Dispersal Metrics for Non-Contiguous Processor Allocation

Jens Mache and Virginia Lo

CIS-TR-96-13
December 1996

Department of Computer and Information Science
University of Oregon

Dispersal Metrics for Non-Contiguous Processor Allocation

Jens Mache and Virginia Lo
Department of Computer and Information Science
University of Oregon, Eugene, OR 97403
{jens, lo}@cs.uoregon.edu

Abstract

Resource management is a key area in the drive to fully realize the performance potential of parallel and distributed systems. For the task of assigning a set of processors of a massively parallel processing (MPP) system to a given job, various processor allocation strategies have been proposed in the research community and are in use at supercomputing sites. With the advent of the class of non-contiguous allocation strategies, the allocation performance bottleneck shifted from fragmentation to message-passing contention.

This paper presents a method to estimate and minimize contention incurred by non-contiguous allocation strategies. Our approach is to analyze the spatial layout of dispersed nodes. Our contribution is a set of *dispersal metrics* that predict contention well and that are efficient to implement for a variety of interconnection topologies. We put the dispersal metrics to the test by comparing their contention estimates with measurements taken from a message-passing simulator. Our analysis and experiments consider different topologies of machines, a wide range of communication patterns and different workloads. Because our results show very high correlations between dispersal metrics and contention, we conclude that dispersal metrics have the potential to help evaluate and improve processor allocation strategies.

Keywords: massively parallel processing (MPP), resource management, processor allocation contention, dispersal, performance evaluation, performance metrics

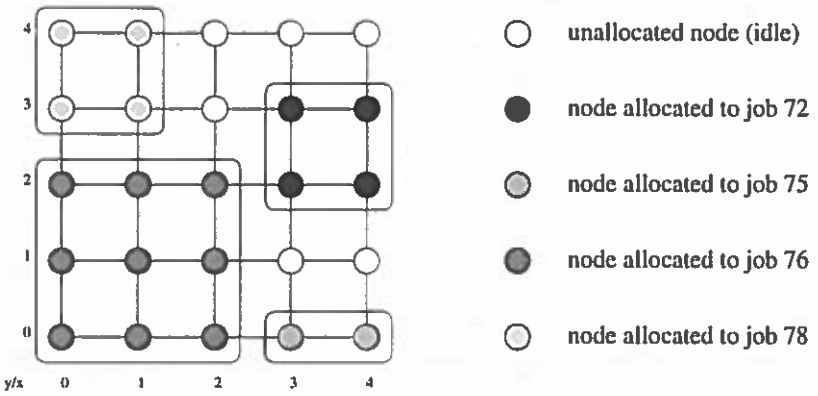

Figure 1: Snapshot showing contiguous allocations of four jobs

1 Introduction

Parallel jobs submitted to a massively parallel processing (MPP) system with n processors often need less than n processors [16]. This is exploited by modern systems that employ *space sharing* and run multiple jobs on different processors at the same time. Fig. 1 shows a snapshot of a 25-node machine and how the four jobs that are currently running share the processor space. A *processor allocation strategy* controls the assignment of a set of nodes/ processors to a scheduled job. The goal is to fully utilize the system and to maximize the throughput over a workload/ stream of jobs. In addition to space sharing, some systems use time sharing of nodes and a gang-scheduling scheme [6]. However, this paper will focus on the issues related to space sharing, namely processor allocation. Typical assumptions are that each job gets as many nodes as it requested and that jobs are not reallocated once they started running.

Various processor allocation strategies have been proposed in the research community and are in use at supercomputing sites. The class of *contiguous allocation strategies* restricts the nodes allocated to a given job to form a "convex" shape. Fig. 1 shows an example. Performance suffers significantly due to processors being wasted because of internal and external *fragmentation.* Utilizations of only 34% to 66% are reported [9, 18, 8, 10]. In contrast, the class of *non-contiguous allocation strategies* allocates nodes that are dispersed throughout the system. Fig. 3 shows examples. They experience no fragmentation and thus outperform contiguous strategies reaching utilizations of up to 78% [10, 15, 14]. To further improve the performance of non-contiguous strategies, it is necessary to select allocations that cause minimal *message-passing contention.* External contention is due to job interference: communication between dispersed nodes of a given job may require a communication link that is currently used by another job's messages. Contention affects the overall communication time of messages, the execution time of jobs, and ultimately job throughput of the machine.

In this paper, we present a method to estimate and minimize contention incurred by non-contiguous allocation strategies. Our approach is to analyze the spatial layout of dispersed nodes, referred to as dispersal. We devise several dispersal metrics and discuss their efficient implementations for different interconnection topologies. A *dispersal metric* measures the degree of dispersal of a given job's allocation and, ideally, reflects the increase in contention in the whole

system that is due to this allocation. To test the usefulness of dispersal metrics, we employ them to estimate contention and compare these estimates with measurements taken from a message-passing simulator. Our analysis and experiments consider different topologies of machines, a wide range of communication patterns and different workloads. Our results show that there is a very high correlation between dispersal metrics and contention. We conclude that dispersal metrics can be used to predict contention and thus have the potential to help evaluate and improve processor allocation strategies.

We focus on dimension-ordered routing and wormhole switching [13] (both are adopted in many existing machines) and on massively parallel processing (MPP) systems with direct interconnection networks having *mesh* or *k-ary n-cube* topologies. Intuitively, a k-ary n-cube is an n-dimensional mesh with wrap-around edges in each dimension. Hypercubes, tori and rings are special cases of k-ary n-cubes. Examples of existing machines with mesh or k-ary n-cube topologies are the Intel TFLOP, Intel Paragon, iPSC/860, Cray T3E and Cray T3D. We do not consider indirect networks (also known as multistage interconnection networks (MIN) or switch-based networks). Regarding contention, the experiments in [12] showed similar behavior for direct and indirect networks.

Section 2 covers processor allocation strategies and performance issues. In Section 3 through 5, we motivate and develop several dispersal metrics. Our experiments are described in Section 6. Section 7 concludes the paper and indicates future work.

2 Processor allocation strategies

2.1 Contiguous allocation strategies and fragmentation

Contiguous strategies assign only contiguous sets of nodes. Contiguity can be defined in the following way: All the nodes allocated to a job form a *"convex" shape*. One way to enforce this constraint is to require the nodes allocated to a job to form a subgraph of the original topology (e.g. a submesh in a mesh). Fig. 1 is an example of four jobs allocated contiguously in a two-dimensional mesh topology.

Contiguity in mesh topologies ensures that messages from different jobs do not interfere with each other even if the jobs run at the same time. This has the following advantages: Each job (or application) can think of owning the machine exclusively, and several runs of the same job result in approximately equal execution times.

Examples of contiguous allocation strategies for mesh topologies are 2D Buddy [9], Frame Sliding [5] and First Fit & Best Fit [18]. For hypercube topologies, examples are Gray Code [4], Partners [1] and Cyclic Buddy [11].

However, contiguous processor allocation strategies suffer from fragmentation. Consider an additional job in Fig. 1 that requests four nodes. Six nodes are unallocated at this time, but there is no contiguous block of four nodes available. The nodes (2,3), (2,4), (3,4) and (4,4) for example are not contiguous because they do not form a convex shape. This situation is called *external fragmentation*. In addition, *internal fragmentation* can arise when a job requests j nodes but receives $i > j$ nodes because of constraints of the machine or constraints of the allocation strategy. Both types of fragmentation result in unallocated nodes and thus low *utilization* of the

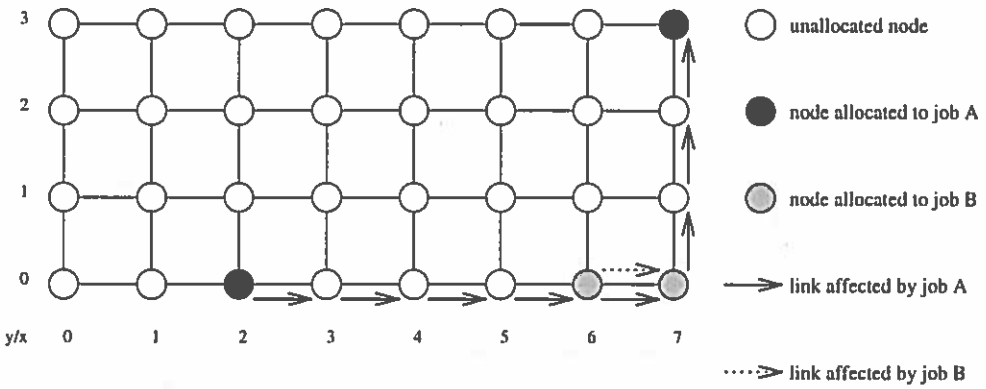

Figure 2: External contention for the link between (6,0) and (7,0)

machine. Using First-Come, First-Served scheduling, utilizations of only 34% to 66% are reported [9, 18, 8, 10] due to serious fragmentation problems. This is unfavorable and contradicts the goal of high throughput over a stream of jobs.

2.2 Non-contiguous allocation strategies and contention

To utilize unallocated nodes that are not necessarily contiguous, non-contiguous processor allocation strategies have been proposed [7, 10, 15, 14]. They are also known as scattered or fragmentation-free allocation strategies. Nodes allocated to the same job are allowed to be dispersed throughout the interconnection network. Fig. 3 shows examples.

What are the effects of *dispersed nodes*? On one hand, there are several advantages: Neither internal nor external fragmentation occurs, and this results in higher utilization of the machine. Moreover, non-contiguous strategies easily allow for adding another (possibly dispersed) node. This is important to two kinds of extensions, fault tolerance and dynamic allocation. In dynamic allocation, jobs are allowed to change (at runtime) the number of nodes they request, e.g. if their degree of parallelism changes.

On the other hand, dispersed nodes are typically further apart from each other. This in itself does not cause a problem, because using wormhole switching the network latency is almost independent of the path length. However, as discussed below, bigger physical distance increases the potential for message-passing contention.

Examples of non-contiguous strategies for mesh topologies are defined in [10] and are shown in Fig. 3: "Random" selects unallocated nodes randomly, "Paging" scans the topology in row-major order for unallocated nodes, and "Multiple Buddy Strategy (MBS)" allocates a job to (possibly several) contiguous blocks. [15] describes similar algorithms for k-ary n-cube topologies.

Message-passing contention occurs when several messages want to use the same communication link at the same time. Each channel/ link of the interconnection network can only be used by one message at a time, others have to wait. (Even with virtual channels, the more messages contend for a link, the longer it takes until all are through.) *External contention* is caused by two or more messages of different jobs contending for a link. Fig. 2 shows an example. Two communicating nodes of job A and two communicating nodes of job B would both like to use

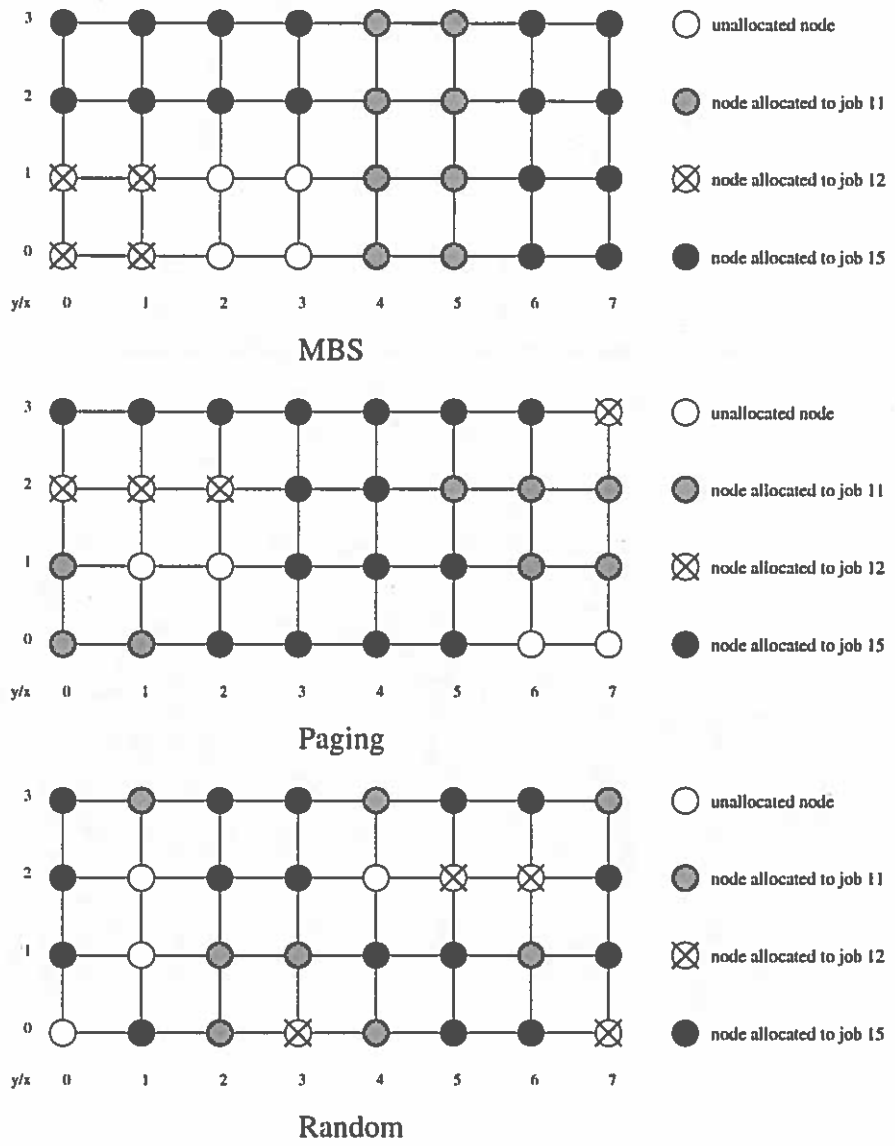

Figure 3: Snapshots of different non-contiguous strategies

the link between node (6,0) and node (7,0) at the same time. External contention occurs only if a non-contiguous allocation strategy is used (since different jobs do not interfere if a contiguous allocation strategy is used). In contrast, *internal contention* is caused by two or more messages of the same job contending for a link. It can occur no matter whether a non-contiguous or a contiguous allocation strategy is used. (In some cases, the dispersal of nodes due to a non-contiguous allocation may even reduce internal contention.)

The bottomline is that contention affects the overall communication time of messages, the execution time of jobs, and ultimately job throughput of the machine. Although contention has shown to be negligible in case of small messages [10, 12] or high software latency [12], it in general is the deciding performance factor within the class of non-contiguous processor allocation strategies [10, 12].

3 Motivation for dispersal metrics

As seen in Section 2, non-contiguous processor allocation strategies assign nodes that are possibly dispersed, thus achieving the best performance so far. For further improvement, it is important to study message-passing contention, the new performance bottleneck.

To our knowledge, previous research on contention in non-contiguous allocation strategies focused on the following: [10] studied whether the advantages of non-contiguity overcome the disadvantage of contention, [12] studied under what conditions contention matters and [14] implemented a non-contiguous strategy taking contention into account.

In this paper, we present a method to estimate and minimize contention under non-contiguous allocation strategies. Our contribution is a set of *dispersal metrics* that measure the degree of dispersal of a given job's allocation and, ideally, reflect the increase in contention in the whole system due to this allocation.

Our approach is to analyze the spatial layout of dispersed nodes. This is motivated by the following: First, the differences in dispersal are visually striking. Fig. 3 shows snapshots of different non-contiguous allocation strategies [10] servicing a given jobstream. All snapshots show the same jobs 11, 12 and 15. For the MBS strategy, the nodes allocated to jobs 11 and 12 are contiguous and the nodes allocated to job 15 form two convex rectangles. In contrast, the job allocations for the Paging strategy look "more dispersed", job 11 and job 12 consist of two clusters each and the nodes of job 15 are somewhat adjacent but do not form a convex shape at all. The job allocations for the Random strategy seem to be even "more dispersed".

Second, spatial layout obviously affects job interference. If in Fig. 2 the second node of job A had been (3,0) instead of (7,3), then there would have been no contention.

Third, known results suggested a relation: contiguous allocation strategies restrict the location of nodes allocated to a given job to form a convex shape and do not experience external contention, while non-contiguous allocation strategies do not restrict the location of the nodes allocated to a given job and do experience external contention. We are interested in whether non-contiguous strategies with "less dispersed" allocations are likely to experience less contention and how to define "less dispersed".

The final motivation is that dispersal is easy to compute, as will be discussed below. Per job dispersal metrics are based only on the addresses of the nodes allocated to the job in question. The addresses of those nodes are available at allocation time and are static over the lifetime of the job.

Sections 4 and 5 discuss different dispersal metrics and their efficient implementations. In Section 6, we test their ability to estimate contention.

4 Per job dispersal metrics

Per job dispersal metrics are calculated on a per job basis. Fig. 4 shows a snapshot of an 8 x 4 mesh topology with three allocated jobs. This example is used throughout Section 4. What is the degree of dispersal of each job? We discuss three different categories of per job dispersal metrics as well as algorithms to compute them. We consider two-dimensional mesh topologies first. Afterwards, we extend the algorithms so that they work for k-ary n-cube topologies as well.

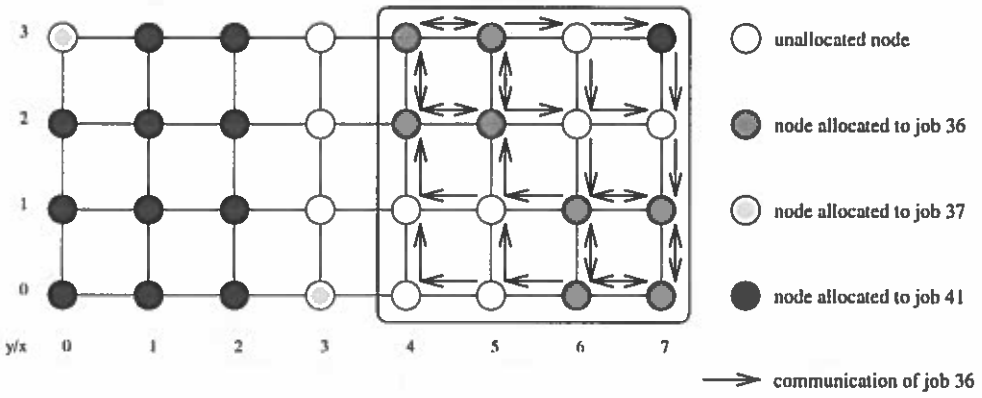

Figure 4: Enclosing rectangle and communication for job 36

The three categories are: nodes affected, links affected and distances & diameter.

4.1 Nodes affected

This metric is pretty simple: it represents the number of nodes in the system that may be *affected* if the *allocated* nodes (i.e. the nodes allocated to the job we look at) do an all-to-all communication. Why is this indicative of contention? The more nodes are affected, the more likely is external contention. If a *foreign* node (i.e. a node not allocated to this job) is affected and it belongs to another job that does communication as well, messages from both jobs are likely to contend, resulting in external contention. Fig. 4 shows the all-to-all communication of job 36. If foreign node (7,3) of job 41 wants to communicate with node (2,3) for example, external contention is likely to occur.

An algorithm that approximates the number of affected nodes (*nodes_affected*) finds the minimal enclosing rectangle that includes all the job's nodes and counts the number of nodes inside this rectangle. Let min_x be the minimal and max_x be the maximal x-coordinate of allocated nodes. Analogously, min_y and max_y are the minimal and maximal y-coordinates.

$$nodes\_affected = (max_x - min_x + 1) * (max_y - min_y + 1)$$

Fig. 4 and 5 show the enclosing rectangle and the communication for job 36 and 37, respectively. *Nodes_affected* for jobs 36, 37 and 41 is 16, 16 and 32, respectively. The enclosing rectangle for job 41 is the entire 8 x 4 mesh.

This algorithm is efficient. To calculate the enclosing rectangle, we look at each allocated node once and for each dimension record the minimal and maximal values seen so far. The algorithmic complexity for an n-dimensional mesh and jobsize j is $O(n * j)$. In contrast, the complexity of an exact count (that does the routing) would be $O(n * j^2 * k)$ where k is the maximal number of nodes in one dimension.

The enclosing rectangle includes all nodes that are affected. This is clearly due to minimal dimension-ordered routing. On the other hand, some nodes in the rectangle may not be affected at all. Fig. 5 shows job 37 as an example. Nodes (1,1), (1,2), (2,1) and (2,2) are inside the minimal enclosing rectangle, but they are not affected by communication of job 37. The

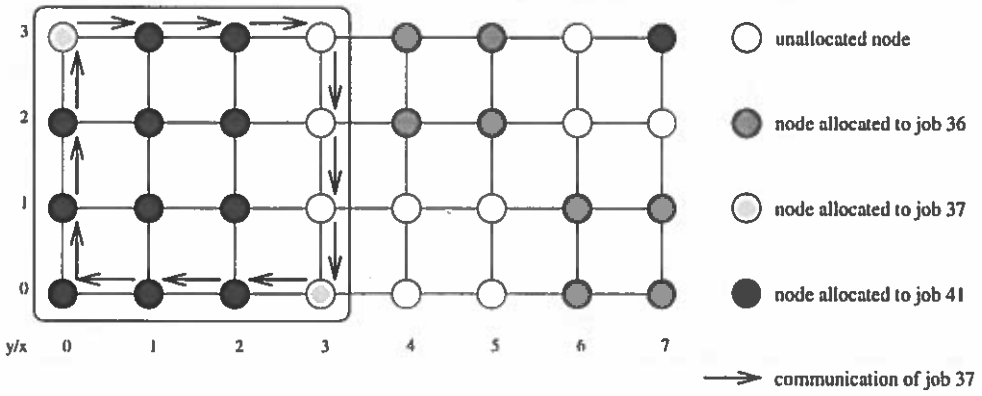

Figure 5: Enclosing rectangle and communication for job 37

enclosing rectangle sometimes overestimates the number of nodes that are really affected. The worst case for a k x k mesh topology is a situation with one node in the upper left and one node in the lower right corner: *nodes_affected* would be k^2, while the number of actual affected nodes would be roughly $4k$.

4.2 Links affected

This metric has less overestimate and links are more appropriate because messages typically contend for links, not for nodes. This time, we represent the number of links in the system that may be affected if the allocated nodes of a given job do an all-to-all communication. Is this approach indicative of contention as well? The argument from the previous metric applies similarly: The more links are affected, the more likely is external contention. A link from or to a foreign node indicates possible external contention.

An algorithm that approximates the number of those links (*links_affected*) projects the nodes allocated to the job into both dimensions and for each dimension records the minimal (min) and maximal (max) coordinate as well as the number of distinct coordinates ($count$) by using a bitvector.

$$links\_affected = (max_x - min_x) * count_y + (max_y - min_y) * count_x.$$

Fig. 6 shows the bitvectors and the communication for job 41. *Links_affected* for jobs 36, 37 and 41 is 24, 12 and 40, respectively. While *nodes_affected* for jobs 36 and 37 were both 16, this time, the values for *links_affected* differ, more accurately reflecting the actual interference potential.

This algorithm is efficient, its complexity for an n-dimensional mesh and jobsize j is $O(n * j)$ as well. It looks at each allocated node once and for each dimension records the minimal and maximal values seen so far as well as checks whether it has seen that coordinate before.

Why does it work? Take the communication of job 41 in Fig. 6 as an example and consider the links in x-dimension first. $(max_x - min_x)$ represents how many links it takes to route from max_x to min_x or vice versa. In how many rows does routing in x-dimension occur? In each row that has at least one allocated node. This equals to $count_y$. That is why $(max_x - min_x)$ is multiplied by $count_y$. The same argument holds for the links in y-dimension.

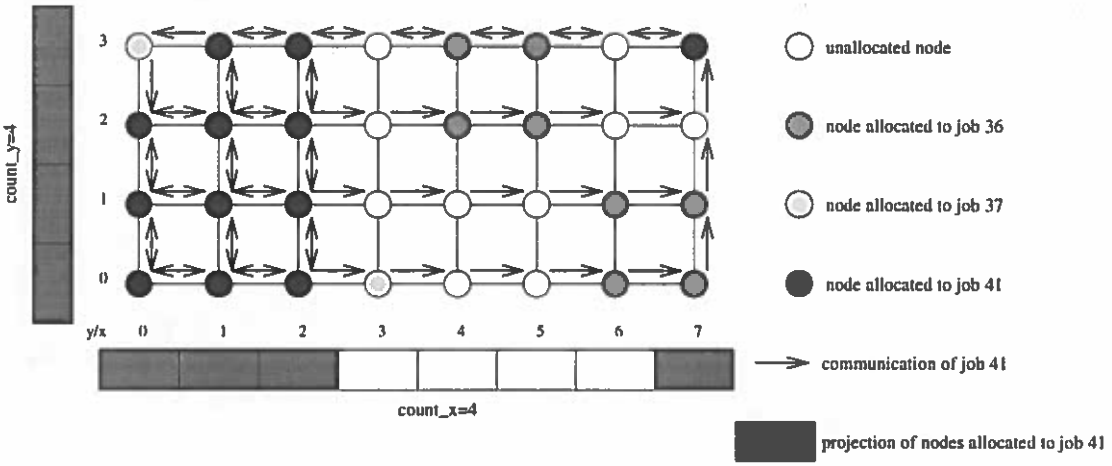

Figure 6: Bitvectors and communication for job 41

It is easy to see that this time there is no overestimate for mesh topologies. (The overestimate for k-ary n-cube topologies is marginal but can be seen in Fig. 7.)

4.3 Distances and diameter

This category of dispersal metrics is motivated by cluster analysis.

- *Average_distance* represents the average distance over all pairs of nodes allocated to the job.
- *Summed_distance* represents the sum of the distances between all pairs of allocated nodes. *Summed_distance* is different from *links_affected* in that it does a multiple count of affected links.
- *Distance_from_center* represents the sum of the distances of each allocated node to the allocated node that is most central, i.e. the allocated node for which this sum is minimal.
- *Diameter* represents the maximal distance between any two allocated nodes. *Diameter* is somewhat similar to *nodes_affected*, especially if the job allocation is rectangular.

Why is this indicative of contention? The distance between two nodes equals the path length if the pair communicates using minimal dimension-ordered routing. The longer the path, the more likely is contention.

The algorithm looks at all $j*(j-1)$ pairs (l,m) of allocated nodes, computes their $distance(l,m) = abs(l_x - m_x) + abs(l_y - m_y)$ and returns

$$average\_distance = \frac{\sum_{l=1}^{j}\sum_{m=1}^{j} distance(l,m)}{j*(j-1)}$$

$$summed\_distance = \sum_{l=1}^{j}\sum_{m=1}^{j} distance(l,m)$$

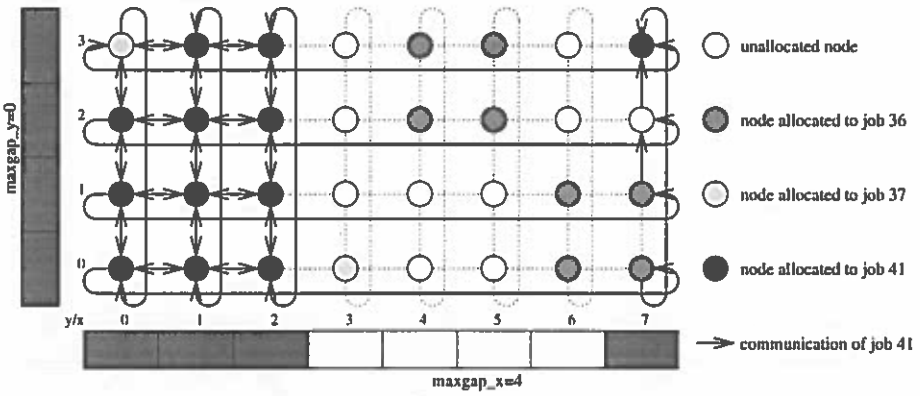

Figure 7: K-ary n-cube: *maxgap* and communication for job 41

$$distance\_from\_center = \min_{l}(\sum_{m=1}^{j} distance(l,m))$$

$$diameter = \max_{l,m}(distance(l,m))$$

These formulas are accurate. The algorithmic complexity for an n-dimensional mesh and jobsize j is $O(n*j^2)$ because we have to look at all pairs. This algorithm is more expensive than the previous two. But it is still efficient compared to low-level message-passing of a simulator. Also, the j^2 distances could be roughly approximated by computing the j distances to one randomly selected node only.

4.4 Extension to k-ary n-cube topologies

While extensions to higher dimensional mesh topologies are straightforward, we need some modifications for k-ary n-cube topologies. K-ary n-cube topologies differ from mesh topologies in that they have wrap-around edges. Dimension-ordered routing traverses one dimension at a time. To travel through one dimension, the message can travel in either direction (e.g. left or right in x-dimension) because of the wrap-around. The direction taken depends on which one takes fewer hops.

We need the following changes to the algorithms from the previous subsections so that they work for k-ary n-cubes topologies. For *nodes_affected* and *links_affected* in mesh topologies, we took *min* and *max* for each dimension because affected nodes or affected links were only found in between these extrema. With wrap-arounds, we change both algorithms to find the maximal gap *maxgap* in each dimension where there are no nodes or links affected. This is done by a linear scan of the bitvector that is constructed for each dimension. If *maxgap* is bigger than $k/2-1$ then nodes or links in this maximal gap are not affected (we set $realgap = maxgap$) because routing around this gap is always shorter than routing through it. If *maxgap* is not bigger than $k/2-1$ then all the nodes or links might be affected (we set $realgap = 0$). In the k-ary n-cube version of the first two algorithms, we have to change $max_i - min_i$ to $k - realgap_i - 1$. Fig. 7 shows a k-ary n-cube topology (for the sake of comparison to Fig. 6 it has 8 nodes in the first and 4 nodes in the second dimension). For job 41, $maxgap_x = 4 = realgap_x$ and $maxgap_y = 0 = realgap_y$

results in *nodes_affected* and *links_affected* having values of 16 and 28, respectively. The scanning of the bitvectors (of length k) changes the algorithmic complexity to $O(n * (j + k))$.

For our distance and diameter metrics, we computed the distance in each dimension by subtracting the coordinates. For k-ary n-cube topologies we have to take both directions into account. This is accomplished by using the Lee distance [3]. With the address of l and m being $l_1 l_2 ... l_n$ and $m_1 m_2 ... m_n$, resectively, we compute

$$lee\_distance(l, m) = \sum_{dim=1}^{n} min(abs(l_{dim} - m_{dim}), k - abs(l_{dim} - m_{dim})).$$

Other than that, we use the formulas from the previous subsection. The algorithmic complexity remains at $O(n * j^2)$.

5 Average dispersal

All previous dispersal metrics are computed for a single job ("per job dispersal metric"). *Average dispersal* is defined as the average value of a per job dispersal metric over a jobstream (workload) for a given allocation strategy ("dispersal metric per jobstream").

There are two arguments for average dispersal. First, it is typical to run a jobstream (to evaluate allocation strategies) because some previously allocated jobs are still running and constrain the current situation.

Second, the following two scenarios show the problems exhibited by dispersal metrics per job and how they are resolved by average dispersal. Dispersal metrics per job are static, whereas the processor allocation problem in general and contention in particular are highly dynamic. Dispersal metrics per job do not consider other jobs, they are blind-folded so to speak:

- To experience external contention, at least two different jobs have to interfere. A highly dispersed job A is likely to have a high value of per job dispersal metric. But if there is no other job to interfere with, external contention is zero.

- If a dispersed job B interferes with a contiguous job C, both experience external contention but only B is likely to have a high value of per job dispersal metric.

One approach to resolve these problems is to take a snapshot and average the per job dispersal metrics over all jobs that are running at this point in time. This helps in the second scenario, because although the contiguous job C doesn't expect contention, the job B does.

In our experiments, we go a step further and average over all jobs of the jobstream, instead of taking a lot of snapshots. This is more efficient and it also helps in the first scenario: It is very likely that there was another job in addition to job A a short time earlier or that there will be another job a short time later (in both simulation studies and the real world, machines normally have a high load) and contention was or will be occuring.

6 Experiments

The purpose of our experiments is to examine the ability of different dispersal metrics to estimate contention. We conducted two sets of experiments: The first set of experiments tests per job dispersal metrics. The second set of experiments uses average dispersal which considers the whole jobstream. We compare contention estimates based on dispersal metrics to contention measurements produced by a message-passing simulator. Our experiments consider both mesh and k-ary n-cube topologies of interconnection networks, a wide range of communication patterns and different workloads.

6.1 Simulation environment

To obtain contention measurements, we use ProcSimity [17], a discrete-event simulator. It models the arrival, service and departure of a stream of jobs in a multicomputer. It supports selected allocation and scheduling strategies on architectures with a range of network topologies and several routing and flow control mechanisms. The detailed message-passing behavior is simulated down to the level of individual flits and message-passing buffers. Contention is measured by recording the amount of time the header flit of each message is blocked in the network waiting for a channel to become free.

To get a variety of spatial layouts we employ different allocation strategies. For mesh topologies we use the non-contiguous strategies Random [10], MBS [10] and Paging [10] (with different page sizes and different indexing schemes) as well as the contiguous strategies First Fit [18], Best Fit [18] and Frame Sliding [5]. For k-ary n-cube topologies we use the non-contiguous strategies Random [15], MBS [15], Paging [15] and Multipartner [15] as well as the contiguous strategies Buddy [9], Gray Code [4] and Partner [1].

Our simulation model uses wormhole switching and minimal dimension-ordered routing (XY routing for mesh and Lee routing for k-ary n-cube topologies). Concerning architecture, we have two uni-directional links between adjacent nodes and either a 16 x 32 mesh topology or a 8-ary 3-cube topology. The scheduling policy is set to FCFS, because for testing dispersal metrics, we saw no significant differences when the scheduling algorithm was changed.

6.2 Workload

Realistic workload parameters are important. Both contention and the spatial layout of allocated nodes depends on workload characteristics. We set the workload parameters in our simulation according to workload parameters observed in traces of production machines (e.g. an Intel Paragon at the San Diego Supercomputing Center [16]). Our jobstreams consist of 1000 jobs. Jobsizes have an exponential distribution with mean of 16. For the purpose of non-empty waiting queues, we choose short interarrival times (Poisson distribution).

Five communication patterns are modeled: all-to-all broadcast, one-to-all broadcast, fast fourier transform (FFT) as well as multigrid benchmark and kernel CG benchmark from NAS parallel benchmarks [2]. These cover many typical communication patterns and the complexity ranges from $O(j)$ to $O(j^2)$, j being the jobsize. Fig. 8 shows all-to-all and one-to-all communication for a job of size 4 as well as the different communication phases of FFT for a job of size

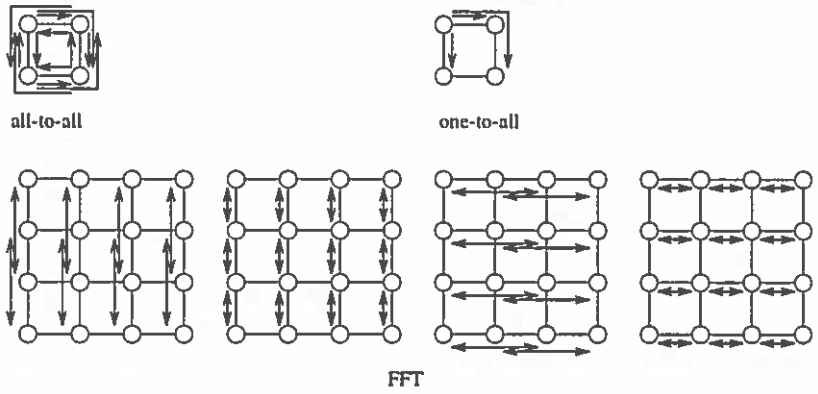

Figure 8: Communication patterns

16. If the communication pattern has different phases, barrier synchronization is used between the phases.

Since our focus is on contention, we consider communication only (no computation). To study contention, we use heavy communication loads which are designed to produce heavy contention. Message quota is a parameter to ProcSimity related to the communication load which in the current version is independent from the communication pattern. We keep it fixed so that a typical node sends on the average as many messages as there are other nodes in the same job.

6.3 Results

6.3.1 Per job dispersal

We conduct two sets of experiments. The first set of experiments uses *per job dispersal* metrics. We run one allocation strategy and one communication pattern. For each of the 1000 jobs of the jobstream we compute (for each dispersal metric) the correlation between dispersal metric and contention. Contention is the average blocking time of messages of that job as measured by the simulator.

For correlation computations, we use the Pearson correlation coefficient. Results reported represent the statistical mean after 20 simulation runs with identical parameters, and given 95% confidence level, mean results have less than 3% error. Additional details about the simulator are described in [10].

Table 1 shows the correlation results for a 16 x 32 mesh topology, MBS allocation strategy and two different communication patterns. Fig. 9 shows the scatter plot for all-to-all communication and the dispersal metric *nodes_affected*.

Correlation ranges from 0.22 to 0.50 and the points of the scatter graph do not fall in a straight line. This is due to the fact that dispersal metrics per job do not consider other jobs, while contention depends on them (see the scenarios in Section 5). Therefore, contention estimates by *per job dispersal* are likely to be either too high or too low if we look at single datapoints.

To see a possible trend, we group the datapoints with similar dispersal values into intervals, compute the average contention for each interval and connect these points. The resulting curve is shown in Fig. 9. It is increasing, i.e. on average, bigger dispersal correlates with bigger

	all-to-all	one-to-all
nodes_affected	.391583	.256183
links_affected	.444805	.225672
average_distance	.407117	.499982
distance_from_center	.499775	.279724
summed_distance	.501765	.507082
diameter	.361437	.222952

Table 1: Correlation for per job dispersal metrics (MBS strategy, 16x32 mesh)

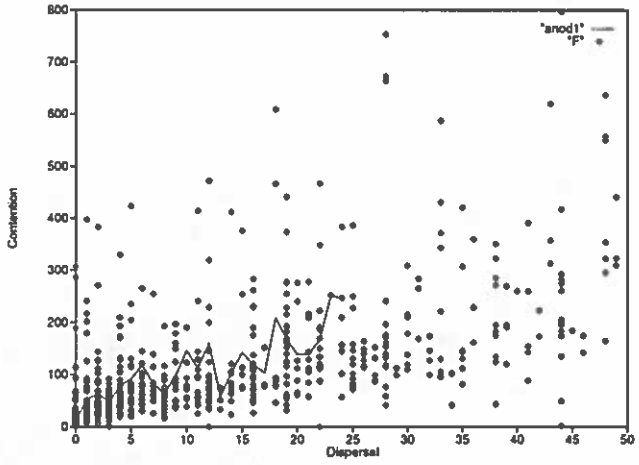

Figure 9: Scatter plot of per job dispersal metric (*nodes_affected*) vs. contention (all-to-all communication, MBS allocation strategy, 16x32 mesh)

contention.

Different communication patterns and different allocation strategies result in graphs very similar to Fig. 9. The same is true for k-ary n-cube topologies.

6.3.2 Average dispersal

The second set of experiments uses *average dispersal* (per jobstream). As discussed in Section 5, they resolve the problem of dispersal metrics per job being blind-folded.

Table 2 shows the correlation for an 16 x 32 mesh topology and five different communication patterns. Table 3 shows the correlations for a 8-ary 3-cube and two different communication patterns. The correlations are very high, varying between 0.890 and 0.998 overall. For both topologies the dispersal metric *summed_distance* has the highest correlation for all-to-all communication. For other communication patterns in mesh topologies, the dispersal metric *diameter* achieves the highest correlation. For one-to-all communication in the k-ary n-cube topology, the dispersal metric *links_affected* has the highest value.

Fig. 10 shows the scatter plot for *average diameter* and for all five communication patterns in one graph. For each communication pattern, the datapoints are close to a straight line. This visually shows the high correlations. The slopes of the regression lines vary for different

	all-to-all	one-to-all	FFT	MultiGrid	NASCG
nodes_affected	.910878	.996691	.985869	.986541	.992155
links_affected	.927287	.997315	.993006	.989280	.992311
average_distance	.913628	.998490	.990213	.991278	.995726
distance_from_center	.979404	.973173	.980911	.960587	.959212
summed_distance	.996203	.925024	.934681	.895492	.890912
diameter	.939838	.996631	.995940	.994470	.996535

Table 2: Correlation for average dispersal (16x32 mesh)

	all-to-all	one-to-all
nodes_affected	.959740	.990280
links_affected	.942668	.993962
average_distance	.953251	.991443
distance_from_center	.973884	.964315
summed_distance	.981271	.936519
diameter	.971608	.964080

Table 3: Correlation for average dispersal (8-ary 3-cube)

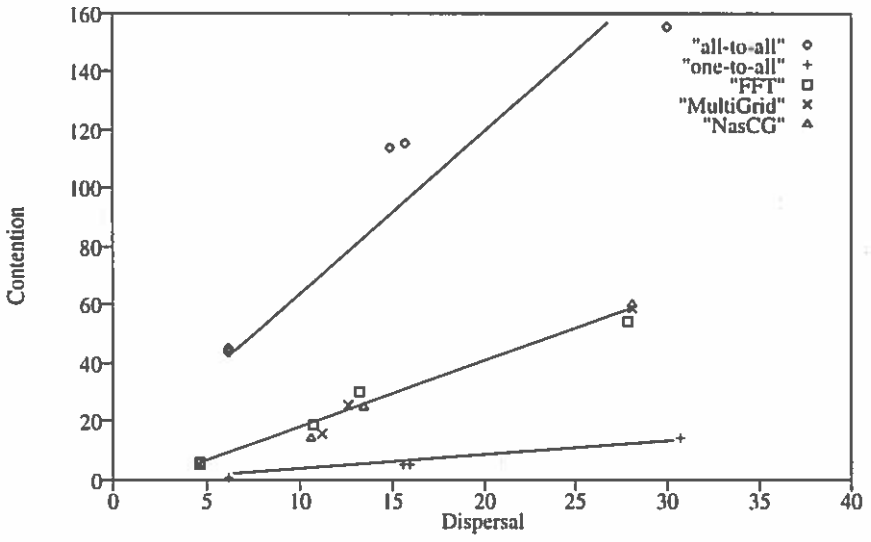

Figure 10: Scatter plot for five communication patterns (average dispersal metric *diameter*, 16x32 mesh)

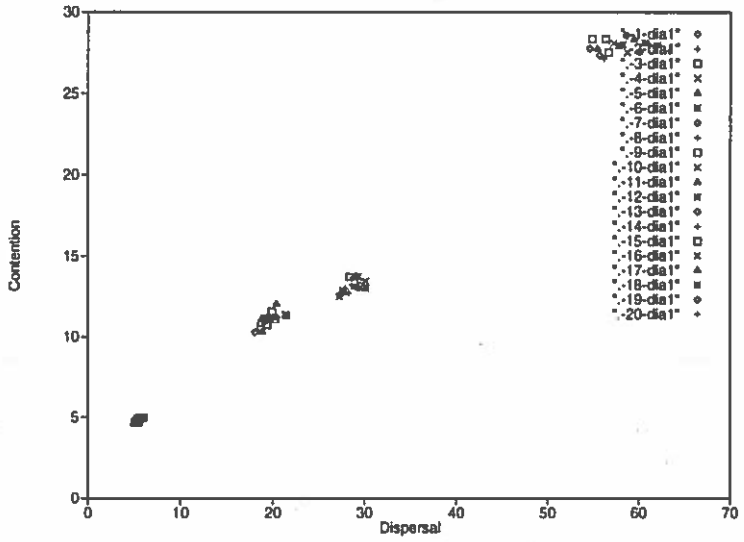

Figure 11: Scatter plot for 20 jobstreams (average dispersal metric *diameter*, 16x32 mesh, FFT communication)

communication patterns. All-to-all communication has the biggest slope and one-to-all shows the smallest slope. Other average dispersal metrics show similar graphs.

Fig. 11 shows the scatter plot for FFT communication, for *average diameter* and for 20 jobstreams that differ in the seeds for the probabilistic parameters. It shows four clusters, the lowest for the contiguous strategies First Fit, Best Fit and Frame Sliding, the second for MBS (non-contiguous), the third for Paging (non-contiguous) and the fourth for Random (non-contiguous) allocation strategy. In this graph, the slope of the regression line for different workloads is almost identical.

Fig. 12 shows rank correlation, the example chosen is all-to-all communication pattern and the average dispersal metric *links_affected*. Dispersal metric, measured contention and measured average service time rank the different allocation strategies in the same order. (First Fit, Best Fit and Frame Sliding have the lowest values, followed by Paging and MBS, Random is last.) Since dispersal metrics are much easier to obtain, they have the potential to replace costly low-level simulations.

In additional experiments, we reduce the average number of messages sent per job. Much less contention is measured. This results in lower correlations (0.08 to 0.61) and indicates that the ability of dispersal metrics to estimate contention is worse if there is only little contention. This is not bad, because we are more interested in estimation of contention if contention is a problem (i.e. high), not if contention is not a problem (i.e. low). Considering the case with non-marginal contention, we are on the safe side. If contention is marginal, dispersal metrics are still likely to prefer one allocation strategy over another, even if both strategies experience almost the same low contention.

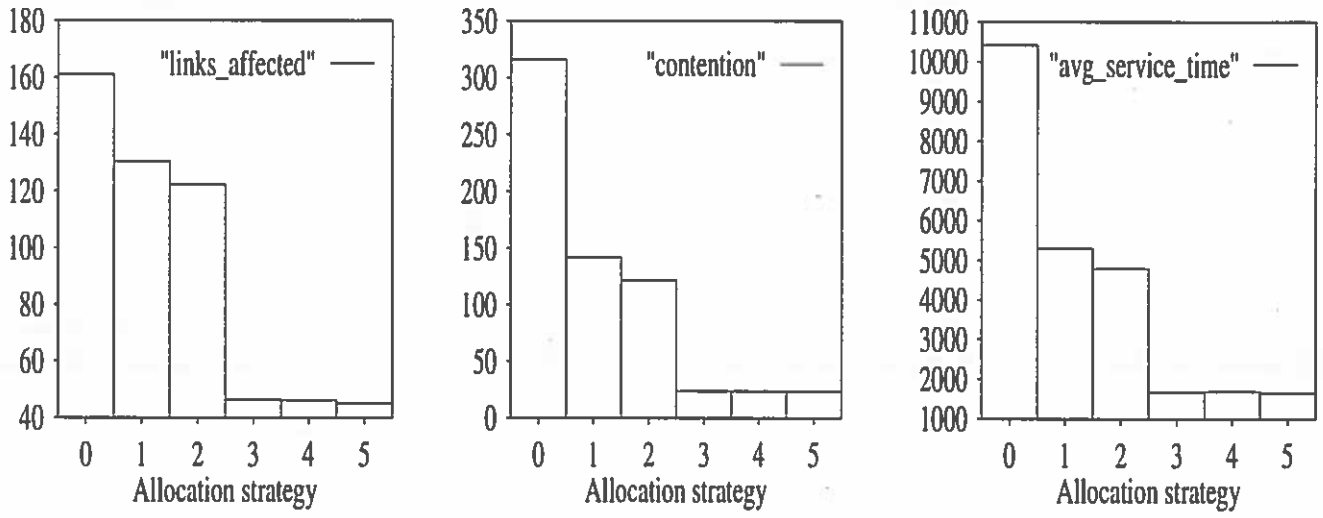

Figure 12: Rank correlation for six allocation strategies (average dispersal metric *links_affected*, 16x32 mesh, all-to-all communication)

7 Conclusions and future work

Non-contiguous processor allocation strategies assign nodes that are possibly dispersed, thus achieving the best performance so far. For further improvement, message-passing contention should be minimized. Our contribution towards this goal is a set of dispersal metrics that are based on the spatial layout of dispersed nodes. Our six dispersal metrics are called *nodes_affected*, *links_affected*, *average_distance*, *summed_distance*, *distance_from_center* and *diameter*.

To summarize our results:

- Our dispersal metrics are efficient to implement for mesh and k-ary n-cube interconnection topologies.
- Our dispersal metrics predict contention well. Our simulation experiments show that our dispersal metrics have very high correlations with message-passing contention (if communication is non-marginal). Correlations of contention estimated by average dispersal metrics (over a workload) with contention measured by the simulator range from 0.890 to 0.998 for a wide range of communication patterns and two network topologies (mesh and k-ary n-cubes).
- All six dispersal metrics perform well, the vast number of correlation coefficients are bigger than 0.9. However, our analysis and experiments show some differences among dispersal metrics. First, the algorithmic complexity for *nodes_affected* and *links_affected* is lower. Second, different average dispersal metrics perform best in estimating contention depending on communication pattern and network topology. For all-to-all communication *summed_distance* is best. An explanation might be that its design is tailored to all-to-all communication. For the other communication patterns in mesh topologies, *diameter* has the highest correlation.

- Dispersal metrics have the potential to help evaluate and improve non-contiguous processor allocation strategies. Selecting the job allocation causing minimal contention is necessary to improve non-contiguous allocation strategies. Being efficient to compute and having high correlation with contention, average dispersal metrics can be used to evaluate allocation strategies and thus in many cases replace costly low-level simulations. Being computed at allocation time and contributing to average dispersal, per job dispersal metrics can guide non-contiguous allocation strategies to help minimize contention and thus optimize overall performance.

Future work includes employing dispersal metrics to improve non-contiguous processor allocation strategies (controlling the spatial layout, so to speak). Investigating the impact of I/O nodes in the topology is a next step because they might introduce other patterns of contention and in general I/O often is a bottleneck. We also consider further experimentation with dispersal metrics. Additional experiments could investigate the cut-off point between marginal and non-marginal communication, the influence of message size distribution and mean message size as well as the influence of different jobs doing different communication patterns.

References

[1] A. Al-Dhelaan and B. Bose. A new strategy for processor allocation in an nCUBE multiprocessor. In *Proceedings of the International Phoenix Conference on Computers and Communication*, pages 114–118, March 1989.

[2] D. H. Bailey, E. Barszcz, L. Dagum, and H. D. Simon. NAS parallel benchmark results 3-94. Technical Report RNR-94-006, NASA Ames Research Center, Moffett Field, CA 94035-1000, March 1994.

[3] B. Bose, B. Broeg, Y. Kwon, and Y. Ashir. Lee distance and topological properties of k-ary n-cubes. Technical Report 93-60-11, Oregon State University, 1993.

[4] M. Chen and K. G. Shin. Processor allocation in an nCUBE multiprocessor using Gray codes. *IEEE Transactions on Computers*, C-36(12):1396–1407, December 1987.

[5] P. Chuang and N. Tzeng. An efficient submesh allocation strategy for mesh computer systems. In *Proceedings of the International Conference on Distributed Computer Systems*, pages 256–263, 1991.

[6] D. G. Feitelson. A survey of scheduling in multiprogrammed parallel systems. Technical Report RC 19790 (87657), IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY 10598, October 1994.

[7] H.-U. Heiss. Processor management in two-dimensional grid-architectures. Technical Report 20/92, Universität Karlsruhe, 1992.

[8] P. Krueger, T. Lai, and V. A. Dixit-Radiya. Job scheduling is more important than processor allocation for hypercube computers. *IEEE Transactions on Parallel and Distributed Systems*, 5(5):488–497, May 1994.

[9] K. Li and K.-H. Cheng. A two-dimensional buddy system for dynamic resource allocation in a partitionable mesh connected system. *Journal of Parallel and Distributed Computing*, 12:79–83, 1991.

[10] W. Liu, V. Lo, K. Windisch, and B. Nitzberg. Non-contiguous processor allocation algorithms for distributed memory multicomputers. In *Proceedings of Supercomputing '94*, pages 227–236, 1994. Best student paper award.

[11] M. Livingston and Q. F. Stout. Parallel allocation algorithms for hypercubes and meshes. In *Proceedings of the 4th Conference on Hypercube Concurrent Computers and Applications*, pages 59–66, 1989.

[12] S. Q. Moore and L. M. Ni. The effects of network contention on processor allocation strategies. Technical Report MSU-CPS-ACS-106, Michigan State University, 1995.

[13] L. M. Ni and P. K. McKinley. A survey of wormhole routing techniques in direct networks. *IEEE Transactions on Computers*, pages 62–76, February 1993.

[14] M. Wan, R. Moore, G. Kremenek, and K. Steube. A batch scheduler for the Intel Paragon MPP system with a non-contiguous node allocation algorithm. In *Proceedings of the 2nd Workshop on Job Scheduling Strategies for Parallel Processing, IPPS '96*, 1996.

[15] K. Windisch, V. Lo, and B. Bose. Contiguous and non-contiguous processor allocation algorithms for k-ary n-cubes. In *Proceedings of the International Conference on Parallel Processing*, 1995.

[16] K. Windisch, V. Lo, D. Feitelson, B. Nitzberg, and R. Moore. A comparison of workload traces from two production parallel machines. In *Proceedings of the Sixth Symposium on the Frontiers of Massively Parallel Computation*, 1996.

[17] K. Windisch, J. V. Miller, and V. Lo. ProcSimity: an experimental tool for processor allocation and scheduling in highly parallel systems. In *Proceedings of the Fifth Symposium on the Frontiers of Massively Parallel Computation*, pages 414–421, 1995.

[18] Y. Zhu. Efficient processor allocation strategies for mesh-connected parallel computers. *Journal of Parallel and Distributed Computing*, 16:328–337, 1992.