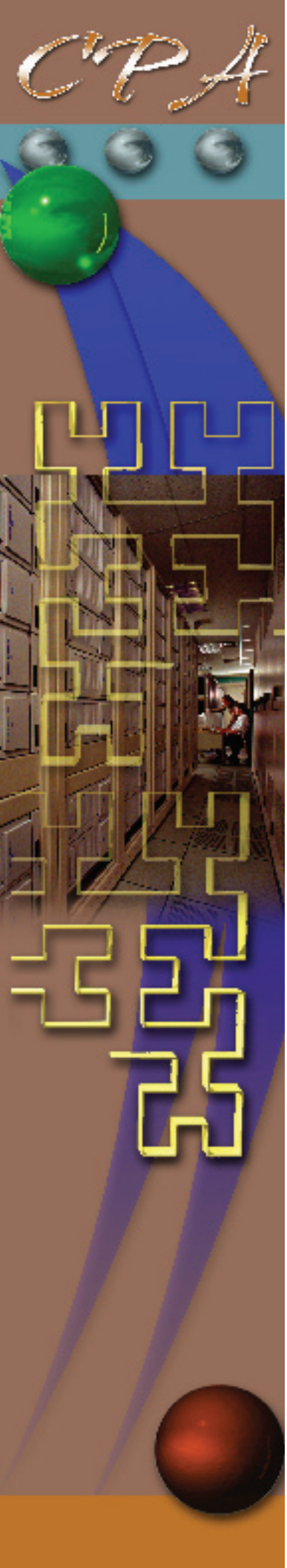
R&D 100 Nomination 2006 SAND2006-1142P

CPA

Compute Process Allocator

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under contract DE-AC04-94AL85000.



"Compute Process Allocator (CPA)"

1. **Submitting Organization**

Sandia National Laboratories
P.O. Box 5800, MS 1110
Albuquerque, NM 87185-1110

Submitter's Name
Vitus J. Leung, Senior Member of the Technical Staff
(505) 844-1896, (505) 845-7442 (fax), vjleung@sandia.gov

Affirmation

I affirm that all information submitted as a part of, or supplemental to, this entry is a fair and accurate representation of this product.

Submitter's Signature

2. **Joint entry with**

State University of New York
Department of Computer Science
Stony Brook, NY 11794-4400
Michael A. Bender, Associate Professor
(631) 632-7835, (631) 632-8334 (fax), bender@cs.sunysb.edu

University of Illinois
Department of Computer Science
Urbana, IL 61801
David P. Bunde, Ph.D. Candidate
(217) 244-6433, (217) 265-6591 (fax), bunde@uiuc.edu

Sandia National Laboratories
P.O. Box 5800, MS 1110
Albuquerque, NM 87185-1110
Kevin T. Pedretti, Senior Member of the Technical Staff
(805) 685-0888, (505) 845-7442, ktpedre@sandia.gov

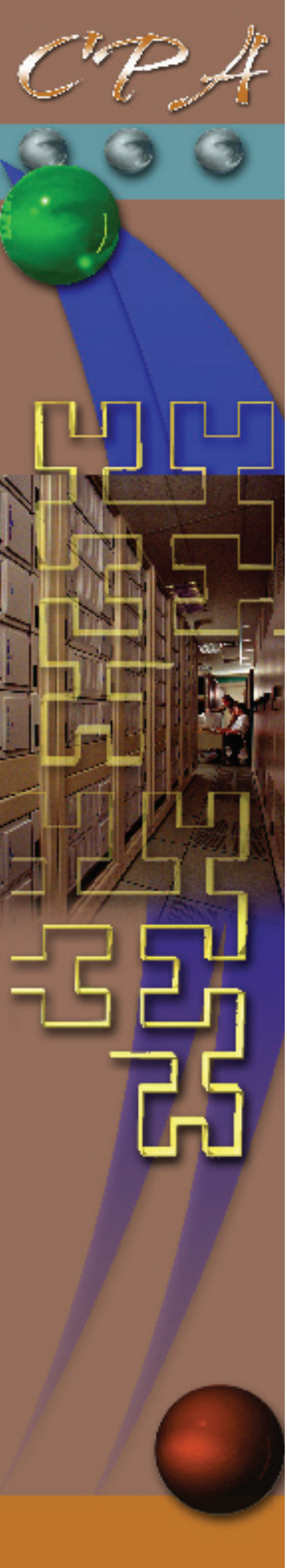
Sandia National Laboratories
P.O. Box 5800, MS 1110
Albuquerque, NM 87185-1110
Cynthia A. Phillips, Distinguished Member of the Technical Staff
(505) 845-7296, (505) 845-7442 (fax), caphill@sandia.gov

3. **Product Name**

Compute Process Allocator (CPA)

4. **Description of Entry**

CPA is the first allocator to balance individual job allocation with future allocations over 10,000 processors, allowing jobs to be processed faster and more efficiently.



5. When Product Was First Marketed/Available for Order

After extensive prototype development, the CPA was licensed to Cray Inc. in June 2005.

6. Inventor or Principal Developer

Vitus J. Leung, SMTS
Discrete Algorithms and Math Department
Sandia National Laboratories
P.O. Box 5800, MS 1110
Albuquerque, NM 87185-1110
(505) 844-1896, (505) 845-7442 (fax), vjleung@sandia.gov

Additional Developers:

Michael A. Bender, Associate Professor
Dept. of Computer Science
State University of New York
Stony Brook, NY 11794-4400
(631) 632-7835, (631) 632-8334 (fax), bender@cs.sunysb.edu

David P. Bunde, Ph.D. Candidate
Dept. of Computer Science
University of Illinois
Urbana, IL 61801
(217) 244-6433, (217) 265-6591 (fax), bunde@uiuc.edu

Kevin T. Pedretti, SMTS
Scalable Computing Systems Department
Sandia National Laboratories
P.O. Box 5800, MS 1110
Albuquerque, NM 87185-1110
(805) 685-0888, (505) 845-7442 (fax), ktpedre@sandia.gov

Cynthia A. Phillips DMTS
Discrete Algorithms and Math Department
Sandia National Laboratories
P.O. Box 5800, MS 1110
Albuquerque, NM 87185-1110
(505) 845-7296, (505) 845-7442 (fax), caphill@sandia.gov

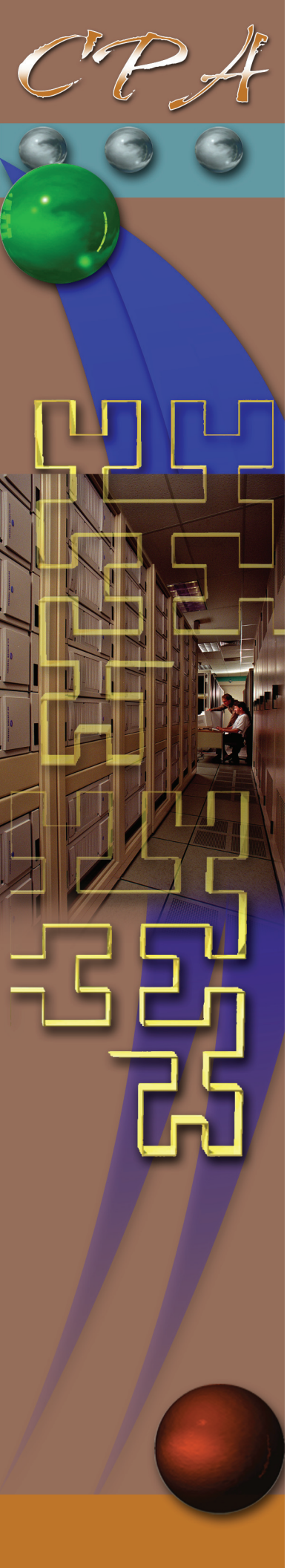
7. Product Price

CPA is licensed to Cray at a price that we cannot disclose.

8. Patents Held/Pending

Applied for patents April 2005

U.S. Patent Application Serial No. 11/110,206
U.S. Patent Application Serial No. 11/110,466



9. Product's Primary Function

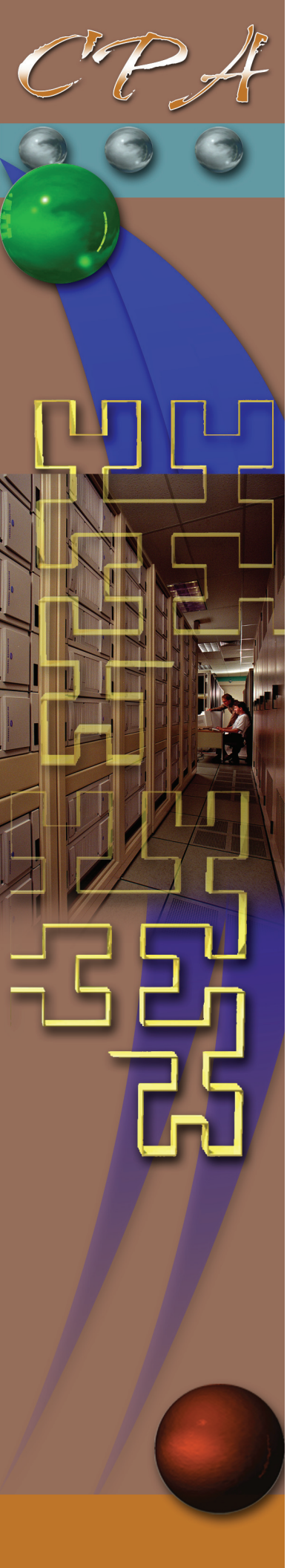
CPA uses resource-allocation strategies to achieve processor locality for parallel jobs in supercomputers. Specifically, given a stream of jobs from a job scheduler, the CPA allocates processors to optimize processor locality for each job while preserving locality options for future jobs and therefore increasing the number of jobs processed. Furthermore, the CPA does not need to alter the behavior of the scheduler (e.g., block jobs). In experiments at Sandia National Laboratories, the CPA increased throughput by 23%; in effect, with CPA, five jobs were processed in the time it normally took to process four.

The CPA operates in the following computing environment. Supercomputer users submit parallel jobs to a job queue. Each parallel job *j* is dispatched for processor allocation with a requested number of processors p_j. The jobs arrive online, that is job *j* is only known to the CPA after it arrives. When a job is scheduled to run, the CPA assigns it to a set of processors, which are exclusively dedicated to the job until it terminates.

CPA increased throughput by 23%, processing five jobs in the time it normally took to process four.

Preemption and migration are not allowed, meaning that once a job begins, it must be executed to completion on the same set of processors. To obtain maximum throughput, the CPA chooses processors for a job that are physically near each other, thereby minimizing communication costs and avoiding bandwidth contention caused by jobs sharing communication paths. In experiments, the average number of communication hops between processors allocated to a job gave the strongest correlation to job completion time. The CPA optimizes this natural fragmentation metric.

Remarkably, the CPA can achieve processor locality for a stream of jobs in massively parallel supercomputers using simple, one-dimensional allocation strategies. The CPA accomplishes this reduction using a space-filling curve, which imposes an ordering on the network of processors such that locations near each other on the curve are also near each other in the network of processors. The CPA's approach is applicable even when the processors are connected by irregular, higher dimensional networks.



The one-dimensional strategies for the CPA work as follows: When job *j* is dispatched, the CPA determines if the job can be allocated contiguously along the space-filling curve. When a job can be allocated contiguously, the CPA chooses the interval using an adaptation of the Best Fit, one-dimensional, bin-packing algorithm. When a job cannot be allocated contiguously, the CPA chooses the allocation that minimizes the span of the job. (The span of a job is the distance on the curve between the first and last processors of the job.) Minimizing the span empirically minimizes the fragmentation metric above.

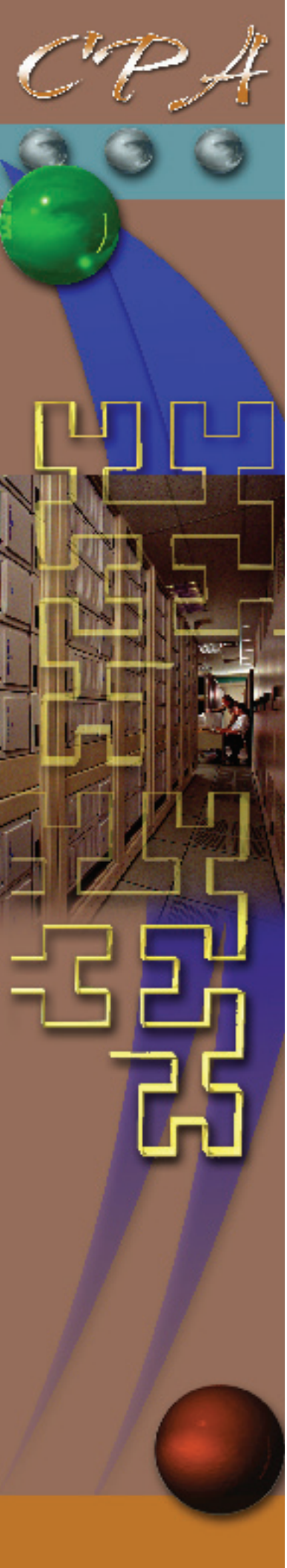
The CPA can be ported to any system that uses a variant of NASA's Portable Batch System (PBS), which maintains code to interface with the CPA. PBS is the defacto standard in cluster and parallel system resource management. In addition to Cray, PBS is used to manage systems built by Dell, Hewlett Packard, IBM, and Silicon Graphics. PBS is supported on most Linux, UNIX, Windows, and Mac OS X based systems.

10A. Product's Competitors

While there are no direct competitors, there are many overlapping products. Some of them are:

- Altair's PBS Pro 7.0
- Cluster Resources' Maui Cluster Scheduler 3.2.6, Moab Workload Manager 4.2.2, and Torque 2.0
- IBM's LoadLeveler 3.2
- LLNL's SLURM 0.6
- NASA's OpenPBS 2.3.16
- Platform's LSF 6.x, and
- Quadrics' RMS Hawk

All of these products provide allocation in addition to other functions. The closest match to CPA is the Maui Cluster Scheduler and the Moab Workload Manager, which are similar products. The Maui Cluster Scheduler is open source, while the Moab Workload Manager is commercial. Both can provide allocation for all of the other products.



10B. Product's and Competitors' Key Features Table or Matrix

Allocator Technology	Processor Reordering	Contiguous Allocation	Noncontiguous Allocation	Scalability
Compute Process Allocator (CPA)	Based on machine topology & routing	Best Fit	Span minimizing	Over 10,000 nodes
Maui Cluster Scheduler 3.2.6	None	Yes	Arbitrarily bad	4,096 nodes
Moab Workload Manager 4.2.2	None	Yes	Arbitrarily bad	4,096 nodes

10C. How Product Improves on Competitive Products/Technologies

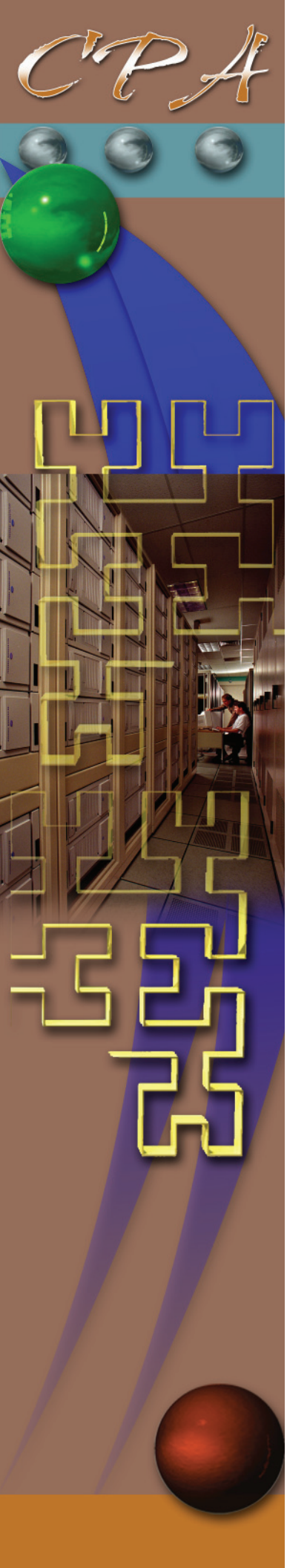
CPA is superior to other allocators for the following reasons:

1. The CPA is the only allocator that uses a space-filling curve to reorder a network of processors so that locations near each other in the reordering are also near each other in the network of processors. Simulations and experiments show that such a reordering alone improved locality and system throughput by 14-19% over allocators that do not use a space-filling curve. (See Appendix 1.)

2. The CPA is the only allocator that uses Best Fit packing for contiguous allocation and span minimization for non-contiguous allocation. Simulations and experiments show that Best Fit packing and span minimization improved locality and system throughput by an additional 5-11% over allocators that do not use Best Fit packing and span minimization. (See Appendix 1.)

3. The CPA is the only allocator that uses the above two improvements together. Simulations and experiments show that reordering a network of processors, using Best Fit packing for contiguous allocation, and minimizing span for non-contiguous allocation improved locality and system throughput by 23% over allocators that do not use these techniques. (See Appendix 1.) Simulations show that combining these simple, one-dimensional allocation strategies improved locality over higher-dimensional allocation strategies by an average of one percent over a stream of jobs. The locally better decisions of these higher-dimensional allocation strategies seemed to paint the algorithm into a corner over time. (See Appendix 2.)

4. The CPA is the only allocator that uses a distributed software architecture. The CPA is scalable to over 10,000 nodes. Non-distributed allocators are scalable to only 4,096 nodes.



11A. Product's Principal Applications

CPA's principal application is to maximize throughput on massively parallel supercomputers by balancing locality of processors assigned to a parallel job over a stream of parallel jobs. The CPA is distributed and scalable to tens of thousands of processors and is currently being used on these mesh supercomputers:

- 10,880-node Red Storm machine at Sandia National Laboratories
- 5,200-node Jaguar machine at Oak Ridge National Laboratory
- 4,096-node machine at the Engineer Research and Development Center Major Shared Resource Center (ERDC MSRC)
- 2,060-node machine at the Pittsburgh Supercomputing Center
- 1,100-node machine at the Swiss Scientific Computing Center, and
- four other machines at three sites that cannot be specified

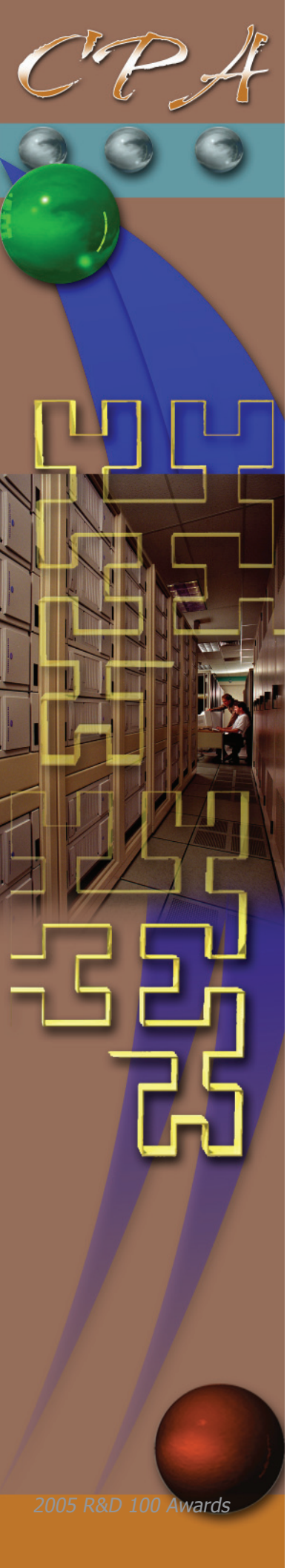
11B. Other Applications for Which Product Can Now Be Used

The versatility of CPA allows it to work on all other architectures, including fat trees.

CPA is currently licensed to nine sites, but the sites are unable to specify exactly what applications CPA impacts. Work applications at these sites vary: nanoscience, astrophysics, global climate changes, and military and civil work missions. In each case, CPA improves the speed of the applications run on the machines using CPA and improves the performance of the hardware by 23%. Moreover, the versatility of CPA allows it to work on any system that runs PBS, not just the Cray XT3.

12. Summary

CPA is an example of how a small investment in computer algorithms can dramatically leverage the return on a large investment in computer hardware. CPA is less than 1% of the cost of a parallel computer. In simulations and experiments, CPA increased the locality and throughput on a parallel computer by 23% over simpler one-dimensional allocators. In simulations, CPA increased the locality on a parallel computer by 1% over more time-consuming, higher-dimensional allocators. The CPA is distributed and scales to over 10,000 nodes, while non-distributed allocators have been scaled to only 4,096 nodes.



ORGANIZATION DATA

13. Contact Person

Vitus J. Leung, SMTS
Discrete Algorithms and Math Department
Sandia National Laboratories
P.O. Box 5800, MS 1110
Albuquerque, NM 87185-1110
(505) 844-1896, (505) 845-7442 (fax), vjleung@sandia.gov

APPENDIX

Appendix 1: From "Proc. IEEE International Conference on Cluster Computing" 2002 "Processor Allocation on Cplant: Achieving General Processor Locality Using One-Dimensional Allocation Strategies" Leung, V. J.; Arkin, E. M.; Bender, M. A.; Bunde, D. P.; Johnston, J. R.; Lal, A.; Mitchell, J. S. B.; Phillips, C. A.; Seiden, S. S.

Appendix 2: From "Proc. 9th Workshop on Algorithms and Data Structures" 2005 "Communication-Aware Processor Allocation for Supercomputers" Bender, M. A.; Bunde, D. P.; Demaine, E. D.; Fekete, S. P.; Leung, V. J.; Meijer, H.; Phillips, C. A.

Processor Allocation on Cplant: Achieving General Processor Locality Using One-Dimensional Allocation Strategies

Vitus Leung* Esther M. Arkin† Michael A. Bender ‡ David Bunde§ Jeanette Johnston*
Alok Lal¶ Joseph S. B. Mitchell† Cynthia Phillips* Steven S. Seiden‖

We dedicate this article to the memory of Steve Seiden, who was killed in a tragic cycling accident on June 11, 2002.

Abstract

The Computational Plant or Cplant is a commodity-based supercomputer under development at Sandia National Laboratories. This paper describes resource-allocation strategies to achieve processor locality for parallel jobs in Cplant and other supercomputers. Users of Cplant and other Sandia supercomputers submit parallel jobs to a job queue. When a job is scheduled to run, it is assigned to a set of processors. To obtain maximum throughput, jobs should be allocated to localized clusters of processors to minimize communication costs and to avoid bandwidth contention caused by overlapping jobs.

This paper introduces new allocation strategies and performance metrics based on space-filling curves and one dimensional allocation strategies. These algorithms are general and simple. Preliminary simulations and Cplant experiments indicate that both space-filling curves and one-dimensional packing improve processor locality compared to the sorted freelist strategy previously used on Cplant. These new allocation strategies are implemented in the new release of the Cplant System Software, Version 2.0. This version was phased into the Cplant systems at Sandia by May 2002.

*MS 1110, Sandia National Laboratories, Albuquerque, NM 87185-1110. {vjleung , jjohnst , caphill@mp.sandia.gov .

†Dept. of Appl. Math. and Statistics, SUNY Stony Brook, NY 11794-3600. {estie , jsbm }@ams.sunysb.edu .

‡Dept. of Computer Science, SUNY Stony Brook, NY 11794-4400. bender@cs.sunysb.edu .

§Dept. of Computer Science, University of Illinois, Urbana, IL 61801. bunde@uiuc.edu .

¶Dept. of Electrical Engineering and Computer Science, Tufts University, Medford, MA 02155. alal@eecs.tufts.edu.

‖Dept. of Computer Science, Louisiana State University, Baton Rouge, LA 70803. sseiden@acm.org .

1. Introduction

As part of the Accelerated Strategic Computing Initiative [31], the Department of Energy Laboratories are purchasing a sequence of increasingly powerful custom supercomputers. In a parallel effort to increase the scalability of commodity-based supercomputers, Sandia National Laboratories is developing the Computational Plant or Cplant [44, 23, 9, 40, 7, 8]. This paper describes resource-allocation algorithms to optimize processor locality in Cplant and other related supercomputers.

Although Sandia maintains a diverse set of computing resources, the tools for managing these resources commonly rely on scheduling/queueing software such as NQS or PBS to determine which of the available jobs should be run next. These jobs are prioritized based on a variety of factors including computing resources already allocated to the owners of the jobs, number of processors requested, running-time estimates, waiting time so far, and even day of week and time of day. But this decision is *not* based on the locations of the free processors. The scheduler simply verifies a sufficient number of processors are free before dispatching a job.

When a job is selected to run, the processor allocator assigns it to a set of processors, which are exclusively dedicated to this job until it terminates. To obtain maximum throughput, the processors allocated to a single job should be physically near each other to minimize communication costs and to avoid bandwidth contention caused by overlapping jobs. Processor locality is particularly important in commodity-based supercomputers, which typically have higher communication latencies and lower bandwidth than supercomputers with custom networks.

Processor locality is an issue for Cplant. We have shown that if two high-communication jobs are hand-placed on the machine so that their communication paths overlap signif-

icantly, both jobs' running times are approximately doubled. Subramani et al. [48] reach similar conclusions. The Cplant switches are usually connected in a two or three-dimensional mesh topology. Most switches contain four processors. Thus, a good processor allocation includes all processors in a rough subcube of switches.

For the problem addressed in this paper, we have no control over the scheduler. Given a stream of jobs from the scheduler, we wish to allocate processors to maximize processor locality. More precisely, we address the following problem. Each parallel job j has an *arrival time* a_j (the time when it is dispatched from the scheduler for processor allocation), a requested *number of processors* p_j, and a *processing time* (the user submits an estimated processing time; the true processing time is known when the job completes; the user's job gets truncated if it does not complete within some specified factor of the estimated processing time). The jobs arrive *online*, that is, job j is only known to the allocator after the time a_j when it is dispatched from the scheduler. *Preemption* and *migration* are not allowed, that is, once a job is begun it must be executed to completion on the same set of processors. The objective is to assign a set of processors to each job to optimize some global measure of locality. For example, if the machine is a mesh, we may choose to optimize the average expansion of the bounding box, i.e. the ratio of the bounding box for the allocated processor set and the minimum possible bounding box. Section 2.3 defines a new locality measure motivated by this work.

The thesis of this paper is that processor locality can be achieved in massively parallel supercomputers using simple, one-dimensional allocation strategies. This approach is applicable even when the processors are connected by irregular, higher dimensional networks. We accomplish this reduction using an space-filling curve which imposes an ordering on the network of processors such that locations near each other on the curve are also near each other in the network of processors.

In this paper we describe our experience applying this strategy to Cplant. In Cplant supercomputers, the nodes are usually connected by two or three-dimensional meshs with toroidal wraps in one or more dimensions, but some of the oldest systems have more highly-connected irregular topologies. We use *Hilbert curves* in two dimensions and have an integer program for general networks. We present preliminary experimental results and motivating simulations for 2D and 3D meshes.

The remainder of the paper is organized as follows. The next subsection describes related theoretical and simulation work. Section 2 describes our allocation strategies given a processor ordering. Section 3 summarizes our simulations. Section 4 describes our experimental results. Section 5 offers some concluding remarks.

1.1 Related Work

The simulation-based investigations of Subramani et al. [48] show that fragmentation is necessary for high performance. Their work is directly motivated by the Cplant system, though some of it can be applied to more general systems. They investigated the effect on system throughput of a policy forbidding fragmentation. They used trace files from the Cornell Supercomputing Center. In their simulation, they queued jobs until a set of contiguous processors were available, scaling the running times down to indicate the benefit of contiguous allocation. They determined that fragmentation must cause at least a factor-of-two slowdown in order for the benefit of completely contiguous allocation to compensate for the loss of processor utilization. Thus any real system must allow for fragmentation.

Subramani et al. [48] also investigated a strategy that allows fragmentation, motivated by the buddy strategy for memory allocation. They considered 2D and 3D meshes. The machine is subdivided geometrically. For example two halves of the machine are a buddy pair, two quarters within the half, etc. Jobs are allocated to these predefined subblocks. Their system holds some jobs back rather than fragmenting them. This buddy approach does not directly apply to Cplant because the allocator cannot ever delay jobs.

A problem closely related to Cplant processor allocation is *memory allocation*. In this problem there is an array of memory, and contiguous sub-arrays are allocated and deallocated online [41, 42, 32]. One objective function is to minimize the highest memory address used and consequently the required memory size. Memory allocation differs from processor allocation because memory allocators leave empty space to guarantee contiguity and are allowed to refuse requests that do not fit contiguously.

Another related problem is online *bin packing.* In bin packing, the objective is to pack a set of items with given sizes into bins. Each bin has a fixed capacity and cannot be assigned items whose total size exceeds this capacity. The goal is to minimize the number of bins used. The offline version is NP-hard [21] and bin packing was one of the first problems to be studied in terms of both online and offline approximability [26, 27, 28]. Multi-dimensional bin packing, where the items and bins are hyperrectangles, has also been studied. The seminal offline and online results appear in [12, 14], while the latest results are in [45]. For a more detailed review of bin packing, see the surveys [13, 17]. Bin packing results cannot be directly applied to our problem since we have only a single "bin". Also objects can leave the system, creating multiple holes within this bin because jobs cannot migrate.

Our work adapts several of the algorithms for one-dimensional online bin packing. A common feature of these algorithms is they keep a list of partially-filled bins. As ob-

jects arrive, they may be placed in one of these bins (assuming they fit) or they may be placed in a new bin, which is then added to the list. The First Fit algorithm [26] places a new object in the first bin in which it fits. Best Fit [26] places a new object in the bin whose remaining space will be smallest. When the bins and objects have integral sizes, the more complicated Sum of Squares algorithm [16] is also available. This algorithm bases its decisions on a vector N, where $N(i)$ is the number of bins whose contents have remaining size i. It places a new item in the bin which minimizes the resulting value of $\sum N(i)^2$. This allocation policy encourages a variety of sizes of unallocated regions. When the input comes from a discrete distribution, this algorithm has near-optimal behavior [15].

Other researchers have used space-filling curves for a variety of problems. Originally, space-filling curves were introduced by Hilbert [25] and Peano [36]. Recent presentations appear in [18] and [43]. Hilbert curves have been shown to preserve several measures of "locality" [34, 22]. An alternative with better performance in two dimensions is given in [35]. Generalizations of Hilbert curves to higher dimensions are given in [1]. Specific applications include matrix multiplication [11, 19], domain decomposition [3, 24, 37], and image processing [2, 33, 4, 49, 30, 29]. They are also a standard tool in the creation of *cache-oblivious algorithms* [20, 38, 5, 39, 6, 10], which have asymptotically optimal memory performance on multilevel memory hierarchies while avoiding memory-specific parameterization.

There is a large body of work on scheduling and online scheduling, in particular. We do not attempt to review all this work here, but refer the reader to the survey of Sgall [46].

2 Allocation Strategies

2.1 Baseline Cplant Allocation

Our test Cplant system is a 2D (toroidally-wrapped) mesh. The Cplant version 1.9 default allocator uses a sorted free list based on a left-to-right, top-to-bottom linear processor order. Even for the Cplant machines with non-mesh interconnection topologies, the processors are physically placed on planes so that such an ordering is still possible. When a job j requiring p_j processors is dispatched to the allocator, the allocator queries the system to determine which processors are free and gathers these processors into a sorted list. The job is allocated to the first p_j processors in the list. These processors may be far apart with respect to the linear order (and the real machine), even if there is a contiguous piece of sufficient size available later in the list.

We use the latest version 1.9 default Cplant system as our baseline against which to measure improvement.

2.2 Transforming to One-Dimensional Allocation

As with the current Cplant node-allocation algorithms, we impose a linear ordering on the processors. We use a *Hilbert curve* (also called a *fractal curve*), rather than an arbitrary order or sorting by row and column. We then allocate to obtain locality within this linear ordering.

The Hilbert curve only applies to grid topologies. We consider the problem of finding good one-dimensional orderings for general parallel interconnection topologies and formulate this problem as an integer program. If two processors' ranks in the one-dimensional ordering differ by k, then their contribution to the objective function (which we minimize) is a parameter $w(k)$ times their distance in the graph. The parameter $w(k)$ decreases rapidly (e.g., inverse exponentially) with k, so that close pairs in the linear order are coerced to be close physically. We can also use this objective function to compare different curves for a given topology.

The above integer-programming problem for computing a good one-dimensional ordering is NP-complete since it is a generalization of the Hamiltonian path (HP) problem. This problem is HP if we set $w(k) = 0$ for all $k > 1$, and $w(1) = 1$. The graph has a Hamiltonian path if and only if the integer program has a solution with an objective function value of $n - 1$ where n is the number of nodes in the graph. Though the problem is NP-complete we may be able to solve particular instances optimally or to within a provable instance-specific error tolerance using PICO (Parallel Integer and Combinatorial Optimizer), a massively-parallel branch-and-bound code developed at Sandia National Laboratories and Rutgers University. PICO includes a (branch-and-cut) mixed-integer program solver. Though this computation may be time-consuming, it is performed only once for any given physical machine and choice of $w(k)$.

2.3 One-Dimensional Allocation Strategies

We modify existing memory-allocation and bin-packing algorithms for the Cplant processor-allocation problem. This is not a straightforward generalization because it is not required (although desirable) that processors be allocated contiguously. We use analogs to bin-packing algorithms when processors can be allocated contiguously. The intervals of contiguous free processors are analogous to free space in unfilled bins. However, we must determine a different allocation strategy when there is no contiguous interval of sufficient size.

Span Metrics Our one-dimensional processor-locality metric is motivated by a true line or ring topology. Let r_p be the *rank* of processor p in the linear ordering. This will be an integer in the range $1, \ldots, |P|$, where P is the set

of processors. Let M_j be the set of processors assigned to job j. The *linear span*, s_j^ℓ is the number of processors potentially involved in message propagation/delivery for job j if the processors are connected only in a line. That is, s_j^ℓ is the maximum difference in rank between any pair of processors assigned to job j (plus one): $s_j^\ell = \max_{p \in M_j} r_p - \min_{p \in M_j} r_p + 1$. All processors with ranks between this minimum and maximum rank (plus the endpoints) are involved in routing a message between these two processors. These are the processors "owned" by job j and those "trapped" between pieces of job j. The *ring span* s_j^w is a measure of locality if the processors are connected in a ring, again corresponding to the processors "owned" by job j and those "trapped" by these segements. Computationally, it's easier to determine the largest "free" set of processors, accounting for the ring wraparound. Let r_{ji} be the ith-smallest rank of a processor in M_j for $i = 0 \ldots p_j - 1$. Then we define $s_j^w = |P| - \max(\max_{i=0}^{p_j-2} r_{j,i+1} - r_{ji} - 1, P - rj, k - 1 + r_{j,0} + 1)$. In this paper, we use ring span which we call *span* and denote s_j for brevity. Span s_j is a measure of the processor locality of job j for more general topologies provided the space-filling curve closely reflects processor locality. The integer program described in Section 2.2 computes a processor ranking for ring span provided difference in rank is computed as the minimum distance around the ring.

In this paper we test heuristic methods for span minimization. (Minimizing metrics based on span is computationally difficult.) Examples of such metrics include the sum of the spans of jobs ($\sum_{i=1}^n s_i$), the max of the spans of jobs ($\max_{i=1}^n s_i$), the sum (resp. max) of the spans divided by the requested number of processors ($\sum_{i=1}^n s_i/p_i$), the sum (resp. max) of the spans weighted by the processing times ($\sum_{i=1}^n s_i t_i$), etc.

Strategies When job j is dispatched, we determine if there is a contiguous interval of free processors large enough to run job j. When a job *cannot* be allocated contiguously, then it is allocated across multiple intervals. We choose the allocation that *minimizes the span* of the job. In a tie we push the job to the smallest rank possible. When a job *can* be allocated contiguously, we choose which interval to use based on adaptations of one-dimensional bin-packing algorithms. We consider three strategies:

- *First-Fit Allocation* – Allocate j to the first interval that is large enough.
- *Best-Fit Allocation* – Allocate j to the interval that minimizes the number of unallocated processors remaining in the interval.
- *Sum-of-Squares Allocation* – For each interval to which j could be allocated, determine the number of intervals of each size that would remain. Allocate j to the interval that minimizes the sum of squares of these numbers of intervals.

All of these strategies are easy to implement and run quickly. The gains in system throughput (described in Section 5) far outweigh the additional computation time of the allocator.

3 Simulations

We built an event-driven Cplant simulator, which tests the allocation strategies from Section 2.3 on space-filling curves. The objective of the simulator is to exhibit tendencies rather than to predict running times precisely. Our simulations suggest that one-dimensional allocation strategies coupled with space-filling curves yield processor locality in higher dimensions. A variety of performance metrics gauge the processor locality.

Trace Files The Cplant simulator was run on traces from October, November, and December 2000. These trace files contain data about all jobs submitted to a Cplant machine configured as a heavily augmented 2D mesh with 592 compute processors. The trace file includes the times that the jobs were dispatched from the scheduler, the number of processors requested, and the actual running times. These traces did not contain the processors on which the job was actually run so we cannot compute the fragmentation/runtime environment of these jobs.

From a trace it is hard to predict how the running time of the jobs would change if the allocation were different. The difficulty is because the running times depend on factors that are hard or impossible to model. These factors include the processor allocation, the communication patterns of the jobs, the overlaps of the jobs, and the properties of the communication network.

Rather than make potentially spurious estimates about the change in the running time of the job with different allocations, our simulations hold the running times constant and use metrics based on processor locality. The assumption is that increased locality improves performance, but that the actual speed-ups should be determined through experimentation.

We transformed the traces into many inputs that model different workloads. We developed one parameterized set of inputs by increasing or decreasing the running times of the jobs by a factor that we call the *work multiple*. Increasing running times makes processors busier since jobs are in the system for a longer amount of time. We developed a second set of parameterized inputs by duplicating jobs and perturbing the arrival times; the number of times that a job is duplicated is called the *replication factor*. Since

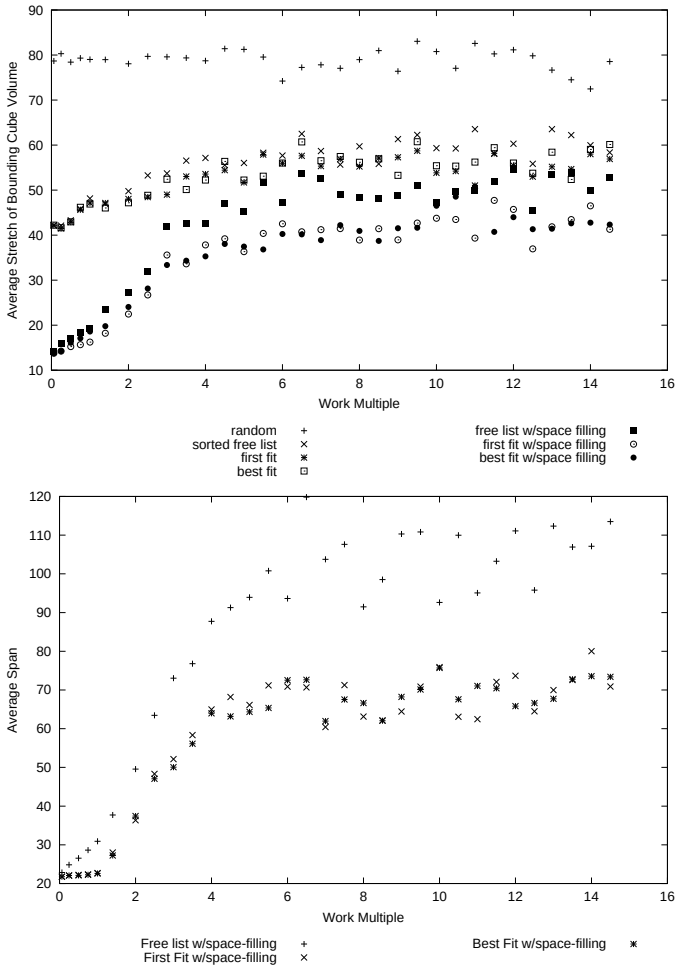


Figure 1. Top: The average bounding cube divided by the smallest bounding cube. Bottom: The average span of jobs. The x-axis plots the work multiple, where we simulate the strategies on a range of workloads.

both types of inputs yield similar results, we report only the work-multiple results.

Metrics One-dimensional metrics include the *average span* and the *average span divided by the number of processors* (*stretch-span*). Three-dimensional metrics include the *average size of a bounding box* (size of the region defined by the maximum difference between the x, y, and z dimensions of the job allocation), the *average sum of the dimensions of the bounding box*, the *average size of the bounding cube*, the *average number of connected components* per job, as well as metrics based on the maximum and sum-of-squares of these parameters as well as metrics weighted by the running times or divided by the number of processors.

Simulator The simulator assumes a single 8x8x5 grid with one processor per vertex, for a total of 320 processors. This topology is a simplification of the production Cplant architecture at the time the traces were obtained.

Our simulator models the Cplant job queue and scheduler so that the workloads are similar to those on Cplant. When a job arrives it is placed in a job queue. The job queue is sorted first by number of requested processors and then by requested processing time. (Thus, fairness between users and different night and day priorities are not modeled.) Periodically, the scheduler polls to determine which processors are free to execute jobs, and jobs are removed from the front of the queue.

Results Our results suggest that one-dimensional allocation strategies coupled with space-filling curves yield processor locality in higher dimensions. We tested a variety of performance metrics; for the sake of brevity, only a few representative results appear in Figure 1.

We do not know how much the increased locality speeds up the running time. However, the work-multiple parameterization demonstrates that as workloads increase, it becomes harder to obtain processor locality and as workloads decrease it becomes easier. Thus, as the locality of the jobs improves, the running time decreases which further decreases the load, thus further decreasing the running time.

The overall trend is that the processor locality improves through our approach. The simulation results were sufficiently promising to justify implementing the allocation algorithms on Cplant.

4 Experiments

We have performed a limited number of experiments on a 128-processor Cplant machine configured as a 2D mesh. This development machine has an 8×8 mesh of switches with toroidal wraps in both dimensions. Four of the rows have four processors per switch. The other rows contain no compute processors; they contain service and I/O nodes, but fewer than four per switch on average. This pilot study serves as a proof of the concept: careful one-dimensional ordering and allocation to preserve locality within this one-dimensional ordering both improve system throughput.

All runs use identical job streams containing replicas of various-sized instances of a single communication test suite. The communication test suite contains all-to-all broadcast, all-pairs ping-pong (message sent in each direction), and ring communication. Each communication test is repeated a hundred times in each suite. The suite computes a variety of statistics, which computation consumes a small fraction of the total running time. Because locality is most important for jobs with high communication demand, this

Allocation Strategy	Average Makespan	Standard Deviation
baseline (no curve)	5:46:31	0:10:10
best fit (no curve)	5:27:58	0:05:48
baseline (Hilbert)	4:58:52	0:07:37
sum of squares (Hilbert)	4:32:09	0:03:16
first fit (Hilbert)	4:30:22	0:06:09
best fit (Hilbert)	4:25:23	0:03:00

Table 1. Effect of allocation policy on the makespan of the test stream

test suite represents a best-case scenario for the benefits of allocation improvements.

Our test job stream had 91 jobs of size 2, 33 jobs of size 5, 31 jobs of size 25, and 33 jobs of size 30. This gives a small range of "large" (approximately $1/4$ or $1/5$ of the machine) and small jobs. The stream starts with some large jobs to fill up the machine. Small jobs are interspersed among the large ones to cause fragmentation. The last job submitted is small, but it always finishes in front of the last large job. The machine is busy through the release of the last job.

Running times on the Cplant system are nondeterministic. If we run the same job stream twice with the same allocation algorithm, same job ordering, same release times, starting from an empty machine, and having dedicated processors, the running times are not the same. Cplant has inherent nondeterminism in the network. There is variability in time to load executables, in message delivery times, and so on. If the completion time of a single job changes, the options available for the allocation of subsequent jobs also changes. This effect propagates so that later jobs can be allocated significantly better or worse than in a previous run. We can even see different job execution orderings, when a job that is held up for insufficient free processors in one run finds enough free processors in a different run. We found that this nondeterminism did not significantly affect the makespan of the job stream,[1] but the running times of individual job types did vary by 4-16%.

We ran the job stream two to five times (an average of four) for each of the following strategies: first fit and sum of squares with the Hilbert curve, and baseline and best fit with and without the curve.

Table 1 shows the effect of the allocation algorithm on the makespan of the job stream. For this particular job stream, it is better to use a space-filling curve than the row-based ordering. It is also better to pack a job into a consecutive interval if possible. However, the performance of the various bin-packing-based allocation strategies were indistinguishable.

[1]The makespan of a set of jobs is the time between the start of the first job and the completion of the last job.

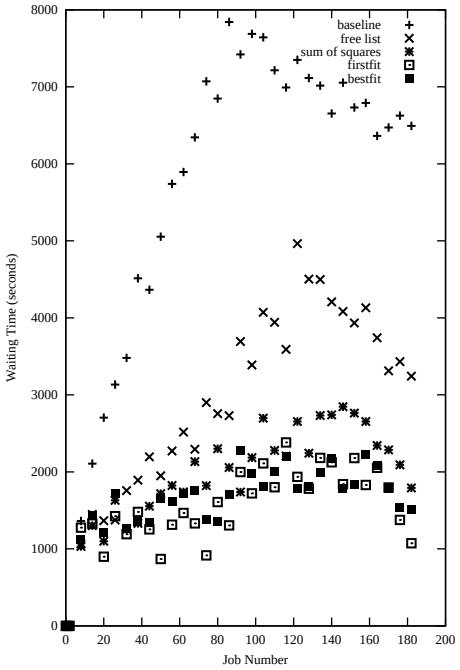


Figure 2. The x-axis shows order of job release. The y-axis shows waiting time. The baseline points use the default processor ordering. All other runs are for the indicated algorithm with the Hilbert curve. Jobs of size 2, 5, and 25 are not represented since these would all be near the line $y = 0$.

Figure 2 shows the waiting times of the 30 node jobs as a function of their order in the job stream. Recall the job stream is identical for all runs, so job order is comparable across runs. Wait time measures the amount of time a job sits in a queue waiting for a sufficient number of free processors. This plot does not include the 2-node, 5-node, and 25-node jobs. Their wait time was so insignificant compared to that of the 30-node jobs that they all sit near the x axis. This figure shows that waiting time is yet another metric that orders the methods the same way with substantial separation.

Figure 3 examines job completion time as a function of two job-fragmentation metrics, one inherent to the topology of the job placement and one used by the algorithms. A natural geometric fragmentation metric is the average of the number of communication hops between processors allocated to this job. Figure 3(a) plots job completion time as a function of this average for the 30-node jobs. Figure 3(b) is a similar plot for span with the Hilbert curve. We do not

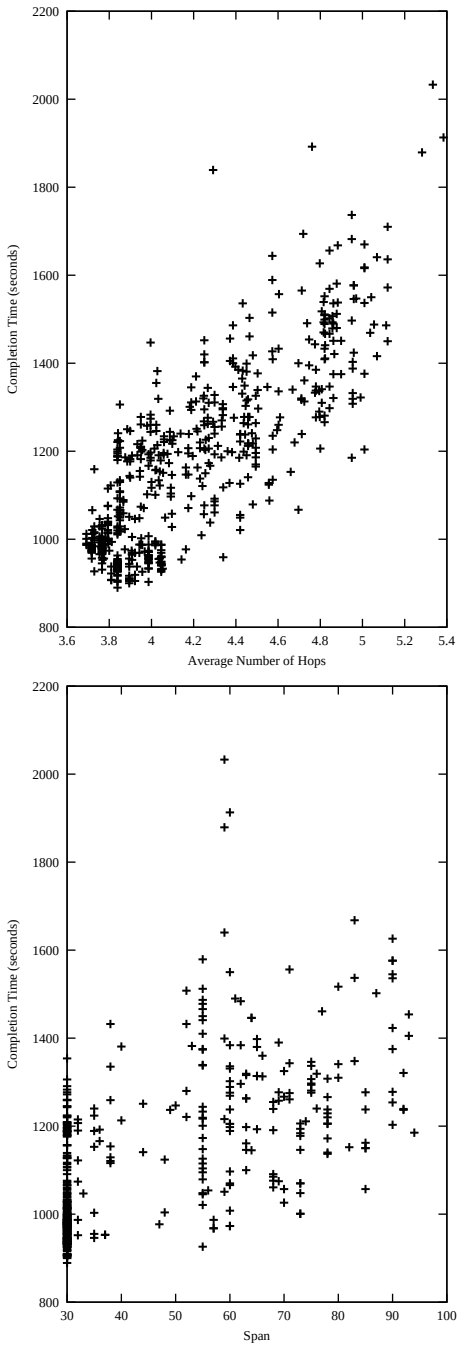


Figure 3. Top: (a) Completion time as a function of the average number of communication hops between processors. Bottom: (b) Completion time as a function of span. Comparison of fragmentation metrics. These plots include only 30-processor jobs across all allocation algorithms. (a) includes all processor orderings. (b) is for Hilbert curve only.

include the 2, 5, and 25-node jobs in these plots. The 2, 5, and 25-node jobs differ enough from the 30-node jobs to add noise to the plots. When the 2, 5, and 25-node jobs are plotted by themselves, they show the same weak correlation on a different scale and with a different slope. These plots include all 30-node jobs placed with all algorithms since the effect of fragmentation should be a function of the amount of fragmentation and independent of how that placement decision was made.

We observe a weak correlation for both metrics. As expected, there is a stronger correlation of completion time to the average number of communication hops because this is a closer match to the topology of the job placement. We are encouraged that the general span metric, which can be easily computed, still tracks this correlation. Neither of these metrics captures the full environment in which a job is run.

5 Concluding Remarks and Future Work

We are cautiously optimistic that the simple general allocation methods discussed in this paper will improve the performance of Cplant systems and apply to more general systems.

Our experiments were limited by the small size of our test machine and the specialized nature of the test jobs/stream. We emphasize that fully rigorous testing will be very challenging because even our limited test suite required 4.5 to 6 hours per run. In order to do these runs, we must take a system away from other users. This is particularly challenging for the 1500+ node production systems. Therefore our future work will have to rely on simulation to some extent. However, these simulations must convincingly account for the effects of locality on job completion time.

Our experiments support the use of span as a fragmentation metric for the design of algorithms and as a measure of locality. Jobs with large span do generally take longer. However, the relationship between span and completion time is not very tight. More work is needed to determine how much of this variability is inherent in the problem and how much results from the imprecision of using span.

We also think that finding the minimum span for a given machine and set of jobs is an interesting theoretical problem. It is related to, yet distinct from, well-studied problems such as memory allocation and bin packing. We have a simple reduction to show that finding the exact minimum span is NP-hard, but do not yet know if it is approximable.

We have also studied these problems in the online setting, where the standard (worse-case) model is competitive analysis [47]. While we omit the proof here, we have been able to show that no online algorithm for minimizing maximum span can achieve a competitive ratio better than $\Omega(n)$ even for randomized strategies.

It may be possible to improve the allocation further

by considering the actual processor topology rather than working entirely within a linear ordering of the processors. When the processors are arranged as a mesh, this makes the allocation problem a multidimensional packing problem, but other processor topologies such as toruses do not have obvious analogs in the packing literature.

It may also be beneficial to consider scheduling and processor allocation together. Currently the allocator is forced to allocate jobs passed from the scheduler even if these jobs must be highly fragmented. Combining these modules might allow more intelligent decisions to be made, but any replacement would need to provide other functionality of the scheduler such as preventing starvation and allocating resources fairly between users.

We intend to evaluate non-greedy allocation methods for jobs that cannot be allocated a contiguous interval. In particular sum-of-square-like algorithms are more likely to leave flexibility in the allocation options for future jobs. In particular, on Cplant, once a job has span of half the machine size, it effectively consumes bandwidth across the entire machine. We particularly wish to avoid this situation.

Subramani et al. [48] indicates that any real machine must allow fragmentation. Our work indicates it is acceptable to fragment, except possibly at the extremes (a large job shattered by its allocation). It may suffice to use a simple one-dimensional strategy as long as the system recognizes situations of shattered jobs that will hurt everyone and queues these jobs.

Acknowledgments

E. Arkin is partially supported by HRL Laboratories, NSF grant CCR-0098172, and Sandia National Laboratories. M. Bender acknowledges support from HRL Laboratories, NSF Grant EIA-0112849, and Sandia National Laboratories. David Bunde is supported in part by Sandia National Laboratories. J. Mitchell is supported in part by HRL Laboratories, NASA Ames Research, NSF grant CCR-0098172, the U.S.-Israel Binational Science Foundation, and Sandia National Laboratories. S. Seiden is supported in part by Sandia National Laboratories and by AFOSR grant No. F49620-01-1-0264.

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.

References

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

[2] V. V. Alexandrov, A. I. Alexeev, and N. D. Gorsky. A recursive algorithm for pattern recognition. In *Proc. IEEE Intl. Conf. Pattern Recognition*, pages 431–433, 1982.

[3] S. Aluru and F. Sevilgen. Parallel domain decomposition and load balancing using space-filling curves. In *Proc. 4th International Conference on High-Performance Computing*, pages 230–235, 1997.

[4] A. Ansari and A. Fineberg. Image data compression and ordering using Peano scan and lot. *IEEE Trans. on Consumer Electronics*, 38(3):436–445, 1992.

[5] M. A. Bender, E. Demaine, and M. Farach-Colton. Cache-oblivious B-trees. In *Proc. 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 399–409, 2000.

[6] M. A. Bender, Z. Duan, J. Iacono, and J. Wu. A locality-preserving cache-oblivious dynamic dictionary. In *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 29–38, 2002.

[7] R. Brightwell, H. E. Fang, and L. Ward. Scalability and performance of CTH on the Computational Plant. In *Proc. 2nd International Conference on Cluster-Based Computing*, 2000.

[8] 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.

[9] R. Brightwell and A. Maccabe. Scalability limitations of VIA-based technologies in supporting MPI. In *Proc. 4th MPI Developer's and User's Conference*, 2000.

[10] G. S. Brodal, R. Fagerberg, and R. Jacob. Cache oblivious search trees via binary trees of small height (extended abstract). In *Proc. 13th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 2002.

[11] S. Chatterjee, A. R. Lebeck, P. K. Patnala, and M. S. Thottethodi. Recursive array layouts and fast matrix multiplication. In *Proc. 11th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 222–231, 1999.

[12] F. R. K. Chung, M. R. Garey, and D. S. Johnson. On packing two-dimensional bins. *SIAM Journal on Algebraic and Discrete Methods*, 3:66–76, 1982.

[13] E. G. Coffman, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: A survey. In D. Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*, chapter 2. PWS Publishing Company, 1997.

[14] D. Coppersmith and P. Raghavan. Multidimensional online bin packing: Algorithms and worst case analysis. *Operations Research Letters*, 8:17–20, 1989.

[15] 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.

[16] 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.

[17] J. Csirik and G. Woeginger. On-line packing and covering problems. In A. Fiat and G. Woeginger, editors, *On-Line Algorithms—The State of the Art*, Lecture Notes in Computer Science, chapter 7. Springer-Verlag, 1998.

[18] Eric Weisstein's World of Mathematics. Hilbert curve. http://mathworld.wolfram.com/HilbertCurve.html .

[19] J. D. Frens and D. S. Wise. Auto-blocking matrix-multiplication or tracking BLAS3 performance from source code. *ACM SIGPLAN Notices*, 32(7):206–216, 1997.

[20] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *Proc. 40th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 285–297, 1999.

[21] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

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

[23] D. S. Greenberg, R. Brightwell, L. A. Fisk, A. McCabe, and R. Riesen. A system software architecture for high-end computing. In *Proc. High Performance Networking and Computing (SC97)*, 1997.

[24] M. Griebel and G. W. Zumbusch. Hash-storage techniques for adaptive multilevel solvers and their domain decomposition parallelization. In J. Mandel, C. Farhat, and X.-C. Cai, editors, *Proc. Domain Decomposition Methods 10, DD10*, number 218 in Contemporary Mathematics, pages 279–286, Providence, 1998. AMS.

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

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

[27] D. S. Johnson. Fast algorithms for bin packing. *J. Comput. Syst. Sci.*, 8:272–314, 1974.

[28] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM J. Comput.*, 3:256–278, 1974.

[29] S. Kamata, R. O. Eason, and Y. Bandou. A new algorithm for N-dimensional Hilbert scanning. *IEEE Trans. on Image Processing*, 8(7):964–973, 1999.

[30] S. Kamata, R. O. Eason, and E. Kawaguchi. An implementation of the Hilbert scanning algorithm and its application to data compression. *IEICE Trans. on Information and Systems*, E76-D(4):420–428, Apr. 1993.

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

[32] M. G. Luby, J. S. Naor, and A. Orda. Tight bounds for dynamic storage allocation. *SIAM Journal on Discrete Mathematics*, 9(1):155–166, 1996.

[33] Y. Matias and A. Shamir. A video scrambling technique based on space filling curves. In *Proc. Advances in Cryptology—CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pages 398–417. Springer-Verlag, 1987.

[34] 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.

[35] 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.

[36] G. Peano. Sur une courbe, qui remplit toute une aire plane. *Math. Annalen*, pages 157–160, 1890.

[37] J. R. Pilkington and S. B. Baden. Dynamic partitioning of non-uniform structured workloads with spacefilling curves:. *IEEE Transactions on Parallel and Distributed Systems*, 7(3):288–300, 1996.

[38] H. Prokop. Cache-oblivious algorithms. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 1999.

[39] N. Rahman, R. Cole, and R. Raman. Optimized predecessor data structures for internal memory. In *Proc. 5th Workshop on Algorithms Engineering (WAE)*, 2001.

[40] R. Riesen, R. Brightwell, L. A. Fisk, T. Hudson, J. Otto, and A. B. Maccabe. Cplant. In *Proc. 2nd Extreme Linux workshop at the 1999 USENIX Annual Technical Conference*, 1999.

[41] J. M. Robson. An estimate of the store size necessary for dynamic storage allocation. *Journal of the ACM*, 18(3):416–423, 1971.

[42] J. M. Robson. Bounds for some functions concerning dynamic storage allocation. *Journal of the ACM*, 21(3):491–499, 1974.

[43] H. Sagan. *Space-Filling Curves*. Springer-Verlag, 1994.

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

[45] S. Seiden and R. van Stee. New bounds for multi-dimensional packing. In *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 486–495, 2002.

[46] J. Sgall. On-line scheduling. In A. Fiat and G. Woeginger, editors, *On-Line Algorithms—The State of the Art*, Lecture Notes in Computer Science, chapter 9. Springer-Verlag, 1998.

[47] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[48] 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.

[49] K. S. Thyagarajan and S. Chatterjee. Fractal scanning for image compression. In *Conference Record of the 25th Asilomar Conference on Signals, Systems and Computers*, pages 467–471, 1992.

Communication-Aware Processor Allocation for Supercomputers*

Michael A. Bender[1], David P. Bunde[2], Erik D. Demaine[3], Sándor P. Fekete[4], Vitus J. Leung[5], Henk Meijer[6], and Cynthia A. Phillips[5]

[1] Department of Computer Science, SUNY Stony Brook, Stony Brook, NY 11794-4400, USA
bender@cs.sunysb.edu
[2] Department of Computer Science, University of Illinois, Urbana, IL 61801, USA
bunde@uiuc.edu
[3] MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA 02139, USA
edemaine@mit.edu
[4] Dept. of Mathematical Optimization, Braunschweig University of Technology, 38106 Braunschweig, Germany
s.fekete@tu-bs.de
[5] Discrete Algorithms & Math Department, Sandia National Laboratories, Albuquerque, NM 87185-1110, USA
{vjleung, caphill}@sandia.gov
[6] Dept. of Computing and Information Science, Queen's University, Kingston, Ontario, K7L 3N6, Canada
henk@cs.queensu.ca

Abstract. We give processor-allocation algorithms for grid architectures, where the objective is to select processors from a set of available processors to minimize the average number of communication hops.

The associated clustering problem is as follows: Given n points in $\Re^d$, find a size-k subset with minimum average pairwise L_1 distance. We present a natural approximation algorithm and show that it is a $\frac{7}{4}$-approximation for 2D grids. In d dimensions, the approximation guarantee is $2 - \frac{1}{2d}$, which is tight. We also give a polynomial-time approximation scheme (PTAS) for constant dimension d and report on experimental results.

1 Introduction

We give processor-allocation algorithms for grid architectures. Our objective is to select processors to run a job from a set of available processors so that the average number of communication hops between processors assigned to the job is minimized. Our problem is restated as follows: given a set P of n points in $\Re^d$, find a subset S of k points with minimum average pairwise L_1 distance.

* Extended Abstract. A full version is available as [5].

F. Dehne, A. López-Ortiz, and J.-R. Sack (Eds.): WADS 2005, LNCS 3608, pp. 169–181, 2005.

Motivation: Processor Allocation in Supercomputers. Our algorithmic work is motivated by a problem in the operation of supercomputers. The supercomputer for which we targeted our simulations and experiments is called Computational Plant or Cplant [7, 25], a commodity-based supercomputer developed at Sandia National Laboratories. In Cplant, a scheduler selects the next job to run based on priority. The allocator then independently places the job on a set of processors which exclusively run that job to completion. Security constraints forbid migration, preemption, or multitasking. To obtain maximum throughput in a network-limited computing system, the processors allocated to a single job should be physically near each other. This placement reduces communication costs and avoids bandwidth contention caused by overlapping jobs. Experiments have shown that processor allocation affects throughput on a range of architectures [3,17,20,21,23]. Several papers suggest that minimizing the *average number of communication hops* is an appropriate metric for job placement [20, 21, 16]. Experiments with a communication test suite demonstrate that this metric correlates with a job's completion time [17].

Early processor-allocation algorithms allocate only convex sets of processors to each job [18, 9, 29, 6]. For such allocations, each job's communication can be routed entirely within processors assigned to that job, so jobs contend only with themselves. But requiring convex allocations reduces the achievable system utilization to levels unacceptable for a government-audited system [15, 26].

Recent work [19,22,8,17,26] allows discontiguous allocation of processors but tries to cluster them and minimize contention with previously allocated jobs. Mache, Lo, and Windisch [22] propose the MC algorithm for grid architectures: For each free processor, algorithm MC evaluates the quality of an allocation centered on that processor. It counts the number of free processors within a submesh of the requested size centered on the given processor and within "shells" of processors around this submesh. The cost of the allocation is the sum of the shell numbers in which free processors occur; see Figure 1 reproduced from [22]. MC chooses the allocation with lowest cost. Since users of Cplant do not request processors in a particular shape, in this paper, we consider MC1x1, a variant in which shell 0 is 1×1 and subsequent shells grow in the same way as in MC.

Until recently, processor allocation on the Cplant system was *not* based on the locations of the free processors. The allocator simply verified that enough processors were free before dispatching a job. The current allocator uses space-filling curves and 1D bin-packing techniques based upon work of Leung et al. [17].

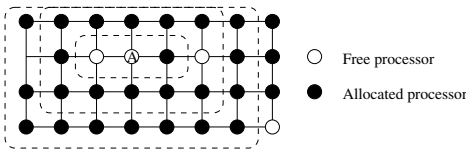


Fig. 1. Illustration of MC: Shells around processor A for a 3×1 request

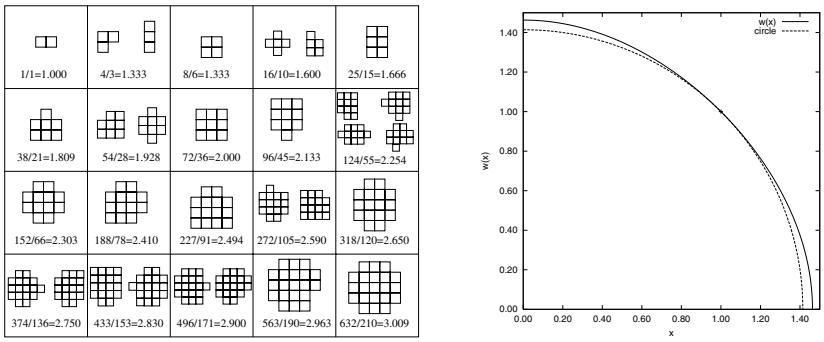


Fig. 2. (Left) Optimal unconstrained clusters for small values of k; numbers shown are the average L_1 distances, with truncated decimal values. (Right) Plot from [4] of a quarter of the optimal limiting boundary curve; the dotted line is a circle

Related Algorithmic Work. Krumke et al. [16] consider a generalization of our problem on arbitrary topologies for several measures of locality, motivated by allocation on the CM5. They prove it is NP-hard to approximate average pairwise distance in general, but give a 2-approximation for distances obeying the triangle inequality.

A natural special case of the allocation problem is the *unconstrained* problem, in the absence of occupied processors: For any number k, find k grid points minimizing average pairwise L_1 distance. For moderate values of k, these sets can be found by exhaustive search; see Figure 2. The resulting shapes appear to approximate some "ideal" rounded shape, with better and better approximation for growing k. Karp et al. [14] and Bender et al. [4] study the exact nature of this shape. Surprisingly, the resulting convex curve can only be described by a differential equation; the closed-form solution is unknown. The complexity of this special case remains open, but its mathematical difficulty emphasizes the hardness of obtaining good solutions for the general constrained problem.

In reconfigurable computing on field-programmable gate arrays (FPGAs), varying processor sizes give rise to a generalization of our problem: place a set of rectangular modules on a grid to minimize the overall weighted sum of L_1 distances between modules. Ahmadinia et al. [1] give an optimal $\Theta(n \log n)$ algorithm for finding an optimal feasible location for a module given a set of n existing modules. At this point, no results are known for the general off-line problem (place n modules simultaneously) or for on-line versions.

Another related problem is *min-sum k-clustering*: separate a graph into k clusters to minimize the sum of distances between nodes in the same cluster. For general graphs, Sahni and Gonzalez [24] show it is NP-hard to approximate this problem to within any constant factor for $k \geq 3$. In a metric space, Guttmann-Beck and Hassin [12] give a 2-approximation, Indyk [13] gives a PTAS for $k = 2$, and Bartel et al. [2] give an $O((1/\epsilon) \log^{1+\epsilon} n)$-approximation for general k.

Fekete and Meijer [11] consider the problem of *maximizing* the average L_1 distance. They give a PTAS for this *dispersion* problem in $\Re^d$ for constant d, and show that an optimal set of any fixed size can be found in $O(n)$ time.

Our Results. We develop algorithms for minimizing the average L_1 distance between allocated processors in a mesh supercomputer. A greedy heuristic we analyze called MM and a 3D version of MC1x1 have been implemented on Cplant. In particular, we give the following results:

- We prove that MM is a $\frac{7}{4}$-approximation algorithm for $2D$ grids, reducing the previous best factor of 2 [16], and we show that this analysis is tight.
- We present a simple generalization to general d-dimensional space with fixed d and prove that the algorithm gives a $2-\frac{1}{2d}$-approximation algorithm, which is tight.
- We give an efficient polynomial-time approximation scheme (PTAS) for points in $\Re^d$ for constant d.
- Using simulations, we compare the allocation performance of our algorithm to that of other algorithms. As a byproduct, we get insight on how to place a stream of jobs in an online setting.

In addition, we have a number of other results whose details are omitted due to space constraints: We have a linear-time exact algorithm for the 1D case based on dynamic programming. We prove that the d-dimensional version of MC1x1 has approximation factor at most d times that of MM. We have an algorithm to solve the 2-dimensional case for $k = 3$ in time $O(n \log n)$.

2 Manhattan Median Algorithm for Two-Dimensional Point Sets

2.1 Median-Based Algorithms

Given a set S of k points in the plane, a point that minimizes the total L_1 distance to these points is called an (L_1) median. Given the nature of L_1 distances, this is a point whose x-coordinate (resp. y-coordinate) is the median of the x (resp. y) values of the given point set. We can always pick a median whose coordinates are from the coordinates in S. There is a unique median if k is odd; if k is even, possible median coordinates may form intervals.

The natural greedy algorithm for our clustering problem is as follows:

> Consider the $O(n^2)$ intersection points of the horizontal and vertical lines through the points in P. For each of these points p do:
>
> 1. Take the k points closest to p (using the L_1 metric), breaking ties arbitrarily.
> 2. Compute the total pairwise distance between all k points.
>
> Return the set of k points with smallest total pairwise distance.

We call this strategy MM, for **M**anhattan **M**edian. We prove that MM is a $\frac{7}{4}$-approximation on 2D meshes. (Note that Krumke et al. [16] call this algorithm Gen-Alg and show it is a 2-approximation in arbitrary metric spaces.)

2.2 Analysis of the Algorithm

For $S \subseteq P$, let $|S|$ denote the sum of L_1 distances between points in S. For a point p in the plane, we use p_x and p_y to denote its x- and y-coordinates respectively.

Lemma 1. *MM is not better than a* $7/4$ *approximation.*

Proof. For a class of examples establishing the lower bound, consider the situation shown in Figure 3. For any $\epsilon > 0$, it has clusters of $k/2$ points at $(0, 0)$ and $(1, 0)$. In addition, it has clusters of $k/8$ points at $(0, \pm(1-\epsilon))$, $(1, \pm(1-\epsilon))$, $(2-\epsilon, 0)$, and $(-1+\epsilon, 0)$. The best choices of median are $(0, 0)$ and $(1, 0)$, which yield a total distance of $7k^2(1-\Theta(\epsilon))/16$. The optimal solution is the points at $(0, 0)$ and $(1, 0)$, which yield a total distance of $k^2/4$. □

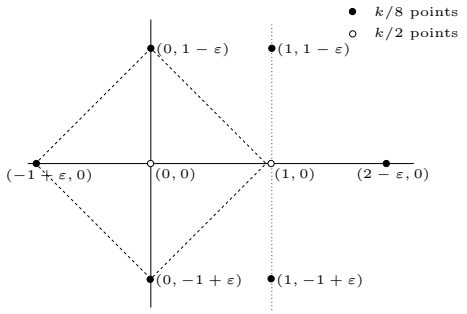


Fig. 3. A class of examples where MM yields a ratio of 7/4

Now we show that 7/4 is indeed the worst-case bound. We focus on possible worst-case arrangements and use local optimality to restrict the possible arrangements until the claim follows.

Let OPT be a subset of P of size k for which $|OPT|$ is minimum. Without loss of generality assume that the origin is a median point of OPT. This means that the number of points of OPT with positive or negative x- or y-coordinates is at most $k/2$. Let MM be the set of k points closest to the origin. (Since this is one candidate solution for the algorithm, its sum of pairwise distances is at least as high as that of the solution returned by the algorithm.)

Without loss of generality, assume that the largest distance of a point in MM to the origin is 1, so MM lies in the unit circle C. We say that points are either inside C, on C, or outside C. All points of P inside C are in MM and at least some points on C are in MM. If there are more than k points on and inside C, we select all points inside C plus those points on C maximizing $|MM|$.

Clearly $1 \leq |MM|/|OPT|$. Let ρ_k be the supremum of $|MM|/|OPT|$ over all input configurations P. By assuming that ties are broken badly, we can assume that there is a configuration $S \subseteq P$ for which $|MM|/|OPT| = \rho_k$:

Lemma 2. *For any* n *and* k*, there are point sets* P *with* $|P| = n$ *for which* $|MM|/|OPT|$ *attains the value* ρ_k*.*

Proof. The set of arrangements of n points in the unit circle C is a compact set in $2d$-dimensional space. By our assumption on breaking ties, $|MM|/|OPT|$ is upper semi-continuous, so it attains a maximum. □

For $k \leq 8n/11$ we show $|MM|$ is at most $7/4$ times larger than $|OPT|$.

Lemma 3. *For $k \leq 8n/11$ we have $\rho_k \leq 7/4$.*

Sketch of Proof. We assume that we have a point set P for which ρ_k is equal to $7/4$. We can assume without loss of generality that $P = MM \cup OPT$. If there is a point $p \in P$ that does not lie in a corner of C or on the origin, we look at all points that lie on the axis-parallel rectangle through p with corners on C. We move these points simultaneously, in such a way that they stay on an axis-parallel rectangle with corners on C. This move changes $|MM|$ by some small amount δ_a and $|OPT|$ by some amount δ_o. However if we move all points in the opposite direction $|MM|$ and $|OPT|$ change by $-\delta_a$ and $-\delta_o$ respectively. So if $\delta_a/\delta_o \neq \rho_k$, one of these two moves increases $|MM|/|OPT|$, which is impossible. If $\delta_a/\delta_o = \rho_k$ we keep moving the points in the same direction until there is a combinatorial change in P. We can then repeat this argument until all points of P lie on a corner of C or on the origin.

It is now not too hard to show that the ratio MM/OPT is maximal if there are $k/2$ points at the origin, $k/2$ points in one corner of C and $k/8$ points at each of the other three corners. So we have $|MM|/|OPT| = 7/4$. Notice that n has to be at least $11k/8$ for this value to be obtained. □

For larger values of k it can be shown that ρ_k decreases, so we summarize:

Theorem 1. *MM is a $7/4$-approximation algorithm for minimizing the sum of pairwise L_1 distances in a 2D mesh.*

3 PTAS for Two Dimensions

Let $w(S,T)$ be the sum of all the distances from points in S to points in T. Let $w_x(S,T)$ and $w_y(S,T)$ be the sum of x- and y- distances from points in S to points in T, respectively. So $w(S,T) = w_x(S,T) + w_y(S,T)$. Let $w(S) = w(S,S)$, $w_x(S) = w_x(S,S)$, and $w_y(S) = w_y(S,S)$. We call $w(S)$ the *weight* of S.

Let $S = \{s_0, s_1, \ldots, s_{k-1}\}$ be a minimum-weight subset of P, where k is an integer greater than 1. We label the x- and y-coordinates of a point $s \in S$ by some (x_a, y_b) with $0 \leq a < k$ and $0 \leq b < k$ such that $x_0 \leq x_1 \leq \ldots \leq x_{k-1}$ and $y_0 \leq y_1 \leq \ldots \leq y_{k-1}$. (Note that in general, $a \neq b$ for a point $s = (x_a, y_b)$.) We can derive the following equations: $w_x(S) = (k-1)(x_{k-1} - x_0) + (k-3)(x_{k-2} - x_1) + \ \ldots$ and $w_y(S) = (k-1)(y_{k-1} - y_0) + (k-3)(y_{k-2} - y_1) + \ \ldots$ We show that there is a polynomial-time approximation scheme (PTAS), i.e., for any fixed positive $m = 1/\varepsilon$, there is a polynomial approximation algorithm that finds a solution within $(1+\varepsilon)$ of the optimum.

The basic idea is similar to the one used by Fekete and Meijer [11] to select a set of points maximizing the overall distance: We find (by enumeration) a subdivision of an optimal solution into $m \times m$ rectangular cells C_{ij}, each containing a

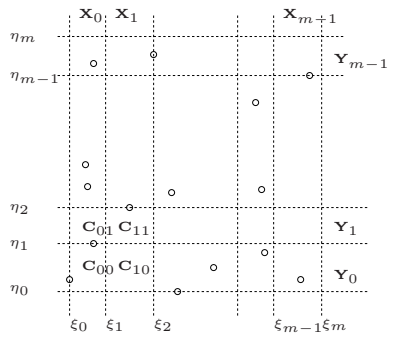


Fig. 4. Dividing the point set in horizontal and vertical strips

specific number k_{ij} of selected points. The points from each cell C_{ij} are selected in a way that minimizes the total distance to all other cells except for the $m-1$ cells in the same "horizontal" strip or the $m-1$ cells in the same "vertical" strip. As it turns out, this can be done in a way that the total neglected distance within the strips is bounded by a small fraction of the weight of an optimal solution, yielding the desired approximation property. See Figure 4 for the setup.

For ease of presentation, we assume that k is a multiple of m and $m > 2$. Approximation algorithms for other values of k can be constructed in a similar fashion. Consider a division of the plane by a set of $m+1$ x-coordinates $\xi_0 \leq \xi_1 \leq \ldots \leq \xi_m$. Let $X_i := \{p = (x, y) \mid \xi_i \leq x \leq \xi_{i+1}\}$ be the vertical strip between coordinates ξ_i and ξ_{i+1}. By enumeration of possible values of $\xi_0, \ldots, \xi_m$ we may assume that each of the m strips X_i contains precisely k/m points of an optimal solution. (A small perturbation does not change optimality or approximation properties of solutions. Thus, without loss of generality, we assume that no pair of points share either x-coordinate or y-coordinate.)

In a similar manner, assume we know $m+1$ y-coordinates $\eta_0 \leq \eta_1 \leq \ldots \leq \eta_m$ so that an optimal solution has precisely k/m points in each horizontal strip $Y_i := \{p = (x, y) \mid \eta_i \leq y \leq \eta_{i+1}\}$.

Let $C_{ij} := X_i \cap Y_j$, and let k_{ij} be the number of points in OPT that are chosen from C_{ij}. Since for all $i, j \in \{1, 2, \ldots, m\}$,

$$\sum_{0 \leq l < m} k_{lj} = \sum_{0 \leq l < m} k_{il} = k/m,$$

we may assume by enumeration over the $O(k^m)$ possible partitions of k/m into m pieces that we know all the numbers k_{ij}.

Finally, define the vector $\nabla_{ij} := ((2i + 1 - m)k/m, (2j + 1 - m)k/m)$. Our approximation algorithm is as follows: from each cell C_{ij}, choose k_{ij} points that are minimum in direction ∇_{ij}, i.e., select points $p = (x, y)$ for which $(x(2i + 1 - m)k/m, y(2j + 1 - m)k/m)$ is minimum. For an illustration, see Figure 5.

It can be shown that selecting points of C_{ij} this way minimizes the sum of x-distances to points not in X_i and the sum of y-distances to points not in Y_j.

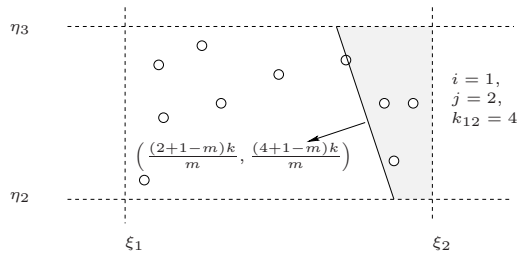


Fig. 5. Selecting points in cell C_{12}

The details are somewhat technical and are described in the full version of the paper [5]. We summarize:

Theorem 2. *The problem of selecting a subset of minimum total L_1 distance for a set of points in $\Re^2$ allows a PTAS.*

4 Higher-Dimensional Spaces

Using the same techniques, we also generalize our results to higher dimensions. We start by describing the performance of *MM*.

4.1 $\left(2 - \frac{1}{2d}\right)$ -Approximation

As in two-dimensional space, *MM* enumerates over the $O(n^d)$ possible medians. For each median, it constructs a candidate solution of the k closest points.

Lemma 4. *MM is not better than a $2 - 1/(2d)$ approximation.*

Proof. We construct an example based on the cross-polytope in d dimensions, i.e., the d-dimensional L_1 unit ball. Let $\varepsilon > 0$. Denote the origin with O and the i^{th} unit vector with e_i. The example has $k/2$ points at O and $O + e_1$. In addition, there are $k/(4d)$ points at $O - (1 - \varepsilon)e_1$, $O + (2 - \varepsilon)e_1$, $O \pm (1 - \varepsilon)e_i$ for $i = 2, \ldots, d$, and $O + e_1 \pm (1 - \varepsilon)e_i$ for $i = 2, \ldots, d$. *MM* does best with O or $O + e_1$ as median, giving a total distance of $(k^2/4)\,(2 - 1/(2d))\,(1 + \Theta(\varepsilon))$. Optimal is the points at O and $O + e_1$, giving a total distance of $k^2/4$. □

Establishing a matching upper bound can be done analogously to Section 2. Lemma 2 holds for general dimensions. The rest is based on the following lemma:

Lemma 5. *Worst-case arrangements for MM can be assumed to have all points at positions $(0, \ldots, 0)$ and $\pm e_i$, where e_i is the ith unit vector.*

Sketch of Proof. Consider a worst-case arrangement within the cross-polytope centered at the origin with radius 1. Local moves consist of continuous changes in point coordinates, performed in such a way that the number of coordinate values is kept. This means that to move a point having a coordinate value different from

$0, 1, -1$, then all other points sharing that coordinate value are moved to keep the identical coordinates the same, analogous to the proof of Lemma 3.

Note that under these moves, the functions OPT and MM are locally linear, so the ratio of MM and OPT is locally constant, strictly increasing, or strictly decreasing. If a move decreases the ratio, the opposite move increases it, contradicting the assumption that the arrangement is worst-case.

If the ratio is locally constant during a move, it will continue to be extremal until an event occurs, i.e., when the number of coordinate identities between points increases, or the number of point coordinates at $0, 1, -1$ increase. While there are points with coordinates different from $0, 1, -1$, there is always a move that decreases the total degrees of freedom, until all dn degrees of freedom have been eliminated. Thus, we can always reach an arrangement with point coordinates values from the set $\{0, 1, -1\}$. These leaves the origin and the $2d$ positions $\pm e_i$ as only positions within the cross-polytope. □

The restricted set of arrangements can be evaluated with symmetry to yield

Theorem 3. *For points lying in d-dimensional space, MM is a* $2-1/2d$*-approximation algorithm, which is tight.*

4.2 PTAS for General Dimensions

Theorem 4. *For any fixed d, the problem of selecting a subset of minimum total* L_1 *distance for a set of points in* $\Re^d$ *allows a PTAS.*

Sketch of Proof. For $m = \Theta(1/\varepsilon)$, we subdivide the set of n points with $d(m+1)$ axis-aligned hyperplanes, such that $(m+1)$ are normal for each coordinate direction. Moreover, any set of $(m+1)$ hyperplanes normal to the same coordinate axis is assumed to subdivide the optimal solution into k/m equal subsets, called *slices*. Enumeration of all possible structures of this type yields a total of n^m choices of hyperplanes in each coordinate, for a total of n^{md} possible choices. For each choice, we have a total of m^d cells, each containing between 0 and k points; thus, there are $O(m^{kd})$ different distributions of cardinalities to the different cells. As in the two-dimensional case, each cell picks the assigned number of points extremal in its gradient direction.

It is easily seen that for each coordinate x_i, the above choice minimizes the total sum of x_i-distances between points not in the same x_i-slice. The remaining technical part (showing that the sum of distances within slices are small compared to the distances between different slices) is analogous to the details described in the full version of the paper [5] and omitted. □

5 Experiments

The work discussed so far is motivated by the allocation of a single job. In the following, we examine how well our algorithms allocate streams of jobs; now the set of free processors available for each job depends on previous allocations.

Table 1. Average sum of pairwise distances when the decision algorithm makes allocations with input provided by the situation algorithm

Situation	Decision Algorithm			
Algorithm	MC1x1	MM	MM+Inc	HilbertBF
MC1x1	5256	5218	5207	5432
MM	5323	5285	5276	5531
MM+Inc	5319	5281	5269	5495
HilbertBF	5090	5059	5046	5207

To understand the interaction between the quality of an individual allocation and the quality of future allocations, we ran a simulation involving pairs of algorithms. One algorithm, the *situation algorithm*, places each job. This determines the free processors available for the next job. Each allocation decision serves as an input to the other algorithm, the *decision algorithm*. Each entry in Table 1 represents the average sum of pairwise distances for the decision algorithm with processor availability determined by the situation algorithm.

Our simulation used the algorithms MC1x1, MM, MM+Inc, and HilbertBF. MM+Inc uses local improvement on the allocation of MM, replacing an allocated processor with an excluded processor that improves average pairwise distance until it reaches a local minimum. HilbertBF is the 1-dimensional strategy of Leung et al. [17] used on Cplant. The simulation used the LLNL Cray T3D trace[1] from the Parallel Workloads Archive [10]. This trace has 21323 jobs run on a machine with 256 processors, treated as a 16×16 mesh in the simulation.

In each row, the algorithms are ranked in the order MM+Inc, MM, MC1x1, and HilbertBF. This is consistent with the worst-case performance bounds; MM is a 7/4-approximation, MC1x1 is a 7/2-approximation, and HilbertBF has an unbounded ratio[2].

6 Conclusions

The algorithmic work described in this paper is one step toward developing algorithms for scheduling mesh-connected network-limited multiprocessors. We have given provably good algorithms to allocate a single job. The next step is to study the allocation of job sequences, a markedly different algorithmic challenge.

The difference between making a single allocation and a sequence of allocations is already illustrated by the diagonal entries in Table 1, where the free processors depend on the same algorithm's previous decisions. These give the ranking (from best to worst) HilbertBF, MC1x1, MM+Inc, and MM. The locally better decisions of MM+Inc seem to paint the algorithm into a corner over time. Figures 1 and 2 help explain why. When starting on an empty grid, MC

[1] We thank Moe Jette and Bill Nitzberg for providing the LLNL and NASA Ames iPSC/860 traces, respectively, to the Parallel Workloads Archive.

[2] On an $N \times N$ mesh, the approximation ratio can be $\Omega(N)$.

produces connected rectangular shapes. Locally, these shapes are slightly worse than the round shapes produced by MM, but rectangles have better packing properties because they avoid small patches of isolated grid nodes.

We confirmed this behavior over an entire trace using Procsimity [27, 28], which simulates messages moving through the network. We ran the NASA Ames iPSC/860 trace[3] from the Parallel Workloads Archive [10], scaling down the number of processors for each job by a factor of 4. This made the trace run on a machine with 32 processors, allowing us to find the greedy placement that minimizes average pairwise distance at that step. For average job flow time, MC1x1 was best, followed by MM, and then greedy. We did not run MM+Inc in this simulation. HilbertBF was much worse than all three of the algorithms mentioned in part due to difficulties using it on a nonsquare mesh.

Thus, the online problem in an iterated scenario is the most interesting open problem. We believe that a natural attack may be to consider online packing of rectangular shapes of given area. We plan to pursue this in future work.

Acknowledgments

We thank Jens Mache for informative discussions on processor allocation. Michael Bender was partially supported by Sandia and NSF Grants EIA-0112849 and CCR-0208670. David Bunde was partially supported by Sandia and NSF grant CCR 0093348. Sándor Fekete was partially supported by DFG grants FE 407/7 and FE 407/8. Henk Meijer was partially supported by NSERC. Sandia is a multipurpose laboratory operated by Sandia Corporation, a Lockheed-Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

References

1. A. Ahmadinia, C.Bobda, S. Fekete, J.Teich, and J. der Veen. Optimal routing-conscious dynamic placement for reconfigurable computing. In *International Conference on Field-Programmable Logic and its applications*, volume 3203 of *LNCS*, pages 847–851. Springer, 2004.
2. Y. Bartal, M. Charikar, and D. Raz. Approximating min-sum k-clustering in metric spaces. In *Proc. 33rd Symp. on Theory of Computation*, pages 11–20, 2001.
3. 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.
4. C. M. Bender, M. A. Bender, E. D. Demaine, and S. P. Fekete. What is the optimal shape of a city? *J. Physics A: Mathematical and General*, 37:147–159, 2004.
5. M. A. Bender, D. P. Bunde, E. D. Demaine, S. P. Fekete, V. J. Leung, H. Meijer, and C. A. Phillips. Communication-aware processor allocation for supercomputers. Technical Report cs.DS/0407058, Computing Research Repository, `http://arxiv.org/abs/cs.DS/0407058`, 2004.

6. 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.
7. 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.
8. 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.
9. P.-J. Chuang and N.-F. Tzeng. An efficient submesh allocation strategy for mesh computer systems. In *Proc. Int. Conf. Dist. Comp. Systems*, pages 256–263, 1991.
10. D. Feitelson. The parallel workloads archive. `http://www.cs.huji.ac.il/labs/parallel/workload/index.html`.
11. S. P. Fekete and H. Meijer. Maximum dispersion and geometric maximum weight cliques. *Algorithmica*, 38:501–511, 2004.
12. N. Guttmann-Beck and R. Hassin. Approximation algorithms for minimum sum p-clustering. *Disc. Appl. Math.*, 89:125–142, 1998.
13. P. Indyk. A sublinear time approximation scheme for clustering in metric spaces. In *Proc. 40th Ann. IEEE Symp. Found. Comp. Sci. (FOCS)*, pages 154–159, 1999.
14. R. M. Karp, A. C. McKellar, and C. K. Wong. Near-optimal solutions to a 2-dimensional placement problem. *SIAM Journal on Computing*, 4:271–286, 1975.
15. 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.
16. S. Krumke, M. Marathe, H. Noltemeier, V. Radhakrishnan, S. Ravi, and D. Rosenkrantz. Compact location problems. *Th. Comp. Sci.*, 181:379–404, 1997.
17. V. Leung, E. Arkin, M. Bender, D. Bunde, J. Johnston, A. Lal, J. Mitchell, C. Phillips, and 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. 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.
19. V. Lo, K. Windisch, W. Liu, and B. Nitzberg. Non-contiguous processor allocation algorithms for mesh-connected multicomputers. *IEEE Transactions on Parallel and Distributed Computing*, 8(7), 1997.
20. J. Mache and V. Lo. Dispersal metrics for non-contiguous processor allocation. Technical Report CIS-TR-96-13, University of Oregon, 1996.
21. J. Mache and V. Lo. The effects of dispersal on message-passing contention in processor allocation strategies. In *Proc. Third Joint Conf. on Information Sciences, Sessions on Parallel and Distributed Processing*, volume 3, pages 223–226, 1997.
22. J. Mache, V. Lo, and K. Windisch. Minimizing message-passing contention in fragmentation-free processor allocation. In *Proc. 10th Intern. Conf. Parallel and Distributed Computing Systems*, pages 120–124, 1997.
23. S. Moore and L. Ni. The effects of network contention on processor allocation strategies. In *Proc. 10th Int. Par. Proc. Symp.*, pages 268–274, 1996.
24. S. Sahni and T. Gonzalez. p-complete approximation problems. *JACM*, 23(3): 555–565, 1976.
25. Sandia National Laboratories. The Computational Plant Project. `http://www.cs .sandia.gov/cplant`.

26. 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.
27. University of Oregon Resource Allocation Group. Procsimity. `http://www.cs.uoregon.edu/research/DistributedComputing/ProcSimity.html\%`.
28. 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.
29. Y. Zhu. Efficient processor allocation strategies for mesh-connected parallel computers. *J. Parallel and Distributed Computing*, 16:328–337, 1992.