SAND2014-2469C

Local search to improve task mapping

Evan Balzuweit*, David P. Bunde*, Vitus J. Leung†, Austin Finley*, Alan C.S. Lee*
*Department of Computer Science
Knox College, Galesburg, IL, USA
{ebalzuwe,dbunde,afinley,cslee}@knox.edu
†Sandia National Laboratories
Albuquerque, NM, USA
vjleung@sandia.gov

***Abstract*—We present a local search strategy to improve the mapping of a parallel job's tasks to the MPI ranks of its parallel allocation in order to reduce network congestion and the job's communication time. The goal is to reduce the number of network hops between communicating pairs of ranks. Our target is applications with a nearest-neighbor stencil communication pattern running on mesh systems with non-contiguous processor allocation, such as Cray XE and XK Systems. Using the miniGhost mini-app, which models the shock physics application CTH, we demonstrate that our strategy reduces application running time while also reducing the runtime variability.**

***Keywords*-Task mapping, stencil communication pattern, non-contiguous allocation, local search.**

I. Introduction

Task mapping is the assignment of a job's tasks to the set of computational elements allocated to that job. When using MPI programming, it is the decision of which MPI rank performs which part of the computation. Improved task mapping has been shown to significantly reduce job running times for a variety of scientific applications (e.g. [1], [2], [3], [4], [5]), including achieving a speedup of 1.64 on a quantum system simulation [3]. Better task mapping reduces the number of hops between communicating tasks and hence the amount of system bandwidth consumed by each message. As systems grow larger and processor performance continues to improve faster than network performance, the importance of task mapping will continue to grow.

The job's communication pattern and the system's network topology are both important for task mapping. Here, we focus on jobs with a nearest-neighbor stencil communication pattern, a very common pattern in computational science applications. For this pattern, the tasks correspond to integer points in a grid and communicate with their nearest neighbors, the 4 closest points in the $+x$, $-x$, $+y$, and $-y$ directions for 2D or the 6 closest points for 3D (add the $+z$ and $-z$ directions). This pattern arises naturally from spatial decompositions into hyperrectangular regions.

In this paper, we target machines whose network topology is a 3D mesh and allow the possibility that a job is allocated to a non-contiguous set of nodes. This is appropriate for the Cray XT, XE, and XK series of systems, including the Cray XE6 (Cielo) we used for our experiments. An alternate allocation model is used by systems such as the Blue Gene, which always assigns each job a contiguous set of nodes isolated from each other [6]. This isolation provides benefits [7], but also decreases system utilization (e.g. [8], [9]). There are task mapping algorithms designed specifically for systems with contiguous allocation (e.g. [10], [11]). Note that our algorithms can be applied to the contiguous setting as well, but that the reverse is not true.

Although jobs are allocated nodes on our target machines, our task mapping algorithms actually work in terms of MPI *ranks* rather than compute nodes allocated to that job. Each MPI rank is a single process in a distributed memory program; we will refer to them simply as ranks. In general, each compute node may support many ranks depending on the number of cores it has and the mix of distributed- and shared-memory programming models (e.g. MPI and OpenMP) used.

A recent task mapping algorithm developed for our setting is Geometric (Geom) [12], which operates by finding corresponding decompositions of the job tasks and allocated ranks. This algorithm was shown to outperform a wide variety of other algorithms, reducing application running time by around 30% [12], [13].

Contribution: The main contributions of this paper are as follows:

- We present a local search algorithm that improves on Geom by swapping pairs of tasks when doing so improves the average distance between communicating tasks.
- We demonstrate our algorithm in a proxy application and show that it slightly improves the application's total running time. Furthermore, it does this while reducing the variability in total running time.
- We examine the number of swaps made by our algorithm, showing it is reasonable in practice. We also show that some processor allocations require more, but use the distribution of swaps made to provide guidance on when to cut off the search and avoid pathological cases.

At a high level, our results again demonstrate that GEOM is a good task mapping algorithm, but show that local search can be cheap enough to improve upon it.

Outline: The rest of this paper is organized as follows. Section II describes our algorithms. Section III describes the setup for our experiments and simulations. Section IV describes our results. Section V summarizes related work. Section VI concludes and discusses future work.

II. GEOM AND GSEARCH

We now describe our task mapping algorithms. GEOM first rotates the job so that its dimension lengths have the same order as the bounding box of the ranks (i.e. if the bounding box of ranks is largest in the x dimension, then the job's largest dimension will also be x and so on).

After any necessary rotations, GEOM develops corresponding decompositions of the set of tasks in the job and the set of ranks allocated to it. Each of these are represented with a list of coordinates, the coordinates of a task being its position in the job's communication pattern and the coordinates of a rank being its position in the machine's 3D grid. In addition, the job is represented with a triple giving its size in each dimension; the job is always a hyperrectangle. At a given step, the job's tasks are split into two hyperrectangles as evenly as possible along its longest dimension. For example, a $3 \times 4 \times 5$ job would be initially split into two hyperrectangles of size $3 \times 4 \times 3$ and $3 \times 4 \times 2$ respectively. The same dimension is then used to split the ranks into two subsets with corresponding sizes. This would put the 36 ranks with smaller z coordinates into the first subset and the 24 ranks with larger z coordinates into the second subset (ties broken consistently). The corresponding subsets of tasks and ranks are then mapped to each other recursively. The base case is subsets of size one, which are handled by mapping the only task to the only rank.

Hoefler and Snir [14] describe a recursive bisection algorithm for general job communication patterns. They represent the communication pattern with a graph and use the METIS graph partitioner [15] to split the graph. Often a more general algorithm is preferable, but in this case GEOM has two important advantages over the more general algorithm. First, it is a simpler and faster algorithm. Second and more importantly, it gives better mappings; Deveci et al. [13] showed that it resulted in job communication times roughly 15% shorter than the more general algorithm. We attribute this advantage to the benefit of using task coordinates at every level of the recursion. Consider Figure 1, which shows two levels of a possible GEOM decomposition. Because task coordinates are used, tasks with low x coordinates are near each other (in regions A and C) even when they are separated by the decomposition's first cut. This would not necessarily be the case when a general graph partitioner is used since reversing the roles of C and D or using different dimensions to split the tasks and ranks could give equally good cuts. Using the coordinates allows GEOM to easily exploit the full structure of the problem.

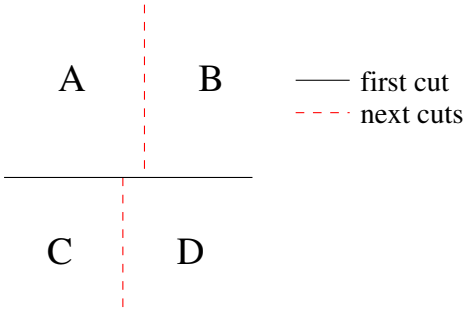

Figure 1. Two levels of cuts in decomposition created by GEOM

Our other algorithm, GEOM-BASED LOCAL SEARCH (GSEARCH), uses the observation from [12] that job running time correlates with the average number of hops between communicating tasks (*average hops metric*). GSEARCH aims to improve the mapping quality by performing a local search to improve the average hops metric. Specifically, it begins with the mapping generated by GEOM and examines pairs of tasks, swapping the ranks executing them whenever doing so reduces the average hops metric. The search continues until a local minimum is found, meaning that no pairs can be swapped to improve the metric. (Some slight variations of GSEARCH are discussed in Section IV-C.)

III. EXPERIMENTAL SETUP

We evaluate these algorithms using experiments on a large system and on a trace-based simulator. The simulations were made comparable to the experiments when possible, with a similar machine size, interconnect topology, and processor allocation algorithm. This was done to correlate the simulations to the experiments so that simulations could be used when experiments were impractical.

The simulations did not model the effect of mapping on running time so the algorithms are evaluated based on their effect on average hops, but in exchange they allowed us to examine our algorithms on many more jobs, each of them taken from a real trace and allocated with a realistic processor allocation algorithm.

A. Cielo

Our experiments were run on the ACES [16] system Cielo [17], located at Los Alamos National Laboratories. Cielo is a Cray XE6 with 8,944 compute nodes plus a smaller number of service nodes connected in a Cray Gemini 3D torus in a sixteen by twelve by twenty-four (XYZ) topology, with two nodes (sockets) per Gemini. Each compute node is a dual AMD Opteron 6136 eight-core "Magny-Cours" socket G34 running at 2.4 GHz. Each service node is a 272 AMD Opteron 2427 six-core "Istanbul" socket F running at 2.2 GHz. The bi-section bandwidth is 6.57 by

Nodes	Job Dimensions
4	$1 \times 4 \times 1$
8	$2 \times 4 \times 1$
16	$2 \times 4 \times 2$
32	$2 \times 8 \times 2$
64	$4 \times 8 \times 2$
128	$4 \times 8 \times 4$
256	$4 \times 16 \times 4$
512	$8 \times 16 \times 4$
1K	$8 \times 16 \times 8$
2K	$8 \times 32 \times 8$
4K	$16 \times 32 \times 8$

Figure 2. Job Dimensions used in miniGhost experiments

4.38 by 4.38 (XYZ) TB/s. As of November 2013, Cielo was number 26 on the Top500 list [18].

The application used in the experiments was miniGhost. As part of the exascale research program, the DOE lab community is developing mini applications (miniApps) that are representative of the computational core of major advanced simulation and computing codes. MiniGhost is a miniApp for exploring boundary exchange strategies using stencil computations in scientific parallel computing. The miniGhost application [19] is a bulk-synchronous message passing code whose structure is modeled on the computational core of CTH [20]. CTH is a multi-material, large deformation, strong shock wave, solid mechanics code developed at Sandia National Laboratories.

A set of experiments consists of miniGhost runs for various numbers of nodes (powers of 2). In each experiment, the two mapping algorithms are run one after the other with the same allocation to minimize the experimental variances other than the task mapping algorithm used. All experiments in a set were submitted to the system queue at roughly the same time. We ran a total of six sets.

The communication pattern for each job was a 3D nearest neighbor stencil with dimensions as shown in Figure 2. (These dimensions are the same as those used for this application in prior work on task mapping [12], [21]; the aspect ratio comes from a shaped charge problem for CTH.) Internode communication was performed with MPI. Each MPI rank ran on all 16 cores in a node, with intranode parallelism managed with OpenMP. The miniGhost output includes total time, communication time as a percentage of total time, and average hops between neighboring ranks in the application. The application spends about thirty percent of its time communicating.

B. Simulator

For our simulation, we ran a trace-based simulator with the LLNL-Atlas trace (version 1.1 clean) from the Parallel Workloads Archive [22]. The trace is from a cluster with 9,216 nodes. To match this node count with a mesh system, our simulation assumed it was a $24 \times 24 \times 16$ mesh. From this trace, we took job start times, execution times, and number of nodes needed. (Note that most Parallel Workloads Archive traces give job arrival times and thus require scheduling; since the LLNL-Atlas trace gives start times instead, every job runs exactly when it did on the real system.)

To identify the nodes allocated for each job, we used a linear allocation algorithm called snake best fit that combines ideas of Lo et al. [8] and Leung et al. [23]. This algorithm organizes the nodes in a linear order along a "snake" or "s-curve", which curves back and forth along the machine's shortest dimension (z). The free nodes are grouped into intervals by their position along the curve and the algorithm allocates nodes from the smallest interval with enough nodes (best fit). If no interval is large enough, then nodes are selected to minimize the *span*, the maximum distance along the curve between selected nodes. This scheme is fast and generates good allocations [24]. It is also similar to the algorithm used in practice on Cray systems.

IV. Results

Now we present our experimental results in three parts. First, we demonstrates that GSearch yields better task mappings than Geom. Second, we look at the number of swaps that GSearch makes, a potential concern for its running time. Third, we look at a couple of slight variations on GSearch that were designed to reduce the number of swaps needed.

A. Mapping quality

The main criteria for task mapping quality is application running time. Figure 3 shows the average running times for miniGhost runs of different sizes using Geom and GSearch. The two algorithms are essentially tied at small job sizes, with GSearch gradually becoming better as the job size increases to 2K nodes and then essentially tying again at 4K nodes. GSearch gives a better average time for all these sizes, with the largest difference being 0.83 seconds at 2K.

The relatively close performance of the two algorithms at 4K nodes breaks the apparent performance trends up to that point. In particular, it appears anomalous since the running time of Geom breaks its clear upward path prior to that job size. One possible cause of this variation is random noise due to contention from other jobs, which is known to significantly affect running times [7]. An examination of individual runs does reveal significant variation, but not in a way that supports this explanation, however. In only one of the six runs at 4K nodes did Geom take longer than 24.25 seconds, its average time at 2K nodes. Even the best 4K run for GSearch relative to Geom only beat it by 0.52 seconds, less than the average amount by which GSearch beat Geom for the 2K node size. The apparent anomaly of

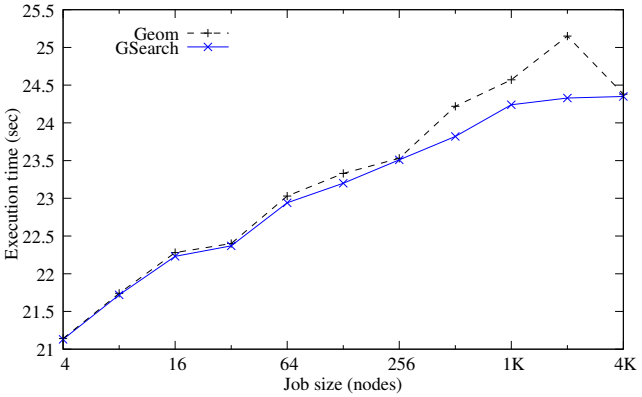

Figure 3. Running time by job size for miniGhost on Cielo (Average over 6 sets of experiments)

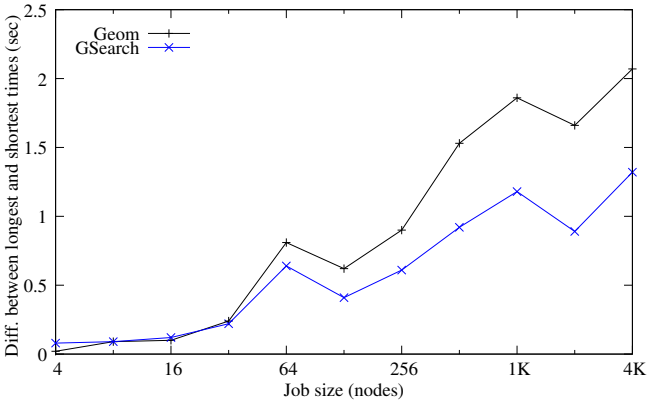

Figure 4. Difference between max and min running time by job size for miniGhost on Cielo

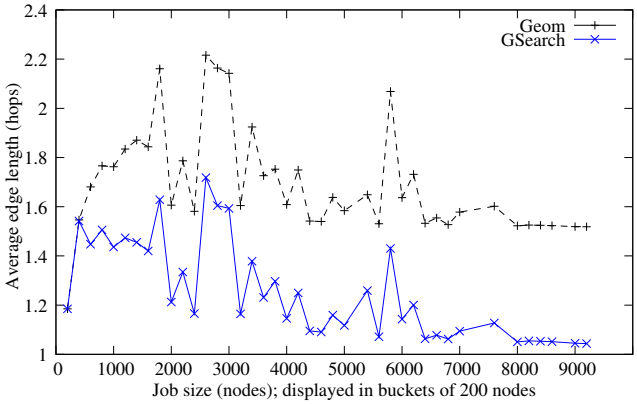

Figure 5. Average edge length by job size for LLNL-Atlas trace

GEOM's performance at 4K is an interesting open issue and we return to it in Section VI.

A second criteria for task mapping quality is predictability of performance. Figure 4 shows the difference between the longest and shortest running times for each algorithm on each job size. We see that the range of running times increases with job size for both algorithms, but that GSEARCH generally delivers only about two thirds of the runtime variability of GEOM. Thus, the performance of GSEARCH seems to be both slightly better and more predictable.

Our simulation results also support the idea that GSEARCH improves upon GEOM task mappings except for the smallest jobs. Figure 5 shows the average distance metric as a function of job size for both task mapping algorithms; recall that the simulations do not model the effect of mapping on running time but that average hops has been shown to correlate with it. Since GSEARCH explicitly optimizes this metric, it always achieves average hops that are at least as low as GEOM. Just as in the experiments, there is neglible difference between the algorithms for small jobs; with few nodes, there is little opportunity to improve on a reasonable mapping. With larger jobs, GSEARCH gives consistently better average hops. The curves are roughly parallel, indicating that variations are caused by allocation quality, which would affect both task mapping algorithms. We believe that the specific peaks and valleys are artifacts of the allocations; these are not smoothed out because some of the job sizes correspond to only a few jobs. Nonetheless, the general behavior is clear, with GSEARCH consistently reducing the average hops metric.

This reduction in average distance may contribute to the reduction in runtime variability that GSEARCH provides relative to GEOM. Much "random" variability in job running times is caused by congestion on network links, which can delay messages [7]. Reducing the number of hops between a pair of communicating tasks makes that pair's messages less susceptible to congestion. It also makes that pair's messages less likely to interfere with other messages in the system, including messages between other tasks in the same job.

B. Number of swaps needed

Having discussed mapping quality, we now examine the number of swaps made by GSEARCH. This was one of our initial concerns about GSEARCH; if the number of swaps is too large, the extra time spent improving the task mapping would eliminate the benefit of doing so. In order to study this issue, we looked at the numbers of swaps used in our experiments and simulations, as well as performing additional simulations specifically to get a sense of the distribution of swap frequencies. This work assuaged our concerns and suggested bounds that could be used to limit the number of swaps and avoid extreme worst-case behavior.

Figure 6 shows the average and maximum number of swaps GSEARCH performed on jobs of each size during our experiments on Cielo. Both the average and maximum values seem to grow linearly with the job size. The difference between them shrinks in proportion of the average value, though its absolute magnitude generally increases with job size (not visible from the graph because of the log-log scale).

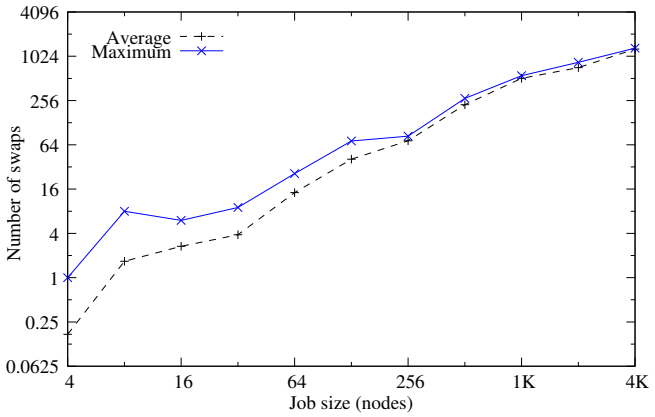

Figure 6. Number of swaps made by GSEARCH as a function of job size (average and max over 6 sets of experiments on Cielo)

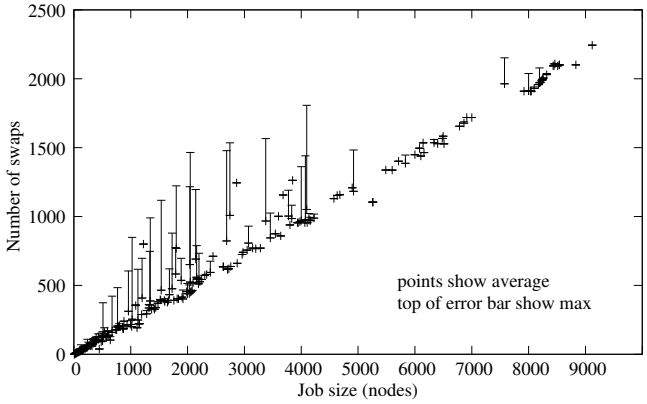

Figure 7. Number of swaps made by GSEARCH as a function of job size on LLNL-Atlas trace

The exception is that the values are relatively close at 4K, possibly because this size uses so much of the machine. Jobs occupying a large fraction of the machine must have large contiguous components and GEOM does well in this case; it gives a perfect mapping if the allocation and job communication pattern have matching shapes.

Figure 7 shows the number of swaps as a function of job size for the trace-based simulation. The points show the average value and the top of the error bar shows the maximum value. This figure shows considerably more noise than Figure 6 because many sizes are represented by only a few jobs, but the trends are very similar. Again, the average number of swaps seems to increase linearly with the job size; here we can see the slope is around one fourth. The upper envelope of the maximum numbers of swaps also seems to be growing linearly up through jobs of size 4K. We attribute this to the size affect mentioned above and also to the fact that nearly all of these large sizes were represented by relatively few jobs.

To further demonstrate that the number of swaps is not overwhelming, we ran some additional simulations: for a

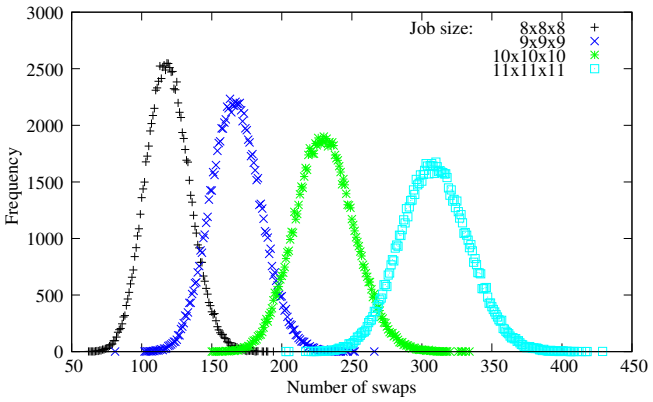

Figure 8. Swap count frequencies from 100,000 random allocations on $16 \times 24 \times 24$ system

variety of job sizes, we counted the number of swaps made by GSEARCH on a randomly-chosen allocation of nodes. Each job size was tested with 100,000 allocations. Figure 8 plots the number of allocations that resulted in each number of swaps. All of sizes give bell-shaped frequency curves, indicating that the number of swaps is fairly concentrated about the mean value. As the job size increases, the curves move to the right and do flatten somewhat. Even for the largest job ($11 \times 11 \times 11$), however, the maximum number of swaps (429) is only 120=39% above the mean for its curve (309) and 99% of the allocations use at most 368 swaps, only 19% above the mean.

Random sampling like this may miss the outermost tails of a probability distribution, but these results are useful for determining a threshold at which to stop seeking swaps. The mean number of swaps made seems to consistently be around $n/4$ for a job of size n. To allow all the values shown in Figure 8, we increase this by 40% to $1.4 \times n/4 = 0.35n$. Increasing this by a small additive term so that small jobs are given some flexibility gives a threshold $0.35n + 10$. The exact constants can be tuned, but using this threshold would allow the search to continue in all except the most extreme cases.

To check the behavior of GSEARCH if its search is terminated due to hitting a threshold on the number of swaps, we had our simulator print the average hops metric after each 100 swaps. Figure 9 shows the results for some jobs of size 4K. There are 286 jobs of this size in the trace so the figure only depicts a few of them selected to show the range of behavior. Depicted are the five jobs which received the worst initial mapping from GEOM, required the largest number of swaps, improved the most in the local search, improved the least, and received the best initial mapping from GEOM (this job also had the best final mapping). There were other jobs in the trace with similar values for all these characteristics (initial/final average hops, number of swaps, and amount of improvement). Some jobs appeared exactly the same; at least

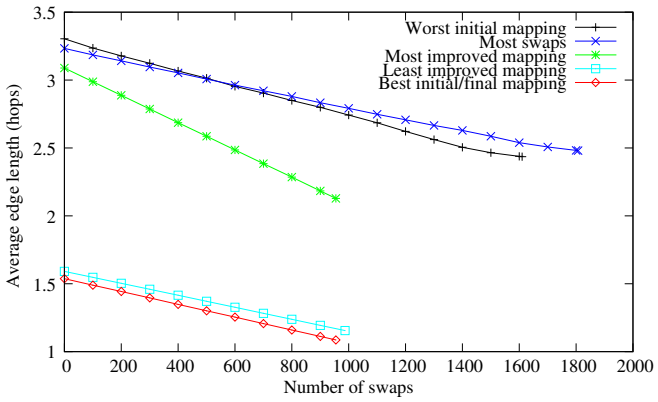

Figure 9. Average edge length as a function of the number of swaps made on sample jobs from the LLNL-Atlas trace. All selected jobs used 4K nodes

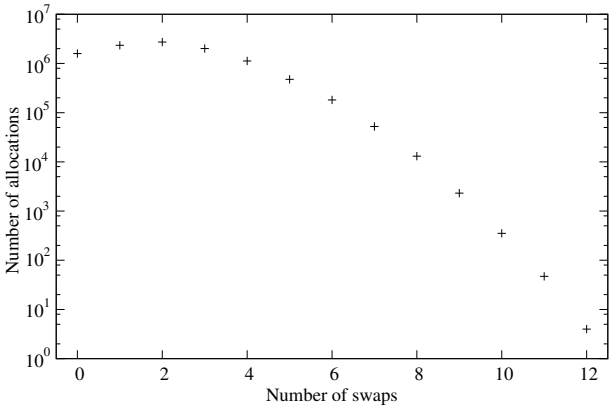

Figure 10. Swap count frequencies from all possible allocations of a $4 \times 2 \times 1$ job on a $4 \times 4 \times 2$ system

in some cases this occurred because the jobs ran one after another and thus received identical allocations. There were also many jobs that had intermediate values for all of the listed characteristics. The amount of improvement for most jobs was similar in each 100 swaps (appearing as a straight line in the figure), but there was some variation; the rate of improvement noticeably trails off for the job with the worst initial mapping. Different jobs also improved at significantly different rates (appearing as different slopes) in a way that does not appear to be related to the quality of the initial mapping; the lines for many jobs cross even though this happens only once among the jobs selected for depiction in Figure 9. (Recall that those jobs were selected because they are boundary cases.)

In addition to these simulations of the effect of a threshold, we ran a couple of experiments with swap thresholds on Cielo. Each consisted of a series of miniGhost runs with a different threshold. Every run in an experiment had the same allocation and they were executed consecutively, but random noise causes the performance to jump much more than the smooth progression shown in Figure 9 suggests. Despite this, using a threshold does not appear to harm the algorithm, which is all we really need since the goal of using a threshold is to prevent pathological behavior.

In order to investigate the true worst case number of swaps, we also exhaustively tried GSEARCH on all allocations for some small cases. We found that a $4 \times 2 \times 1$ job can require as many as 12 swaps (1.5 times the job size) on a $4 \times 4 \times 2$ machine. Using this many swaps is vanishingly rare, however, with only 4 allocations of $\binom{32}{8} = 10,518,300$ requiring this many; see Figure 10 for the full distribution.

We also observed that it is important to start with a good initial task mapping. We were able to hand construct bad initial mappings that gave longer sequences of swaps than any we saw with GSEARCH. In particular, we found improving sequences of swaps of length 14 for a 2×3 job, 23 for a 3×3 job, 31 for a 3×4 job. These give ratios of number of swaps to job size of $2.\overline{3}$, $2.\overline{5}$, and $2.58\overline{5}$ respectively, much higher than even the 1.5 that we saw above when running GSEARCH on all allocations; clearly the quality of the initial mapping matters.

It seems challenging to give an absolute bound on the number of swaps that are possible. One idea is to bound the number of times a given pair of tasks can swap. Surprisingly, we observed several cases in which a pair of tasks can swap more than once, with the second swap putting the tasks back onto their original ranks. This can happen when intervening swaps move their neighbors in a way that makes the original swap detrimental.

C. *Variations on* GSEARCH

Thus far, this paper has discussed a particular local search strategy which makes any swap that improves the average distance as soon as it is identified. Early in this research, we used simulations to compare this algorithm (GSEARCH) with a couple of alternate ways to choose swaps. In particular, we looked at the following alternatives:

1) Consider swaps rank by rank, testing all swaps involving a given rank before making the best one (i.e. the one which improves the average distance by the most).
2) Test swaps of all pairs before making the single best one.

The goal of each of these alternatives was to reduce the number of swaps by avoiding long sequences of swaps that provide little benefit. We found that the alternate algorithms accomplish this, reducing the number of swaps performed by 12% and 35% respectively while giving final mappings that are of essentially indistinguishable quality (average distance within 0.2%). Unfortunately, the additional time required to find each of the swaps eliminated the savings of doing fewer of them; the first alternative took 2% longer than GSEARCH and the second took 2.5 times as long. The simple strategy used by GSEARCH seems to be the best of these ideas.

V. Related work

There is a wide variety of prior research on task mapping in different settings, which we now summarize. Before GEOM, the main algorithms for task mapping with non-contiguous allocations were based on linear orderings. MiniGhost's default behavior is to assign tasks to ranks in row-major order; this is typical for applications that do not specifically consider task mapping. With ALPS and Moab, MPI ranks are determined by the allocation order, which comes from a node numbering along a space-filling curve [25] using an algorithm similar to the one described in Section III-B. The default mapping then assigns rows along this curve, which can create long edges in the task columns and at the end of each row. Barrett et al. [21] observed that this mapping did not scale well above 4K cores (256 nodes) on Cielo and improved it by renumbering the tasks so that a submeshes of the job are assigned to each node.

GEOM was proposed by Leung et al. [12], who originally called it RCB. They showed that it beat the strategies above and adaptations of task mapping algorithms from the contiguous setting proposed by Bhatelé et al. [11]. Deveci et al. [13] created MULTI-JAGGED (MJ), a slight generalization of GEOM that uses multi-way partitions instead of bisections to decompose the sets of tasks and ranks. They also considered shifts to account for the wraparounds of a torus interconnect and trying all rotations rather than just the one that gives the job and ranks the same dimension ordering. The shifts and extra rotations gave a slight benefit to miniGhost and more for miniMD [26], a miniapp based on a molecular dynamics application whose communication pattern can propagate information further in a single step. For both applications, GEOM and MJ outperformed the graph partitioning-based mapping algorithm by Hoefler and Snir [14] described in Section II.

Other heuristic strategies have also been applied to task mapping with non-contiguous allocations, including greedy [27], genetic algorithms [28], simulated annealing [29], and partitioning by spectral methods [30].

VI. Discussion

We have shown that the local search strategy GSEARCH can slightly improve the running time of stencil-based applications while meaningfully reducing the variation between runs. On one hand, our results further demonstrate the quality of the GEOM algorithm, which seems to obtain solutions near a local optimum. On the other hand, they also show that local search is cheap enough to be beneficial.

The main outstanding question from our work is to explain the apparent anomaly of GEOM improving between the 2K and 4K sizes. A similar phenomenon was reported in a slightly different setting in prior work by Leung et al. [12]. Specifically, they reported a slight decline in GEOM running time for miniGhost going from 2K nodes to 4K nodes when each MPI rank ran on 4 cores rather than 16 as in our experiments. One likely explanation is that this improvement comes from the job using such a large part of the machine; this necessarily means that many of the allocated processors are contiguous, which simplifies task mapping, and also lessens the interference from other jobs since there are fewer of them. This explanation is not entirely consistent with Figure 5, however, since that figure shows GSEARCH improving the average hops metric for all job sizes. Possible reasons for this discrepancy include the following:

- Architectural differences; the simulated system has a longer y-dimension instead of 2 nodes at each mesh location as Cielo does.
- Differences in job shapes.
- Imperfections in metric; average hops is not the same as job running time, despite the correlation between them and other evidence of the metric's usefulness as a proxy for running time (like the benefits from GSEARCH).

We plan to investigate these issues and also consider other possible causes of GEOM's performance improvement at the largest size.

Beyond this open question, a logical next step in this work is to develop a fully parallel implemenation of GSEARCH; our current version computes the mapping serially even though all nodes in the allocation are available. The search space of possible swaps is large and most swaps are independent so parallelism should help reduce the search time, but it is not obvious how to decompose the search space or handle swaps that will affect multiple parts of it. One idea is to build on MJ [13], which already investigates different shifts and rotations in parallel.

Acknowledgments

E. Balzuweit, D.P. Bunde, A. Finley, and A. Lee were partially supported by contract 899808 from Sandia National Laboratories. Sandia National Laboratories is a multiprogram laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. We also thank Moe Jette for contributing the LLNL-Atlas trace to the Parallel Workloads Archive.

References

[1] R.F. Barrett, C.T. Vaughan, S.D. Hammond, and D. Roweth. Reducing the bulk in the bulk synchronous parallel mode. *Parallel processing letters*, to appear.

[2] A. Bhatele and L.V. Kale. Benefits of topology-aware mapping for mesh topologies. *Parallel Processing Letters*, 18(4):549–566, 2008.

[3] F. Gygi, Erik W. Draeger, M. Schulz, B.R. de Supinski, J.A. Gunnels, V. Austel, J.C. Sexton, F. Franchetti, S. Kral, C.W. Ueberhuber, and J. Lorenz. Large-scale electronic structure calculations of high-Z metals on the BlueGene/L platform. In *Proc. ACM/IEEE Conf. High Performance Networking and Computing (SC)*, 2006.

[4] H. Subramoni, S. Potluri, K. Kandalla, B. Barth, J. Vienne, J. Keasler, K. Tomko, K. Schulz, A. Moody, and D. K. Panda. Design of a scalable infiniband topology service to enable network-topology-aware placement of processes. In *Proc. Conf. High Performance Computing, Networking, Storage and Analysis (SC)*, 2012.

[5] S. von Alfthan, I. Honkonen, and M. Palmroth. Topology aware process mapping. In *Applied Parallel and Scientific Computing*, volume 7782 of *LNCS*, pages 297–308. Springer, 2013.

[6] Y. Aridor, T. Domany, O. Goldshmidt, J.E. Moreira, and E. Shmueli. Resource allocation and utilization in the Blue Gene/L supercomputer. *IBM J. Research and Development*, 49(2/3):425, 2005.

[7] A. Bhatele, K. Mohror, S.H. Langer, and K.E. Isaacs. There goes the neighborhood: Performance degradation due to nearby jobs. In *Proc. Conf. High Performance Computing, Networking, Storage and Analysis (SC)*, page 41, 2013.

[8] V. Lo, K. Windisch, W. Liu, and B. Nitzberg. Non-contiguous processor allocation algorithms for mesh-connected multicomputers. *IEEE Trans. Parallel and Distributed Systems*, 8(7):712–726, 1997.

[9] V. Subramani, R. Kettimuthu, S. Srinivasan, J. Johnson, and P. Sadayappan. Selective buddy allocation for scheduling parallel jobs on clusters. In *Proc. 4th IEEE Intern. Conf. on Cluster Computing*, pages 107–116, 2002.

[10] H. Yu, I-H. Chung, and J. Moreira. Topology mapping for Blue Gene/L supercomputer. In *Proc. ACM/IEEE Conf. High Performance Networking and Computing (SC)*, 2006.

[11] A. Bhatelé, G.R. Gupta, L.V. Kalé, and I.-H. Chung. Automated mapping of regular communication graphs on mesh interconnects. In *Proc. Intern. Conf. High Performance Computing (HiPC)*, 2010.

[12] V.J. Leung, D.P. Bunde, J. Ebbers, S.P. Feer, N.W. Price, Z.D. Rhodes, and M. Swank. Task mapping stencil computations for non-contiguous allocations. In *Proc. 19th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming (PPoPP)*, pages 377–378, 2014.

[13] M. Deveci, S. Rajamanickam, V.J. Leung, K.T. Pedretti, S.L. Olivier, D.P. Bunde, Ü.V. Çatalyürek, and K.D. Devine. Exploiting geometric partitioning in task mapping for parallel computers. In *Proc. 28th IEEE Intern. Parallel and Distributed Processing Symp. (IPDPS)*, 2014.

[14] T. Hoefler and M. Snir. Generic topology mapping strategies for large-scale parallel architectures. In *Proc. 25rd ACM Intern. Conf. Supercomputing (ICS)*, 2011.

[15] K. Schloegel, G. Karypis, and V. Kumar. Parallel static and dynamic multi-constraint graph partitioning. *Concurrency and Computation: Practice and Experience*, 14(3):219–240, 2002.

[16] J. Ang, D. Doerfler, S. Dosanjh, S. Hemmert, K. Koch, J. Morrison, and M. Vigil. The Alliance for Computing at the Extreme Scale. In *Proc. 52nd Cray User Group*, 2010.

[17] Los Alamos National Laboratory. High-performance computing: Cielo supercomputer. http://www.lanl.gov/orgs/hps/cielo/index.html, viewed September 2013.

[18] Top 500 Supercomputer Sites. http://www.top500.org/.

[19] R.F. Barrett, C.T. Vaughan, and M.A. Heroux. MiniGhost: A miniApp for exploring boundary exchange strategies using stencil computations in scientific parallel computing. Technical Report SAND2011-5294832, Sandia National Laboratories, 2011.

[20] E.S. Hertel, R.L. Bell, M.G. Elrick, A.V. Farnsworth, G.I. Kerley, J.M. McGlaun, S.V. Petney, S.A. Silling, P.A. Taylor, and L. Yarrington. CTH: A software family for multi-dimensional shock physics analysis. In *Proc. 19th International Symposium on Shock Waves*, 1993.

[21] R. Barrett, S. Hammond, C. Vaughan, D. Doerfler, J. Luitjens, and D. Roweth. Navigating an evolutionary fast path to exascale. In *Proc. 3rd Intern. Workshop Performance Modeling, Benchmarking and Simulation of High Performance Computing Systems (PMBS)*, 2012.

[22] D. Feitelson. The parallel workloads archive. http://www.cs.huji.ac.il/labs/parallel/workload/index.html.

[23] 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 Intern. Conf. on Cluster Computing*, pages 296–304, 2002.

[24] P. Walker, D.P. Bunde, and V. Leung. Faster high-quality processor allocation. In *Proc. 11th LCI Intern. Conf. High-Performance Clustered Computing*, 2010.

[25] C. Albing, N. Troullier, S. Whalen, R. Olson, J. Glenski, and H. Mills. Topology, bandwidth and performance: A new approach in linear orderings for application placement in a 3D torus. In *Proc. Cray User Group*, 2011.

[26] M.A. Heroux, D.W. Doerfler, P.S. Crozier, J.M. Willenbring, H.C. Edwards, A. Williams, M. Rajan, E.R. Keiter, H.K. Thornquist, and R. W. Numrich. Improving performance via mini-applications. Technical Report SAND2009-5574, Sandia National Laboratories, 2009.

[27] Tarun Agarwal, Amit Sharma, and L.V. Kalé. Topology-aware task mapping for reducing communication contention on large parallel machines. In *Proc. 20th IEEE Intern. Parallel and Distributed Processing Symp. (IPDPS)*, 2006.

[28] T. Chockalingam and S. Arunkumar. Genetic algorithm based heuristics for the mapping problem. *Computers and Operations Research*, 22(1):55–64, 1995.

[29] S.W. Bollinger and S.F. Midkiff. Heuristic technique for processor and link assignment in multicomputers. *IEEE Trans. Computers*, 40(3), 1991.

[30] I-Hsin Chung, Che-Rung Lee, Jiazheng Zhou, and Yeh-Ching Chung. Hierarchical mapping for HPC applications. In *Proc. Workshop on Large-Scale Parallel Processing*, pages 1810–1818, 2011.