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

Vitus J. 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 free list strategy previously used on Cplant. These new allocation strategies are implemented in the new release of the Cplant System Software, Version 2.0, phased into the Cplant systems[1] at Sandia by May 2002.

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

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

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

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

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

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

[1] 128 to 1536 node, two and three-dimesional meshes with wrap around in two dimensions

1. Introduction

As part of the Accelerated Strategic Computing Initiative [32], 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 also developing the Computational Plant or Cplant [46, 24, 9, 42, 7, 8]. This paper describes resource-allocation algorithms to optimize processor locality in Cplant and other 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 [15] or PBS [36] 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 significantly, both jobs' running times are approximately doubled. Subramani et al. [50] 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 may get truncated if it does not complete by 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 switches 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* (also called *fractal 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. [50] 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, using 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. [50] 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 our problem 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 [43, 44, 33]. One objective 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 [22] and bin packing was one of the first problems to be studied in terms of both online and offline approximability [27, 28, 29]. 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 [47]. For a more detailed review of bin packing, see the surveys [13, 18]. 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. Arriving objects 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 [27] places a new object in the first bin in which it fits. Best Fit [27] 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 [17] is also available. This algorithm bases its decisions on a vector N, where $N(i)$ is the number of bins with 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 [16].

Other researchers have used space-filling curves for a variety of problems. Originally, space-filling curves were introduced by Hilbert [26] and Peano [38]. Recent presentations appear in [19] and [45]. Hilbert curves have been shown to preserve several measures of "locality" [35, 23]. An alternative with better performance in two dimensions is given in [37]. Generalizations of Hilbert curves to higher dimensions are given in [1]. Specific applications include matrix multiplication [11, 20], domain decomposition [3, 25, 39], and image processing [2, 34, 4, 51, 31, 30]. They are also a standard tool in the creation of *cache-oblivious algorithms* [21, 40, 5, 41, 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 [48].

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 possible. When a job j requiring p_j processors is dispatched by the scheduler, the allocator queries the system to determine which processors are free and gathers these processors into a sorted list. Job j 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, 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. (We omit the full formulation.) 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. The modification 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 linear 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 (including the endpoints) are involved in routing a message between these two processors. These are the processors "owned" by job j plus 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 is easier to determine the size of the largest "free" set of processors, accounting for the ring wraparound, and subtract it from the number of processors. Let $r_{j,i}$ be the ith-smallest rank of a processor in M_j for $i = 0 \ldots p_j - 1$. Then we define $s_j^w = |P| - \max(\max_{i=0}^{p_j-2} r_{j,i+1} - r_{j,i} - 1, |P| - 1 - r_{j,p_j-1} + r_{j,0})$. 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, it is allocated across multiple intervals. We choose the allocation that *minimizes the span* of the job. In a tie we start the job at 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*. All the jobs were increased by this work multiple. Increasing running times makes processors busier since jobs are in the system for a longer amount of time. Note that we do not change release times so that the interaction between jobs are different. 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*. The results for both types of inputs were similar, so 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. The gains in system throughput (described in Section 4) are consistent with these simulation results.

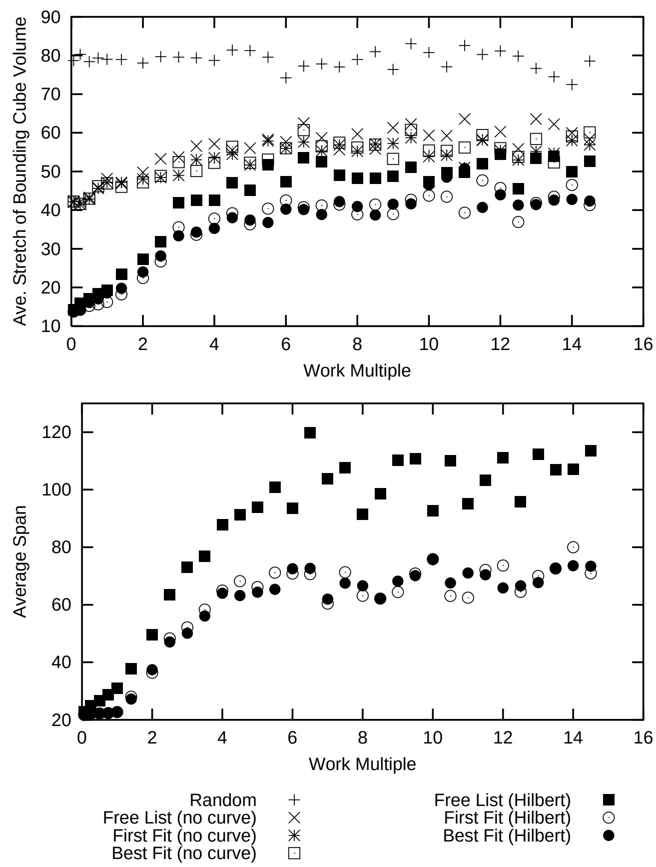


Figure 1. Top: The average bounding cube divided by the smallest bounding cube. Bottom: The average span of jobs. The x-axis plots the work multiple, where we simulate the strategies on a range of workloads.

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 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

Allocation Strategy	Average Makespan	Standard Deviation
Free List (no curve)	5:46:31	0:10:10
Best Fit (no curve)	5:27:58	0:05:48
Free List (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

repeated a hundred times in each suite. The suite computes a variety of statistics, whose computation consumes a small fraction of the total running time. Because locality is most important for jobs with high communication demand, this 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 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,[2] 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 Free List and Best Fit with and without the curve.

Table 1 shows the effect of the allocation algorithm on

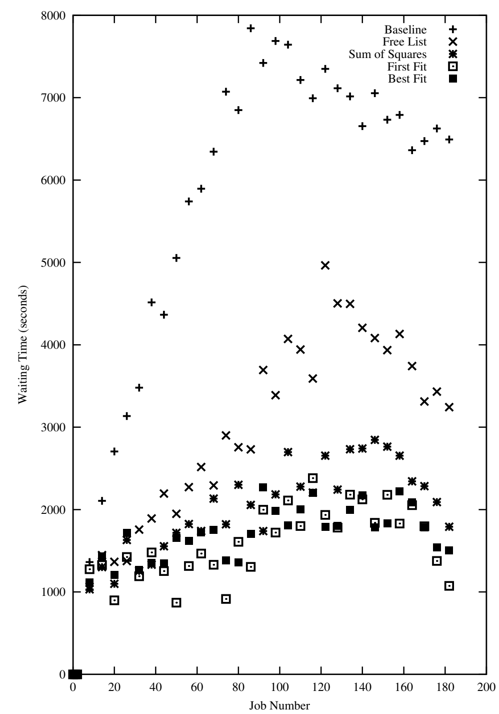


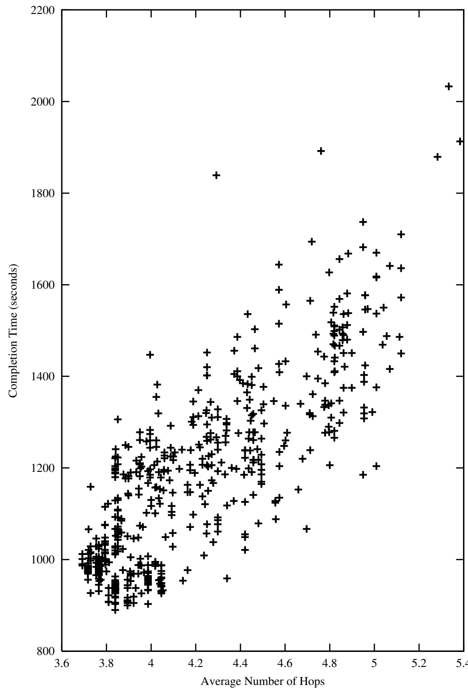
Figure 2. The x-axis shows order of job release. The y-axis shows waiting time. The baseline points use the default processor ordering. All other runs are for the indicated algorithm with the Hilbert curve. Jobs of size 2, 5, and 25 are not represented since these would all be near the line $y = 0$.

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 largely indistinguishable.

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 identical 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.

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



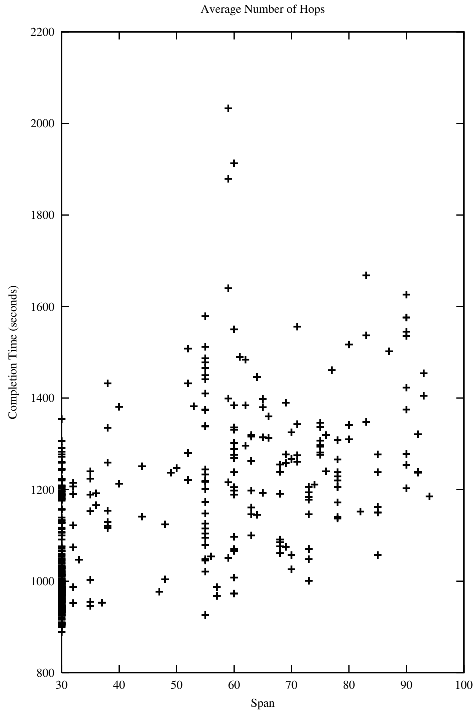


Figure 3. Top: (a) Completion time as a function of the average number of communication hops between processors. Bottom: (b) Completion time as a function of span. Comparison of fragmentation metrics. These plots include only 30-processor jobs across all allocation algorithms. (a) includes all processor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

A natural geometric fragmentation metric is the average of the number of communication hops between processors allocated to a 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 include the 2-node, 5-node, and 25-node jobs in these plots. The 2-node, 5-node, and 25-node jobs differ enough from the 30-node jobs to add noise to the plots. When the 2-node, 5-node, 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, albeit more weakly. We do not show the similar plot for bounding box perimeter that gives an intermediate strength correlation. None 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 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 [49]. 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.

We intend to evaluate non-greedy allocation methods for jobs that cannot be allocated a contiguous interval. In particular, Sum-of-Squares-like algorithms are more likely to leave flexibility in the allocation options for future jobs. On some Cplant machines, once a job has span of half the machine size, it effectively consumes bandwidth across the entire machine. Our hope is that additional flexibility will allow us to avoid such situations.

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.

Our experiments were limited by the small size of our test machine and the specialized nature of the test jobs/stream. 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.

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. D. Bunde is supported in part by Sandia National Laboratories and NSF grant CCR-0093348. 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 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] Cray Inc. Network queuing environment. `http://www.cray.com/products/software/nqe.html`.

[16] 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.

[17] 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.

[18] 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.

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

[20] 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.

[21] 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.

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

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

[24] 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 (SC)*, 1997.

[25] 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, (DD)*, number 218 in Contemporary Mathematics, pages 279–286, Providence, 1998. AMS.

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

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

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

[29] 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.

[30] 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.

[31] 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.

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

[33] 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.

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

[35] 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.

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

[37] 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.

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

[39] 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.

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

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

[42] 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.

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

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

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

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

[47] 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.

[48] 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.

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

[50] 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.

[51] 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.