An Empirical Study of Hyper-Threading in High Performance Computing Clusters

Tau Leng, Rizwan Ali, Jenwei Hsieh, Victor Mashayekhi, Reza Rooholamini
Dell Computer Corp.
U.S.A.

Abstract

The effects of Intel Hyper-Threading technology on a system performance vary according to the type of applications the system is running. Hyper-Threading affects High Performance Computing (HPC) clusters similarly. The characteristics of application run on a cluster will determine whether Hyper-Threading will help or hinder performance. In addition, the operating system's support for scheduling tasks, with Hyper-Threading enabled, is an important factor in the overall performance of the system.

In this paper, we used an experimental approach to demonstrate the performance gain or degradation of various parallel benchmarks running on a Linux cluster. The results of these benchmarks show that performance varies as a function of the number of nodes and the number of processors per node. Furthermore, we used a performance analysis tool to determine the cause of these performance differences when Hyper-Threading was enabled versus disabled. Our analysis shows the correlation between the cluster performance and the program characteristics, such as computational type, cache and memory usage, and message-passing properties. We conclude the paper by providing guidance on how to best apply Hyper-Threading technology to application classes.

1. Introduction

Intel's Hyper-Threading technology makes a single physical processor appear as two logical processors. The physical processor resources are shared and the architectural state is duplicated for the two logical processors [1]. The premise is that this duplication allows a single physical processor to execute instructions from different threads in parallel rather than in serial, and therefore, could lead to better processor utilization and overall performance.

1.1. Level of Parallelism

Two levels of parallelism have been addressed in the modem computer processor design to improve performance. Instruction-level-parallelism (ILP) refers to techniques of increasing the number of instructions executed each clock cycle. Although it is possible that the multiple execution units in a processor can execute multiple instructions at the same time, the dependencies existed among instructions makes it a challenge of finding enough instructions to execute simultaneously. Several mechanisms have been implemented to increase ILP. For example, "out-of-order execution" is a technique of evaluating a set of instructions and sending them for execution in parallels, regardless their original order defined by the program, and yet preserving the dependencies among the instructions.

Thread-Level Parallelism (TLP), on the other hand, enables a processor or multiprocessor system to concurrently run multiple threads from an application or from multiple, independent programs. SMT, or Simultaneous Multi-Threading technology, upon which Hyper-Threading is based, permits a processor to exploit both ILP and TLP. Multiple threads can run on an SMT processor, and the processor will dynamically allocate resources between the threads, enabling a processor to adapt to the varying requirements of the workload. Intel's Hyper-Threading implements SMT in such a way that each logical processor maintains a separate architectural state, which consists of general-purpose, control, machine state, and advanced programmable interrupt controller (APIC) registers [1]. The chip real estate required for the architectural states is negligible compared to the total die size. Thus, threads or separate programs using separate architectural states must share most of the physical processor resources, such as trace cache, L2-L3 unified caches, translation look aside buffer, execution units, branch history table, branch target buffer, control logic, and buses. This simultaneous sharing of resources between two threads creates a potential for performance degradation.

1.2. Multithreading and Message-passing Applications

In general, processors enabled with Hyper-Threading technology can improve the performance of applications with high degree of parallelism. Previous studies have shown that the Hyper-Threading technology improves multi-threaded applications' performance by the range of 10 to 30 percentages depending on the characteristics of the applications [2]. These studies also suggest that the potential gain is only obtained if the application is multi-threaded by any means of parallelization techniques. A multithreading program is capable of creating multiple processes, or threads, at a time without having to have multiple copies of the program running in the computer.

With the addition of Hyper-Threading support in Linux kernels 2.4.9-31 and above, Linux cluster practitioners have started to assess its performance impact on their applications. In our area of interest, high performance computing (HPC) clusters, applications are commonly implemented by using standard message-passing

interface, such as MPI or PVM. Applications developed from message-passing programming model usually employ a mechanism, "*mpirun"* for example, to spawn multiple processes and map them to processors in the systems. Parallelism is achieved through the message-passing interface among the processes to coordinate the parallel tasks. Unlike the multithreaded programs in which the values of application variables are shared by all the threads, a message-passing application runs as a collective of autonomous processes, each with its own local memory.

This type of applications can also benefit from Hyper-Threading technology in the sense that the number of processes spawned can be doubled and the parallel tasks can potentially execute faster. Applying Hyper-Threading and doubling the processes that simultaneously run on the cluster will increase the utilization rate of the processors' execution resources. Therefore, the performance can be improved. On the other hand, overheads might be introduced in the following ways:

- Logical processes may compete for access to the caches, and thus could generate more cache-miss situations
- More processes running on the same node may create additional memory contention
- More processes on each node increase the communication traffic (message passing) between nodes, which can oversubscribe the communication capacity of the shared memory, the I/O bus or the interconnect networking, and thus create performance bottlenecks.

Whether the performance benefits of Hyper-Threading – better resource utilization – can nullify these overhead conditions depends on the application's characteristics.

In this paper, we have used an experimental approach to demonstrate the impact of Hyper-Threading on a Linux cluster by using various MPI benchmark programs, and discussed the adaptability of this new technology into HPC clusters for improving performance. In the next section, we describe the cluster configurations and the performance tool for our experiments. Section 3 introduces the performance benchmarks and the results, alone with the performance analysis. In this section, we use a performance tool called Vtune™ to analyze the system behavior while running the benchmark programs on the cluster. The causes of performance gain or degradation for applying Hyper-Threading on the cluster can be understood through the performance analysis. Section 4 is the conclusion.

2. Experimental Environment

2.1 The Cluster Configuration

Our testing environment is based on a cluster consisting of 32 Dell PowerEdge 2650 servers interconnected with Myrinet. Each PowerEdge 2650 has two Intel Xeon processors running at 2.4 GHz with 512KB L2 cache, 2GB of DDR-RAM (double data rate RAM) memory operating on a 400 MHz Front Side Bus. The chipset of PowerEdge 2650 is the ServerWorks GC-LE, which accommodates up to six

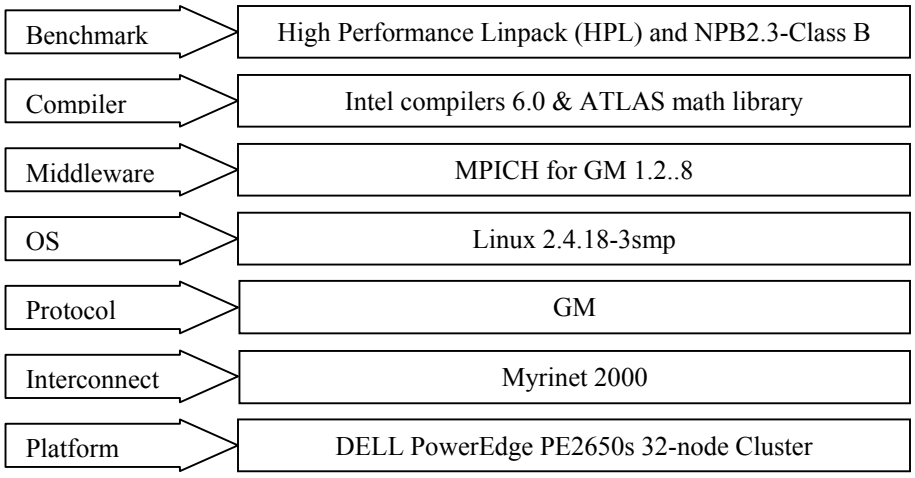

Figure 1. Architectural stack of the test environment. The benchmarks were compiled with Intel C or Fortran compilers. The OS is RedHat 7.3 distribution with kernel version 2.4.18-3smp.

registered DDR 200 (PC1600) DIMMs with a 2-way interleaved memory architecture. Each of the two PCI-X controllers on the 2650 has its own dedicated 1.6 GB/s full duplex connection to the North Bridge to accommodate the peak traffic generated by the PCI-X busses it controls.

The operating system installed for the cluster is RedHat 7.3 with kernel version 2.4.18-3smp[1]. The benchmark programs were compiled with Intel C or FORTRAN compilers, and ATLAS (Automatically Tuned Linear Algebra Software) math library. Figure 1 shows the architectural stack of our test environment.

2.2 The Performance Analysis Tool

We use Vtune™, an Intel implemented tool, for our performance analysis. A Windows desktop is then utilized in our test environment for the Vtune Performance Analyzer to display performance data in graphical formats, as well as collecting statistical data of the system behavior. Through a "data collector", a Linux agent, the analyzer is able to gather targeted Linux system's information remotely on the fly. The collecting method we selected is called "sampling", which is a non-intrusive, instruction-address collector mechanism [3]. During sampling, the performance analyzer monitors all the software executing on the system including the OS kernel and the benchmark program. The monitoring areas that the analyzer emphasizes on

[1] The Linux information collector of Vtune™ performance analysis tools 6.1 used for this study supports Linux Kernel version 2.4.18-3.

are according to the user's specification. For our study, we focus on the following system information in particular.

- Cycle per Instruction (CPI) Retired – The lower the CPI is, the faster the program has been executed. For P4 Xeon processor, 0.75 or less of CPI is considered "good". This event count is used to understand the performance of each node and to verify the performance results of our benchmarks.
- Floating-point Computation Instructions – shows all the floating-pint instructions that had been retired.
- 2^{nd}-Level Cache Read Misses % – 2^{nd} Level cache read misses reduce performance for the processor must then access main memory
- Streaming SIMD Extension 2 (SSE2) – The floating-point SIMD instructions allow computations to be performed on packed double-precision floating-point values (two double-precision values per XMM register). Our benchmark programs were compiled with – SSE2 option, which allows the code taking advantage of the SSE2 feature. This event count is for understanding if the program threads are fully utilizing this resource.
- x87 Instruction Retired – This event count increments for each x87 floating-point micro-op, specified through the event mask for detection. All of our benchmarks, except IS, are floating-point intensive. This event count will provide a fair understanding of the utilization of floating-point execution units.

3. Benchmarking Results and Analysis

The MPI programs we used for the experiments are the High-Performance Linpack (HPL) and the NAS Parallel Benchmark (NPB), benchmarks commonly used in the High Performance Computing arena.

3.1 Benchmarking with the High Performance Linpack (HPL)

HPL uses a number of linear algebra routines to measure the time it takes to solve dense linear equations in double precision (64 bits) arithmetic using the Gaussian elimination method [4]. The measurement obtained from Linpack is in the number of floating-point operations per second (FLOPS). Linpack mainly exercises the floating-point calculation capability of the system. However, the communication latency of the system for running Linpack also plays a significant role on the overall performance; when using dual processors compute nodes interconnected with high-speed networking, such as Myrinet, the actual performance of a cluster may reach almost 60% of its theoretical peak performance, and the percentage could be less than 30% when using slower interconnect like Fast Ethernet [5]. When running Linpack, the more the memory used of the system or the larger the problem size specified for executing the program, the better the performance of the system. However, as a rule of thumb, the problem size or the memory usage should not exceed 80% of the total memory in the system for avoiding the swapping situation, which will decrease the performance significantly.

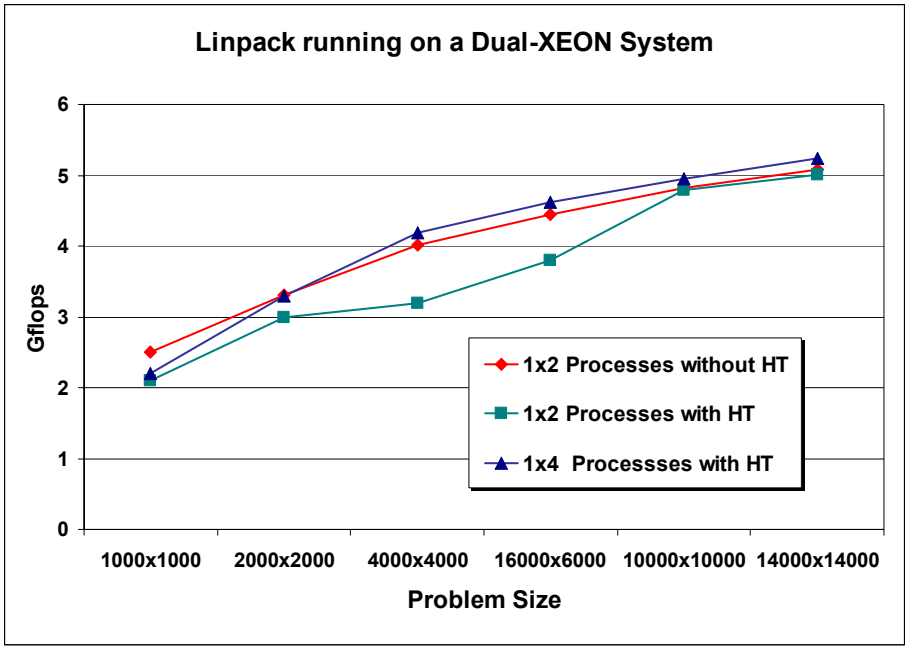

Figure 2. The Linpack performance results on single node. The worse case in here is to spawn two processes when the Hyper-Threading (HT) is on.

To understand the impact of Hyper-Threading on single compute node, we first conducted a series of HPL runs from small problem size to large. The results shown in Figure 2 indicate that only when the problem size or the memory used is larger to some extent, 2000x2000 blocks or more, we can see modest performance improvement (around 5%) on Hyper-Threading configurations. This is due to the initiating overhead of Linpack, which is larger when spawning more processes. This overhead is not disguised when running a very small problem size. Since Linpack was compiled with the highly optimized ATLAS library, the floating-point functional units including the SSE2 were almost fully utilized during the execution. This leaves very little room for improving the Linpack performance by switching on Hyper-Threading to increase CPUs' resources utilization.

Also note that in Figure 2, for the runs where the number of processes is less than the number of logical processors with Hyper-Threading enabled, the performance is considerably worse. This observation is more apparent on 16-node runs. Figure 3 shows that the result of 16x2 with Hyper-Threading enabled has only small performance improvement compared to the 16x1 runs with HT disabled.

For the cluster runs, the average CPIs shown in the Vtune analyzer are 0.42, 0.46, and 0.59 for running 4 processes on each node with Hyper-Threading enabled,

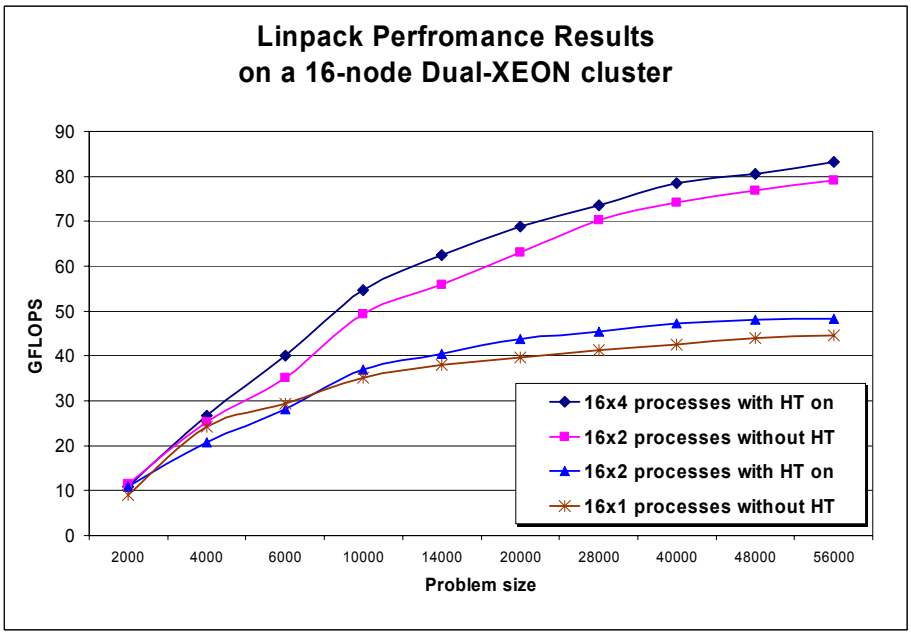

Figure 3. The HPL Linpack performance results running on the cluster of 16 nodes. Small performance difference between using 2 logical processors and using 1 physical processor indicates the limitation of HT support in Linux.

running 2 processes on each node without Hyper-Threading, and running 2 processes on each node with Hyper-Threading respectively. These statistical sampling data are in accordance with the actual performance results – when Hyper-Threading is enabled, running 4 processes on each node increases performance around 5%, and running 2 processes on each node reduces performance around 25% and more.

From the observation of "Instruction Retired Rate" in Vtune analyzer data, in the two-processes runs with Hyper-Threading enabled, three of the logical processors had been utilized substantially while the fourth one had not been used. In addition, the "2nd level cache misses %" showed inconsistent rates for the two physical processors (83% and 75%). Which means the Linux OS had been allocating the CPU resources for load-balancing the two threads. But the OS scheduler, without knowing the association of the physical processors and the logical processors, had been scheduling the two threads on two logical processors which may be designated to the same physical processor. This is a limitation of Hyper-Threading support in the Linux current kernel. The issue has been addressed in the Linux community and expecting to be resolved in the future releases [9].

Although our preliminary study indicates that Linpack-type of applications can benefit from Hyper-Threading, we have seen mixed results when running the NAS parallel benchmarks suite.

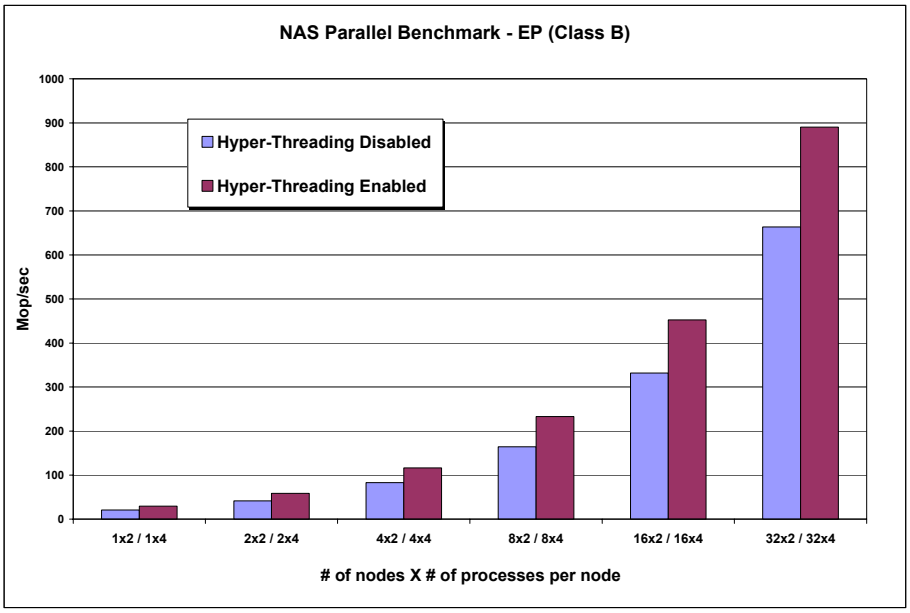

Figure 4. The EP (Class B) benchmark results of Hyper-Threading enabled and disabled. The results of Hyper-Threading enabled shows 40% improvement regardless the number of nodes.

3.2 Benchmarking with the NAS Parallel Benchmark (NPB)

The NAS benchmark suite comprises five kernels and three pseudo-applications and is designed to gauge parallel computing performance. Each of the programs solves a specific numerical problem [6]. The performance results are measured in Million Operations per Second (Mop/s). Since each program represents a specific type of CFD applications, from the benchmark results, one can realize the under-testing system's performance characteristics from various aspects. In this paper, we used three of the eight programs for our experiments, EP, FT, and IS.

In the ***Embarrassingly Parallel (EP) Benchmark***, two-dimensional statistics are accumulated from a large number of Gaussian pseudo-random numbers, which are generated according to a particular scheme that is well suited for parallel computation. This problem is typical of many *Monte Carlo* applications. Since it requires almost no communication, in some sense this benchmark provides an estimate of the upper achievable limits for arithmetic operations' performance on a particular system.

Since EP is a computation-bound program and requires almost no communication during the runs, with Hyper-Threading enabled, it could effectively utilize the CPUs'

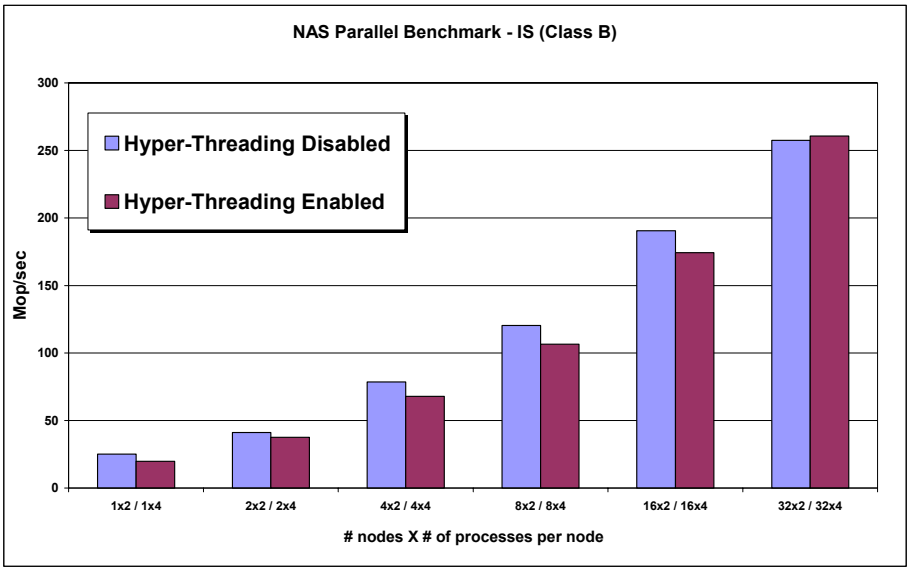

Figure 5. The IS (Class B) benchmark results of Hyper-Threading enabled and disabled. The percentage of performance difference becomes smaller while the node count is larger.

resources without being concerned with the communication overhead, which makes the performance improve significantly. This condition is true regardless of the number of nodes or CPUs of the cluster. Figure 4 shows the EP performance improved linearly from one node to 32 nodes, as well as the constant performance gains on Hyper-Threading enabled runs.

Integer Sort (IS) tests a sorting operation that is important in *particle method* codes. This type of application is similar to particle-in-cell applications of physics, wherein particles are assigned to cells and may drift out. The sorting operation is used to reassign particles to the appropriate cells. This benchmark tests both integer computation speed and communication performance. This problem is unique in that floating-point arithmetic is not involved. Significant data communication, however, is required.

With Hyper-Threading enabled, doubling the IS processes running on each node from 2 processes to 4 processes, creates much more communication traffics among processes and memory contentions inside the nodes. Yet the CPU floating-point execution units are still underutilized. In both cases, the "x87 Instruction Retired Rate", observed from Vtune Performance Analyzer, are around 2.3% indicating that there is almost no floating-point calculation. Hence, the performance will not be improved through increasing the resource's utilization. Figure 5 shows the IS results comparison from 1 node to 32 nodes.

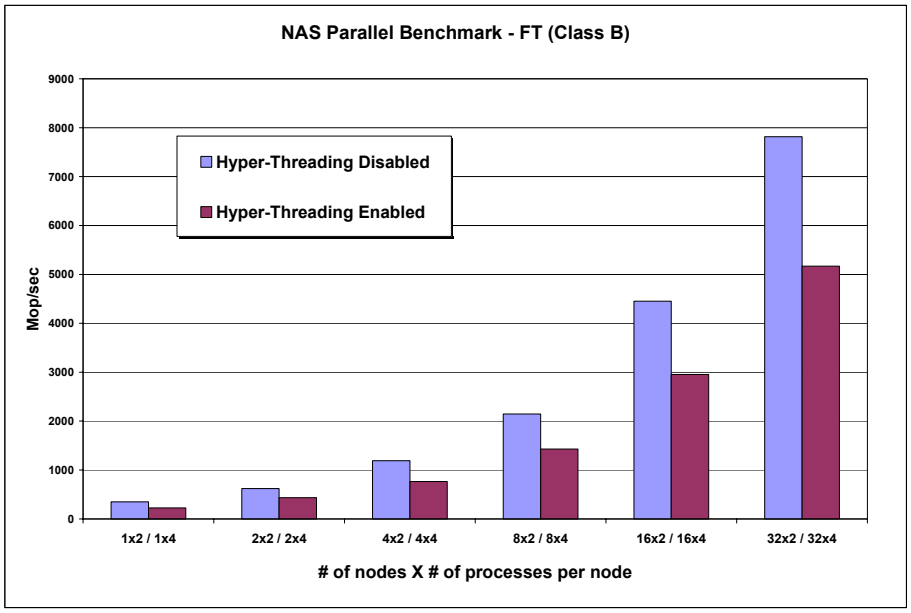

Figure 6. The FT (Class B) benchmark results of Hyper-Threading enabled and disabled. The performances were degraded for 50% for all the configurations when using Hyper-Threading.

It also can be observed that the IS performance dissimilarity between Hyper-Threading disabled and enabled is getting smaller when the node count is larger. As showed in Figure 3, when the node count is equal to 32, the performance becomes better for 128 (or 32x4) processes running IS with Hyper-Threading enabled. The reason is that the proportions of the communication through interconnect network becomes larger than that through the shared memory, which comparatively releases the memory contentions and communication traffics of having four logical processors running four processes in each node. Therefore, we expect to see performance improvement for the cluster larger than 32 nodes running IS with the same configuration.

Note that this phenomenon might not be applicable for other cluster configurations. For example, using Fast Ethernet as the interconnect for the cluster decreases the communication capability dramatically; therefore the shared memory communication capability of the cluster becomes relatively higher. In such a case, increasing the node count will obstruct the performance of IS, instead of facilitating it [5].

In the ***3-D FFT PDE (FT) benchmark***, a 3-D partial differential equation is solved using FFTs. This program performs the essence of many *spectral method*s. It is a good test of long-distance communication performance. FT requires intensive float-point operations and messages passing among processes. FT is also a cache-friendly benchmark; the large cache will facilitate the performance [7][10].

From the Vtune performance data, the level 2 cache misses is increased from 68% for non-Hyper-Threading runs to 76% for Hyper-Threading runs. Also, the "SSE2" and the "x87 instructions rate" were both up to 99.9%. Moreover, the communication bandwidth among processes required by FT creates memory contentions and communication bottlenecks, which lead to 50% constant performance degradation for the Hyper-Threading enabled runs. The results shown in Figure 5 indicate that for applications like FT, Hyper-Threading will not provide any gain rather will degrade the cluster performance for any node count.

2 Conclusions

Hyper-Threading could improve the performance of some MPI applications running on a cluster, but not all. Depending on the cluster configurations and more importantly the nature of the application running on the cluster, the performance gain can vary or even be negative. By using performance analysis tool, we were able to understand what areas contribute to the performance gains, and what areas contribute to the overheads, which lead to performance degradation. Based on our analysis, the following observations are made for applying Hyper-Threading on a Linux cluster.

- Computational intensive applications with fine-tuned floating-point operations have less chance to be improved in performance from Hyper-Threading, because the CPU resources could already be highly utilized.
- Cache-friendly applications might suffer from Hyper-Threading enabled, because logical processors share the caches and thus the processes running on the logical processors might be competing for the caches' access, which might result in performance degradation.
- Communication-bound or I/O-bound parallel applications may benefit from Hyper-Threading, if the communication and the computation can be performed in an interleaving fashion between processes. However, the additional I/O traffic might create communication bottleneck and reduce the overall performance.
- The current version of Linux OS's support on Hyper-Threading is limited, which could cause performance degradation significantly if Hyper-Threading is not applied properly.

Hyper-Threading technology introduces yet another element of designing or operating a balanced cluster system. The challenge of incorporating the technology in HPC environment is not just for the users to decide whether or how it should be applied, but also for the OS, compiler, and performance tool developers to make the technology more applicable and useful.

References

[1] D. T. Marr el al, "Hyper-Threading Technology Architecture and Micro architecture", Intel Technology Journal, Vol. 6, Issue 01, February, 2002.

[2] W. Magro, P. Petersen, S. Shah, "Hyper-Threading Technology: Impact on Compute-Intensive Workloads", Intel Technology Journal, Vol. 6, Issue 01, February 2002.
[3] Intel Vtune Performance Analyzer 6.1, http://www.intel.com/software/products/vtune/vtune61/index.htm
[4] "HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers", *Netlib Repository at UTK and ORNL*, http://www.netlib.org/benchmark/hpl.
[5] J. Hsieh, T. Leng, V. Mashayekhi, and R. Rooholamini. "Architectural and Performance Evaluation of Giganet and Myrinet Interconnects on Cluster of Small-Scale SMP Servers", in the *Proceedings of Super-Computing '00*, Dallas, TX, November 2000.
[6] NAS Parallel Benchmark suite, http://www.nas.nasa.gov/Software/NPB/
[7] J. Hsieh, T. Leng, V. Mashayekhi and R. Rooholamini. "Impact of Level 2 Cache and Memory Subsystem on the Scalability of Clusters of Small-Scale SMP Servers", in the *Proceedings of International Conference on Cluster Computing*, Cluster'00, November 2000.
[8] M. Kravetz, H. Franke, S. Nagar, R. Ravindran, "Enhancing Linux Scheduler Scalability", in the 2001 Ottawa Linux Symposium, July 2001.
[9] The Linux Kernel Achieve, ChangeLog-2.4.19, http://www.kernel.org/.
[10] F.C. Wong, R.P. Martin, R.H. Arpaci-Dusseau, and D.E. Culler. "Architectural Requirements and Scalability of the NAS Parallel Benchmarks", In the Proceedings of SuperComputing '99, Portland, Oregon, November 1999.