International Journal of Networked and Distributed Computing, Vol. 1, No. 4 (November 2013), 213-225

# Performance Characteristics of Hybrid MPI/OpenMP Scientific Applications on a Large-scale Multithreaded BlueGene/Q Supercomputer

**Xingfu Wu**

*Department of Computer Science & Engineering, Texas A&M University*
*College Station, Texas 77843, USA*
*E-mail: wuxf@cse.tamu.edu*

**Valerie Taylor**

*Department of Computer Science & Engineering, Texas A&M University*
*College Station, Texas 77843, USA*
*E-mail: taylor@cse.tamu.edu*

Received 31 January 2013

Accepted 29 July 2013

**Abstract**

Many/multi-core supercomputers provide a natural programming paradigm for hybrid MPI/OpenMP scientific applications. In this paper, we investigate the performance characteristics of five hybrid MPI/OpenMP scientific applications (two NAS Parallel benchmarks Multi-Zone SP-MZ and BT-MZ, an earthquake simulation PEQdyna, an aerospace application PMLB and a 3D particle-in-cell application GTC) on a large-scale multithreaded BlueGene/Q supercomputer at Argonne National laboratory, and quantify the performance gap resulting from using different number of threads per node. We use performance tools and MPI profile and trace libraries available on the supercomputer to analyze and compare the performance of these hybrid scientific applications with increasing the number OpenMP threads per node, and find that increasing the number of threads to some extent saturates or worsens performance of these hybrid applications. For the strong-scaling hybrid scientific applications such as SP-MZ, BT-MZ, PEQdyna and PLMB, using 32 threads per node results in much better application efficiency than using 64 threads per node, and as increasing the number of threads per node, the FPU percentage decreases, and the MPI percentage (except PMLB) and IPC per core (except BT-MZ) increase. For the weak-scaling hybrid scientific application such as GTC, the performance trend (relative speedup) is very similar with increasing number of threads per node no matter how many nodes (32, 128, 512) are used.

*Keywords: Performance analysis, hybrid MPI/OpenMP, multithreaded, BlueGene/Q.*

## 1. Introduction

Many/multicore supercomputers provide a natural programming paradigm for hybrid MPI/OpenMP scientific applications. Current hybrid parallel programming paradigms such as hybrid MPI/OpenMP need to efficiently exploit the potential offered by such many/multicore supercomputers. When using these supercomputers to execute a given hybrid MPI/OpenMP application, one issue to be addressed is how many threads per node to use for efficient execution (Assume 1 MPI process per node for hybrid application execution). It is expected that the best number of threads per node is dependent upon the application characteristics and the system architectures. In this paper, we investigate how a hybrid application is sensitive to different memory access patterns, and quantify the performance gap resulting from using different number of threads per node for application execution on a large scale multithreaded BlueGene/Q supercomputer [1] at Argonne National Laboratory using five different hybrid MPI/OpenMP scientific

applications (two NAS Parallel benchmarks Multi-Zone SP-MZ and BT-MZ [4], an earthquake simulation PEQdyna [20], an aerospace application PMLB [19] and a 3D particle-in-cell application GTC [2]).

There is a lot of research work in this area, which mainly focused on multithreaded applications on a single compute node [5, 11, 8]. They found that increasing the number of threads may saturate or worsen performance of multithread applications because concurrently executing threads compete for shared data (data-synchronization) and shared resources (off-chip bus). Some techniques in choosing the best number of threads for the applications are analyzed in [11]. In this paper, we focus on large-scale hybrid MPI/OpenMP applications on a large-scale supercomputer which consists of many compute nodes, and investigate their performance characteristics as increasing number of threads per node on different large number of compute nodes.

Levesque et al. [6] observed from the AMD architectural discussion that when excluding messaging performance, the primary source of contention when moving from single core to dual core is memory bandwidth. In our previous work [19, 18], we also found that memory bandwidth contention is the primary source of performance degradation for L2 shared architectures such as CrayXT4 and IBM Power4 and Power5 systems using NAS Parallel benchmarks and large-scale scientific applications such as GTC as increasing the number of cores per node.

Other work in this area has focused on using all processor cores per node. Petrini et al. [10] found that application execution times may vary significantly between 3 processors per node and 4 processors per node on a large scale supercomputer, ASCI Q. In our previous work [17, 18, 19], we conduct an experimental performance analysis to identify the application characteristics that affect processor partitioning and to quantify the performance difference among different processor partitioning schemes.

As we found in [20], the hybrid MPI/OpenMP earthquake application is memory bound, and using 12 OpenMP threads per MPI process on Cray XT5 (with 12 cores per node) at Oak Ridge National Laboratory has more OpenMP overhead than using 4 OpenMP threads per MPI process on Cray XT4 (with 4 cores per node) at Oak Ridge National Laboratory for the hybrid execution on the same number of nodes with 1 MPI process per node although Cray XT5 is much faster than Cray XT4. This motivates us for this work.

The experiments conducted for this work utilize a multicore supercomputers BlueGene/Q [1] at Argonne National Lab (ANL) has 16 compute cores per node. We investigate the performance characteristics of the five hybrid scientific applications, and quantify the performance gap resulting from using different number of threads per node. We use performance tools and MPI profile and trace libraries available on the supercomputer [15] to analyze and compare the performance of the hybrid scientific applications as increasing the number OpenMP threads per node. For the strong-scaling hybrid scientific applications such as SP-MZ, BT-MZ, PEQdyna and PLMB, using 32 threads per node results in much better application efficiency than using 64 threads per node, and with increasing the number of threads per node, the FPU (Floating Point Unit) percentage decreases, and the MPI percentage (except PMLB) and IPC (Instructions per cycle) per core (except BT-MZ) increase. For the weak-scaling hybrid scientific application such as GTC, the performance trend (relative speedup) is very similar with increasing number of threads per node no matter how many nodes (32, 128, 512) are used. We also find increasing the number of threads to some extent saturates or worsens performance of these hybrid applications.

The remainder of this paper is organized as follows. Section 2 discusses the architecture and memory hierarchy of the BlueGene/Q supercomputer. Section 3 describes hybrid MPI/OpenMP applications used. Section 4 analyzes application performance characteristics with different number of threads per node in detail. Section 5 concludes this paper.

In the remainder of this paper, all experiments were executed multiple times to ensure consistency of the performance data. Prophesy system [12] and IBM HPCT and MPI profiling and trace libraries [14] are used to collect all application performance data. Notice that, **for a hybrid MPI/OpenMP application execution, one MPI process per node is applied to all our experiments**.

## 2. Execution Platforms

In this section, we briefly describe a large-scale multithreaded BlueGene/Q supercomputer Vesta [1] used for our experiments. The supercomputer is one same rack of world Top 4 supercomputer Mira at Argonne Leadership Computing Facility [1, 13] at Argonne National Laboratory for early access and use. It has 1024 nodes with 16 compute cores per node (total 16,384 compute cores) and 16 GB memory per node. Each node has a BlueGene/Q compute chip. Note that the compiler option “-O3 -qsmp=omp” is applied to

compile all our hybrid scientific applications.

A die photograph of the BlueGene/Q compute chip in Figure 1 shows 18 processor units (PU00 to PU17) surrounding a large Level-2 (L2) cache that occupies the center of the chip. The chip is a System-on-a-Chip (SOC) ASIC with 18 4-way SMT (Symmetric MultiThreading) PowerPC A2 cores clocked at 1.6 GHz. Of the 18 cores, 16 are exposed to user applications, 1 is used for system software functionality, and 1 is for yield purposes. A quad floating point unit is associated with each core shown in Figure 2, and instantiates four copies of a fused multiply-add data flow (MAD), creating a four-way SIMD floating-point microarchitecture. The first level (L1) data cache of the A2 core is 16 KB, 8-way set associative, with 64 B lines. The L1 instruction cache is 16 KB, 4-way set associative.

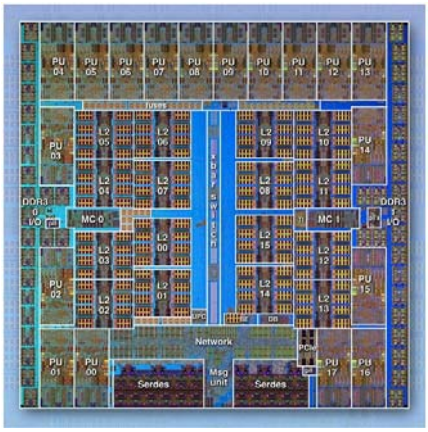


**Figure 1. A die photograph of BlueGene/Q Chip [3]**

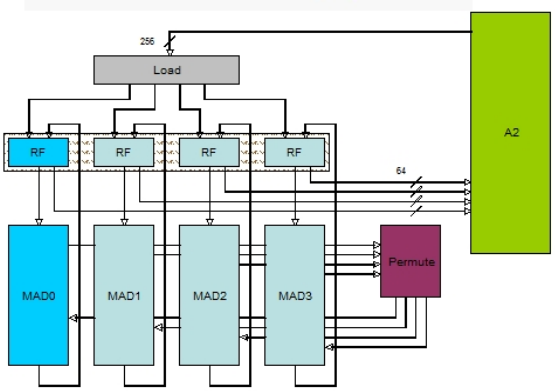


**Figure 2. Quad FPU in each BlueGene/Q Core [3]**

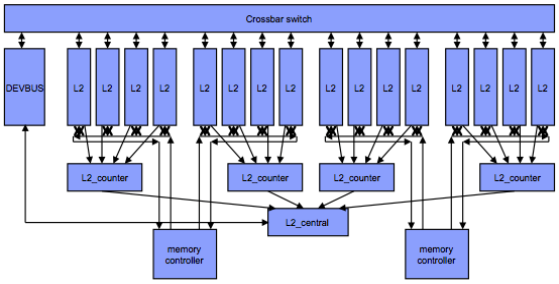


**Figure 3. L2 cache of BlueGene/Q Compute Chip [9]**

The crossbar switch is the central connection structure among all the PUs, the L2 cache, the networking logic, and various low-bandwidth units on the chip. All processor cores share the L2 cache shown in Figure 3. To provide sufficient bandwidth, the cache is split into 16 slices. Each slice is 16-way set associative, operates in write-back mode, and has 2MB capacity. Physical addresses are scattered across the slices via a programmable hash function to achieve uniform slice utilization. The L2 cache provides hardware assist capabilities to accelerate sequential code as well as thread interactions. There is no L3 cache on BlueGene/Q chip.

The Blue Gene/Q network consists of a set of compute nodes (BQC chip plus memory) arranged in a five- dimensional (5D) torus configuration. On compute nodes, 10 of the 11 chip-to-chip communication links are used to build the 5D torus. A subset of the compute nodes, called bridge nodes, use the 11th communication link to connect with an I/O node.

## 3. Hybrid MPI/OpenMP Scientific Applications

In this section, we describe the scientific applications that are used throughout this paper. These applications include the two NAS Parallel Benchmarks Multi-Zone (version 3.3) BT-MZ and SP-MZ [4], and three large-scale scientific applications PMLB [19], PEQdyna [20] and GTC [2]. Table 1 provides an overview of these applications. The first four applications (BT-MZ, SP-MZ, PMLB, PEQdyna) are strong scaling, and the last one, GTC, is weak scaling. These hybrid MPI/OpenMP applications are written in different languages such as Fortran 77, Fortran 90 or C.

**Table 1. Overview of Hybrid HPC Applications**

| Application | Discipline | Problem Size | Languages |
|---|---|---|---|
| BT-MZ | CFD | Class C<br>Class D | Fortran77, MPI/OpenMP |
| SP-MZ | CFD | Class C<br>Class D | Fortran77, MPI/OpenMP |
| PMLB | CFD | 256x256x256<br>512x512x512 | C, MPI/OpenMP |
| PEQdyna | Geophysics | 100m<br>200m | Fortran90, MPI/OpenMP |
| GTC | Magnetic Fusion | 100 particles per cell | Fortran90, MPI/OpenMP |

The BT-MZ benchmark [4] computes discrete solutions of the unsteady, compressible Navier-Stokes equations in three spatial dimensions using the Block Tri-diagonal (BT) algorithm. The SP-MZ benchmark [4] computes discrete solutions of the unsteady, compressible Navier-Stokes equations in three spatial dimensions using the Scalar Penta-diagonal (SP) algorithm. Both are computational fluid dynamics (CFD) applications. The problem sizes used for both are Class C (480x320x28) and Class D (1632x1216x34).

The PMLB application [19] is a CFD aerospace application using a Parallel Multiblock Lattice Boltzmann (PMLB) method, and is written in C, MPI and OpenMP. The problem sizes used in the experiments are 256x256x256 and 512x512x512.

The PEQdyna [20] is a parallel finite element earthquake rupture simulation using the Southern California Earthquake Center (SCEC) TPV210 benchmark, which is the convergence test of the benchmark problem TPV10. In TPV10, a normal fault dipping at 60° (30 km long along strike and 15 km wide along dip) is embedded in a homogeneous half space. The application is written in Fortran 90, MPI and OpenMP. The problem sizes used are 100m and 200m, which are the resolution of the 3D mesh.

The Gyrokinetic Toroidal code (GTC) [2] 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 DOE SciDAC fusion microturbulence code written in Frotran90, MPI and OpenMP. The GTC application is executed in weak scaling to keep a constant workload per processor as the number of processors increase using 100 particles per cell and 100 time steps.

## 4. Performance Analysis

In this section, we use these five hybrid applications to investigate their performance characteristics on the multithreaded BlueGene/Q supercomputer Vesta.

### *4.1. Performance Analysis of SP-MZ*

As we described in Section 2, each node of the BlueGene/Q supercomputer has 16 compute cores. Each compute core supports four threads. Figures 4 and 5 show the performance comparison of SP-MZ with class D and C on 32 and 128 nodes with different number of threads per node. Both have similar performance trend. The relative speedup increases using up to 32 OpenMP threads per node, then significantly decreases using 64 threads per node.

For instance, Table 2 presents performance (per node) comparison of SP-MZ with class D on 128 nodes. These derived metrics are for the performance per node by using HPCT and MPI profiling and trace libraries [15]. IPC per core stands for the instructions per cycle completed per core; GFlops stands for the total weighted GFlops for the node. L1 D-cache, L1P Buffer and L2 Cache stand for the hit rates for the loads that hit in L1 D-cache, L1 prefetch (L1P) buffer, and L2 cache, respectively. When a load misses in the L1 D-cache, the next place to look is the L1P buffer; if there is a miss in the L1P buffer, the request goes to L2 cache, and if there is a L2 cache miss, the request goes to the DDR memory. The total DDR traffic includes all load and store activity, which is often dominated by stream prefetching rather than demand loads [14]. %MPI stands for the ratio of the median MPI communication time to the total application execution time.

From Table 2, with increasing the number of OpenMP threads per core from 1 to 4, the application execution time for using 32 threads decreases from 37.39s to 30.07s, however, the application execution time for using 64 threads increases from 30.07s to 41.58s. This results in the significant decrease in speedup for using 64 threads shown in Figure 5. For the instruction mix, the FPU percentage decreases a little bit with increasing the number of threads per core because of the overhead in the threading, for example, instructions executed by threads are basically waiting for synchronization and/or serialization.

As shown in Figure 2, each BG/Q compute core supports 4 hardware threads, this results in the increase in IPC per core shown in Table 2 with increasing the number of threads per node. Because of the fixed total float-point operations per node, the GFlops per node for

using 32 threads is the highest. It is interesting to see that using 32 threads has a relatively low hit rate in the L1 data cache, but benefits from the highest hit rates in the L1P buffer and L2 cache, and the highest memory bandwidth (DDR traffic).

Notice that the number of OpenMP threads per node for the hybrid SP-MZ is limited by number of cores per node in the underlying system and the loop sizes to which OpenMP parallelization is applied. When increasing the number of cores for a given problem size, decreasing parallelized loop sizes may cause some idle cores per node because the loop sizes are not larger than the number of OpenMP threads per node.

For instance, after digging into source codes of SP-MZ (version 3.3), we find that OpenMP parallelization is applied to the Z dimension of the 3D mesh, however, the problem size for Class D is 1632x1216x34 so that the maximum number of threads is 34; the problem size for Class C is 480x320x28 so that the maximum number of threads is 28. This is the limitation, which limits potential for OpenMP speed-up and causes more idle threads when using 64 threads per node. The threads that have no work to do are just spinning, that results in increasing number of instructions processed by the integer/load/store units. That can also affect MPI performance because using the threaded MPI library can affect scheduling of messaging threads. Overall, this causes the big decrease in the FPU percentage and the big increase in the MPI percentage for using 64 threads. So before running these hybrid benchmarks on a large-scale multicore supercomputer, these limitations should be examined.

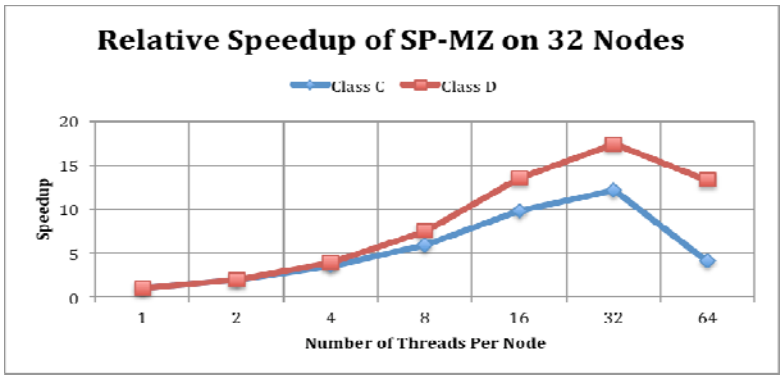


**Figure 4. Performance comparison of SP-MZ on 32 Nodes with different number of threads per node**

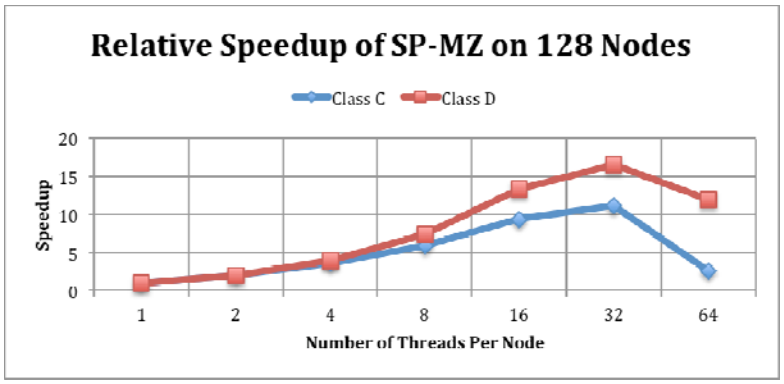


**Figure 5. Performance comparison of SP-MZ on 128 Nodes with different number of threads per node**

**Table 2. Performance (per node) comparison of SP-MZ with Class D on 128 Nodes**

| #Threads | 16 | 32 | 64 |
|---|---|---|---|
| **Runtime (s)** | 37.39 | 30.07 | 41.58 |
| **Instruction Mix** | FPU:40.50%<br>FXU:59.50% | FPU:36.03%<br>FXU:63.97% | FPU:22.54%<br>FXU:77.46% |
| **IPC per core** | 0.3654 | 0.5149 | 0.6015 |
| **GFlops** | 6.260 | 7.847 | 5.687 |
| **L1 D-cache** | 90.42% | 89.06% | 92.34% |
| **L1P Buffer** | 7.81% | 8.56% | 5.46% |
| **L2 Cache** | 1.45% | 1.79% | 1.74% |
| **DDR traffic (Bytes/cycle)** | 1.637 | 4.511 | 3.802 |
| **%MPI** | 1.89% | 2.59% | 7.60% |

### *4.2. Performance Analysis of BT-MZ*

Figures 6 and 7 show the performance comparison of BT-MZ with class D and C on 32 and 128 nodes with different number of threads per node. We find the similar performance trend as SP-MZ has, that is, the relative speedup increases using up to 32 OpenMP threads per node, then significantly decreases using 64 threads per node.

Table 3 presents performance (per node) comparison of BT-MZ with class D on 128 nodes. Note that using 64 threads causes the large performance degradation in the total execution time and MPI communication time, because OpenMP parallelization is applied to the Z dimension of the 3D mesh in BT-MZ, however, the problem size for Class D is 1632x1216x34 so that the maximum number of threads is 34; the problem size for Class C is 480x320x28 so that the maximum number of threads is 28. Similarly, this limits potential for OpenMP speed-up and causes more idle threads (30) when using 64 threads per node. The threads that have no work to do are just spinning, that results in increasing number of instructions processed by the integer/load/store units. That can also affect MPI performance because using the threaded MPI library can affect scheduling of messaging threads. Overall, this causes the big increase in the MPI percentage for using 64 threads.

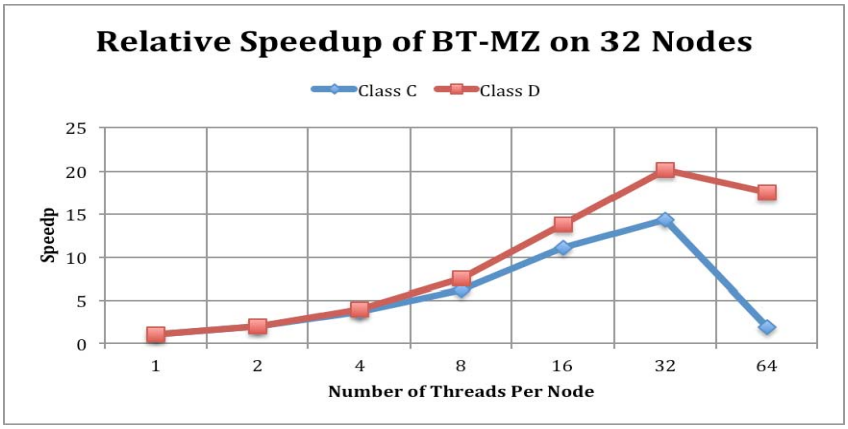


**Figure 6. Performance comparison of BT-MZ on 32 Nodes with different number of threads per node**

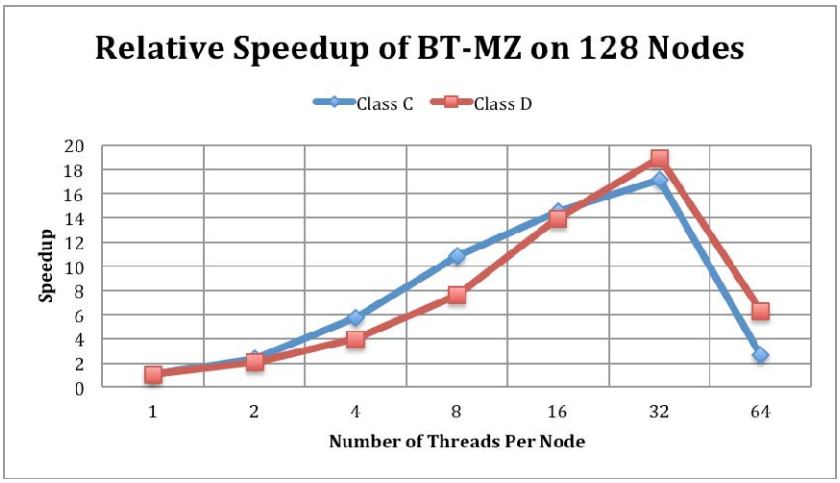


**Figure 7. Performance comparison of BT-MZ on 128 Nodes with different number of threads per node**

**Table 3. Performance (per node) comparison of BT-MZ with Class D on 128 Nodes**

| #Threads | 16 | 32 | 64 |
|---|---|---|---|
| **Runtime (s)** | 57.28 | 42.16 | 127.44 |
| **Instruction Mix** | FPU:37.86%<br>FXU:62.14% | FPU:36.89%<br>FXU:63.11% | FPU:30.99%<br>FXU:69.01% |
| **IPC per core** | 0.4234 | 0.5928 | 0.2341 |
| **GFlops** | 7.183 | 9.855 | 3.268 |
| **L1 D-cache** | 94.77% | 94.50% | 95.08% |
| **L1P Buffer** | 3.89% | 3.92% | 3.22% |
| **L2 Cache** | 1.26% | 1.11% | 1.28% |
| **DDR traffic (Bytes/cycle)** | 0.448 | 3.093 | 1.064 |
| **%MPI** | 5.50% | 8.03% | 68.26% |

### *4.3. Performance Analysis of PEQdyna*

Figures 8 and 9 show the performance comparison of the PEQdyna with the element sizes of 100m or 200m on 32 and 128 nodes with different number of threads per node. With increasing number of OpenMP threads per node, the relative speedup increases for 32 nodes shown in Figure 8, however, the efficiency (0.65 or higher) for using 32 threads per node is much higher than that (0.46 or less) using 64 threads. The relative speedup reaches the maximum using 32 threads per node for 128 nodes shown in Figure 9.

From Table 4, we see that the instruction mix is more heavily weighted on the integer side (FXU), and MPI communication time dominates the total execution time on 128 nodes. For the strong scaling application, with increasing the number of threads per node from 16 to 64, the MPI percentage increases from 73.6% to 85.7% because the pure application computation time decreases and the threaded MPI library was used.

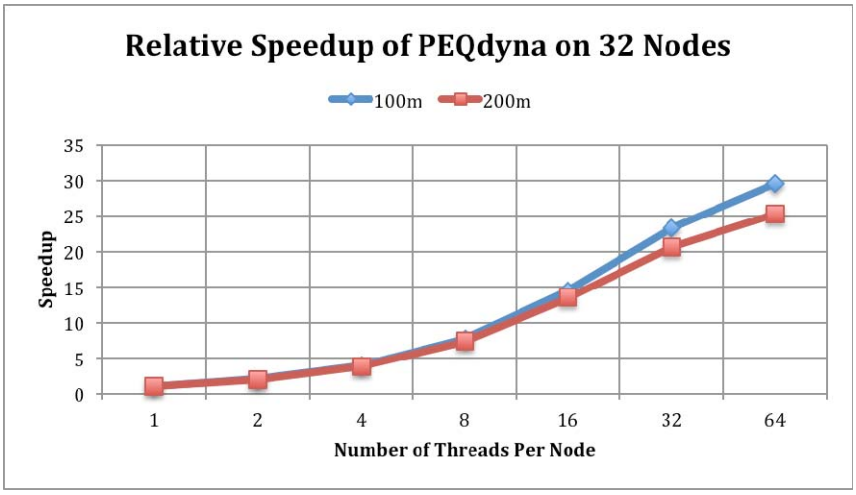


**Figure 8. Performance comparison of PEQdyna on 32 Nodes with different number of threads per node**

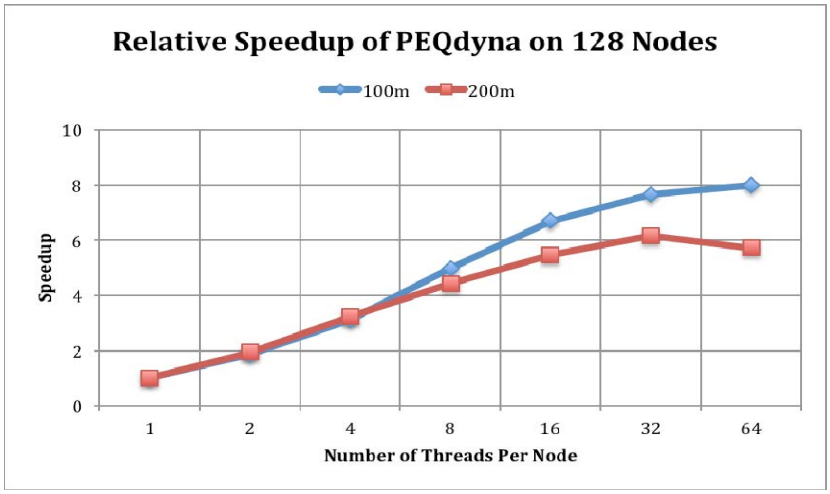


**Figure 9. Performance comparison of PEQdyna on 128 Nodes with different number of threads per node**

**Table 4. Performance (per node) comparison of PEQdyna with 200m on 128 Nodes**

| #Threads | 16 | 32 | 64 |
|---|---|---|---|
| **Runtime (s)** | 110.29 | 97.81 | 105.48 |
| **Instruction Mix** | FPU: 10.57%<br>FXU:89.43% | FPU:10.36%<br>FXU:89.64% | FPU:9.45%<br>FXU:90.55% |
| **IPC per core** | 0.0999 | 0.1149 | 0.1172 |
| **GFlops** | 0.375 | 0.423 | 0.392 |
| **L1 D-cache** | 98.00% | 98.28% | 95.93% |
| **L1P Buffer** | 0.94% | 0.66% | 0.48% |
| **L2 Cache** | 1.03% | 1.04% | 3.55% |
| **DDR traffic (Bytes/cycle)** | 0.124 | 0.139 | 0.129 |
| **%MPI** | 73.6% | 82.9% | 85.7% |

### *4.4. Performance Analysis of PMLB*

Figures 10 and 11 show the performance comparison of PMLB with the problem sizes of 256x256x256 and 512x512x512 on 32 and 128 nodes with different number of threads per node. With increasing number of OpenMP threads per node, the relative speedup increases, however, the relative efficiency is very low for using 16 threads or more because of the dominated MPI communication times.

From Table 5, we see that the instruction mix is more heavily weighted on the integer side, and MPI communication time dominates the total execution time on 128 nodes. The application performance (runtime) has smaller improvement by using more threads. This results in the very low efficiency.

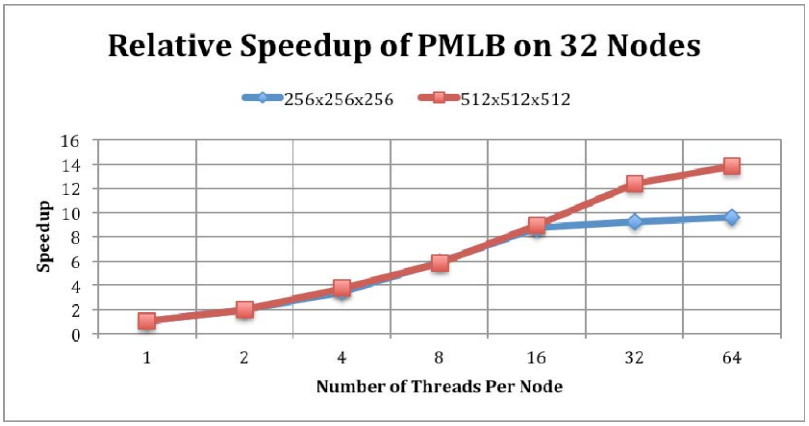


**Figure 10. Performance comparison of PMLB on 32 Nodes with different number of threads per node**

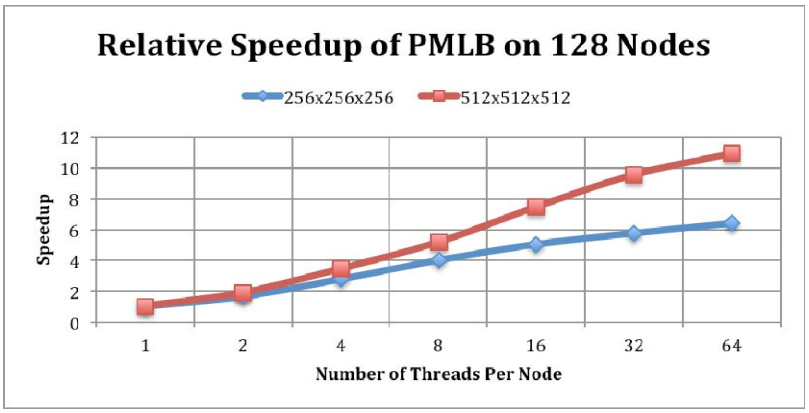


**Figure 11. Performance comparison of PMLB on 128 Nodes with different number of threads per node**

**Table 5. Performance (per node) comparison of PMLB with the problem size of 256x256256 on 128 Nodes**

| #Threads | 16 | 32 | 64 |
|---|---|---|---|
| **Runtime** | 8.68 | 7.59 | 6.82 |
| **Instruction Mix** | FPU:25.87%<br>FXU:74.13% | FPU:25.74%<br>FXU:74.26% | FPU:22.72%<br>FXU:77.28% |
| **IPC per core** | 0.0523 | 0.0604 | 0.0683 |
| **GFlops** | 0.510 | 0.585 | 0.582 |
| **L1 D-cache** | 85.60% | 85.91% | 87.07% |
| **L1P Buffer** | 6.81% | 6.57% | 5.73% |
| **L2 Cache** | 7.55% | 7.43% | 7.11% |
| **DDR traffic (Bytes/cycle)** | 0.017 | 0.038 | 0.049 |
| **%MPI** | 77.0% | 86.50% | 73.56% |

### *4.5. Performance Analysis of GTC*

Figure 12 shows the performance comparison of GTC on 32, 128, and 512 nodes with different number of threads per node. With increasing number of OpenMP threads per node, the relative speedup increases. Because the application is weak scaling (with number of MPI processes), the performance trend (relative speedup) is very similar with increasing number of threads per node no matter how many nodes (32, 128, 512) are used. This is a very interesting. However, this application could not be executed using 32 or more threads per node because this causes the execution crash.

From Table 6, because of the same amount of workload per node, with increasing the number of threads, the execution time decreases significantly, and the GFlops per node, the memory bandwidth (DDR traffic) and the MPI percentage increase a little bit. However, it is interesting to see that IPC per core decreases with increasing the number of threads, and the hit rates in L1 data cache, L1P buffer and L2 cache are very similar.

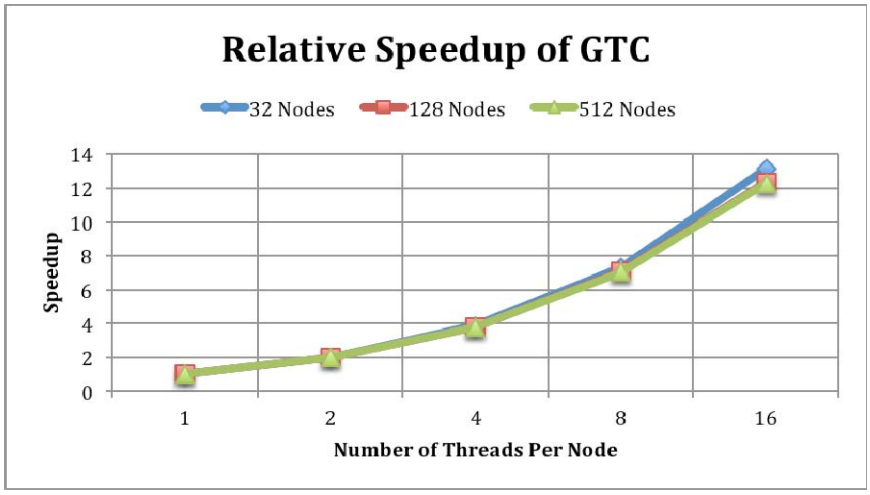


**Figure 12. Performance comparison of GTC with different number of threads per node**

**Table 6. Performance (per node) comparison of GTC on 128 Nodes**

| #Thread | 1 | 2 | 4 | 8 | 16 |
|---|---|---|---|---|---|
| **Runtime** | 2178.0 | 1110.0 | 575.8 | 309.5 | 176.9 |
| **Instruction Mix** | FPU: 38.15%<br>FXU:61.85% | FPU: 38.21%<br>FXU:61.79% | FPU: 38.17%<br>FXU:61.83% | FPU: 38.02%<br>FXU:61.98% | FPU: 37.56%<br>FXU:62.44% |
| **IPC per core** | 0.3227 | 0.3162 | 0.305 | 0.2849 | 0.2525 |
| **GFlops** | 0.279 | 0.548 | 1.056 | 1.965 | 3.441 |
| **L1 D-cache** | 95.97% | 95.97% | 95.97% | 95.99% | 96.03% |
| **L1P Buffer** | 0.79% | 0.79% | 0.79% | 0.79% | 0.78% |
| **L2 Cache** | 2.80% | 2.80% | 2.80% | 2.78% | 2.75% |
| **DDR traffic (Bytes/cycle)** | 0.133 | 0.262 | 0.505 | 0.940 | 1.663 |
| **%MPI** | 0.39% | 0.59% | 0.96% | 1.68% | 2.85% |

## 5. Conclusions

In this paper, we analyzed and compared the performance of five hybrid scientific applications with increasing the number OpenMP threads per node on the large-scale multithreaded BlueGene/Q supercomputer Vesta, and quantified the performance gap resulting from using different number of threads per node. We also found that increasing the number of threads to some extent saturated or worsened performance of these hybrid applications. For the strong-scaling hybrid scientific applications such as SP-MZ, BT-MZ, PEQdyna and PLMB, using 32 threads per node results in much better efficiency than using 64 threads per node, and with increasing the number of threads per node, the FPU percentage decreases, and the MPI percentage (except PMLB) and IPC per core (except BT-MZ) increase. For the weak-scaling hybrid scientific application such as GTC, the performance trend (relative speedup) is very similar with increasing number of threads per node no matter how many nodes (32, 128, 512) are used.

For these hybrid applications, how many OpenMP threads per node is limited by number of cores per node and number of hardware threads supported per core in the underlying system and the loop sizes to which OpenMP parallelization is applied. When increasing the number of cores for a given problem size, decreasing parallelized loop sizes may cause some idle cores per node because the loop sizes are not larger than the number of OpenMP threads per node. For instance, for SP-MZ and BT-MZ (version 3.3), we find that OpenMP

parallelization is applied to the Z dimension of the 3D mesh, however, the problem size for Class D is 1632x1216x34. So the maximum number of threads is 34. This limits potential for OpenMP speed-up and causes more idle threads when using 64 threads per node. So before running these hybrid applications on a large-scale multicore supercomputer, these limitations should be examined.

This work identified the optimal number of OpenMP threads per node used for efficient application execution. This will aid in developing energy-efficient applications and performance-power trade-off models in our MuMMI project [7].

## Acknowledgements

This work is supported by NSF grant CNS-0911023. The authors would like to acknowledge Argonne Leadership Computing Facility for the use of BlueGene/Q under DOE INCITE project "Performance Evaluation and Analysis Consortium End Station" and BGQ Tools project. We would also like to thank Stephane Ethier from Princeton Plasma Physics Laboratory for providing the GTC code.

## References

1. Argonne Leadership Computing Facility BlueGene/Q (Vesta), http://www.alcf.anl.gov/ resources, https://wiki.alcf.anl.gov/parts /index.php/ Blue_Gene/Q.
2. S. Ethier, First Experience on BlueGene/L, *BlueGene Applications Workshop*, April 27-28, 2005. http://www.bgl.mcs.anl.gov/Papers/GTC_ BGL_20050520.pdf.
3. R. A. Haring, M. Ohmacht, et al., The IBM Blue Gene/Q Compute chip. *IEEE Micro* 32(2): 48–60, 2012.
4. H. Jin, R. F. Van der Wijngaart, "Performance characteristics of the Multi-Zone NAS Parallel Benchmarks," *J. Parallel Distributed Computing*, vol. 66, no. 5, pp. 674-685, May 2004.
5. H. Jin, M. Frumkin and J. Yan, *The OpenMP Implementation of NAS Parallel Benchmarks and Its Performance*, NAS Technical Report NAS-99-011, 1999.
6. J. Levesque, J. Larkin, M. et al., *Understanding and Mitigating Multicore Performance Issues on the AMD Opteron Architecture*, LBNL-62500, March 7, 2007.
7. Xi. Wu, V. Taylor, C. Lively, H. Chang, C. Su, K. Cameron, S. Moore, D. Terpstra and C. Lee, MuMMI: Multiple Metrics Modeling Infrastructure for Exploring Performance and Power Modeling, *XSEDE 2013*, July 22 - 25, 2013, San Diego, CA. Also see Multiple Metrics Modeling Infrastructure (MuMMI) project, http://www.mummi.org.
8. J. Nieplocha, A. Mrquez, J. Feo, D. Chavarra-Miranda, G. Chin, C. Scherrer, and N. Beagley, Evaluating the potential of multithreaded platforms for irregular scientific computations, *the 4th International Conference on Computing Frontiers*, 2007.
9. M. Ohmacht, Memory Speculation of the BlueGene/Q Compute Chip, *PACT2011 Workshop on Wild and Sane Ideas in Speculation and Transactions*, 2011.
10. F. Petrini, D. J. Kerbyson, and S. Pakin, The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q, *SC03*, 2003.
11. M. A. Suleman, M. K. Qureshi, and Y. N. Patt, Feedback-Driven Threading: Power-Efficient and High-Performance Execution of Multi-threaded Workloads on CMPs, *the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2008.
12. V. Taylor, X. Wu, and R. Stevens, Prophesy: An Infrastructure for Performance Analysis and Modeling System of Parallel and Grid Applications, *ACM SIGMETRICS Perf. Eva. Review*, Vol. 30, Issue 4, 2003.
13. Top 500 list, http://www.top500.org/list/2012/11/
14. Bob Walkup, *Internal Use MPI Wrappers for BGQ*, Sep. 7, 2011.
15. Bob Walkup, Blue Gene/Q Power-Efficient Parallel Computation, *Blue Gene/Q Summit*, Argonne National Laboratory, Oct. 2, 2012.
16. X. Wu and V. Taylor, Performance Characteristics of Hybrid MPI/OpenMP Implementations of NAS Parallel Benchmarks SP and BT on Large-Scale Multicore Clusters, *The Computer Journal*, Volume 55 Issue 2, Feb. 2012, pp. 154-167.
17. X. Wu and V. 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. 2007.
18. X. Wu and V. Taylor, Using Processor Partitioning to Evaluate the Performance of MPI, OpenMP and

Hybrid Parallel Applications on Dual- and Quad-core Cray XT4 Systems, *the 51st Cray User Group Conference (CUG2009)*, May 2009.

19. X. Wu, V. Taylor, C. Lively, and S. Sharkawi, Performance Analysis and Optimization of Parallel Scientific Applications on CMP Cluster Systems, *Scalable Computing: Practice and Experience*, Vol.10, No.1, 2009.
20. X. Wu, B. Duan, and V. Taylor, Parallel Finite Element Earthquake Rupture Simulations on Quad- and Hex-core Cray XT Systems, *the 53rd Cray User Group Conference*, May 23-26, 2011, Fairbanks, Alaska.