A Divide and Conquer Strategy for Scaling Weather Simulations with Multiple Regions of Interest

Preeti Malakar*, Thomas George‡, Sameer Kumar§, Rashmi Mittal‡, Vijay Natarajan*†,
Yogish Sabharwal‡, Vaibhav Saxena‡, Sathish S. Vadhiyar†
*Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India
†Supercomputer Education and Research Centre, Indian Institute of Science, Bangalore, India
‡IBM India Research Lab, New Delhi, India
§IBM T.J. Watson Research Center, Yorktown Heights, NY, USA

***Abstract*—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. Current weather applications execute these nested simulations sequentially using all the available processors, which is sub-optimal due to their sub-linear scalability. In this work, we present a strategy for parallel execution of multiple nested domain simulations based on partitioning the 2-D processor grid into disjoint rectangular regions associated with each domain. We propose a novel combination of performance prediction, processor allocation methods and topology-aware mapping of the regions on torus interconnects. Experiments on IBM Blue Gene systems using WRF show that the proposed strategies result in performance improvement of up to 33% with topology-oblivious mapping and up to additional 7% with topology-aware mapping over the default sequential strategy.**

***Index Terms*—weather simulation; performance modeling; processor allocation; topology-aware mapping;**

I. 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 on-line visualization to comprehend the simulation output on-the-fly. Weather simulations mainly comprise of solving non-linear partial differential equations numerically. Ongoing 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.

Simulating and tracking multiple regions of interest at fine resolutions is important in understanding the interplay between multiple weather phenomena and for comprehensive predictions. For example, Figure 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.

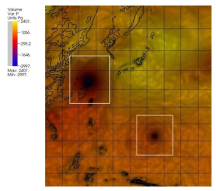


Fig. 1. Visualization of multiple depressions in August 2010 on Pacific Ocean

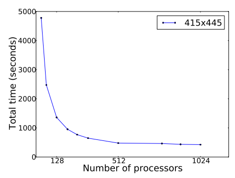


Fig. 2. Execution time of a weather simulation over a 10^7 sq. kms. domain on Blue Gene/L

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 a large amount 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. Typically, these I/O costs constitute a substantial fraction (20-40%) of the total simulation time. Thus, reducing the I/O costs can also improve the overall performance.

Existing weather applications employ a default strategy of executing the nested simulations corresponding to a single parent domain sequentially one after the other using the full set of 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 the popular Weather Research and Forecasting model (WRF) [1], [2] is scalable up to large number of cores [3] when executed without

This research is partly supported by Centre for Development of Advanced Computing, India, project ref. no. CDAC/ESE/STV/C014.

a subdomain, but exhibits poor scalability when executed with subdomains. Figure 2 shows the scalability of WRF on a rack of IBM Blue Gene/L. The simulation corresponded to a region with parent domain of size 286×307 and involving a subdomain of size 415×445. 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 equally among the subdomains for simultaneous execution will give better performance than serial execution on all the 1024 cores.

We focus on optimizing the parallel execution of high-resolution nested simulations so as 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. This 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. 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. We estimate these relative execution times 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 2011 list based on torus network topology [4]. 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.

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.

To summarize, following are our primary contributions.

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.

II. Related Work

Our work is primarily related to three research areas: (1) performance modeling and prediction, (2) static load balancing for parallel applications, and (3) mapping of processes in the virtual topology onto the physical network topology.

Performance Modeling and Prediction. There has been a lot of research on performance modeling and prediction of applications running on HPC systems. Allan et al. [5] compare tools for predicting performance on a range of architectures and applications. Due to the rich history of this field, we only focus on prior work involving weather forecasting applications. Kerbyson et al. [6] 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 real systems - an AMD Opteron based cluster system and IBM Blue Gene/L. Unlike [6], 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. Delgado et al. [7] (extending their earlier work in [8]) 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. Our prediction approach makes uses of the application specific factors (i.e. grid size and aspect ratio) without explicitly incorporating system related factors. Further, [6] and [7] 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 execution of nested domains.

Static Load Balancing for Parallel Applications. Parallelization of scientific applications involving numerical modeling is a well-studied area with vast amount of literature. Most of the earlier existing work on static data parallel simulations such as those in weather modeling applications is based on domain decomposition where the domain of interest is divided into smaller subdomains that are assigned to individual processors such that the load is balanced across the processors while minimizing the communication costs. There currently exists a number of approaches such as recursive bisection [9] and graph partitioning [10] that cater to both regular and irregular domains and yield very good performance. In recent years, there has also been work [11] addressing 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 while taking into account other constraints. In our current work, application-specific constraints require that a single nested domain is assigned to a rectangular processor grid to achieve optimal performance. We employ an alternate strategy of partitioning 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. Interestingly, our algorithm for partitioning the processor space is based on recursive bisection that is often used for decomposing regular 2D domains.

Mapping. Past work [12]–[16] 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], [13] 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 application 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 application simulations, multiple sibling domains running simultaneously present a harder problem. This is because we need to optimize the communication in the main domain as well as the multiple nested domains. In this paper, we present mapping optimizations for WRF that selectively map each subdomain to a sub-rectangle of the torus, while also keeping the number of hops minimal for the subdomain and parent domain processors.

III. Parallelization of Subdomains

In simulations involving nested domains, the simulation of the high-resolution nests are highly 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 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 III-A. The performance prediction of the simulation times is then used for partitioning the available processor space for the nested simulations as described in Section III-B.

The simulations of the high-resolution nests 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 III-C.

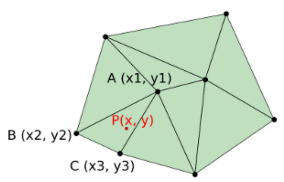
A. Performance Prediction

We propose a performance model that predicts the relative execution times of the nested simulations with a low error. 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

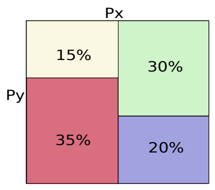
1) Total number of points in the domain, given by $n_x \cdot n_y$.
2) Aspect ratio of the domain, given by n_x / n_y.

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 total number of points in the domain and the y-coordinate denotes the aspect ratio. The convex hull of these 13 points is triangulated using Delaunay triangulation [17]. Figure 3(a) shows a snapshot of the triangulation. The vertices of 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(a). 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 [18] of P are obtained from A, B and C by Equations (1), (2) and (3). The predicted execution time $\mathcal{T}_D$ of P can be interpolated from the execution times of the domains represented by the vertices of $\triangle ABC$



(a) Delaunay Triangulation of points representing known execution times



(b) Partitions of processor space in the ratio of execution times of nested simulations

Fig. 3.

as shown in Equation (4).

$$\lambda_1 = \frac{(y_2-y_3)*(x-x_3)+(x_3-x_2)*(y-y_3)}{(y_2-y_3)*(x_1-x_3)+(x_3-x_2)*(y_1-y_3)} \quad (1)$$

$$\lambda_2 = \frac{(y_3-y_1)*(x-x_3)+(x_1-x_3)*(y-y_3)}{(y_2-y_3)*(x_1-x_3)+(x_3-x_2)*(y_1-y_3)} \quad (2)$$

$$\lambda_3 = \lambda_1 - \lambda_2 \quad (3)$$

$$\mathcal{T}_D = \lambda_1 * \mathcal{T}_{(x_1,y_1)} + \lambda_2 * \mathcal{T}_{(x_2,y_2)} + \lambda_3 * \mathcal{T}_{(x_3,y_3)} \quad (4)$$

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 94x124 to 415x445 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 that point. 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.

Our performance model based on Delaunay triangulation can be very 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.

B. *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(b) – it shows the sub-grids 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 [19]. We develop some heuristics for this problem. The pseudocode for this is shown in Algorithm 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.

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 grid sizes for the two children as shown in lines 14–18.

The partitioning is always done along the longer dimension to ensure that the rectangles are as square-like as possible. Figure 4 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 4(a) than in Figure 4(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 $1 \leq i \leq k$
1  Construct a Huffman tree, $H$, over the nested domains with
   execution time ratios as weights ;
   /* Construct a balanced split-tree using the
      Huffman tree                                    */
2  for every internal node $u$ of $H$ traversed in BFS order do
3      if ($u$=root) then
4          $(P_x(u), P_y(u)) = (P_x, P_y)$
5      if ($P_x(u) \leq P_y(u)$) then
6          $P_{ShortDim} = P_x(u)$, $P_{LongDim} = P_y(u)$ ;
7      else
8          $P_{ShortDim} = P_y(u)$, $P_{LongDim} = P_x(u)$ ;
9      end
10     Let $l$ and $r$ denote the left and right children of $u$ respectively ;
11     Let $Subtree_l$ and $Subtree_r$ denote the nested domains in the
       subtrees rooted at $l$ and $r$ respectively ;
12     $W_l = \sum_{j \in Subtree_l} R_j$, $W_r = \sum_{j \in Subtree_r} R_j$ ;
13     Divide $P_{LongDim}$ into $P_l$ & $P_r$ in the ratio of $W_l : W_r$;
14     if ($P_x(u) \leq P_y(u)$) then
15         Set $(P_x(l), P_y(l)) = (P_{ShortDim}, P_l)$ &
           $(P_x(r), P_y(r)) = (P_{ShortDim}, P_r)$ ;
16     else
17         Set $(P_x(l), P_y(l)) = (P_l, P_{ShortDim})$ &
           $(P_x(r), P_y(r)) = (P_r, P_{ShortDim})$ ;
18     end
19 end
```

Algorithm 1: Partitioning algorithm

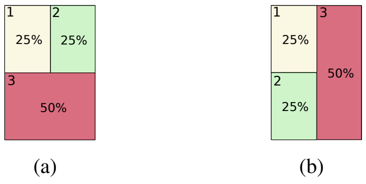
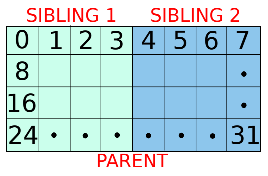


Fig. 4. Partitions for $k = 3$ when the first partition is along the longer dimension (a) and when it is along the shorter dimension (b)

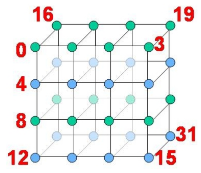
C. Mapping

Weather simulations are very communication intensive. For example, in WRF, each integration time-step involves 144 message exchanges with the four neighbouring processes [3]. IBM's HPCT [21] profiling tools show that about 40% of the total execution time in WRF is spent in communication.

Mapping is the placement of processes in the virtual topology onto the physical network topology. In this work we consider supercomputers with 3D torus interconnects and hence we address the problem of 2D to 3D mapping as shown in Figure 5. Figure 5(a) shows the 2D virtual process topology for a WRF input with 2 sibling domains (sibling 1 and sibling 2) of identical size. This virtual topology is used by the application for MPI communications. Figure 5(b) shows the 3D torus architecture of many modern-day 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



(a) Virtual process topology for 32 processes



(b) Topology-oblivious mapping on 4x4x2 torus

Fig. 5. 2D to 3D mapping

the 2D topology because of the difference in the 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.

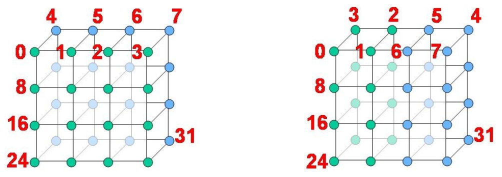
1) Topology-oblivious mapping: The partitioning scheme subdivides the processor space into rectangular regions for simultaneous executions of the nested simulations. The case of 2 nested simulations is shown in Figure 5(a). The processes 0–3, 8–11, 16–19, 24–27 are allocated to one nested simulation 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 5(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.

This simple mapping scheme 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.

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.

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 5(a) is shown in Figure 6(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 5(a) and green nodes in Figure 6(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 5(a) and blue nodes in Figure 6(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 6(a).



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

Fig. 6. 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 two z-planes so that half the rectangle is in first plane and the other half is in the second plane. This is illustrated in Figure 6(b).

Processes in the first rectangle of Figure 5(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 $(1, 0, 1)$, and so on. This ensures the processes in the first rectangle have 1-hop distant neighbours. Processes in the second rectangle of Figure 5(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 $(2, 0, 0)$, 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.

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 also applicable where there is an overlap between the processors allocated to different dependant sub-tasks of an application.

IV. Experiments and Results

A. Domain Configurations

We used WRF [2] for all our experiments. The parent simulation domain in WRF can have multiple child domains, called *nests*, which in turn can have children at the second level. Nests at the same level are called *siblings*. 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 178x202 and 925x820. For empirical evaluation, we chose the following two regions.

South East Asia – This covers 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 modelling framework. Figure 7 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.

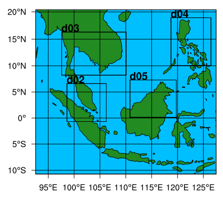


Fig. 7. Sample domain with four sibling nests at 1.5 km resolution.

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 94x124 to 415x445 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 x 307 at 24 km resolution and the nests have 8 km resolution, with up to 4 siblings at the first level of nesting.

B. Experimental Setup

IBM Blue Gene/L – Blue Gene/L (BG/L) [22] 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. 3D torus network is the primary communication network in BG/L. We have experimented on maximum of 1024 cores on BG/L.

IBM Blue Gene/P – Blue Gene/P (BG/P) [23] is the second generation of Blue Gene supercomputers. Each node contains

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

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 support 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.

WRF Runtime Setup. WRF-ARW version 3.3.2, was used for the experiments. Parallel netCDF (PnetCDF) [24] 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.

C. 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.

1) Improvement in per-iteration time: The average and maximum performance improvement in terms of decrease in the time required for the integration step of WRF is 21.14% and 33.04%. This is the overall improvement in the simulation performance on 1024 cores (512 nodes in VN mode) on BG/L from 85 configurations of varying nest sizes and varying number of siblings. The minimum nest size considered is 178x202 and the maximum nest size in these configurations is 394x418. The number of siblings vary from 2–4. This improvement in integration time 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.

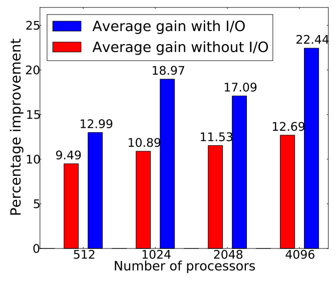


Fig. 8. Performance improvement of execution time on up to 4096 BG/P cores including and excluding I/O times

Figure 8 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.

2) Improvement in communication time: The average and maximum percentage improvement in MPI_Wait time is shown in Table I. 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. This leads to less congestion and smaller delay for point to point message transfer between neighbouring processes.

TABLE I
AVERAGE AND MAXIMUM IMPROVEMENT IN MPI_WAIT TIMES ON BG/L AND BG/P

# processors	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

3) 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 partitioning strategy are shown in Table II. Figure 9 compares the nested execution times for this configuration. The first 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, the execution times add up resulting in 1.1 seconds. In our parallel strategy, when the siblings are solved on subset of processors, the times taken 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 nest solve step for the 4 siblings is 0.7 seconds in our approach and 1.1 seconds in the default approach, thereby resulting in 36% performance gain for the sibling domains.

TABLE II
SIBLING CONFIGURATIONS FOR 4 SIBLINGS ON BG/L

	Sibling 1	Sibling 2	Sibling 3	Sibling 4
Nest size	394x418	232x202	232x256	313x337
#Processors	18x24	18x8	14x12	14x20

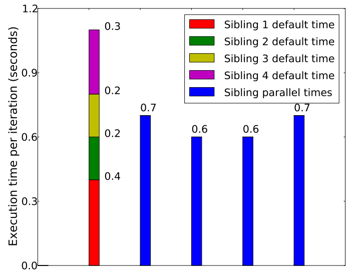


Fig. 9. Sibling execution times on 1024 processors on BG/L for 4 siblings

Our 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 3 large siblings of sizes 586x643, 856x919 and 925x850. The performance improvement for different number of processors and the nest execution times for default sequential strategy and our approach are shown in Figure 10. 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 and hence we observe a performance improvement of 20.64% for 8192 processors.

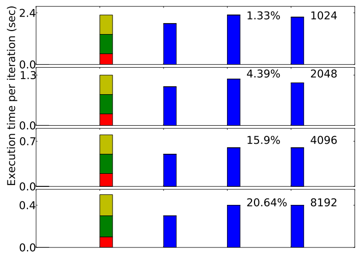


Fig. 10. Sibling execution times on up to 8192 processors on BG/P. The legends on the rightmost side show the number of processors.

4) 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 2 siblings is 19.43% whereas the average improvement in execution time for experiments involving 4 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 III.

TABLE III
SIBLING CONFIGURATIONS AND PERFORMANCE IMPROVEMENT FOR VARYING NEST SIZES ON UP TO 8192 BG/P CORES

Maximum nest size	205x223	394x418	925x820
% improvement	25.62	21.87	10.11

D. *Improvement with topology-aware mapping*

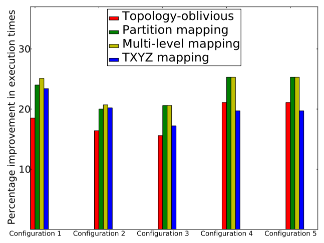
In this section, we present the performance improvement achieved by the topology-aware mappings discussed in Section III-C2. Table IV shows the execution times per iteration for the default strategy, the topology-oblivious and topology-aware mappings on 1024 BG/L cores. The first three rows correspond 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.

TABLE IV
EXECUTION TIMES (SEC) FOR DEFAULT, TOPOLOGY-OBLIVIOUS AND TOPOLOGY-AWARE MAPPINGS FOR VARIOUS SIBLING CONFIGURATIONS ON BG/L

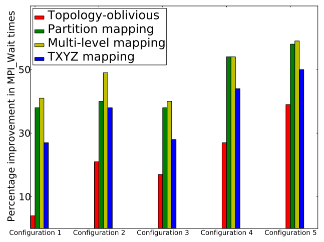
Default	Topology-oblivious	Partition mapping	Multi-level mapping	TXYZ mapping
2.77	2.25	2.10	2.07	2.12
3.69	3.08	2.95	2.92	2.95
3.43	2.89	2.72	2.72	2.83
4.98	3.92	3.72	3.72	3.99
4.75	3.53	3.39	3.33	3.44

Figure 11(a) illustrates the percentage improvement in execution times over the default strategy. 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 III-C2, the overall simulation is not adversely affected because the nested simulations are executed r times more than the parent simulations.

Table V shows the execution times per iteration for the default strategy, the topology-oblivious and topology-aware

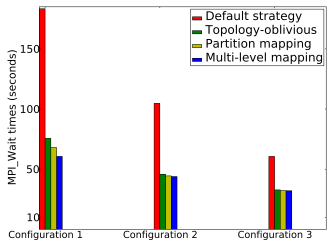


(a) Percentage improvement in execution times

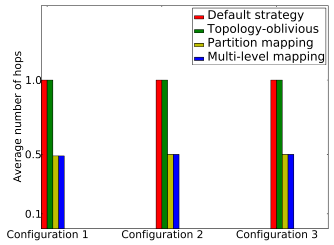


(b) Percentage improvement in MPI_Wait times

Fig. 11. Percentage improvement with and without topology-aware mapping on 1024 BG/L cores



(a) MPI_Wait times



(b) Average number of hops

Fig. 12. Reduction in MPI_Wait times and average number of hops with and without topology-aware mapping on 4096 BG/P cores

mappings for various sibling configurations on BG/P. The 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 the load imbalance in WRF.

TABLE V
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 12 illustrates the reduction in the MPI_Wait times and the average number of hops over the default strategy. 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 a 50% reduction in the average number of hops. This is due to our efficient mapping heuristics that map the neighbouring processes in the virtual topology to the neighbouring torus nodes. 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. The increase in communication times for the default approach is due to more load on the network because of halo exchanges across multiple hops and hence more delay. We plan to explore other mapping/contention optimization algorithms as future work.

E. *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 ten 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 13(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 13(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 14. 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 13(c).

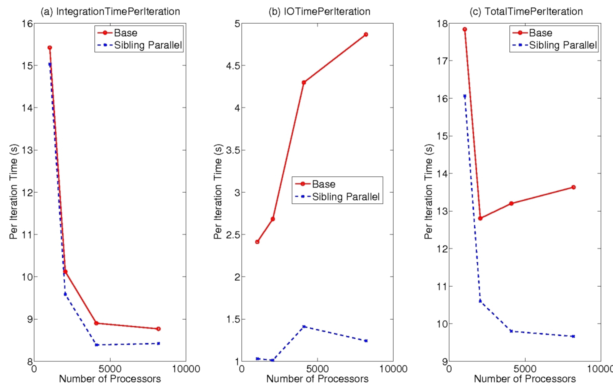


Fig. 13. Variation of integration, I/O, and total per iteration times with number of processors on BG/P.

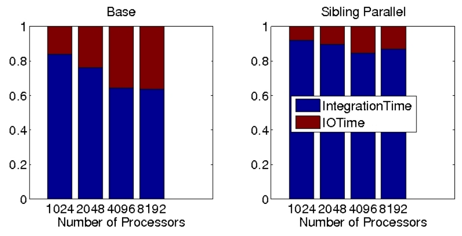


Fig. 14. Variation of fraction of integration and I/O times averaged over all the different configurations vs. number of processors on BG/P.

F. 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.

G. Scalability and speedup

We executed a simulation with two sibling nests of 259x229 size for the default sequential approach and our simultaneous execution approach, varying the number of processors from 32 to 1024. Figure 15 shows the scalability and speedup curves. Both the approaches have similar scalability saturation limits. However, our approach exhibits lower execution times for all processor sizes. 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.

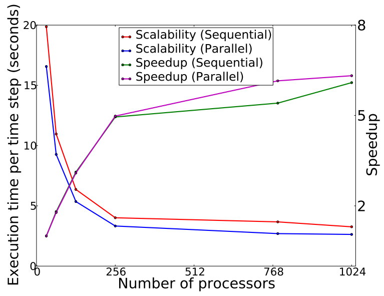


Fig. 15. Scalability and speedup of default sequential strategy and our concurrent execution approach.

V. Discussion

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 [25]. 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 [26].

VI. Conclusions and Future work

In this paper, we presented a comprehensive scheme to optimize weather simulations involving multiple nested regions of interest. We show 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. We developed 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 7% with respect to the total execution time. We developed 2D to 3D mapping heuristics that take into consideration communication in the nested simulations as well as the parent simulation. We achieve up to 33% improvement in performance with up to 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, ours is the first work which optimizes the parallel execution of weather simulations involving multiple nested regions of interest.

In the current work, we experimented with topology-aware mappings for foldable mappings. In future, we plan to extend the mapping heuristics for non-foldable mappings as well as develop novel schemes for the 5D torus topology of Blue Gene/Q system. We also plan to simultaneously steer these multiple nested simulations.

References

[1] J. Michalakes, J. Dudhia, D. Gill, T. Henderson, J. Klemp, W. Skamarock, and W. Wang, "The Weather Reseach and Forecast Model: Software Architecture and Performance," in *Proceedings of the 11th ECMWF Workshop on the Use of High Performance Computing In Meteorology*, October 2004.

[2] W. C. Skamarock and et al., "A Description of the Advanced Research WRF version 3," *NCAR Technical Note TN-475*, 2008.

[3] J. Michalakes and et al., "WRF Nature Run," in *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, 2007.

[4] "Top 500 Supercomputing Sites," http://www.top500.org.

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

[6] D. J. Kerbyson, K. J. Barker, and K. Davis, "Analysis of the Weather Research and Forecasting (WRF) Model on Large-Scale Systems," in *PARCO*, 2007, pp. 89–98.

[7] J. Delgado and et al., "Performance Prediction of Weather Forecasting Software on Multicore Systems," in *IPDPS, Workshops and PhD Forum*, 2010.

[8] S. M. Sadjadi and et al., "A modeling approach for estimating execution time of long-running scientific applications," in *IPDPS, Fifth High-Performance Grid Computing Workshop*, 2008.

[9] H. D. Simon and S.-H. Teng, "How good is recursive bisection?" *SIAM J. Sci. Comput.*, vol. 18, no. 5, pp. 1436–1445, Sep. 1997.

[10] G. Karypis and V. Kumar, "Multilevel k-way partitioning scheme for irregular graphs," *Journal of Parallel Distributed Computing*, vol. 48, no. 1, pp. 96–129, 1998.

[11] I. Moulitsas and G. Karypis, "Architecture aware partitioning algorithms," in *International Conference on Algorithms and Architectures for Parallel Processing*, 2008, pp. 42–53.

[12] A. Bhatele, G. Gupta, L. V. Kale, and I.-H. Chung, "Automated Mapping of Regular Communication Graphs on Mesh Interconnects," in *Proceedings of International Conference on High Performance Computing (HiPC)*, 2010.

[13] I.-H. Chung, R. E. Walkup, H.-F. Wen, and H. Yu, "MPI performance analysis tools on blue gene/l," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, ser. SC '06, 2006.

[14] A. Bhatele, N. Jain, W. D. Gropp, and L. V. Kale, "Avoiding hot-spots on two-level direct networks," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11, 2011, pp. 76:1–76:11.

[15] T. Hoefler and M. Snir, "Generic topology mapping strategies for large-scale parallel architectures," in *Proceedings of the international conference on Supercomputing*, ser. ICS '11, 2011, pp. 75–84.

[16] 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*, ser. C3P, 1988, pp. 210–221.

[17] B. N. Delaunay, "Sur la sphere vide," *Bulletin of Academy of Sciences of the USSR*, vol. 6, pp. 793–800, 1934.

[18] H. S. M. Coxeter, "Barycentric Coordinates," in *Introduction to Geometry*, 2nd ed. Wiley, 1969, pp. 216–221.

[19] A. Lingas, R. Pinter, R. Rivest, and A. Shamir, "Minimum Edge Length Rectilinear Decompositions of Rectilinear Figures," in *20th Allerton Conference on Communication, Control, and Computing*, 1982, pp. 53–63.

[20] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. The MIT Press, 2001.

[21] I.-H. Chung, R. Walkup, H.-F. Wen, and H. Yu, "MPI tools and performance studies - MPI performance Aanalysis Tools on Blue Gene/L," in *Proceedings of the ACM/IEEE conference on Supercomputing*, 2006.

[22] I. B. G. Team, "Overview of the Blue Gene/L system architecture," *IBM Journal of Research and Development*, vol. 49, p. 195, 2005.

[23] ——, "Overview of the Blue Gene/P project," *IBM Journal of Research and Development*, vol. 52, p. 199, 2008.

[24] J. Li, W.-k. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale, "Parallel netCDF: A High-Performance Scientific I/O Interface," in *Proceedings of the ACM/IEEE conference on Supercomputing*, 2003.

[25] "LAMMPS Molecular Dynamics Simulator," http://lammps.sandia.gov.

[26] "Regional Ocean Modeling System," http://www.myroms.org.