Communication Patterns and Allocation Strategies

David P. Bunde*
U. Illinois Urbana-Champaign
201 N. Goodwin Ave.
Urbana, IL 61801-2302
bunde@uiuc.edu

Vitus J. Leung
Sandia National Laboratories†
P.O. Box 5800
Albuquerque, NM 87185-1110
vjleung@sandia.gov

Jens Mache
Lewis & Clark College
0615 S.W. Palatine Hill Rd
Portland, OR 97219-7899
jmache@lclark.edu

Abstract

Motivated by observations about job runtimes on the CPlant system, we use a trace-driven microsimulator to begin characterizing the performance of different classes of allocation algorithms on jobs with different communication patterns in space-shared parallel systems with mesh topology. We show that relative performance varies considerably with communication pattern. The Paging strategy using the Hilbert space-filling curve and the Best Fit heuristic performed best across several communication patterns.

1. Introduction

To increase the scalability of commodity-based supercomputers, Sandia National Laboratories is developing the Computational Plant or Cplant [4, 28]. Although Sandia maintains a diverse set of computing resources, the tools for managing these resources commonly rely on scheduling/queuing software such as NQS [7] or PBS [26] to determine which of the available jobs should be run next. This decision is based on several factors, largely motivated by fairness and policy enforcement such as the job owner's past use of computing resources, number of processors requested, running-time estimates, waiting time, and even day of week and time of day.

When a job is selected to run, it is passed to the allocator, which must immediately assign it to a set of processors. On Cplant systems, these processors are exclusively dedicated to this job until it terminates. The allocator is a separate module from the scheduler and has no control over it.

The quality of an allocator is ultimately judged by the throughput of the managed system. Since job placement affects the system's network contention, allocation is a major factor in determining system performance, particularly in commodity-based supercomputers such as Cplant, which typically have higher communication latencies and lower bandwidth than supercomputers with custom networks. Experiments on Cplant showed that poor allocation could increase the running time of a pair of high-communication jobs by as much as a factor of two [18]. Other researchers have shown in a variety of studies that interprocessor communication can reduce throughput [2, 21, 22, 25].

Unfortunately, while it is easy to agree that some allocations are "very good" and others are "very poor", the allocations in the middle are hard to judge. Progress in this direction was made by Mache and Lo [21, 22], who proposed various metrics, including *average number of communication hops* between the processors of a job. This metric was supported by the experiments of Leung et al. [17], as shown in Figure 1, which plots running time versus the average distance between processors assigned to a job. Each plotted job uses 30 processors and performs a communication test consisting of all-to-all broadcast, all-pairs ping-pong (message sent in each direction), and ring communication. Each of these patterns is repeated one hundred times.

Based on these experiments, it was decided to switch the Cplant allocators from the 1-dimensional scheme described in Section 2.1 to MC1x1, described in Section 2.3, which explicitly tries to minimize the pairwise distance metric. However, more recent observations on Cplant suggest that, while all-to-all broadcast jobs seem to complete faster under MC1x1 than the 1-dimensional scheme, ring communication jobs seem to complete faster under the 1-dimensional scheme [13]. Perhaps the all-to-all broadcast dominated Leung et al.'s communication test suite.

This paper discusses the results of trace-driven microsimulations attempting to determine the interaction between allocation algorithms and communication patterns. In Section 2, we briefly survey known allocation algorithms and describe those used in our simulations. In Section 3, we give the details of the simulations. In Section 4, we present

*Partially supported by NSF grant CCR 0093348.

†Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

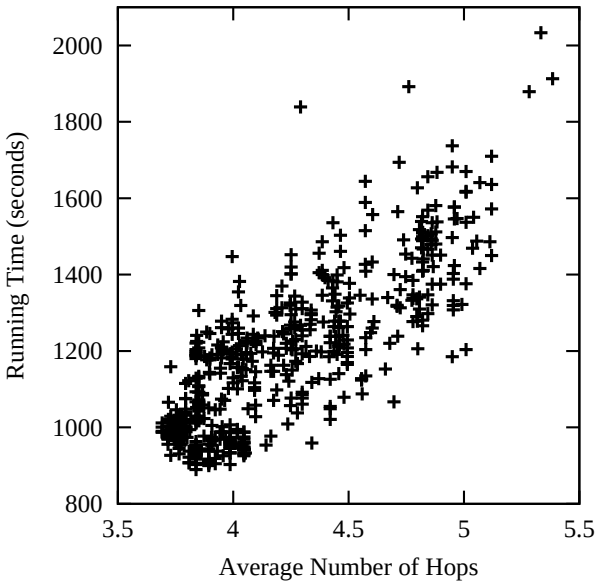


Figure 1. Relationship of pairwise distance and running time for large jobs in test suite.

the results and then we discuss them in Section 5.

2. Allocation Algorithms

Now we briefly summarize processor allocation research, paying special attention to the algorithms used in our simulator. Initial processor-allocation algorithms allocated only convex sets of processors to a job [3, 6, 19, 33]. Doing so has the potential to eliminate interjob communication contention if each job's communication is routed entirely within the set of processors assigned to that job. Unfortunately, requiring that jobs be allocated to convex sets of processors reduces system utilization to levels unacceptable for any government-audited system [14, 29].

More recent work [5, 15, 17, 20, 23, 29] allows noncontiguous allocations, attempting to cluster processors and minimize contention with previously-allocated jobs. Noncontiguous allocations allow jobs to be allocated whenever enough processors are available, but they greatly increase the number of possible allocations that must be considered.

2.1. One-Dimensional Reduction

One of the first algorithms proposed for noncontinuous processor allocation was Paging by Lo et al. [20]. In this algorithm, the processors of a machine are subdivided into $2^s \times 2^s$ submeshes called *pages*, where s is the *page size*. A sorted free list of pages is maintained and incoming jobs are assigned a prefix of the list with appropriate size. This algorithm can result in *fragmentation* if some free processors cannot be used because they are contained in pages that have been allocated to a job. To avoid fragmentation, we consider only $s = 0$, making each page a single processor.

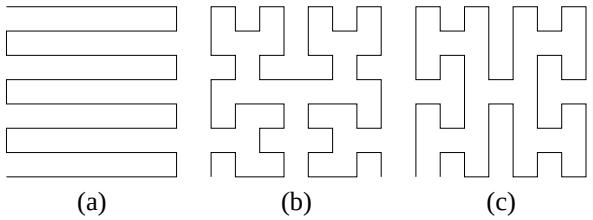


Figure 2. (a) S-curve (b) Hilbert curve (c) H-indexing

Specification of the Paging algorithm also requires giving the ordering of the pages. Lo et al. considered several page orderings, including row-major and s-curve (see Figure 2(a)). Leung et al. [17] independently developed an algorithm similar to Paging, but they proposed using space-filling or fractal curves to order the pages. These curves are recursively defined and are known to preserve several measures of "locality" [10, 24]. Two-dimensional space-filling curves include Hilbert curves [11] (see Figure 2(b)) and H-indexing [27] (see Figure 2(c)). Higher dimensional space-filling curves are discussed by Alber and Niedermeier [1]. For non-mesh machines, Leung et al. developed an integer program to find curves with locality properties.

In addition to using different page orderings, the algorithm of Leung et al. differs from Paging by selecting pages using *bin-packing* heuristics rather than a sorted free list. Their adaptation of these heuristics views each maximal interval of free processors with contiguous ranks as a partially-filled "bin". When no bin contains enough free processors to satisfy the incoming request, the set of processors with the smallest range of ranks along the curve is allocated. Otherwise, the First Fit algorithm allocates processors to a job from the first bin that is large enough and the Best Fit algorithm allocates processors from the bin that will have the fewest processors remaining. Both of these algorithms are close analogs to algorithms for bin packing proposed by Johnson [12]. Leung et al. also considered a variation of the more-complicated Sum-of-Squares Algorithm [9, 8], but this algorithm did not seem to perform as well in the processor allocation setting. The experiments of Leung et al. indicate that the choice of curve is more important than the algorithm used to select processors along the curve, but that both choices affect machine performance.

2.2. Gen-Alg

Krumke et al. [15] consider the discrete problem of selecting a subset of k points from a set of n points to minimize their average pairwise distance. They describe an algorithm Gen-Alg, given in Figure 3 and prove it is a $(2-2/k)$-approximation. This approximation ratio depends only on the triangle inequality, so it holds even in non-mesh architectures.

For each possible point p do:

1. Take the $k-1$ points closest to p.
2. Compute the total pairwise distance between all k points.

Return the set of k points with smallest pairwise distance.

Figure 3. Algorithm Gen-Alg

2.3. MC

The processor allocation algorithm proposed by Mache et al. [23], called MC, assumes that jobs request processors in a particular shape, such as a 4×6 submesh. Each free processor evaluates the quality of an allocation centered on itself. It does so by counting the number of free processors within a submesh of the requested size centered on itself and within "shells" of processors surrounding this submesh. The processors are weighted by the shell containing them; 0 for the initial submesh, 1 for the first shell out, 2 for the second, and so on. The sum of the weights gives the cost of the allocation. The algorithm chooses the allocation with lowest cost. This is illustrated in Figure 4, reproduced from Mache et al. [23].

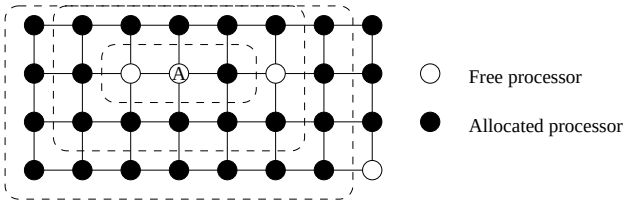


Figure 4. Illustration of MC: Shells around processor A for a 3×1 request.

The MC algorithm cannot be directly applied to Cplant because its users do not request processors in a particular shape. Thus, we consider a variant that we call MC1x1. In this variant, shell 0 is a 1×1 submesh and subsequent shells grow in the same way as in MC. The work of Krumke et al. [15] implies that MC1x1 is a $(4-4/k)$-approximation for average pairwise distance on k processors.

3. Simulation

For our simulations, we used ProcSimity [32, 30], a simulator designed to compare processor scheduling and allocation algorithms. Since our focus is on allocation rather than scheduling, we scheduled using First Come, First Serve (FCFS) in all our simulations. ProcSimity models communication at the flit level, allowing it to measure how network contention affects machine throughput.

3.1. Trace

We used a trace-driven simulation. The trace consists of all jobs submitted to the 352-node NQS partition of the Intel Paragon at the San Diego Supercomputer Center during the last three months of 1996 [31]. It consisted of 6087 jobs with the following statistical characteristics: the mean interarrival time was 1301 seconds, with a coefficient of variance of 3.7; the average job size was 14.5 nodes, with a coefficient of variance of 1.5, and with the distribution heavily favoring sizes that are powers of two; the mean job runtime was 3.04 hours with a coefficient of variance of 1.13.

3.2. Communication

Rather that assigning specific durations to our jobs, we specified that each job sends one message per second of trace run time. When these messages have all arrived, the job terminates. We varied the message intensity by contracting all job arrival times by a *load factor*, taking values 1, 0.8, 0.6, 0.4, and 0.2 so that effective system load increases by up to a factor of 5.

In each experiment, we assume that all jobs use the same communication pattern. While not realistic, this maximizes the interaction between the pattern and the allocation algorithm. We considered the following communication patterns, repeated as necessary to meet the message quotas for each job:

All-to-all In the all-to-all pattern, each processor sends a message to all other processors running the same job.

N-body In the n-body pattern, the processors assigned to a job form a virtual ring. For a job using p processors, each processor sends a message to its successor in the ring in

each of $\lfloor p/2 \rfloor$ *ring subphases* (see Figure 5(a)) and then sends a message to the processor halfway across the ring during a single *chordal subphase* (see Figure 5(b)). This pattern represents an algorithm for computing interparticle forces: Each processor "owns" a set of particles and keeps a copy of them at all times. Another copy migrates around the ring during the ring subphases. Between subphases, each processor computes the forces between the particles it owns and migrating particles currently at that processor. The chordal subphase accumulates all the forces acting on each particle at its owning processor, which can then update their positions before the next time step.

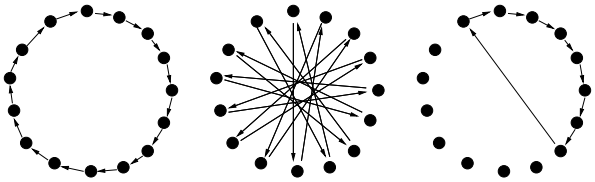


Figure 5. Messages sent during an N-body calculation with 15 processors. (a) Messages during ring subphase. (b) Messages during chordal subphase. (c) Migration of data away from owning node and back.

Random In the random pattern, each message goes between a random pair of processors assigned to the job.

4. Results

We performed simulations on two mesh sizes. The first, 16×22, was selected to closely match the size of the machine which generated the trace. However, having dimensions of differing sizes potentially cause problems for the Paging algorithms. The Hilbert and H-indexing space-filling curves are described having a square shape where the dimensions are a power of two. To get a curve for the 16×22 machine, we truncated a 32×32 curve to the appropriate size. The result is "curves" with gaps along the top edge, as shown in Figure 6. Although the S-curve does not become discontinuous on a non-square mesh, such a mesh presents the choice of whether the long part of each curve will move in the longer or shorter direction. Quick simulations seemed to indicate that the short direction is better so we used this convention. However, because of these issues with non-square meshes, we also performed simulations on a 16×16 mesh, using the same trace except for removing 3 jobs of 320 nodes each that are too large to fit the smaller machine.

Rather than showing all the data and cluttering our graphs, we omit the results of the Paging algorithms running First Fit. Generally speaking, First Fit was intermediate in performance between free list and Best Fit, though we note some important exceptions to this in the following section.

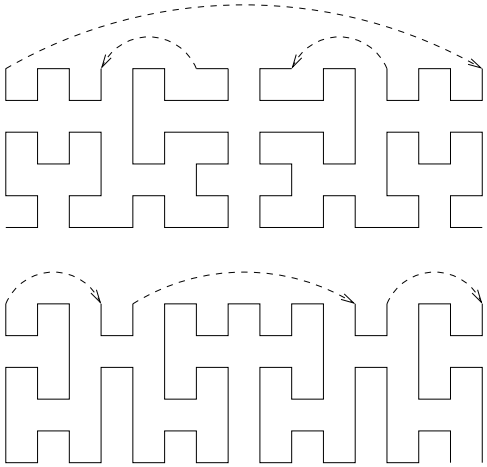


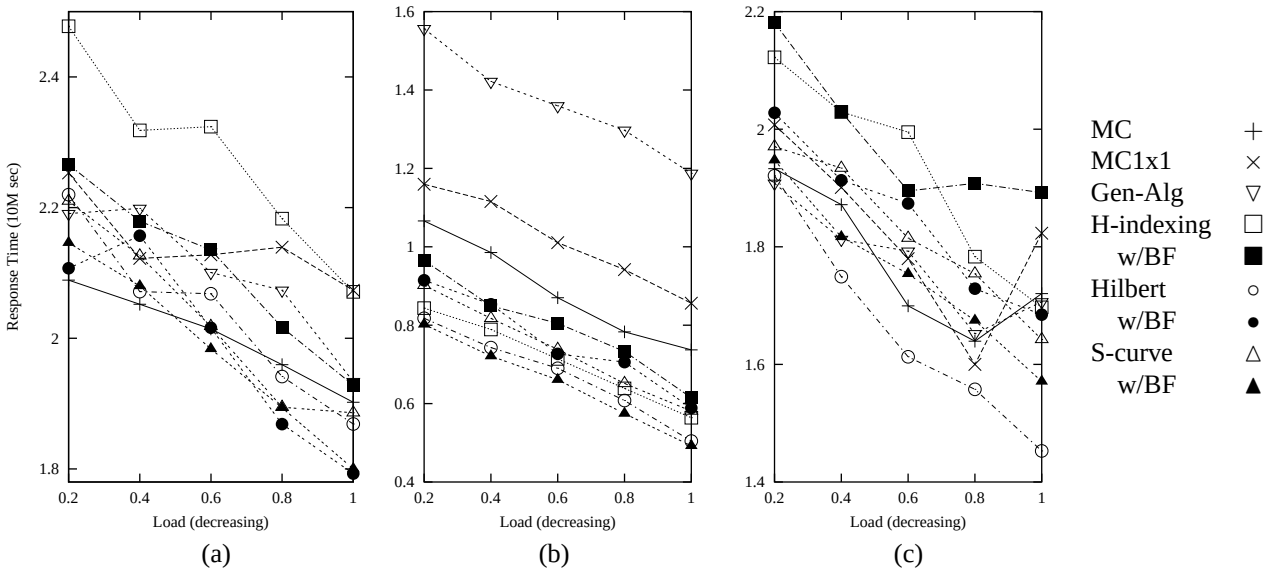
Figure 6. Top 16×6 processors in Hilbert curve (top) and H-indexing (bot) on 16×22 mesh. Arrows indicate the processor after a gap

4.1. 16×22 Mesh

Figure 7(a) shows the results for all-to-all communication on a 16×22 mesh. The horizontal axis is the load factor by which we multiplied job interarrival times. The vertical axis is job response times in seconds. The response time of a job is its completion time minus it arrival time, or the total time it spent in the system. The ordering of the algorithms varies considerably in this graph. The best algorithm at high load is MC and at lower loads it is either Hilbert with Best Fit or S-curve with Best Fit. H-indexing with free list is consistently the worst. More generally, the Paging algorithms perform better with a packing algorithm than with the free list. Among the other algorithms, MC is best, followed by Gen-Alg, and then MC1x1.

Figure 7(b) shows the results for n-body communication on a 16×22 mesh. The algorithms are almost completely ordered best to worst as follows:

1. S-curve with Best Fit,
2. Hilbert with free list,
3. H-indexing with free list,
4. S-curve with free list,



(a) (b) (c)

Figure 7. Results for trace on 16×22 mesh for various communication patterns. (a) All-to-all (b) N-body (c) Random

5. Hilbert with Best Fit,
6. H-indexing with Best Fit,
7. MC,
8. MC1x1, and
9. Gen-Alg.

Although Hilbert and H-indexing with free list beat Best Fit, First Fit (not shown) ran faster than free list or Best Fit.

Figure 7(c) shows the results for random communication. Again, the ranking is muddled. Hilbert with free list is consistently the best. H-indexing is consistently the worst, with free list generally better than Best Fit. (As in the previous case, Hilbert and H-indexing with First Fit (not shown) ran faster than free list and Best Fit.) MC, MC1x1, and Gen-Alg were similar, with the order depending on load.

4.2. 16×16 Mesh

Figure 8(a) shows the results for all-to-all communication on a 16×16 mesh. The algorithms are ordered best to worst as follows:

1. MC and Hilbert with Best Fit,
2. Gen-Alg, MC1x1, and H-indexing with Best Fit,
3. Hilbert with free list,
4. S-curve with Best Fit,
5. H-indexing with free list and S-curve with free list.

Looking for a specific shape seems to yield an advantage to MC over MC1x1, and the 1-dimensional strategies seem to be sensitive to the curve used; Hilbert with Best Fit is tied for the best, but S-curve always performs poorly.

Figure 8(b) shows the results for n-body communication on a 16×16 mesh. The algorithms are ordered best to worst as follows:

1. Hilbert with Best Fit,
2. H-indexing with Best Fit,
3. Hilbert with freelist,
4. H-indexing with freelist,
5. S-curve, first with Best Fit and then free list,
6. MC,
7. MC1x1, and
8. Gen-Alg.

Figure 8(c) shows the results for random communication on a 16×16 mesh. The algorithms are ordered best to worst as follows:

1. Hilbert with Best Fit,
2. MC and H-indexing with Best Fit,
3. MC1x1, Gen-Alg, and Hilbert with freelist,

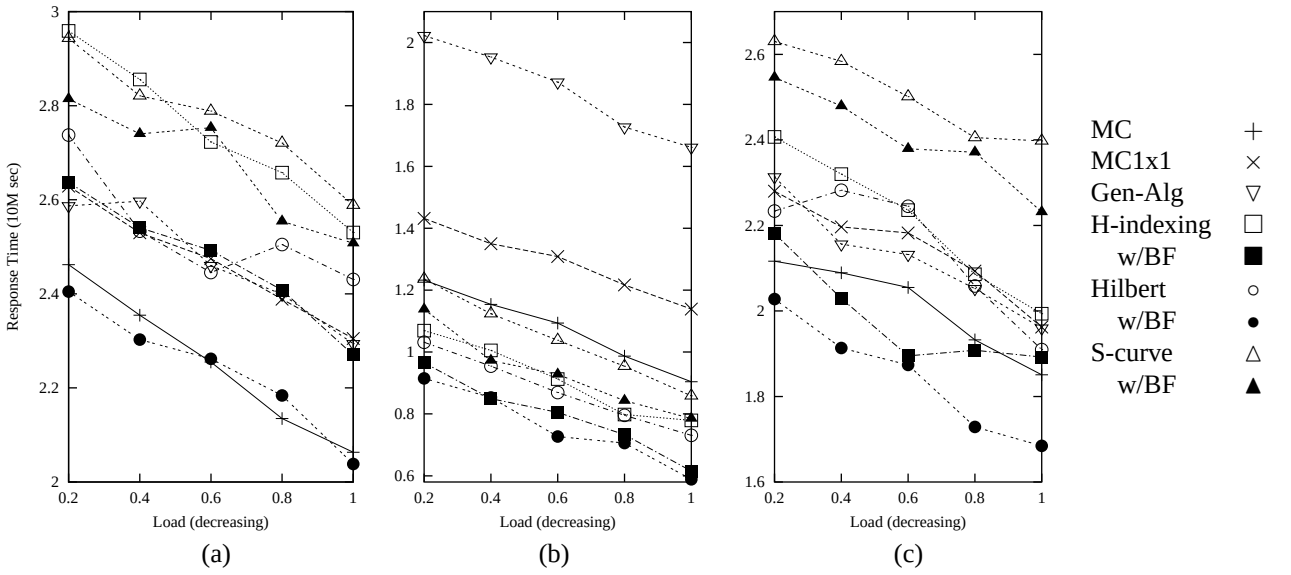


Figure 8. Results for trace on 16×16 mesh for various communication patterns. (a) All-to-all (b) N-body (c) Random

4. H-indexing with freelist,
5. S-curve with Best Fit, and
6. S-curve with freelist.

Although the specific communication pattern depends on random choices, one might expect the random pattern to be similar to all-to-all since all pairs are equally likely to communicate. Surprisingly, while Hilbert with Best Fit does well in both and S-curve does poorly, MC does better for all-to-all and H-indexing does worse.

4.3. Correlation to Alternate Metrics

We also used our simulations to test the correlation between job running time and the average distance between its processors. On the square mesh running n-body communication, we considered instances of the largest jobs (128 processors) sending between 39,900 and 44,000 messages. This range gave us 24 jobs in each simulation sending approximately the same number of messages. As shown in Figure 9, there is no clear relationship between pairwise distance and running time for these jobs. There is however a reasonably tight relationship, shown in Figure 10, between running time and average message distance, the average distance traveled the messages of a job. Plotting running time against total message distance gives a similar graph.

Another metric to consider is how often jobs are allocated to contiguous processors. More formally, we say that a set of processors assigned to a job form a *component* if there is a rectilinear path between any pair of them through processors assigned to that job. A job is allocated *contiguously* if all its processors form a single component. (Note that such a job may still interfere with others since messages use x-y routing rather than arbitrary paths.) Figure 11 shows the percentage of jobs allocated contiguously and the average number of components into which jobs were allocated. Observe that the curve-based strategies allocate into fewer components than the others. Unfortunately, neither of the metrics seems to capture the behavior observed in simulations.

5. Discussion

Although our experiments are too few to fully understand the interaction between the allocation algorithm and the jobs communication patterns, we do see some trends. The most dramatic difference occurs for MC, which is among the best for all-to-all, but gives relatively poor performance for n-body. S-curve and H-indexing are somewhat the reverse, being among the worst for all-to-all and better for n-body. The ideal would be an algorithm that performs well for all communication patterns. The closest to an overall best algorithm is Hilbert with Best Fit. It is among the best for all patterns on the 16×16 mesh and the all-to-all pattern on the 16×22 mesh.

There are differences between the results on the 16×16 mesh (Figure 8) and the 16×22 mesh (Figure 7). The discontinuities in the Hilbert and H-indexing curves may explain why only the S-curve is always improved by the

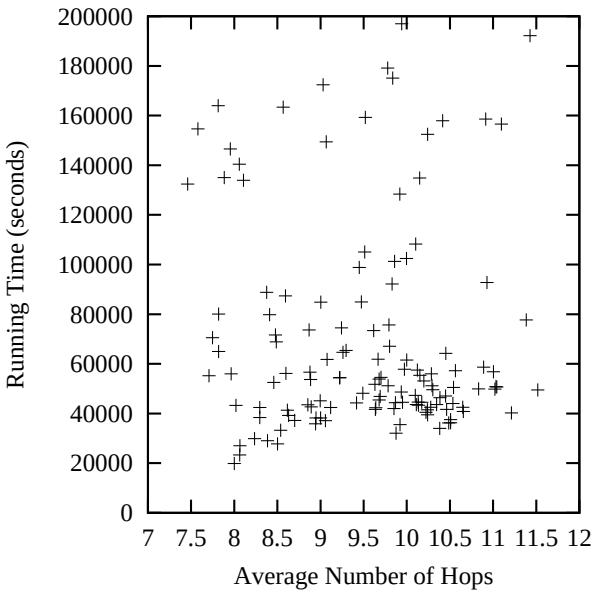


Figure 9. Relationship of processor pairwise distance and running time for n-body communication.

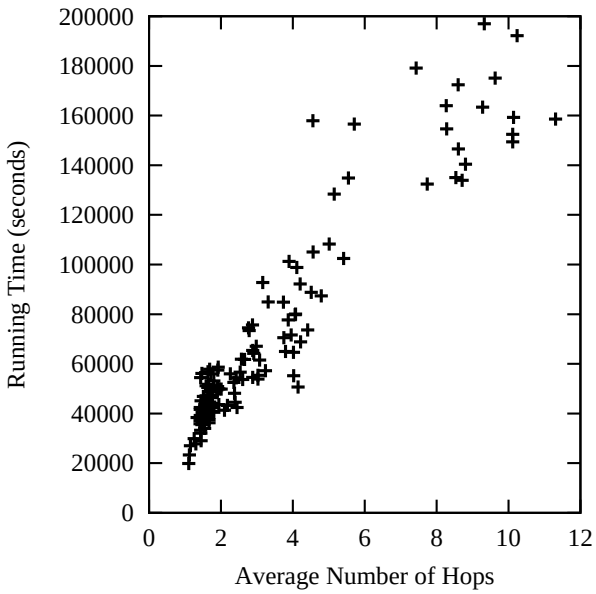


Figure 10. Relationship between average message distance and running time for n-body communication.

packing heuristics and why the S-curve does relatively well on the 16×22 mesh. On the other hand, Hilbert with free list performs very well, despite having the largest discontinuity.

Beyond these interactions, our experiments support a number of other observations. As observed by Leung et al. [17], the choice of curve seems have the dominant effect on performance for Paging algorithms. Generally, using sorted free list for a curve gives the worst performance and using Best Fit gives the best. In addition, MC consistently outperforms MC1x1. This is not surprising since users request an allocation with dimensions that can fit the job, biasing MC toward rectangular allocations, lessening interjob contention. We expect the superiority of MC to MC1x1 to be even greater for real programs since users are likely to request shapes particularly good for their communication pattern, as in the case of a simulation where the user knows the shape of the object being simulated. Although this observation is not applicable to Cplant since its software does not get a user-supplied job shape, it is an argument for designing future systems to gather shape information from the user.

More experiments of this type need to be performed. Since allocation algorithms are an important factor in determining machine throughput, they need to be evaluated with a variety of communication patterns and using different traces. Obviously, the ideal it to find a general purpose allocation algorithm that works reasonably well for all types of problems, but a strategy to harness the strengths of different algorithms would also be useful.

References

[1] J. Alber and R. Niedermeier. On multidimensional Hilbert indexings. *Theory of Computing Systems*, 33:295–312, 2000.

[2] S. Baylor, C. Benveniste, and Y. Hsu. Performance evaluation of a massively paralel I/O subsystem. In R. Jain, J. Werth, and J. Browne, editors, *Input/Output in parallel and distributed computer systems*, volume 362 of *The Kluwer International Series in Engineering and Computer Science*, chapter 13, pages 293–311. Kluwer Academic Publishers, 1996.

[3] S. Bhattacharya and W.-T. Tsai. Lookahead processor allocation in mesh-connected massively parallel computers. In *Proc. 8th International Parallel Processing Symposium*, pages 868–875, 1994.

[4] R. Brightwell, L. A. Fisk, D. S. Greenberg, T. Hudson, M. Levenhagen, A. B. Maccabe, and R. Riesen. Massively parallel computing using commodity components. *Parallel Computing*, 26(2-3):243–266, 2000.

[5] C. Chang and P. Mohapatra. Improving performance of mesh connected multicomputers by reducing fragmentation. *Journal of Parallel and Distributed Computing*, 52(1):40–68, 1998.

[6] P.-J. Chuang and N.-F. Tzeng. An efficient submesh allocation strategy for mesh computer systems. In *Proc. International Conf. on Distributed Computer Systems*, pages 256–263, 1991.

Algorithm	% contiguous	Ave. components
S-curve w/BF	81.5%	1.34
Hilbert w/BF	81.3%	1.33
Hilbert w/FF	80.7%	1.36
H-index w/BF	79.8%	1.35
S-curve w/FF	79.3%	1.37
H-index w/FF	79.1%	1.38
MC	68.5%	1.91
MC1x1	67.5%	1.99
S-curve	67.5%	1.59
H-index	65.8%	1.64
Gen-Alg	65.6%	2.27
Hilbert	64.6%	1.66

Figure 11. Percent of jobs allocated contiguously and the average number of components each job is allocated into for all-to-all communication on a 16×16 mesh with load 1.0.

[7] Cray Inc. Network queuing environment. `http://www.cray.com/products/software/nqe.html`.

[8] J. Csirik, D. Johnson, C. Kenyon, J. Orlin, P. Shor, and R. Weber. On the sum-of-squares algorithm for bin packing. In *Proc. 32nd Annual ACM Symposium on Theory of Computation (STOC)*, pages 208–217, 2000.

[9] J. Csirik, D. Johnson, C. Kenyon, P. Shor, and R. Weber. A self-organizing bin packing heuristic. In *Proc. Algorithm Engineering and Experimentation: International Workshop (ALENEX)*, volume 1619 of *Springer Lecture Notes in Computer Science*, pages 246–265, 1999.

[10] C. Gotsman and M. Lindenbaum. On the metric properties of discrete space-filling curves. *IEEE Trans. on Image Processing*, 5(5):794–797, 1996.

[11] D. Hilbert. Über die stetige abbildung einer linie auf ein flachenstück. *Math. Ann.*, 38:459–460, 1891.

[12] D. S. Johnson. *Near-optimal bin packing algorithms*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1973.

[13] J. R. Johnston. private communication, October 2003.

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

[15] S. Krumke, M. Marathe, H. Noltemeier, V. Radhakrishnan, S. Ravi, and D. Rosenkrantz. Compact location problems. *Theoretical Computer Science*, 181(2):379–404, 1997.

[16] Lawrence Livermore National Laboratory. Advanced Simulation and Computing (ASCI). `http://www.llnl.gov/asci/`.

[17] V. J. Leung, E. M. Arkin, M. A. Bender, D. P. Bunde, J. R. Johnston, A. Lal, J. S. B. Mitchell, C. A. Phillips, and S. S. Seiden. Processor allocation on Cplant: achieving general processor locality using one-dimensional allocation strategies. In *Proc. 4th IEEE International Conference on Cluster Computing*, pages 296–304, 2002.

[18] V. J. Leung, C. A. Phillips, M. A. Bender, and D. P. Bunde. Algorithmic support for commodity-based parallel computing systems. Technical Report SAND2003-3702, Sandia National Laboratories, 2003.

[19] 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.

[20] V. Lo, K. Windisch, W. Liu, and B. Nitzberg. Noncontiguous processor allocation algorithms for mesh-connected multicomputers. *IEEE Transactions on Parallel and Distributed Computing*, 8(7), 1997.

[21] J. Mache and V. Lo. Dispersal metrics for non-contiguous processor allocation. Technical Report CIS-TR-96-13, University of Oregon, 1996.

[22] J. Mache and V. Lo. The effects of dispersal on message-passing contention in processor allocation strategies. In *Proc. Third Joint Conference on Information Sciences, Sessions on Parallel and Distributed Processing*, volume 3, pages 223–226, 1997.

[23] J. Mache, V. Lo, and K. Windisch. Minimizing message-passing contention in fragmentation-free processor allocation. In *Proc. 10th International Conf. Parallel and Distributed Computing Systems*, pages 120–124, 1997.

[24] B. Moon, H. V. Jagadish, C. Faloutsos, and J. Saltz. Analysis of the clustering properties of Hilbert space-filling curve. *IEEE Trans. on Knowledge and Data Engineering*, 13(1):124–141, 2001.

[25] S. Moore and L. Ni. The effects of network contention on processor allocation strategies. In *Proc. 10th International Parallel Processing Symposium*, pages 268–274, 1996.

[26] NASA. The portable batch system. `http://www.nas.nasa.gov/Software/PBS/`.

[27] R. Niedermeier, K. Reinhardt, and P. Sanders. Towards optimal locality in mesh-indexings. In *Proc. 11th Intl Symp on Fund. Computation Theory*, volume 1279 of *LNCS*, pages 364–375, 1997.

[28] Sandia National Laboratories. The Computational Plant Project. `http://www.cs.sandia.gov/cplant`.

[29] V. Subramani, R. Kettimuthu, S. Srinivasan, J. Johnson, and P. Sadayappan. Selective buddy allocation for scheduling parallel jobs on clusters. In *Proc. 4th IEEE International Conference on Cluster Computing*, 2002.

[30] University of Oregon Resource Allocation Group. Procsimity. `http://www.cs.uoregon.edu/research/DistributedComputing/ProcSimity.html%`.

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

[32] K. Windisch, J. Miller, and V. Lo. Procsimity: An experimental tool for processor allocation and scheduling in highly parallel systems. In *Proc. Fifth Symp. on the Frontiers of Massively Parallel Computation*, pages 414–421, 1995. `ftp://ftp.cs.uoregon.edu/pub/lo/procsimity.ps.gz`.

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