Power-aware MPI Task Aggregation Prediction for High-End Computing Systems

Dong Li[†] Dimitrios S. Nikolopoulos[‡] Kirk Cameron[†] Bronis R. de Supinski[⋆] Martin Schulz[⋆]

[†] Virginia Tech
Blacksburg, VA, USA
{lid,cameron}@cs.vt.edu

[⋆] Lawrence Livermore National Lab
Livermore, CA, USA
{bronis,schulzm}@llnl.gov

[‡] FORTH-ICS and University of Crete
Heraklion, Crete, GREECE
dsn@ics.forth.gr

***Abstract*—Emerging large-scale systems have many nodes with several processors per node and multiple cores per processor. These systems require effective task distribution between cores, processors and nodes to achieve high levels of performance and utilization. Current scheduling strategies distribute tasks between cores according to a count of available cores, but ignore the execution time and energy implications of task aggregation (i.e., grouping multiple tasks within the same node or the same multicore processor). Task aggregation can save significant energy while sustaining or even improving performance. However, choosing an effective task aggregation becomes more difficult as the core count and the options available for task placement increase. We present a framework to predict the performance effect of task aggregation in both computation and communication phases and its impact in terms of execution time and energy of MPI programs. Our results for the NPB 3.2 MPI benchmark suite show that our framework provides accurate predictions leading to substantial energy saving through aggregation (64.87% on average and up to 70.03%) with tolerable performance loss (under 5%).**

***Keywords*-MPI; performance modeling; power-aware high-performance computing.**

I. Introduction

Modern high-end computing systems have many nodes with several processors per node and multiple cores per processor. The distribution of tasks across the cores of multiple nodes impacts both execution time and energy. Current job management systems, which typically rely on a count of available cores for assigning jobs to cores, simply treat parallel job submissions as a 2D chart with time along one axis and number of cores along the other [1], [2]. They regard each job as a rectangle with width equal to the number of cores requested by the job and height equal to the estimated job execution time. Most scheduling strategies are based on this model, which has been extensively studied [3], [4], [5]. Some job scheduling systems also consider communication locality factors such as the network topology [6]. Unfortunately, job schedulers ignore the power-performance implications of the layouts of cores available in compute nodes to execute tasks from parallel jobs.

Task aggregation refers to aggregating multiple tasks within a node with shared memory. A fixed number of tasks can be distributed across a variable number of nodes, using different degrees of task aggregation per node. Aggregated tasks share system resources, such as the memory hierarchy and network interface, which has an impact on performance. This impact may be destructive, because of contention for resources. However, it may also be constructive. For example, an application can benefit from the low latency and high bandwidth of intra-node communication through shared memory. Although earlier work has studied the performance implications of communication through shared-memory in MPI programs [7], [8], [9], the problem of selecting the best distribution and aggregation of a fixed number of tasks has been left largely to ad hoc solutions.

Task aggregation significantly impacts energy consumption. A job uses fewer nodes with a higher degree of task aggregation. Unused nodes can be set to a deep low-power state while idling. At the same time, aggregating more tasks per node implies that more cores will be active running tasks on the node, while memory occupancy and link traffic will also increase. Therefore, aggregation tends to increase the power consumption of active nodes. In summary, task aggregation has complex implications on both performance and energy. Job schedulers should consider these implications in order to optimize energy-related metrics while meeting performance constraints.

In this paper we propose a model to predict the impact of task aggregation, which entails several issues:

- How can we model and predict the performance impact of aggregation on computation phases? The prediction should capture the likely impact of increased hardware utilization and contention on scalability.
- How can we model and predict the impact of aggregation on communication phases? Can we predict this impact based on the aggregation pattern?
- How can we integrate predictions for computation and communication to predict the optimal aggregation level, given an optimization criterion based on performance, energy, or a combination thereof?

The main contributions of this paper are:

- A framework to collect information from execution samples in HPC applications and use this information to predict the impact of MPI task aggregation;
- A model for predicting the impact of aggregation on the computation phases of MPI programs;
- An analysis of the effects of concurrent inter-task communication under varying aggregation levels and a method of predicting an upper bound of communication

978-1-4244-6443-2/10/$26.00 ©2010 IEEE

time across different aggregation levels;
- A formalization of the problem of deciding which MPI tasks should be aggregated, given the aforementioned analysis of communication; this formalization maps the problem into a graph partitioning problem, which we solve with a heuristic algorithm;
- An evaluation of task aggregation on system scales of up to 1024 cores.

Our results show that our prediction captures accurately the performance impact of different aggregation patterns. Our prediction for the computation phases has 1.08% error on average. Our prediction for total execution time time, which includes an upper bound of communication time, has 28.97% error on average. This seemingly large error arises due to the inability of modeling overlapping communication operations and communication with computation in our framework. Nevertheless, the error tends to be uniform across aggregation patterns, therefore our prediction tends to rank correctly aggregations in terms of the objective metric (minimizing energy under a performance constraint). The predicted task aggregations yield substantial energy saving (64.87% on average and up to 70.03%) with tolerable performance loss (under 5%). We correctly predict the optimal aggregation in most cases on system scales from 16 nodes up to 128 nodes. Even in mispredicted cases, our prediction is close to the optimal and achieves 68.12% energy saving on average. In addition, our scaling study (up to 1024 cores) demonstrates improved performance with more aggregation.

The rest of this paper is organized as follows. Section II formulates the problem of task aggregation. Section III presents our method for predicting the performance of computation phases after task aggregation. Section IV presents our graph partitioning algorithm for task grouping. Section V presents our communication performance prediction method. Section VI discusses our method for ranking aggregation patterns. Section VII presents our experimental analysis. Section VIII discusses related work and Section IX concludes the paper.

II. Problem Statement

We target the problem of how to distribute MPI tasks between and within nodes in order to minimize execution time, or minimize energy, under a given performance constraint. The solution must make two decisions: how many tasks to aggregate per node; and how to assign the tasks scheduled on the same node to cores, which determines how these tasks will share hardware components such as caches, network resources, and memory bandwidth. In all cases, we select a task aggregation pattern based on performance predictions.

We assume the following:

1) The number of MPI tasks is given and fixed throughout the execution of the application;
2) The number of nodes used to execute the application and the number of tasks per node is decided at job submission time and this decision depends on a prediction of the impact of different aggregation patterns on performance and energy;
3) Any aggregation must assign the same number of tasks to each node;
4) Jobs are SPMD (Single Program Multiple Data) programs;
5) MPI communication patterns—including message size and communication target—can vary across tasks;
6) Aggregation patterns must not result in total DRAM requirements that exceed a node's physical memory.

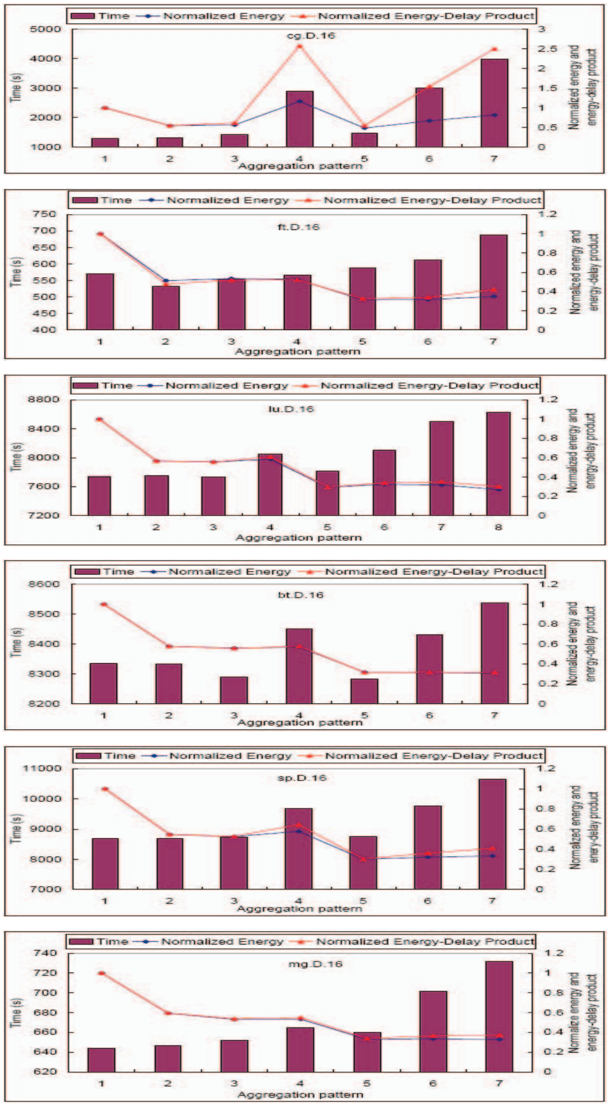


Figure 1: Impact of task aggregation on the NAS PB suite

Allowing aggregation patterns that place more tasks on some nodes than others may be more efficient in imbalanced applications, however, the resulting load imbalance would hurt performance and waste energy in well balanced SPMD applications. In these cases, the system could leverage slack

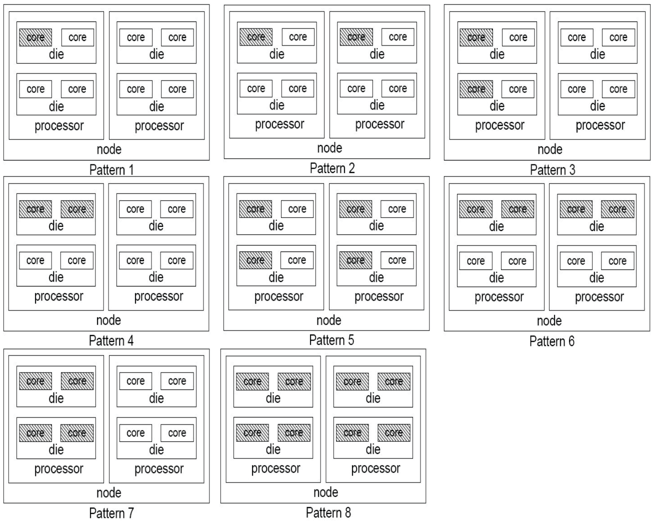


Figure 2: Aggregation patterns on our test platform

to save energy. Energy saving opportunities due to slack have been studied elsewhere [10] and are beyond the scope of this work.

A series of tests with the NPB 3.2 MPI benchmarks [11] (problem size D) on 16 nodes of a cluster under various aggregation patterns demonstrates the effect of aggregation on execution time and energy. Each node of this cluster has two Xeon E5462 quad-core processors, each of which has two dies shared by two cores and 8 GB of physical memory. Figure 1 shows the results for the eight possible aggregation patterns on this platform. Energy and energy-delay product data are normalized to the results with aggregation pattern 1. Figure 2 depicts the aggregation patterns with the cores assigned to tasks indicated with stripes. Pattern 8 is not available for some benchmarks as each task requires more than an eighth of the memory on a node. Our results for FT use more grid points and iterations than in the standard problem size D to achieve longer execution time. As seen in the execution time and energy consumption of the benchmarks across all feasible aggregation patterns, no single pattern always provides the best results. For example, using one core of each die (pattern 5) for CG uses the least energy but results in sub-optimal execution time, while using one core of each processor (pattern 2) for FT provides the lowest execution time but does not minimize energy consumption.

We decompose the aggregation problem into three subproblems:

1) Predicting the impact of task count per node on computation;
2) Predicting the communication cost of all aggregation patterns;
3) Combining the computation and communication predictions.

We study the impact of aggregation on computation and communication separately, since the same aggregation pattern can impact computation and communication differently.

We present a prediction-based framework to solve the aggregation problem. The framework exploits the iterative structure of parallel phases that dominate execution time in scientific applications. We collect samples of hardware event counters and communication pattern information at runtime. From this data, we predict the performance under all feasible aggregation patterns and then rank the patterns to identify a preferred aggregation pattern.

III. Predicting Computation Performance

We predict performance during computation phases by predicting IPC. We derive an empirical model based on previous work [12], [13], [14], which predicts IPC of computation phases in OpenMP applications. We use iteration samples collected at runtime on specific aggregation patterns to predict the IPC for each task on other untested aggregation patterns. The IPC for each aggregation pattern is the average value of the IPC of all tasks.

The execution samples provide statistical indicators about the execution properties of computation phases that impact IPC. Different aggregation patterns imply different patterns in resource sharing and contention, which in turn influence IPC. If we can accurately predict the impact of sharing and contention on IPC we can also identify the aggregations that improve energy efficiency.

Based on this discussion, we build an empirical model derivation shown in equation (1):

$$IPC_t = \sum_{i=1}^{|S|}(IPC_i \cdot \alpha_{(t,i)}(e_{(1\cdots n,i)})) + \lambda_t(e_{(1\cdots n,S)}) + \sigma_t \quad (1)$$

The model predicts the IPC of a specific aggregation pattern t, based on information collected in S samples. We collect n hardware event rates $e_{(1\cdots n,i)}$ and IPC_i in each sample i. The function $\alpha_{(t,i)}()$ scales the observed IPC_i in sample i up or down based on the observed values of event rates while λ_t is a function that accounts for the interaction between events and σ_t is a constant term for the aggregation pattern t. For a specific sample s, α_t is defined as:

$$\alpha_t(e_{(1\cdots n,s)}) = \sum_{j=1}^{n}(x_{(t,j)} \cdot e_{(j,s)} + y_{(t,j)}) + z_t \quad (2)$$

where $e_{(j,s)}$ is a hardware event in sample s, and $x_{(t,j)}$, $y_{(t,j)}$ and z_t are coefficients.

λ_t is defined as:

$$\lambda_t(e_{(1\cdots n,S)}) = \sum_{i=1}^{n}(\sum_{j=1}^{|S|-1}(\sum_{k=j+1}^{|S|}(\mu_{(t,i,j,k)} \cdot e_{(i,j)} \cdot e_{(i,k)})))+ \sum_{j=1}^{|S|-1}(\sum_{k=j+1}^{|S|}(\mu_{(t,j,k,IPC)} \cdot IPC_j \cdot IPC_k)) + l_t \quad (3)$$

where $e_{(i,j)}$ is the i_{th} event of the j_{th} sample. $\mu_{(t,i,j,k)}$, $\mu_{(t,j,k,IPC)}$ and l_t are coefficients.

We approximate the coefficients in our model with multivariate linear regression. IPC, the product of IPC and each event rate, and the interaction terms in the sample aggregation patterns serve as independent variables, while the IPC on each target aggregation pattern serves as the dependent variable. We record IPC and a predefined collection of event rates while executing the computation phases of each training benchmark with all aggregation patterns. We use the hardware event rates that most strongly correlate with the target IPC in the sample aggregation patterns. We develop a model separately for each aggregation pattern and derive sets of coefficients independently. The training benchmarks are the twelve SPEC MPI 2007 benchmarks [15] under different problem sets, which demonstrate wide variation in execution properties such as scalability and memory-boundedness.

We classify computation phases into four categories based on their observed IPC during the execution of the sample aggregation patterns and use separate models for different categories in order to improve prediction accuracy. Specifically, we classify phases into four categories with IPC $[0, 1), [1, 1.5), [1.5, 2.0)$ and $[2.0, +\infty)$. Thus our model is a piecewise linear regression that attempts to describe more accurately the relationship between dependent and independent variables by separately handling phases with low and high scalability characteristics.

We test our model by comparing our predicted IPC with the measured IPC of the computation phases of several NPB MPI benchmarks. We present results from tests on the Virginia Tech System G supercomputer (see Section VII) in Figure 1. Patterns 4 and 5 from Figure 2 serve as our sample aggregation patterns. Our model is highly accurate, as the results in Table I show, with worst-case absolute error of 2.109%. The average error in all predictions is 1.079% and the standard deviation is 0.7916.

IV. Task Grouping

An aggregation pattern determines how many tasks to place on each node and processor; we must also determine which tasks to collocate. If an aggregation groups k tasks per node and a program uses n tasks, there are $\binom{n}{k}$ ways to group the tasks to achieve the aggregation. For nodes with $p \geq k$ processors, we then can place the k tasks on one node in $\binom{p}{k}k! = \frac{p!}{(p-k)!}$ ways on the available cores. The grouping of tasks on nodes and their placement on processors has an impact on the performance of MPI point-to-point communication. Computation phases are only sensitive to how tasks are laid out in each node and not to which subset of tasks is aggregated in each node since we assume SPMD applications with balanced workloads between processors. The impact of task placement for MPI collective operations depends on the characteristics of the network; they are relatively insensitive to task placement with flat networks such as fat trees. Thus, we focus on point-to-point operations as we decide which specific MPI ranks to locate on the same node or processor.

	Measured IPC	Predicted IPC	Error rate
lu.D.16, pattern1	1.926	1.944	0.9510%
lu.D.16, pattern2	1.921	1.937	0.8234%
lu.D.16, pattern3	1.924	1.942	0.9109%
lu.D.16, pattern6	1.846	1.834	0.6689%
lu.D.16, pattern7	1.763	1.743	1.137%
lu.D.16, pattern8	1.699	1.669	1.741%
bt.D.16, pattern1	2.033	2.028	0.2475%
bt.D.16, pattern2	2.048	2.032	0.7474%
bt.D.16, pattern3	2.051	2.041	0.4738%
bt.D.16, pattern6	2.014	2.008	0.3041%
bt.D.16, pattern7	1.989	1.995	0.3033%
ft.D.16, pattern1	1.441	1.469	1.934%
ft.D.16, pattern2	1.568	1.601	2.109%
ft.D.16, pattern3	1.512	1.543	2.032%
ft.D.16, pattern6	1.451	1.479	1.948%
ft.D.16, pattern7	1.291	1.314	1.730%
sp.D.16, pattern1	1.952	1.976	1.259%
sp.D.16, pattern2	1.956	1.971	0.7839%
sp.D.16, pattern3	1.99	1.955	0.3166%
sp.D.16, pattern6	1.755	1.789	1.951%
sp.D.16, pattern7	1.610	1.628	1.110%

Table I: IPC prediction of computation phases

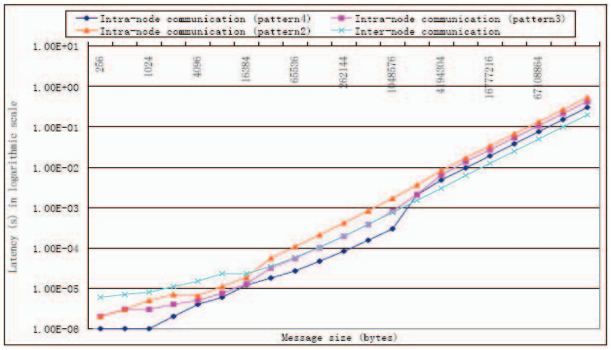


Figure 3: Intra-node vs. inter-node latency comparison

We demonstrate how MPI point-to-point communication is sensitive to locations of communication source and target in Figure 3, which shows results of single ping-pong pattern tests with the Intel MPI benchmark [16] using OpenMPI-1.3.2 [17]. Each test has two tasks involved in an MPI point-to-point communication. For the inter-node results, we used two nodes connected with a 40Gb/s InfiniBand network. Each node has two Intel Xeon E5462 quad-core processors. We also test the three possible intra-node placements (same die; different dies, same processor; different processors — see Figure 2).

The results reveal that intra-node communication has low latency for small messages, while inter-node high bandwidth communication is more efficient for large messages. These conclusions are consistent with previous results on the Myrinet2000 network [7]. In theory, intra-node communication can leverage the low latency and high throughput of the node memory system. Therefore, it should outperform inter-node communication. In practice, sharing the node's memory bandwidth between communicating tasks while they exchange large messages incurs sufficient overhead to make it less efficient than inter-node communication. We also find that the performance of intra-node communication is sensitive to how the tasks are laid out within a node: intra-node communication can benefit from cache sharing due to processor die sharing or whole processor sharing.

Based on these results, we prefer aggregations that colocate tasks based on whether their communication is in the latency or bandwidth regime. However, we cannot decide whether to colocate a given pair of tasks based only on individual point-to-point communications between them. Instead, we must consider all communication performed between those tasks and all communication between all tasks. Overall performance may be best even though some (or all) point-to-point communication between two specific tasks is not optimized.

Task grouping is an NP-complete problem [18]. We formalize the problem as a graph partitioning problem and use an efficient heuristic algorithm [19] to solve it. We briefly review this algorithm in the following section.

A. Algorithm Review

The algorithm partitions a graph G of kn nodes with associated edge costs into k subpartitions, such that the total cost of the edge cut, the edges connecting subpartitions, is minimized. The algorithm starts with an arbitrary partitioning into k sets of size n and then tries to bring the partitioning as close as possible to being pairwise optimal by repeated application of a 2-way partitioning procedure.

The 2-way partitioning procedure starts with an arbitrary partitioning $\{A, B\}$ of a graph G and tries to decrease the initial external cost T (i.e., the total cost of the edge cut) by a series of interchanges of subsets of A and B. The algorithm stops when it cannot find further pair-wise improvements. To choose the subsets of A and B, the algorithm first selects two graph nodes a_1, b_1 such that the gain g_1 after interchanging a_1 with b_1 is maximum. The algorithm temporarily sets aside a_1 and b_1 and chooses the pair a_2, b_2 from $A - \{a_1\}$ and $B - \{b_1\}$ that maximizes the gain g_2. The algorithm continues until it has exhausted the graph nodes. Then, the algorithm chooses m to maximize the partial sum $\sum_{i=1}^{m} g_i$. The corresponding nodes $a_1, a_2, ..., a_m$ and $b_1, b_2, ..., b_m$ are exchanged.

This algorithm has a reasonable probability of finding the optimal partition. The number of subset exchanges before the algorithm converges to a final partition for a 2-way partitioning is between 2 and 4 for a graph with 360 edges [19]. In our experiments we execute programs with at most 128 tasks (as shown in Section VII). In this case, the subset exchanges for 2-way partitioning (2 cluster nodes) is at most 2 and the number of total subset exchanges for 16-way partitioning is at most 1200.

B. Applying the Algorithm

Task aggregation must group T tasks into n partitions, where n is the number of nodes we want to use for a specific aggregation pattern. We regard each MPI task as a graph node and communication between tasks as edges. Aggregating tasks into the same node is equivalent to placing graph nodes into the same partition. We now define an edge cost based on the communication between task pairs.

The original algorithm tries to minimize the total cost of the edge cut. In other words, it tries to place graph nodes with a small edge cost into different partitions. Thus, we must assign a small (large) cost value on the edge which favors inter-node (intra-node) communication. We observe two further edge cost requirements:

1) The difference between the small cost value (for inter-node communication) and the large cost value (for intra-node communication) should be large;
2) The edge values should form a range that reflects the relative benefit of intra-node communication.

A large difference between edge costs reduces the probability of the heuristic algorithm selecting a poor partitioning. The range of values reflects that colocation benefits some task pairs that communicate frequently more than others.

To assign edge costs, we measure the size of every message between each task pair during execution to obtain a communication table. We then estimate the communication time for each pair of communicating tasks i and j if we place them in the same partition (t_{ij}^{intra}) and if we place them in different partitions (t_{ij}^{inter}). We estimate these communication times experimentally by using data similar to that shown in Figure 3. Our intra-node communication time prediction is conservative, since we use the worst-case intra-node communication (i.e., two tasks with no processor or die sharing). Finally, we compare t_{ij}^{intra} and t_{ij}^{inter} to decide whether the tasks i, j benefit from colocation. If $t_{ij}^{intra} > t_{ij}^{inter}$ then we set the edge cost to $c_{ij} = 1.0/(t_{ij}^{intra} - t_{ij}^{inter})$. Alternatively, if $t_{ij}^{intra} \leq t_{ij}^{inter}$, we set $c_{ij} = C + (t_{ij}^{inter} - t_{ij}^{intra})$. These edge costs provide a range of values that reflect the relative benefit of intra-node communication as needed.

C is a parameter that ensures the difference of edge costs for pairs of tasks that favor intra-node communication and pairs of tasks that favor inter-node communication is large. We define C as:

$$C = k^2 \Delta t \tag{4}$$

where k is the number of tasks per node (i.e., $k = T/n$). k^2 is the maximum number of edge cuts between the two partitions. Δt is defined as $\max\{1.0/(t_{ij}^{intra} - t_{ij}^{inter})\}$ between all task pairs (i, j) that benefit from inter-node communication.

Overall, our edge costs reflect whether the communication between a task pair is in the latency or bandwidth regime. We apply the graph partitioning algorithm based on these edge costs to group tasks into n nodes. We then use the same algorithm to determine the placement of tasks on processors within a node. Thus, this algorithm calculates a task placement for each aggregation pattern.

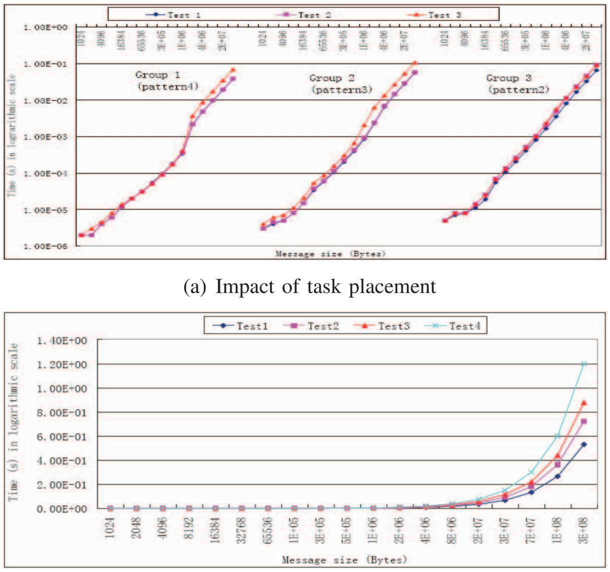
V. Predicting Communication Performance

Communication performance prediction must estimate the impact of sharing resources such as memory, buses or other node-level interconnects, network interfaces, links, and switches. We use the term *communication interference* if this sharing causes performance loss.

We study how communication operations performed by tasks in the same node affect performance. In particular, we investigate if the placement of tasks in the node (i.e., how MPI tasks are distributed between processors, sockets, and dies in the node) and task intensity (i.e., the number of tasks assigned to the node) affect performance. Figure 4 displays the performance of intra-node MPI point-to-point communication between two tasks, which we call the "observed communication operation", while there is interference from other concurrent intra-node communication operations, which we call "noise". We use the Intel MPI benchmark to perform concurrent ping-pong tests within a node, and present results from the same system as in Section IV. The observed communication operation and noise start at the same time and use the same message size.

Figure 4(a) shows that task placement impacts communication performance. In the following, we refer to the numbering of patterns presented in Figure 2 for conciseness. The intra-node communication in groups 1, 2, and 3 follows communication patterns 4, 3 and 2 respectively. Each group has three tests: test 1 is a reference with no noise; test 2 and test 3 each have a task pair introducing noise by performing intra-node communication. In test 2, the layout of the task pair introducing noise and the layout of the observed task pair follow pattern 6 for group 1 and pattern 5 for group 2 and group 3. In test 3, the layout of the task pair introducing noise and the layout of the observed task pair follow pattern 7 in group 1 and 2, and pattern 6 in group 3. The performance penalty of intra-node communication under noise can range from negligible to as high as 182%, depending on where the tasks that introduce noise are located.

Figure 4(b) shows that task intensity has a significant impact on communication performance. In these tests, the two tasks performing the observed communication operation do not share a processor. The tasks introducing noise do not share a processor either. Test 1 is again a reference with no noise, while test 2 has one pair of tasks introducing noise and test 3 has two pairs of tasks introducing noise. The intra-node communication of tasks introducing noise in test 2 and test 3 occupy a different processor than the task pair performing the observed communication. Test 4 has three task pairs introducing noise by performing intra-node communication that, together with the observed pair, fully occupy all cores. The performance of intra-node communication is significantly affected by other intra-node communication operations running concurrently on the same node, especially if the message size is large.



(a) Impact of task placement

(b) Impact of task intensity

Figure 4: Impact of communication interference

We conducted exhaustive tests to cover all combinations of intra-node communication and inter-node communication under different aggregation patterns. The tests show that both intra-node communication and inter-node communication are sensitive to interference from concurrent communication operations. They also show that the performance of MPI communication is sensitive to task placement and task intensity. Thus, we must capture the impact of these factors and integrate them into our prediction framework.

Modeling and predicting communication time in the presence of noise is challenging, due to the following reasons:

- Computation/communication overlap;
- Overlap and interference of concurrent communication operations;
- Even in the absence of overlap, many factors, including task placement, task intensity, communication type (i.e., intra-node or inter-node), and communication volume and intensity impact communication interference.

Thus, we propose an empirical method to predict a reasonable upper bound for MPI point-to-point communication time.

We trace MPI point-to-point operations to gather the endpoints of communication operations. We also estimate

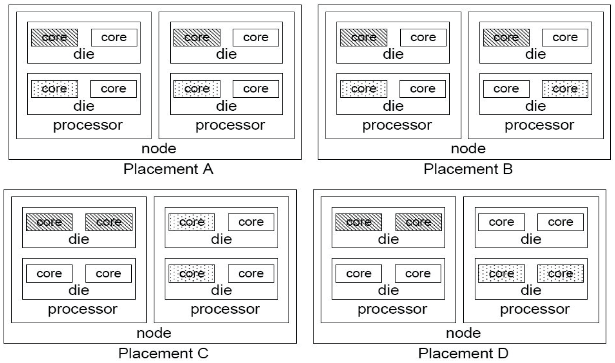


Figure 5: Examples of symmetric task placements

potential interference based on the proximity of the calls. We use this information to estimate parameters for task placement and task intensity that interfere with each communication operation for each aggregation pattern. Since we predict an upper bound, we assume that the entire MPI latency overlaps with noise from other concurrent communication operations. This assumption is reasonable for well-balanced SPMD applications, because of their bulk-synchronous execution pattern.

We construct a prediction table based on our extracted parameters, namely type of communication (intra-node/inter-node), task intensity, task placement for both communicating and interfering tasks, and message size. We populate the table by running MPI point-to-point communication tests under various combinations of input parameters. We reduce the space that we must consider for the table by considering groups of task placements with small performance difference as symmetric. The symmetric task placements have identical hardware sharing characteristics with respect to observed communication and noise communication. Figure 5 depicts two symmetric examples with one observed intra-node task pair and one task pair introducing noise. We mark the cores occupied by task pairs that introduce noise with dots and the cores occupied by the observed task pairs with stripes. Placement A and placement B are symmetric, so are placement C and placement D.

We use a similar empirical scheme for MPI collectives. However, the problem is simplified since collectives on MPI_COMM_WORLD involves all tasks; we leave extending our framework to handle collective operations on derived communicators as future work. Thus, we only need to test the possible task placements for specific task counts per node for the observed communication.

We apply our communication prediction methodology to the NPB 3.2 MPI benchmarks and compare it with the communication time measured and reported by mpiP [20]. Figure 6 shows a subset of the results. We use 10 iterations of the main computation loop in bt.D.16 and ft.D.16, and 50 iterations of the main computation loop in mg.D.16, to minimize measurement error. Most MPI operations in BT and MG are point-to-point operations. We clearly overpredict communication overhead due to the overlap of computation and communication and our pessimistic prediction of the overlap between interfering communication operations. On the contrary, MPI operations in FT are collective operations and our prediction methodology is accurate in these cases.

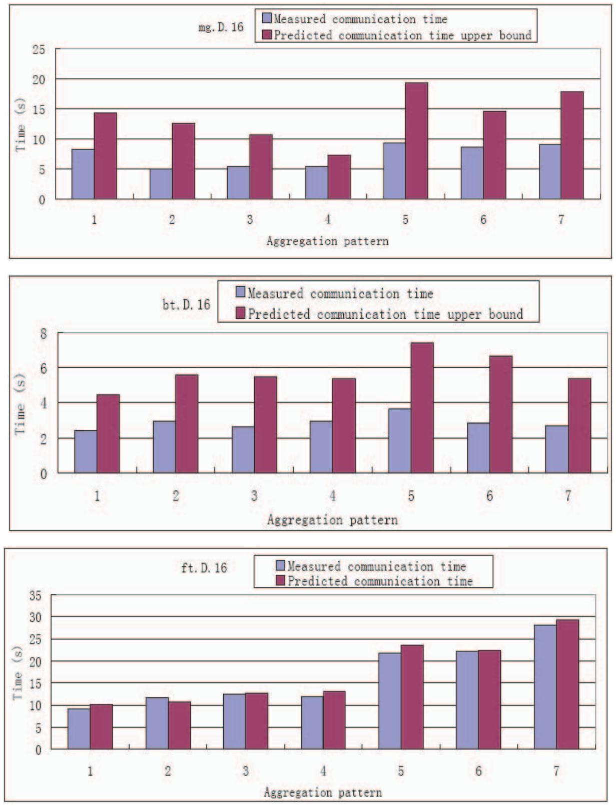


Figure 6: Measured vs. predicted communication time

VI. Choosing an Aggregation Pattern

Our prediction framework allows us to predict the aggregation pattern that either optimizes performance, or optimizes energy under a given performance constraint.

We predict the best aggregation pattern based on our computation and communication performance predictions. Since our goal is to minimize energy under a performance constraint, we pick candidates based on their predicted performance and then rank them considering a ranking of their energy consumption.

We predict performance in terms of IPC (Section III). To predict performance in terms of time, we measure the number of instructions executed with one aggregation pattern and assume that this number remains constant across aggregation patterns. We verify this assumption by counting the number of instructions under different aggregation patterns for 10 iterations of all NPB MPI benchmarks on a node of our cluster. The maximum variance in the number of instructions executed between different aggregation patterns is a negligible 8.5E-05%.

We compare aggregation patterns by measuring their difference to a reference pattern, where there is no aggregation of tasks in a node. We compute the difference as:

$$\Delta t = t_1^{comp} + t_1^{comm} - t_0^{comp} - t_0^{comm} \tag{5}$$

where t_1^{comp}, t_1^{comm} is our estimated computation time and communication time upper bound for the given aggregation pattern respectively and t_0^{comp}, t_0^{comm} is the computation and communication time for the reference pattern respectively. Comparing patterns in terms of difference with a reference pattern partially compensates for the effect of overlap and other errors of time prediction, such as the gap between the actual and predicted communication time. Our analysis in Sections III and V estimates performance for each task. For a specific aggregation pattern, Equation (5) uses the average computation time of all tasks and the longest communication time.

We choose candidate patterns for aggregation using a threshold of 5% for the performance penalty that any aggregation pattern may introduce when compared to the reference pattern. We discard any aggregation pattern with a higher performance penalty, which ensures that we select aggregations that minimally impact user experience. An aggregation may actually *improve* performance; obviously, we consider any such aggregations.

We choose the best aggregation candidate by considering energy consumption. Instead of estimating actual energy consumption, we rank aggregation patterns based on how many nodes, processors, sockets, and dies they use. We rank aggregation patterns that use fewer nodes (more tasks per node) higher. Among aggregation patterns that use the same number of nodes, we prefer aggregation patterns that use fewer processors. Finally, among aggregation patterns that use the same number of nodes and processors per node, we rank aggregation patterns that use fewer dies per processor higher. In the event of a tie, we prefer the aggregation pattern with the better predicted performance. According to this ranking method, the energy ranking of the eight aggregation patterns for our platform in Figure 2 from most energy-friendly to least energy-friendly corresponds with their pattern IDs.

VII. Performance

We implemented a tool suite for task aggregation in MPI programs. The suite consists of a PMPI wrapper library that collects communication metadata, an implementation of the graph partitioning algorithm, and a tool to predict computation and communication performance and choose aggregation patterns. To facilitate collection of hardware event rates for computation phases, we instrument applications with calls to a hardware performance monitoring and sampling library.

We evaluate our framework with the NAS 3.2 MPI benchmark suite, using OpenMPI-1.3.2 as the MPI communication library. We present experiments from the System G supercomputer at Virginia Tech. The system has thousands of power and thermal sensors and uses power-scalable components. It is also equipped with intelligent power strips to log power data for each node. System G has 320 nodes powered by Mac Pro computers, each with two Quad-Core Xeon E5462 processors clocked at 2.8GHz. The nodes are connected by 40Gb/s Infiniband.

We set the threshold of performance loss to 5% and use one task per node as the reference aggregation pattern. The choice of the reference aggregation pattern is intuitive, since we aim at demonstrating the potential energy and performance advantages of aggregation and our reference performs no task aggregation. More specifically, energy consumption is intuitively greatest with one task per node since it uses the maximum number of nodes for a given run. Task aggregation attempts to reduce energy consumption through reduction of the node count. Given that each node consumes a few hundred Watts, we will save energy if we can reduce the node count without sacrificing performance. Using one task per node will often improve performance since that choice eliminates destructive interference during computation or communication phases between tasks running on the same node. However, using more than one task per node can improve performance, e.g., if tasks exchange data through a shared cache. Since the overall performance impact of aggregation varies with the application, our choice of the reference aggregation pattern enables exploration of the energy saving potential of various aggregation patterns.

Figure 7 shows that our prediction selects the best observed aggregation pattern, namely the pattern that minimizes energy while not violating the performance constraint, in all cases. We indicate the best observed and predicted task aggregations with stripes. The performance loss threshold is shown with a dotted line. We achieve the maximum energy saving with sp.D.16 (70.03%) and average energy saving of 64.87%. Our prediction of the time difference between aggregation patterns for both computation and communication follows the variance of actual measured time. For FT and BT, we measure performance gains from some aggregation patterns in computation phases and our predictions correctly capture these gains.

The applications exhibit widely varied computation to communication ratios, ranging from communication-intensive (FT) to computation-intensive (LU). The communication time difference across the different aggregation patterns depends on message size and communication frequency. Small messages or less frequent communication result in a smaller communication time difference. For example, 99.98% of the MPI communication operations in lu.D.16 transfer small messages of size close to 4KB. The communication time differences in patterns 2–6 are all less than 10.0%; the communication time differences in patters 7 and 8 (most intensive aggregation patterns) are less than 22.7%.

On the contrary, the FT benchmark runs with an input of size $1024 \times 512 \times 1024$ and has MPI_Alltoall operations, in which each task sends 134MB data to other tasks and receives 134MB data from other tasks; the communication time differences in patterns 2–6 range between 28.96% and 144.1%; the communication time difference in pattern 7 (most intensive aggregation pattern) is as much as 209.7%.

We also observe that CG is very sensitive to the aggregation pattern: different patterns can have significant performance differences due to CG's memory intensity [21]. Colocating tasks saturates the available memory bandwidth, resulting in significant performance penalties. Finally, we observe MG communication can benefit from task aggregation due to the low latency of communicating through shared-memory. In particular, communication time at patterns 2, 3 and 4 reduce by 12.08%, 25.68% and 48.81% respectively.

To investigate how aggregation affects energy consumption on a larger system scale, Figure 8 shows the results for LU (Class D) with more nodes in strong scaling tests. As we scale up the processor count, performance improves with more aggregation. In particular, lu.D.32 has an optimal aggregation pattern of four tasks per node with each task occupying a separate die, while both lu.D.64 and lu.D.128 have optimal aggregation patterns of eight tasks per node. This difference occurs because tasks have smaller workloads and therefore exercise less pressure on shared resources at large processor counts. This result confirms the intuition that maximal aggregation is the preferred way of running parallel jobs on large-scale clusters, for the purpose of economizing on energy and hardware resources, while sustaining performance.

We also note that we predict a different best aggregation than the observed best for lu.D.64 and lu.D.128 although our choices perform similarly. Our predictions choose the second optimal aggregations with energy saving 68.46% (lu.D.128) and 67.77% (lu.D.64), while the optimal aggregations have energy saving 80.70% (lu.D.128) and 80.65% (lu.D.64). In both cases, our prediction of time difference for the observed best is higher than the real time difference. As a result, we eliminate that aggregation pattern as exceeding the maximum acceptable performance loss. In general, prediction error increases as the time scale during which the program runs decreases. Using more samples for prediction is a potential solution to this problem.

VIII. Related Work

Communication-aware task mapping techniques are an active research area. Leng et al. [7] use the High Performance Linpack benchmark to demonstrate how task re-mapping can improve overall performance. However they manually arrange the process mapping based on communication path characteristics and MPI message information, which is not a feasible solution for complex systems.

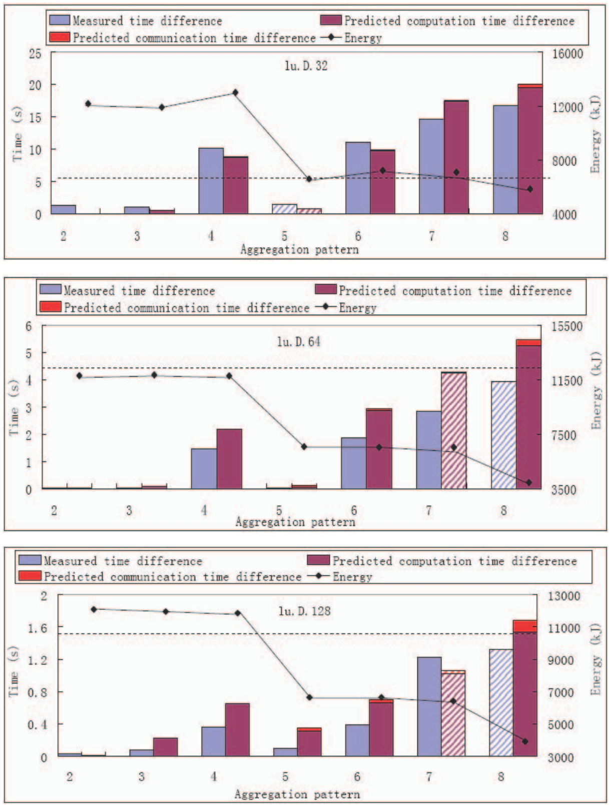


Figure 8: Strong scaling and task aggregation for lu.D

Orduna et al. [18], [22], [23], [24], [25] explore communication-aware task mapping strategies that account for the communication requirements of parallel tasks and bandwidth in different parts of the network. They model the network resources as a table of costs for each pair of communicating processors. Their heuristic random search method attempts to identify the best mapping for a given network topology and communication pattern. Their work targets heterogeneous inter-node networks and does not consider intra-node communication. Also, their evaluation only maps one process to each processor. Thus, they do not account for potential interference between tasks executing communication operations on the same node.

Our work improves on prior research by considering both intra-node and inter-node communication for large scale SMP clusters, which are the most common HPC platforms. Our work further improves on prior research by accounting for the impact of task aggregation on computation phases, which we find to be significant.

Network communication models have been studied ex-

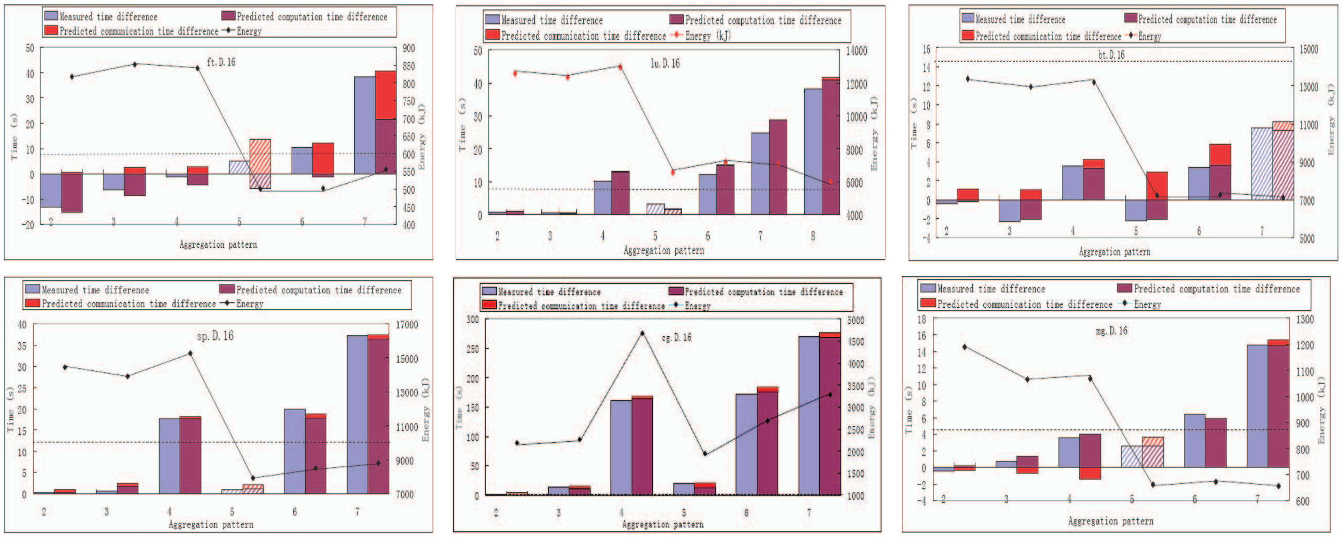


Figure 7: Results for the NPB 3.2 MPI benchmark suite

tensively. Well-known communication models include the LogP [26] and LogGP models [27]. They usually predict communication time with linear equations enclosing measured parameters. These models, however, do not consider concurrent communication operations between groups of tasks and ignore resource sharing, both essential factors in understanding performance under different aggregation scenarios.

Several researchers have focused on communication sharing effects. Kim et.al [28] develop a model for predicting delays of messages that share links in Myrinet. Their study is based on the GM and BIP network protocols, whereas we focus on MPI. Also their study does not consider intra-node communication.

Martinasso et. al [29], [30] introduce a notion of resource sharing within communication patterns. Their work decomposes a chain of sources of communication interference into several elementary sources of interference and then predicts communication time by the flow cut of each source. They evaluate their model on SMP clusters with dual processors. Nodes with more processors and complex core layouts introduce complications in resource sharing that have not been considered in their work. Furthermore, their model requires careful analysis to decompose communication into sources of interference manually, which makes it infeasible for complex communication patterns. Our work considers both intra-node and inter-node communication interference. In addition, we predict worst case performance for the communication conflicts to provide an upper bound on communication time, a simplification that leads to a feasible solution.

Several works have explored dynamic voltage and frequency scaling (DVFS) to save energy. Software-controlled energy saving in MPI programs has attracted considerable interest recently. Rountree et al. [31] use linear programming to estimate the maximum energy saving possible in MPI programs based on critical path analysis. The same authors [32], propose a critical path-based online algorithm that uses simple predictions of execution time for program regions based on prior executions of the same regions. Springer [33] propose a scheduler that selects node counts and CPU frequencies to minimize energy consumption and execution time. DVFS strategies for saving energy complement our approach, which reduces energy by aggregating tasks and releasing system resources.

IX. Conclusions

High-end computing systems continuously scale up to more nodes, with more processors per node and more cores per processor. The multitude of options for mapping parallel programs to these systems creates optimization challenges. In this paper we show that varying the aggregation and placement of MPI tasks can provide significant energy saving and occasional performance gain. Our framework predicts a task aggregation pattern that minimizes energy under performance constraints in MPI applications. We derive an empirical model to predict computation time under all feasible aggregation patterns. We formalize the problem of grouping tasks as a graph partitioning problem and solve it with a heuristic algorithm that considers communication patterns. We also derive a communication time upper bound for concurrent MPI point-to-point communication. Overall, our predictions capture the performance impact of aggregation for both computation and communication phases. Based on our prediction model, we further propose a method to select aggregations. We apply our framework to the NPB 3.2 MPI benchmark suite and observe significant energy saving (64.87% on average and up to 70.03%). We also apply it at scales of up to 128 nodes and observe increasing energy saving opportunities which allow more intensive task aggregation under tight performance constraints.

In future work, we plan on designing an online framework that can dynamically change the task aggregation pattern according to dynamic performance predictions. This approach will require process migration, which is challenging to implement efficiently and to model accurately in MPI programs, yet feasible through the use of native MPI task migration tools or virtualization frameworks. We also plan to apply our framework to parallel programming models other than MPI, including hybrid models.

Acknowledgment

This work has been supported by NSF (CNS-0905187, CNS-0910784, CCF-0848670, CNS-0709025, CNS-0720750) and the European Commission through the MCF IRG project I-Cores (IRG-224759) and the HiPEAC Network of Excellence (IST-004408, IST-217068). Part of this work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344 (LLNL-PROC-422991).

References

[1] D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, "Parallel Job Schueduling - A Status Report," *Lecture Notes in Computer Science*, vol. 3277, pp. 1–16, 2005.

[2] S. Srinivasan, R. Keetimuthu, V. Subramani, and P. Sadayappan, "Characterization of Backfilling Strategies for Parallel Job Scheduling," in *Proceedings of the 2002 International Workshops on Parallel Processing*, 2002.

[3] D. Tsafrir, D. G. Feitelson, and Y. Etsion, "Backfiling Using System-Generated Predictions Rather than User Runtime Estimates," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, pp. 789–803, 2007.

[4] C. B. Lee and A. E. Snavely, "Precise and Realistic Utility Functions for User-Centric Performance Analysis of Schedulers," in *Proceedings of the 16th international symposium on High performance distributed computing*, 2007.

[5] L. Barsanti and A. Sodan, "Adaptive Job Scheduling via Predictive Job Resource Allocation," *Lecture Notes in Computer Science*, vol. 4376, pp. 115–140, 2007.

[6] O. Sonmez, H. Mohamed, and D. Epema, "Communication-aware job placement policies for the koala grid scheduler," in *Proc. of the Second IEEE International Conference on e-Science and Grid Computing (e-Science'06)*, 2006.

[7] T. Leng, R. Ali, J. Hsieh, V. Mashayekhi, and R. Rooholamini, "Performance Impact of Process Mapping on Small-Scale SMP Clusters-A Case Study Using High Performance Linpack," in *Proceedings of IEEE Parallel and Distributed Processing Symposium*, 2002.

[8] L. Chai, Q. Gao, and D. K. Panda, "Understanding the Impact of Multi-Core Architecture in Cluster Computing: A Case Study with Intel Dual-Core System," in *7th IEEE International Symposium on Cluster Computing and the Grid*, 2007.

[9] L. Chai, A. Hartono, and D. K. Panda, "Designing High Performance and Scalable MPI Intra-node Communication Support for Clusters," in *IEEE International Conference on Cluster Computing*, 2006.

[10] N. Kappiah, V. Freeh, and D. Lowenthal, "Just In Time Dynamic Voltage Scaling: Exploiting InterNode Slack to Save Energy in MPI Programs," in *Proceedings of the international conference on Supercomputing*, 2005.

[11] "NAS Parallel Benchmarks." [Online]. Available: http://www.nas.nasa.gov/Resources/Software/npb.html

[12] M. Curtis-Maury, F. Blagojevic, C. D. Antonopoulos, and D. S. Nikolopoulos, "Prediction-Based Power-Performance Adaptation of Multithreaded Scientific Codes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, pp. 1396–1410, 2008.

[13] M. Curtis-Maury, J. Dzierwa, C. D. Antonopoulos, and D. S. Nikolopoulos, "Online Power-Performance Adaptation of Multithreaded Programs using Event-Based Prediction," in *Proc. of the 20th ACM International Conference on Supercomputing (ICS)*, 2006, pp. 157–166.

[14] M. Curtis-Maury, A. Shah, F. Blagojevic, D. S. Nikolopoulos, B. R. de Supinski, and M. Schulz, "Prediction Models for Multi-dimensional Power-Performance Optimization on Many Cores," in *Proc. of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2008.

[15] "SPEC MPI 2007." [Online]. Available: http://www.spec.org/mpi

[16] "Intel MPI Benchmarks 3.2." [Online]. Available: http://software.intel.com/en-us/articles/intel-mpi-benchmarks/

[17] "Open MPI: Open Source High Performance Computing." [Online]. Available: http://www.open-mpi.org/

[18] J. M. Orduna and F. S. J. Duato, "On the Development of A Communication-aware Task Mapping Technique," *Journal of Systems Architecture*, vol. 50, pp. 207–220, 2004.

[19] B.W.Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," *Bell System Technical Journal*, vol. 49, pp. 291–308, 1970.

[20] J. Vetter and C. Chambreau, "mpiP: Lightweight, Scalable MPI Profiling."

[21] Y. Zhang, V. Tipparaju, J. Nieplocha, and S. Hariri, "Parallelization of the NAS Conjugate Gradient Benchmark Using the Global Arrays Shared Memory Programming Model," in *Proceedings of International*

Parallel and Distributed Processing Symposium, 2005.

[22] R. Tornero, J. Orduna, M. Palesi, and J. Duato, "A Communication-aware Topological Mapping Technique for NoCs," *Lecture Notes in Computer Science*, vol. 5168, pp. 910–919, 2008.

[23] J. Orduna, V. Arnau, A. Ruiz, R. Valero, and J. Duato, "On the Design of Communication-aware Task Scheduling Strategies for Heterogeneous Systems," in *Proceedings of International Conference on Parallel Processing (ICPP-2000)*, 2000.

[24] J. Orduna, V. Arnau, and J. Duato, "Characterization of Communication between Processes in Message-passing Applications," in *Proceedings of International Conference on Cluster Computing (Cluster-2000)*, 2000.

[25] J. Orduna, F.Silla, and J. Duato, "A New Task Mapping Technqiues for Communication-aware Scheduling Strategies," in *Proceedings of International Conference on Parallel Processing (ICPP-2001)*, 2001.

[26] D. Culler, R. Karp, D. Patterson, A. Sahay, E. Santos, K. Schauser, R. Subramonian, and T. von Eicken, "LogP: A Practical Model of Parallel Computation," *Communications of ACM*, vol. 39, pp. 78–85, 1996.

[27] A. Alexandrov, M. Ionescu, K. Schauser, and C. Scheiman, "LogGP: Incorporating Long Messages into the LogP model - One step closer towards a realistic model for parallel computation," in *7th Annual Symposium on Parallel Algorithms and Architectures*, 1995.

[28] S. Kim and S. Lee, "Measurement and Prediction of Communication Delays in Myrinet Network," *Journal of Parallel and Distributed Computing*, vol. 61, pp. 1692–1704, 2001.

[29] M. Martinasso and J.-F. Mehaut, "Model of Concurrent MPI Communications over SMP Clusters," in *Technical Report 00071352, HAL-INRIA*, 2006.

[30] J. Vienne, M. Martinasso, J.-M. Vincent, and J.-F. Mehaut, "Predictive Models for Bandwith Sharing in High Performance Clusters," in *Proceedings of the IEEE Cluster Conference*, 2008.

[31] B. Rountree, D. Lowenthal, S. Funk, V. Freeh, B. R. de Supinski, and M. Schulz, "Bounding Energy Consumption in Large-Scale MPI Programs," in *SC '07: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, 2007.

[32] B. Rountree, D. K. Lownenthal, B. R. de Supinski, M. Schulz, V. W. Freeh, and T. Bletsch, "Adagio: Making DVS Practical for Complex HPC Applications," in *Proceedings of the 23rd international conference on Supercomputing*, 2009.

[33] R. Springer, D. Lowenthal, B. Routree, and V. Freeh, "Minimizing Execution Time in MPI Programs on an Energy-Constrained, Power-Scalable Cluster," in *Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2006.