Using Processor Partitioning to Evaluate the Performance of MPI, OpenMP and Hybrid Parallel Applications on Dual- and Quad-core Cray XT4 Systems

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

Abstract: *Chip multiprocessors (CMP) are widely used for high performance computing. While this presents significant new opportunities, such as on-chip high inter-core bandwidth and low inter-core latency, it also presents new challenges in the form of inter-core resource conflict and contention. A challenge to be addressed is how well current parallel programming paradigms, such as MPI, OpenMP and hybrid, exploit the potential offered by such CMP clusters for scientific applications. In this paper, we use processor partitioning as a term about how many cores per node to use for application execution to analyze and compare the performance of MPI, OpenMP and hybrid parallel applications on two dual- and quad-core Cray XT4 systems, Jaguar with quad-core at Oak Ridge National Laboratory (ORNL) and Franklin with dual-core at the DOE National Energy Research Scientific Computing Center (NERSC). We conduct detailed performance experiments to identify the major application characteristics that affect processor partitioning. The experimental results indicate that processor partitioning can have a significant impact on performance of a parallel scientific application as determined by its communication and memory requirements. We also use the STREAM memory benchmarks and Intel's MPI benchmarks to explore the performance impact of different application characteristics. The results are then utilized to explain the performance results of processor partitioning using NAS Parallel Benchmarks. In addition to using these benchmarks, we also use a flagship SciDAC fusion microturbulence code (hybrid MPI/OpenMP): a 3D particle-in-cell application Gyrokinetic Toroidal Code (GTC) in magnetic fusion to analyze and compare the performance of these MPI and hybrid programs on the dual- and quad-core Cray XT4 systems, and study their scalability on up to 8192 cores. Based on the performance of GTC on up to 8192 cores, we use Prophesy system to online generate its performance models to predict its performance on more than 10,000 cores on the two Cray XT4 systems.*

Keywords*:* Performance Evaluation, processor partitioning, MPI, Hybrid MPI/OpenMP, Cray XT4, performance modeling

1. Introduction

Today, the trend in high performance computing systems has been shifting towards cluster systems with CMPs. Further, CMPs are usually configured hierarchically to form a compute node of cluster systems. For instance, Franklin [NERS] at National Energy Research Scientific Computing Center (NERSC) is a dual-core Cray XT4 system, and Jaguar [NCCS] at Oak Ridge National Lab (ORNL) is a quad-core Cray XT4 system. Such systems have two or more processor cores per node. When using the CMP clusters to execute a given application (MPI or hybrid), one issue to be addressed is how many processor cores per node to use for efficient execution. It is expected that the best number is dependent upon application characteristics and system configuration. In this paper, we define processor partitioning as a term about how many processors per node to use for application execution, and conduct detailed performance experiments to identify the major application characteristics that affect processor partitioning. We also use processor partitioning to systematically analyze and compare the performance of MPI, OpenMP and hybrid programs on two Cray XT4 clusters.

The experiments conducted for this work utilize CMP clusters with different number of cores per node. Franklin has 2 cores per node, and Jaguar has 4 cores per node. Further, each system has a different node memory hierarchy. We use Intel's MPI benchmarks, IMB [IMB] and the MPI and OpenMP STREAM memory benchmarks [McC] to provide initial performance analysis of MPI and OpenMP programs on the two CMP clusters. The standard MPI and OpenMP programming paradigms of NAS Parallel Benchmarks (NPB) Version

3.2.1 [NPB3] make it possible for us to use these benchmarks as a basis for a systematically comparative performance analysis of MPI and OpenMP programs on the CMP clusters. In addition to using these benchmarks, we also use a large-scale scientific application: a 3D particle-in-cell application Gyrokinetic Toroidal Code (**GTC**) in magnetic fusion [ES05] to analyze and compare the performance of these MPI and hybrid programs.

Our experimental results indicate that, on the two CMP clusters, using processor partitioning changes the memory access pattern and communication pattern of a MPI program. Processor partitioning has significant performance impact of a MPI program except embarrassingly parallel applications such as EP. The memory bandwidth per core is the primary source of performance degradation when increasing the number of cores per node using processor partitioning. The results for hybrid GTC executed on up to 8192 cores indicates that MPI versions of these programs outperform their OpenMP and hybrid counterparts in most cases, but hybrid GTC scales better than its MPI counterpart and outperforms its MPI counterpart on 4096 and 8192 cores on Franklin.

The remainder of this paper is organized as follows. Section 2 discusses the architectures of two large-scale CMP clusters used in our experiments, and compares their performance using IMB and STREAM benchmarks. Section 3 analyzes and compares performance of MPI and OpenMP programs using NPB 3.2.1. Section 4 investigates performance and scalability of hybrid MPI/OpenMP programs on these CMP clusters, and use processor partitioning to analyze and compare the performance of the MPI GTC and its components. Section 5 uses Prophesy system [TW03] to online generate performance models for GTC and predicts its performance on 10,240 cores. Section 6 discusses some related work and concludes this paper in Section 7.

In the remainder of this paper, we assume that the job scheduler for each CMP cluster always dispatches one process to one core or one thread to one core. We describe **a processor partitioning scheme as NxM whereby N denotes the number of nodes with M cores per node.** All experiments were executed multiple times to insure consistency of the performance data. The unit for an execution time is seconds.

2. Execution Platforms and Performance

In this section, we briefly describe two large-scale CMP clusters used for our experiments, use Intel' MPI Benchmark Ping Pong [IMB] to measure and compare MPI bandwidth and latency on these CMP clusters, and use MPI and OpenMP STREAM memory benchmark [McC] to measure and compare sustainable memory bandwidth for MPI and OpenMP programs on these CMP clusters.

2.1 Descriptions of Execution Platforms

Details about the two large-scale CMP clusters used for our experiments are given in Table 1. These systems differ in the following main features: number of processors per node, configurations of node memory hierarchy, CPU speed and multi-core processors.

Table 1. Specifications of two dual- and quad-core Cray XT4 architectures

Configurations	Franklin	Jaguar
Total Cores	19,320	31,328
Total Nodes	9,660	7,832
Cores/chip	2	4
Cores / Node	2	4
CPU type	2.6 GHz Opteron	2.1 GHz Opteron
Memory/Node	4GB	8GB
L1 Cache/CPU	64/64 KB	64/64 KB
L2 Cache/chip	1MB	2MB
Network	3D-Torus	3D-Torus

NERSC Franklin [NERS] is a dual-core Cray XT4 system with 9,660 compute nodes. Each node has dual-core AMD Opteron processors and 4 GB of memory, and is connected to a dedicated SeaStar2 router through Hypertransport with a 3D Torus topology. The L2 cache is a victim cache which holds only the cache lines evicted from L1, whereas most data loaded from memory go directly to L1.

National Center for Computational Sciences Jaguar [NCCS] at ORNL is a quad-core Cray XT4 system with 7,832 compute nodes. Each node has quad-core AMD Opteron processors and 8 GB of memory, and is connected to a dedicated SeaStar router through Hypertransport with a 3D Torus topology.

2.2 MPI Bandwidth and Latency Comparison

In this section, we use Intel's MPI benchmark PingPong [IMB] to measure uni-directional point-to-point performance between two processes within the same node (one process per processor, the processor partitioning scheme 1x2) for intra-node performance, and between two nodes (one process per node, the processor partitioning scheme 2x1) for inter-node performance on Franklin and Jaguar.

The results shown in Figure 1 indicate almost doubling of the latency and a significant reduction in bandwidth when going from communication within a node to communication between nodes using PingPong on Franklin and Jaguar. Jaguar has much lower inter-core latency and higher inter-core bandwidth than that for Franklin. Both systems have similar inter-node latency and bandwidth for message sizes of less than 128KB. Jaguar has a little bit lower inter-core latency and higher inter-core bandwidth than that for Franklin for message sizes of 128KB or more.

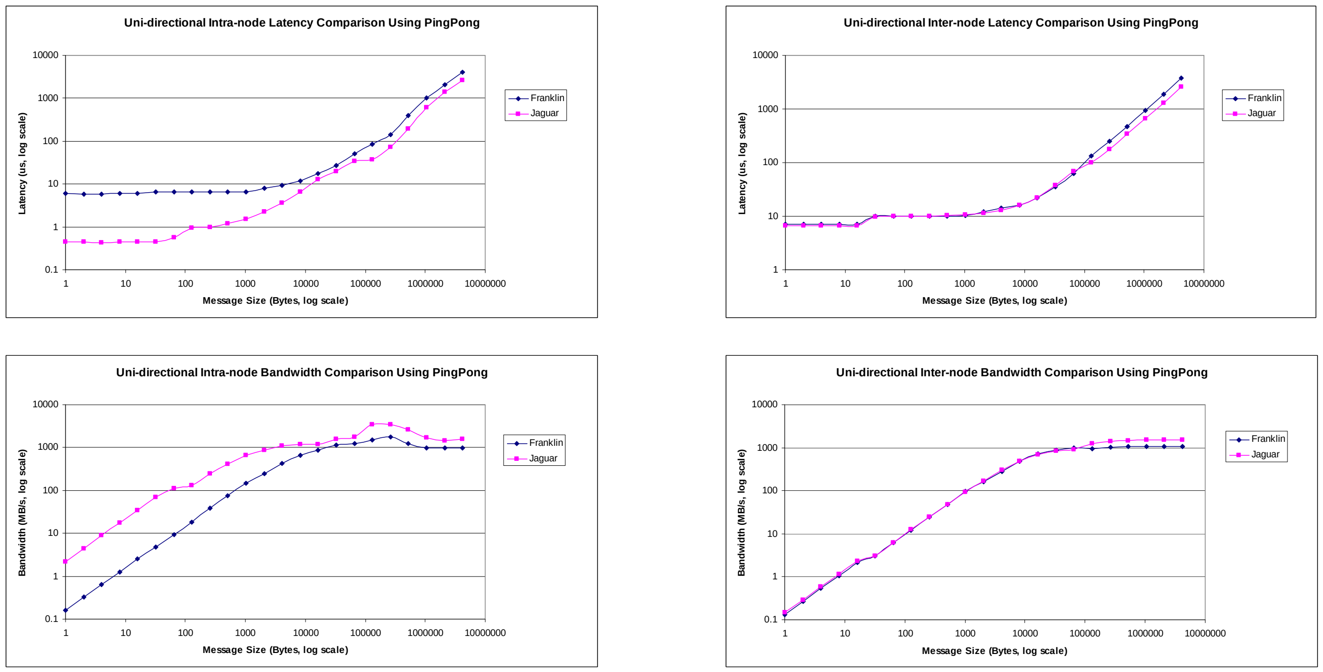


Figure 1. Uni-directional Intra/Inter-node Latency and Bandwidth Comparison Using PingPong

Table 2. Sustainable Memory bandwidth on Franklin

Program Type	**MPI**		**OpenMP**
Processor partitioning scheme	**1x2**	**2x1**	**2 threads**
Memory Bandwidth (MB/s)	4026.53	6710.89	3565.71

Table 3. Sustainable Memory bandwidth on Jaguar

Program Type	**MPI**			**OpenMP**
Processor partitioning scheme	**1x4**	**2x2**	**4x1**	**4 threads**
Memory Bandwidth (MB/s)	5752.19	10066.33	10066.33	5606.77

2.3 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 processor partitioning schemes. 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 CMP 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 Franklin has 2 cores per node, we use MPI and OpenMP STREAM benchmarks to measure the sustainable bandwidths on 2 cores. For different processor partitioning schemes, Table 2 shows the sustainable memory bandwidth increases from 4026.53MB/s to 6710.89MB/s with decreasing the number of cores per node from 2 cores per node to 1 core per node for using 2 MPI processes because fewer MPI processes compete for memory. We also see that the sustainable memory bandwidth for using 2 OpenMP threads is smaller than that for using 2 MPI processes for the scheme 1x2.

Table 3 shows that the sustainable memory bandwidth for using 4 OpenMP threads is smaller than that for any schemes for using 4 MPI processes, and the memory bandwidths for two schemes 2x2 and 4x1 are the same on Jaguar. These memory bandwidths are larger than that for the scheme 1x4.

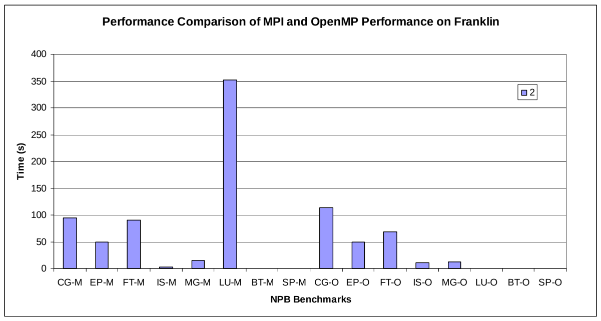
In summary, Tables 2-3 present the sustainable memory bandwidths for MPI and OpenMP on the Dual-core and quad-core Cray XT4 clusters. This indicates that processor partitioning significantly impacts the sustainable memory bandwidth. Hence, using fewer cores per node results in better sustainable memory bandwidth, and using the maximum number of cores per node does not result in the highest sustainable memory bandwidth.

3. MPI and OpenMP Programs

In this section, we use the standard NPB benchmarks (MPI and OpenMP) (NPB version 3.2.1) with problem size of Class B to analyze and compare the performance of MPI and OpenMP programs on Franklin and Jaguar. NPB has 8 benchmarks (5 kernel benchmarks: CG, EP, FT, IS, and MG; 3 application benchmarks: BT, LU, and SP). BT and SP require a square number of cores for execution and others are executed on a power-of-two number of cores. The compiler *ftn* with the options *-O3 –fastsse* is used to compile all benchmarks on both systems.

3.1 Performance Comparison

Figure 2 shows MPI and OpenMP performance comparison and ratio using NPB on Franklin, where CG-M stands for MPI version of CG and CG-O stands for OpenMP version of CG, and so on. The ratio of MPI to OpenMP performance on Franklin indicates that MPI versions of CG, EP and IS outperform their OpenMP counterparts. However, OpenMP versions of FT and MG outperform their MPI counterparts. Especially, the OpenMP version of FT outperforms its MPI counterpart by a factor of more than 1.32. Note that the OpenMP LU does not work on both Franklin and Jaguar.



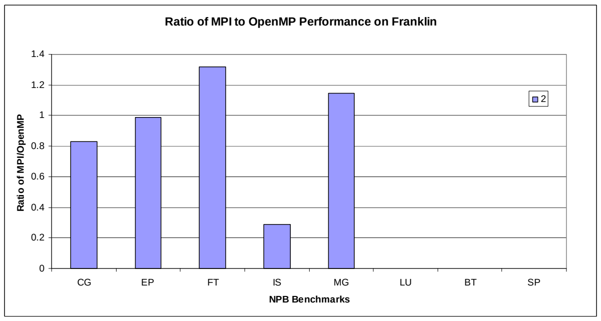
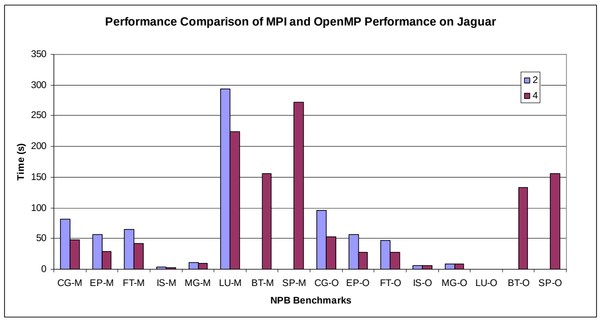


Figure 2. MPI and OpenMP performance comparison for Class B on Franklin



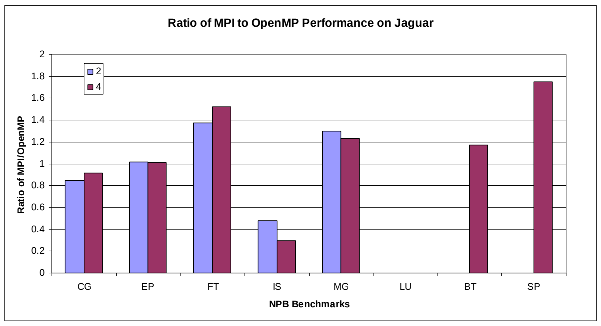


Figure 3. MPI and OpenMP performance comparison for Class B on Jaguar

Figure 3 shows MPI and OpenMP performance comparison and ratio using NPB on Jaguar. The ratio of MPI to OpenMP performance on Jaguar indicates that MPI versions of CG, EP and IS outperform their OpenMP counterparts. However, OpenMP versions of FT, MG, BT and SP outperform their MPI counterparts. Especially, the OpenMP versions of FT and SP outperform their MPI counterparts by a factor of more than 1.52.

In summary, Figures 2 and 3 show MPI and OpenMP performance comparison and ratio using NPB on each compute node of Franklin and Jaguar. The ratio of MPI to OpenMP performance indicates that the MPI versions of CG, IS, and EP outperform their OpenMP counterparts, however, the OpenMP versions of FT, MG, BT and SP outperform their MPI counterparts. As described in [BB94] for MPI implementations and in [JF99] for OpenMP implementations, OpenMP implementations are to reduce the communication overhead of MPI at the expense of introducing OpenMP overhead due to thread creation and increased memory bandwidth contention to take advantage of on-chip high bandwidth and low latency provided by the CMP clusters. In addition, for instance, the OpenMP implementation of FT also eliminates a 3D data array which was needed in the MPI version. This change has improved the memory utilization such that OpenMP FT outperforms its MPI counterpart. So which parallel programming is the most suitable for CMP clusters depends on the nature of an application, available parallel programming software and algorithms, and compiler support on the CMP clusters.

3.2 Processor Partitioning for MPI NPB

In this section, we use MPI version of NPB benchmarks (CG, EP, FT, IS, MG, LU, BT and SP) to investigate how and why processor partitioning impacts their performance on Jaguar and Franklin. Note that the Cray Performance Analysis Tools (CrayPAT) [CPAT] is used to instrument the benchmark codes to gather run-time performance statistics for our analysis. Two groups of hardware counters (groups 2 and 3) are used in our experiments to understand the cache and memory performance in a USER section generated by CrayPAT.

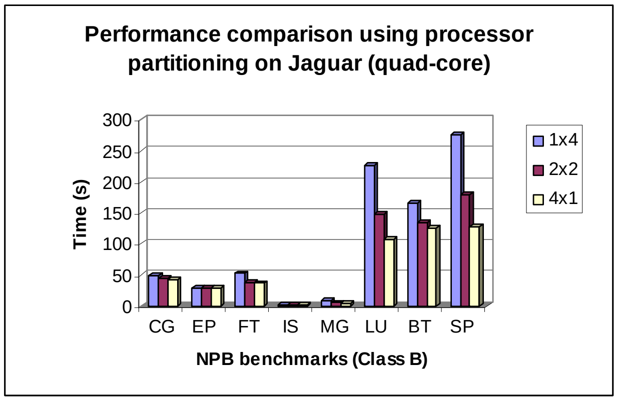


Figure 4. Performance comparisons of NPB benchmarks (MPI) with Class B using processor partitioning on Jaguar

Figure 4 shows the performance comparison of NPB benchmarks with Class B on 4 cores on Jaguar using processor partitioning. We find that using the processor partitioning scheme 4x1 always results in the best performance, and using the scheme 1x4 results in the worst performance. The performance difference between the best and the worst performance is up to 115.1% on 4 cores on Jaguar.

Table 4. Performance comparison of CG with Class B on Jaguar using hardware counters' performance

CG	1x4	2x2	4x1	Diff.(%)
Runtime(s)	48.76	44.16	42.57	14.54%
D1+D2 hit ratio	92.90%	93%	93.60%	
D1 hit ratio	66.50%	66.50%	66.50%	
D2 hit ratio	79.20%	79.40%	79.30%	
Mem-D1 BW (MB/s) per core	1826.96	2033.86	2144.60	17.39%
L2-D1 BW (MB/s) per core	6823.20	7774.85	8175.28	19.82%
L2-Mem BW (MB/s) per core	947.68	1036.18	1066.93	12.58%
Comm. %	2.30%	3.20%	4.30%	

Table 5. Performance comparison of EP with Class B on Jaguar using hardware counters' performance

EP	1x4	2x2	4x1	Diff.(%)
Runtime (s)	28.47	28.39	28.36	0.388%
D1+D2 hit ratio	99.10%	99.10%	99.10%	
D1 hit ratio	99.10%	99.10%	99.10%	
D2 hit ratio	3.30%	3.30%	3.30%	
Mem-D1 BW (MB/s) per core	288.54	288.71	288.73	0.066%
L2-D1 BW (MB/s) per core	2.65	2.64	2.65	0.189%
L2-Mem BW (MB/s) per core	144.32	144.41	144.44	0.082%
Comm. %	0	0	0	

Table 6. Performance comparison of FT with Class B on Jaguar using hardware counters' performance

FT	1x4	2x2	4x1	Diff.(%)
Runtime (s)	52.45	38.52	36.42	44.01%
D1+D2 hit ratio	97.90%	97.90%	97.90%	
D1 hit ratio	90%	90%	90%	
D2 hit ratio	81.60%	81.20%	82.10%	
Mem-D1 BW (MB/s) per core	976.57	1281.864	1307.241	33.86%
L2-D1 BW (MB/s) per core	3619.66	4770.22	4861.41	34.31%
L2-Mem BW (MB/s) per core	418.06	572.86	582.13	39.24%
Comm. %	7.70%	7.90%	8.00%	

Table 7. Performance comparison of IS with Class B on Jaguar using hardware counters' performance

IS	1x4	2x2	4x1	Diff.(%)
Runtime (s)	2.04	1.73	1.67	22.16%
D1+D2 hit ratio	98%	98%	98%	
D1 hit ratio	96.40%	96.40%	96.40%	
D2 hit ratio	50.90%	50.60%	50.40%	
Mem-D1 BW (MB/s) per core	438.78	460.79	471.49	7.45%
L2-D1 BW (MB/s) per core	380.77	399.79	409.08	7.44%
L2-Mem BW (MB/s) per core	230.99	246.31	251.14	8.72%
Comm. %	6.40%	8.60%	9.40%	

Tables 4-11 show the runtime (seconds), the hardware counters' performance and communication percentage for NPB benchmarks with Class B from Figure 4. The D1+D2 hit ratio shows how many memory references were found in cache. The cache hit ratio is affected by cache-line reuse and prefetching, and is useful because the L2 cache serves as a victim cache for L1. "Mem-D1 BW (MB/s) per core" measures the bandwidth (MB/s) per core for the data traffic between memory and L1 data cache; “L2-Mem BW (MB/s) per core” measures the bandwidth (MB/s) per core for the data traffic between L2 and memory; and “L2-D1 BW (MB/s) per core” measures the L2 to Dcache bandwidth per core. “Comm. %” means the communication percentage. “Diff.(%)” means the largest performance difference percentage among difference processor partitioning schemes, 1x4, 2x2 and 4x1.

Table 4 indicates that there is 14.54% performance (runtime) difference between the scheme 4x1 and 1x4 for CG with Class B on Jaguar. When decreasing from using 4 cores per node (1x4) to 1 core per node (4x1), the communication percentage increases a little bit because the communication pattern changes from intra-node communication to inter-node communication. Each cache hit ratio does not change much, however, the memory bandwidth per core (Mem-D1, L2-D1 and L2-Mem) has significant impact of performance degradation of CG. There is 12.58% or more bandwidth difference across the different processor partitioning schemes.

As we know, EP is an embarrassingly parallel benchmark. Table 5 indicates that processor partitioning has very little performance impact of EP. Table 6 presents that there is 44.01% performance difference and 33.86% or more bandwidth difference between the scheme 4x1 and 1x4 for FT with Class B on Jaguar. Table 7 presents that there is 22.16% performance difference and 7.44% or more bandwidth difference between the scheme 4x1 and 1x4, and the communication percentage increases from 6.40% to 9.40% for IS with Class B on Jaguar.

Table 8. Performance comparison of MG with Class B on Jaguar using hardware counters’ performance

MG	1x4	2x2	4x1	Diff.(%)
Runtime (s)	8.72	5.23	4.17	109.11%
D1+D2 hit ratio	96.40%	96.40%	95.20%	
D1 hit ratio	96.10%	96.10%	94.80%	
D2 hit ratio	19.60%	15.30%	9.60%	
Mem-D1 BW (MB/s) per core	1479.01	2456.26	3038.94	105.47%
L2-D1 BW (MB/s) per core	145.52	237.58	299.29	105.68%
L2-Mem BW (MB/s) per core	810.50	1360.86	1673.36	106.46%
Comm. %	2.30%	3.80%	4.40%	

Table 9. Performance comparison of LU with Class B on Jaguar using hardware counters’ performance

LU	1x4	2x2	4x1	Diff.(%)
Runtime (s)	225.54	147.82	106.74	111.30%
D1+D2 hit ratio	95.20%	95.10%	95.20%	
D1 hit ratio	94.80%	94.80%	94.80%	
D2 hit ratio	10.70%	9.70%	9.60%	
Mem-D1 BW (MB/s) per core	1168.9	1822.13	2462.21	110.64%
L2-D1 BW (MB/s) per core	82.09	124.45	177.28	115.95%
L2-Mem BW (MB/s) per core	512.77	882.74	1211.03	136.18%
Comm. %	1.70%	3.50%	4.40%	

Table 10. Performance comparison of BT with Class B on Jaguar using hardware counters’ performance

BT	1x4	2x2	4x1	Diff.(%)
Runtime (s)	165.95	134.65	125.79	31.93%
D1+D2 hit ratio	99.30%	99.30%	99.30%	
D1 hit ratio	98.40%	97.80%	97.80%	
D2 hit ratio	57.30%	67.30%	67.60%	
Mem-D1 BW (MB/s) per core	273.91	288.84	295.53	7.89%
L2-D1 BW (MB/s) per core	339.94	570.60	578.19	70.09%
L2-Mem BW (MB/s) per core	116.58	120.43	179.23	53.74%
Comm. %	1.90%	2.20%	2.40%	

Table 11. Performance comparison of SP with Class B on Jaguar using hardware counters’ performance

SP	1x4	2x2	4x1	Diff.(%)
Runtime (s)	275.35	179.98	128.01	115.10%
D1+D2 hit ratio	92.80%	92.70%	92.70%	
D1 hit ratio	91.20%	91.10%	91.1%	
D2 hit ratio	52.80%	48.30%	45.1%	
Mem-D1 BW (MB/s) per core	1212.99	1861.85	2605.24	114.78%
L2-D1 BW (MB/s) per core	266.18	409.21	589.88	121.61%
L2-Mem BW (MB/s) per core	510.43	992.14	1328.96	160.36%
Comm. %	1.10%	1.10%	1.90%	

Table 8 shows that there is 109.11% performance difference and 105.47% or more bandwidth difference between the scheme 4x1 and 1x4 for MG with Class B on Jaguar. Table 9 shows that there is 111.30% performance difference and 110.64% or more bandwidth difference between the scheme 4x1 and 1x4 for LU with Class B on Jaguar. Table 10 shows that there is 31.93% performance difference and 7.89% or more bandwidth difference between the scheme 4x1 and 1x4 for BT with Class B on Jaguar. Table 11 shows that there is 115.10% performance difference and 114.78% or more bandwidth difference between the scheme 4x1 and 1x4 for SP with Class B on Jaguar. Tables 8-11 also show that the communication percentage does increase a little bit with decreasing number of cores per node.

In brief, using processor partitioning changes the memory access pattern and communication pattern of a MPI program. Regarding the merits of using processor partitioning, the hardware performance counters’ data is conclusive. Processor partitioning has significant performance impact of a MPI program except embarrassingly parallel applications such as EP. The memory bandwidth (Mem-D1, L2-D1 and L2-Mem) per core is the primary source of performance degradation when increasing the number of cores per node.

Figure 5 shows the performance comparison of NPB benchmarks with Class C on 4 cores on Jaguar using processor partitioning. Similarly, we find that using the scheme 4x1 always results in the best performance, and using the scheme 1x4 results in the worst performance.

The performance difference between the best and the worst performance is up to 127.10% on 4 cores. Especially, there is 127.10% performance difference (runtime) for SP between the scheme 4x1 and the 1x4. The hardware performance counters' performance data for Class C also shows that the memory bandwidth per core is the primary source of performance degradation when increasing the number of cores per node.

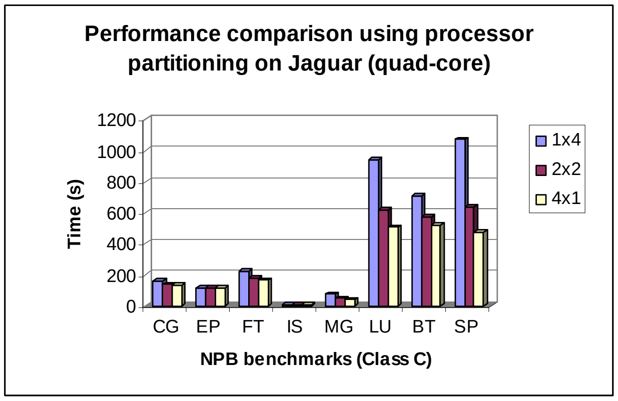


Figure 5. Performance comparisons of NPB benchmarks (MPI) with Class C using processor partitioning on Jaguar

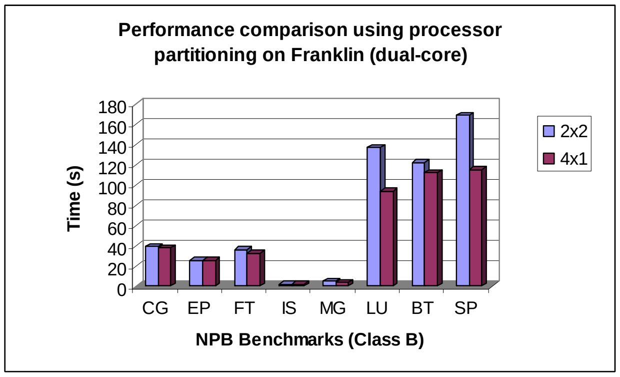


Figure 6. Performance comparisons of NPB benchmarks (MPI) with Class B using processor partitioning on Franklin

Figures 6 and 7 present the performance comparison of NPB benchmarks with Class B and C on 4 cores on Franklin using processor partitioning. We find the similar performance trend, where there is up to 47.09% performance difference for Class B, and up to 52.85% performance difference for Class C among different processor partitioning schemes on dual-core Cray XT4 system Franklin. Especially, there is 47.09%% performance difference and 45.89% or more bandwidth difference between the scheme 4x1 and 2x2 for SP with Class B on Franklin shown in Table 12. The hardware performance counters' performance data for NPB benchmarks on Franklin also shows that the memory bandwidth per core is the primary source of performance degradation when increasing the number of cores per node.

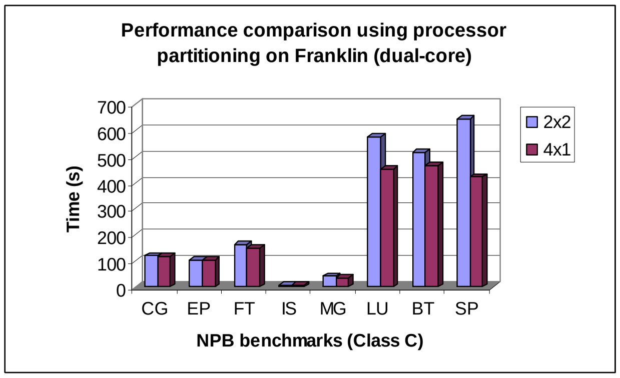


Figure 7. Performance comparisons of NPB benchmarks (MPI) with Class C using processor partitioning on Franklin

Table 12. Performance comparison of SP with Class B on Franklin using hardware counters' performance

SP	2x2	4x1	Difference (%)
Runtime (s)	169.36	115.14	47.09%
D1+D2 hit ratio	92.80%	92.80%	
D1 hit ratio	91.20%	91.10%	
D2 hit ratio	48.90%	45.30%	
Mem-D1 BW (MB/s) per core	1962.18	2866.102	46.07%
L2-D1 BW (MB/s) per core	433.072	647.878	49.60%
L2-Mem BW (MB/s) per core	976.701	1424.925	45.89%
Comm. %	1.00%	1.10%	

4. Hybrid MPI/OpenMP Programs

In this section, we apply processor partitioning to a hybrid MPI/OpenMP Gyrokinetic Toroidal Code (GTC) [ES05] to analyze and compare the performance of the hybrid programs, and discuss how to combine MPI processes and OpenMP threads to achieve good performance on up to 8192 cores.

4.1 Descriptions of Hybrid MPI/OpenMP Programs

The Gyrokinetic Toroidal code (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. Figure 8 shows a visualization of potential contours of microturbulence for a magnetically confined plasma using GTC. The finger-like perturbations (streamers) stretch along the weak field side of the poloidal plane as they follow the magnetic field lines around the torus [SD06].

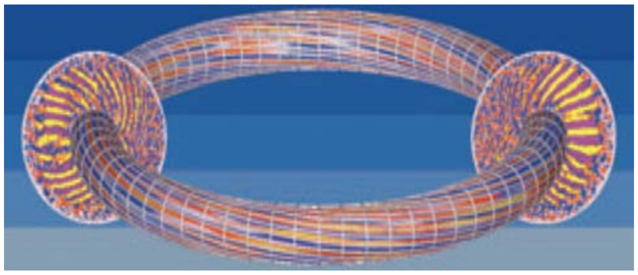


Figure 8. Potential contours of microturbulence for a magnetically confined plasma [SD06].

Figure 9 presents the basic steps in the GTC code. There are 7 major functions: *load, field, smooth, poisson, charge, shift and pusher* in the code. *charge, pusher* and *shift* dominate the most of the application execution time. The test case for GTC studied in this paper is 100 particles per cell and 100 time steps. The problem sizes for the GTC code are listed in Table 13, where micell is the number of ions per grid cell, mecell is the number of electrons per grid cell, mzetamax is the total number of toroidal grid points, and npartdom is the number of particle domain partitions per toroidal domain.

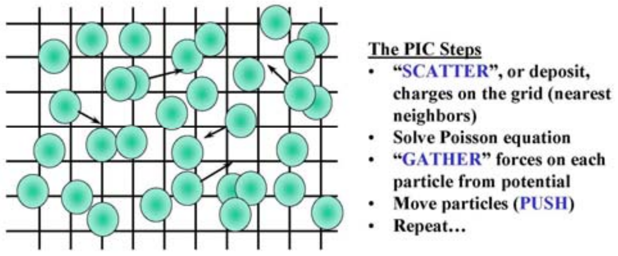

The PIC Steps
"SCATTER", or deposit, charges on the grid (nearest neighbors)
Solve Poisson equation
"GATHER" forces on each particle from potential
Move particles (PUSH)
Repeat…

Figure 9. Particles in cell (PIC) steps [ES05]

Table 13. Datasets with scaling the number of processors

#Procs	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192
micell	100	100	100	100	100	100	200	400	800	1600	3200	6400	12800
mecell	100	100	100	100	100	100	200	400	800	1600	3200	6400	12800
mzetamax	2	4	8	16	32	64	64	64	64	64	64	64	64
npartdom	1	1	1	1	1	1	2	4	8	16	32	64	128

4.2 Processor Partitioning for MPI Programs

In this section, we use the MPI GTC (hybrid GTC with 1 OpenMP thread) to investigate how processor partitioning impacts the performance of MPI programs and their components. Similar to the conclusions we drew for NPB benchmarks, the hardware performance counters' performance data for GTC on Franklin and Jaguar shows that the memory bandwidth per core is the primary source of performance degradation when increasing the number of cores per node. For sake of simplicity, in this section, we just focus on how processor partitioning affects the performance of the components of the GTC code.

Table 14. Performance for different processor partitioning schemes on 2 cores on Franklin

2 Cores	1x2	2x1	Difference (%)
Runtime	562.23	552.34	1.79%
load	3.54	3.16	11.99%
field	1.37	1.15	18.85%
smooth	2.06	2.04	1.08%
poisson	6.18	5.36	15.34%
charge	239.9	238	0.798%
shift	10.59	9.38	12.85%
pusher	296.9	291.2	1.96%

Table 15. Performance for different processor partitioning schemes on 64 cores on Franklin

64 Cores	32x2	64x1	Difference (%)
Runtime	645.69	617.49	4.57%
load	3.47	3.15	10.13%
field	1.35	1.11	21.46%
smooth	2.04	1.9	7.53%
poisson	5.97	5.31	12.25%
charge	263.1	259.4	1.43%
shift	53.21	33.27	59.93%
pusher	316.1	312.9	1.02%

Tables 14-15 show that different processor partitioning schemes impact the performance of the MPI GTC on Franklin. There is 1.79% performance difference between the scheme 1x2 and 2x1 on 2 cores, and 4.57% performance difference between the scheme 32x2 and 64x1 on 64 cores on Franklin. Using one MPI process per node achieved the best performance for all cases. Although processor partitioning has significant impacts of the performance of the functions *load, field, poisson* and *shift* with 11.99% or more performance difference on 2 cores and with 10.13% or more performance difference on 64 cores, especially, there is 59.93% performance difference on 64 cores for the function *shift*, the functions *charge* and *pusher* dominate the most of the application execution time. So the contributions from the two dominated functions reflect the overall performance difference.

Table 16. Performance for different processor partitioning schemes on 4 cores on Jaguar

4 Cores	1x4	2x2	4x1	Diff.(%)
Runtime	698.45	668.2	657.47	6.23%
load	4.34	3.88	3.68	18.05%
field	1.64	1.14	1.03	58.90%
smooth	2.46	2.14	2.07	18.95%
poisson	7.51	5.94	5.47	37.19%
charge	304.8	296	292.1	4.35%
shift	16.07	11.76	10.33	55.57%
pusher	361	346.7	342.2	5.49%

Table 17. Performance for different processor partitioning schemes on 64 cores on Jaguar

64 Cores	16x4	32x2	64x1	Diff.(%)
Runtime	792.54	733.85	715.16	10.82%
load	4.53	3.72	3.72	21.79%
field	1.73	1.10	1.09	60.39%
smooth	2.53	2.23	2.06	22.84%
poisson	6.93	5.70	5.70	21.51%
charge	333.8	317.2	313.9	6.34%
shift	55.12	34.6	22.69	142.93%
pusher	386.8	368.7	365.3	5.89%

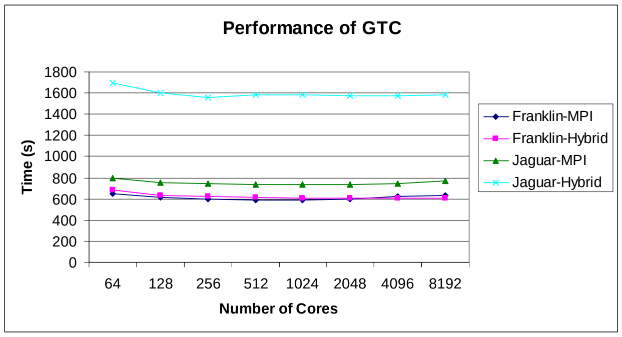
Tables 16-17 also show that different processor partitioning schemes impact the performance of the MPI GTC on Jaguar. There is 6.23% performance difference between the scheme 1x4 and 4x1 on 4 cores, and 10.82% performance difference between the scheme 16x4 and 64x1 on 64 cores on Jaguar. Using one MPI process per node achieved the best performance for all cases. Although processor partitioning also has significant impacts of the performance of the functions *load, field, smooth, poisson* and *shift* with 18.05% or more performance difference on 4 cores and with 21.51% or more performance difference on 64 cores, especially, there is 55.57% performance difference on 4 cores and 142.93% performance difference on 64 cores for the function *shift*, the functions *charge* and *pusher* dominate the most of the application execution time. So the contributions from the two dominated functions reflect the overall performance difference.

In summary, using processor partitioning, we can explore the performance characteristics of the GTC and its components, and understand how each component of GTC is sensitive to different communication patterns and memory access patterns. This should be useful to further optimize the application performance.

4.3 Combination of MPI Processes and OpenMP Threads

In previous section, we discussed that processor partitioning did impact the performance of MPI programs, and using one MPI process per node achieved the best performance for all cases. Therefore, for the hybrid GTC, for sake of simplicity, we use one MPI process per node and one OpenMP thread per core on each node to measure the performance of the hybrid GTC executed on up to 8192 cores, and compare the performance with that for its MPI counterpart.

We use the performance on 64 cores as a baseline, and define that a relative speedup is the base performance on 64 cores divided by the performance on N cores (N>=64). The larger the speedup is, the better the scalability is. In Figure 10, Franklin-MPI means the MPI GTC executed on Franklin and Franklin-Hybrid means the hybrid GTC executed on Franklin, and so on. Figure 10 indicates that the execution times for the hybrid GTC on Jaguar are more than two times larger than that for the MPI GTC, however, its speedup indicates that GTC scales well on Jaguar and Franklin. The hybrid GTC on Franklin has the better scalability.



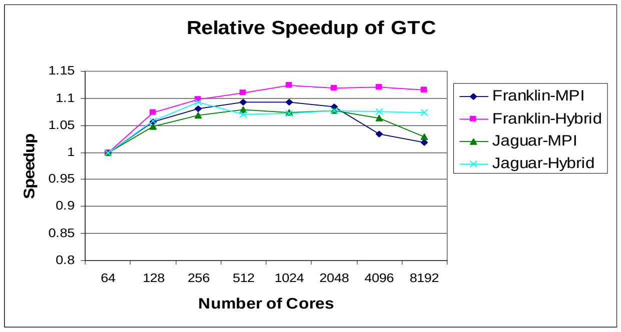


Figure 10. MPI and Hybrid Performance Comparison for GTC

Figure 11 indicates that the MPI GTC outperforms the hybrid MPI/OpenMP GTC on the two CMP clusters in most cases, however, hybrid GTC scales much better than its MPI counterpart shown in Figure 10, and outperforms its MPI counterpart on 4096 and 8192 cores on Franklin because of the large increase of communication time for the MPI GTC on 4096 and 8192 cores. The ratio of MPI to hybrid GTC on Jaguar is less than 0.5. Of course, other combinations of MPI processes and OpenMP threads such as half of cores for MPI processes and another half for OpenMP thread on a CMP node can be considered, however, our experimental results show that, for most cases, using all cores for MPI processes per node results in the best performance for GTC on 2048 cores or less.

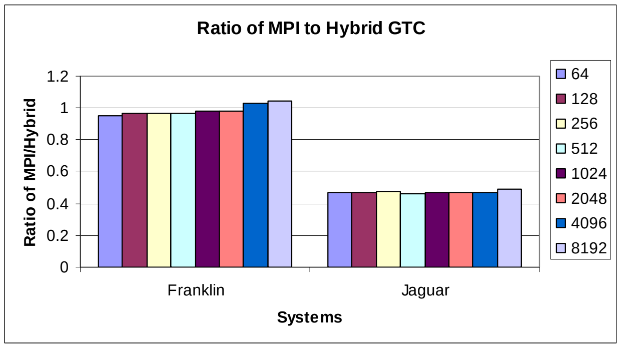


Figure 11. Ratio of MPI to Hybrid Performance for GTC on Franklin and Jaguar

5. Performance Modeling Using Prophesy System

In this section, we use Prophesy system [TW03, WT06] to online generate a performance model based on the performance data collected on up to 8192 processors for GTC on Jaguar and Franklin, and predict its performance on 10,240 processors based on the models as shown in the following figures.

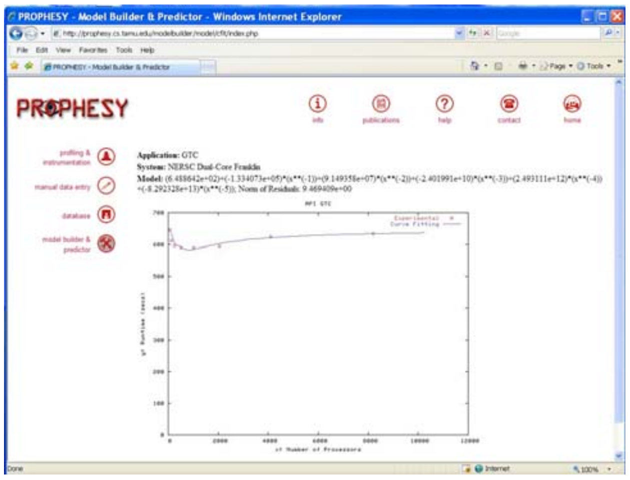

Application: GTC
System: NERSC Dual-Core Franklin
Model: (6.488642e+02)+(-1.334073e+05)*(x**(-1))+(9.149358e+07)*(x**(-2))+(-2.401991e+10)*(x**(-3))+(2.493111e+12)*(x**(-4))+(-8.292328e+13)*(x**(-5)); Norm of Residuals: 9.469409e+00
PROPHESY - Model Builder & Predictor - Windows Internet Explorer
PROPHESY
info
publications
help
contact
home
profiling & instrumentation
manual data entry
database
model builder & predictor
MPI GTC
Experimental
Curve Fitting
# of Number of Processors

Figure 12. Using Prophesy to generate a performance model for GTC to predict the performance of GTC on 10,240 processors for MPI GTC on Franklin

Figures 12 and 13 present the performance models for MPI and hybrid GTC on Franklin. Based on these performance models, their performance on 10,240 cores is predicted shown in the plots from Figures 12 and 13. We find that the hybrid GTC has better scalability than the MPI GTC because the execution of the hybrid GTC on Franklin utilizes the inter-node MPI communication shown in Figure 1 and intra-node OpenMP communication.

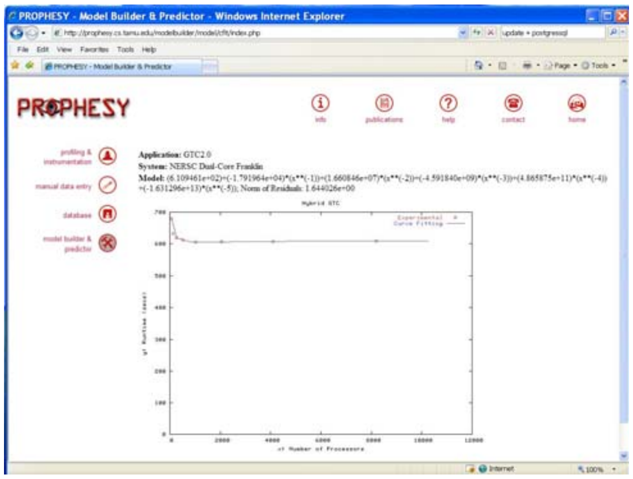

Application: GTC2.0
System: NERSC Dual-Core Franklin
Model: (6.109461e+02)+(-1.791964e+04)*(x**(-1))+(1.660846e+07)*(x**(-2))+(-4.591840e+09)*(x**(-3))+(4.865875e+11)*(x**(-4))+(-1.631296e+13)*(x**(-5)); Norm of Residuals: 1.644026e+00
PROPHESY - Model Builder & Predictor - Windows Internet Explorer
PROPHESY
info
publications
help
contact
home
profiling & instrumentation
manual data entry
database
model builder & predictor
Hybrid GTC
Experimental
Curve Fitting
# of Number of Processors

Figure 13. Using Prophesy to generate a performance model for GTC to predict the performance of GTC on 10,240 processors for hybrid GTC on Franklin

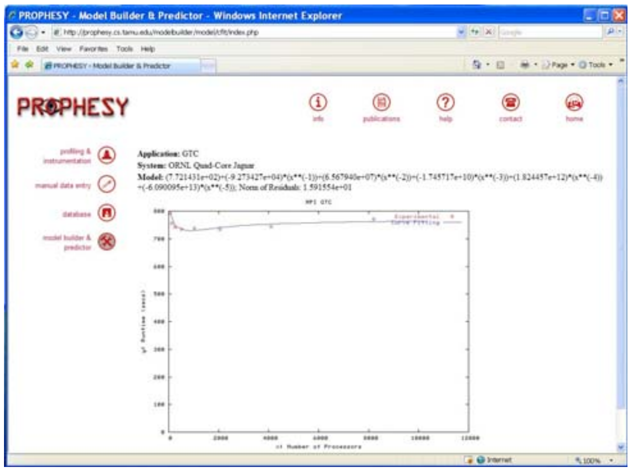
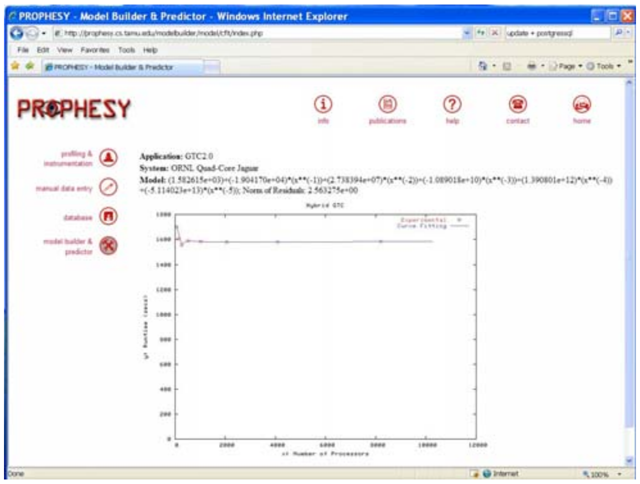

Application: GTC
System: ORNL Quad-Core Jaguar
Model: (7.721431e+02)+(-9.273427e+04)*(x**(-1))+(6.567940e+07)*(x**(-2))+(-1.745717e+10)*(x**(-3))+(1.824457e+12)*(x**(-4))+(-6.090095e+13)*(x**(-5)); Norm of Residuals: 1.591554e+01
PROPHESY - Model Builder & Predictor - Windows Internet Explorer
PROPHESY
info
publications
help
contact
home
profiling & instrumentation
manual data entry
database
model builder & predictor
MPI GTC
Experimental
Curve Fitting
# of Number of Processors

Figure 14. Using Prophesy to generate a performance model for GTC to predict the performance of GTC on 10,240 processors for MPI GTC on Jaguar

Figures 14 and 15 present the performance models for MPI and hybrid GTC on Franklin. Based on these performance models, their performance on 10,240 cores is predicted shown in the plots from Figures 14 and 15. We also find that the hybrid GTC has better scalability than the MPI GTC.


PROPHESY - Model Builder & Predictor - Windows Internet Explorer
File Edit View Favorites Tools Help
PROPHESY
info
publications
help
contact
home
profiling & instrumentation
manual data entry
database
model builder & predictor
Hybrid GTC
Experimental
Curve Fitting
Number of Processors

Application: GTC2.0
System: ORNL Quad-Core Jaguar
Model: (1.582615e+03)+(-1.904170e+04)*(x**(-1))+(2.738394e+07)*(x**(-2))+(-1.089018e+10)*(x**(-3))+(1.390801e+12)*(x**(-4))+(-5.114023e+13)*(x**(-5)); Norm of Residuals: 2.563275e+00

Figure 15. Using Prophesy to generate a performance model for GTC to predict the performance of GTC on 10,240 processors for hybrid GTC on Jaguar

5. Related Work

G. Jost, H. Jin and et al [JJ03] developed two hybrid Block Tridiagonal (BT) benchmarks, and compared them with the NPB MPI BT and OpenMP BT benchmarks on a Sun Fire SMP cluster. They found that the benefit of the hybrid implementations of BT was visible on a slow network, and the MPI BT turned out to be the most efficient for using the high-speed interconnect or shared memory. H. Jin, M. Frumkin and J. Yan [JF99] presented the OpenMP implementation of NAS parallel benchmarks based on the optimized sequential version of NPB2.3-serial, and compared their performance with the performance of MPI implementation of NAS parallel benchmarks on shared memory SGI Origin 2000. They demonstrated that OpenMP can achieve very good performance on the shared memory system although the scalability is worse than the MPI counterpart.

F. Callello and D. Etiemble [CE00] compared MPI and hybrid MPI/OpenMP (OpenMP fine grain parallelization after profiling) for NPB 2.3 benchmarks on two IBM Power3 systems. The hybrid MPI/OpenMP was to add OpenMP parallelization of loop nests in the computation part of the MPI code. Their results showed that a unified MPI approach is better for most of NPB benchmarks, however, the hybrid approach became better only when fast processors make the communication performance significant and the level of parallelization is sufficient.

Wong et al. [WM99] presented a study of the architectural requirements and scalability of the NAS parallel benchmarks, and identified the factors which affected the scalability of benchmark codes through direct measurements and simulations on two relevant and distinct platforms, a cluster of workstations and a cc-NUMA SGI Origin 2000.

In our previous work [WT07], we proposed processor partitioning, and discussed its impacts. In contrast to these approaches, we used processor partitioning to systematically analyze and compare the performance of MPI, OpenMP and hybrid programs on two dual- and quad-core Cray XT4 systems. Because these CMP clusters provide a natural programming paradigm for hybrid programs, if we simply treat the CMPs as traditional SMPs for running MPI programs, we may miss very interesting opportunities for new architectures and algorithm designs that can exploit these new features such as on-chip high inter-core bandwidth and low latency. So we investigated the performance of a hybrid MPI/OpenMP program GTC, and presented its performance model generated online by Prophesy system.

6. Conclusions

In this paper, we used processor partitioning to analyze and compare the performance of MPI, OpenMP and hybrid parallel applications on two dual- and quad-core Cray XT4 systems, Jaguar with quad-core and Franklin with dual-core, and conducted detailed performance experiments to identify the major application characteristics that affect processor partitioning. The experimental results indicated that processor partitioning could have a significant impact on performance of a parallel scientific application as determined by its communication and memory requirements. We also used the STREAM memory benchmarks and Intel's MPI benchmarks to explore the performance impact of different application characteristics. The results were then utilized to explain the performance results of processor partitioning using NAS Parallel Benchmarks. We found that the memory bandwidth per core is the primary source of performance degradation when increasing the number of cores per node using processor partitioning.

In addition to using these benchmarks, we also used a flagship SciDAC fusion microturbulence code (hybrid MPI/ OpenMP): a 3D particle-in-cell application Gyrokinetic Toroidal Code (GTC) in magnetic fusion to analyze and compare the performance of these MPI and hybrid programs on the dual- and quad-core Cray XT4 systems, and studied their scalability on up to 8192 cores. Based on the performance of GTC on up to 8192 cores, we used Prophesy system to online generate its performance models to predict its performance on 10,240 cores on the two Cray XT4 systems.

Acknowledgements

The authors would like to acknowledge DOE NERSC for the use of the Franklin and Oak Ridge NCCS for the use of Jaguar under DOE INCITE project "Performance Evaluation and Analysis Consortium End Station". 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 and datasets.

About the Authors

Xingfu Wu is a TEES Research Scientist in Department of Computer Science & Engineering at Texas A&M University. He is an ACM Senior member and IEEE member. His research areas mainly focus on performance analysis, modeling and prediction of high performance computing applications on large-scale supercomputers and Grids. His contact information is Department of Computer Science & Engineering, Texas A&M University, College Station, TX 77843. His email is wuxf@cse.tamu.edu.

Valerie Taylor is Department Head and Royce E. Wisenbaker Professorship in Engineering, in Department of Computer Science & Engineering at Texas A&M University. Her research areas mainly focus on performance analysis, modeling and prediction of high performance computing applications on large-scale supercomputers and Grids. Her contact information is Department of Computer Science & Engineering, Texas A&M University, College Station, TX 77843. Her email is taylor@cse.tamu.edu.

References

[BB94] D. Bailey, E. Barszcz, et al., *The NAS Parallel Benchmarks*, Tech. Report RNR-94-007, March 1994.

[CE00] F. Cappello and D. Etiemble, MPI versus MPI+OpenMP on the IBM SP for the NAS Benchmarks, *SC2000*.

[CPAT] CrayPAT: Cray Performance Tools, https://www.nersc.gov/nusers/systems/franklin/tools.php#craypat or http://www.nccs.gov/computing-resources/jaguar/software/?&software =craypat.

[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.

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

[JF99] H. Jin, M. Frumkin and J. Yan, *The OpenMP Implementation of NAS Parallel Benchmarks and Its Performance*, NAS Technical Report NAS-99-011, October 1999.

[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, et al., *Understanding and Mutigating 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.

[NPB3] NAS Parallel Benchmarks 3.2.1, http://www.nas.nasa.gov/Resources/Software/npb.html.

[NCCS] NCCS Jaguar, http://www.nccs.gov/computing-resources/jaguar/

[NERS] NERSC Franklin, http://www.nersc.gov/nusers/systems/franklin/.

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

[SD06] Scientific Discovery, A progress report on the US DOE SciDAC program, 2006.

[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, March 2003.

[WM99] F. Wong, R. Martin, R. Arpaci-Dusseau, and D. Culler, Architectural Requirements and Scalability of the NAS Parallel Benchmarks, *SC99*, 1999.

[WT06] Xingfu Wu, Valerie Taylor, and Joseph Paris, A Web-based Prophesy Automated Performance Modeling System, *the International Conference on Web Technologies, Applications and Services (WTAS2006)*, July 17-19, 2006, Calgary, Canada.

[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, Hotel@MIT, Cambridge, MA.

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