Performance Characteristics of Hybrid MPI/OpenMP Implementations of NAS Parallel Benchmarks SP and BT on large-scale Multicore Clusters

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
The NAS Parallel Benchmarks (NPB) are well-known applications with the fixed algorithms for evaluating parallel systems and tools. Multicore clusters provide a natural programming paradigm for hybrid programs, whereby Open MP can be used with the data sharing with the multicores that comprise a node and MPI can be used with the communication between nodes. In this paper, we use SP and BT benchmarks of MPI NPB 3.3 as a basis for a comparative approach to implement hybrid MPI/OpenMP versions of SP and BT. In particular, we can compare the performance of the hybrid SP and BT with the MPI counterparts on large-scale multicore clusters. Our performance results indicate that the hybrid SP outperforms the MPI SP by up to 20.76%, and the hybrid BT outperforms the MPI BT by up to 8.58% on up to 10,000 cores on BlueGene/P at Argonne National Laboratory and Jaguar (Cray XT4/5) at Oak Ridge National Laboratory. We also use performance tools and MPI trace libraries available on these clusters to further investigate the performance characteristics of the hybrid SP and BT.

General Terms
Measurement, Performance, Benchmarks

Keywords
Performance characteristics, Hybrid MPI/OpenMP, NAS Parallel benchmarks, and Multicore clusters.

1. Introduction

The NAS Parallel Benchmarks (NPB) [10] are well-known applications with the fixed algorithms for evaluating parallel systems and tools. These benchmarks exhibit mostly fine-grained parallelism. Implementations in MPI [2] and OpenMP [5] take advantage of this fine-grained parallelism. However, current multicore clusters and many scientific problems feature several levels of parallelism. G. Jost, H. Jin and et al [6] developed two hybrid Block Tridiagonal (BT) benchmarks, and compared them with the MPI BT and OpenMP BT benchmarks on a Sun Fire SMP cluster. They found the MPI BT to be the most efficient for using the high-speed interconnect or shared memory. F. Callello and D. Etiemble [3] compared MPI and hybrid MPI/OpenMP (OpenMP fine grain parallelization after profiling) for NPB 2.3 benchmarks on two IBM Power3 systems. Their results indicated a unified MPI approach to be better for most of NPB benchmarks, especially Scalar Pentadiagonal (SP) benchmark and BT. Although the NPB Multi-Zone (NPB-MZ) versions [7, 14] exploit multilevel parallelism (i.e., coarse-grained parallelism between zones and fine-grained, loop-level parallelism within each zone) there is no comparison for the implementation of NPB-MZ with that of the original NPB because NPB-MZ uses different problem sizes than NPB. In this paper, we use the latest version MPI NPB 3.3 as a basis for a comparative approach to implement hybrid MPI/OpenMP NPB (Hybrid NPB) and compare the performance of the two implementations on large-scale multicore clusters. In particular, we focus on SP and BT benchmarks for our comparative study.

Today, multicore clusters provide a natural programming paradigm for hybrid programs. Generally, MPI is considered optimal for process-level (or coarse) parallelism and OpenMP is optimal for loop-level (or fine grain) parallelism. Combining MPI and OpenMP parallelization to construct a hybrid program reduces 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 implement hybrid MPI/OpenMP implementations of SP and BT benchmarks of MPI NPB 3.3, and compare the performance on three large-scale multicore clusters: BlueGene/P at Argonne National Laboratory (ANL) [1], Jaguar (Cray XT4) and JaguarPF (Cray XT5) at Oak Ridge National Laboratory (ORNL) [11]. BlueGene/P has 4 cores per node, Jaguar has 4 cores per node, and JaguarPF has 12 cores per node. The experiments conducted for this work utilize different number of cores per node. Further, each cluster has a different node memory hierarchy. Our performance results show that the hybrid SP outperforms the MPI SP by up to 20.76%, and the hybrid BT outperforms the MPI BT by up to 8.58% on up to 10,000 cores (depending on the problem sizes) on these clusters. We also use the performance tools and libraries available on these clusters such as MPI Profiling and Tracing Library [9] and Universal Performance Counter [13] available on BlueGene/P and CrayPat [4] available on Jaguar to investigate the performance characteristics of the hybrid SP and BT in detail. We observe that, for the hybrid SP and BT with the given problem size, when increasing the number of cores, the MPI SP and BT outperforms their hybrid counterparts on both BlueGene/P and Jaguar because of the decreased sizes of the parallelized loops combined with more OpenMP overhead and memory bandwidth contention. This paper addresses these issues in detail.

The remainder of this paper is organized as follows. Section 2 describes the original MPI SP and BT of NPB3.3 and the SP and BT of NPB-MZ 3.3, and discusses their difference. Section 3 presents our hybrid MPI/OpenMP implementations of the SP and BT in detail, and compares them with MPI SP and BT and NPB-MZ SP and BT. Section 4 briefly describes three experimental platforms we used for this work, presents a comparative analysis of the SP and BT in detail by using MPI and OpenMP communication performance and performance counters' data, and discusses scalability of the hybrid SP and BT on JaguarPF with 12 cores per node and some limitations of the hybrid SP and BT. Section 5 concludes the paper.

2. SP and BT

SP and BT are two application benchmarks of NPB 3.3 [10]. SP (Scalar Pentadiagonal) solves three sets of uncoupled systems of equations, first in the X dimension, then in the Y dimension, and finally in the Z dimension; these systems are scalar pentadiagonal. BT (Block Tridiagonal) solves three sets of uncoupled systems of equations, first in the X dimension, then in the Y dimension, and finally in the Z dimension; these systems are block tri-diagonal with 5x5 blocks. The iteration procedure of SP is very similar to that of BT although the approximate factorization is different. The high-level flow chart of MPI SP and BT of NPB 3.3 is shown in Figure 1. Each MPI process executes the initialization step. After synchronization (MPI_Barrier) of all processes, the benchmarking loop starts with a time step loop, which consists of three major solvers in X, Y and Z dimensions: X_Solve, Y_Solve and Z_Solve. Finally, the solution is verified for a given problem size class.

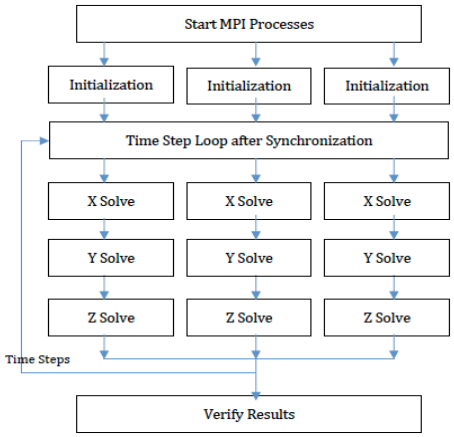

Figure 1. Original MPI BT and SP of NPB3.3

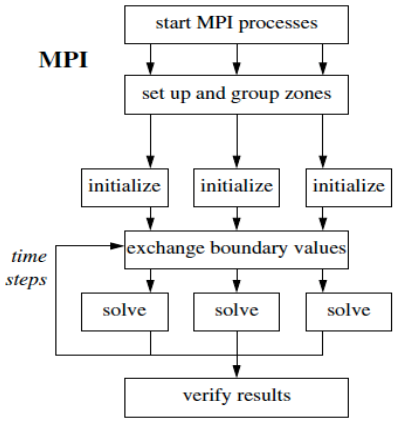

Figure 2. NPB-MZ BT and SP [JW03]

Jin and Wijngaart [7, 14] developed NPB Multi-Zone versions, which contain SP, BT and LU derived from the NPB. These benchmarks exploit two-level parallelism: a coarse-grained parallelization among zones and a fine-grained parallelization within each zone. The implementation of the multi-zone benchmarks is summarized in Figure 2. Each process is first assigned with a group of zones and a given number of OpenMP threads. OpenMP directives are not used in the implementation, only two OpenMP calls (omp_get_max_threads(),omp_set_num_threads()) are used for getting/setting the number of threads to be used. There is no communication during the solving stage (for SP or BT). This is different from MPI NPB version where there is communication within performing the solution of the approximate factorization step in the X-dimension, Y-dimension and Z-dimension.

Table 1 shows the large problem sizes for MPI SP and BT of NPB3.3, and Table 2 shows the large problem sizes for NPB-MZ 3.3. For each problem size class, the total number of mesh points for NPB3.3 and NPB-MZ 3.3 is approximately the same, however, the sizes for X, Y and Z dimensions are different. NPB-MZ 3.3 does not have the problem size of Class E. The differences in the dimensions of the problems for NPB-MS with that of NPB hinder the ability to compare the two implementations.

Table 1. Problem Sizes for MPI SP and BT of NPB3.3

Class	Mesh Dimensions		
	X	Y	Z
A	64	64	64
B	102	102	102
C	162	162	162
D	408	408	408
E	1020	1020	1020

Table 2. Problem Sizes for SP and BT of NPB-MZ3.3

Class	Mesh Dimensions		
	X	Y	Z
A	128	128	16
B	304	208	17
C	480	320	28
D	1632	1216	34

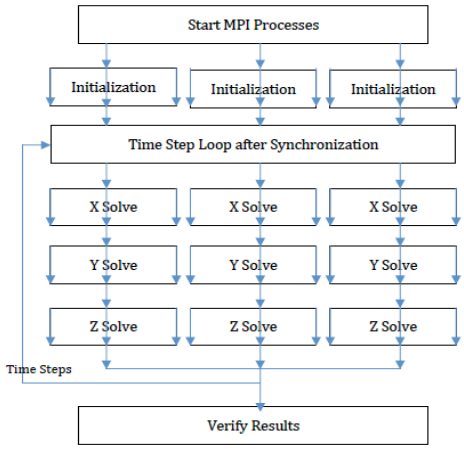

Figure 3. Hybrid NPB BT and SP

3. Hybrid MPI/OpenMP Implementations

Our hybrid MPI/OpenMP implementations of SP and BT are based on the SP and BT of MPI NPB version 3.3. The hybrid MPI/OpenMP implementation of SP and BT is summarized in Figure 3. Each MPI process executes the initialization step on each node of a multicore cluster, and OpenMP is used to parallelize the

step mostly at loop level. After synchronization (MPI_Barrier) of all processes, the benchmarking loop starts with a time step loop, which consists of three major solvers in X, Y and Z dimensions: X_Solve, Y_Solve and Z_Solve. Because each solver (X_Solve, Y_Solve or Z_Solve) has MPI communication, when OpenMP is applied to parallelize each solver mostly at loop level, there is no MPI communication inside an OpenMP parallel region. As illustrated in Figure 3, for example, given a compute resource (three nodes with three cores per node), there are a total of three MPI processes, with a MPI process per node, and three OpenMP threads per MPI process. For a fair comparison, for the given same compute resources, as illustrated in Figure 1, we use a total of nine MPI processes with three MPI processes per node. The combining MPI and OpenMP parallelization to construct the hybrid SP and BT achieves multiple levels of parallelism and reduces the communication overhead of MPI.

Basically, to balance different levels of parallelism provided by a multicore cluster, the hybrid SP and BT use MPI for communication between nodes of the multicore cluster and OpenMP for parallelization within each node. Parallelism in the hybrid SP and BT can be exploited with process-level, coarse parallelism (using MPI) and loop-level, fine grain parallelism (using OpenMP). The steps in the multilevel parallelization process are the following:

1) Identify where MPI communication occurs in each step shown in Figure 3.
2) Identify loops in each step where different iterations can be executed independently. If there is data dependence in a loop, transform the loop so that different iterations can be executed independently.
3) Insert "!$omp parallel do" directives for these loops to ensure large granularity and small OpenMP overhead by grouping several parallel loops into a single parallel region and using "!$omp end do nowait" to remove end-of-loop synchronizations if possible.
4) Make sure that MPI communications occur outside of all OpenMP parallel regions because there is MPI communications within each major step (X_Solver, Y_Solver, or Z_Solver) shown in Figure 3.

Note that without enabling the OpenMP directives, the hybrid SP and BT are approximately identical to their pure MPI versions. In the following, we use hybrid SP as an example to discuss how to transform a loop with data dependence into the loop without data dependence. We take a simple code segment from x_solve.f in SP benchmark to demonstrate our method as follows.

The original code segment from x_solve.f in the SP benchmark:

```
p = 0
n = 0
do   k = start(3,c), ksize-end(3,c)-1
  do   j = start(2,c), jsize-end(2,c)-1
    do   i = iend-1, iend
      out_buffer(p+1) = lhs(i,j,k,n+4,c)
      out_buffer(p+2) = lhs(i,j,k,n+5,c)
      do   m = 1, 3
        out_buffer(p+2+m) = rhs(i,j,k,m,c)
      end do
      p = p+5
    end do
  end do
end do

do   m = 4, 5
  n = (m-3)*5
  do   k = start(3,c), ksize-end(3,c)-1
    do   j = start(2,c), jsize-end(2,c)-1
      do   i = iend-1, iend
        out_buffer(p+1) = lhs(i,j,k,n+4,c)
        out_buffer(p+2) = lhs(i,j,k,n+5,c)
        out_buffer(p+3) = rhs(i,j,k,m,c)
        p = p + 3
      end do
    end do
  end do
end do
```

The transformed code segment with OpenMP directives:

```
      p = 0
      n = 0
!$omp parallel default(shared) private(j,k,m,i)
!$omp do lastprivate(p)
      do   k = start(3,c), ksize-end(3,c)-1
!$      p = (k - start(3,c)) * (jsize-end(2,c)-start(2,c)) * 2 * 5
        do   j = start(2,c), jsize-end(2,c)-1
          do   i = iend-1, iend
            out_buffer(p+1) = lhs(i,j,k,n+4,c)
            out_buffer(p+2) = lhs(i,j,k,n+5,c)
            do   m = 1, 3
              out_buffer(p+2+m) = rhs(i,j,k,m,c)
            end do
            p = p+5
          end do
        end do
      end do
!$omp end do
!$      pp = p
      do   m = 4, 5
        n = (m-3)*5
!$omp do lastprivate(p)
        do   k = start(3,c), ksize-end(3,c)-1
!$        p = pp + (k - start(3,c)) * (jsize-end(2,c)-start(2,c))  * 2 * 3
          do   j = start(2,c), jsize-end(2,c)-1
            do   i = iend-1, iend
              out_buffer(p+1) = lhs(i,j,k,n+4,c)
              out_buffer(p+2) = lhs(i,j,k,n+5,c)
              out_buffer(p+3) = rhs(i,j,k,m,c)
              p = p + 3
            end do
          end do
        end do
!$omp end do
!$        pp = p
      end do
!$omp end parallel
```

From the original code segment, we find that the variable p has data dependence in both nested loops. We set the variable p as lastprivate because the second nested loop needs the final value of the p for the sequentially last iteration in the first nested loop. We also introduce a shared variable pp = p. So when we add p = (k - start(3,c)) * (jsize-end(2,c)-start(2,c)) * 10 in the first nested loop and p = pp + (k - start(3,c)) * (jsize-end(2,c)-start(2,c)) * 6 into the second nested loop, we avoid the data dependence in both loops so that we can use OpenMP to parallelize both loops. Note that without enabling the OpenMP directives, the transformed code segment is identical to its original one.

4. Performance Analysis and Comparison

In this section, we describe three large-scale multicore clusters, and execute the hybrid SP and BT on them to compare their performance with the performance of their MPI counterparts.

4.1 Experimental Platforms

In this paper, we use three machines located at two leadership computing facilities: Intrepid (IBM BlueGene/P) from Argonne National Laboratory and Jaguar (Cray XT5 and XT4) from Oak Ridge National Laboratory. Table 3 shows their specifications and the compilers we used for all experiments. All systems have private L1 and L2 caches and shared L3 cache per node.

Intrepid is the primary system in the ANL Leadership Computing Facility. It is an IBM Blue Gene/P system with 40,960 quad-core compute nodes (163,840 processors) and 80 terabytes of memory. Its peak performance is 557 teraflops. BG/P compute 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. Figure 4 illustrates that each PowerPC 450 core is coupled to a small private prefetching L2 cache which is to prefetch streams of data from the shared on-chip L3 cache. The L3 cache interfaces to two on-chip DDR2 memory controllers, which directly control 2 GB of off-chip DDR2 DRAM.

Jaguar is the primary system in the ORNL Leadership Computing Facility (OLCF). It consists of two partitions: XT5 and XT4 partitions. The Jaguar XT5 partition (JaguarPF) contains 18,688 compute nodes in addition to dedicated login/service nodes. Each compute node contains dual hex-core AMD Opteron 2435 (Istanbul) processors running at 2.6GHz, 16GB of DDR2-800 memory, and a SeaStar 2+ router. The resulting partition contains 224,256 processing cores, 300TB of memory, and a peak performance of 2.3 petaflop/s. The Jaguar XT4 partition (Jaguar) contains 7,832 compute nodes in addition to dedicated login/service nodes. Each compute node contains a quad-core AMD Opteron 1354 (Budapest) processor running at 2.1 GHz, 8 GB of DDR2-800 memory, and a SeaStar2 router. The resulting partition contains 31,328 processing cores, more than 62 TB of memory, over 600 TB of disk space, and a peak performance of 263 teraflop/s. The SeaStar2+ router (XT5 partition) has a peak bandwidth of 57.6GB/s, while the SeaStar2 router (XT4 partition) has a peak bandwidth of 45.6GB/s. The routers are connected in a 3D torus topology, which provides an interconnect with very high bandwidth, low latency, and extreme scalability.

Table 3. Specifications of three multicore cluster architectures

Configurations	JaguarPF (XT5)	Jaguar (XT4)	BlueGene/P
Total Cores	224,256	31,328	163,840
Total Nodes	18,688	7,832	40,960
Cores/Socket	6	4	4
Cores / Node	12	4	4
CPU type	AMD 2.6GHz hex-core	AMD 2.1GHz quad-core	IBM PowerPC 850MHz quad-core
Memory/Node	16GB	8GB	2GB
L1 Cache/Core, private	64 KB	64 KB	32KB
L2 Cache/Core, private	512KB	512KB	4 128Byte-line buffers
L3 Cache/Socket, shared	6MB	2MB	8MB
Compiler	ftn	ftn	mpixf77_r
Compiler Options	-O3 -mp=nonuma -fastsse	-O3 -mp=nonuma -fastsse	-O3 –qsmp=omp -qarch=450d -qtune=450

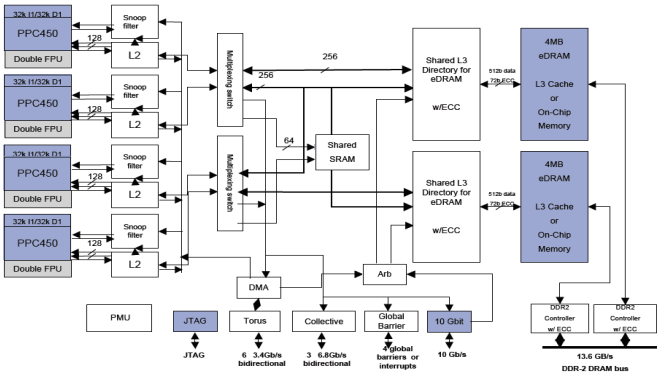

Figure 4. BlueGene/P compute chip architecture [12]

Table 4. Performance (seconds) comparison for SP with class C on BlueGene/P

#nodes	#cores	Hybrid SP	MPI SP	% Improvement
1 node	4	2150.33	2713.59	20.76%
4 nodes	16	599.25	684.35	12.44%
9 nodes	36	266.79	309.51	13.80%
16 nodes	64	152.26	182.91	16.76%
25 nodes	100	104.62	114.18	8.37%

Table 5. Communication (seconds) and performance counters' data for SP with class C on BlueGene/P

#cores	SP Type	Max MPI comm (s)	Min MPI comm (s)	Median MPI comm (s)	D+I Cache Hit rate	Dcache hit rate	L2 hit rate
4	Hybrid	0	0	0	66.54%	57.95%	99.30%
	MPI	7.421	6.263	6.715	64.91%	56.10%	99.18%
16	Hybrid	12.074	11.793	11.811	67.46%	58.79%	98.04%
	MPI	19.618	13.689	17.9	66.76%	57.82%	97.92%
36	Hybrid	11.055	10.515	10.836	69.59%	60.85%	96.82%
	MPI	17.95	14.143	15.818	66.88%	57.69%	96.47%
64	Hybrid	8.907	8.536	8.769	70.63%	61.92%	96.35%
	MPI	15.996	15.053	15.606	69.83%	60.98%	94.03%
100	Hybrid	8.251	7.838	8.03	71.57%	62.66%	95.16%
	MPI	11.803	10.76	11.23	70.34%	61.52%	93.68%

Table 6. DMA performance for the MPI SP with class C on 4 cores of BlueGene/P

Event	Value
BGP_DMA_PACKETS_INJECTED	27007185
BGP_DMA_DESCRIPTORS_READ_FROM_L3	38565
BGP_DMA_FIFO_PACKETS_RECEIVED	19293
BGP_DMA_COUNTER_PACKETS_RECEIVED	26978256
BGP_DMA_REMOTE_GET_PACKETS_RECEIVED	9636
BGP_DMA_IDPU_READ_REQUESTS_TO_L3	178245212
BGP_DMA_READ_VALID_RETURNED	229368255
BGP_DMA_ACKED_READ_REQUESTS	118114827
BGP_DMA_CYCLES_RDPU_WRITE_ACTIVE	239357485
BGP_DMA_WRITE_REQUESTS_TO_L3	215840571

Table 7. Number of writes from L2 to memory and network for SP with class C on 4 cores of BlueGene/P

SP Type	Event	Value
MPI	BGP_PU0_L2_MEMORY_WRITES	527939512412
	BGP_PU0_L2_NETWORK_WRITES	579639741285
	BGP_PU1_L2_MEMORY_WRITES	527936815997
	BGP_PU1_L2_NETWORK_WRITES	579639742161
Hybrid	BGP_PU0_L2_MEMORY_WRITES	420013522102
	BGP_PU0_L2_NETWORK_WRITES	473858864042
	BGP_PU1_L2_MEMORY_WRITES	422820173869
	BGP_PU1_L2_NETWORK_WRITES	473858865623

Table 8. Number of stores and fetches between L3 to memory (DDR) for SP with class C on 4 cores of BlueGene/P

SP Type	Event	Value
MPI	BGP_L3_M0_MH_DDR_STORES	5661830187
	BGP_L3_M0_MH_DDR_FETCHES	10938350591
	BGP_L3_M1_MH_DDR_STORES	5662172940
	BGP_L3_M1_MH_DDR_FETCHES	10947266702
Hybrid	BGP_L3_M0_MH_DDR_STORES	5374630932
	BGP_L3_M0_MH_DDR_FETCHES	9974837821
	BGP_L3_M1_MH_DDR_STORES	5374966279
	BGP_L3_M1_MH_DDR_FETCHES	9977471347

4.2 Performance Analysis and Comparison

In this section, we run the hybrid SP and BT with class C, class D and class E on BlueGene/P and Jaguar (XT4) to compare their performance with that of their MPI counterparts. We focus on the following performance aspects: the total execution time, communication percentage, L1 cache hit rate, L2 cache hit rate, and performance counters.

4.2.1 Performance Comparison of SP

In this section, we compare the performance of the hybrid SP with that of the original MPI SP of NPB3.3 using large problem sizes: class C, class D and class E. The original MPI SP requires the

square number of cores for its execution. Note that we use one OpenMP thread per core and one MPI process per node for the execution of the hybrid SP in our experiments.

Table 4 shows the performance comparison for SP with class C on BlueGene/P, where the unit of the execution time is seconds. There is up to 20.76% performance improvement for the hybrid SP. This is significant. We use HPCT MPI Profiling and Tracing Library [9] and Universal Performance Counter (UPC) Unit [13] to collect MPI performance and performance counters' data shown in Table 5 to explain why this happened.

The MPI communication contributes very small portion of the total execution time as indicated in Tables 4 and 5; the hybrid SP has less MPI communication time of the MPI SP. This, however, is not the primary source for the 20.76% performance improvement on 4 cores. For the hybrid SP executed on 4 cores, we used one OpenMP thread per core and one MPI process on this node, resulting in very little MPI communication time. Table 6 provides the DMA performance for the MPI SP with class C on 4 cores of BlueGene/P; the data indicates that the intra-node MPI communication uses DMA shown in Figure 4; however, for the hybrid SP executed on 4 cores, these values are all zero.

As shown in Table 5, the hybrid SP has higher hit rates of the on-chip memory resources than the MPI SP because of using OpenMP within a node. This results in the better performance for the hybrid SP. As discussed in [8, 12], BlueGene/P UPC provides four performance counter groups with 256 events per group. We use some of performance counter events in group 0 to further analyze the performance improvement. For instance, we compare the performance of the hybrid SP with that of the MPI SP on 4 cores as shown in Tables 7 and 8. Table 7 provides the number of writes from PU0 (core 0)/PU1 (core 1) L2 to memory or network on BlueGene/P. We observe that the number of writes for the hybrid SP is much smaller than that for the MPI SP. The data traffic (writes from L2 to memory) for the hybrid is 20.44% less than that for the MPI SP. Further, the poor L2 cache behavior for the MPI SP significantly increases the amount of off-chip communication and degrades the performance. Table 8 provides the number of stores to DDR2 memory from L3 and the number of fetches from DDR2 memory to L3. These numbers for the hybrid SP are much smaller than that for the MPI as well. This indicates that the data traffic from L3 to off-chip DDR2 memory significantly increases for the MPI SP as compared to the hybrid SP. The memory traffic results in that the hybrid SP having 20.76% better performance than the MPI SP.

Although BlueGene/P UPC provides four performance counter groups with 256 events per group about the processors, cache, memory, and network subsystems, it is hard for a user to derive some performance metrics from them without deep understanding these performance counters. Fortunately, CrayPat [4] developed by Cray provides for better understanding of application performance using performance counters and MPI/OpenMP profiling. We use the derived performance metrics provided by CrayPat to explain how and why an application has such performance. We use CrayPat to give the performance insights of the hybrid SP and MPI SP executed on Jaguar (XT4).

Table 9. Performance (seconds) comparison for SP with class C on Jaguar (XT4)

#nodes	#cores	Hybrid SP	MPI SP	% Improvement
1 node	4	1088.06	1255.3	13.32%
4 nodes	16	268.96	322.73	16.66%
9 nodes	36	140.71	145.91	3.56%
16 nodes	64	66.26	56.01	-18.30%
25 nodes	100	40.13	34.7	-15.65%

Table 10. Communication percentage and performance counters' data for SP with class C on Jaguar

#cores	SP Type	% MPI	D1+D2 hit ratio	Mem to D1 Refill	Mem to D1 BW	% OMP
4	**Hybrid**	0	98.00%	19.509M/s	1190.734 MB/s (14.68%)	0.10%
	MPI	0.80%	97.90%	16.972M/s	1038.318 MB/s (baseline)	---
16	**Hybrid**	1.20%	98.30%	19.108M/s	1166.232 MB/s (19.55%)	0.10%
	MPI	2.40%	98.20%	15.983M/s	975.502MB/s (baseline)	---
36	**Hybrid**	2.40%	98.60%	16.809M/s	1025.928 MB/s (0.7%)	0.10%
	MPI	6.70%	98.20%	16.692M/s	1018.815 MB/s (baseline)	---
64	**Hybrid**	4.2%	98.50%	19.163M/s	1169.631MB/s (-12.31%)	0.20%
	MPI	12.0%	98.50%	21.852M/s	1333.752MB/s (baseline)	---
100	**Hybrid**	7.50%	98.80%	18.262M/s	1114.597MB/s (-0.29%)	0.40%
	MPI	19.50%	98.90%	18.315M/s	1117.880MB/S (baseline)	---

Table 9 presents the performance comparison of SP for class C on Jaguar (XT4). The data indicates up to 16.66% performance improvement for the hybrid SP on up to 36 cores. With the increase of number of cores for the fixed problem size of class C, the workload per core decreases. The number of iteration for most loops also decreases with decreasing workload. For instance, the loop size (ksize-end(3,c) – start(3,c)) for the code segment of x_solve.f mentioned in Section 3 decreases from 161 to 33 with increasing the number of cores from 4 to 100 for the hybrid SP (with one MPI process per node and one OpenMP thread per core).

Further, Table 10 provides the detailed performance data for Table 9, where "% MPI" indicates the MPI communication percentage; "% OMP" indicates the OpenMP overhead percentage. "D1+D2 hit ratio" is the hit ratio for the L1 data cache and L2 data cache, which provides 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 the L1 cache. "Mem to D1 Refill" indicates the number of refill per core from memory to D1 per second (M/s: Million/s); "Mem to D1 BW" indicates the bandwidth (MB/s) per core for the data traffic between memory and L1 data cache. We observe that when the hybrid SP outperforms its MPI counterpart on 4, 16, 36 cores, the hybrid SP has much lower MPI communication percentage and small OpenMP overhead percentage. The hybrid SP also has higher D1+D2 hit ratio, larger number of refills per second, and larger Memory to D1 bandwidth per core.

It is noted that the Memory to D1 bandwidth for the hybrid SP increases by 14.68% on 4 cores, 19.55% on 16 cores, and 0.7% on 36 cores. These percentages are similar to that for performance improvement shown in Table 9. This indicates that the Memory to D1 bandwidth per core is the primary source of the performance improvement for the hybrid SP. We also observe that when the MPI SP outperforms the hybrid SP on 64 and 100 cores, the Memory to D1 bandwidth per core is also the primary source of the performance degradation for the hybrid SP. Although the hybrid SP has less communication overhead than MPI SP, it is at the expense of introducing OpenMP overhead due to thread creation and increased memory bandwidth contention because of shared L3 and memory. When the workload per core and the loop sizes decrease, the impact is significant when the decreased loop size is less than the number of OpenMP threads per node; this results in the under utilization the intranode cores.

Based on the performance trend as discussed above, we observe that for the hybrid SP with the given problem size, when increasing the number of cores, the MPI SP outperforms the hybrid SP on both BlueGene/P and Jaguar. As we discussed in Table 4, the hybrid SP outperformed the MPI SP by up to 20.76% on up to 100 cores. However, the MPI SP outperforms the hybrid SP on 144 cores or more as shown in Figure 5. The loop size (ksize-end(3,c) – start(3,c)) for the code segment of x_solve.f mentioned in Section 3 decreases from 161 to 27 with increasing the number of cores from 4 to 144.

Similarly, for the problem size class D, Figures 6 and 7 indicate that the hybrid SP outperforms the MPI SP by up to 15.42% on up to 576 cores on BlueGene/P, and by up to 7.06% on up to 256 cores on Jaguar. However, the MPI SP outperforms the hybrid on 900 cores or more on BlueGene/P and on 324 cores or more on Jaguar. For the problem size class E, Figure 8 indicates that the hybrid SP outperforms the MPI SP by up to 14.61% on up to 4096 cores on BlueGene/P, however, the MPI SP outperforms the hybrid on 6400 cores or more.

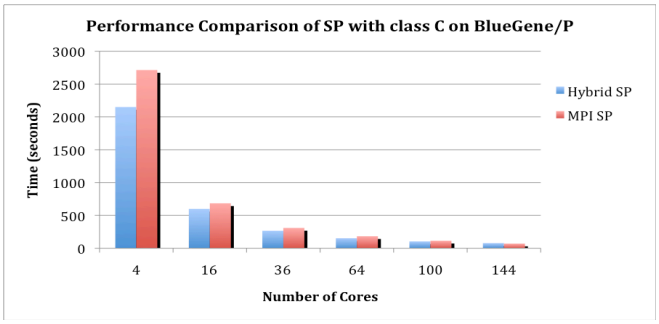

Figure 5. Performance comparison of SP with class C on BlueGene/P

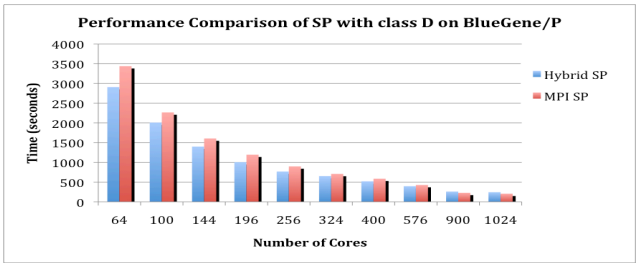

Figure 6. Performance comparison of SP with class D on BlueGene/P

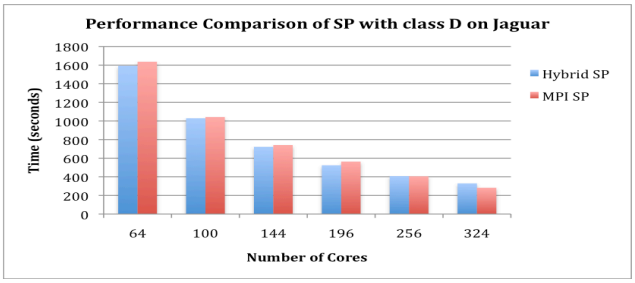

Figure 7. Performance comparison of SP with class D on Jaguar

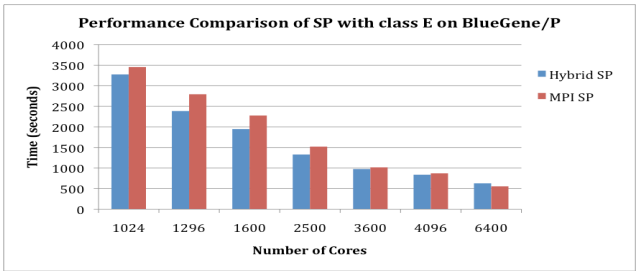

Figure 8. Performance comparison of SP with class E on BlueGene/P

4.2.2. Performance Comparison of BT

Again, we use CrayPat to collect the detailed performance data for our analysis of BT. We observe that BT has similar performance comparison as SP with increasing number of cores. As shown in Table 11, although the hybrid BT with class C outperforms the MPI BT by up to 5.89% on up to 36 cores; after increasing the number of cores (such as 64 or 100 cores), the MPI BT outperforms the hybrid BT. This is especially the case when the total execution time for the BT is very small.

Table 12 presents the detailed performance data, obtained from CrayPat, about communication percentage and performance counters for the BT with class C. Although the hybrid BT reduces the MPI communication percentage very little, we observe that the Memory to D1 bandwidth is still the primary source of the performance improvement. This is the same as that for the SP. Notice that the number of Memory to D1 refills and the Memory to D1 bandwidth for the BT shown in Table 12 are much smaller than that for the SP shown in Table 10; it is noted that CrayPat has much larger instrumentation overhead for the BT than that for the SP. The average CrayPat overhead for the SP is 0.72% of the time; however, the average CrayPat overhead for the BT is 53.6% of the time. This impacts the measurement of the Memory to D1 bandwidth for the BT.

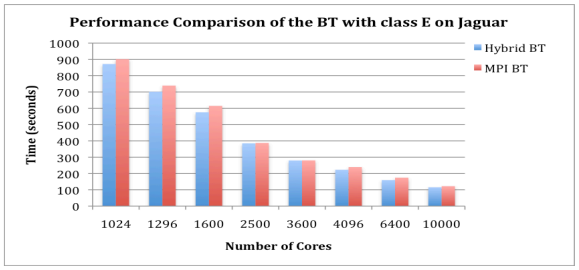

Figure 9. Performance comparison of the BT with class E on Jaguar

For the problem size class E, the hybrid BT outperforms the MPI BT by up to 8.58% on up to 10,000 cores on Jaguar shown in Figure 9. This is a big performance improvement on Jaguar. Figures 10 and 11 indicate that the hybrid BT with class C and class E has the similar performance as the MPI BT, and the MPI BT outperforms the hybrid BT a little bit.

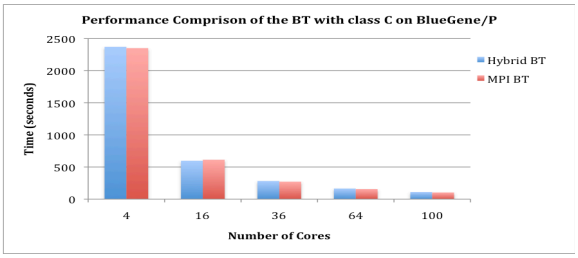

Figure 10. Performance comparison of the BT with class C on BlueGene/P

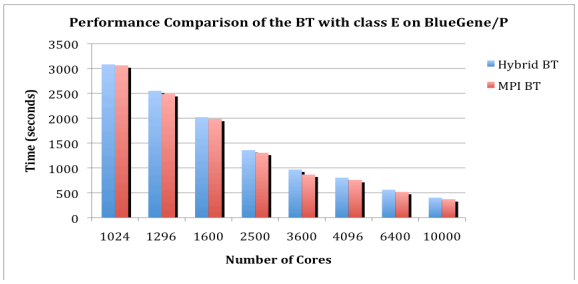

Figure 11. Performance comparison of the BT with class E on BlueGene/P

Table 11. Performance (seconds) comparison for BT with class C on Jaguar

#nodes	#cores	Hybrid BT	MPI BT	% Improvement
1 node	4	701.67	725.14	3.24%
4 nodes	16	166.71	172.94	3.60%
9 nodes	36	76.96	81.78	5.89%
16 nodes	64	45.31	44.94	-0.82%
25 nodes	100	28.66	28.64	-0.07%

Table 12. Communication percentage and performance counters' data for BT with class C on Jaguar

#cores	BT Type	% MPI	D1+D2 hit ratio	Mem to D1 Refill	Mem to D1 BW	% OMP
4	**Hybrid**	0	99.50%	5.832M/s	355.931MB/s (2.48%)	0.20%
	MPI	1.70%	99.40%	5.691M/s	347.331MB/s (baseline)	---
16	**Hybrid**	3.30%	99.50%	5.725M/s	349.448MB/s (3.36%)	0.10%
	MPI	7.10%	99.40%	5.539M/s	338.078MB/s (baseline)	---
36	**Hybrid**	4.20%	99.40%	5.968M/s	364.283MB/s (0.75%)	0.10%
	MPI	7.20%	99.40%	5.924M/s	361.586MB/s (baseline)	---
64	**Hybrid**	11.40%	99.40%	5.052M/s	308.330MB/s (-8.89%)	0.70%
	MPI	15.10%	99.50%	5.545M/s	338.411MB/s (baseline)	---
100	**Hybrid**	15.10%	99.40%	5.002M/s	305.305MB/s (-2.67%)	0.80%
	MPI	18.30%	99.50%	5.140M/s	313.696MB/s (baseline)	---

4.2.3 Performance of the Hybrid SP and BT on JaguarPF

As shown in Table 3, JaguarPF is a Cray XT5 system with dual hex-core AMD Opteron per node (12 cores per node). It is hard for NPB benchmarks to be executed on the system because NPB benchmarks require either power-of-two or square number of cores. However, the hybrid SP and BT can be executed by utilizing all cores per node with one OpenMP thread per core. Figures 12 and 13 show scalability of the hybrid SP and BT on JaguarPF. Figure 12 presents the execution time (seconds), and Figure 13 shows the total TFlops/s (TeraFlops/s) for the hybrid SP and BT on up to 4,9152 cores on JaguarPF. Notice that, for the class E problem size, when increasing the number of cores, the execution time does not reduce much because of increased communication overhead. The execution time stays almost flat on 30,000 cores or more for both the hybrid SP and BT because the hybrid SP and BT are strong scaling. Therefore, we believe that the current or future large-scale multi- or many-core supercomputers require weak-scaling, hybrid MPI/OpenMP benchmarks for better scalability.

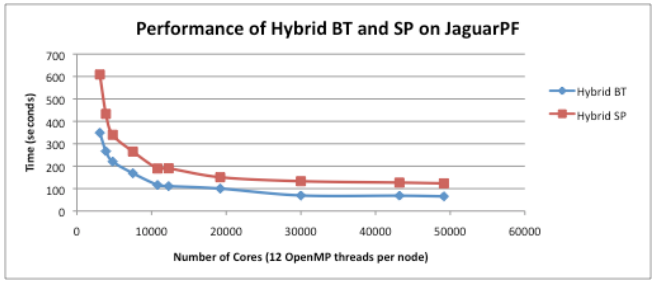

Figure 12. Execution time of Hybrid BT and SP with Class E on JaguarPF

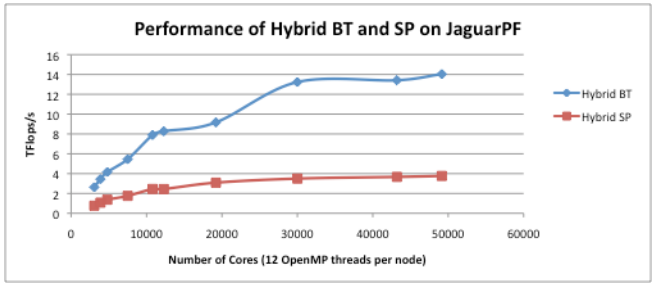

Figure 13. Total TFlops/s for Hybrid BT and SP with Class E on JaguarPF

4.3 Some Limitations of the Hybrid SP and BT

The combining of MPI and OpenMP implementations to construct hybrid SP and BT achieves multiple levels of parallelism and reduces the communication overhead of MPI at the expense of introducing OpenMP overhead due to thread creation and increased memory bandwidth contention. Because NPB benchmarks have the fixed algorithms and problem sizes, especially SP and BT, there are some limitations for the execution of the hybrid SP and BT with increasing the number of cores for different problem sizes. The number of OpenMP threads per node for the hybrid SP and BT is limited by number of cores per node in the underlying system, the underlying system software, as well as the loop size to which OpenMP parallelization is applied. For the given problem size, with increasing number of cores, some parallelized loop sizes become very small, which may cause more OpenMP overhead and memory bandwidth contention so that the execution time of the hybrid codes may become larger than that of their MPI counterparts (as shown in Tables 9 and 11). When increasing the number of cores, decreasing parallelized loop sizes may also cause some idle cores per node because the loop sizes are not larger than the number of OpenMP threads per node. It may also affect result verifications for each benchmark. So before running these hybrid benchmarks on a large-scale multicore cluster, their limitations should be examined.

5. Conclusions

In this paper, we implemented hybrid MPI/OpenMP implementations of SP and BT of MPI NPB 3.3, and compared the performance of the hybrid SP and BT with its MPI counterparts on three large-scale multicore clusters: BlueGene/P, Jaguar (Cray XT4) and JaguarPF (Cray XT5). For JaguarPF, which has 12 cores per node, the hybrid SP and BT can be executed by utilizing all cores per node on the system. It is, however, hard for NPB benchmarks to be executed on the system because NPB benchmarks require either power-of-two or square number of processors. We could not compare our hybrid SP and BT with those from NPB-MZ because of the different problem sizes for the same class. Our performance results show that the hybrid SP outperforms the MPI SP by up to 20.76%, and the hybrid BT outperforms the MPI SP by up to 8.58% on up to 10,000 cores. We also used performance tools and MPI trace libraries available on these clusters to investigate the performance characteristics of the hybrid SP and BT in detail. We observe that, in most cases, although the hybrid SP and BT have much lower MPI communication percentage, the memory to D1 bandwidth per core is the primary source of the performance improvement when comparing to their MPI counterparts.

Because NPB benchmarks have the fixed algorithms and problem sizes, especially SP and BT, there are some limitations for the execution of the hybrid SP and BT with increasing the number of cores. We observe that, for the hybrid SP and BT with the given problem size, when increasing the number of cores, the MPI SP and BT outperforms their hybrid counterparts on both BlueGene/P and Jaguar because of the decreased sizes of the parallelized loops, more OpenMP overhead and more memory bandwidth contention.

For further work, we will work on the hybrid MPI/OpenMP implementations for the rest of NPB3.3, and plan to release our hybrid MPI/OpenMP implementation of NPB3.3 in the near future. We believe that these hybrid MPI/OpenMP benchmarks will be beneficial to HPC communities.

Acknowledgements

This work is supported by NSF grant CNS-0911023. The authors would like to acknowledge Argonne Leadership Computing Facility at Argonne National Laboratory for the use of BlueGene/P and National Center for Computational Science at Oak Ridge National Laboratory for the use of Jaguar and JaguarPF under DOE INCITE project "Performance Evaluation and Analysis Consortium End Station", and Haoqiang Jin from NASA Ames Research Center for providing his BT code.

References

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

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

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

[4] Cray Performance analysis toolkit (CrayPat), http://www.nccs.gov/computing-resources/jaguar/ software/?software=craypat. Also see *Using Cray Performance Analysis Tools*, Cray Doc S-2376-41, 2007.

[5] 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.

[6] 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),* Sep. 2003.

[7] H. Jin and R. Van der Wijngaart, Performance Characteristics of the Multi-Zone NAS Parallel Benchmarks, *IPDPS'04*, 2004.

[8] G. Lakner, I. Chung, G. Cong, S. Fadden, N. Goracke, D. Klepacki, J. Lien, C. Pospiech, S. R. Seelam, and H. Wen, *IBM System Blue Gene Solution: Performance Analysis Tools*, Redbook, REDP-4256-01, November 2008.

[9] HPCT MPI Profiling and Tracing Library, https://wiki.alcf.anl.gov/index.php/HPCT_MPITRACE.

[10] NAS Parallel Benchmarks 3.3, http://www.nas.nasa. gov/Resources/Software/npb.html.

[11] NCCS Jaguar and JaguarPF, Oak Ridge National Laboratory,http://www.nccs.gov/computing-resources /jaguar/

[12] V. Salapura, K. Ganesan, A. Gara, M. Gschwind, J. Sexton, and R. Walkup, *Next-Generation Performance Counters: Towards Monitoring over Thousand Concurrent Events*, IBM Research Report, RC24351 (W0709-061), September 19, 2007.

[13] Universal Performance Counter (UPC) Unit and HPM library for BG/P, https://wiki.alcf.anl.gov/ index.php/Performance

[14] R. Van der Wijngaart and H. Jin, *NAS Parallel Benchmarks, Multi-Zone Versions*, NAS Technical Report NAS-03-010, July 2003.