Characterization of Multithreaded Scientific Workloads on Simultaneous Multithreading Intel Processors

Ryan E. Grant Ahmad Afsahi
Department of Electrical and Computer Engineering
Queen's University
Kingston, ON, Canada K7L 3N6
9reg3@qlink.queensu.ca, ahmad.afsahi@queensu.ca

Abstract

Simultaneous Multithreading (SMT) is a technique that allows multiple independent threads to execute different instructions each cycle. Hyper-Threading (HT) is an implementation of SMT available on recent processors from Intel. Naturally, Multi-threaded applications are very suitable for SMT systems. However, HT due to extensive resource sharing may not suitably benefit OpenMP high performance computing applications.

In this paper, we first present performance of different OpenMP constructs on dual and quad HT-based Intel Xeon servers under 2.4.22 and 2.6.9 kernels. We find that the overhead of OpenMP constructs with HT is an order of magnitude larger than when HT is off. We then use a range of OpenMP applications from the NAS and SPEC OMPM2001 suites to measure performance on a dual Hyper-Threaded SMP. Our performance results indicate majority of applications benefit from having a second thread in one-processor situations. However, only a few applications enjoy performance gain when HT is enabled on both processors. Data from hardware performance counters verifies trace cache misses and its delivery rate are sources of performance bottleneck.

1. Introduction

Simultaneous Multithreading (SMT) [14] is a technique that allows multiple independent threads to execute different instructions each cycle. SMT attempts to increase performance by exploiting *Thread Level Parallelism* in hardware without the need to context switch, in addition to the exploitation of *Instruction Level Parallelism* in modern wide issue superscalar processors. SMT processors have been studied in academia [14], and have recently appeared in mainstream processors from IBM, and Intel [4, 7].

Intel *Hyper-Threaded* (HT) processors, including Intel Xeon MP and Xeon EM64T, are implementations of SMT[1]. An HT-enabled processor appears as two logical processors to the operating system, where each processor maintains a separate run queue. These logical processors share many hardware resources such as cache, execution units, TLBs, branch prediction unit, and load and store buffers. In fact, numerous complex interactions among the shared resources may affect the performance of multi-threaded application [13, 8, 15].

Shared-memory multiprocessor (SMP) systems have gained prominence. Considerable work has gone into the design of such systems. Meanwhile, OpenMP [1] has emerged as the standard for parallel programming on SMP systems. The directive-based interface of OpenMP makes it one of the easiest ways to incrementally parallelize existing and new applications.

OpenMP scientific applications have been designed for conventional SMP systems. With the introduction of SMT capabilities in modern microprocessors, hybrid SMP systems may expose different performance due to workload characteristics of such OpenMP applications. They also present new challenges to the systems software to accommodate inter- and intra- parallelism among threads. To maximize utilization of an SMP with SMT processors, there should be as many threads as available logical processors. However, greater demand on cache subsystem, increased bus transactions, stall cycles, and synchronization overhead among larger group of threads affect the performance of OpenMP applications [13, 15].

The first contribution of this paper is the performance evaluation of OpenMP constructs on a 4-way HT-based Intel Xeon MP SMP server, and a 2-way HT-based Intel Xeon EM64T SMP server with Linux kernel versions 2.4.22 and 2.6.9. Our intention in this paper is to compare the impact of these kernels on the OpenMP constructs when HT is enabled.

The second contribution of this paper is the evaluation of scientific applications in the NAS OpenMP suite

[1] In this paper, we use HT and SMT interchangeably.

(version 3.2) [3], and SPEC OMPM2001 suite (version 3) [11] with kernel 2.6.9. It is interesting to discover the impact of SMT-based SMP systems on the performance of such applications. We consider the effects of resource sharing within the processors on the performance by collecting data from hardware performance counters. We pinpoint architectural limitations of such a system by observing its trace cache performance.

The rest of this paper is organized as follows. Our experimental setup is described in section 2. In Section 3, we present and analyze our performance results. Related research is described in section 4. Finally, the paper is concluded in Section 5 with some thoughts on future research.

2. Experimental Setup

The experiments were conducted on two platforms: Dell PowerEdge 6650, and 2850 servers. The PowerEdge 6650 has four 1.4 GHz Intel Xeon MP processors with 12KB shared execution trace cache, 8KB L1 shared data cache with 4-way associativity, 256KB shared and unified L2 cache, 512KB shared and unified L3 cache (both L2 and L3 with 8-way associativity), and 2GB of DDR-SDRAM on a 400 MHz Front Side Bus. The operating system is based upon the RedHat Linux 9 distribution, but with the Vanilla kernel versions 2.6.9 and 2.4.22. We used two different kernels to evaluate their impact on the performance. Both kernels support Hyper-Threading, however the task scheduler has undergone a complete rewrite in the 2.6.9 kernel. It is known as the O(1) scheduler since it can make a scheduling decision in constant time and independent of the number of processors or the number of runnable tasks. Using the LMbench [9] we have measured the L1, L2, and main memory latencies of the processor as 1.42ns, 13.29ns, and 187.94ns, respectively. The main memory read and write bandwidths are 627 MB/s and 743 MB/s, respectively.

Our second platform is the recent PowerEdge 2850 server from Dell with two 2.8 GHz Intel Xeon EM64T processors with 16KB shared execution trace cache, 16KB L1 shared data cache, 1MB shared and unified L2 cache, and 2GB of DDR2-SDRAM on a 800 MHz Front Side Bus. The Operating system is RedHat Enterprise WS 4.1 with kernel 2.6.9. The L1, L2, and main memory latencies of the processor are 1.44ns, 10.25ns, and 142.80ns, respectively. The main memory read and write bandwidths are much higher than that of PowerEdge 6650, equal to 3856 MB/s and 1855 MB/s, respectively. The bandwidth improvement over PowerEdge 6650 is partially due to the RAM technology and speed (DDR-2 memory at 400 MHz relative to DDR at 266 MHz).

The number of active processors was limited using the *maxcpus=X* boot option of the kernel. This option causes the kernel to only initialize and use *X* logical processors. This method of disabling additional processors is preferable to running fewer threads when determining scalability since it better emulates a smaller system.

Application characteristics were gathered at run-time using the hardware performance counters available on the Intel Xeon processors. We collected data using Intel VTune Performance Analyzer version 7.2 [2]. The Intel Fortran and C/C++ compilers (version 8.1) were used to build the benchmark applications.

2.1. EPCC Microbenchmarks

Overheads due to synchronization, and loop scheduling are an important factor on the performance of shared-memory parallel programs written in OpenMP. The EPCC OpenMP microbenchmarks (version 2.0) [10] were used to measure the overheads of synchronization and loop scheduling in the OpenMP runtime library.

The synchronization benchmark measures the overhead incurred by work-sharing and mutual exclusion directives. The work-sharing directives include *PARALLEL*, *DO/FOR*, *PARALLEL DO/FOR*, and *BARRIER*. The mutual exclusion directives include *SINGLE*, *CRITICAL*, *LOCK/UNLOCK*, *ORDERED*, and *ATOMIC*.

The loop scheduling benchmark compares the scheduling policies available with OpenMP. Specifically, it compares the overhead of the *For* directive when used with three scheduling policies: *STATIC*, *DYNAMIC*, and *GUIDED*. The *STATIC* policy determines scheduling at compile time and is well suited for programs with static workloads that can be easily divided among threads. The *DYNAMIC* and *GUIDED* scheduling policies are intended for programs with dynamic workloads that must be balanced between threads at runtime. All three policies have an additional parameter, *chunk size*, which specifies the size of a single work unit in terms of loop iterations. A smaller chunk size allows for finer-grained scheduling at the cost of more scheduling overhead. The *GUIDED* policy attempts to balance this trade-off by dynamically decreasing the chunk size.

2.2. NAS OpenMP Benchmarks

The NAS Parallel Benchmarks has been widely used to characterize high-performance computers. Recently, the NAS OpenMP suite (version 3.2) [3] has been released. The suite consists of five kernels, (CG, MG, FT, IS, EP), and three simulated CFD applications (BT, SP, LU). We experimented with Class B of CG, MG, BT, SP,

and LU benchmarks. Class B is large enough to provide realistic results, ensuring their working set fits in memory.

2.3. SPEC OMPM2001 Applications

The SPEC OMPM2001 (version 3.0) suite of applications [11] was developed by the SPEC High-Performance Group. The suite consists of a set of OpenMP-based scientific applications. These programs were originally part of the SPEC CPU2000 suite and were parallelized by inserting OpenMP directives. We worked with seven of the nine SPEC applications. Interested reader is referred to [11].

3. Experimental Results and Analysis

This section describes the results and analysis of our experiments on the two servers. The notation HT-X refers to a system with HT enabled with X logical processors (X/2 physical processors). The notation SMP-X refers to a standard SMP system (HT disabled) with X physical processors.

We first present the overhead of OpenMP constructs. Then, we evaluate the performance of applications and metrics that may point to potential architectural bottleneck due to resource sharing on HT processors. We are particularly interested in the effects of sibling logical processors. Particular attention is paid to the performance gaps in the HT-4 and HT-2 cases. Results are shown for kernel 2.6.9, unless otherwise noted.

3.1. EPCC

3.1.1 OpenMP Synchronization

The EPCC synchronization benchmark (version 2) is used to find the overhead of OpenMP directives with varying numbers of threads, and with and without Hyper-Threading. Figure 1 depicts the overhead of different OpenMP synchronization directives for the C version (the Fortran directives are not shown but have slightly larger overhead). Interestingly, the synchronization overhead with Hyper-Threading is an order of magnitude greater than the same number of threads on an SMP system.

We compare the impact of different kernels on OpenMP synchronization directives. Overall, the new Linux kernel 2.6.9 performed on par with the Linux kernel 2.4.22 in terms of synchronization overhead for the SMP case (not shown here). However, the Linux kernel 2.4.22 was found to be superior in case of HT as shown in Figure 2. This has come as a surprise to us.

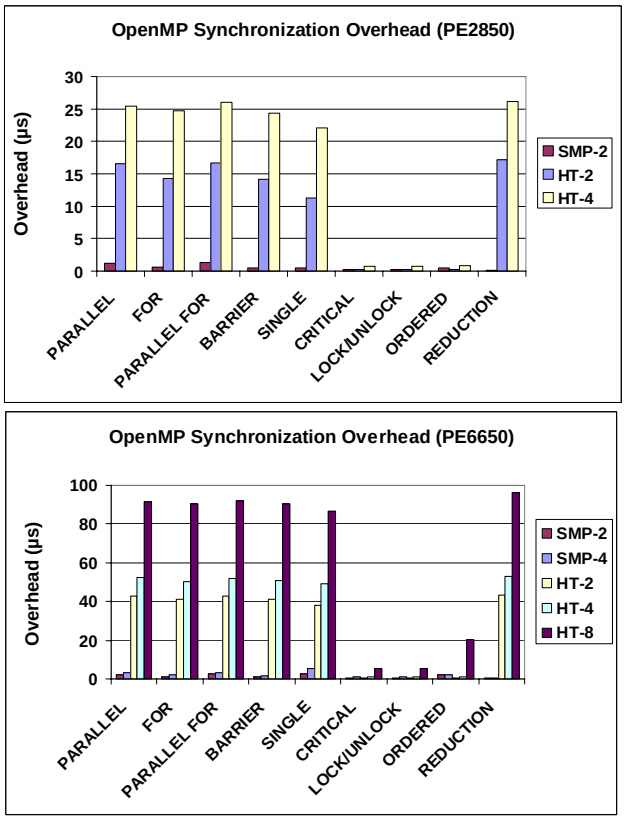

Figure 1. Overhead of OpenMP synchronization.

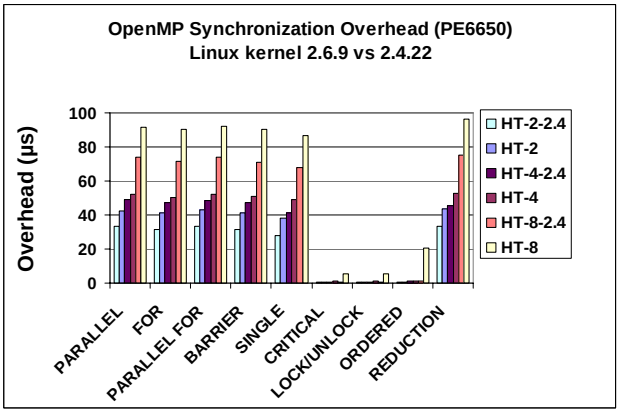

Figure 2. Kernel 2.6.9 vs. 2.4.22 impact on OpenMP synchronization overhead.

3.1.2 OpenMP Scheduling

OpenMP provides three options for scheduling loop iterations among threads: *STATIC*, *DYNAMIC*, and *GUIDED*. Figure 3 shows the loop scheduling overheads for SMP-4 and HT-8. As can be seen, the overhead of different OpenMP loop scheduling schemes is an order of magnitude larger for Hyper-Threading than for conventional SMP.

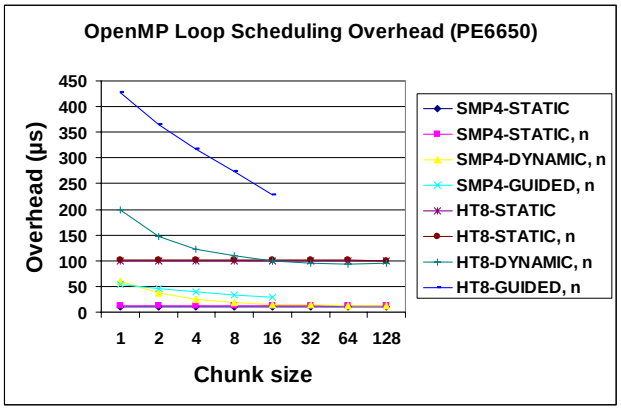

Figure 3. Overhead of OpenMP loop scheduling policies.

The *STATIC* scheduling is much faster than *DYNAMIC* scheduling, but of course does not have the benefit of dynamic load balancing. It is clear that if the workload is balanced between threads, then *STATIC* scheduling with its low overhead is clearly the best choice. However if the workload is dynamic and unbalanced, then either *GUIDED* or *DYNAMIC* scheduling could perform better if the benefit realized outweighs the extra overhead incurred by those policies. The overhead of *STATIC(n)* scheduling is nearly constant across all chunk sizes, *n*. The overhead of *STATIC(n)* matches the overhead of *STATIC*. For both SMT and SMP cases, the overhead of *GUIDED(n)* is worse than *DYNAMIC(n)*. Figure 4 compares the impact of two kernels on synchronization directives. Except for the *Guided* policy and HT-8, 2.4.22 is slightly better than the 2.6.9 kernel.

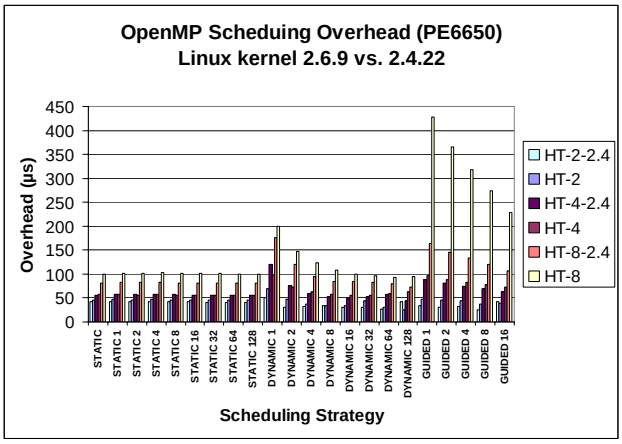

Figure 4. Kernel 2.6.9 vs. 2.4.22 impact on OpenMP loop scheduling policies.

3.2. Application Performance and Analysis

Application execution time is the most acceptable performance metric. In this section, we present the performance of NAS and SPEC applications on our PowerEdge 2850 platform under 2.6.9 kernel (space does not allow us to include the results on our 4-way platform).

Figure 5 shows the speedup for each of the NAS and SPEC application benchmarks on a variety of system configurations. The speedup results are calculated using application runtime relative to the serial case. For each application, the four columns can be considered as two pairs. The first pair is for one physical processor, with and without HT. The second pair is for two physical processors.

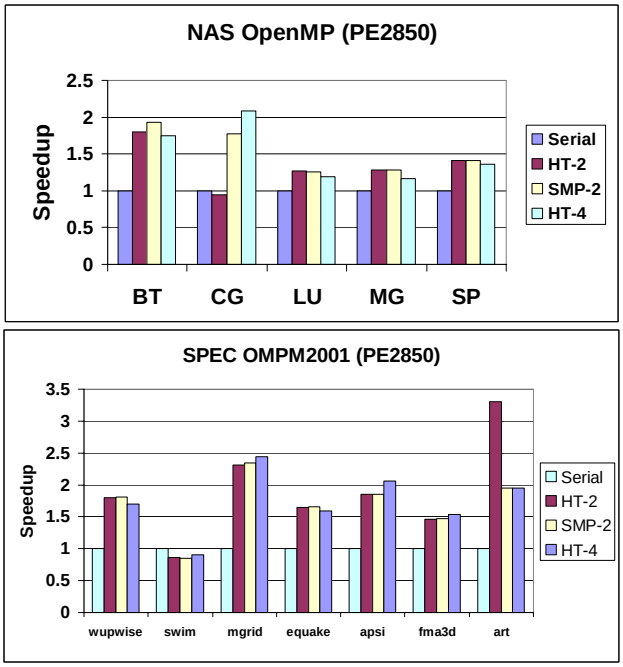

Figure 5. Speedup for NAS OpenMP and SPEC OMPM2001 applications under Kernel 2.6.9 relative to the serial case.

The goal is to understand if scientific multi-threaded applications can benefit from HT on real, commercial SMT-based SMP servers; and if they do not benefit, what architectural limitations exist in such processors. In fact, it is not always clear if it is better to run two threads or one thread on each processor. For instance, although BT, LU, MG, SP, WUPWISE, and EQUAKE benefit from having two threads in one-processor execution, but they suffer a bit in the two-processor execution. On the contrary, CG and SWIM perform the other way around. Applications such as MGRID, APSI, and FMA3D always benefit from HT. Interestingly, in ART and LU, HT-2 not only outperforms serial, but also outperforms SMP-2. This means that a pair of Hyper-Threaded logical processors is able to outperform two real physical processors. SWIM is the poorest application. It does not scale with either HT or SMP.

BT, CG, LU, MG, and SP attain an average speedup of 35%, 6%, 10%, 9%, and 19%, respectively, when Hyper-Threading is enabled. For the SPEC suite of applications, WUPWISE, MGRID, EQUAKE, APSI, and FMA3D achieve an average speedup of 37%, 68%, 31%, 49%, and 26%, respectively. While SWIM is the only application with a slowdown of 4%, ART has the best average speedup of all applications, equal to 115%. Overall, applications achieve a 33% speedup on average.

Although the above results look good, one can easily observe that except for CG no other application can benefit from HT-4 on our platform. Table 1 summarizes the average performance gain when logical processors are on. It is evident that the HT implementation of SMT cannot provide performance gain for 2-processor executions at least for our multi-threaded applications.

Table 1: Average speedup gained by enabling HT for NAS and SPEC Parallel benchmarks.

Physical Processors	NAS OMP	SPEC OMP	Overall
1	34%	89%	66%
2	-2.2%	2.5%	0.5%

3.2.1 Trace Cache Analysis

In this section, we investigate one the reasons behind possible flaws of Hyper-Threading (space does not allow to present stall cycles, CPI, and cycles/uop results, among others). Using hardware performance counters of Intel Xeon EM64T, we studied the behaviour of the trace cache for NAS applications as well as ART, APSI, and SWIM from SPEC. Figure 6 presents trace cache misses, while Figure 7 depicts trace cache delivery rate for these applications.

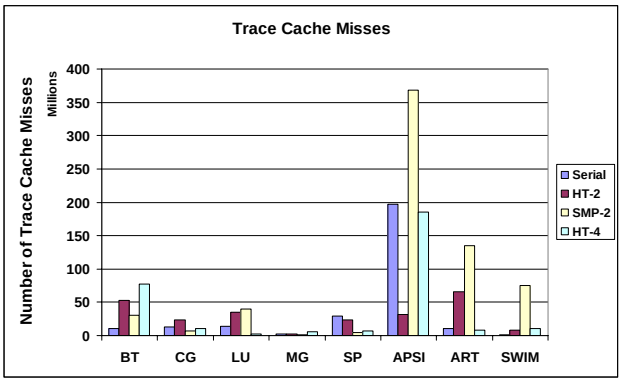

Figure 6. Trace cache misses.

Compared to the baseline single processor system without HT, the HT enabled system outperforms it on almost every task. The exception to this is when the increased processing of the HT system causes a bottleneck in the trace cache, causing performance to suffer. Therefore, except for programs that create a large number of trace cache misses that are also very bus bandwidth dependent, the SMT technology in Intel provides improvements over the single processor system.

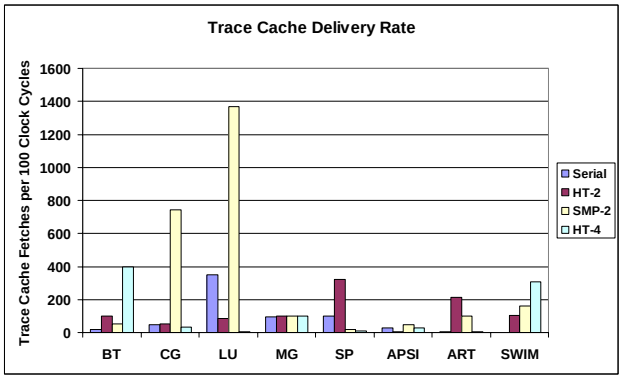

Figure 7. Trace cache delivery rate.

It can be observed in Figure 6 that when the number of trace cache misses increases, and due to memory bandwidth constraints the trace cache delivery cannot be increased, the performance of the HT system suffers. This implies that memory intensive applications are more likely to suffer from trace cache starvation, decreasing the effectiveness of HT when used with such applications. This can be understood by comparing Figure 5 to Figure 6 and Figure 7. For instance, when performance is significantly worse for the CG on HT-2 compared to the SMP-2 and HT-4 cases, one can observe a marked increase in the trace cache misses from the HT-2 system. This also occurs for the SP benchmark, but observe that the SP performance does not suffer because there is a corresponding increase in the trace cache delivery rate. Only in the case when the trace cache misses increase and the cache delivery rate drops, does the performance suffer. The HT-4 system is able to avoid this problem by its increased system resources, which significantly reduce the trace cache miss rate, avoiding the pitfalls seen in the HT-2 case. For the case of diminishing performance at the HT-4 level, one can observe that the main performance bottleneck is the cache memory bandwidth. In Figure 8, the increased number of L2 cache misses over the serial case corresponds to the decreased performance of the SWIM benchmark illustrated in Figure 5.

4. Related Work

Tuck and Tullsen [13] analyzed the Intel Pentium 4 HT processor. They found although up to 25% improvement on parallel applications can be achieved, Pentium 4 HT still shows symbiotic behaviour due to

cache and resource conflicts. Research at Intel focused on Hyper-Threading with multimedia OpenMP applications [12]. This study found that enabling HT increased performance. They showed HT does increase demand on the memory system. However, the benefit gained by increased utilization of other processor resources was more significant than the penalties incurred in the memory systems.

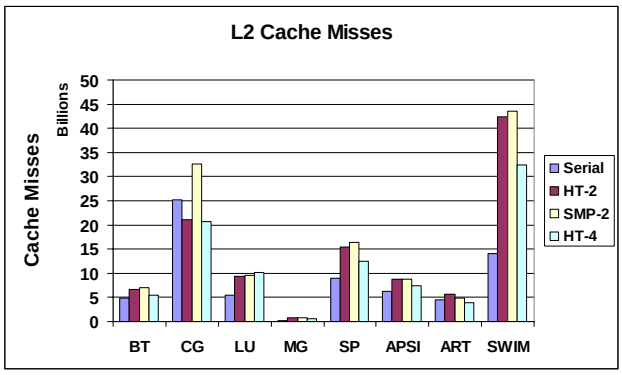

Figure 8. Second level cache misses.

Researchers at Dell published results for some benchmarks from the Message Passing Interface (MPI) version of the NAS Parallel Benchmarks running on a cluster of HT-enabled systems [5]. They conclude that the necessity of doubling the number of processes for HT can degrade performance by increasing demand on the interconnect subsystems.

Very recently, Liao et al. [6] evaluated OpenMP on chip multithreading platforms. They devised several experiments in order to get a better understanding of the behaviour of current OpenMP on such architectures. Our results on OpenMP Overhead are consistent with their findings on a Dell 450 machine. Their NAS and SPEComp benchmark results are fairly consistent, however their testing methodology involved using two physical processors running two threads for HT-2 tests, resulting in a partial load situation, while our results were performed with the load entirely on a single HT-enabled processor. In addition, our newer system has lead to some performance improvements over the system used by Liao et al. [6].

The work in [15, 8] have been the only work on devising algorithms for OpenMP loop scheduling [15], and thread pairing [8]. In [15], the authors focused on tuning the behaviour of OpenMP applications executing on SMPs with SMT processors. They proposed a self-tuning loop scheduler to react to behaviour caused by inter-thread data locality, instruction mix and SMT-related load imbalance. McGregor and his colleagues [8] introduced new scheduling policies that use runtime performance information to identify the best mix of threads to run across processors and within each processor. They have achieved 7 to 28% improvement over the Linux scheduler.

5. Conclusions and Future Work

In this paper, we present performance of different OpenMP constructs on dual and quad HT-based Intel Xeon servers under 2.4 and 2.6 kernels. We discovered that the overhead of OpenMP constructs with HT enabled is an order of magnitude larger than when HT is off. This may affect of the performance of applications with HT. Meanwhile, the 2.4.22 kernel incurs less overhead than the 2.6.9 in terms of synchronization and loop scheduling.

We also show the performance of some well-known scientific applications from NAS OpenMP and SPEC OMPM2001 suites on a range of system sizes with kernel 2.6.9 on a dual Hyper-Threaded SMP. Our performance results indicate majority of applications benefit from having a second thread in one-processor situations. However, only a few applications enjoyed performance gain of HT on both processors. By collecting data from hardware performance counters, we analyzed the effect of HT on the trace cache misses and its delivery rate.

The decisions made by the scheduler is crucial to the performance of HT. Our results indicate that the new O(1) scheduler in 2.6.9 kernel is still not efficient. We will experiment with other schedulers in Linux, and plan to devise optimal and adaptive schedulers. We intend to continue our study by observing the performance of multiprogramming workloads under full and partial load. We will report these results in the near future.

Acknowledgments

The authors would like to thank the anonymous referees for their insightful comments. This work was supported by grants from Natural Sciences and Engineering Research Council of Canada (NSERC), Queen's University, Canada Foundation for Innovation (CFI), and Ontario Innovation Trust (OIT).

References

[1] N.R. Fredrickson, A. Afsahi, and Y. Qian, "Performance Characteristics of OpenMP Constructs, and Application Benchmarks on a Large Symmetric Multiprocessor", In *Proceedings of the 17th Annual ACM International Conference on Supercomputing (ICS'03)*, June, 2003.

[2] Intel Inc., Intel VTune Performance Analyzer, http://www.intel.com/software/products/vtune, 2005.

[3] H. Jin, M. Frumkin, and J. Yan, "The OpenMP Implementation of NAS Parallel Benchmarks and its Performance, Report NAS-99-011", *Nasa Ames Research Center*, October, 1999.

[4] R. Kalla, B. Sinharoy, and J. Tendler, " IBM POWER5 Chip: A Dual-Core Multithreaded Processor", *IEEE Micro*, 24(2):40-47, March, 2004.

[5] T. Leng, R. Ali, J. Hsieh, V. Mashayekhi, and R. Rooholamini, "An Empirical Study of Hyper-Threading in High Performance Computing Clusters", in *Proc. of the Third LCI International Conference on Linux Clusters: The HPC Revolution*, October, 2002.

[6] C. Liao, Z. Liu, L. Huang, and B. Chapman, "Evaluating OpenMP on Chip MultiTreading Platforms", In *Proceedings of First International Workshop on OpenMP (IWOMP 2005)*, June, 2005.

[7] D.T. Marr, F. Binns, D.L. Hill, G. Hinton, D.A. Koufaty, J.A. Miller, and M. Upton, "Hyper-Threading Technology Architecture and Microarchitecture", *Intel Technology Journal*, Vol. 6, Issue 01, February, 2002.

[8] R.L. McGregor, C.D. Antonopoulos, and D.S. Nikolopoulos, "Scheduling Algorithms for Effective Thread Pairing on Hybrid Multiprocessors", In *Proceedings of 19th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2005)*, April, 2005.

[9] L.W. McVoy and C. Staelin, "lmbench: Portable Tools for Performance Analysis", In *USENIX Annual Technical Conference*, 1996, pp. 279-294.

[10] F.J.L. Reid and J.M. Bull, "OpenMP Microbenchmarks Version 2.0", In *Proceedings of 6th European Workshop on OpenMP (EWOMP'04)*, October, 2004.

[11] SPEC OMP Benchmark Suite, http://www.spec.org/omp/.

[12] X. Tian, Y.K. Chen, M. Girkar, S. Ge, R. Lienhart, and S. Shah, "Exploring the Use of Hyper-Threading Technology for Multimedia Applications with Intel® OpenMP Compiler", In *Proceedings of International Parallel and Distributed Processing Symposium (IPDPS'03*) April, 2003.

[13] N. Tuck and D.M. Tullsen, "Initial Observations of the Simultaneous Multithreading Pentium 4 Processor", In *Proceedings of the 2003 International Conference on Parallel Architectures and Compilation Techniques, (PACT'2003)*, September, 2003.

[14] D. Tullsen, S. Eggers, and H. Levy, "Simultaneous Multithreading: Maximizing On-Chip Parallelism", in *Proceedings of 22nd Annual International Symposium on Computer Architecture (ISCA'95)*, ACM, 1995.

[15] Y. Zhang, M. Burcea, V. Cheng, R. Ho, V. Cheng, and M. Voss, "An Adaptive OpenMP Loop Scheduler for Hyperthreaded SMPs", In *Proceedings of 16th International Conference on Parallel and Distributed Computing Systems (PDCS-2004)*, November, 2004.