Topology and Routing Aware Mapping on Parallel Processors

Synopsis of thesis to be submitted for the award of

Doctor of Philosophy

in

Computer Science

by

Devi Sudheer Kumar CH

Supervisor:

Prof. Ashok Srinivasan

Department of Computer Science

Florida State Univeristy, USA

DEPARTMENT OF MATHEMATICS & COMPUTER SCIENCE

Sri Sathya Sai Institute of Higher Learning

(Deemed to be University)

Prasanthi Nilayam, 515134, Ananthpur District, A.P., India

Contents

For better readability and easy navigation, use e-version of the synopsis downloadable at: `http://dmacssite.github.com/Synopsis.pdf`

ABSTRACT

Communication costs of a typical parallel application increase with the number of processes. With the increasing sizes of current and future high end parallel machines, communication costs can account for a significant fraction of the total run time even for massively parallel applications that are traditionally considered scalable. Hence, in order for applications to achieve the scalability for utilizing the very large number of processing nodes/cores in the contemporary and future high end parallel machines, techniques must be developed to minimize the communication costs.

One of the important issues that need to be dealt with in the optimization of communication costs on large machines is the impact of topology and routing. Though small message latency is not very dependent on location in the machine, network contention can play a major role in limiting bandwidth, even if the bisection bandwidth is high, due to the limitation of the routing scheme. Knowledge of topology and routing can be used to assign processes to nodes such that contention is reduced. For example, we showed that such assignment can reduce communication cost associated with load balancing of Quantum Monte Carlo (QMC) application by up to 60% on 10,000 nodes (120,000 cores) of the Jaguar machine at ORNL. The algorithms used in the communication library (MPI) must also take topology and/or routing into account in order to maximize performance. Again, we noticed a 30% improvement in performance of MPI_Allgather on 10,000 nodes on Jaguar by renumbering the process ranks, even though the renumbering was not targeted at optimizing MPI performance.

In order to generalize the mapping techniques, we posed the mapping problem with the hop-byte metric as a quadratic assignment problem and used a heuristic to directly optimize for this metric. We evaluated our approach on realistic node allocations obtained on the Kraken system at NICS. Our approach yields values for the metric that are up to 75% lower than the default mapping and 66% lower than existing heuristics.

For better readability and easy navigation, use e-version of the synopsis downloadable at: http://dmacssite.github.com/Synopsis.pdf

For proper scaling of the applications on these machines, the data movement challenge existing across the nodes as well as within a node should be addressed. We evaluated the effect of mapping on the inter-core communication performance in a Cell processor. The inter-core communication is strongly influenced by the assignment of threads to cores (thread-core affinity) for many realistic communication patterns. We identify the bottlenecks to optimal performance and use this information to determine good affinities for standard communication patterns. Our solutions improve performance by up to a factor of two over the default assignment. We also optimize the affinity on a Cell blade consisting of two Cell processors, and provide a software tool to help with this.

For better readability and easy navigation, use e-version of the synopsis downloadable at: `http://dmacssite.github.com/Synopsis.pdf`

1. Introduction

1.1 Overview

High performance computing power is believed to be the key to scientific & engineering leadership, industrial competitiveness, and national security. And it is without doubt a key enabling technology for many technologically advanced nations in the 21st century. Keeping in view of this, the government of India announced that it is examining a proposal to build national capacity and capability in supercomputing at a cost of Rs. 5000 crores, that is $1 billion.

As large scale systems are built with millions (and even billions) of processors, indirect networks such as fat-trees and crossbars quickly become unfeasibly expensive. Indeed, the number of systems using lower degree interconnects such as the Blue Gene and Cray Torus interconnects has increased from 28 systems in the June 2008 Top 500 list to 78 systems in the more recent Top 500 list of June 2012 [1]. The annual rate at which the time for flop is increasing (59%) is far greater than the rate at which network link bandwidth is increasing (26%). Hence, computation is cheap but communication is not and links are oversubscribed. This research is motivated by studies showing that communication costs can already exceed arithmetic costs by orders of magnitude, and the gap is growing exponentially over time.

As massively parallel computers become larger, the interconnect topology will play a more significant role in communication performance. Although, ideally, we would not want to use the topology information since this will result in techniques that are specific to a topology or an architecture. However, at the scale of tens or hundreds of processing cores, the impact of topology and routing is so significant that they must be taken into consideration in order to achieve necessary scalability.

Some supercomputers such as Cray XT and IBM Blue Gene/P machines have 3D torus topologies. In such computers, minimizing network contention by match-

ing communication pattern with the topology is critical for the communication performance [2]. Other current massively parallel computers such as the TACC Ranger use the non-blocking fat-tree topology. Although the fat-tree is non-blocking, the network has contention especially with static routing in InfiniBand networks [3]. Hence, for massively parallel computers, network contention can have a significant impact on performance: to minimize network contention, topology and routing must be taken into consideration.

There are several ways the topology and routing used for optimizing communication performance. In this work, we consider topology and routing aware process assignment. Recent works [15, 16, 17, 18, 20, 21, 22, 23] have shown substantial communication performance improvement on large parallel machines by suitable assignment of processes or tasks to nodes of the machine. Earlier works on graph embedding are usually not suitable for modern machines because the earlier works used metrics suitable for a store-and-forward communication mechanism. On modern machines on the other hand, in the absence of network congestion, latency is quite independent of location; communication performance is limited by contention on specific links. Yet another significant difference is that the earlier works typically embedded graphs onto standard network topologies such as hypercubes and meshes. On massively parallel machines, jobs typically acquire only a fraction of the nodes available, and the nodes allocated do not correspond to any standard topology, even when the machine does. For example, we show below in figure 1.1 an allocation for 1000 nodes on the 3D torus Jaguar machine at ORNL. We can see that the nodes allocated are several discontinuous pieces of the larger machine. Assignment of tasks to nodes that take this into account can reduce communication overhead.

We have used a topology aware mapping to improve the performance of communication in the load balancing of QMC application on Jaguar. In this scheme, we obtained the physical topology of Jaguar (which implies the routing scheme) from the system administrator. Using this information, we rearrange the order of

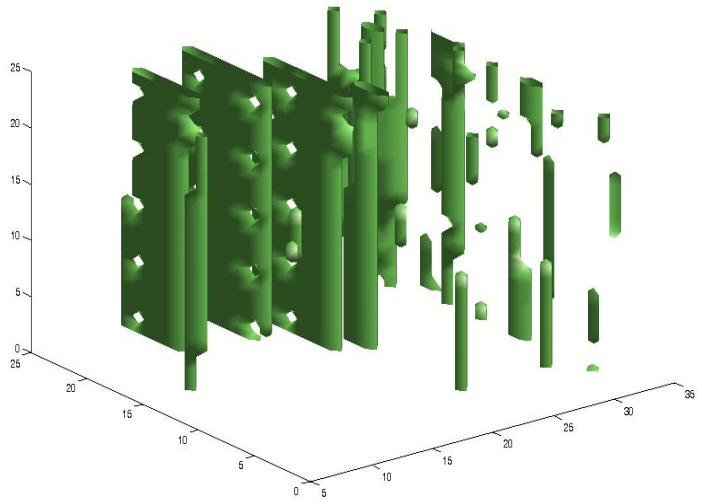

Figure 1.1: Allocation of 1000 nodes on Jaguar. The axes correspond to indices on the 3D torus, and the green region corresponds to allocated nodes.

the MPI processes by creating a new communicator that ranks the nodes according to their relative position on a space-filling curve: this ensures that each node is likely to send data to nearby nodes in the communications in this phase. Using this topology and routing aware process assignment, we were able to reduce the communication time in the load balancing phase by 60% with 120,000 cores and 20% on 12,000 cores, and this mapping also reduced MPI_allgather time by a similar amount.

Encouraged by our results on improving communication performance with topology and routing aware process assignment (60% improvement on 120,000 cores), we investigated the effectiveness of using this approach for several standard regular and irregular communication patterns. The mapping of processes to nodes is an NP-hard problem, and hence heuristics are used to solve it approximately. Recent works use heuristics that can be intuitively expected to reduce network congestion and then evaluate it either empirically or using some metric.

The hop-byte metric has attracted much attention recently. It is defined as the sum over all the messages of the product of the message size and number of hops the message has to traverse. On a store-and-forward network, this would correspond to the total communication volume. The intuition behind this metric is that if the total communication volume is high, then it is also likely to increase the contention for specific links, which would then become communication bottlenecks. Although this metric does not directly measure the communication bottleneck caused by contention, heuristics with low values of this metric tend to have smaller communication overheads. This serves as a justification for this metric. The advantage of this metric is that it requires only the machine topology, while computing contention would require routing information.

In contrast to other approaches, we use the hop-byte metric for producing the mappings too, rather than just using it for evaluating the mapping. Optimizing for the hop-byte metric can easily be shown to be a specific case of the Quadratic Assignment Problem (QAP), which is NP hard. Exact solutions can be determined using branch and bound for small problem sizes. We use the existing GRASP heuristic for medium-sized problem. An advantage of the QAP formulation is that we can use theoretical lower bounds to judge the quality of our solution. The time taken to determine the mapping using an exact solver or GRASP increases rapidly with problem size. In that case, we consider a couple of alternate approaches. In the first case, we develop new heuristics that improve on some limitations of other heuristics for this problem. In the second case, we use graph partitioning to break up the problem into smaller pieces, and then apply GRASP to each partition.

We evaluate our approach on six different communication patterns. We determine the values of the metric for different heuristics using node allocations obtained on the Kraken supercomputer at NICS. In contrast to other works that usually assume some standard topology, our results are based on actual allocations obtained. We see up to 75% reduction in hop-bytes over the default allocation, and up to 66% reduction over existing heuristics. Furthermore, our results

are usually within a factor of two of a theoretical lower bound on the solution. For small problem sizes, that lower bound is usually just a little over half the exact solution. Consequently, it is likely that our solutions are close to optimal.

The latest massively parallel systems are predominantly being built from nodes with multi-cores. This trend is evident when considering the systems of the TOP500 list of supercomputers [1], which are expected to feature, in a close future, fat many-core nodes composed of as many as a hundred of cores. One of the first supercomputer to reach the petaflop mark, the Roadrunner supercomputer at LANL uses IBM Cell processor blades. The bulk of the computational workload on the Cell processor is carried by eight co-processors called SPEs. The SPEs are connected to each other and to main memory by a high speed bus called the Element Interconnect Bus (EIB). The bandwidth utilization on EIB is reduced due to the congestion created by the simultaneous communications. We observed that the effective bandwidth obtained for inter-SPE communication is strongly influenced by the assignment of threads to SPEs (Thread-SPE affinity). We have observed a performance improvement of around 10%-12% for a communication intensive Monte Carlo particle simulation application.

1.2 Related Work

Mapping of processes to nodes based on the network topology attracted much attention in the earlier days of parallel computing. It lost its importance for some time with the advent of communication mechanisms such as worm-hole routing. However, for reasons explained in section 1.1, it had once again attracted much attention recently.

Different topological strategies for mapping torus process topologies onto the torus network of Blue Gene/L were presented by Yu, Chung, and Moreira [21]. Bhatele and Kale [17] proposed topology-aware load-balancing strategies for Molecular Dynamics applications using CHARM++. Their analysis maps mesh and

torus process topologies to other mesh and torus network topologies. Several techniques for mapping regular communication graphs onto regular topologies were developed [19]. The mapping is a NP-hard problem [25]; hence heuristics are used to approximate the optimal solution. Heuristic techniques for mapping applications with irregular communication graphs to mesh and torus topologies were developed, and some of them even take advantage of the physical coordinate configuration of the allocated nodes [20]. The performance of these heuristics were evaluated based on the hop-byte metric. Hoefler and Snir [22] present mapping algorithms that are meant for more generic use and the algorithms are evaluated using the maximum congestion metric – the message volume on the link with maximum congestion. Their heuristics based on recursive bisection and graph similarity were used to map application communication patterns on realistic topologies. The metrics here again are used to evaluate the mappings rather than being used to determining the mapping. The algorithm Greedy Graph Embedding (GGE) proposed in [22] is used by us for comparison, because it performs best among the heuristics they have proposed. Furthermore, the algorithm can be used with arbitrary communication patterns and network topologies, even though the implementation in [22] was more restricted.

Krishna et al. developed topology aware collective algorithms for Infiniband networks. These networks are hierarchical with multiple levels of switches, and this knowledge was used in designing efficient MPI collective algorithms [23]. The reduced scalability of the latency and effective bandwidth due to interconnect hot spot for fat-tree topologies is addressed with topology-aware MPI node ordering and routing-aware MPI collective operations [24]. Graph partitioning libraries such as SCOTCH and Metis provide support for mapping graphs to network topologies.

Massively parallel systems such as Cray XT and Blue Gene/P systems are generally heavily loaded with multiple jobs running and sharing the network resources simultaneously, this results in application performance being dependent

on the node allocation for a particular job. Balaji et al. analysed the impact of different process mappings on application performance on a massive Blue Gene/P system [26]. They show that the difference can be around 30% for some applications and can even be two fold for some. They have developed a scheme whereby the user can describe the application communication pattern before running a job, and the runtime system then provides a mapping that potentially reduces contention.

The rest of the synopsis report is organized as follows. In chapter 2, we discuss the intra-node mapping techniques used for the Cell processor along with the description of a tool developed to automate the mapping. We then describe the inter-node mapping techniques on large supercomputers. We first describe the application specific mapping technique in chapter 3, followed with the description of more generic mapping techniques developed using heuristics in chapter 4. These techniques can used be for any standard regular and irregular communication patterns. We finally conclude with a summary of the research and future research directions.

2. Optimizing Assignment of Threads to SPEs on the Cell BE Processor

2.1 Introduction

The SPEs are connected to each other and to main memory by a high speed bus called the EIB, which has a bandwidth of 204.8 GB/s. The latency between each pair of SPEs is identical for short messages and so affinity does not matter in this case. In the absence of contention on the EIB, the bandwidth between each of pair of SPEs is identical for long messages too, and reaches the theoretical limit. However, we observed that in the presence of contention, the bandwidth falls well short of the theoretical limit. This happens when the message size is greater than 16 KB. It is, therefore, important to assign threads to SPEs to avoid contention, in order to achieve optimal effective bandwidth for the application with a given communication pattern.

The outline of the rest of the chapter is as follows. Important architectural features of the Cell processor will be summarized first. Then, the factors responsible for reduced performance of inter SPE communication are described. A tool to automate the mapping process, and its evaluation using it for few standard communication patterns and a practical application will be described next.

2.2 Cell Communication Architecture

We summarize below the architectural features of the Cell of relevance to this work, concentrating on the communication architecture. Further details can be found in [6]. Figure 2.1 provides an overview of the Cell processor. It contains a cache-coherent PowerPC core called the PPE, and eight co-processors, called SPEs, running at 3.2 GHz each. An XDR memory controller provides access to

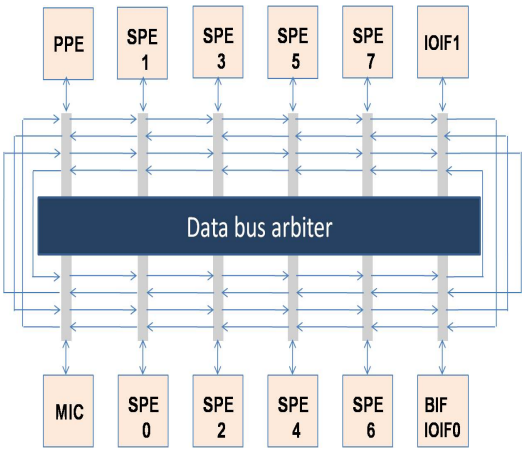

Figure 2.1: Overview of the Cell communication architecture.

main memory at 25.6 GB/s total, in both directions combined. The PPE, SPE, and memory controller are connected via the EIB. The maximum bandwidth of the EIB is 204.8 GB/s. In a Cell blade, two Cell processors communicate over a BIF bus. The numbering of SPEs on processor 1 is similar, except that we add 8 to the rank for each SPE. The data can be transferred much faster between SPEs than between SPE and main memory [6]. It is, therefore, advantageous for the algorithms to be structured such that SPEs communicate directly between themselves over the EIB, and make less use of memory.

The data transfer time between each pair of SPEs is independent of the positions of the SPEs, if there is no other communication taking place simultaneously [7]. However, when many simultaneous messages are being transferred, transfers to certain SPEs may not yield optimal bandwidth, even when the EIB has sufficient bandwidth available to accommodate all messages. In order to explain this phenomenon, we now present further details on the EIB. The EIB contains four rings, two running clockwise and two running counter-clockwise. All rings have identical bandwidths. Each ring can simultaneously support three data transfers, provided that the paths for these transfers don't overlap. The EIB data bus arbiter handles a data transfer request and assigns a suitable ring to a request. When

a message is transferred between two SPEs, the arbiter provides it a ring in the direction of the shortest path. For example, transfer of data from SPE 1 to SPE 5 would take a ring that goes clockwise, while a transfer from SPE 4 to SPE 5 would use a ring that goes counter-clockwise. If the distances in clockwise and anti-clockwise directions are identical, then the message can take either direction, which may not necessarily be the best direction to take, in the presence of contention. From these details of the EIB, we can expect that certain combinations of affinity and communication patterns can cause non-optimal utilization of the EIB.

2.3 Influence of Affinity on Inter-SPE Communication Performance

As mentioned in [7], affinity significantly influences the communication performance when there is contention. The factors that lead to loss in performance are identified.

2.3.1 Summary of Experimental Results

Several experiments using various affinities on different communication patterns were performed [7]. We noticed that the communication patterns where all the messages go in a single direction can use only half of the EIB bandwidth, as they do not use two of the rings of the EIB. And also, messages with overlapping paths create congestion on the EIB, which leads to performance degradation. We observed that messages that travel half-way across the ring can go in either direction.

2.3.2 Affinity on a Cell Blade

Communication on a Cell blade is asymmetric, with around 30 GB/s theoretically possible from Cell 0 to Cell 1, and around 20 GB/s from Cell 1 to Cell

0. However, we observed that communication between a single pair of SPEs on different processors of a blade yields bandwidth much below this theoretical limit [7] and [5]. In fact, this limit is not reached even when multiple SPEs communicate, for messages of size up to 64 KB each. The bandwidth attained by messages between SPEs on different processors is much lower than that between SPEs on the same processor [7]. So, these messages become the bottleneck in the communication. In this case, the affinity within each SPE is not as important as the partitioning of threads amongst the two processors.

2.4 Communication Model

Based on the understanding from the above mentioned experimental analysis on specific communication patterns, we created a communication model that determines a mapping with less communication volume (cost). The main purpose of creating a quantitative model is that we can find an optimal affinity for an arbitrary communication pattern, without being ingenious. We evaluate all possible affinities, and use the model to give a measure of how good each affinity is. We choose the best one. The communication model was developed based on the following guiding principles.

- For good communication performance, the communication load should be distributed equally across all the four EIB rings.
- Each ring can simultaneously support three data transfers, provided that the paths for these transfers don't overlap. Model should look for a mapping, where the paths taken by the messages do not overlap often.
- The Model takes into account the asymmetry in the bandwidth between two processors, and so a partition sending more data will be placed on processor 0.

2.4.1 Evaluation of the model

We now evaluate the effectiveness of the model by using it for determining affinities for some standard communication patterns and also for a real application. Model's affinity in the figures denotes the affinity determined by using the communication model. Figure 2.2 shows the effective bandwidths achieved for the Ring communication pattern using different mappings . Figure 2.3 shows the effective bandwidths for the first phase of Recursive Doubling. These results show that the performance of mapping given by the Model is not as good as performance with the optimal mapping, however it gives better performance than using the Default mapping.

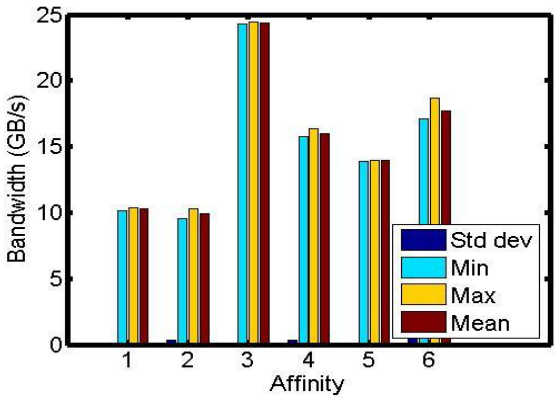

Figure 2.2: Performance with the following mappings: 1) Overlap 2) Default 3) EvenOdd 4) Identity 5) Ring 6) Model's Affinity

We next consider the performance of the communication model on a Monte Carlo application for particle transport, which tracks a number of random walkers on each SPE [8]. We used the diffusion scheme to balance the load between the SPEs. We can see a factor of two difference between the communication time cost for the best and worst affinities in figure 2.4. Figure 2.5 shows a difference in total application performance of over 10% between the best and worst affinities. We can observe from the above figures that the affinity given by the model

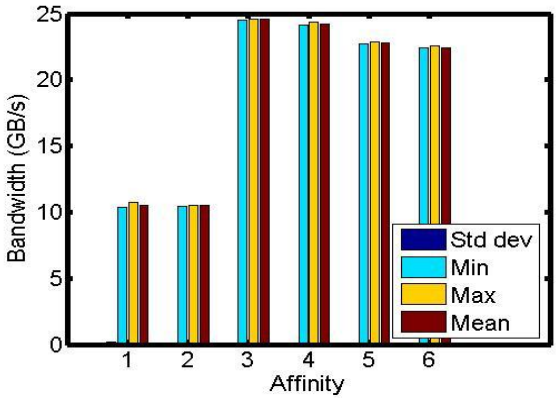

Figure 2.3: Performance with the following mappings: 1) Overlap 2) Default 3) EvenOdd 4) Identity 5) Ring 6) Model's Affinity

results in sub-optimal performance. This could be due to the fact that the three guiding principles used in building the model are not sufficient to accurately represent the communication network. For example, we observed a poor bandwidth performance even with three non overlapping messages between SPEs going in the same direction. This occurs when at least two of them are of path length two or more, and go across the sides of the ring (i.e., through PPE-MIC or BIE-IOIF1 in figure 2.1). This peculiar behaviour of the non overlapping messages is not included in the model. The traffic to the main memory which also go through the same EIB, significantly affects inter-SPE communication performance.

Another issue is that in all the experiments, we considered only messages with equal size. We observed that messages with unequal sizes results in different behaviour in some cases. For example, we observed that the above noted peculiar behaviour of non overlapping messages does not occur if the messages are of different sizes. We also assumed symmetry in rotating the affinity, to reduce the number of affinities tested from 8! to 7!, however, our experiments with some communication patterns indicated that such symmetry does not exist. We could modify our model by taking into consideration all the aspects of the communication network mentioned above, to make it more complete and robust.

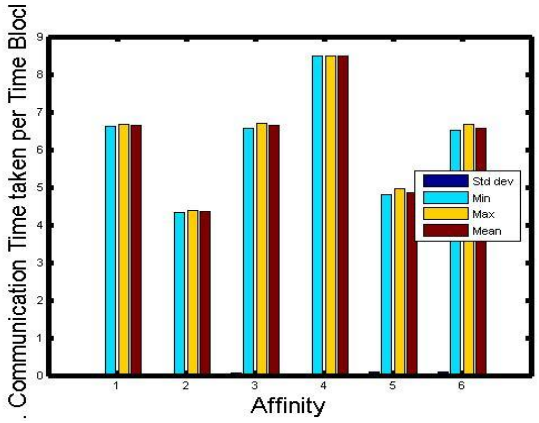

Figure 2.4: Performance with the following mappings: 1) Overlap 2) Default 3) EvenOdd 4) Identity 5) Ring 6) Model's Affinity

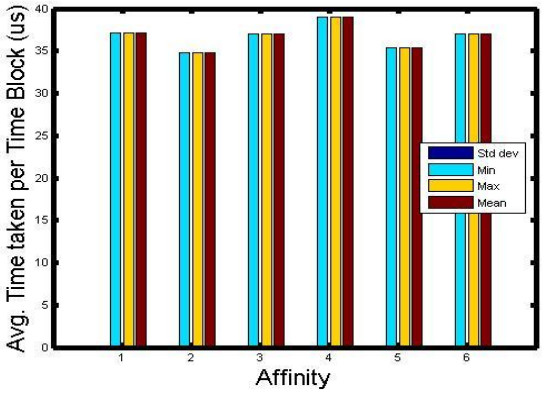

Figure 2.5: Performance with the following mappings: 1) Overlap 2) Default 3) EvenOdd 4) Identity 5) Ring 6) Model's Affinity

3. Application Specific Mapping

3.1 Introduction

Diffusion Monte Carlo is the most accurate widely used Quantum Monte Carlo method for the electronic structure of materials, but it requires frequent load balancing or population redistribution steps to maintain efficiency and avoid accumulation of systematic errors on parallel machines. The load balancing step can be a significant factor affecting performance, and will become more important as the number of processing elements increases.

3.1.1 Load Balancing Model Definitions

Dynamic load balancing methods often consist of the following three steps. (i) In the *flow computation* step, we determine the number of tasks that need to be sent by each process to other processes. (ii) In the *task identification* step, we identify the actual tasks that need to be sent by each process. (iii) In the *migration* step, the tasks are finally sent to the desired processes. Since we deal with identical independent tasks, the second step is not important; any set of tasks can be chosen. Our algorithm determines the flow (step (i)) such that step (iii) will be efficient, under certain performance metrics.

We assume that a collection of P processes need to handle a set of T identical tasks (that is, each task requires the same computation time), which can be executed independently. Before the load balancing phase, the number of tasks with process i, $1 \leq i \leq P$, is T_i. After load balancing, each process will have at most $\lceil T/P \rceil$ tasks (we are assuming that the processors are homogeneous, and therefore process tasks at the same speed). This redistribution of tasks is accomplished by having each process i send t_{ij} tasks to processes j, $1 \leq i, j \leq P$, where non-zero values of t_{ij} are determined by our algorithm for flow computation, which we describe in § 3.2. Of course, most of the t_{ij}s should be zero, in order to reduce

the total number of messages sent. In fact, at most $P-1$ of the possible $P(P-1)$ values of t_{ij} will be non-zero in our algorithm.

The determination of t_{ij}s is made as follows. The processes perform an 'all-gather' operation to collect the number of tasks on each process. Each process k independently implicitly computes the flow (all non-zero values of $t_{ij},\ 1 \leq i, j \leq P$) using the algorithm in § 3.2, and then explicitly determines which values of t_{kj} and t_{jk} are non-zero, $1 \leq j \leq P$.

The algorithm to determine non-zero t_{ij}s takes $O(P)$ time, and is fast in practice. We wish to minimize the time taken in the actual migration step, which is performed in a decentralized manner by each process. In some load balancing algorithms [10], a process may not have all the data that it needs to send, and so the migration step has to take place iteratively, with a process sending only data that it has in each iteration. In contrast, the t_{ij}s generated by our algorithm never require sending more data than a process initially has, and so the migration step can be completed in one iteration. In fact, no process receives a message from more than one process, though processes may need to send data to multiple processes.

The outline of the rest of the chapter is as follows. We first provide our algorithm for dynamic load balancing. We describe the algorithm when T is a multiple of P, and then we show how the algorithm can be modified to deal with the situation when T is not a multiple of P. We also define a few metrics for the time complexity of the load re-distribution step, and theoretically evaluate our algorithm in terms of those. We then report results of empirical evaluation of our method and comparisons with an existing QMC dynamic load balancing implementation.

3.2 The Alias Method Based Algorithm for Dynamic Load Balancing

Our algorithm is motivated by the alias method for generating samples from discrete random distributions. We therefore refer to our algorithm as the Alias

method for dynamic load balancing. There is no randomness in our algorithm. It is, rather, based on the following observation used in a deterministic pre-processing step of the alias method for the generation of discrete random variables. If we have P bins containing kP objects in total, then it is possible to re-distribute the objects so that each bin receives objects from at most one other bin, and the number of objects in each bin, after the redistribution, is exactly k. Walker [12] showed how this can be accomplished in $O(P \log P)$ time. This time was reduced to $O(P)$ by [11] using auxiliary arrays. In algorithm 1 below, we describe our in-place implementation that does not use auxiliary arrays, except for storing a permutation vector.

We assume that the input to algorithm 1 is an integer array A containing the number of objects in each bin. Given A, we can compute k easily in $O(P)$ time, and will also partition it around k in $O(P)$ time so that all entries with $A[i] < k$ occur before any entry with $A[j] > k$. We will assume that $A[i] \neq k$, because such bins do not need to be considered – they have the correct number of elements already, and our algorithm does not require redistribution of objects to or from a bin that has k objects. If we store the permutation while performing the partitioning, then the actual bin numbers can easily be recovered after algorithm 1 is completed. This algorithm runs only with $P \geq 2$, because otherwise all the bins already have k elements each. We assume that a pre-processing step has already accomplished the above requirements in $O(P)$ time.

Algorithm 1: *Input:* An array of non-negative integers $A[1 \cdots P]$ and an integer $k > 0$, such that $\sum_{i=1}^{P} A[i] = kP$, entries of A have been partitioned around k, and $P \geq 2$. $A[i]$ gives the number of objects in bin i, and $A[i] \neq k$.

Output: Arrays $S[1 \cdots P]$ and $W[1 \cdots P]$, where $S[i]$ gives the bin from which bin i should get $W[i]$ objects, if $S[i] \neq 0$.

Algorithm:

1. Initialize arrays S and W to all zeros.

2. $s \leftarrow 1$.

3. $l \leftarrow \min\{j|A[j] > k\}$.

4. while $l > s$

   (a) $S[s] \leftarrow l$.

   (b) $W[s] \leftarrow k - A[s]$.

   (c) $A[l] \leftarrow A[l] - W[s]$.

   (d) if $A[l] < k$ then

      i. $l \leftarrow l + 1$.

   (e) $s \leftarrow s + 1$.

It is straightforward to see the correctness of algorithm 1 based on the following loop invariants at the beginning of each iteration in step 4: (i) $A[i] \geq k,\ l \leq i \leq P$, (ii) $0 \leq A[i] < k,\ s \leq i \leq l-1$, and (iii) $A[i] + W[i] = k,\ 1 \leq i \leq s-1$. Since bin l needs to provide at most k objects to bin s, it has a sufficient number of objects available, and also as a consequence of the same fact, $A[l]$ will not become negative after giving $W[s]$ objects to bin s. The last clause of the loop invariant proves that all the bins will have k objects after the redistribution. We do not formally prove the loop invariants, since they are straightforward.

In order to evaluate the time complexity, note that in the while loop in step 4, l and s can never exceed P. Furthermore, each iteration of the loop takes constant time, and s is incremented once each iteration. Therefore, the time complexity of the while loop is $O(P)$. Step 3 can easily be accomplished in $O(P)$ time. Therefore the time complexity of this algorithm is $O(P)$.

Load balancing when T is a multiple of P Using algorithm 1, a process can compute t_{ij}s as follows, if we associate each bin with a process[1] and the number of objects with the number of tasks:

$$t_{S[i]i} \leftarrow W[i],\ S[i] \neq 0. \tag{3.1}$$

All other t_{ij}s are zero. Of course, one needs to apply the permutation obtained from the partitioning before performing this assignment. Note that the loop invariant mentioned for algorithm 1 also shows that a process always has sufficient data to send to those that it needs to; it need not wait to receive data from any other process in order to have sufficient data to send, unlike some other dynamic load balancing algorithms [10].

The above case considers the situation when the total number of tasks is a multiple of the total number of processes. We can also handle the situation when this is not true, by adding the required fake phantom tasks before applying above algorithm.

We next analyze the performance of the migration step of the load balancing algorithm, when using the t_{ij}s as computed by algorithm 1 in § 3.2. We define a few performance metrics, and give the approximation ratio of our algorithm (that is, an upper bound on the ratio of the time taken by our algorithm to that of an optimal one, in the metric considered). The results are summarized in Table 3.1.

3.2.1 Experimental Setup

The experimental platform is the Cray XT5 Jaguar[2] supercomputer at ORNL. It contains 18,688 dual hex-core Opteron nodes running at 2.6GHz with 16GB

[1] In our algorithm, processes that already have a balanced load do not participate in the redistribution of tasks to balance the load. Therefore, we use P to denote the number of processes with *unbalanced loads* in the remainder of the theoretical analysis.

[2] Jaguar was upgraded recently with XK6 instead of XT5, the experiments were run on the system prior to the upgrade.

Method	Approximation factor
Maximum-Receives	1
Total-Messages	2
Maximum-Tasks-Sent	∞
Maximum-Sends	∞
Total-Tasks-Sent	∞
Maximum-Tasks-Received	1

Table 3.1: *Approximation factors under different metrics.*

memory per node. The peak performance of the machine is 2.3 petaflop/s. The nodes are connected with SeaStar 2+ routers having a peak bandwidth of 57.6GB/s, with a 3-D torus topology. Compute Node Linux runs on the compute nodes.

In running the experiments, we have two options regarding the number of processes per node. We can either run one process per node or one process per core. QMC software packages were originally designed to run one MPI process per core. The trend now is toward one MPI process per node, with OpenMP threads handling separate random walkers on each core. Qmcpack already has this hybrid parallelization implemented, and some of the other packages are expected to have it implemented in the near future. We assume such a hybrid parallelization, and have one MPI process per node involved in the load balancing step.

In our experiments, we consider a granularity of 24 random walkers per node, that is, 2 per core. This is a level of granularity that we desire for QMC computations in the near future. Such scalability is currently limited by the periodic collective communication and load balancing that is required. Both these are related in the following manner. The first step leads to termination or creation of new walkers, which in turn requires load balancing. There is some flexibility in the creation and termination of random walkers. Ideally, the load balancing results both in reduced wall clock time be step (of all walkers) and an improved statistical

efficiency.

The above two steps should ideally be performed after every time step, in order to accelerate convergence. However, the computational overhead on large parallel machines can hinder this, and so one may perform them every few iterations instead. Our goal is to reduce these overheads so that these steps can be performed after every time step. For large physical systems where the computational cost per step is very high, these overheads may be relatively small compared with the computation cost. However, for small to moderate physical systems, these overheads can be relatively large, and we wish to efficiently apply QMC even to small systems, on the largest systems. We consider a small system, a Cr_2 molecule, with an accurate multideterminant trial wavefunction. The use of multideterminants increases computational time over the use of a single determinant. However, it provides greater accuracy, which we desire when performing a large run. The computation time per time step per walker is then around 0.1 seconds. The two collective steps mentioned above consume less than 10% of the total time on a large machine (the first step does not involve just collective communication, but also involves other global decisions, such as branching). Even then, on 100,000 cores, this is equivalent to wasting 10,000 cores. On larger systems, the fraction of time consumed by these steps further increases. We can expect these collective steps to consume a larger fraction of time at even greater scale.

The focus of the current work is on the second overhead – load balancing. In evaluating our load balancing algorithm, we used samples from the load distribution observed in a long run of the above physical system. Depending on the details of the calculations, the amount of data to be transferred for each random walker can vary from 672B to 32KB for Cr_2. We compared our algorithm against the load balancing implementation in QWalk. The algorithm used in QWalk is optimal in the maximum number of tasks sent by any processor and in the total number of tasks sent by any processor, but not on the maximum or total number of messages sent; these are bounded by the maximum imbalance and the sum

of load imbalances respectively. One may, therefore, expect that algorithm to be more efficient than ours for a sufficiently large task sizes, and ours to be better for small sizes. Also, the time taken for the flow computation in QWalk is O(P + total_load), where P is the number of nodes.

Each experiment involved 11 runs. As we show later, inter-job contention on the network can affect the performance. In order to reduce its impact, we ignore the results of the run with the largest total time. In order to avoid bias in the result, we also drop the result of the run with the smallest time. For a given number of nodes, all runs for all task sizes for both algorithms are run on the same set of nodes, with one exception mentioned later.

3.2.2 Results

We next consider typical load imbalances generated. The maximum across all cores, of the difference between load on a core and the average load across all the cores [i.e., max (load_i - average load)], in a particular run is a reasonable measure of the imbalance for that run, because this would limit the computational speed. We compute the average, over all runs, for this quantity, and show it in fig. 3.1, along with the range of values that it takes. The range is 1, indicating not much variation. However, considering that we have a target of 2 tasks per core, the imbalance relative to the mean number of tasks is large. We also see that the imbalance increases, albeit slowly, with increase in the number of cores. As high-end machines get larger, we can expect the imbalance to become increasingly larger.

The maximum time taken by any node can be considered a measure of the performance the algorithm, because the slowest processor limits the performance. Figure 3.2 shows the average, over all the runs, of the maximum time for the following components of the algorithm. (We refer to it in the figure caption as the 'basic alias method', in order to differentiate it from a more optimized implementation described later.) Note that the maximum for each component may occur

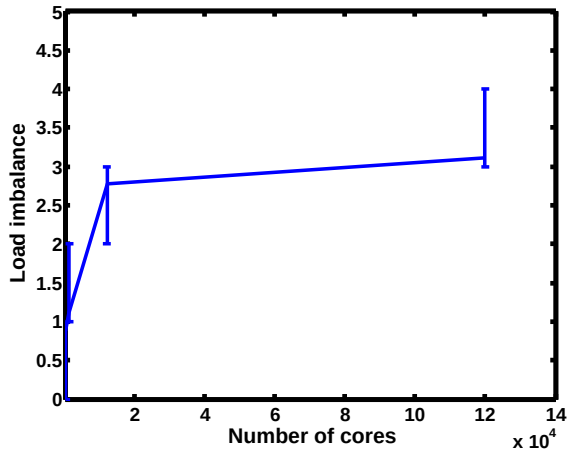

Figure 3.1: Initial load imbalance.

on different cores, and so the maximum total time for the algorithm over all the cores may be less than the sum of the maximum times of each component. We can see that communication operations consume much of the time, and the flow computation is not the dominant factor, even with a large number of nodes. The MPI_Isend and MPI_Irecv operations take little time. However MPI_Waitall and MPI_Allgather consume a large fraction of the time. It is possible to overlap computation with communication to reduce the wait time. However, the all-gather time is still a large fraction of the total time.

In interpreting the plots in this figure, one needs to note that it is drawn on a semi-log scale. The increase in time, which appears exponential with the number of cores, is not really so. A linear relationship would appear exponential on a semi-log scale. On the other hand, one would really expect a sub-linear relationship for the communication cost. The all-gather would increase sub-linearly under common communication cost models. In the absence of contention, the cost of data transfer need not increase with the number of cores for the problem considered here; the maximum imbalance is 4, each node has 6 links, and so, in principle, if the processes are ideally ordered, then it is possible for data to travel on different links to nearby neighbors which would be in need of tasks. The com-

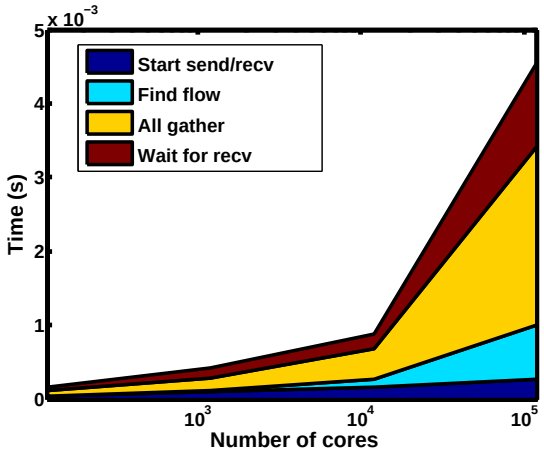

Figure 3.2: Maximum time taken for different components of the basic Alias method with task size 8KB.

munication time can, thus, be held constant. We can see from this figure that the communication cost (essentially the wait time) does increase significantly. The communication time for 12,000 cores is 2-3 times the time without contention, and the time with 120,000 cores is 4-6 times that without contention. The cause for contention is that the routing on this machine uses fixed paths between pairs of nodes, and sends data along the x coordinate of the torus, in the direction of the shortest distance, then in the y direction, and finally in the z direction[3]. Multiple messages may need to share a link, which causes contention.

In fig. 3.3, we consider the mean value of the different components in each run, and plot the average of this over all runs. We can see that the wait time is very small. The reason for this is that many of the nodes have balanced loads. The limiting factor for the load balancing algorithm is the few nodes with large work.

We next optimize the alias method to reduce contention. We would like nodes to send data to nearby nodes. We used a heuristic to accomplish this. We obtained the mapping of node IDs to x, y, and z coordinates on the 3-D torus. We also found a space-filling Hilbert curve that traverses these nodes. (A space-filling

[3]Personal communication from James Buchanan, OLCF, ORNL.

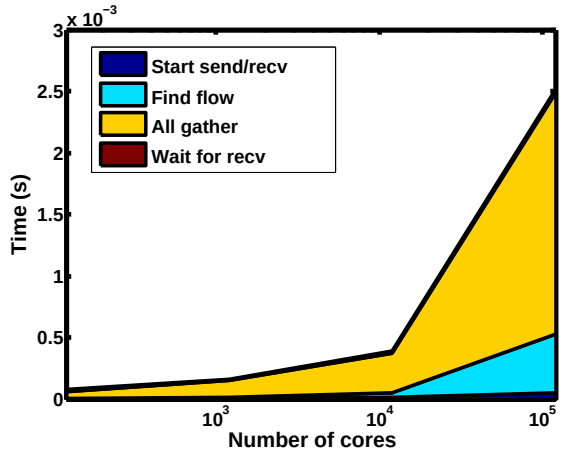

Figure 3.3: Mean time taken for different components of the basic Alias method with task size 8KB.

curve tries to order nodes so that nearby nodes are close by on the curve.) At run time, we obtain the node IDs, and create a new communicator that ranks the nodes according to their relative position on the space-filling curve. We next changed the partitioning algorithm so that it preserves the order of the space-filling curve in each partition. We also made slight changes to the alias algorithm so that it tries to match nodes based their order on the space filling curve. The creation of a new communicator is performed only once, and the last two steps don't have any significant impact on the time taken by the alias method. Thus, the improved algorithm is no slower than the basic algorithm. Figure 3.4 shows that the optimized algorithm has much better performance than the basic algorithm for large core counts. It is close to 30% better with 120,000 cores, and 15-20% better with 1,200 and 12,000 cores.

We next analyze the reason for the improved performance. Figure 3.5 considers the average over all runs for the maximum time taken by different components of the algorithm. As with the analysis of the basic algorithm, the total maximum time is smaller than the sum of the maximum times of each component. We can see that the wait time is smaller than that of the basic algorithm shown in fig. 3.2,

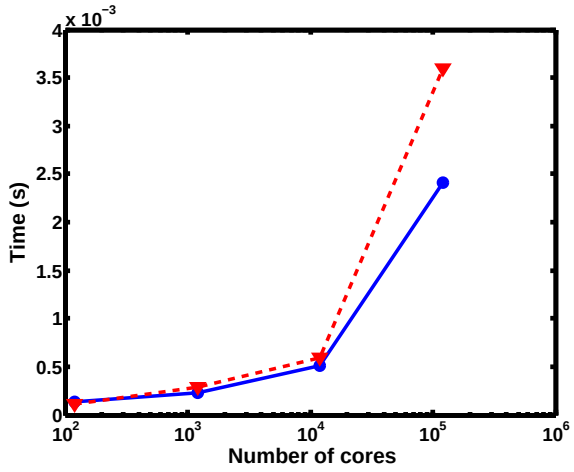

Figure 3.4: Comparison of optimized Alias method against the basic method with task size 8KB.

which was the purpose of this optimization. The improvement is around 60% with 120,000 cores and 20% on 12,000 core. Surprisingly, the MPI_Allgather time also reduces by around 30% on 120,000 cores and 20% on 12,000 cores. It appears that the MPI implementation does not optimize for the topology of the nodes that are actually allocated for a run, and instead uses process ranks. The ranks specified by this algorithm happens to be good for the MPI_Allgather algorithm. This improvement depends on the nodes allocated. In the above experiment, with 120,000 cores, the set of allocated nodes consisted of six connected components. In a different run, we obtained one single connected component. The use of MPI_Allgather with the optimized algorithm did not provide any benefit in that case. It is possible that the MPI implementation optimized its communication routines under the assumption of a single large piece of the torus. When this assumption is not satisfied, perhaps its performance is not that good.

The performance gains are smaller with smaller core counts, which can be explained by the following observations. Figure 1.1 shows the node allocation for the 12,000 core run, We can see that we get a large number of connected components. Thus, inter-job contention can play an important role. Each com-

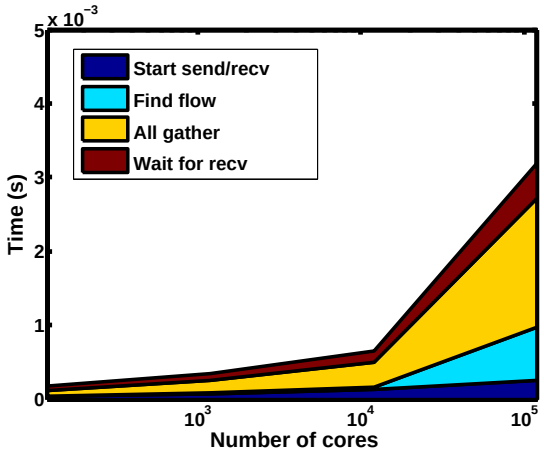

Figure 3.5: Maximum time taken for different components of the Optimized Alias method with task size 8KB.

ponent is also not shaped close to a cube. Instead, we have several lines and 2-D planes, long in the z direction. This makes it hard to avoid intra-job contention, because each node is effectively using fewer links, making contention for links more likely. It is perhaps worthwhile to consider improvements to the node allocation policy. For 120 and 1,200 cores, typically each connected component is a line (or a ring, due to the wrap-around connections), which would lead to contention if there were several messages sent. However, the number of nodes with imbalance is very small, and contention does not appear to affect performance in the load migration phase. Consequently, improvement in performance is limited to that obtained from the all-gather operation.

4. Generic Mapping on Massively Parallel Machines

4.1 Mapping Problem formulation

We model the problem using two graphs. The node graph G is an undirected graph with vertices representing the nodes of the machine that have been allocated to the job submitted, and edge weights e_{ij} representing the number of hops between the vertices i and j linked by that edge. The hops can be determined from the machine topology information and knowledge of the nodes actually allocated. These can usually be obtained on most supercomputing systems. We assume that static routing is used, which is fairly common.

The task graph $G' = (V', E')$ represents the submitted job. Each vertex represents a process running on a node and edge weight e'_{ij} represents the total sizes of messages, in both directions, between vertices i and j linked by that edge. The number of vertices in this graph must equal the number in the node graph. If there are more tasks than nodes, then a graph partitioning algorithm can be applied to aggregate tasks so that this condition is satisfied.

Minimizing the hop-bytes then is the following quadratic assignment problem, where x_{ij} = 1 implies that task j is assigned to node i.

$$min \sum_{ij} \sum_{kl} e_{ik} e'_{jl} x_{ij} x_{kl}, (1)$$

subject to:

$$\sum_{i} x_{ij} = 1, for\ all\ j$$

$$\sum_{j} x_{ij} = 1, for\ all\ i$$

x_{ij} in {0,1}

This is a well known NP hard problem and considered hard to approximate, though there are reasonable approximation algorithms [14] for dense instances. The exact solution can be found for small instances using branch and bound (bounds are used to reduce the search space). A couple of popular lower bounds are the elimination bound and the Gilmore-Lawler bound. We use the exact solution for small instances, and also the bounds for medium size instances, in order to evaluate the effectiveness of our heuristics.

4.2 Mapping Heuristics

4.2.1 GRASP heuristic

Several heuristics have been proposed for QAP, based on meta-heuristics such as simulated annealing, tabu search, ant colony optimization, and greedy randomized adaptive search procedures (GRASP). GRASP is based on generating several random initial solutions, finding local optima close to each one, and choosing the best one. We use a GRASP heuristic for QAP from [13].

4.2.2 MAHD and exhaustive MAHD heuristics

We first describe our faster heuristic, Minimum Average Hop Distance (MAHD), in algorithm 1 below. It improves on the following limitation of the GGE [22] heuristic. GGE replaces step 7 of algorithm 1 with a strategy that places the task on the node closest to its most recently mapped neighbor. We would ideally like this task to be close to all its neighbors. MHT [20] addresses this by placing the node closest to the centroid of all previously mapped neighbors. On the other hand MHT works only on meshes. Our algorithm works on a general graph, and places the task, in this step, on the node that has the minimum average hop dis-

tance to nodes on which all previously mapped neighbors of the task have been mapped. MHT also selects a random node on which to place the initial task. We intuitively expect a task with the maximum number of neighbors to be a "central" vertex in a graph, and so try to map it to a node which is "central" in its graph. We do this by placing it on the node with the minimum average hop distance to any other vertex. The first task selected may not actually be "central" (for instance, in the sense centrality measures such as betweenness centrality). We introduce an Exhaustive MAHD (EMAHD) heuristic to see if a better choice of initial vertex is likely to lead to significant improvement in mapping quality. In this heuristic, we try all possible nodes as starting vertices, and then choose the one that yields the best time.

Algorithm 1 MAHD(G, G')

1. s = vertex in G with maximum number of neighbors
2. p = vertex in G' with minimum average hop distance to all other vertices
3. Assign task s to node p
4. Insert all neighbors of s into max-heap H, where the heap is organized by the number of neighbors
5. **while** H in not empty **do**
6. s = H.pop();
7. s is mapped to a node with minimum average hop distance to processes hosting mapped neighbors of s;
8. Insert neighbors of s into H if they are not in H and have not been mapped;
9. **end while**

4.2.3 Hybrid heuristic with graph partitioning

If $|V'|$ is too large for GRASP to be feasible, then we use the following hybrid heuristic. We partition graphs G and G' into partitions of size p each. Any graph partitioning algorithm can be used. We use a multilevel heuristic available in

parMetis. We create graphs H and H' corresponding to the partitions of G and G' respectively. In H, each vertex corresponds to a partition in G and in H', each vertex corresponds to a partition in G'. Each edge in H has weight corresponding to the average hops from nodes between the two partitions linked by that edge. Each edge in H' has weight corresponding to the total message sizes between the two partitions linked by that edge. A mapping of partitions in H' to partitions in H is performed using GRASP and mapping of tasks to nodes within each partitions is again performed using GRASP with corresponding subgraphs of G and G'.

4.3 Evaluation of Heuristics

4.3.1 Experimental platform

The experimental platform is the Cray XT5 Kraken supercomputer at NICS. It contains 18,816 dual hex-core Opteron nodes running at 2.6 GHz with 8 GB memory per socket. The nodes are connected by SeaStar 2+ routers with a 3-D torus topology having dimensions 25 x 16 x 24. Compute Node Linux runs on the compute nodes. We used the native Cray compiler with optimization flag "$-O3$".

The QAP codes were obtained from QAPlib (http://www.opt.math.tu-graz.ac.at/qaplib/codes.html), which includes codes from a variety of sources. A branch and bound algorithm was used for the exact solution, the Gilmore-Lawler bound for a lower bound[1] and a dense GRASP heuristic for larger problem sizes.

Three standard collective communication patterns used in MPI implementations – recursive doubling, Bruck, and binomial tree – were studied. We also used the following three irregular communication patterns. A 3-D spectral element elastic wave modeling problem (3Dspectralwave) and a 2-D PDE (aug2dc) from

[1]The Gilmore-Lawler bound performed better than the elimination bound for large problem sizes.

the University of Florida sparse matrix collection, and a 2D unstructured mesh pattern from the ParFUM framework available in CHARM++ library.

OSU MPI micro benchmark suite was used for the empirical tests on the Kraken machine to observe the impact of the heuristic based mapping on MPI collective calls.

4.3.2 Experimental Results

Figure 4.1 compare the default mapping, GRASP result, and the Gilmore-Lawler bound for small problem sizes. The quality (hop-byte metric value) is divided by that for the exact solution to yield a normalized quality.

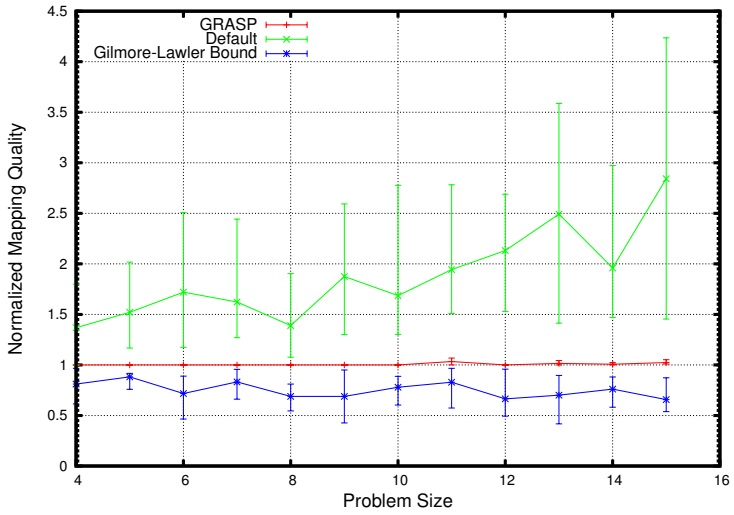

Figure 4.1: Quality of solutions on the recursive doubling pattern for small problem sizes.

We can see that the GRASP is close to the exact solution for these problem sizes. We also note that the lower bound is a little over the half of the exact solutions toward the higher end of this size range. Similar trend were observed for the other patterns, which are not shown here.

Figures 4.2- 4.4 compare the heuristics for medium problem sizes. The quality is normalized against the default solution, because computing with the exact

solution is not feasible. These figures, therefore, show improvement over the default mapping.

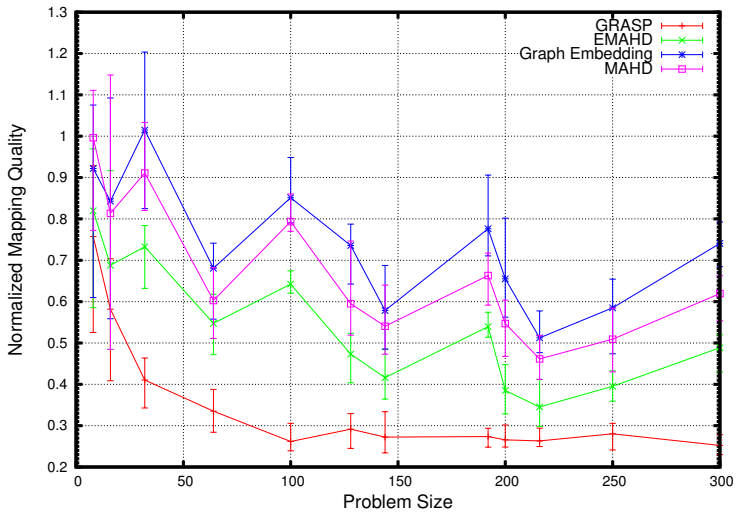

Figure 4.2: Quality of solution on the recursive doubling pattern for medium problem sizes.

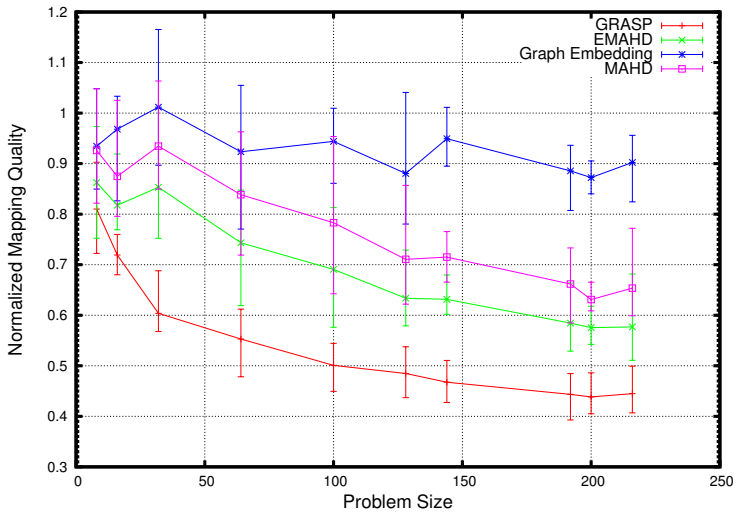

Figure 4.3: Quality of solution on the 3D spectral pattern for medium problem sizes.

We can see that the GRASP heuristic is consistently better than the default and the GGE heuristic, while MAHD and EMAHD are sometimes comparable

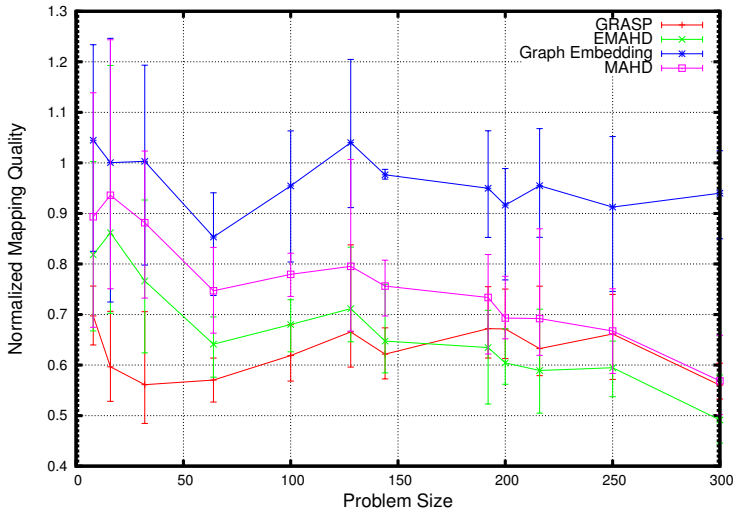

Figure 4.4: Quality of solution on the mesh pattern for medium problem sizes.

to GRASP. EMAHD is often much better than MAHD, suggesting that a better choice of the initial vertex has potential to make significant improvement to MAHD.

However, the time taken by GRASP and EMAHD are significantly larger than that for GGE or MAHD, as shown in figure 4.5. Consequently, they are more suited to static communication patterns. Since EMAHD typically does not produce better quality than GRASP either, it does not appear very useful. On the other hand, its quality suggests that if a good starting vertex can be found for MAHD without much overhead, then MAHD's quality can be improved without increasing its run time. When the communication pattern changes dynamically, then MAHD is a better alternative to the above two schemes and also to GGE. It is as fast as GGE, while producing mappings of better quality. Its speed also makes it feasible to use it dynamically, while GRASP is too slow.

An alternative to MAHD for large problem sizes with dynamic communication patterns is the hybrid algorithm. Preliminary results on 1000 nodes with partitions of size 125 are shown in figure 4.6. The hybrid algorithm performs better than the default and GGE with both communication patterns. It is better than MAHD

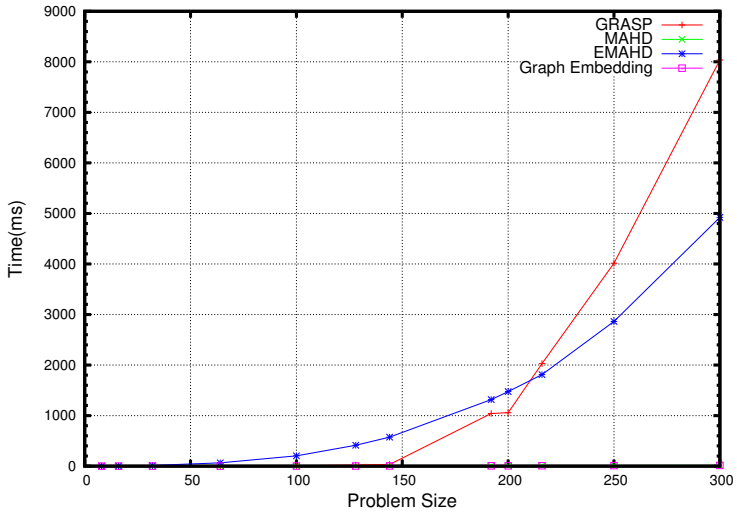

Figure 4.5: Comparison of time taken by the heuristics.

and EMAHD for recursive doubling, but is worse with the binomial tree. The binomial tree has less communication volume than recursive doubling, and further experiments are necessary to check if the hybrid algorithm tends to perform better when the communication volume is larger. We note that even for medium sized problems, GRASP (which is the underlying heuristic behind the hybrid algorithm), was comparable with EMAHD and MAHD for the binomial tree, but much better for recursive doubling. The hybrid scheme produces a further reduction in quality, which makes it worse than MAHD and EMAHD for binomial tree, but since GRASP is much better for recursive doubling, the hybrid algorithm is better than MAHD and EMAHD for it, though by a smaller margin.

We next evaluate how well GRASP compares with the lower bound for medium problem sizes. Figures 4.7- 4.9 show that GRASP is around a factor of two from the Gilmore-Lawler bound. As shown earlier, the above bound was usually a little higher than half the exact solution for small problem sizes, and did not get tighter with increased sizes. These results, therefore, suggest that GRASP is close to the optimal solution.

Finally, we wish to verify if improving the hop-byte metric actually improves

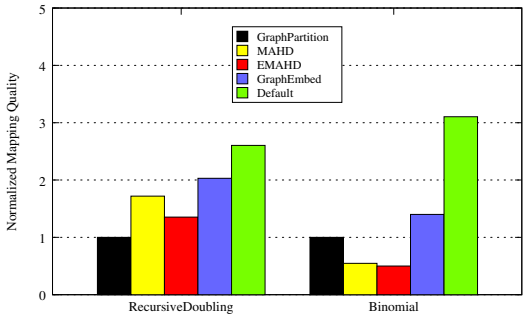

Figure 4.6: Comparison of heuristics on 1000 nodes (12,000 cores).

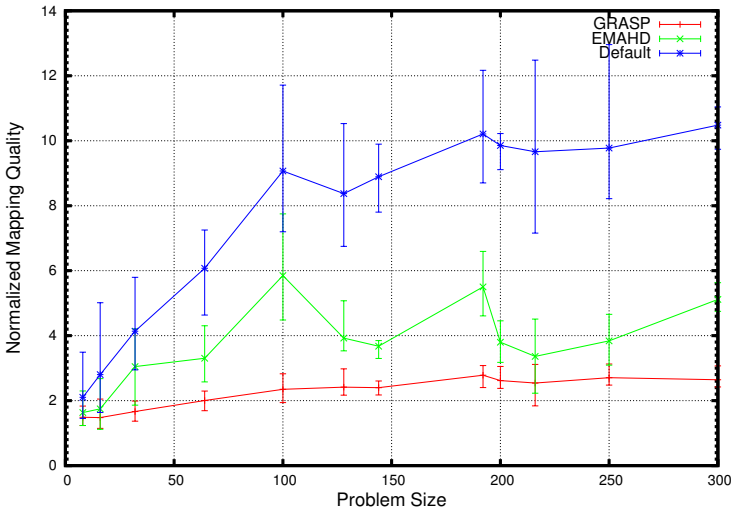

Figure 4.7: Quality of solution on the recursive doubling pattern compared with a lower bound.

the communication performance. (Related works mentioned in section 1.2, dealing with this metric, have provided further evidence in favor of this.) Preliminary studies with recursive doubling on the MPI_Allgather implementation with 1KB messages on problems with 128 nodes showed that GRASP and EMAHD are about 25% faster than the default and 20% faster than GGE. The improvement over the default is significant, though not as large as that indicated by the hop-byte metric, because that metric is only an indirect indication of the quality of the mapping. However, it does suggest that optimizing the hop-byte metric leads to

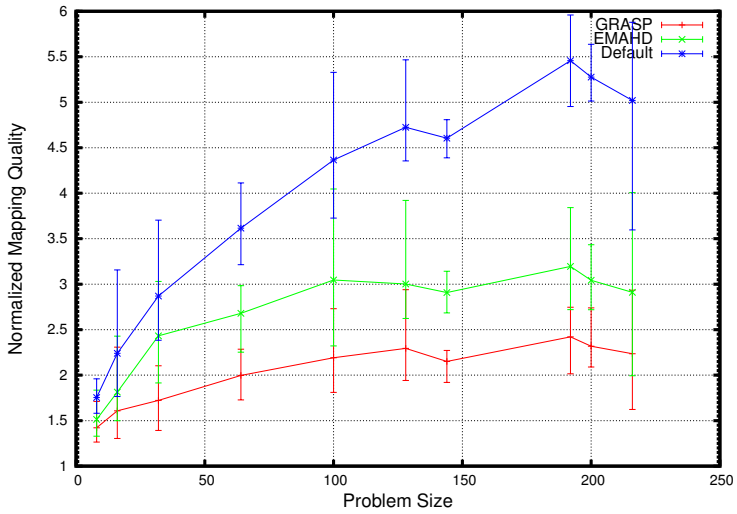

Figure 4.8: Quality of solution on the 3D spectral pattern compared with a lower bound.

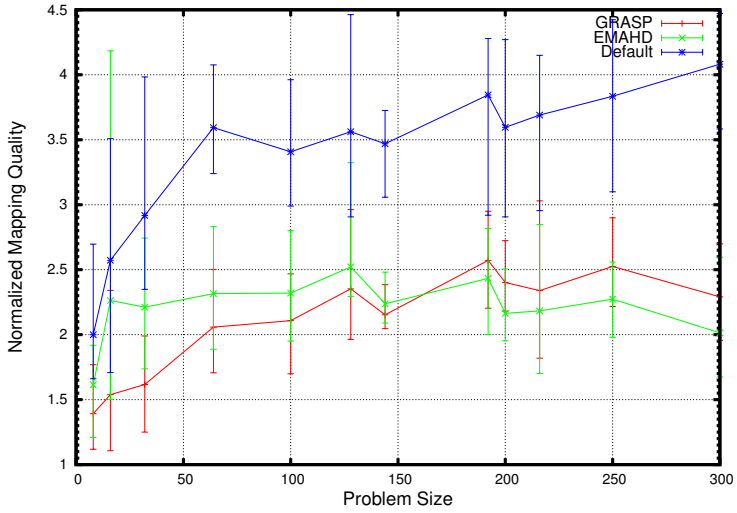

Figure 4.9: Quality of solution on the mesh pattern compared with a lower bound.

improved performance.

5. Conclusions

The research conducted as part of this thesis reinstated the importance of topology and routing aware mapping in scaling the application performance on current supercomputers. Our results show that in spite of worm-hole routing, link contention can severely affect message latencies. Other important observation is that the nodes allocated by the job scheduler do not correspond to any standard topology, even when the machine does. This arbitrariness of the node allocation makes the mapping more interesting.

We have used topology and routing aware mapping to improve the performance of communication in the load balancing of QMC application on Jaguar. We were able to reduce the communication time in the load balancing phase by 60% with 120,000 cores and 20% on 12,000 cores, and this mapping also reduced MPI_allgather time by a similar amount. The new dynamic load balancing algorithm developed can be used for computations with independent identical tasks, and the algorithm has some good theoretical properties. We have shown that it performs better than the current methods used in Quantum Monte Carlo codes in empirical tests.

In order to generalize the mapping techniques, we posed the mapping problem with the hop-byte metric as a quadratic assignment problem and used a heuristic to directly optimize for this metric. We have shown that using the GRASP heuristic leads to a better mapping than existing methods, which typically use some metric just to evaluate the heuristic, rather than to guide the optimization. We have evaluated the heuristic on realistic node allocations, which typically consist of many disjoint connected components. GRASP performs better than GGE, which is among the best prior heuristics that can be applied to arbitrary graphs with arbitrary communication patterns, and performs much better than the default mapping. In fact, GRASP is optimal for small problem sizes. Comparison with the lower bound suggests that GRASP may be close to optimal for medium problem sizes too. Since the above method do not scale, we proposed few alternatives such as

MAHD and a hybrid algorithm for larger problem sizes. We evaluated our approach on realistic node allocations obtained on the Kraken system at NICS. Our approach yields values for the metric that are up to 75% lower than the default mapping and 66% lower than existing heuristics.

Preliminary experiments on MPI also suggest that optimizing for the hop-byte metric improves the actual MPI collective communication latency, though not to the extent predicted by this metric. This is reasonable, because the metric does not directly account for the congestion bottleneck. One direction for future work is in reducing the time taken by the GRASP heuristic. GRASP is a general solution strategy, rather than a specific implementation. The particular implementation that we used is for a general QAP problem. We can develop an implementation specific to our mapping problem. For instance, solutions generated by the fast heuristics can be used as starting points in GRASP, thereby reducing the search space. Furthermore, we used a dense GRASP implementation because the node graph is complete. We can remove edges with heavy weights (corresponding to nodes that are far away) so that a sparse algorithm can be used. A different direction lies in optimizing for a different metric. The actual bottleneck is contention on specific links. We have posed the problem of minimizing the maximum contention as an integer programming problem, and will develop heuristics to solve it.

The use of on-chip interconnect on some of the current and future multi-core processors lead to the importance of mapping even with in a node. Using the Cell processor, we observed that the SPE-thread affinity has a significant effect on inter-SPE communication throughput. The performance with an optimal affinity is a factor of two over the performance with default assignment. We developed a tool that automatically determines the ideal mapping when given a communication pattern. Using this tool on a particle transport application showed a difference in total application performance of over 10% between the best and worst mappings.

Bibliography

[1] http://www.top500.org/

[2] H. Yu, I. Chung, and J. Moreira, Topology mapping for Blue Gene/L Supercomputer, ACM/IEEE conference on Supercomputing, SC06, 2006.

[3] T. Hoefler, T. Schneider, and A. Lumsdain, "Multistage Switches are not Crossbars: Effects of Static Routing in High-Performance Networks," IEEE International Conference on Cluster Computing, pages 116-125, 2008.

[4] J. L. Traff, Implementing the MPI Process Topology Mechanism, ACM/IEEE conference on Supercomputing, SC02, 2002.

[5] P. Altevogt, H. Boettiger, T. Kiss, and Z. Krnjajic. IBM blade center QS21 hardware performance. TR, WP101245, IBM, 2008.

[6] M. Kistler, M. Perrone, and F. Petrini. Cell multiprocessor communication network: Build for speed. IEEE Micro, 26:10 23, 2006.

[7] C.D. Sudheer, T. Nagaraju, P.K. Baruah, and A. Srinivasan, Optimizing Assignment of Threads to SPEs of the Cell BE Processor, 10th PDSEC, Proceedings of the 23rd IPDPS, IEEE, (2009).

[8] G.. Okten and A. Srinivasan. Parallel quasi-Monte Carlo methods on a heterogeneous cluster. In Proceedings of the MCQMC,Springer-Verlag.

[9] Pipeline Pattern, modified by Yunsup Lee, Ver 1.0 (March 11,2009), based on the pattern

[10] Y. F. Hu, R. J. Blake, and D. R. Emerson. An optimal migration algorithm for dynamic load balancing. *Concurrency: Practice and Experience*, 10: 467–483, 1998.

[11] R. A. Kronmal and A. V. Peterson. On the alias method for generating random variables from a discrete distribution. *The American Statistician*, 33: 214–218, 1979.

[12] A. J. Walker. An efficient method for generating discrete random variables with general distributions. *ACM Transactions on Mathematical Software*, 3: 253–256, 1977.

[13] Y. Li, P.M. Pardalos, and M.G.C. Resende, A Greedy Randomized Adaptive Search Procedure for the Quadratic Assignment Problem. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 16 (1994) 237-261.

[14] S. Arora, A. Frieze, and H. Kaplan, A New Rounding Procedure for the Assignment Problem with Applications to Dense Graph Arrangement Problems. Mathematical Programming, Vol. 92 (2002), 1-36.

[15] G. Bhanot, A. Gara, P. Heidelberger, E. Lawless, J.C. Sexton, and R. Walkup, Optimizing Task Layout on the Blue Gene/L Supercomputer. IBM Journal of Research and Development, Vol. 49 (2005) 489-500.

[16] P. Balaji, R. Gupta, A. Vishnu, and P. Beckman, Mapping Communication Layouts to Network Hardware Characteristics on Massive-Scale Blue Gene Systems. Comput. Sci. Res. Dev., Vol. 26 (2011) 247-256.

[17] A. Bhatele, L.V. Kale, and S. Kumar, Dynamic Topology Aware Load Balancing Algorithms for Molecular Dynamics Applications. In proceedings of ICS, 2009.

[18] A. Bhatele and L.V. Kale, Application-Specific Topology-Aware Mapping for Three Dimensional Topologies. In proceedings of IPDPS, 2008.

[19] A. Bhatele, G. Gupta, L. V. Kale, and I.-H. Chung, Automated Mapping of Regular Communication Graphs on Mesh Interconnects, in Proceedings of International Conference on High Performance Computing (HiPC), 2010.

[20] A. Bhatele and L.V. Kale, Heuristic-based techniques for mapping irregular communication graphs to mesh topologies, Proceedings of Workshop on Extreme Scale Computing Application Enablement - Modeling and Tools, 2011.

[21] H. Yu, I.-H. Chung, and J. Moreira. Topology mapping for Blue Gene/L supercomputer. In SC06, page 116, New York, NY, USA, 2006. ACM.

[22] T. Hoefler and M.Snir, Generic Topology Mapping Strategies for Large-scale Parallel Architectures, In proceedings of the 2011 ACM International Conference on Supercomputing (ICS 2011).

[23] K. Kandalla, H. Subramoni, A. Vishnu, and D.K. Panda, ”Designing Topology-Aware Collective Communication Algorithms for Large Scale Infiniband Clusters: Case Studies with Scatter and Gather”. The 10th Workshop on Communication Architecture for Clusters (CAC 10), held in conjunction with Int’l Parallel and Distributed Processing Symposium (IPDPS 2010).

[24] E. Zahavi, ”Fat-trees routing and node ordering providing contention free traffic for MPI global collectives”. Journal of Parallel and Distributed Computing, February 2012, ISSN 0743-7315, 10.1016/j.jpdc.2012.01.018.

[25] Shahid H. Bokhari, On the Mapping Problem, IEEE Trans. Computers, vol. 30, no. 3, pp. 207214, 1981.

[26] Pavan Balaji, Rinku Gupta, Abhinav Vishnu, Pete Beckman, ”Mapping communication layouts to network hardware characteristics on massive-

scale blue gene systems", Comput Sci Res Dev (2011) 26: 247256 DOI 10.1007/s00450-011-0168-y.

[27] P. Altevogt, H. Boettiger, T. Kiss, and Z. Krnjajic. IBM blade center QS21 hardware performance. Technical Report WP101245, IBM, 2008.

[28] M. Kistler, M. Perrone, and F. Petrini. Cell multiprocessor communication network: Built for speed. *IEEE Micro*, 26:10–23, 2006.

[29] T. Nagaraju, P.K. Baruah, and A. Srinivasan. Optimizing assignment of threads to SPEs on the Cell BE processor. Technical Report TR-080906, Computer Science, Florida State University, 2008.

[30] G. Okten and A. Srinivasan. Parallel quasi-Monte Carlo methods on a heterogeneous cluster. In *Proceedings of the Fourth International Conference on Monte Carlo and Quasi Monte Carlo (MCQMC)*. Springer-Verlag, 2000.

[31] M.K. Velamati, A. Kumar, N. Jayam, G. Senthilkumar, P.K. Baruah, S. Kapoor, R. Sharma, and A. Srinivasan. Optimization of collective communication in intra-Cell MPI. In *Proceedings of the 14th IEEE International Conference on High Performance Computing (HiPC)*, pages 488–499, 2007.

Outline of the thesis

1. Introduction
2. Optimizing Mapping on Multi-core
3. Application Specific Mapping
4. Generic Mapping on Massively Parallel Machines
5. Conclusions

http://dmacssite.github.com/Synopsis.pdf

Publications

1. "Optimizing assignment of threads to SPEs on the cell BE processor", *Sudheer C.D*, T. Nagaraju, P.K. Baruah, Ashok Srinivasan, ipdps, pp.1-8, 2009 IEEE International Symposium on Parallel & Distributed Processing, 2009 (PDSEC workshop)

2. "Investigating Algorithmic Techniques for Enhancing Application Performance on Multicore Processors" - *Sudheer C.D* (Advisor: Ashok Srinivasan), (PhD Forum at IPDPS-09)

3. " A communication model for determining optimal affinity on the Cell BE processor", *Sudheer C.D*, Sriram, S., P.K, B.: In: Proc. Int.l Conference on High Performance Computing (Student Research Symposium at HiPC-09, Dec 2009).

4. Dynamic load balancing for petascale quantum monte carlo applications: The alias method. *Sudheer C.D*, S. Krishnan, A. Srinivasan, and P. R. C. Kent. Submitted to Computer Physics Communications (2012).

5. "Optimization of the Hop-Byte Metric for Effective Topology Aware Mapping", *Sudheer C.D*, Ashok Srinivasan, (Accepted for HiPC 2012).

Presentations

1. "An Overview of the Global Arrays Toolkit", Five-days Technology Workshop on Heterogeneous Computing - Many Core/ Multi GPU - Performance of Algorithms, Application Kernels (HeMPa 2011) at CMSD, UoHYD by C-DAC Pune & CMSD.

2. Programming for Performance on the Cell BE processor, at the Performance Enhancement on Emerging Parallel Processing Platforms Workshop (PEEP-2008), jointly organized by C-DAC and IUCAA.

Professional Service

1. PC member for the track "Parallel/Multicore and Distributed Algorithms and Applications" high performance computing and communications (HPCC-11, HPCC-12)

2. PC member, The 12th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP-12)

Computer Access time Grants

1. XSEDE research allocation, "Scaling communication performance for massively parallel applications", 800,000 SUs, PI: Prof. Ashok Srinivasan, FSU.

2. XSEDE Education Allocation: "Programming for Performance on multicore and many-core processors", PI: Prof. Ravi Mukkamala, ODU.

3. Teragrid Startup Allocation, PI: Prof. P Sadayappan, OHU.

Teaching:

1. High Performance Computing with Accelerators, Summer 2012 http://dmacssite.github.com/.

2. Programming for performance, Winter 2011.

3. Computer Organization and Design, Summer 2012.