag.

As discussed in Section 7.3.1, one way to reduce lag is to drop frames. But dropping many frames may result in too much loss of information. Another approach to reduce lag without losing too much information is to reduce the size of each frame. Since we employ frame compression as a size reduction technique in all our strategies, we consider one more kind of size reduction in this adaptive strategy. The specific size reduction technique is to remove less important information from a frame. Scientific data produced by simulations have different sets of parameter values, and these sets can be prioritized into different levels of importance based on the specific needs of the scientists. For example, weather data has different sets of variables like pressure, temperature, wind velocities, humidity, precipitation, etc. For the critical weather application of cyclone tracking, pressure is the most important variable. A cyclone is characterized by the

continuous drop in pressure and high wind velocity at the centre of the cyclone. So we form different levels of information by retaining different sets as enumerated below.

1. Level 0: All variables
2. Level 1: Pressure, Wind Velocities, Temperature
3. Level 2: Pressure

Thus, we can reduce the size of a frame to different levels by retaining different sets of most important parameters/data in the frame and eliminating the others. Since the total time to send data across the network depends on the size of data, the time to transfer will decrease if we reduce the size of each frame by sending the most important information in the frame.

Input: The set of pending frames $\mathcal{F}$ and the set $\mathcal{L}$ of different levels of information in descending order of amount of information content

`/* Invoke auto-clustering algorithm to get the set of representative frames` $\mathcal{R}'$ `*/`

$\mathcal{R}'$ = auto-clustering($\mathcal{F}$)

`/* Adaptively choose an appropriate level for the chosen frames */`

foreach *representative frame* $i \in \mathcal{R}'$ **do**
  foreach *level* $j \in \mathcal{L}$ **do**
    $curr\_frame \leftarrow j^{th}$ level of information in i;
    $curr\_transfer\_time \leftarrow$ time to transfer $curr\_frame$;
    if $(curr\_transfer\_time \leq LAG\_UB)$ **then**
      add i to $\mathcal{R}$;
      break;
    end
  end
end

Output: The set of representative frames $\mathcal{R}$

Algorithm 7.2: Adaptive Algorithm

It may not be always possible to send the full simulation output to the visualization site within the lag limit, so this algorithm tries to send as much information as possible for visualization. At first, the adaptive algorithm invokes the auto-clustering algorithm explained in Algorithm 7.1. For each of the representative frames output by auto-clustering, the adaptive algorithm checks if the full frame can be sent without violating the lag limit LAG_UB, i.e. it checks whether the difference in the times between when the frame will reach the visualization site and when it was produced by the simulation process will be less than LAG_UB. If it cannot send the full frame without violating the lag limit, then it checks whether it can send the frame with the next level of reduced information content such that the lag is less than LAG_UB. There can be multiple such levels of reduced information content depending on the amount of information in the simulation output. With each level of reduced information in a frame, the time to send the frame also decreases since the time to transfer data is directly proportional to its size. If even the lowest level of reduced frame content cannot be sent, then the adaptive algorithm discards the frame and considers the next representative frame. The pseudocode for this is shown in Algorithm 7.2.

This technique will ensure that the lag for the visualized frames is always less than LAG_UB. When the algorithm decides to send a frame with partial data, the time to transfer that frame is also less which implies that the number of pending frames accumulated in the queue within that time is fewer. Hence the rate at which the queue length increases is lesser when reduced frames are sent. When accumulation of pending frames is less, then in the next iteration of the Frame Selector, the algorithm will most likely select a full frame for visualization within the lag limit. Hence the *adaptive* algorithm is able to adapt to network conditions and current queue size. It adaptively decides whether to send or not and how much information to send. We elaborate this using experimental results in Section 7.4. The time complexity of this algorithm is similar to that of *auto-clustering*.

InSt invokes the frame selection algorithm when the frames are in transit from the simulation to the visualization site, hence this ensures that the time required for the frame

selection does not increase the simulation-visualization time. Hence the time for frame selection need not be considered in the optimization problem described in Section 5.2.4. For example, in the high-bandwidth case, the maximum queue length is 3 when frame selection algorithm is used. The transfer time for a full frame at 18 km resolution is around 1 minute and the frame selection algorithm runs in less than 0.3 seconds for clustering 3 frames. However, in certain cases like slow network bandwidths, where the queue size can be very large, the frame selection algorithm execution time may exceed the frame transfer time. In those cases, the frame selection algorithm considers a subset of the pending frames so that its execution time does not surpass the transfer time. The limitation of this approach is that considering a subset of frames may result in choosing different representative frames. However, we have found experimentally that this simple modification maintains the quality of the data reasonably well as has been shown in Section 7.4.

7.4 Experiments and Results

In this section we present experimental results of using the frame selection algorithms for simulation-visualization lag reduction by InSt. The frame selection algorithms are sequential in the current work, and are executed on a single processor at the simulation site. The experiments have been conducted on the high-bandwidth, medium-bandwidth and the low-bandwidth configurations, shown in Table 7.1.

7.4.1 Evaluation Strategies

We measure performance improvement over the original *all* algorithm in terms of simulation-visualization lag. We also compare the frame selection algorithms in terms of reproducibility of information with respect to the original set of frames. The latter comparison is done with the aim of demonstrating that the resulting visualization does not hinder the quality of data for scientific analysis. We measure the quality of frames in the following three ways.

Table 7.1: Simulation and Visualization Configurations

Configuration	*Simulation Configuration*	*Maximum Cores for Simulation*	*Maximum Disk Space Used*	*Average Sim-Vis Bandwidth*
High-bandwidth	*fire*: 12x2 dual-core AMD Opteron 2218 based 2.64 GHz Sun Fire servers, CentOS release 4.3, each with 4 GB RAM, 250 GB Hard Drive, and connected by Gigabit Ethernet	48	180 GB	56 Mbps
Medium-bandwidth	*gg-blr*: HP Intel Xeon quad-core processor X5460, 40 nodes, 320 3.16 GHz cores, RHEL 5.1 on Rocks 5.0 operating system, each with 16 GB RAM and 500 GB SATA based storage, and connected by Infiniband	90	150 GB	40 Mbps
Low-bandwidth	*kraken*: Two 2.6 GHz six-core AMD Opteron processors (Istanbul) per node, 9,408 compute nodes, Cray Linux Environment (CLE) 2.2, 16 GB of memory per node and connected by Cray SeaStar2+ router	288	700 GB	1.1 Mbps

First, we find the root mean square (rms) distance between the successive frames because mean squared error has been used in literature as an objective distortion measure. Minimizing mean squared error leads to better perceptual quality [90]. The root mean square distance between the successive frames gives a measure of continuity for the chosen output interval and captures the variation between successive frames. If a frame selection algorithm has the same variation as the original *all* algorithm, then the frame selection algorithm closely follows the original algorithm. In the absence of singular events, higher output interval will correspond to higher rms.

Second, we look at the probability distribution of the data which gives the spread or variability in the data values output from the *all* and the frame selection algorithms. A similarity in the probability distributions will imply similarity in the frequencies of the range of data values output by the algorithms.

In WRF, a moving nest captures the movement of the cyclone and the nest centre is placed at the eye of the cyclone. An important feature of the cyclone tracking application is to track the eye of the cyclone. Hence the nest movement in WRF is important because it follows the movement of the eye of the cyclone. However, the eye of the cyclone does not move every time step and hence the nest also does not move every time step. Though the frame selection algorithms drop frames in order to reduce the lag, they ideally should capture the frames in which the nest position changes. Hence we also compare the algorithms in terms of how well the frames chosen by the algorithms capture the nest movement.

7.4.2 High-bandwidth Configuration

We present the results for high network bandwidth between simulation and visualization sites in this section. As explained in Section 7.3.3, for the *adaptive* algorithm, the user/scientist can specify an upper bound on the simulation-visualization lag. We have experimented with upper bounds of 20 and 30 minutes for simulation-visualization lag.

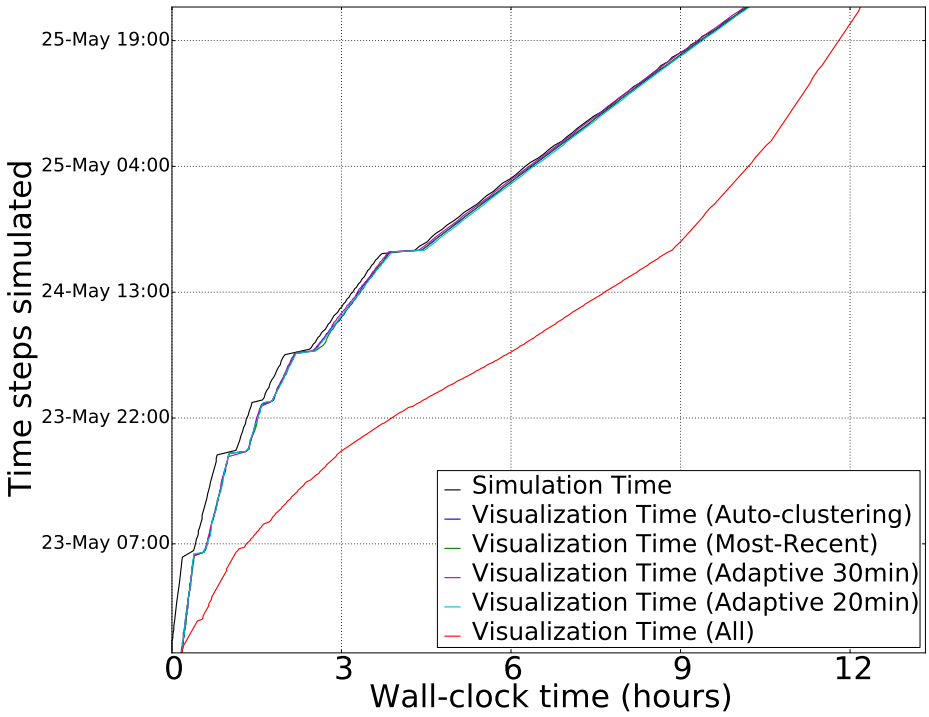


Figure 7.5: Simulation and visualization times for high-bandwidth configuration. Visualization curves (except for *all*) are very close to the simulation curve.

Simulation-visualization lag

Figure 7.5 illustrates the lag between the simulation and visualization times for our frame selection algorithms, by showing the simulation times when the frames are generated at the simulation site and the visualization times at the visualization site corresponding to the different algorithms. The red "all" curve in the figure corresponds to the default policy of sending all the frames generated at the simulation site to the visualization site. It shows the visualization times for *all*. It can be observed that the visualization time lags by almost 4 hours for the default policy for the 24^{th} May 13:00 hours simulation time step. The visualization times for almost all the frame selection algorithms are very close to the simulation times. This is because the frame selection algorithms are able to prune the frames to be sent and hence can reduce the *lag accumulation* of pending

frames that was explained in Section 7.3. The dropping of frames by the frame selection algorithms and the high network bandwidth keeps the queue length to the minimal and hence reduces the lag.

Rms distance between successive frames

Table 7.2 shows the rms distance for *all* and the frame selection algorithms. We have taken the rms distance with respect to the variable perturbation pressure P. The table shows that there is not much variation in the average rms distance between the successive frames for all the frame selection algorithms. The average rms distance for the frame selection algorithms do not differ much from the original *all* strategy. This suggests that there is no major information lost even if frames were dropped between two successive frames at the visualization site.

Table 7.2: Statistics for rms distance between successive frames for high-bandwidth configuration

Frame Selection Algorithm	Minimum	Maximum	Average
All	0.0016	0.0110	0.0039
Auto-clustering	0.0017	0.0114	0.0045
Most-recent	0.0019	0.0114	0.0045
Adaptive (30 minutes)	0.0023	0.0114	0.0047
Adaptive (20 minutes)	0.0018	0.0111	0.0044

Histogram of data distribution

The probability distribution of data can be obtained from the histogram of the data distribution in the frames selected by the frame selection algorithms. The histogram for *all* is shown in Figure 7.6 for the variable *perturbation pressure*. It can be seen that most of the pressure values for all the time steps lies in the bin number 150, which correspond to the perturbation pressure range of -92 Pa to -82 Pa.

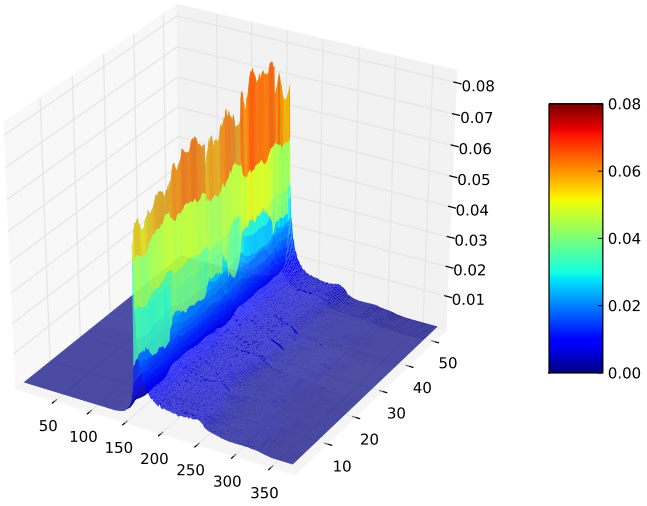


Figure 7.6: Histogram for *all* for high-bandwidth configuration.

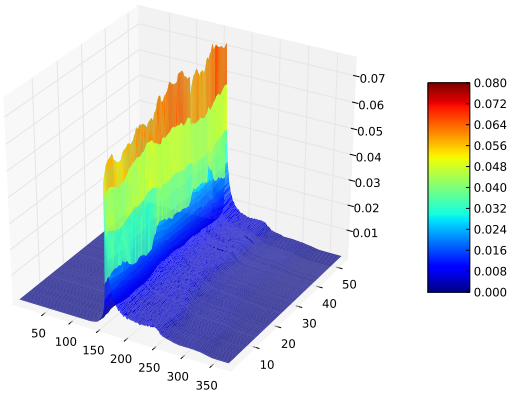


Figure 7.7: Histogram for the auto-clustering algorithm for high-bandwidth configuration.

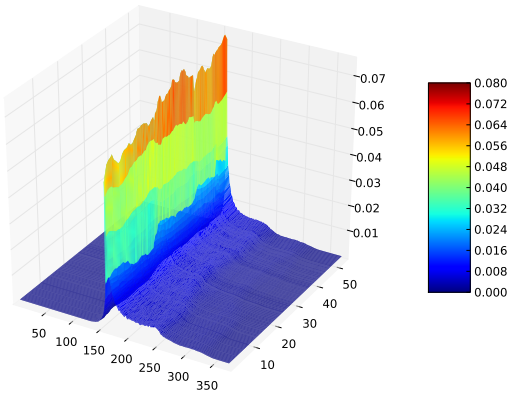


Figure 7.8: Histogram for the most-recent algorithm for high-bandwidth configuration.

Figures 7.7 and 7.8 show the histograms for auto-clustering and most-recent frame selection algorithms. Figures 7.9 and 7.10 show the histograms for adaptive algorithm with lag bound of 30 and 20 minutes. It can be observed that similar to histogram for *all* in Figure 7.6, most of the pressure values for all the time steps lies in the bin number 150 for the frame selection algorithms.

We show the histogram similarity between the *all* and the frame selection algorithms

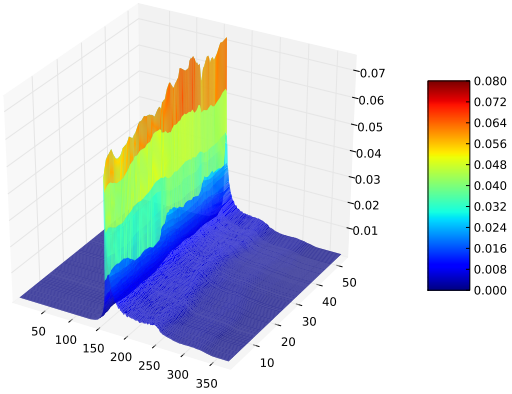


Figure 7.9: Histogram for the adaptive algorithm with lag bound of 30 minutes for high-bandwidth configuration.

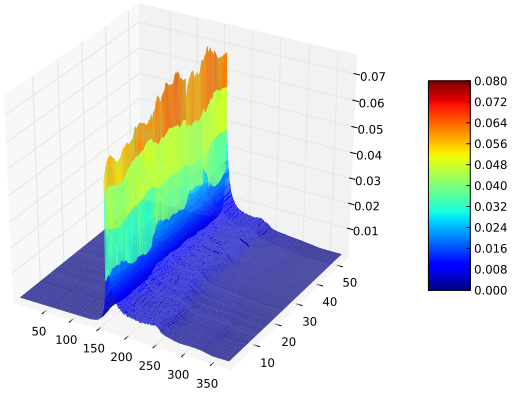


Figure 7.10: Histogram for the adaptive algorithm with lag bound of 20 minutes for high-bandwidth configuration.

by calculating the volume between the histograms. The volume enclosed between the histograms of a frame selection algorithm and the *all* algorithm is shown in Table 7.3. It should be noted that the lesser the volume, the more similar are the histograms. It

Table 7.3: Volume between All and Frame Selection Algorithms for high-bandwidth configuration

Frame Selection Algorithm	Volume
Auto-clustering	12.1632
Most-recent	14.0009
Adaptive (30 minutes)	16.1188
Adaptive (20 minutes)	9.9505

can be observed that the volume is minimum for the *adaptive* algorithm with lag bound of 20 minutes, followed by *auto-clustering*. The *adaptive* algorithm with lag bound of 20 minutes sends some frames with reduced information which leads to reduced transfer times and hence more frames can be sent. Therefore it is able to perform better than *auto-clustering* which always sends full frames. Both these algorithms perform better than the *most-recent* because the most recently produced frame may not be the most

representative frame in the queue. The histogram for the *adaptive* algorithm with lag bound of 30 minutes is the most dissimilar to the histogram for *all* algorithm. This is because the lag limit of 30 minutes allows full frames to be transferred initially because of higher lag limit in comparison to the lag of 20 minutes. Hence in the case of 30 minutes bound, after sending full frames initially, the framework is unable to send frames even with reduced information. This is because the full frames incur high transfer times and that increases the number of pending frames. Therefore, during the simulation, sometimes the frames in the front of the queue cannot be sent in order to maintain the lag bound. This leads to discarding many representative frames and thus the *adaptive* algorithm with lag limit of 30 minutes performs worse than others.

Nest positions

Figure 7.11 shows the frames in which the nest position changes from the position in the previous frame. The dots in the graphs depict the frames corresponding to changes in nest positions. These are shown for those frames which are chosen by the frame selection algorithms. The number of nest position changes captured by *all*, *auto-clustering*, *most-recent*, *adaptive* with lag bound of 30 minutes and *adaptive* with lag bound of 20 minutes are 114, 102, 97, 94 and 106 respectively. This shows that the *adaptive* with lag bound of 20 minutes and the *auto-clustering* algorithms are able to capture most of these nest movements. Nest position changes also imply temporal phase changes in the frames and since we select representative frames, we are able to capture most of these changes.

7.4.3 Medium-bandwidth Configuration

We present results for medium network bandwidth between simulation and visualization sites in this section. For the *adaptive* algorithm, we experimented with upper bound of 15 minutes for the simulation-visualization lag.

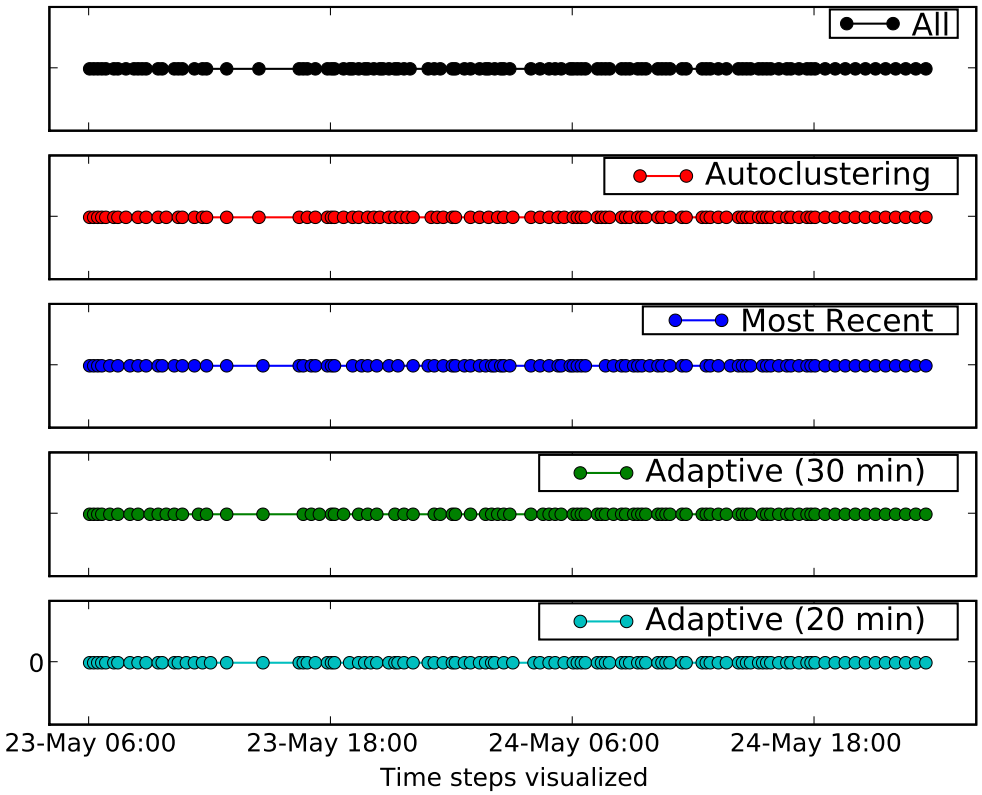


Figure 7.11: Nest position changes for high-bandwidth configuration.

Simulation-visualization lag

Figure 7.12 illustrates the lag between the simulation and visualization times for our frame selection algorithms. The red "all" curve shows the visualization times for the default policy of sending all the frames generated at the simulation site to the visualization site. It can be observed that the visualization time lag is more than the high-bandwidth network. Our frame selection algorithms are able to reduce the lag considerably. All the three strategies *most-recent*, *auto-clustering* and *adaptive* perform almost similarly. In comparison to high-bandwidth case, the simulation-visualization lag is higher for our frame selection algorithms because of slower network bandwidth.

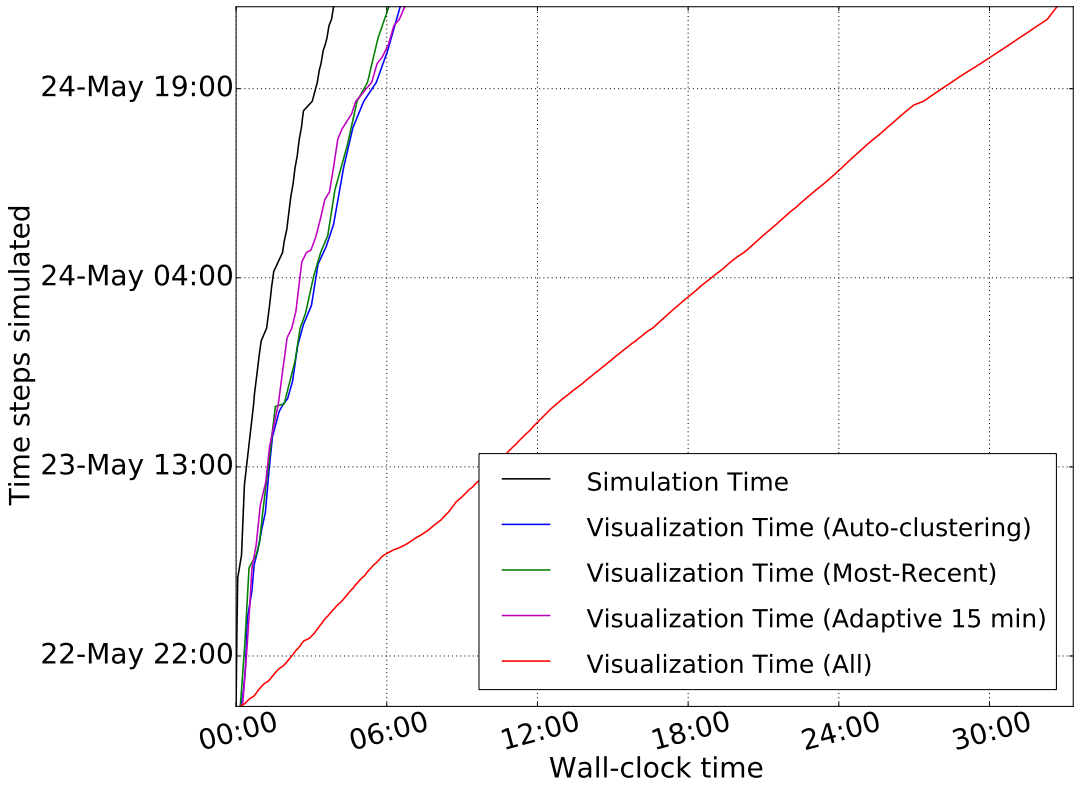


Figure 7.12: Simulation and visualization times for medium-bandwidth configuration.

7.4.4 Low-bandwidth Configuration

We present the results for low network bandwidth between simulation and visualization sites in this section. For the *adaptive* algorithm, we experimented with upper bound of 45 minutes for the simulation-visualization lag.

Simulation-visualization lag

Figure 7.13 illustrates the lag between the simulation and visualization times for our frame selection algorithms. It can be observed that *most-recent* performs the best in terms of lag. The *adaptive* algorithm considerably improves the simulation-visualization lag because it sends the representative frames only if it can meet the lag bound. The *adaptive* algorithm reduces the lag by almost 86%. The lag for *auto-clustering* is better

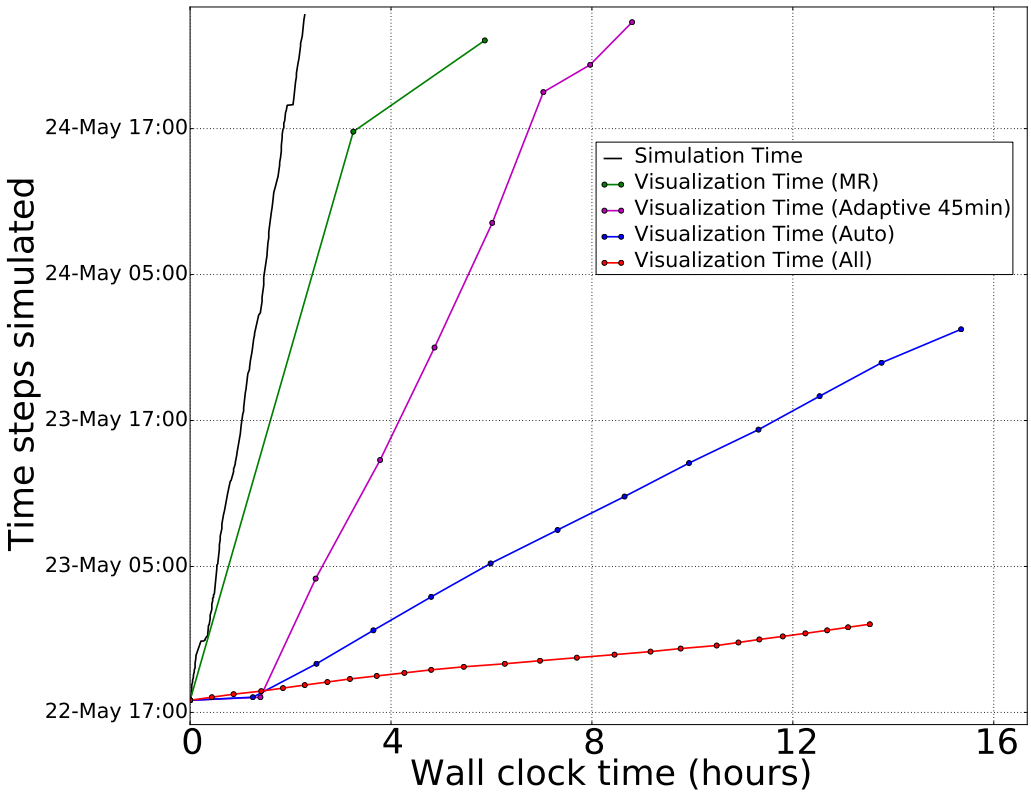


Figure 7.13: Simulation and visualization times for low-bandwidth configuration.

than *all* but worse than the *adaptive* because *auto-clustering* algorithm sends all the representative frames. Sending full representative frames on a low-bandwidth network requires more transfer time and hence auto-clustering incurs more lag. Adaptive sends partial or full frame depending on the current lag, and hence reduces the lag substantially.

Rms distance between successive frames

Figure 7.14 shows the rms distance between successively visualized frames. The minimum, maximum and average rms distance for the frame selection algorithms are shown in Table 7.4. It can be seen that the rms distance for frames sent by *most-recent* are quite high due to the huge gap between the successive frames. The difference in average rms distance between frames for *most-recent* and *all* is the highest. The average rms

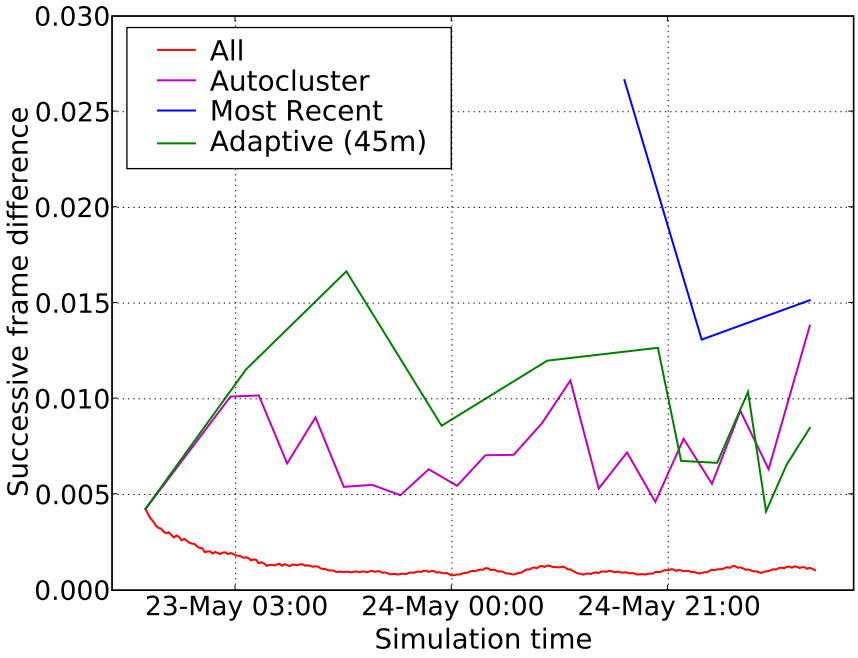


Figure 7.14: Rms between successively visualized frames.

distance for the *adaptive* algorithm is higher than *auto-clustering* because the latter has lesser output interval between two successive frames. Thus, though the *auto-clustering* algorithm has more simulation-visualization lag, it provides with a better continuity between the frames sent. The *adaptive* algorithm is thus able to strike a good balance in providing reduced simulation-visualization lag and choosing the most representative frames.

Table 7.4: Statistics for rms distance between successive frames for low-bandwidth configuration

Frame Selection Algorithm	Minimum	Maximum	Average
All	0.0007	0.0042	0.0012
Auto-clustering	0.0042	0.0138	0.0073
Most-recent	0.0130	0.0266	0.0182
Adaptive (45 minutes)	0.0041	0.0166	0.0090

Histogram of data distribution

The histogram for *all* is shown in Figure 7.15 for the variable perturbation pressure. Most of the pressure values for all the time steps lie in the bin number 500. Similarly,

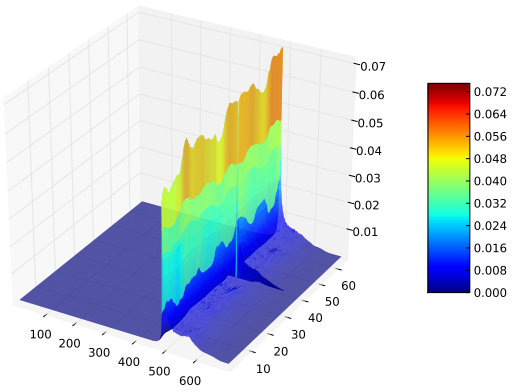


Figure 7.15: Histogram for *all* for low-bandwidth configuration for the variable perturbation pressure.

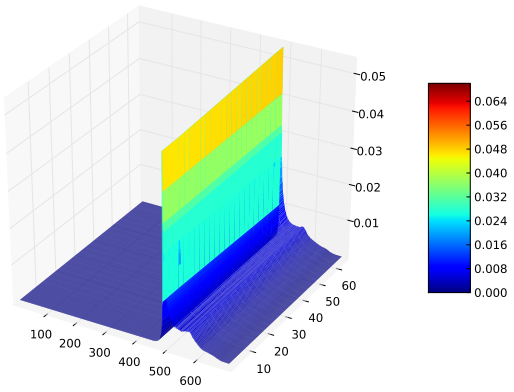


Figure 7.16: Histogram for the auto-clustering algorithm for low-bandwidth configuration.

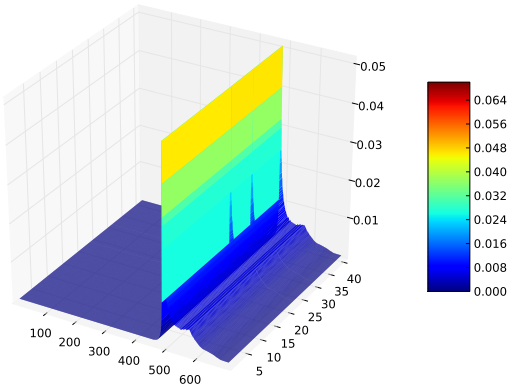


Figure 7.17: Histogram for the most-recent algorithm for the low-bandwidth configuration.

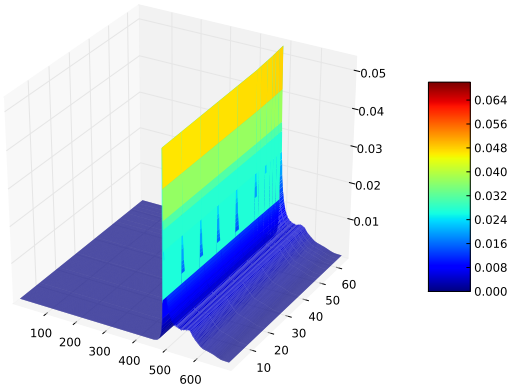


Figure 7.18: Histogram for the adaptive algorithm with lag bound of 45 minutes for the low-bandwidth configuration.

most of the pressure values lie in bin number 500 for auto-clustering, most-recent and adaptive algorithm with lag bound of 45 minutes, as shown in Figures 7.16, 7.17 and 7.18 respectively. There is some dissimilarity between the histogram for our frame selection

algorithms and the histogram for *all* because our algorithms send fewer frames to reduce the lag.

Table 7.5 shows the volume enclosed between the histogram for *all* and the histograms for the frame selection algorithms. It can be seen that the volume for *auto-clustering* is

Table 7.5: Volume between All and Frame Selection Algorithms for low-bandwidth configuration

Frame Selection Algorithm	Auto-clustering	Most-recent	Adaptive (45 minutes)
Volume	58.38	161.12	101.52

the lowest, i.e. it is the most similar to the *all*. This is because *auto-clustering* sends all the representative frames unlike the *most-recent* and the *adaptive*. Though *most-recent* has the minimum lag as shown in Figure 7.13, it is the most dissimilar to *all*. Hence sending the most recent frame in the queue may not be the best strategy to reduce simulation-visualization lag in terms of the quality of the frames sent. The *adaptive* algorithm performs better than the *most-recent* but its volume difference from the *all* strategy is larger than for *auto-clustering* because the *adaptive* never sends any frame if it is unable to meet the specified lag bound. This suggests that higher the lag bound, more the number of frames that can be sent by the *adaptive* algorithm. Therefore increasing the lag bound may improve the information content of the frames sent. To verify this, we executed the simulation and visualization in an emulated environment in which the low-bandwidth configuration was emulated on a high-bandwidth network by introducing delays while transferring output frames from the simulation to the visualization site. We used the high-bandwidth configuration between simulation and visualization in Indian Institute of Science for the emulations. The emulated low network bandwidth corresponded to 1.4 Mbps. The volume for the histograms for this experiment is shown in Table 7.6. We can see that the volume decreases as the lag bound is increased. The volume for the *adaptive* strategy with upper bound lag of 150 minutes approaches the volume for *auto-clustering*, while the corresponding lag for *adaptive* strategy is half of

that for *auto-clustering*.

Table 7.6: Volume and Lag between All and Frame Selection Algorithms for emulated low-bandwidth configuration

Frame Selection Algorithm	Volume	Lag at 24 May 20:00 hours
Auto-clustering	62.56	300 minutes
Adaptive (45 minutes)	188.79	45 minutes
Adaptive (75 minutes)	110.01	60 minutes
Adaptive (150 minutes)	85.49	150 minutes

Nest positions

Figure 7.19 shows the frames in which the nest position changes from the position in the previous frame. The dots in the graphs depict the frames corresponding to changes in nest positions. These are shown for those frames which are chosen by the frame selection algorithms. The number of nest position changes captured by *all*, *auto-clustering*, *most-recent* and *adaptive* with lag bound of 45 minutes are 148, 12, 2 and 6 respectively. When compared to the high-bandwidth configuration results, the low-bandwidth configuration leads to very small number of nest position changes captured by *most-recent*, *auto-clustering* and *adaptive* strategies due to the slow network. The *auto-clustering* and *adaptive* strategies lead to better distribution of nest position changes than the *most-recent* that captures the nest position changes only in the later part of the application progress. This is because the *most-recent* selects the most recent frame in the queue without any consideration about the representativeness of the chosen frame, so it is possible that the chosen frame does not have changes in the nest position. *Auto-clustering* best captures the nest position changes at the expense of increased simulation-visualization lag. When compared to the *adaptive* strategy, *auto-clustering* exhibits a steady-state behavior of sending the most representative frames, and hence has a higher chance of sending frames with nest position changes.

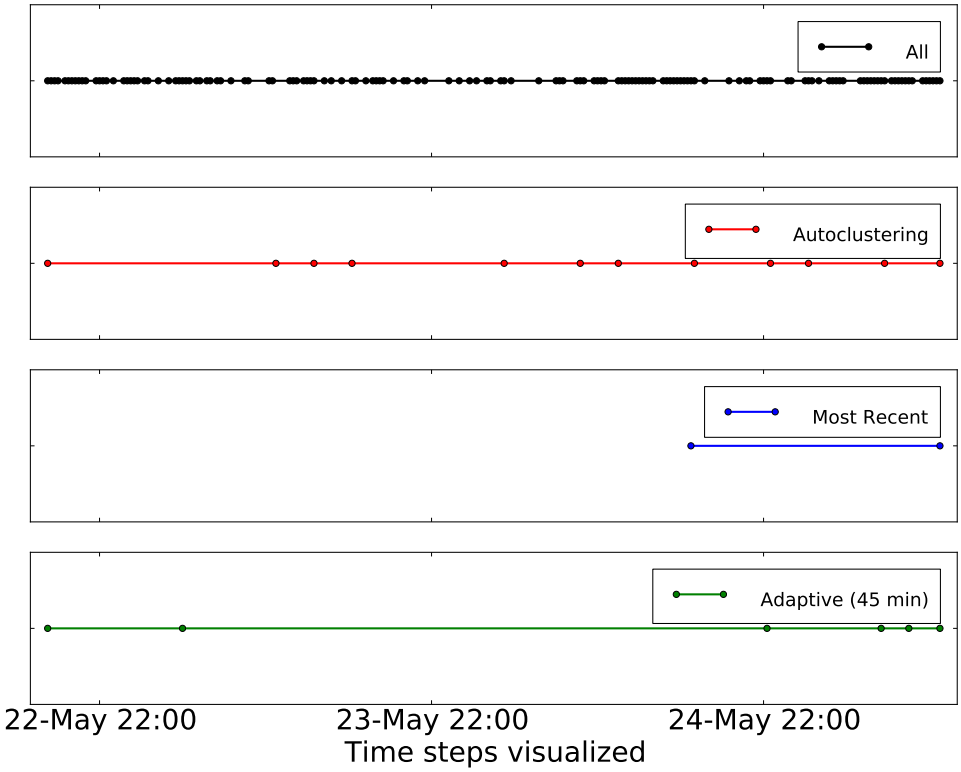


Figure 7.19: Nest position changes for low-bandwidth configuration.

7.5 Putting it all together

The InSt framework (refer Section 6.2) manages coordination between the frame selection algorithm that minimizes simulation-visualization lag, and the decision algorithm that determines the simulation parameters for given resource constraints. The frame selection algorithm discards some frames from the queue of pending frames and this results in change in available disk space. Due to this, there is slight reduction in the rate at which the disk space gets filled, which is taken into account in the optimization problem. The current available disk space is considered in every invocation of the decision algorithm to determine the output interval and the number of processors as discussed in Section 5.2.4. The decision algorithm is helpful in maintaining a steady simulation rate and continuous visualization considering the resource characteristics like I/O bandwidth,

network bandwidth and computation speed. Furthermore, the frame selection algorithm helps minimize the lag between the simulation and visualization times.

7.6 Summary

In this chapter, we presented an adaptive framework for efficient online remote visualization of critical weather applications like cyclone tracking. We described *most-recent*, *auto-clustering* and *adaptive* algorithms for frame selection at the simulation site in order to reduce the simulation-visualization lag. We showed that *most-recent* performs the best in terms of lag-reduction but it cannot ensure that representative frames are selected. The *adaptive* algorithm helps both in reducing the lag as well as improving the information content of the visualized frames. *Adaptive* algorithm adapts according to available network bandwidth, number of pending frames, and user-specified lag bound. *Auto-clustering* performs well in terms of sending representative frames from the simulation site and it also reduces the lag as compared to *all*. These strategies are quite generic and are applicable to other scientific domains.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

As we enter the exascale era, it is important to efficiently perform high-performance scientific simulations and visualization, despite the different growth rates in the computation speeds and the memory and network bandwidths. In this thesis, we explored few techniques for high-throughput simulations, online data analysis and visualization for weather applications. We performed extensive experiments using different resource configurations and for different weather events like cyclones, depressions and cloud formations.

We presented an adaptive integrated steering framework for simulation and visualization of critical applications like cyclone tracking across various resource configurations. We demonstrated that it is important to consider the resource constraints for continuous simulation and visualization across varying network bandwidths. A simple and intuitive greedy approach may lead to low throughput, stalling of simulation and disk overflow. Our framework adapts the simulation rate and the output interval based on the disk space and network speed constraints. In the process, the framework also considers the dynamics of the application and changing resource configurations. Our optimization method is able to provide about 30% higher simulation rate and consumes about 25-50% lesser storage space, and provides higher and more consistent rate of visualization

than a simple greedy approach. Our framework also enables scientists to input steering parameters like nest location and simulation resolution. Additionally the framework combines user-driven steering and automatic tuning and recommends optimal simulation parameters for smooth simulation and visualization.

We introduced algorithms for reducing delay between simulation and visualization times. We described most-recent, auto-clustering and adaptive algorithms for selecting frames to be sent from the simulation site to the visualization site. Most-recent strategy selects the most recent frame output by the simulation and hence performs the best in terms of reducing the delay. The auto-clustering algorithm selects a subset of significant frames for visualization. The adaptive algorithm selects a subset of significant frames and reduces the information content of selected frames depending on the simulation-visualization lag. We found that auto-clustering performs well in terms of sending representative frames from the simulation site and it also reduces the lag as compared to sending all frames. We also showed that the adaptive algorithm helps both in reducing the lag as well as selecting important frames. From experiments on different network configurations, we conclude that the adaptive algorithm strikes a good balance in reducing lag and visualization of most representative frames, with up to 72% smaller lag when compared to auto-clustering, and 37% larger representativeness than most-recent for slow networks. Hence, an adaptive algorithm is best suited for lag reduction as it is able to adapt to the information content in frames based on the current lag and the number of pending frames.

We presented a novel combination of performance modeling, processor allocation strategies, and topology-aware mapping heuristics to improve the performance of simulations with multiple high-resolution nested simulations. We showed that the performance of such weather simulations can be improved by allocating subsets of processors to each region of interest instead of the entire processor space. Our linear interpolation based performance prediction model predicts execution times with low error. Our processor partitioning strategy based on Huffman tree construction and recursive bisection outperforms a naïve proportional allocation by 8% with respect to the total execution

time. We developed 2D to 3D mapping heuristics that reduce communication times in the nested simulations as well as the parent simulation. Our experiments showed up to 33% improvement in performance with up to an additional 7% improvement with our topology-aware mapping heuristics. Our topology-oblivious and topology-aware mappings reduce the communication times by a maximum of 66%. Hence, we observe that significant improvements in performance are achieved with a different model of execution, such as simultaneous execution of nests in our case.

We presented a parallel data analysis algorithm that detects organized cloud systems using a variant of nearest neighbour clustering. We presented a tree-based processor reallocation algorithm for dynamic and concurrent weather events. Our tree-based diffusion algorithm considers the existing processor allocation and recommends a new subset of processors for multiple simultaneous simulations so that the data redistribution cost for the persistent nests is minimized. We achieved this by reorganization of the existing tree corresponding to the existing processor allocation. Experiments showed that we were able to reduce the redistribution times by up to 25% as compared to partitioning the processor space from scratch. We also developed a dynamic scheme that attempts to select the best of the two approaches, namely, partition from scratch and our hierarchical diffusion approach. Hence, it is important to design algorithms that consider distance between communicating processes and hence minimize the average number of hops between senders and receivers.

8.2 Future Work

The methodologies developed in this work are not tightly-coupled to weather simulations. The described techniques like detection of regions of interest and selection of representative frames are generic and can be extended to other applications with minimal effort. Our kmeans-based temporal clustering can be applied to scenarios where number of clusters are unknown and significant phases are required to be captured. We list below few possible extensions of our work.

- **Online data analysis and visualization**: In our work, we perform automatic tuning of I/O frequency depending on resource constraints. In addition to this, it may be worthwhile to do automatic tuning of parameters related to on-the-fly data analysis. Efficient large data analysis techniques can be employed to determine the usefulness of the current simulation time step, and hence reduce the number of time steps written to storage for visualization. Depending on the resource characteristics, it may not be possible to analyze every time step. An optimization approach, similar to ours, can be used to determine parameters like frequency of transferring time steps for analysis and frequency of storing time steps into secondary memory for visualization.

- **Communication-aware mapping for multiscale simulations**: Multiscale simulations often require all-to-all based communications. Using collective communications like all-to-all may incur high communication times due to network contention, specially in supercomputers with thousands of nodes. Hence, algorithmic redesign to communicate within a subset of nodes with efficient process-to-node mapping can improve performance of multiscale simulations. The topology-aware mappings discussed in this thesis are mainly for foldable mappings on a 3D torus. It will be challenging to explore extension of our techniques for optimizing overall communication times in nested simulations on different network topologies like 5D torus of Blue Gene/Q and for non-foldable mappings. Additionally, it will be worthwhile to explore contention optimization algorithms.

  In this thesis, we have experimented on IBM Blue Gene machines which allocate jobs on contiguous partitions. Hence the entire partition was considered as a single torus for mapping. Newer approaches will be required to extend our mapping strategies on systems with non-contiguous job allocations, such as the Cray supercomputers. Multiple disjoint sub-tori may have to be considered for mapping across the non-contiguous partitions in these systems.

- **Performance modeling for multiscale simulations**: Our performance model

and the processor allocation and reallocation strategies are applicable to nested simulations. It will be interesting to apply these techniques to other applications that involve multiscale simulations. The linear interpolation-based performance modeling approach described in this thesis requires profiling experiments for a few nest configurations. Future work may explore possibilities of combining analytical performance modeling and our approach to reduce the number of profiling experiments and give accurate predictions for nested simulation execution times.

- **Scalability prediction on exascale machines**: Petascale computing enables high fidelity simulations. However, it is also important to use an accurate performance model of the application to determine the appropriate number of processes to use for a given problem size. Using the maximum number of available processors may not always lead to satisfactory speedup as described in this thesis. This may be due to smaller problem size per process or increasing communication cost. However, redesigning the traditional methods used by domain scientists can boost performance on higher number of cores. It will be interesting to develop reliable performance models that can be useful in adaptively determining the appropriate number of cores and the algorithm to be used in the application prior to execution on large number of available cores.

- **Multi-site multiple events steering**: We performed single-site steering in our work. It will be more challenging to perform multi-site steering by different users, which will involve simultaneous inputs from different users. The framework can be extended to support various visualization requirements from different geographically distributed users. The framework has to reconcile automatic tuning decision with different inputs from different users. It will be even more challenging to perform steering for multiple event simulations from multiple visualization sites. The framework may need to deal with conflicting requirements from different users with respect to simulation parameters for the multiple events.

Bibliography

[1] Centre for Development of Advanced Computing. http://cdac.in/.

[2] Intel 64 Cluster Abe. http://www.ncsa.illinois.edu/UserInfo/Resources/Hardware/Intel64Cluster/.

[3] J. Ahrens, K. Heitmann, S. Habib, L. Ankeny, P. McCormick, J. Inman, R. Armstrong, and K.-L. Ma. Quantitative and Comparative Visualization applied to Cosmological Simulations. *Journal of Physics: Conference Series*, 46:526–534.

[4] Cyclone Aila. http://en.wikipedia.org/wiki/Cyclone_Aila.

[5] R Allan and A Mills. Survey of HPC Performance Modelling and Prediction Tools. *Science and Technology*, 2010.

[6] Rosa M. Badia, Francesc Escale, Edgar Gabriel, Judit Gimenez, Rainer Keller, Jesus Labarta, and Matthias S. Muller. Performance Prediction in a Grid Environment. In *Grid Computing*, volume 2970 of *Lecture Notes in Computer Science*, pages 257–264. 2004.

[7] K.J. Barker, Scott Pakin, and D.K. Kerbyson. A Performance Model of the Krak Hydrodynamics Application. In *International Conference on Parallel Processing*, pages 245–254, 2006.

[8] Bradley J. Barnes, Barry Rountree, David K. Lowenthal, Jaxk Reeves, Bronis de Supinski, and Martin Schulz. A Regression-based Approach to Scalability Prediction. In *Proceedings of the 22nd International Conference on Supercomputing*, ICS '08, pages 368–377, 2008.

[9] P. Bayrasy, M. Burger, C. Dehning, I. Kalmykov, and M. Speckert. Applications for MBS-FEM-coupling with MpCCI using automotive simulation as example. In

Proceedings of the 2nd Commercial Vehicle Technology Symposium (CVT 2012), pages 375–384, 2012.

[10] David M. Beazley and Peter S. Lomdahl. Lightweight Computational Steering of Very Large Scale Molecular Dynamics Simulations. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '96, 1996.

[11] Francine Berman. High-Performance Schedulers. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*, pages 279–309. Morgan Kaufmann, 1999.

[12] Abhinav Bhatele, Gagan Gupta, Laxmikant V. Kale, and I-Hsin Chung. Automated Mapping of Regular Communication Graphs on Mesh Interconnects. In *Proceedings of International Conference on High Performance Computing (HiPC)*, 2010.

[13] Abhinav Bhatele, Nikhil Jain, William D. Gropp, and Laxmikant V. Kale. Avoiding Hot-spots on Two-level Direct Networks. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, 2011.

[14] Abhinav Bhatele and Laxmikant V. Kale. Heuristic-Based Techniques for Mapping Irregular Communication Graphs to Mesh Topologies. In *13th IEEE International Conference on High Performance Computing & Communication*, pages 765–771, 2011.

[15] John Brooke, Thomas Eickermann, and Uwe Woessner. Application Steering in a Collaborative Environment. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '03, 2003.

[16] Junwei Cao, D.K. Kerbyson, E. Papaefstathiou, and Graham R. Nudd. Performance Modeling of Parallel and Distributed Computing using PACE. In *Performance, Computing, and Communications Conference, 2000. IPCCC '00. Conference Proceeding of the IEEE International*, pages 485–492, 2000.

[17] Henry R. Childs, Eric Brugger, Kathleen S. Bonnell, Jeremy S. Meredith, Mark

Miller, Brad Whitlock, and Nelson Max. A Contract Based System For Large Data Visualization. In *IEEE Visualization*, 2005.

[18] I-Hsin Chung, Robert Walkup, Hui-Fang Wen, and Hao Yu. MPI Performance Analysis Tools on Blue Gene/L. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '06, 2006.

[19] I-Hsin Chung, Robert E. Walkup, Hui-Fang Wen, and Hao Yu. MPI Performance Analysis Tools on Blue Gene/L. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '06, 2006.

[20] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2 edition, 2001.

[21] H. S. M. Coxeter. Barycentric Coordinates. In *Introduction to Geometry*, pages 216–221. Wiley, 2nd edition, 1969.

[22] Boris N. Delaunay. Sur la sphere vide. *Bulletin of Academy of Sciences of the USSR*, 6:793–800, 1934.

[23] J. Delgado, S.M. Sadjadi, M. Bright, M. Adjouadi, and H.A. Duran-Limon. Performance Prediction of Weather Forecasting Software on Multicore Systems. In *IPDPS, Workshops and PhD Forum*, 2010.

[24] P.A. Dinda. Online Prediction of the Running Time of Tasks. In *Proceedings of the 10^{th} IEEE International Symposium on High Performance Distributed Computing*, pages 383–394, 2001.

[25] J. Dongarra, P. Beckman, and T. Moore. International Exascale Software Project Roadmap. Technical report, DOE and NSF, Nov 2009.

[26] Isaac Dooley and Laxmikant V. Kale. Control Points for Adaptive Parallel Performance Tuning. *PPL Technical Report*, 2008.

[27] P.-F. Dutot, T. N'Takpe, F. Suter, and H. Casanova. Scheduling Parallel Task Graphs on (Almost) Homogeneous Multicluster Platforms. *IEEE Transactions on Parallel and Distributed Systems*, 20(7):940–952, 2009.

[28] David Ellsworth, Bryan Green, Chris Henze, Patrick Moran, and Timothy Sandstrom. Concurrent Visualization in a Production Supercomputing Environment.

IEEE Transactions on Visualization and Computer Graphics, 12(5):997–1004, 2006.

[29] F. Ercal, J. Ramanujam, and P. Sadayappan. Task Allocation onto a Hypercube by Recursive Mincut Bipartitioning. In *Proceedings of the third conference on Hypercube concurrent computers and applications: Architecture, software, computer systems, and general issues - Volume 1*, C3P, pages 210–221, 1988.

[30] N. Fabian, K. Moreland, D. Thompson, A.C. Bauer, P. Marion, B. Gevecik, M. Rasquin, and K.E. Jansen. The ParaView Coprocessing Library: A Scalable, General Purpose In Situ Visualization Library. In *IEEE Symposium on Large Data Analysis and Visualization*, 2011.

[31] NCEP Final Analyses. http://www.mmm.ucar.edu/wrf/OnLineTutorial/DATA/FNL/.

[32] Takashi Furumura and Li Chen. Large Scale Parallel Simulation and Visualization of 3D Seismic Wave Field Using the Earth Simulator. *Journal of Computer Modeling in Engineering & Sciences*, 6(2):153–168, 2004.

[33] G. A. Geist, J. A. Kohl, , and P. M Papadopoulos. CUMULVS: Providing Fault-Tolerance, Visualization and Steering of Parallel Applications. *International Journal of High Performance Computing Applications*, 11:224–236, 1996.

[34] GNU Linear Programming Kit. http://www.gnu.org/software/glpk.

[35] Guojun Gu and Chidong Zhang. Cloud components of the Intertropical Convergence Zone. *Journal of Geophysical Research: Atmospheres*, 107(D21):ACL 4–1–ACL 4–12, 2002.

[36] Weiming Gu, G. Eisenhauer, E. Kraemer, K. Schwan, J. Stasko, J. Vetter, and N. Mallavarupu. Falcon: On-line Monitoring and Steering of Large-scale Parallel Programs. *Proceedings of the Fifth Symposium on the Frontiers of Massively Parallel Computation*, pages 422–429, 1995.

[37] Weiming Gu, Jeffrey Vetter, and Karsten Schwan. An Annotated Bibliography of Interactive Program Steering. *ACM SIGPLAN Notices*, 1994.

[38] Charles D. Hansen and Chris R. Johnson. *The Visualization Handbook*. Academic

Press, 2005.

[39] Torsten Hoefler and Marc Snir. Generic Topology Mapping Strategies for Large-scale Parallel Architectures. In *Proceedings of the International Conference on Supercomputing*, ICS '11, pages 75–84, 2011.

[40] Gene Hou, Jin Wang, and Anita Layton. Numerical Methods for Fluid-Structure Interaction - A Review. *Communications in Computational Physics*, 12(2):337–377.

[41] Y.F. Hu and R.J. Blake. An Improved Diffusion Algorithm for Dynamic Load Balancing. *Parallel Computing*, 25(4):417–444, 1999.

[42] B. Huang1, D. Xiong, and H. Li. An Integrated Approach to Real-time Environmental Simulation and Visualization. *Journal of Environmental Informatics*, 3(1):42–50, 2004.

[43] James Hurrell and et al. The Community Earth System Model: A Framework for Collaborative Research. *Bulletin of the American Meteorological Society*, 94(9):1339–1360, 2013.

[44] Joseph A. Insley, Michael E. Papka, Suchuan Dong, George Karniadakis, and Nicholas T. Karonis. Runtime Visualization of the Human Arterial Tree. *IEEE Transactions on Visualization and Computer Graphics*, 13(4), 2007.

[45] Intergovernmental Panel on Climate Change. http://ipcc.ch/.

[46] Integrated Performance Monitoring. http://ipm-hpc.sourceforge.net.

[47] D.J. Jablonowski, J.D. Bruner, B. Bliss, and R.B. Haber. VASE: The Visualization and Application Steering Environment. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '93, 1993.

[48] Raj Jain. *Art of Computer Systems Performance Analysis Techniques For Experimental Design Measurements Simulation And Modeling*. Wiley Computer Publishing, John Wiley & Sons, Inc., 1991.

[49] Stephen A. Jarvis, Daniel P. Spooner, Helene N. Lim Choi Keung, Junwei Cao, Subhash Saini, and Graham R. Nudd. Performance Prediction and its use in Parallel and Distributed Computing Systems. *Future Generation Computer Systems*,

22(7):745–754, 2006.

[50] Y. Jean, T. Kindler, W. Ribarsky, Weiming Gu, G. Eisenhauer, K. Schwan, and F. Alyea. Case Study: An Integrated Approach for Steering, Visualization, and Analysis of Atmospheric Simulations. In *Proceedings of IEEE Visualization '95*, 1995.

[51] C. Ryan Johnson, Markus Glatter, Wesley Kendall, Jian Huang, and Forrest Hoffman. Querying for Feature Extraction and Visualization in Climate Modeling. In *Proceedings of the 9th International Conference on Computational Science*, ICCS 2009, pages 416–425, 2009.

[52] Christopher R. Johnson and Steven G. Parker. A Computational Steering Model Applied to Problems in Medicine. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '94, 1994.

[53] William L. Jorgensen. The Many Roles of Computation in Drug Discovery. *Science*, 303(5665):1813–1818, 2004.

[54] Gregory Karagiorgos, Nikolaos M. Missirlis, and Filippos Tzaferis. Fast Diffusion Load Balancing Algorithms on Torus Graphs. In *Euro-Par '06, 12th International Euro-Par Conference*, 2006.

[55] George Karypis and Vipin Kumar. Multilevel k-way Partitioning Scheme for Irregular Graphs. *Journal of Parallel Distributed Computing*, 48(1):96–129, 1998.

[56] Johannes Kehrer, Florian Ladstadter, Philipp Muigg, Helmut Doleisch, Andrea Steiner, and Helwig Hauser. Hypothesis Generation in Climate Research with Interactive Visual Data Exploration. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1579–1586, 2008.

[57] Wesley Kendall, Markus Glatter, Jian Huang, Tom Peterka, Robert Latham, and Robert Ross. Terascale Data Organization for Discovering Multivariate Climatic Trends. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '09, 2009.

[58] Darren J. Kerbyson, Kevin J. Barker, and Kei Davis. Analysis of the Weather Research and Forecasting (WRF) Model on Large-Scale Systems. In *PARCO*,

pages 89–98, 2007.

[59] James A. Kohl, Torsten Wilde, and David E. Bernholdt. Cumulvs: Interacting with High-Performance Scientific Simulations, for Visualization, Steering and Fault Tolerance. *International Journal of High Performance Computing Applications*, 2006.

[60] Kraken Cray XT5, NICS, Tennessee. http://www.nics.tennessee.edu/computing-resources/kraken.

[61] Sameer Kumar, Yogish Sabharwal, Rahul Garg, and Philip Heidelberger. Optimization of All-to-all Communication on the Blue Gene/L Supercomputer. In *Proceedings of the International Conference on Parallel Processing*, 2008.

[62] Marc Labadens, Damien Chapon, Daniel Pomarede, and Romain Teyssier. Visualization of Octree Adaptive Mesh Refinement (AMR) in Astrophysical Simulations. In *Proceedings of the Astronomical Data Analysis Software and Systems (ADASS) XXI*, 2011.

[63] Guan-Joe Lai. A Communicative-aware Task Scheduling Algorithm for Heterogeneous Systems. In *Proceedings of the 14th International Workshop on Database and Expert Systems Applications*, pages 161–166, 2003.

[64] LAMMPS Molecular Dynamics Simulator. http://lammps.sandia.gov.

[65] Zhiling Lan, Valerie E. Taylor, and Greg Bryan. Dynamic Load Balancing of SAMR applications on Distributed Systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '01, 2001.

[66] Samuel Lang, Philip H. Carns, Robert Latham, Robert B. Ross, Kevin Harms, and William E. Allcock. I/O Performance Challenges at Leadership Scale. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '09, 2009.

[67] Averill M. Law. *Simulation Modeling and Analysis*. McGraw-Hill, fourth edition, 2007.

[68] C.E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901, Oct 1985.

[69] Jianwei Li, Wei-keng Liao, Alok Choudhary, Robert Ross, Rajeev Thakur, William Gropp, Rob Latham, Andrew Siegel, Brad Gallagher, and Michael Zingale. Parallel netCDF: A High-Performance Scientific I/O Interface. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '03, 2003.

[70] A. Lingas, R.Y. Pinter, R.L. Rivest, and A. Shamir. Minimum Edge Length Rectilinear Decompositions of Rectilinear Figures. In *20^{th} Allerton Conference on Communication, Control, and Computing*, pages 53–63, 1982.

[71] Hua Liu, Lian Jiang, Manish Parashar, and Deborah Silver. Rule-based Visualization in the Discover Computational Steering Collaboratory. *Future Generation Computer Systems*, 21(1):53–59, 2005.

[72] Tianming Liu, Hong-Jiang Zhang, and Feihu Qi. A Novel Video Key-frame-extraction Algorithm based on Perceived Motion Energy Model. *IEEE Transactions on Circuits and Systems for Video Technology*, 2003.

[73] Jiebo Luo, C. Papin, and K. Costello. Towards Extracting Semantically Meaningful Key Frames From Personal Video Clips: From Humans to Computers. In *IEEE Transactions on Circuits and Systems for Video Technology*, pages 289–301, 2009.

[74] E. Luque, A. Ripoll, A. Cortes, and T. Margalef. A Distributed Diffusion Method for Dynamic Load Balancing on Parallel Computers. In *Proceedings of the Euromicro Workshop on Parallel and Distributed Processing*, pages 43–50, 1995.

[75] K.-L. Ma, C. Wang, H. Yu, and A. Tikhonova. In Situ Processing and Visualization for Ultrascale Simulations. *Journal of Physics (Proceedings of SciDAC 2007 Conference)*, 78, 2007.

[76] Kwan-Liu Ma. In Situ Visualization at Extreme Scale: Challenges and Opportunities. *IEEE Computer Graphics and Applications*, 29(6):14–19, 2009.

[77] J. B. MacQueen. Some Methods for Classification and Analysis of MultiVariate Observations. In *Proceedings of the fifth Berkeley Symposium on Mathematical*

Statistics and Probability, volume 1, pages 281–297, 1967.

[78] Henning Meyerhenke, Burkhard Monien, and Stefan Schamberger. Graph partitioning and disturbed diffusion. *Parallel Computing*, 35(10–11):544–569, 2009.

[79] J. Michalakes, J. Dudhia, D. Gill, T. Henderson, J. Klemp, W. C. Skamarock, and W. Wang. The Weather Research and Forecast Model: Software Architecture and Performance. In *Proceedings of the 11th ECMWF Workshop on the Use of High Performance Computing In Meteorology*, Oct 2004.

[80] J. Michalakes, Josh Hacker, Richard Loft, Michael O. McCracken, A. Snavely, N.J. Wright, Tom Spelce, Brent Gorda, and R. Walkup. WRF Nature Run. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '07, 2007.

[81] John Michalakes. RSL: A Parallel Runtime System Library For Regional Atmospheric Models With Nesting. Technical Report ANL/MCS-TM-197, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, 1997.

[82] Charles Moad and Beth Plale. Portal Access to Parallel Visualization of Scientific Data on the Grid. Technical Report TR593, Computer Science Department, Indiana University, February 2004. http://www.cs.indiana.edu/pub/techreports/TR593.pdf.

[83] Anirudh Modi, Lyle N. Long, and Paul E. Plassmann. Real-time Visualization of Wake-vortex Simulations using Computational Steering and Beowulf Clusters. In *VECPAR'02: Proceedings of the 5th International conference on High Performance Computing for Computational Science*, 2002.

[84] Irene Moulitsas and George Karypis. Architecture Aware Partitioning Algorithms. In *International Conference on Algorithms and Architectures for Parallel Processing*, pages 42–53, 2008.

[85] National Knowledge Network, Department of Information Technology, Government of India. http://www.mit.gov.in/content/national-knowledge-network.

[86] G. Nudd, D. Kerbyson, E. Papaefstathiou, S. Perry, J. Harper, and D. Wilcox. PACE: A Toolset for the Performance Prediction of Parallel and Distributed

Systems. *International Journal of High Performance Computing Applications*, 14(3):228–251, 2000.

[87] Leonid Oliker and Rupak Biswas. Efficient Load Balancing and Data Remapping for Adaptive Grid Calculations. In *Proceedings of the ninth annual ACM Symposium on Parallel Algorithms and Architectures*, pages 33–42, 1997.

[88] J.M. Orduna, V. Arnau, A. Ruiz, R. Valero, and J. Duato. On the design of communication-aware task scheduling strategies for heterogeneous systems. In *International Conference on Parallel Processing*, pages 391–398, 2000.

[89] J.M. Orduna, F. Silla, and J. Duato. A New Task Mapping Technique for Communication-aware Scheduling Strategies. In *International Conference on Parallel Processing Workshops*, pages 349–354, 2001.

[90] A. Ortega and K. Ramchandran. Rate-distortion Methods for Image and Video Compression. *Signal Processing Magazine, IEEE*, 15(6):23–50, Nov 1998.

[91] T. Ozcelebi, A.M. Tekalp, and M.R. Civanlar. Delay-Distortion Optimization for Content-Adaptive Video Streaming. *IEEE Transactions on Multimedia*, 9(4):826–836, June 2007.

[92] Steven G. Parker, Charles D. Hansen, Christopher R. Johnson, and Michelle Miller. Computational Steering and the SCIRun Integrated Problem Solving Environment. *Scientific Visualization Conference*, 1997.

[93] Steven G. Parker and Christopher R. Johnson. SCIRun: A Scientific Programming Environment for Computational Steering. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '95, 1995.

[94] Steven G. Parker, Christopher R. Johnson, and David Beazley. Computational Steering Software Systems and Strategies. *Computational Science and Engineering, IEEE*, 1997.

[95] Steven G. Parker, Michelle Miller, Charles D. Hansen, and Christopher R. Johnson. An Integrated Problem Solving Environment: the SCIRun Computational Steering System. *Proceedings of the Thirty-First Hawaii International Conference*

on System Sciences, 7:147–156, Jan 1998.

[96] Robert Patro, Cheuk Yiu Ip, and Amitabh Varshney. Saliency Guided Summarization of Molecular Dynamics Simulations. In *Scientific Visualization: Advanced Concepts*, pages 321–335, 2010.

[97] A. R. Porter, M. Ashworth, A. Gadian, R. Burton, P. Connolly, and M. Bane. WRF code Optimisation for Meso-scale Process Studies (WOMPS) dCSE Project Report. Jun 2010.

[98] UCAR CISL Research Data Archive. http://rda.ucar.edu.

[99] Daniel A. Reed, Christopher L. Elford, Tara M. Madhyastha, Evgenia Smirni, and Stephen E. Lamm. The Next Frontier: Interactive and Closed Loop Performance Steering. In *ICPP Workshop*, pages 20–31, 1996.

[100] R. Rew and G. Davis. The Unidata netCDF: Software for Scientific Data Access. In *6^{th} International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology, California, American Meteorology Society*, 1990.

[101] Randy L. Ribler, Huseyin Simitci, and Daniel A. Reed. The Autopilot Performance-directed Adaptive Control System. *Future Generation Computer Systems*, 2001.

[102] R.L. Ribler, J.S. Vetter, H. Simitci, and D.A. Reed. Autopilot: Adaptive Control of Distributed Applications. In *Proceedings of the Seventh International Symposium on High Performance Distributed Computing*, Jul 1998.

[103] M. Riedel, T. Eickermann, W. Frings, S. Dominiczak, D. Mallmann, T. Dussel, A. Streit, P. Gibbon, F. Wolf, W. Schiffmann, and T. Lippert. Design and Evaluation of a Collaborative Online Visualization and Steering Framework Implementation for Computational Grids. In *8^{th} IEEE/ACM International Conference on Grid Computing*, pages 169–176, 2007.

[104] Badia R.M., Labarta J., Gimenez J, and Escale F. DIMEMAS: Predicting MPI Applications Behavior in Grid Environments. In *Workshop on Grid Applications and Programming Tools (GGF8)*, 2003.

[105] Robert Sisneros and Markus Glatter and Brandon Langley and Jian Huang and

Forrest Hoffman and David Erickson III. Time-Varying Multivariate Visualization for Understanding Terrestrial Biogeochemistry. *Journal of Physics: Conference Series (SciDAC 08)*, 125(1), July 2008.

[106] C. Roig, A. Ripoll, and F. Guirado. A New Task Graph Model for Mapping Message Passing Applications. *IEEE Transactions on Parallel and Distributed Systems*, 18(12):1740–1753, 2007.

[107] Regional Ocean Modeling System. http://www.myroms.org.

[108] D. Rosenfeld. and I. M. Lensky. Satellite-based Insights into Precipitation Formation Processes in Continental and Maritime Convective Clouds. *Bulletin of the American Meteorological Society*, 79:2457–2476, 1998.

[109] Seyed Masoud Sadjadi, Shu Shimizu, Javier Figueroa, Raju Rangaswami, Javier Delgado, Hector A. Duran, and Xabriel J. Collazo-Mojica. A Modeling Approach for Estimating Execution Time of Long-running Scientific Applications. In *IPDPS, Fifth High-Performance Grid Computing Workshop*, 2008.

[110] Hans Sagan. *Space-Filling Curves*. Springer-Verlag, 1994.

[111] Sandeep Sahany, V. Venugopal, and Ravi S. Nanjundiah. The 26 July 2005 Heavy Rainfall Event over Mumbai: Numerical Modeling Aspects. *Meteorology and Atmospheric Physics*, 109:115–128, 2010.

[112] Kirk Schloegel, George Karypis, and Vipin Kumar. Multilevel Diffusion Schemes for Repartitioning of Adaptive Meshes. *Journal of Parallel and Distributed Computing*, 47:109–124, 1997.

[113] William J. Schroeder, Kenneth M. Martin, and William E. Lorensen. *The Visualization Toolkit*. Kitware Inc., third edition, Aug 2004.

[114] Gilad Shainer and Jacob Liberman. Weather Research and Forecast (WRF) Model Performance and Profiling Analysis on Advanced Multi-core HPC Clusters. In *10^{th} LCI International Conference on High-Performance Clustered Computing*, 2009.

[115] Han-Wei Shen and Christopher R. Johnson. Differential Volume Rendering: A Fast Volume Visualization Technique for Flow Animation. In *Proceedings of the conference on Visualization '94*, pages 180–187, 1994.

[116] Alex Shenfield, Peter J. Fleming, and Muhammad Alkarouri. Computational Steering of a Multi-objective Evolutionary Algorithm for Engineering Design. *Engineering Applications of Artificial Intelligence*, 2007.

[117] Horst D. Simon and Shang-Hua Teng. How Good is Recursive Bisection? *SIAM Journal on Scientific Computing*, 18(5):1436–1445, 1997.

[118] Shweta Sinha and Manish Parashar. Adaptive System Sensitive Partitioning of AMR Applications on Heterogeneous Clusters. *Cluster Computing*, 5:343–352, 2002.

[119] Oliver Sinnen and Leonel A. Sousa. Communication Contention in Task Scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 16(6):503–515, Jun 2005.

[120] Oliver Sinnen, Leonel Augusto Sousa, and Frode Eika Sandnes. Toward a Realistic Task Scheduling Model. *IEEE Transactions on Parallel and Distributed Systems*, 17(3):263–275, Mar 2006.

[121] W. C. Skamarock, J. B. Klemp, J. Dudhia, D. O. Gill, D. M. Barker, M. Duda, X.-Y. Huang, W. Wang, and J. G. Powers. A Description of the Advanced Research WRF version 3. *NCAR Technical Note TN-475*, 2008.

[122] Edward Smith, David Trevelyan, and Tamer A. Zaki. Scalable coupling of Molecular Dynamics and Direct Numerical Simulation of multi-scale flows. Technical report, Mechanical Engineering, Imperial College London, 2013. http://www.hector.ac.uk/cse/distributedcse/reports/transflow02/.

[123] T. Takei, J. Bernsdorf, N. Masuda, and H. Takahara. Lattice Boltzmann Simulation and Its Concurrent Visualization on the SX-6 Supercomputer. In *Proceedings of the Seventh International Conference on High Performance Computing and Grid in Asia Pacific Region*, pages 212–219, 2004.

[124] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., 2005.

[125] C. Tapus, I-Hsin Chung, and J.K. Hollingsworth. Active Harmony: Towards Automated Performance Tuning. In *Proceedings of the International Conference for*

High Performance Computing, Networking, Storage and Analysis, SC '02, 2002.

[126] IBM Blue Gene Team. Overview of the Blue Gene/L System Architecture. *IBM Journal of Research and Development*, 49, 2005.

[127] IBM Blue Gene Team. Overview of the Blue Gene/P Project. *IBM Journal of Research and Development*, 52, 2008.

[128] Top 500 Supercomputing Sites. http://www.top500.org.

[129] T. Tu, H. Yu, L. Ramirez-Guzman, J. Bielak, O. Ghattas, K.-L. Ma, and D. O'Hallaron. From Mesh Generation to Scientific Visualization: an End-to-End Approach to Parallel Supercomputing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '06, 2006.

[130] Rob F. Van der Wijngaart, Srinivas Sridharan, and Victor W. Lee. Extending the BT NAS Parallel Benchmark to Exascale Computing. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, 2012.

[131] Jeffrey S. Vetter and Karsten Schwan. High Performance Computational Steering of Physical Simulations. In *IPPS '97: Proceedings of the 11th International Symposium on Parallel Processing*, 1997.

[132] V. Vishwanath, M. Hereld, and M. E. Papka. Toward Simulation-Time Data Analysis and I/O Acceleration on Leadership-Class Systems. In *Proceedings of IEEE Symposium on Large-Scale Data Analysis and Visualization*, pages 9–14, 2011.

[133] N. Vydyanathan, S. Krishnamoorthy, G.M. Sabin, U.V. Catalyurek, T. Kurc, P. Sadayappan, and J.H. Saltz. An Integrated Approach to Locality-Conscious Processor Allocation and Scheduling of Mixed-Parallel Applications. *IEEE Transactions on Parallel and Distributed Systems*, 20(8):1158–1172, 2009.

[134] Rick Walker, Peter Kenny, and Jingqi Miao. Exploratory Simulation for Astrophysics. *Proceedings of the SPIE, Conference on Visualization and Data Analysis*, 6495, 2007.

[135] Chaoli Wang, Hongfeng Yu, and Kwan-Liu Ma. Importance-Driven Time-Varying

Data Visualization. In *IEEE Transactions on Visualization and Computer Graphics*, 2008.

[136] Brad Whitlock, Jean M. Favre, and Jeremy S. Meredith. Parallel In Situ Coupling of Simulation with a Fully Featured Visualization System. In *Eurographics Symposium on Parallel Graphics and Visualization*, 2011.

[137] Nancy Wilkins-Diehr, Dennis Gannon, Gerhard Klimeck, Scott Oster, and Sudhakar Pamidighantam. TeraGrid Science Gateways and Their Impact on Science. *Computer*, 41(11):32–41, Nov 2008. https://portal.xsede.org/.

[138] Bryan Worthen, Thomas C. Henderson, Justin Luitjens, and Martin Berzins. Scalable Parallel AMR for the Uintah Multi-Physics Code. In *Petascale Computing Algorithms and Applications*, pages 67–82. Chapman and Hall/CRC, 2008.

[139] H. Wright, R. H. Crompton, S. Kharche, and P. Wenisch. Steering and Visualization: Enabling Technologies for Computational Science. *Future Generation Computer Systems*, 26(3):506–513, 2010.

[140] Nicholas J. Wright, Wayne Pfeiffer, and Allan Snavely. Characterizing Parallel Scaling of Scientific Applications using IPM. In *10^{th} LCI International Conference on High-Performance Clustered Computing*, 2009.

[141] Qishi Wu, Jinzhu Gao, Mengxia Zhu, N.S.V. Rao, Jian Huang, and S.S. Iyengar. Self-Adaptive Configuration of Visualization Pipeline Over Wide-Area Networks. In *IEEE Transactions on Computers*, pages 55–68, 2008.

[142] Qishi Wu, Mengxia Zhu, Yi Gu, and Nageswara S. V. Rao. System Design and Algorithmic Development for Computational Steering in Distributed Environments. *IEEE Transactions on Parallel and Distributed Systems*, 21(4), 2010.

[143] X.-P. Xu and A. Needleman. Numerical Simulations of Fast Crack Growth in Brittle Solids. *Journal of the Mechanics and Physics of Solids*, 42(9):1397–1434, 1994.

[144] H. Yu, R. K. Sahoo, C. Howson, G. Almasi, J. G. Castanos, M. Gupta, J. E. Moreira, and J. J. Parker. High Performance File I/O for The Blue Gene/L Supercomputer. In *Proceedings of the 12^{th} International Symposium on High-Performance*

Computer Architecture, 2006.

[145] Hao Yu, I-Hsin Chung, and J. Moreira. Topology Mapping for Blue Gene/L Supercomputer. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '06, 2006.

[146] Hongfeng Yu, Kwan-Liu Ma, and Joel Welling. A Parallel Visualization Pipeline for Terascale Earthquake Simulations. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '04, 2004.

[147] Florence Zara, François Faure, and Vincent Jean-Marc. Parallel Simulation of Large Dynamic System on a PCs Cluster: Application to Cloth Simulation. *International Journal of Computers and Applications*, 26(3), Mar 2004.

[148] Jidong Zhai, Wenguang Chen, and Weimin Zheng. PHANTOM: Predicting Performance of Parallel Applications on Large-scale Parallel Machines using a Single Node. In *Proceedings of the 15^{th} ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '10, pages 305–314, 2010.

[149] Xu-Dong Zhang, Tie-Yan Liu, Kwok-Tung Lo, and Jian Feng. Dynamic Selection and Effective Compression of Key Frames for Video Abstraction. *Pattern Recognition Letters*, 24:1523–1532, Jun 2003.

[150] Elena V. Zudilova. Simulation-Visualization Complexes as Generic Exploration Environment. In *ICCS '01: Proceedings of the International Conference on Computational Science*, volume 2074 of *Lecture Notes in Computer Science*, pages 903–911, 2001.