IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

TRACKING AND ANALYSING THE VARIATIONS IN THE WEATHER PHENOMENA BY UTILIZING TREE ANDDIFFUSION-BASED PROCESSOR REALLOCATION STRATEGY

[1]Author: B Rebecca, Research scholar-CSE department at Sri Satya Sai University of Technology & Medical Sciences

[2]Author: Dr. Pankaj Kawadkar, Professor at CSE department at Sri Satya Sai University of Technology & Medical Sciences

ABSTRACT

Earth is a living, breathing planet that changes on a regular basis. Weather patterns and events play a significant role in this development. While these patterns and occurrences are required for our planet to remain habitable, they may also wreak significant damage and cost billions of dollars in repair and rescue attempts. Weather phenomena are natural events caused by one or more of the following: the water cycle, pressure systems, and the Coriolis effect. Precipitation, wind, and heat are frequently involved or related to them. The main aim of this study is tracking and analysing the variations in the weather phenomena by utilizing tree-based diffusion-based processor reallocation strategy. We devised ways for efficiently partitioning and repartitioning the nests among the processors in this paper. We look at an application of tracking numerous ordered cloud clusters in tropical weather systems as a case study. To discover such clouds, we first present a parallel data analysis approach. We created a tree-based hierarchical diffusion method that reallocates processors for the nests at a lower cost of redistribution. We do this via a novel tree restructuring method. We demonstrate that our approach has a lower redistribution cost and 53 percent fewer hop-bytes than a processor reallocation solution that ignores existing processor allocation.Fortran90 programming language have been used for implementation of algorithm.

KEYWORDS *– Weather phenomenon, Tree based, Redistribution; processor reallocation; data analysis; cloud tracking etc.*

1. INTRODUCTION

Wind, cloud, rain, snow, fog, and dust storms are all prevalent meteorological phenomena on Earth. Natural disasters such as tornadoes, hurricanes, typhoons, and ice storms are less common. Weather is caused by changes in air pressure, temperature, and moisture from one location to another.

Events in Weather phenomenon

- Blizzard
- Cloud
- Dust Devils
- Fog
- Frost
- Haboob
- Hailstorm
- Heat Wave
- Hurricane

- Lightning

- Rain
- Rainbow
- Snow
- Thunderstorm
- Tornado

Weather forecasting is primarily concerned with predicting weather conditions for a specific time in the future. Weather forecasts are essential for predicting the weather in the future. Weather forecasting can be done in a variety of ways, ranging from simple sky observation to extremely complicated computerised mathematical models. Weather forecasting is necessary for a variety of applications. Climate monitoring, drought detection, severe weather prediction, agricultural and production, energy industry planning, aviation industry planning, communication, pollution dispersal, and so on are only a few of them. When it comes to military operations, there is a long history of occasions where weather conditions have changed the outcome of conflicts. Due to the dynamic nature of the atmosphere, accurate weather forecasting is challenging. Some variables can be used to indicate the weather at any given time. One discovered that the most significant factors are being chosen to be involved in the prediction process.

Weather simulations are essential for weather forecasting. PCs have been assuming an imperative part in weather forecasting since the initiation of weather displaying. A weather model is a scientific portrayal of the air forms in view of physical, biological and chemical standards. The physical procedures are portrayed by standard and halfway differential conditions, which are understood numerically utilizing techniques like finite-contrast approximations. Different meteorological associations around the globe create local to-worldwide scale weather models to understand the logical premise of weather phenomena, potential effects of environmental change and choices for mitigation. Weather models that can predict disasters ahead of time can save lives and save damage. Fast variations in rare weather occurrences have recently been observed, owing to advancements in barometrical synthesis, and are capable of impacting the earth's surface atmosphere.

2. LITERATURE REVIEW

Xiaoqiang Liu, et al (2021) - The limitation of the classic FAO-56 Penman–Monteith technique, which requires complete meteorological input data, must be addressed to increase the accuracy of predicting reference crop evapotranspiration for effective water resource management and optimal irrigation scheduling design. The impacts of applying five data splitting procedures and three distinct input dataset time periods on predicting ET0 are investigated in this study. To achieve this goal, the random forest (RF) and extreme gradient boosting (XGB) models were used, together with a K-fold cross-validation strategy.

C. Lennard, and G. Hegerl, (2015) - They devised a supervised technique called SOM for analysing surface rainfall in conjunction with synoptic circulation. It was investigated for two types of stations in South Africa's distinct rainfall zones. These synoptic circulations were identified as mid-latitude-based cyclones in the winter and summer, but no circulations were linked to rainfall in the spring and autumn. The capacity of SOMs to match the synoptic movers of observed rainfall records is evaluated in this work, which effectively downscales large-scale synopses data to an accurate resolute reaction of the surface.

Piyush Kapoor (2013) - The fluctuation in historical weather circumstances must be used to anticipate future weather conditions. It's extremely unlikely that the weather on the day in question will be identical to the same day the prior year. However, the chances of it matching within the next fortnight of the prior year are extremely high. As a result, for the previous year's fortnight, a sliding window of the size of a week is chosen. Every week of the sliding window is then compared to the week of the current year. The approach's findings revealed that the method for forecasting meteorological conditions is quite effective, with an average accuracy of 92.2 percent.

Malakar, P., et al (2012) -High-fidelity compute demanding simulations of several finer regions of interest within a large simulation domain are required for accurate and timely prediction of meteorological phenomena such as hurricanes and flash floods. Due to their sub-linear scalability, current weather apps run these stacked models sequentially utilising all available CPUs, which is inefficient. We offer a method for running several nested domain simulations in parallel, based on

partitioning the 2-D processor grid into distinct rectangular sections for each domain. On torus interconnects, we offer a novel mix of performance prediction, processor allocation algorithms, and topology-aware region mapping.

PreetiMalakar, et al (2011) - Critical weather applications like as cyclone tracking require online remote visualisation and steering for effective and timely analysis by a geographically dispersed climate scientist community. A steering framework for controlling high-performance simulations of critical weather events must take into account both the scientists' steering inputs as well as the application's criticality requirements, such as a minimum simulation progress rate and continuous visualisation of significant events. We created an integrated user-driven and automated steering framework for simulations, online remote viewing, and analysis for crucial weather applications in this research. This gives the user control over a variety of application parameters, such as the region of interest, simulation resolution, and data frequency for visualisation.

3. PROPOSED METHODOLOGY

3.1 Parallel Data to Find Organized Cloud Clusters

We will describe an approach for parallel data processing of simulation output in this part. To detect tall clouds in tropical weather systems, the technique examines the cloud water mixing ratio (QCLOUD) and outgoing long wave radiation (OLR) in WRF simulation output. Cumulonimbus clouds are the name given to these types of clouds. They extend vertically from 1 km to more than 10 km above the surface. The amount of liquid water in a cloud is measured by QCLOUD. In general, high QCLOUD values equate to lofty clouds. The infrared radiation at the top of the atmosphere is known as OLR. Low OLR patterns suggest the presence of organised cloud systems (such as tropical depressions and cyclones), which would be characterised by tall cumulonimbus clouds. The use of OLR and QCLOUD together improves the detection of such systems and eliminates the identification of solitary cumulonimbus (as QCLOUD alone would). We set the highest limit for OLR at 200.Each WRF process produces output for its subdomain and saves it to a split file. As seen in Algorithm 1, these split files are evaluated in parallel. This algorithm creates small clusters that are contiguous, non-overlapping, and do not expand out of control. It is easy and quick, making it ideal for online analysis. Let P be the number of processes that run WRF and N represent the number of processes that examine the QCLOUD values in the split files. The split files are fed into the algorithm as input $\{F_1, F_2, \cdots, F_P\}$. The N processes are given these split files to work with. k files (lines 1–2) are analysed by each of the N processes. S is a subset of files in which $|S| = k$, as a rectangle subset is chosen of (P_x, P_y), where $P_x \cdot P_y = P$ In WRF, this is the rectangular process decomposition. As a result, P is split into N rectangular subsets.If the outgoing long wave radiation OLR 200 (lines 4–9), the value of QCLOUD at each grid point in each split file is aggregated. The olrfraction, or the fraction of grid points that meet the above requirements, is determined (lines 7–8). All N processes send the aggregated QCLOUD values, one value per file, to a root process, rank 0 in our example. Each process will send a maximum of k values. It's worth noting that some of the split files may lack areas with OLR$\leq$ 200, in which case the process that owns them will submit fewer than k values. The aggregated QCLOUD values and the olr fraction values are gathered by the root process (line 11).The remainder of the procedure is only run on the root process. To begin, the non-increasing order of the aggregated QCLOUD values acquired from the split files is sorted (line 13). Multiple split files processed by multiple processes can be spanned by a contiguous region with heavy cloud cover. We use a variation of closest neighbour clustering (NNC) to create a contiguous region (line 14). NNC generates a collection of clusters, each of which contains a contiguous zone of high cloud cover. Each cluster is surrounded by a rectangle (lines 16–19), which serves as a nest for fine-resolution simulations in WRF.

Nearest Neighbour Clustering: Algorithm 2 shows the pseudo code for the NNC algorithm. It takes the sorted list of QCLOUD values, qcloudinfo, as an input. Each qcloudinfo element is a tuple of aggregate QCLOUD values for a split file and the fraction of the split file with OLR$\leq$200. The cloud cover for a subdomain is represented by the QCLOUD value of each element in the list. This algorithm uses the spatial position of a subdomain, i.e. the latitude and longitude extents of a subdomain, to determine proximity between two subdomains.The method loops through each element of the qcloudinfo input array (lines 2–20). Line 3 determines whether the total QCLOUD value and the fraction of the subdomain with OLR $\leq$200 exceed a threshold, which in our case is 0.005. This

eliminates the need to analyse smaller cloud-covered regions with a low QCLOUD score. Clusters are constructed depending on element proximity (lines 4–18). Each cluster depicts a large area of dense cloud cover. If an element is 1-hop or 2-hop away from an existing cluster, it is added to it. The list of clusters is initially empty. First, we look to see if the current element is within one hop of any other element in an existing cluster (lines 6–9). If this isn't the case, we look to see if the element is two hops distant from any other element in an existing cluster (lines 10–13).

Algorithm 1: Parallel Data Analysis (PDA) algorithm

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**

The DISTANCE function is used to calculate proximity in lines 6 and 10. If true, the element is added to the list. If an element is within a hop distance of a member, it is included to the cluster list iff it does not deviate the QCLOUD mean by more than a threshold (30% in our example) (lines 23–29). This assures a low standard deviation for a cluster of contiguous cloud regions and also aids in regulating the size of an existing cluster.A new cluster new-list is created if an element is not within two hops of any other element in any of the existing clusters. element is added to new-list, which is then added to the clusters collection. Clusters (16–18 lines). Clusters is a set of clusters produced by NNC that represents different contiguous regions of cloud cover.

Algorithm 2: Nearest Neighbour Clustering (NNC) algorithm

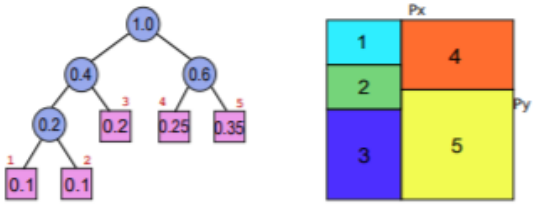
```
Input: Sorted array qcloudinfo
Output: Clusters: List of elements, clustered by proximity
1  Clusters = ∅;
2  LOOP: foreach element ∈ qcloudinfo do
3      if (element.qcloud ≥ threshold and
       element.olrfraction ≥ threshold) then
           /* Check if this element is physically
              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
```

The parallel data analysis technique runs on a distinct group of processors from the processors that operate the WRF simulation. As a result, PDA execution has no bearing on WRF execution times. Because the analysis of QCLOUD values in each split file is the most time-consuming phase in Algorithm 1, it is done in parallel. Experiments demonstrate that for most of the time steps, the number of items acquired at the root process is less than 200 for a maximum of 1024 split files. The sequential NNC algorithm (Algorithm 2) clusters such few values in less than a second. Parallel clustering would have been excessive for online analysis in this situation. In the future, we'd like to parallelize the NNC algorithm for simulations with a larger number of processors.

3.2 Processor Allocation

PDA is an algorithm for computing a collection of regions of interest (ROI) in a domain, which in our case are the places with a high amount of cloud cover. Simulations are spawned over the regions of interest in nested fashion. We mimic these nests at high resolutions in order to get greater precision. In these layered simulations, the resolutions are three times higher than those of the parent simulation. We made changes to the WRF code to allow us to generate nests on the go without having to halt the simulation. The initial data for nested domains is interpolated from the data for the parent domain.

It has been demonstrated that by running the nests on different subsets of the total number of processors, P, considerable performance increases can be realised, P. In order to calculate the size of the subset of processors for a nest and their position in the processor grid, we employ performance modelling and a Huffman tree-based technique, which are both implemented in Java processor grid $Px \times Py$ where $Px.Py$ = P. Based on the size and aspect ratio of the nests, the performance model is utilised to anticipate the execution times. The initial processor allocation for each nested domain is determined using the Huffman tree-based technique.



(a) Huffman tree for 5 nests with execution time ratios 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 1: Illustration of processor allocation for nests.

Figure 1 shows an example of processor allocation for five nests. Assume the ratios of the nests' projected execution times are 0.1: 0.1: 0.2: 0.25: 0.35. As shown in Figure 1, these ratios are used as weights in the construction of the Huffman tree (a). Figure 1 depicts the relevant processor sub-grid for each nest (b). The five sub-rectangles represent the number of processors used to run each of the nests. Table 1 shows the start rank of each processor sub-grid for this example setup, which is the rank of the processor in the north-west corner of the sub-rectangle, as well as the rectangular dimensions of each processor sub-grid for a maximum of 1024 cores.

Table 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

In succeeding time steps, the regions of interest may persist or vanish. The regions with a lot of cloud cover are the ones we're interested in. Clouds can build and dissipate over a long period of time. The PDA technique is used to find areas of interest (ROI) in the output of the current simulation time step on a regular basis (every 2 minutes). When a new ROI is discovered, a nest is created. When PDA fails to output an existing ROI, the nest is removed. A retained nest is one that has been output by PDA both in the previous and current invocations. Nest insertion, deletion, and retention alter the Huffman tree topology and, as a result, the processor allocation. As a result, the newly allocated set of processors (receivers) for a retained nest may differ from the previously allotted set of processors (senders). The data from the nest domain must be distributed to the receivers by the senders. To carry out this redistribution, we updated the WRF code. After calculating the quantity of data to be redistributed based on the nest size, MPI_Alltoallv is used to redistribute data for each nest. During the MPI_Alltoallv for that nest, processors that are neither senders nor receivers send and receive 0 value.

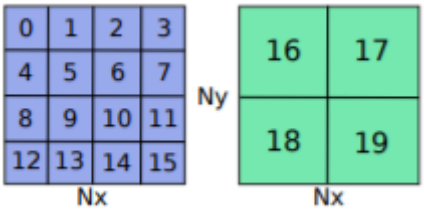
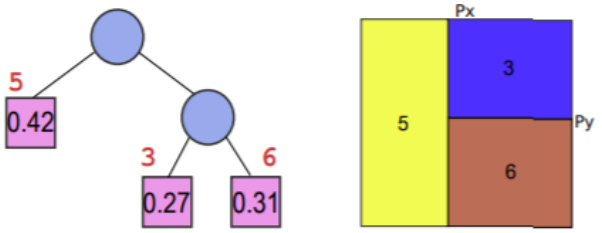


Figure 2:Data is redistributed from an old set of processors to a new set of processors assigned to a nest

Figure 2 shows an example for a nest size of $Nx \times Ny$. As indicated in the left grid, a nest is equally distributed across its allocated processors 0 to 15. As illustrated in the right grid of the picture, these processors send the nest data to the newly allocated processors 16 – 19. It can be seen that processor 16 now owns the part of the nest domain that was previously held by 0, 1, 4, and 5. As a result, 16 receives domain data from 0, 1, 4, and 5. In this scenario, the other receivers similarly receive data from four senders.The senders and receivers in the preceding example are nonintersecting sets. The cost of data redistribution between senders and receivers can be reduced if the senders and receivers are in close proximity. The redistribution cost in torus networks can be reduced by reducing the number of hops between senders and recipients.

- **Partition from scratch**

We segment the entire process grid in this method $Px \times Py$as mentioned in the preceding section, for processor allocation based on Huffman tree formed using the projected execution times of the nests as weights The tree is built without taking into account the current processor allocation. As a result of this method, there may be no overlap between senders and recipients, resulting in higher redistribution costs.



(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 3:Using a partition from scratch to allocate processors to nests

Take, for example, the configuration shown in Figure 1. Assume that PDA outputs the nests 3, 5, and 6 as regions of interest at the next invocation. As a result, nests 1, 2, and 4 will be removed, and a new nest, nest 6, will be created. The expected execution times of the nests 3, 5, and 6 should be 0.27: 0.42: 0.31. Figure 3 depicts the matching Huffman tree as well as the processor partition. Table 2 shows the start rank and rectangular dimensions of each processor sub-grid for each nest with a maximum of 1024 cores. We can see that there is no overlap between senders and recipients when we compare the previous and new allocations for nests 3 and 5 in Tables 1 and 2. This may raise the cost of redistribution.

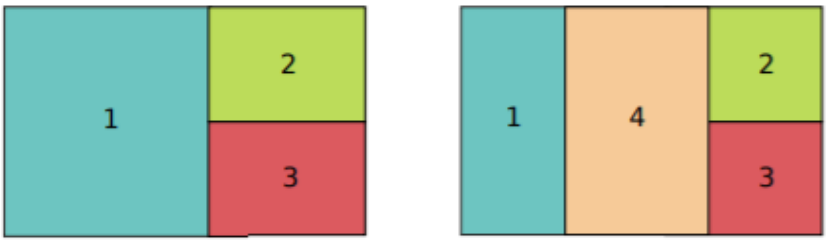
Table 2: Processor Allocation On 1024 Cores

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

The cost of redistribution with this strategy may be substantial in some circumstances. Due to the tree's creation in order of increasing weights, the rectangular divisions based on the Huffman tree are as square-like as possible. The nests' execution times are reduced by the square like partitions.

- **Tree-based hierarchical diffusion**

We try to maximise the overlap between the senders and receivers of the retained nests using this method. The main concept is to move the boundaries of rectangular partitions for the retained nests so that data is distributed across neighbouring processes and the overlap between old and new nest data is maximised. This reduces the cost of redistribution, particularly on torus networks. Figure 4 shows an illustration of this. The present processor partitioning for nests 1, 2, and 3 is shown in Figure 4(a). Existing partitions are resized when a new nest is added. As illustrated in Figure 4, the right boundary of rectangle for nest 1 is shifted to the left, while the left boundaries of nests 2 and 3 are relocated to the right, freeing up some processors for inserting the new nest (b). As a result, the old and new processor partitions for nests 1, 2, and 3 have a lot of overlap.

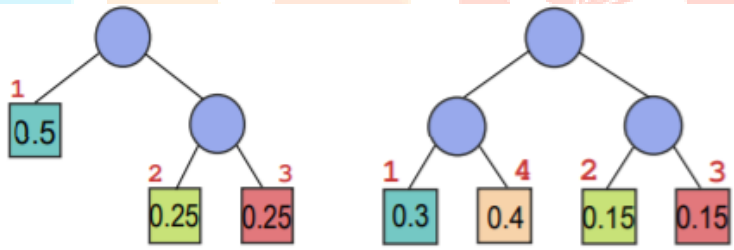


(a) Existing **(b) new processor**

Figure 4:Processor allocation in the hierarchical diffusion approach.

Rather than creating the Huffman tree from start, this repartitioning method works by changing the tree that corresponds to the present allocation. In the tree, the orientations of the nodes relating to the maintained nests are preserved. Because the weights represent the ratios of the multiple processors that will execute each nest, the weights of the old nodes, i.e., the retained nests, may be adjusted. The processor shares of current nests may change as new nests are added and/or old nests are eliminated.

When there is no deletion and simply insertion of new nodes, the new nodes are inserted near existing nodes with weights that are similar to the new nodes. We try to get rectangular partitions for the nests that are more square-like by inserting a new node near a node in the Huffman tree with equal weight. Inserting a new node near a node with a big weight differential, on the other hand, will result in skewed rectangles. As a result, square-like partitions result in shorter nest execution times, whereas skewed rectangular partitions result in longer nest execution durations.



(a) Existing trees in the hierarchical diffusion approach **(b) new trees in the hierarchical diffusion approach**

Figure 5: The weights at the leaf nodes are the predicted execution time ratios of the nests

This is demonstrated in Figure 4 as an example. Figure 5 depicts the current and new trees corresponding to the processor partitions of Figure 4. Node 4 is inserted near node 1 to create the new tree in Figure 5(b). This is because node 4's weight is the closest to that of node 1's new weight. The size of each node's partition is proportional to its weight. Thus, the nodes 1 and 4 get $\frac{3}{7}^{th}$ and $\frac{4}{7}^{th}$ the number of processors assigned to their parent node Because the weight differences between nodes 1 and 4 are smaller, the resulting rectangles for 1 and 4 will be as square as possible. This would not have been the case if node 4 had been placed near node 2, which has a weight of 0.15. This is due to the fact that the shares for 4 and 2 would have been $\frac{0.4}{0.55} = \frac{8}{11}$ and $\frac{0.15}{0.55} = \frac{3}{11}$.Due to the substantial discrepancy in weights, the rectangle for node 2 would not have been square. Figure 6 shows how this works. Rectangle 2 appears to be warped when compared to rectangle 4.

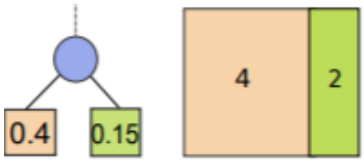


Figure 6: Skewed rectangle Due to the substantial weight differential between the two nodes

When nests are both inserted and deleted from the tree, the nodes corresponding to the deleted nests are removed. Furthermore, new nodes are placed in the sites of removed nests to preserve as much as possible the positions of the retained nests. This may increase the likelihood of existing and new nest processor allocations for the retained nests overlapping.

Algorithm 3 describes the algorithm for changing the existing tree for fresh processor allocation. The current tree oldtree, the deleted node list deletednodes, the updated weights of the retained nests rweights, and the weights of the new nodes nweights are used as inputs. The updated tree newtree is the result.

Algorithm 3: Tree-based hierarchical diffusion algorithm

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*

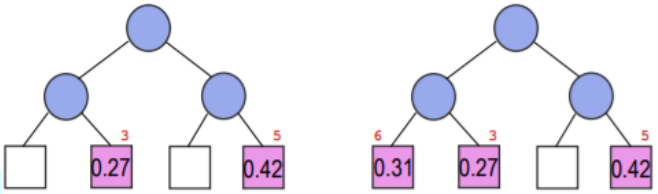
1 $freenodes = \emptyset$, $siblings = \emptyset$;
2 **foreach** $node \in deletednodes$ **do**
3 Mark *node* as free in the *oldtree*;
4 Add *node* to $freenodes$;
5 Add sibling of *node* to *siblings*;
6 **end**
7 **foreach** $weight \in rweights$ **do**
8 Update *weight* for the corresponding retained 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** $new\_weight \in nweights$ **do**
12 **if** $(|freenodes| > 1))$ **then**
13 Add *new_weight* to the position of *node*, where $node \in freenodes \wedge sibnode$ is sibling of $node \wedge d = Weight(sibnode) - new\_weight \wedge d = \underset{\forall\, s \in siblings}{\text{argmin}} (Weight(s) - new\_weight)$
14 Delete *node* from $freenodes$;
15 Delete *sibnode* from *siblings*;
16 **end**
17 **end**
18 **if** $(|nweights| \geq |deletednodes|))$ **then**
19 Build Huffman tree for the remaining new weights rooted at $node \in freenodes$;
20 **else**
21 Delete the remaining nodes in $freenodes$ from *oldtree*;
22 **end**
23 Copy *oldtree* to *newtree*;

To begin, deleted nodes in old tree are designated as free and added to the set free nodes (lines 2–6). These nodes' siblings are added to the set of siblings (line 5). These will be utilised as insertion places later. The weights of the nodes that are kept are changed (lines 7–9). The weights of internal nodes are updated based on the deletion and alteration of weights of maintained nodes (line 10). In the positions of the deleted nodes (lines 11–17), new weights are added. As previously stated, new nodes should be placed near those with the closest weights. As a result, we look at the weights of the deleted nodes' siblings. The existing tree structure will be minimally modified when a new node is

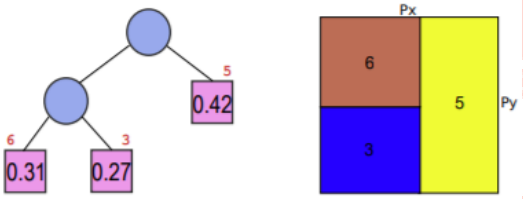
inserted in lieu of a removed node. Line 13 demonstrates this. A new weight is added to the position of the node, which was previously indicated as empty. The weight of a node is chosen so that the gap between the weight of its sibling sibnode and the new weight is as small as possible. Lines 14–15 remove node and sibnode from their respective sets.

The operation in line 13 is only performed when the set free nodes contain multiple nodes. Because we build a Huffman tree using the remaining unmatched weights in nweights when the frequency of deletions is fewer than the number of insertions, and this subtree is rooted at the position of the last element in free nodes. Lines 18–20 demonstrate this. We destroy the remaining nodes of freenodes if there are more insertions than deletions (line 21). The new tree is created from the updated old tree.

When compared to the partition from scratch method, this methodology reduces data transit between senders and receivers, resulting in a considerable reduction in redistribution time. This is because we try to arrange receivers in such a way that there is a lot of overlap between senders and receivers, and the receivers are senders' neighbours.



(a) Deleted nodes marked empty and weights of retained nests modified. (b) Node 6 inserted near node 5



(c) Remaining deleted nodes removed (d) Sub-division of the processor grid based on the modified tree.

Figure 7: Delete nests 1, 2, 4, keep nests 3, 5, and add a new nest 6 using the tree-based hierarchical diffusion technique.

Figure 7 shows the processor allocation for the case in Figure 1 using a tree-based hierarchical diffusion approach. Let us assume the same output of PDA that was considered in Figure 3 to compare with the partition from scratch strategy. Nests 1, 2, and 4 have been removed; nests 3 and 5 have been kept, and nest 6 has been added as a new zone of interest. Figure 7(a) depicts the tree after nodes 1, 2, and 4 have been eliminated and the weights of nodes 3 and 5 have been changed. Because the two free rectangles represented by deleted nodes 1, 2 can be regarded one free rectangle, they have been concatenated as one empty node. As a result, there are two free spaces for inserting new node 6 - one sibling node has a weight of 0.27, while the other has a weight of 0.42. Because 0.31-0.27 <0.42-0.31, i.e., the weight of node 3 is closer to that of node 6, node 6 is inserted in the position of sibling of node 3. Figure 7 depicts the rectangular division based on this tree (d). When comparing this to the partitioning produced using the partitioning from scratch method (shown in Figure 3(b), we can see that there is a lot of overlap between the old and new set of processors for nests 3 and 5, but the partitioning from scratch method has no overlap. Also, because we aim to keep the placements of retained nests as intact as possible, the rectangles for 3 and 5 spread to neighbouring processes.

Note that with this method, the changed tree may no longer be a Huffman tree. The changes, however, cause some overlap between new and old processors, as well as redistribution among neighbouring processes. Our methods can handle a huge number of processors. With a bigger overall processor count, the maximum number of hops between the old and new set of processors is expected to grow for the scratch approach. As a result, the scratch method's data redistribution time

may grow as the number of processors increases. The number of nests determines how many processors are reallocated via Huffman tree building or reorganisation, which is unaffected by an increase in processor count.

➢ **Dynamic Strategy**

The variations in performance between the two techniques, namely the partition from scratch method and our diffusion-based method, are determined by the execution times of the generated partitions as well as the redistribution costs. In both the partition from scratch approach and our diffusion-based method, the execution time ratios of the nests, and hence the percentage of total number of processors allotted to the nests, are the same. The rectangular grids and aspect ratios of the rectangles for same nest configuration may not be precisely the same due to integral sides of the sub-rectangles. One way might allocate 16×18 whereas the other might allocate 17× 17. As a result, the execution times of the nests for the two techniques may differ slightly.

Similarly, while we expect our diffusion-based method to have lower redistribution costs than the partition from scratch method, there may be circumstances where the redistribution costs are almost the same in both methods. This is because both methods rely on tree construction with weights based on the ratios of predicted execution times of nests. The weights' relative order influences the tree's construction and, as a result, the rectangular processor grid given to the nests. Identical relative weights of those nests that remain between reconfigurations could lead to similar trees for both techniques, and hence similar redistribution costs. As a result, we present a dynamic method that chooses the approach that takes the least amount of redistribution and execution time. To do so, we must forecast both of these times.

- **Performance model for redistribution time:**MPI_Alltoallv between the processors is the most important component of the redistribution time. In mesh and torus-based networks, we assume a direct MPI_Alltoallv method between the processors. The maximum communication time between senders and receivers is predicted to be MPI_Alltoallv time. The size of the message that a sender will deliver to its receiver(s) is determined first, followed by the number of hops between the sender and its receivers. We may calculate the communication time for each sender-receiver pair using this method. MPI_Alltoallv is projected to take the longest of these communication times. For non-mesh networks, such as switched networks, the time it takes for the sender to transmit messages to all receivers can be added to anticipate the MPI_Alltoallv time.
- **Performance model for execution time**: We measured the execution timings of a small group of domains (size = 13) with various domain sizes on a few (10 in our case) processor sizes within the maximum number of processors (1024 in our case). The execution timings of these 13 domains are interpolated using Delaunay triangulation from the execution times of the nests produced in our simulation. We also forecast the execution times of the nests for each of the 10 processor sizes. We conduct linear interpolation on these times to anticipate the execution time on the desired number of processors. As seen in Section V, this results in good forecast accuracies. The prediction execution durations are employed for dynamic method selection as well as determining the weights of the nests required for processor allocation in both our tree-based approaches and the partition from scratch.

3.3 Programming language

WRF was created and designed to run efficiently on massively parallel systems. It's written in Fortran90 and may be configured to run in serial, parallel (MPI), or mixed-mode (OpenMP and MPI) mode.

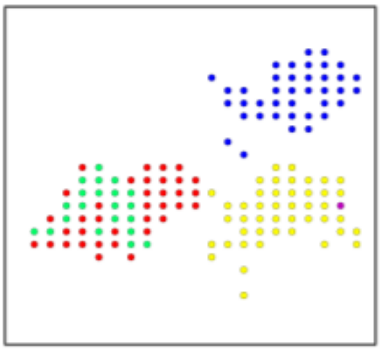
4. RESULT AND DISCUSSIONS

4.1 Data analysis algorithm

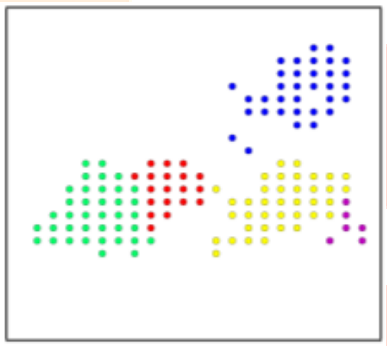
The data analysis technique is used to identify clouds and create nests, which is one of the most important aspects of our work. Using QCLOUD values in non-increasing sequence, we create clusters of continuous places with high cloud cover. In this list, a QCLOUD value reflects the aggregated QCLOUD over a subdomain, where OLR ≤200. The clustering of contiguous regions is based on the proximity of the subdomains.

In this part, we compare our disclosed closest neighbour clustering algorithm against a straightforward nearest neighbour clustering

approach. We present clustering using only two hop distance criteria in Figure 8(a). This technique determines whether the list entry is within two hops of a cluster that already exists. There are some overlapping clusters, as we can see. The clusters created by our technique are shown in Figure 8(b). Because we first check for 1 hop distance and then 2 hop distance, the clusters created by our method are non-overlapping. Only if the list entry is not within one hop of an existing cluster are we checking for a two-hop distance. This guarantees that the list element is added to the cluster that is closest to it. To ensure that the cluster size does not rise uncontrolled, we only insert into a cluster if the mean deviation is less than 30%.



(a) Clustering of nearest neighbours using a 2-hop distance and no mean deviation condition. Clusters are spatially overlapping



(b) Clustering of nearest neighbours using 1-hop and 2-hop distances and a 30% mean deviation criterion. Clusters do not cross each other.

Figure 8: For our parallel data processing approach, we use nearest neighbour clustering

4.2 Domain Configurations or tools used

For all of our experiments, we used a popular open-source weather forecasting tool called Weather Research and Forecast Model (WRF) version 3.3.1, It is a regional and mesoscale numerical weather prediction model and is used by weather agencies all over the world. The parent simulation domain can have several offspring domains, which are referred to as nests. During the simulation, these nests formed over several regions of interest. For dynamic insertion and deletion of nested domains, we changed the WRF source code. We ran a simulation over the Indian region from 60°E - 120°E and 5°N - 40°N for the Mumbai rainstorm of July 2005 the simulation took place from July 24, 2005, at 18:00 hours, until July 27, 2005, at 18:00 hours. The parent simulation resolution was 12 km, while the nested domain resolutions were 4 km. For both real and simulated test scenarios, we compared our tree-based hierarchical diffusion strategy against the partition from scratch method. We experimented with simulated test cases for the dynamic method.

Real: Nests formed over areas with a lot of cloud cover, which our parallel data analysis system picked up on. During these runs, the greatest number of nests created was 7. The nests formed have a maximum and minimum size were 202 × 349 and 175 × 175. There were about 100 processor allocation reconfigurations for the nests.

The real traces for our application showed less configuration changes and (4–5) nests on average than the synthetic traces. To test our algorithm for higher numbers of nests per time step and more redistributions per adaptation point, we created

some synthetic test cases. Up to 70 different nest layout alterations were investigated, with the number of nests ranging from 2 to 9. Nests were added and removed at random. The nests formed have a maximum and minimum size were 361 × 361 and 181 × 181.

4.3 Experimental

Our simulations were run on two different types of systems: a Blue Gene/L system and a fist Intel Xeon cluster. Our experimental setups are listed in Table 3. We created a folding-based topology-aware mapping that maps neighbouring processes to neighbouring processors on the 3D torus for the Blue Gene/L experiments. All of our studies used this topology-aware mapping, which ensured that processes in the process grid were only one hop apart. This benefits both the partition from scratch method and the diffusion-based strategy in terms of execution times. All of our studies were carried out on a graphics workstation at the Indian Institute of Science (IISc) with an Intel(R) Pentium(R) 4 CPU running at 3.40 GHz and an NVIDIA GeForce 7800 GTX graphics card.

Table 3: Configurations on Simulation

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

4.4 Improvement in redistribution time

For the real test instances, our tree-based hierarchical diffusion method improved redistribution times by 14 percent and 12 percent over the partition from scratch method on 512 and 1024 Blue Gene/L cores, respectively.

Table 4: Redistribution Times for Synthetic Test Cases based on Average Improvement

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

For the simulated test scenarios, Table 4 illustrates the average % improvement in redistribution times for our tree-based hierarchical diffusion method over the partition from scratch method. It can be seen that in the case of Blue Gene/L, which features a 3D torus network, the performance improvement is greater. This is due to the fact that our tree-based hierarchical strategy picks the new processor allocation based on the process grid's neighbours. Because of our topology-aware mapping, Blue Gene/L neighbours in the process grid are also neighbours in the processor topology. However, because there is no regular mesh/torus architecture in the first cluster, the gains are reduced. However, because of the overlap between the senders and receivers in our strategy, we still gain a ten percent improvement over the scratch method. When there is a lot of overlap, there is less data exchange during the redistribution. For 256 cores, we get even more improvement. We speculate that this is due to higher per-core data for redistribution in the case of fewer cores.

We saw a 4 percent increase in execution durations for our approach over the partition from scratch method for both actual and simulated test situations. This is because the Huffman tree isn't built from the ground up in our method, and we strive to maximise the overlap. As a result, the resulting divisions are not necessarily square. When the number of adaption points is large, however, it is more crucial to keep the redistribution cost low.

4.5 Distance between senders and receivers

For 70 simulated test cases on 1024 Blue Gene/L cores, Figure 8 illustrates the average hop-bytes during sender-receiver communication for partition from scratch versus our technique. The hop-bytes metric is the weighted sum of message sizes, where

the weights indicate the number of hops (links) each message has traversed. A larger hop-byte count indicates a higher network communication demand. The average in the case of partitioning from scratch is 5.25, whereas the average in our approach is 2.44. This is because the receiver process grid in our technique is situated closer to the sender process grid, reducing the number of hops between a sender and receiver pair.

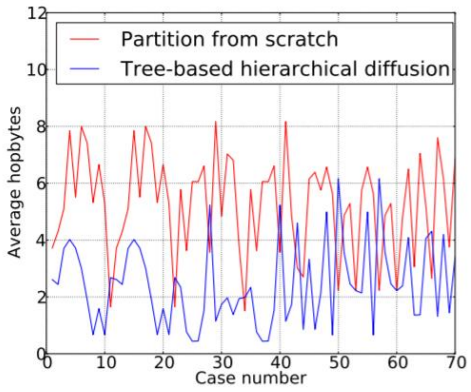


Figure 9: For both the partition from scratch method and the tree-based hierarchical diffusion strategy, the percentage overlap between senders and receivers is calculated.

For 70 synthetic test cases on 1024 Blue Gene/L cores, Figure 10 displays the percentage of data points overlap between senders and receivers for partitioning from scratch versus our technique. It can be seen that our method has a higher overlap, implying that our method requires less redistribution time.

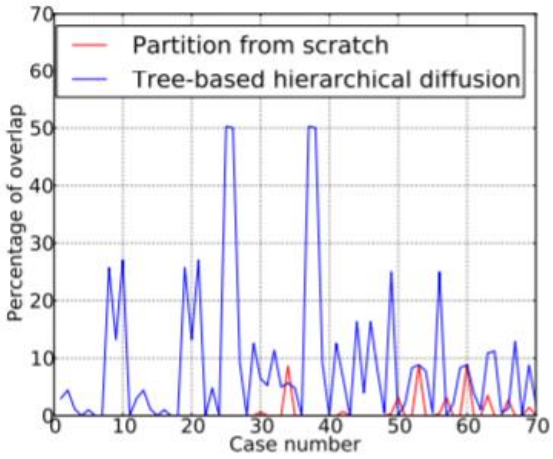


Figure 10: For the partition from scratch method and the tree-based hierarchical approach, the percentage overlap between senders and recipients is shown.

The X-axis represents the number of test cases, while the Y-axis represents the percentage overlap. The scratch method has greater overlap than the tree-based hierarchical approach.

The X-axis represents the number of test cases, while the Y-axis represents the percentage overlap. The scratch method has greater overlap than the tree-based hierarchical approach. For our tree-based hierarchical method, we discovered a 27 percent data point overlap between senders and recipients in the fist cluster. There was 15% overlap with the scratch method. This is because we strive to maximise the overlap between senders and recipients in our system so that data communication during redistribution is minimised.

4.6 Dynamic Approach

The findings of our dynamic method, which chooses between scratch and tree-based approaches, are presented in this section. For a 4-hour simulation session, we tested 12 reconfigurations for simulated situations on 1024 BG/L cores. The dynamic approach chose the approach with the shortest sum of expected execution and redistribution times. We computed the Pearson's correlation coefficient between the actual and expected execution times since the efficacy of the dynamic selection strategy is dependent on the ability to forecast the execution timings of different nest configurations. Pearson's correlation coefficient was 0.9 when we used our prediction

approach. This demonstrates a linear relationship between the two, indicating that our execution time forecast is nearly accurate.

Scratch method was chosen twice out of the 12 reconfiguration scenarios, whereas tree-based approach was chosen ten times. In ten of the twelve examples, the dynamic approach made the correct conclusion. In 9 situations, our tree-based diffusion method produced a lesser sum of execution and redistribution times than the partition from scratch method, while in the remaining three cases, the partition from scratch method produced a smaller sum.

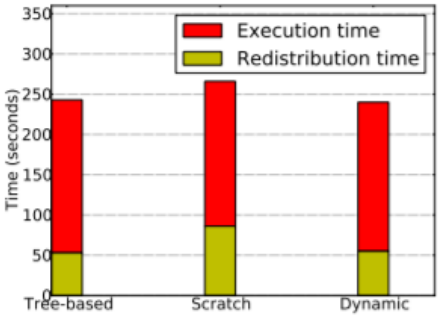


Figure 11: Time of execution and redistribution

Figure 10 shows the total time for the tree-based technique, partition from scratch method, and dynamic approach, including execution and redistribution times. It can be seen that our tree-based method takes the least amount of time to redistribute data, while the partition from scratch method takes the least amount of time to execute. The dynamic selection method combines the benefits of both methods, with redistribution times comparable to those of the tree-based method and execution times comparable to those of the partition from scratch method. The dynamic strategy reduced overall execution time by 3% compared to the next best-performing tree-based solution. It should be highlighted that in our real runs (around 70 adaptation points), more frequent adaptation points result in larger performance increase for the dynamic scheme.

5. CONCLUSION

To detect and monitor towering clouds in tropical weather systems, we presented a parallel data analysis approach and an effective processor reallocation algorithm in this paper. Using a form of closest neighbour clustering, our data analysis approach discovers ordered cloud networks. For the places with a lot of cloud cover, we ran layered high-resolution simulations. A disjoint subset of the entire number of processors was used to run the layered simulations. The nests may form and dissolve over time due to the dynamic nature of clouds. We devised a tree-based efficient processor allocation approach for persistent nests that has a low data redistribution cost.

Our method takes into account the current processor allocation and chooses a new subset of processors that has the most overlap with the rectangular subset of processors. In comparison to the partition from scratch strategy, we were able to reduce redistribution times by up to 25% with only a minor increase in execution times. We also devised a dynamic scheme to determine which of the two approaches, partitioning from scratch and our approach, is the best.

6. FUTURE SCOPE

Our tracking and detection techniques are rather generic. We hope to use these techniques in the future for other applications that need simultaneous tracking of several dynamic events.

REFERENCES

1. G. Gu and C. Zhang, “Cloud components of the Intertropical Convergence Zone,” Journal of Geophysical Research: Atmospheres, vol. 107, no. D21, pp. ACL 4–1–ACL 4–12, 2002. \
2. S. Kumar, Y. Sabharwal, R. Garg, and P. Heidelberger, “Optimization of All-to-all Communication on the Blue Gene/L Supercomputer,” in International Conference on Parallel Processing, 2008.
3. S. Sahany, V. Venugopal, and R. Nanjundiah, “The 26 July 2005 heavy rainfall event over Mumbai: numerical modeling aspects,” Meteorology and Atmospheric Physics, vol. 109, pp. 115–128, 2010.
4. IBM Blue Gene Team, “Overview of the Blue Gene/L system architecture,” IBM Journal of Research and Development, vol. 49, 2005.

5. H. Yu, I.-H. Chung, and J. Moreira, "Topology Mapping for Blue Gene/L Supercomputer," in Proceedings of the 2006 ACM/IEEE conference on Supercomputing.
6. A. Bhatele, G. Gupta, L. V. Kale, and I.-H. Chung, "Automated Mapping of Regular Communication Graphs on Mesh Interconnects," in International Conference on High Performance Computing, 2010.
7. Liu, Xiaoqiang&wu, Lifeng& Zhang, Fenxia& Huang, Guomin& Yan, Fulai& Bai, Wenqiang. (2021). Splitting and Length of Years for Improving Tree-Based Models to Predict Reference Crop Evapotranspiration in the Humid Regions of China. Water. 13. 3478. 10.3390/w13233478.
8. C. Lennard and G. Hegerl, "Relating changes in synoptic circulation to the surface rainfall response using self-organising maps", Climate Dynamics, Vol.44, No.3-4, pp.861-879, 2015.
9. Piyush Kapoor (2013) - Weather Forecasting Using Sliding Window Algorithm. Hindawi Publishing Corporation ISRN Signal Processing Volume 2013, Article ID 156540, 5 pages
10. Malakar, P., George, T., Kumar, S., Mittal, R., Natarajan, V., Sabharwal, Y., Saxena, V., &Vadhiyar, S.S. (2012). A divide and conquer strategy for scaling weather simulations with multiple regions of interest. *2012 International Conference for High Performance Computing, Networking, Storage and Analysis*, 1-11.
11. PreetiMalakar, et al (2011) - INST: An Integrated Steering Framework for Critical Weather Applications. Procedia Computer Science 00 (2011) 1–10
12. Reddy, S.S. Bat algorithm-based back propagation approach for short-term load forecasting considering weather factors. ElectrEng 100, 1297–1303 (2018). https://doi.org/10.1007/s00202-017-0587-2
13. P. Malakar, T. George, S. Kumar, R. Mittal, V. Natarajan, Y. Sabharwal, V. Saxena, and S. S. Vadhiyar, "A Divide and Conquer Strategy for Scaling Weather Simulations with Multiple Regions of Interest," in Proceedings of the 2012 ACM/IEEE conference on Supercomputing.
14. W. C. Skamarock and et al., "A Description of the Advanced Research WRF version 3," NCAR Technical Note TN-475, 2008.
15. S. Sinha and M. Parashar, "Adaptive System Sensitive Partitioning of AMR Applications on Heterogeneous Clusters," Cluster Computing, vol. 5, 2002.
16. A. Bhatele, G. Gupta, L. V. Kale, and I.-H. Chung, "Automated Mapping of Regular Communication Graphs on Mesh Interconnects," in International Conference on High Performance Computing, 2010.
17. Z. Lan, V. E. Taylor, and G. Bryan, "Dynamic load balancing of SAMR applications on distributed systems," in Proceedings of the 2001 ACM/IEEE conference on Supercomputing.