Performance Modeling of Hybrid MPI/OpenMP Scientific Applications on Large-scale Multicore Cluster Systems

Xingfu Wu and Valerie Taylor
Department of Computer Science and Engineering
Texas A&M University, College Station, TX 77843
{wuxf, taylor}@cse.tamu.edu

Abstract

In this paper, we present a performance modeling framework based on memory bandwidth contention time and a parameterized communication model to predict the performance of OpenMP, MPI and hybrid programs with weak scaling on three large-scale multicore clusters: IBM POWER4, POWER5+ and BlueGene/P, and analyze the performance of these MPI, OpenMP and hybrid programs. We use STREAM memory benchmarks to provide initial performance analysis and model validation of MPI and OpenMP programs on these multicore clusters because the measured sustained memory bandwidth can provide insight into the memory bandwidth that a system should sustain on scientific applications with the same amount of workload per core. In addition to using these benchmarks, we also use a weak-scaling hybrid MPI/OpenMP large-scale scientific application: Gyrokinetic Toroidal Code in magnetic fusion to validate our performance model of the hybrid program on these multicore clusters. The validation results for our performance modeling method show less than 7.77% error rate in predicting the performance of hybrid MPI/OpenMP GTC on up to 512 cores on these multicore clusters.

1. Introduction

Today, the trend in high performance computing systems has been shifting towards cluster systems with multicore. Further, Multicore processors are usually configured hierarchically (e.g., multiple multicores compose a multi-chip module to form a node) to form a compute node of parallel systems. While multicore presents significant new opportunities such as on-chip high inter-core bandwidth and low latency, it also presents new challenges in the form of inter-core resource conflict and contention. For many years, there has been considerable concern over the growing imbalance between memory subsystem performance and processor performance. Many of us fear that multicore continues to exacerbate memory bandwidth problems, however, the stall in processor frequency has diminished the corresponding worsening of memory latency in terms of processor-clocks [LL07].

Levesque et al. [LL07] observed from the AMD architectural discussion that when excluding messaging performance, the primary source of contention when moving from single core to dual core is memory bandwidth, and confirmed this assumption by the testing with STREAM and Membench microbenchmarks. In our previous work [WT09, WT09a], we also found that memory bandwidth contention is the primary source of performance degradation for L2 shared architectures such as CrayXT4 and IBM Power4 and Power5 systems using NAS Parallel benchmarks and large-scale scientific applications such as the Gyrokinetic Toroidal Code (GTC) [ES05] when increasing the number of cores per node. Therefore, in this paper, we focus on scientific applications with memory bandwidth contention problems, and present a performance modeling framework based on memory bandwidth contention time on small number of cores to predict the performance on larger number of cores.

Multicore clusters provide a natural programming paradigm for hybrid programs. Current hybrid parallel programming paradigms such as hybrid MPI/OpenMP need to efficiently exploit the potential offered by such multicore clusters. Generally, MPI is considered optimal for process-level coarse parallelism and OpenMP is optimal for loop-level fine grain parallelism. Combining MPI and OpenMP parallelization to construct a hybrid program is not only to achieve multiple levels of parallelism but also to reduce the communication overhead of MPI at the expense of introducing OpenMP overhead due to thread creation and increased memory bandwidth contention. In this paper, we analyze the performance of MPI, OpenMP and hybrid programs on three large-scale multicore clusters: IBM POWER4, POWER5 and BlueGene/P clusters, and present a performance modeling framework based on memory bandwidth contention time and parameterized communication model to predict the performance of OpenMP, MPI and hybrid programs with weak scaling on these multicore clusters.

The experiments conducted for this work utilize multicore clusters with different number of cores per node. P655 has 8 cores per node, Hydra [TSCF] has 16 cores per node, and BlueGene/P [ALCF] at Argonne National Lab (ANL) has 4 cores per node. Further, each system has a different node memory hierarchy. We use the MPI and OpenMP STREAM

memory benchmarks [McC] to provide initial performance analysis of MPI and OpenMP programs on these multicore clusters. In addition to using these benchmarks, we also use a hybrid MPI/OpenMP large-scale scientific application: a 3D particle-in-cell application GTC in magnetic fusion to validate our performance models of these MPI, OpenMP and hybrid programs. Our experimental results for our performance modeling mehtod show less than 7.77% error rate in predicting the performance of MPI and OpenMP GTC on up to 512 cores of these multicore clusters.

The remainder of this paper is organized as follows. Section 2 discusses the architecture and memory hierarchy of three large-scale multicore clusters used in our experiments, and compares their performance using STREAM benchmarks. Section 3 proposes and discusses a performance modeling framework for hybrid MPI/OpenMP programs based on memory bandwidth contention time and a parameterized communication model. Section 4 validates our performance models using the hybrid GTC. Section 5 discusses some related work and Section 6 concludes this paper.

In the remainder of this paper, we assume that the job scheduler for each multicore cluster always dispatches one process to one core or one thread to one core. We describe the system configuration as **MxN whereby M denotes the number of nodes with N cores per node.** All experiments were executed multiple times to ensure consistency of the performance data. Prophesy system [TW03] is used to collect all application performance data.

2. Execution Platforms and Performance

In this section, we briefly describe three large-scale multicore clusters used for our experiments, and use MPI and OpenMP STREAM memory benchmark [McC] to measure and compare sustainable memory bandwidth for MPI and OpenMP programs on these multicore clusters.

2.1 Descriptions of Execution Platforms

Details about three large-scale multicore clusters used for our experiments are given in Table 1. These clusters differ in the following main features: number of cores per node, configurations of node memory hierarchy, CPU speed, multicore processors, operating systems, and communication networks.

SDSC DataStar P655 [SDSC] is an IBM POWER4 cluster with 176 (8-way) compute nodes with 1.5GHz POWER4 and 16GB memory, and 96 (8-way) compute nodes with 1.7GHz POWER4 and 32GB memory. Each node of P655 has one MCM (multiple-chip module) with 4 chips per MCM. The use of 8-way nodes for P655 is exclusive.

Hydra at Texas A&M University Supercomputer Facility [TSCF] is an IBM POWER5+ cluster with 40 POWER5-575 nodes, and each node has 32 GB of memory and 8 DCMs (Dual-Chip Modules) with a dual-core POWER5+ processor per DCM.

Intrepid at Argonne Leadership Computing Facility [ALCF], Argonne National Lab, is an IBM BlueGene/P supercomputer with 40,960 quad-core nodes and 80 terabytes of memory. Its computer nodes are each connected to multiple inter-node networks, including a high-performance, low-latency 3D Torus, a highly scalable collective network, and a fast barrier network.

Table 1. Specifications of three multicore cluster architectures

Configurations	Hydra	P655	BlueGene/P
Total Cores	640	2,176	163,840
Total Nodes	40	272	40,960
Cores/chip	2	2	4
Cores / Node	16	8	4
CPU type	1.9GHz POWER5+	1.5, 1.7GHz POWER4	850MHz PowerPC
Memory/Node	32GB	16, 32GB	2GB
L1 Cache/CPU	64/32 KB	64/32 KB	32KB
L2 Cache/chip	1.92MB	1.41MB	16 128B lines
L3 Cache/chip	36MB	32MB	8MB
Network	Federation	Federation	3D Torus

Table 2. Sustainable Memory bandwidth on P655

Program Type	MPI				OpenMP
Configuration	**1x8**	**2x4**	**4x2**	**8x1**	**8 threads**
Memory bandwidth (MB/s)	16106.13	20132.66	26843.55	40265.32	18249.16

Table 3. Sustainable memory bandwidth on Hydra

Program Type	MPI					OpenMP
Configuration	**1x16**	**2x8**	**4x4**	**8x2**	**16x1**	**16 threads**
Memory bandwidth (MB/s)	32212.25	53687.09	80530.64	80530.64	80530.64	57063.45

Table 4. Sustainable Memory bandwidth on BlueGene/P

Program Type	MPI			OpenMP
Configuration	1x4	2x2	4x1	4 threads
Memory bandwidth (MB/s)	16106.13	16106.13	16106.13	8977.32

2.2 Sustainable Memory Bandwidth Comparison

In this section, we use the MPI and OpenMP versions of the STREAM memory benchmark [McC] to investigate memory performance for different system configurations. The STREAM benchmark is a synthetic benchmark program, written in standard Fortran 77 and MPI for MPI version and in C and OpenMP for OpenMP version. It measures the performance of four long vector operations (double precision): COPY (i.e., a(i)=b(i)), SCALE (i.e., a(i)=q*b(i)), SUM (i.e., a(i)=b(i)+c(i)), and TRIAD (i.e., a(i)=b(i)+q*c(i)), and it is specifically intended to eliminate the possibility of data re-use (either in registers or caches). The TRIAD allows chained/overlapped/fused multiple/add operations. In this paper, we only use unit-stride TRIAD benchmark to measure the sustainable memory bandwidth. We find that most multicore clusters we used only support array sizes of at most *4M* (2^{22}) with 8 bytes per double precision word because of lack of sufficient memory to start the benchmark. So we set the array size *4M* for MPI and OpenMP STREAM benchmarks.

Because P655 has 8 cores per node, so we use STREAM benchmarks to measure the sustainable bandwidths on 8 cores. For different configurations using processor partitioning [WT07, WT09a], Table 2 shows the sustainable memory bandwidth increases from 16106.13MB/s to 40265.32MB/s with decreasing the number of cores per node from 8 cores per node to 1 core per node for using 8 MPI processes because fewer MPI processes compete for memory. We see that the sustainable memory bandwidth for using 8 OpenMP threads is larger than that for using 8 MPI processes with the configuration 1x8, but smaller than the others.

Table 3 shows the memory bandwidths for three configurations 16x1, 8x2 and 4x4 are the same on Hydra. These memory bandwidths are more than two times larger than that for the configuration 1x16. We also see that the sustainable memory bandwidth for using 16 OpenMP threads is larger than that for using 16 MPI processes with the configuration 1x16. Table 4 indicates no difference in memory bandwidths for difference configurations on BlueGene/P, and the memory bandwidth for using 4 OpenMP threads is almost half less than that for using 4 MPI processes.

In summary, Tables 2-4 present the sustainable memory bandwidths for MPI and OpenMP on the three multicore clusters. For all these systems, different system configurations impact the sustainable memory bandwidth. Hence, using fewer cores per node results in better sustainable memory bandwidth, and except on BlueGene/P, using the maximum number of cores per node never resulted in the highest sustainable memory bandwidth.

3. Performance Models for Hybrid MPI/OpenMP Scientific Applications

In this section, we propose a performance modeling framework based on memory bandwidth contention time and parameterized communication model, and use the performance modeling framework to model and predict the performance of OpenMP, MPI and hybrid programs.

3.1 Performance Models for OpenMP Programs

Figure 2 illustrates a simple OpenMP multithread process, where the program consists of sequential components S1 and S2 and parallelized components P1, P2, P3 and P4. The OpenMP program proceeds in the fork-join-like model. First, it processes S1, then the master thread (the process itself) forks three threads. The four threads process P1, P2, P3 and P4 in parallel. When they are finished, they are joined to the master thread. Then the program processes S2. Note that there is some overhead in forking and joining OpenMP threads. Since all the OpenMP threads belong to a single process, they share the same address space and it is easy to reference data that other threads have updated.

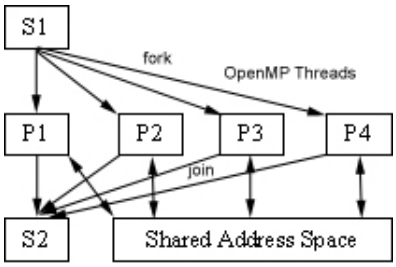


Figure 2. One multithreads process

As discussed above in Section 2.3, Tables 2-4 indicate that sustainable memory bandwidth decreases with increasing number of processors per node except on BlueGene/P. The experimental results also indicate that the application execution time decreases with increasing the number of processors per node [WT07]. In [WT09, WT09a], we found that memory bandwidth contention is the primary source of performance degradation for L2 shared architectures when increasing the number of processors per node. In [LL07], the primary source of contention when

moving from single core to dual core on AMD Opteron architecture is memory bandwidth, and a simple performance model for a sequential program was proposed to extrapolate the quad-core performance by assuming the time spent in the execution component remains the same, but the time spent in memory bandwidth contention will increase proportional to the reduction in effective memory bandwidth per core. In this section, we generalize the performance model to support parallel programs (OpenMP, MPI or hybrid) to extrapolate the performance of the multicore node (which consists of one or several multicores) by enumerating the following assumptions of our model:

i. The primary source of performance difference between single- and multi-core runs is memory bandwidth contention when excluding message passing performance.
ii. Computation time of a parallel program can be broken into the portion that is stalled on shared resources (memory bandwidth) and the portion that is stalled on non-shared resources.
iii. For an application with a fixed workload per processor core, assume that the time spent in the computation component per core (T_C) remains the same, however, the time spent in memory bandwidth contention (T_M) will increase proportional to the reduction in effective memory bandwidth per core.

The assumption (i) was confirmed in [LL07, WT09, WT09a], which means that our focus is on scientific applications with memory bandwidth contention problems. I/O performance is not considered in these kinds of applications. The assumption (ii) is consistent to that CPU execution time equals CPU time plus memory stall time defined by Hennessy and Patterson [HP03].

The assumption (iii) was confirmed in [WT09a, WT09], especially in [WT09a], when excluding message passing performance, the decrease percentage for the bandwidth of the data traffic between memory and D1 cache, between L2 and D1 cache or between L2 and memory is similar to the increase percentage for the total execution times of 8 NAS parallel benchmarks with classes B and C when using from 1 core per node, 2 cores per node to 4 cores per node, where the workload per core remains the same.

If the assumptions above hold true, we expect execution time to obey the following relationships:

$$\begin{aligned} T_1 &= T_C + T_M \\ T_2 &= T_C + \gamma_2 T_M \end{aligned} \quad (1)$$

Where T_1 and T_2 denote the execution time on one core and two cores, respectively; γ_2 denotes memory bandwidth ratio which is the memory bandwidth for the baseline (one core) divided by the memory bandwidth for the target (two cores) ($\gamma_2 > 1$ because of memory bandwidth contention. We use the STREAM memory benchmark to measure the sustained memory bandwidths on one or two cores to calculate the memory bandwidth ratio.) From Equation 1, we have

$$\begin{aligned} T_M &= \frac{T_2 - T_1}{\gamma_2 - 1} \\ T_C &= T_1 - \frac{T_2 - T_1}{\gamma_2 - 1} \end{aligned} \quad (2)$$

We can use Equation 2 to predict the execution time T_4 on four cores as follows. Assume that γ_4 denotes memory bandwidth ratio, which is the memory bandwidth for the baseline (one core) divided by the memory bandwidth for the target (four cores). From the assumption (iii), we have

$$T_4 = T_C + \gamma_4 T_M \quad (3)$$

From Equations 2 and 3, we have the execution time on four cores

$$T_4 = (T_1 - \frac{T_2 - T_1}{\gamma_2 - 1}) + \gamma_4 \frac{T_2 - T_1}{\gamma_2 - 1} = T_1 + (\gamma_4 - 1)\frac{T_2 - T_1}{\gamma_2 - 1} \quad (4)$$

This demonstrates how to use the performance model based on performance data on one core or two cores to predict the performance on four or more cores. Note that the assumption (iii) means that the performance model is only for weak scaling applications where the time spent in the computation component per core remains the same. The memory bandwidth ratios γ_2 and γ_4 for OpenMP programs are only measured once, then can be used for modeling any OpenMP programs on the system. We can generalize Equation 4 to a general case for using n cores. Assume that γ_n denotes memory bandwidth ratio, which is the memory bandwidth for the baseline (one core) divided by the memory bandwidth for the target (n cores). From Equation 4, we have the general model for the application execution time on n cores

$$T_n = T_1 + (\gamma_n - 1)\frac{T_2 - T_1}{\gamma_2 - 1}$$

This equation indicates that, given the fixed workload per core, the equation is used to predict the application performance on n cores based on the performance for single and dual cores and the memory bandwidth ratios for dual (γ_2) and many cores (γ_n), where the memory bandwidth ratios are associated with the sustained memory bandwidths on 1, 2 and n cores, which provide insight into the memory bandwidth that a system should sustain on various classes of scientific applications. Therefore, this performance model will be helpful to understand the application performance on multi- and many cores.

The performance modeling method for OpenMP programs can also be applied to computation components of a MPI program for the given problem size and number of cores. Given problem size and number of cores, using processor partitioning [WT07, WT09a], we measure the sustainable memory bandwidths for MPI STREAM memory

benchmark on three multicore clusters, P655, Hydra, and BlueGene/P as shown in Tables 2-4, then use them to calculate the memory bandwidth ratio γ_2 and γ_4 (the baseline bandwidth for using one core per node). Assume that T_1 and T_2 denote the computation times of a MPI program for using one core per node and two cores per node, respectively. Because of using processor partitioning for the given number of cores, the time spent in the computation component per core remains the same. Therefore, we can use Equation 4 to predict the computation time of the MPI program for using four cores per node. For communication component of the MPI program, we can parameterize the communication component based on each MPI subroutine to generate a parameterization model. We will discuss the communication parameterization model in detail in the following section (Section 3.2).

3.2 Performance Models for Hybrid MPI/OpenMP Programs

For hybrid MPI/OpenMP programs, modeling their performance is more complicated because of so many different combinations of MPI processes and OpenMP threads. In previous work [WT09], for the MPI GTC, we found that using fewer processors per node resulted in the better performance, and using one processor per node resulted in the best performance for the MPI version of GTC. In this section, for the sake of simplicity, we consider using one MPI process per node and one OpenMP thread per core on each node to model the performance of the hybrid programs executed on these multicore clusters.

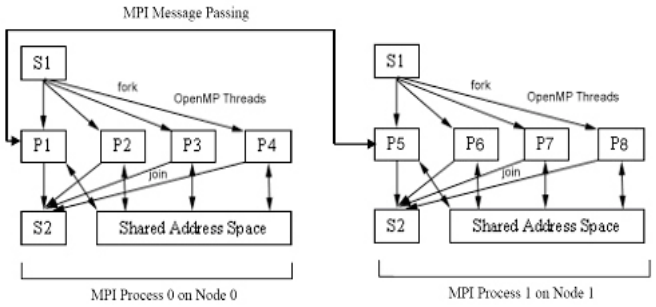


Figure 3. One multithreads process per node

Figure 3 illustrates one simple OpenMP-multithread MPI process per node. To utilize the shared memory feature of multicore nodes, we run one MPI process with OpenMP multithreads on each node so that intranode communication uses shared memory and internode communication uses message passing. This can take advantage of inter-core high bandwidth and low latency provided by multicore. As shown in Figure 3, the hybrid program is a SPMD program. The program execution consists of two MPI processes with four OpenMP threads per process. Modeling the performance of the hybrid program requires modeling its computation and communication. We can use the performance model for OpenMP programs discussed in Section 3.1 to model the OpenMP performance of the hybrid program. To model the communication of the hybrid program, we just consider internode MPI communication cost. The intranode communication cost (caused by OpenMP directives) is already included in the performance model for OpenMP programs.

For the given problem size, number of cores, and system, we define that the factor α for overlapping computation and communication for the hybrid program is

$$\alpha = \frac{TotalExecutionTime}{ComputationTime + CommunicationTime}$$

The overlapping factor α reflects the overlapping rate between the computation and communication within a hybrid program, and it is associated with the problem size, number of cores and system used. To compute the overlapping factor, we need three measurements for the total execution time, the computation time and communication time.

Assume that T_{omp}, T_{mpi} represent the performance models for intranode OpenMP performance and internode MPI communication cost, respectively. When we take the overlapping factor into account, we have the performance model T of the hybrid program

$$T = \alpha(T_{omp} + T_{mpi})$$

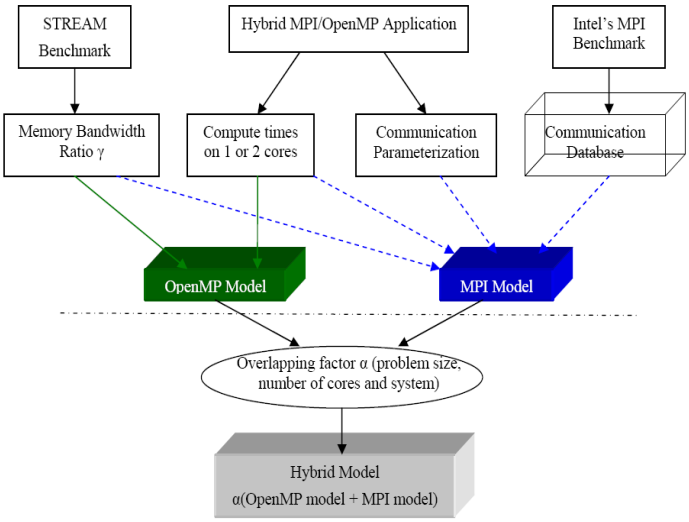


Figure 4. Framework for modeling hybrid MPI/OpenMP programs

Figure 4 present a framework for modeling hybrid MPI/OpenMP programs. Modeling MPI or OpenMP programs discussed in the previous section (Section 3.1) is considered as a special case of the framework above the horizontally dashed line shown in Figure 4. Modeling an

OpenMP-only program requires two components with solid arrows (green); modeling a MPI-only program needs four components with dashed arrows (blue). Note that, *To model a hybrid MPI/OpenMP program, MPI model in Figure 4 means MPI communication parameterization model (with communication database)*, because OpenMP model includes all computation components.For example, for the given hybrid MPI/OpenMP GTC shown in Section 4, because the workload per core remains the same (weak scaling), the model T_{omp} can be generated by using the modeling method from Section 3.1. Because there are few (only 12) MPI subroutines such as MPI_Sendrecv, MPI_Allreduce, MPI_Reduce, MPI_Allgather, MPI_Gather, MPI_Bcast, MPI_Send, MPI_Recv, MPI_Comm_size, MPI_Comm_rank, MPI_Init and MPI_Finalize in the GTC code (Note that this is a general case for a MPI program), we can parameterize the MPI communications based on these subroutines as follows. For a given number of cores, MPI_Init and MPI_Finalize can only be called once in a MPI program, so we can manually count how many calls and how big the message size for each MPI subroutine to form a communication parameterization model. Generally, this is a straightforward method to generate a parameterization model for the communication cost of a MPI application. We use the popular Intel's MPI benchmark [IMB] to measure the performance of each MPI subroutine with different message sizes on different number of cores on each multicore cluster, and store the performance data to the Communication Database shown in Figure 4. This kind of measurement just is done once for each cluster. Finally, we use the communication parameterization model to compute the total communication time based on the performance data from the Communication Database.

4. Case Study: a Hybrid MPI/OpenMP Scientific Application GTC

In this section, we use the hybrid GTC to validate our performance model on the multicore clusters.

4.1 Descriptions of Hybrid GTC

GTC [ES05] is a 3D particle-in-cell application developed at the Princeton Plasma Physics Laboratory to study turbulent transport in magnetic fusion. GTC is currently the flagship SciDAC fusion microturbulence code written in Frotran90, MPI and OpenMP. The test case for GTC studied in this paper is 100 particles per cell and 100 time steps.

4.2. Performance Modeling Validation

We start to use the scientific application GTC (MPI/OpenMP) to validate our performance model for MPI/OpenMP versions of the application on the multicore clusters, and use the performance model to calculate and compare the memory bandwidth contention time for MPI/OpenMP programs. Note that, the workload per processor (or core) remains the same for GTC.

Table 9. Predicted and Actual Performance of OpenMP GTC on P655

Metrics	**2 threads**	**4 threads**	**8 threads**
Actual Execution Time (s)	1103.37	1202.70	1246.04
Memory Bandwidth Ratio γ	1 (baseline)	1.75	2.29
Predicted Execution Time (s)	baseline	baseline	1274.22
Predicted Error Rate	--	--	2.26%
ctime (s)	970.93		
mtime (s)	132.44		

Table 10. Predicted and Actual Performance of OpenMP GTC on Hydra

Metrics	**2 threads**	**4 threads**	**8 threads**	**16 threads**
Actual Execution Time (s)	917.91	980.9	1022.83	1153.07
Memory Bandwidth Ratio γ	1 (baseline)	3.41	7.52	9.21
Predicted Execution Time (s)	baseline	baseline	1088.34	1132.52
Predicted Error Rate	--	--	6.40%	1.70%
ctime (s)	891.77			
mtime (s)	26.14			

Table 11. Predicted and Actual Performance of OpenMP GTC on BlueGene/P

Metrics	2 threads	4 threads
Actual Execution Time (s)	3279.74	3631.99
Memory Bandwidth Ratio γ	1 (baseline)	1.98
Predicted Execution Time (s)	baseline	baseline
Predicted Error Rate	--	--
ctime (s)	2920.30	
mtime (s)	359.44	

4.2.1 OpenMP Programs

To illustrate how to use the performance model in Section 3.1, we use OpenMP GTC to test our performance model on Hydra, P655 and BlueGene/P. We use P655 as an example to illustrate how to generate a performance model in detail as follows:

We use OpenMP STREAM memory benchmark to measure sustainable memory bandwidths for 2 to 8 threads on a compute node on P655, and use the memory bandwidth for 2 threads as a baseline to calculate the bandwidth ratio γ shown in Table 9. We assume the time spent in the computation component per core is T_C, and the time spent in memory bandwidth contention is T_M for 2 threads (Note that comparing to T_M, the overhead for forking and joining the threads is very small.) Then, we have

$$T_C + T_M = 1103.37 \quad (5)$$

Because the workload per core remains the same for GTC, like Equation 3, we have the following equation for 4 threads

$$T_C + 1.75 * T_M = 1202.70 \quad (6)$$

From Equations 5 and 6, we have

$$T_C = 132.44 \text{ and } T_M = 970.93$$

From Equation 6, we can predict the performance for 8 threads as follows:

$$T_C + 2.29 * T_M = 970.93 + 2.29 * 132.44 = 1274.22 \text{ (s)}$$

As shown in Table 9, the predicted error rate is only 2.26% for 8 threads. The same method is also applied to other multicore clusters shown in Tables 10 and 11. Table 10 shows that the predicted error rate is less than 6.5% for predicting the performance on 8 and 16 threads. It is interesting to see that Hydra has the smallest memory bandwidth contention time for OpenMP GTC.

4.2.2 MPI Programs

Similarly, in this section, we use MPI GTC to test our performance model on Hydra, P655 and BlueGene/P. For P655, Table 2 shows the sustainable memory bandwidths using MPI STREAM benchmark. We use the memory bandwidth for 8 nodes with 1 core per node as a baseline to calculate the bandwidth ratio *r* shown in Table 12.

As shown in Table 12, the predicted error rate is only 1.05% for the 2x4. We can also use the performance model to predict the performance for the 1x8 is 1132.38 seconds, and its predicted error rate is just 1.99%. The same method is applied to other multicore clusters as showed in Tables 13-14. The memory bandwidth ratio *r* in Tables 13-14 are calculated from Tables 3-4 respectively.

Table 13 shows that the predicted error rate is 4.7% for predicting the performance for the 1x16. The difficulty for our model to deal with is the same memory bandwidth for the 16x1, 8x2 and 4x4 although their execution times are a little bit different. We just pick the execution time for the 16x1 as a baseline. This case also happens on BlueGene/P shown in Table 14. Thus, we will need to consider other factors for our performance model to deal with this case in the future work.

Compare Table 12 with Table 9, on P655, we find that the baseline memory bandwidth contention time for MPI GTC is almost six times smaller than that for OpenMP GTC because of OpenMP overhead due to thread creation and increased memory bandwidth contention, however, the pure CPU computation time for MPI GTC is larger than that for OpenMP GTC. Compare Table 13 with Table 10, on Hydra, the baseline memory bandwidth contention time for MPI GTC is over two times larger than that for OpenMP GTC. This is consistent with Table 3 where the sustainable memory bandwidth for using 16 OpenMP threads is larger than that for any configurations for using 16 MPI processes.

Table 12. Predicted and Actual Performance on 8 processors on P655

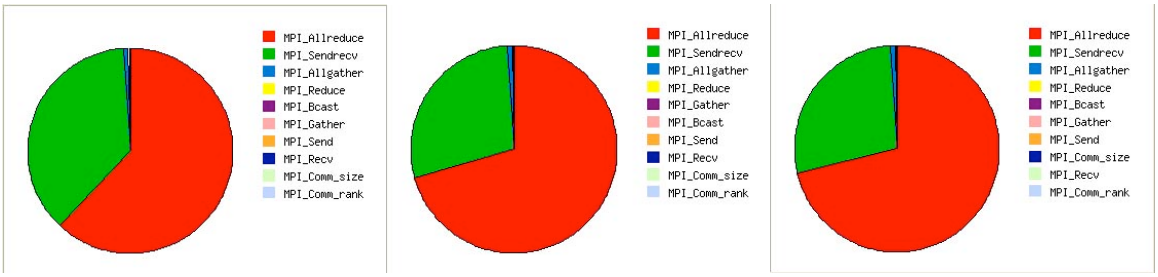
Configuration	1x8	2x4	4x2	8x1
Actual Computation Time (s)	1155.38	1133.15	1110.18	1099.08
Memory Bandwidth Ratio *r*	2.50	2.00	1.50	1(baseline)
Predicted Computation Time (s)	1132.38	1121.28	baseline	baseline
Predicted Error Rate	1.99%	1.05%	--	--
ctime (s)	1076.88			
mtime (s)	22.2			

Table 13. Predicted and Actual Performance on 16 processors on Hydra

Configuration	1x16	2x8	4x4	8x2	16x1
Actual Computation Time (s)	981.62	967.99	944.80	940.02	938.10
Memory Bandwidth Ratio *r*	2.5	1.5	1	1	1 (baseline)
Predicted Computation Time (s)	1027.77	baseline	--	--	baseline
Predicted Error Rate	4.70%	--	--	--	--
ctime (s)	878.32				
mtime (s)	59.78				

Table 14. Predicted and Actual Performance on 4 processors on BlueGene/P

Configuration	1x4	2x2	4x1
Actual Execution Time (s)	3237.26	3237.29	3237.33
Memory Bandwidth Ratio *r*	1	1	1(baseline)



(a) 16 MPI processes (b) 32 MPI processes (c) 64 MPI processes

Figure 6. Communication percentage of each MPI subroutine for hybrid GTC on P655

4.2.3 Hybrid MPI/OpenMP Programs

In this section, we apply our performance modeling framework shown in Figure 4 to model the performance of the hybrid MPI/OpenMP GTC on the three multicore clusters.

In Section 4.2.1, we discussed the OpenMP models for the OpenMP GTC on the three clusters. We use the OpenMP models to model the performance of the hybrid GTC because the hybrid GTC is a weak scaling application. Now we mainly focus on generating a parameterized communication model for the hybrid GTC. As we discussed in Section 3.2, there are few (only 12) MPI subroutines such as MPI_Sendrecv, MPI_Allreduce, MPI_Reduce, MPI_Allgather, MPI_Gather, MPI_Bcast, MPI_Send, MPI_Recv, MPI_Comm_size, MPI_Comm_rank, MPI_Init and MPI_Finalize in the GTC code, we can parameterize the MPI communications based on these subroutines to build the parameterized communication model for the application. We use IPM [IPM] to collect MPI communication performance and its profiling (message size and number of calls). We observe that the MPI subroutines MPI_Allreduce, MPI_Sendrecv and MPI_Allgather dominate more than 98%

of the total communication time shown in Figure 6, which shows the communication percentage of each MPI subroutine for the hybrid GTC on P655. Note that there is one MPI process per node and 8 OpenMP threads per node for the execution of the hybrid program because P655 has 8 processors per node. Because MPI_Allreduce, MPI_Sendrecv and MPI_Allgather dominate the communication time, we focus on the three MPI subroutines in our discussion.

Table 15. MPI profiling of GTC for 16, 32, and 64 MPI processes

16 processes	Message Size (bytes)	Number of Calls
MPI_Allreduce	4	3200
MPI_Allreduce	364	3600
MPI_Allreduce	1168164	3200
MPI_Allreduce	20	1600
MPI_Sendrecv	129796	28800
MPI_Sendrecv	8	6400
MPI_Allgather	519184	3200
32 processes	**Message Size (bytes)**	**Number of Calls**
MPI_Allreduce	4	6400
MPI_Allreduce	364	7200
MPI_Allreduce	1168164	6400
MPI_Allreduce	20	3200
MPI_Sendrecv	129796	57600
MPI_Sendrecv	8	12800
MPI_Allgather	259592	6400
64 processes	**Message Size (bytes)**	**Number of Calls**
MPI_Allreduce	4	12800
MPI_Allreduce	364	14400
MPI_Allreduce	1168164	12800
MPI_Allreduce	20	6400
MPI_Sendrecv	129796	115200
MPI_Sendrecv	8	25600
MPI_Allgather	129796	12800

Table 15 shows the main MPI profilings of the hybrid GTC for 16, 32 and 64 MPI processes. MPI_Allreduce with four different message sizes is performed for each program execution, however, the number of calls for MPI_Allreduce is double with doubling the number of MPI processes. Similarly, MPI_Sendrecv with four different message sizes is performed for each program execution, however, the number of calls for MPI_Sendrecv is doubled with doubling the number of MPI processes. For MPI_Allgather, the message size decreases by half and the number of calls is doubled with doubling the number of MPI processes because of the fixed total message size (array size). Based on the communication characteristics of the hybrid GTC shown in Table 15, we can manually parameterize the communication time for each MPI subroutine by the product of the time for each MPI subroutine with the given message size and number of processors and the number of calls. Now, we have the parameterized communication model.

For internode communication, we use Intel MPI benchmarks to measure the performance for each MPI subroutine with different message sizes on different number of cores, then store them to the Communication database. For each MPI subroutine, we query the Communication database to get the communication time for a message size on a gived number of nodes, then use the product of the time and the number of calls for the subroutine to calculate its total communication time. Based on the performance data for IMB from the Communication database, we can use the parameterized communication model to calculate the total communication time.

Because of the use of blocking communications in the whole GTC, we observe that the overlapping factor for computation and communication is almost 1, so we use the factor to generate our performance model for GTC. In the following, we present the experimental results for our performance models to predict the performance of the hybrid GTC on P655, Hydra and BlueGene/P.

Table 16 shows the predicted performance of hybrid GTC using our performance model on P655. Because the workload per core remains the same for GTC (weak scaling), we use the OpenMP model for OpenMP GTC on 8 cores as baseline, then sum the baseline OpenMP performance and total MPI communication time to predict the performance of the hybrid GTC. The error rate is less than 5.58%.

Table 17 shows the predicted performance of hybrid GTC using our performance model on Hydra. We use the OpenMP model for OpenMP GTC on 16 cores as baseline, then sum the baseline OpenMP performance and total MPI communication time to predict the performance of the hybrid GTC. The error rate is less than 7.77%.

Table 16. Predicted performance of hybrid GTC on P655

#Cores	8	16	32	64	128	256	512
Actual Runtime (s)	1246.04	1306.89	1363.96	1370.24	1388.23	1347.04	1424.53
Prediction (s)	1274.22	1280.15	1288.00	1301.71	1332.41	1397.19	1461.03
Error (%)	2.26%	2.05%	5.57%	5.00%	4.02%	3.72%	2.56%

Table 17. Predicted performance of hybrid GTC on Hydra

#cores	16	32	64	128	256	512
Actual Runtime (s)	1153.07	1200.66	1239.26	1225.91	1210.45	1190.5
Prediction (s)	1132.52	1137.27	1143.07	1154.66	1175.10	1230.03
Error (%)	1.70%	5.28%	7.76%	5.81%	2.92%	3.32%

Table 18. Predicted performance of hybrid GTC on BlueGene/P

#Cores	4	8	16	32	64	128	256	512
Actual Runtime (s)	3631.99	3681.05	3716.67	3738.52	3771.67	3761.81	3741.22	3742.49
Prediction (s)	--	3644.98	3659.62	3685.78	3707.14	3794.42	3927.34	3906.22
Error (%)	--	0.98%	1.54%	1.41%	1.71%	0.87%	4.97%	4.37%

Table 18 shows the predicted performance of hybrid GTC using our performance model on BlueGene/P. We use the OpenMP model for OpenMP GTC on 4 cores as baseline, then sum the baseline OpenMP performance and total MPI communication time to predict the performance of the hybrid GTC. The error rate is less than 4.98%.

In summary, we apply our performance modeling framework for weak-scaling hybrid programs to model and predict the performance of the hybrid GTC on up to 512 cores on P655, Hydra and BlueGene/P with less than 7.77% error rate. This indicates that our modeling method is practical and accurate.

6. Related Work

Levesque et al [LL07] observed that the primary source of contention when moving from single core to dual core on AMD Opteron architecture is memory bandwidth, and proposed a simple performance model for sequential programs to extrapolate the quad-core performance by assuming the time spent in the execution component remains the same, but the time spent in memory bandwidth contention will increase proportional to the reduction in effective memory bandwidth per core. In this paper, we generalize the performance model for OpenMP and hybrid parallel programs to extrapolate the performance of the multicore node (which consists of one or several multicores).

Barker et al [BD09] discussed using performance modeling to design large scale systems throughout their life cycle from early design through production use, presented an application-centric performance model development framework, and addressed the importance of performance modeling. Our performance model has more focus on multicore.

Adhianto and Chapman [AC07] proposed a cost efficient approach to model and evaluate parallel OpenMP, MPI and hybrid MPI + OpenMP with reasonable accuracy based on a combination of static analysis (using the OpenUH compiler) and feedback from runtime benchmarks (Sphinx and Perfsuite) for both communication and multithreading efficiency measurement, and predicted the performance of a parallel matrix multiply using Cannon algorithm.

Aversa et al. [AM05] described simulation-based techniques for performance prediction of hybrid MPI/OpenMP code in different working conditions using HeSSE (Heterogeneous System Simulation Environment), and predicted the performance of a parallel N-body code on a SMP cluster.

There are many approaches on modeling MPI communication cost, such as LogP [CK96], LogGP[AI97], and so on, we just take a straightforward empirical method to manually parameterize the communication time for each MPI subroutine by the product of the time for each MPI subroutine with the given message size and number of processors and the number of calls on the given system to generate a parameterized communication model, then based on the performance data for each MPI subroutine stored in the Communication database, we can calculate the total communication time on the system.

7. Conclusions

In this paper, we analyzed and compared the performance of MPI, OpenMP and hybrid programs on three large-scale multicore clusters, IBM POWER4, POWER5+ and BlueGene/P clusters using MPI and OpenMP STREAM benchmarks and the hybrid MPI/OpenMP GTC, and presented a practical performance modeling framework for weak-scaling hybrid MPI/OpenMP programs based on memory bandwidth contention time and parameterized communication model to model and predict the performance of hybrid and OpenMP GTC on these multicore clusters. The experimental results for our performance modeling mehtod showed less than 7.77% error rate in predicting the performance of hybrid and OpenMP GTC on up to 512 cores on the three multicore clusters. We also used the performance model to calculate and compare the memory bandwidth contention times for MPI and OpenMP GTC.

For weak-scaling scientific applications, the measured sustained memory bandwidth can provide insight into the memory bandwidth that a system should sustain on these scientific applications with the same amount of workload per core. However, for strong scaling scientific applications, it is hard to measure the sustained memory bandwidth for the different amount of decreased workload per core. This is a limitation for our performance modeling framework. We believe that our performance modeling framework can be applied to any weak-scaling hybrid MPI/OpenMP programs with memory bandwidth contention problems. Our future work will focus on exploring performance modeling for strong scaling scientific applications, and investigating performance-power trade-off models in our MuMI project [MuMI].

8. Acknowledgements

This work is supported by NSF grant CNS-0911023 and the Award No. KUS-I1-010-01 made by King Abdullah University of Science and Technology (KAUST). The authors would like to acknowledge Argonne Leadership Computing Facility for the use of BlueGene/P under DOE INCITE project "Performance Evaluation and Analysis Consortium End Station", the SDSC for the use of DataStar P655 under TeraGrid project TG-ASC040031, and TAMU Supercomputing Facilities for the use of Hydra. We would also like to thank Stephane Ethier from Princeton Plasma Physics Laboratory and Shirley Moore from University of Tennessee for providing the GTC code.

References

[AC07] L. Adhianto and B. Chapman, Performance Modeling of Communication and Computation in Hybrid MPI and OpenMP Applications, *Simulation Modeling Practice and Theory*, Vol 15, 2007.

[AI97] A. Alexandrov, M. Ionescu, K. Schauser, and C. Scheiman, LogGP: Incorporating Long Messages into the LogP Model for Parallel Computation, *Journal of Parallel and Distributed Computing*, 44(1), 1997.

[ALCF] Argonne Leadership Computing Facility BlueGene/P (Intrepid), http://www.alcf.anl.gov/resources.

[AM05] R. Aversa, B. Martino, M. Rak, S. Venticinque, and U. Villano, Performance Prediction Through Simulation of a Hybrid MPI/OpenMP Application, *Parallel Computing*, Vol. 31, 2005.

[BD09] K. Barker, K. Davis, A. Hoisie, D. Kerbyson, M. Lang, S. Pakin, and J. C. Sancho, Using Performance Modeling to Design Large-scale System, *IEEE Computer*, Nov. 2009

[CK96] D. Culler, R. Karp, D. Patterson, A. Sahay, E. Santos, K. Schauser, R. Subramonian, and T. Eicken, LogP: A Practical Model of Parallel Computation, *Communication of the ACM*, Vol. 39, No. 11, 1996.

[ES05] S. Ethier, First Experience on BlueGene/L, *BlueGene Applications Workshop*, ANL, April 27-28, 2005. http://www.bgl.mcs.anl.gov/Papers/GTC_BGL_20050520 .pdf.

[HP03] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, 2003.

[IMB] Intel MPI Benchmarks, Users Guide and Methodolgy Description (Version 2.3), http://www.intel.com/cd/ software/products/asmona/eng/cluster/mpi/219848.htm.

[IPM] IPM: Integrated Performance Monitoring, http://www.sdsd. edu/ us/tools/ top/ipm/.

[JJ03] G. Jost, H. Jin, D. Mey, and F. Hatay, Comparing the OpenMP, MPI, and Hybrid Programming Paradigms on an SMP Cluster, the *Fifth European Workshop on OpenMP (EWOMP03),* September 2003.

[LL07] J. Levesque, J. Larkin, M. Foster, J. Glenski, G. Geissler, S. Whalen, B. Waldecker, J. Carter, D. Skinner, H. He, H. Wasserman, J. Shalf, H. Shan, and E. Strohmaier, *Understanding and Mitigating Multicore Performance Issues on the AMD Opteron Architecture*, LBNL-62500, March 7, 2007

[McC] John D. McCalpin, STREAM: Sustainable Memory Bandwidth in High Performance Computers, http://www.cs. virginia.edu/stream.

[MuMI] MuMI project, Multicore application Modeling Infrastructure (MuMI), http://www.mumi-tool.org.

[OH07] K. Olukotun, L. Hammond, and J. Laudon, Chip *Multiprocessor Architecture: Techniques to Improve Throughput and Latency*, Morgan & Claypool Pub., 2007.

[SDSC] SDSC DataStar, http://www.sdsc.edu/user_services /datastar/.

[TSCF] Texas A&M University Supercomputer Facility Hydra, http://sc.tamu.edu/systems/hydra.

[TW03] Valerie Taylor, Xingfu Wu, and Rick Stevens, Prophesy: An Infrastructure for Performance Analysis and Modeling System of Parallel and Grid Applications, *ACM SIGMETRICS Performance Evaluation Review*, Volume 30, Issue 4, 2003.

[WT07] Xingfu Wu and Valerie Taylor, Processor Partitioning: An Experimental Performance Analysis of Parallel Applications on SMP Cluster Systems, *the 19th International Conference on Parallel and Distributed Computing and Systems (PDCS 2007)*, Nov. 19-21, 2007.

[WT09a] Xingfu Wu and Valerie Taylor, Using Processor Partitioning to Evaluate the Performance of MPI, OpenMP and Hybrid Parallel Applications on Dual- and Quad-core Cray XT4 Systems, *the 51st Cray User Group Conference (CUG2009)*, Atlanta, May 4-7 2009.

[WT09] Xingfu Wu, Valerie Taylor, Charles Lively, and Sameh Sharkawi, Performance Analysis and Optimization of Parallel Scientific Applications on CMP Cluster Systems, *Scalable Computing: Practice and Experience*, Vol. 10, No. 1, 2009.