A Comprehensive Analysis of OpenMP Applications on Dual-Core Intel Xeon SMPs

Ryan E. Grant Ahmad Afsahi
Department of Electrical and Computer Engineering
Queen's University, Kingston, ON, CANADA K7L 3N6
ryan.grant@ece.queensu.ca ahmad.afsahi@queensu.ca

Abstract

Hybrid chip multithreaded SMPs present new challenges as well as new opportunities to maximize performance. Our intention is to discover the optimal operating configuration of such systems for scientific applications and to identify the shared resources that might become a bottleneck to performance under the different hardware configurations. This knowledge will be useful to the research community in developing software techniques to improve the performance of shared memory programs on modern multi-core multiprocessors.

In this paper, we study a two-way dual-core Hyper-Threaded (HT) Intel Xeon SMP server under single program and multi-program multithreaded workloads using the NAS OpenMP benchmark suite. Our performance results indicate that in the single-program case, the CMP-based SMP and CMT-based SMP configurations have the highest average speedup across all of the applications. The most efficient architecture is a single HT-enabled dual-core processor that is almost comparable to the performance of a 2-way dual-core HT-disabled system.

1. Introduction

Increasing energy consumption and excessive heat generation resulting from high CPU clock rates and increased chip density has driven the processor industry to develop aggressive *Chip Multithreading* (CMT) [15] processors for general-purpose applications. The Intel Xeon [5], IBM Power5 [8], Intel Itanium 2 [11], and Sun UltraSPARC T1 [16] are examples of such microprocessors.

CMT processors combine *Chip Multiprocessing* (CMP) [4] and *Simultaneous Multithreading* (SMT) [18] to provide better support for *thread-level parallelism*. CMPs contain multiple cores on a single chip allowing more than one thread to be executed at a time. Each core has its own resources as well as shared resources, such as the memory bus. The extent of shared resources varies between different implementations.

SMT is a technique that allows multiple independent threads to execute different instructions each cycle. Intel Hyper-Threaded (HT) processors [10] are implementations of SMT. An HT-enabled core 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. Numerous complex interactions among the shared resources may affect the performance of multithreaded applications running on SMTs [17, 3].

CMT-based *Symmetric Multiprocessors* (SMPs) present new challenges as well as new opportunities to maximize performance provided the resources available could be shared efficiently. As the number of possible concurrently executing threads increases the sharing and interaction amongst the threads must be considered. The increased thread load results in heavy demand on the cache subsystem, growing numbers of bus transactions, and more stall cycles [2] that may significantly affect the performance of OpenMP [13] applications.

To the best of our knowledge, this paper is the first to study a real two-way Intel Xeon CMT-based SMP server running OpenMP applications. The Intel Xeon CMT processors studied in this work have two distinct cores with separate 2MB L2 caches. Each core has two hardware contexts, when enabled. The introduction of dual-channel main memory further complicates the situation with chip multithreaded SMPs. The layout of the Intel dual-core Xeon processors has an additional bottleneck as each individual physical chip shares a memory bus amongst its two cores.

This paper contributes by evaluating the performance of various multithreaded hardware configurations under a

number of parallel applications in the NAS OpenMP suite [7]. We consider the effects of resource sharing within the processors and/or cores on system performance by collecting data from the hardware performance counters. We attempt to pinpoint the architectural limitations of different multithreaded architectures available in such a CMT-based SMP system by observing its overall cache performance, TLB miss rates, stalled CPU cycles, branch prediction rate, bus transactions, and overall CPI. The work presented here is the first step towards devising optimal schedulers to improve the performance of multithreaded applications running on emerging multithreaded, multi-core architectures.

Our performance results indicate that for single-program workloads, the CMP-based SMP and CMT-based SMP configurations have the highest average speedup across all of the applications. However, the CMT architecture despite having half as many available computational resources has only a 13.6% slowdown over the CMP-based SMP case.

The rest of this paper is organized as follows. Related work is discussed in Section 2. Our experimental framework is described in Section 3. Section 4 presents our performance results and analysis. Finally, the paper is concluded in Section 5 with some thoughts on future research.

2. Related Work

SMT [18], CMP [4], and CMT [15] architectures have been well studied and analyzed by researchers to determine their potential pitfalls. However, most of the research has been done with single-threaded applications, simulations, or analytical methods. Some work has been reported on multi-program, single-threaded workloads [17, 1] as well as multithreaded parallel applications [2, 19, 17, 9, 3]. Our work in this paper concerns parallel workloads on real CMT-based SMP systems.

In the analysis of SMT implementations on real machines, Tuck and Tullsen [17] analyzed the Intel Pentium 4 HT processor. They found that up to a 25% improvement in the performance of parallel applications could be achieved, but the Pentium 4 HT shows symbiotic [14] behavior due to cache and resource conflicts. Bulpin and Pratt [1] studied the performance of a dual-CPU HT processor. They found high cache miss rate could be detrimental on simultaneously executing threads. Grant and Afsahi [3] experimented with dual and quad HT-based Intel servers. They discovered majority of parallel 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. They identified the trace cache misses and its delivery rate as a potential performance bottleneck.

Liao et al. [9] evaluated OpenMP on Sun UltraSPARC IV chip multiprocessor servers. They devised several experiments in order to get a better understanding of the behavior of OpenMP on such architectures. They observed that the OpenMP implementation designed for traditional SMP architecture might not achieve good scalability. Zhu et al. [20] studied performance of basic OpenMP constructs on the IBM Cyclops-64 architecture, consisting of 160 hardware thread units in a chip.

Curtis-Maury and others [2] introduced new scheduling policies that use runtime performance information to identify the best mix of threads to run across processors and within each processor. In [19], Zhang and Voss focused on tuning the behavior of OpenMP applications executing on SMPs with SMT processors. They proposed a self-tuning loop scheduler to react to behavior caused by inter-thread data locality, instruction mix and SMT-related load imbalance.

3. Experimental Methodology

The experiments were conducted on a Dell PowerEdge 2850 SMP server. The PowerEdge 2850 has two dual-core 2.8GHz Intel Xeon EM64T processors (Paxville core), with a 12KB shared execution trace cache, and 16KB L1 shared data cache on each core. A private 2MB L2 cache is available for each core on the chip. There are 4GB of DDR-2 SDRAM on an 800 MHz Front Side Bus.

Using LMbench [12] we have measured the L1, L2, and main memory latencies of the processor at 1.43ns, 10.61 ns, and 136.85ns, respectively. The main memory read and write bandwidths are 3.57 GB/s and 1.77 GB/s, respectively. The system was tested to determine if any memory bandwidth differences existed between the operation of threads on a single physical chip and the operation of those same threads spread out to both physical chips. It was discovered that the main memory read and write bandwidths when using two physical chips are 4.43 GB/s and 1.61 GB/s respectively. The memory bandwidth of system is important, as it is a known source of performance limitation. Although memory latencies and bandwidth have a major effect on such commercial systems, they are unavoidable and conclusions made on the performance of such architectures must be made in an environment inherently bound by such memory limitations.

The operating system used for the testing was a Red Hat Linux Enterprise WS 4.1 distribution with Kernel 2.6.9-11. This kernel provided the option to limit the number of processors that can be activated by the operating system. 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 when determining scalability since

it better emulates a smaller system. To test the system in a variety of configurations some of the tests were run while masking off some of the available processors in the system, enabling us to test the performance using different thread distributions between the existing resources. The default Linux scheduler was used for assigning the individual threads amongst the specified processors.

3.1. Terminology

Figure 1 presents the labeling used for the hardware contexts in the HT-enabled and HT-disabled systems. Such labeling will help understand the hardware contexts available for use in each configuration.

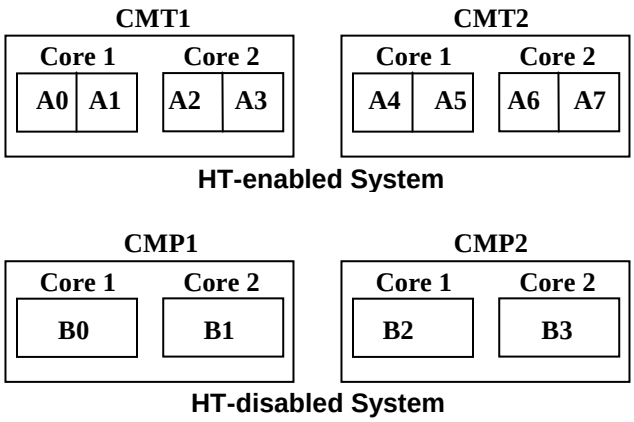

Figure 1. System configuration labeling.

Table 1 shows the various configurations used in our study. The basic terminology used to describe these configurations is comprised of three parts. The first part is either HT_{off} or HT_{on}, which describes the state of Hyper-Threading in the system. The second term indicates the total number of application threads that were used. The third term represents the number of physical processor chips used in the tests (i.e. the number of dual-core processors used, either 1 or 2).

Table 1. Configuration information

Terminology	H/W Contexts	Architecture
Serial	B0	Serial
HT_{on} -2-1	A0, A1	SMT
HT_{off} -2-1	B0, B1	CMP
HT_{on} -4-1	A0, A1, A2, A3	CMT
HT_{off} -2-2	B0, B2	SMP
HT_{on} -4-2	A0, A1, A4, A5	SMT-based SMP
HT_{off} -4-2	B0, B1, B2, B3	CMP-based SMP
HT_{on} -8-2	A0, A1, A2, A3, A4, A5, A6, A7	CMT-based SMP

3.2. NAS OpenMP Benchmarks

The NAS Parallel Benchmarks have been widely used to characterize high-performance computers. We used the NAS OpenMP benchmark suite (version 3.2) [7] in our experiments. 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, FT, SP, and LU benchmarks. Class B is large enough to provide realistic results, while ensuring that the working set fits in memory.

Application characteristics were gathered at run-time using the hardware performance counters available on the Intel Xeon processor. We collected the data using Intel VTune Performance Analyzer version 7.2 [6]. The Intel Fortran and C/C++ compilers (version 8.1) were used to build the benchmark applications. The default compiler flags for the NAS make utility were used during compilation.

4. Experimental Results and Analysis

In order to present the results in a fair manner, the configurations must be divided into several groups. The HT_{on}-2-1 configuration must be examined in comparison to the serial case. In the second group, the HT_{off}-2-1 configuration can be compared directly to the HT_{on}-4-1 configuration, the only difference between the two being the presence of HT. The third group comprises the HT_{on}-4-2 and HT_{off}-2-2 configurations, which can also be compared to the second group, as the difference between these two groups is the use of both physical chips, although only at 50% usage to create resources similar to those available to the second group. The fourth and final group is the HT_{off}-4-2 and HT_{on}-8-2 group, which utilizes all of the resources available in our platform with HT on and off. Comparisons may also be made between groups, but only in the context of performance per resources available to help determine the configurations that make the best use of the resources available to them.

4.1. Multithreaded, Single-Program Results

In this section, we present the wall clock performance of individual applications running under various multithreaded configurations. Their impact on architectural components is shown in Figure 2.

4.1.1. Cache Performance. The performance of the cache is very important to overall system performance, and can illustrate potential benefits and drawbacks of certain system configurations. The L1 cache miss rates are flat across the different configurations. This seems counter-intuitive but is a byproduct of the benchmark applications

themselves as they use a large amount of infrequently changing variables in their calculations, and only a small number of new variables for each loop. This means that a large number of L1 cache requests are hits, while only a small number are misses, requesting the few new variables required for the next loop. This leads to a good L1 cache miss percentage due to the large number of requests.

Examining the HT_{on}-2-1 configuration, we can see that it has relatively good trace cache performance and excellent overall hit rates within each level of cache for all of the applications with the exception of the L2 miss rate for the LU benchmark. The second group is similar to the third group in trends in memory performance, with the HT_{on} configurations having a higher miss rate than the HT_{off} configurations. This is not unexpected as the HT_{on} configurations have less memory available to the system per execution thread, thereby causing more cache evictions, which are not offset by the sharing of data between the threads.

The final group of HT_{off}-4-2 and HT_{on}-8-2 has fairly comparable memory performances, with the HT_{on}-8-2 having the advantage only on the trace cache of the CG and MG benchmarks while the HT_{off}-4-2 configuration has the advantage in the L2 miss rate on the LU benchmark.

4.1.2. TLB Performance. ITLB misses rise significantly between the different groups, with the number of ITLB misses increasing as the complexity and resources of the architectures increase. DTLB misses are relatively flat across all groups indicating that the increases in complexity do not significantly impact the performance of the DTLB.

4.1.3. Stall Cycles. The examination of the number of cycles spent stalled due to memory order clearing, mis-predicted branches, lack of instructions in the trace cache, or the delay caused by the need to load data in from memory can help to determine why some of the configurations are performing the way they do. The number of cycles spent in a stalled state for the HT_{on}-2-1 configuration is poor relative to the other configuration groups. This is most likely due to its lack of computational resources. Group 2, 3 and 4 show similar patterns with the HT_{on} configurations having more stalled cycles than the HT_{off} configurations. This is an indication of thread contention for shared resources in the cores. Interestingly, the configurations from group 3 are worse in terms of percentage of stalled cycles throughout all of the tests compared to group 2. The average percentage of stalled cycles for the HT_{off} configurations is 0.83%, while the average for the HT_{on} configurations is 1.62%.

4.1.4. Branch Prediction. Branch prediction rates are excellent for almost all of the benchmarks across all configurations, with the exception of the HT_{on} configurations from groups 2 and 3 for CG and HT_{on}-8-2 for MG. This helps to explain the number of stalled cycles and consequently the high CPI that the HT_{on} configurations in groups 2 and 3 have for the CG benchmark.

4.1.5. Bus Transactions. When examining the bus transaction characteristics of the configurations it is clear that group 1 is the only group that has the memory bandwidth capacity left over to perform any kind of pre-fetching activities, spending ~50% of its time in 4 of 5 of the benchmarks pre-fetching data into the cache. The only other instance of significant pre-fetching is the HT_{on}-8-2 configuration for the CG benchmark.

4.1.6. CPI. When examining the Cycles per Instruction (CPI) of the different configurations, many of the observations made in the previous sections can be observed impacting the efficiency of the system. It is interesting to note that the high CPIs of the HT_{on} configurations from groups 2 and 3 running the CG benchmark correlate directly to very poor branch prediction rates and relatively high L2 cache miss rates, which combine to give these two configurations higher CPIs than those in their respective groups. The poor CPI of the HT_{on}-8-2 configuration executing MG also corresponds to a poor branch prediction rate but without the high L2 cache miss rate. This makes sense as a high branch misprediction rate would cause many flushes of the execution pipeline and therefore reduce overall efficiency. The configurations with high relative CPIs also have a correspondingly lower speedup.

4.1.7. Wall Clock Performance. The NAS benchmarks were run through a series of ten independent trials, with minimal variance between tests (<~1-5%). The speedup of NAS applications is depicted in Figure 3, while the average speedup across all applications for various multithreaded architectures is shown in Table 2.

Overall, the CMP-based SMP and CMT-based SMP configurations have the highest average speedup across all of the applications. The runtimes of the NAS benchmarks show an interesting trend that is new to the dual-core Intel Xeon architecture, that the use of SMT on a CMT chip can be extremely beneficial to the performance of a system. Of particular interest are the results for HT_{on}-4-1. In this case, the overall performance of a single CMT chip is comparable (13.6% slowdown) to the performance of two dual-core processors operating with HT_{off} (CMP-based SMP) despite having half as many available computational resources.

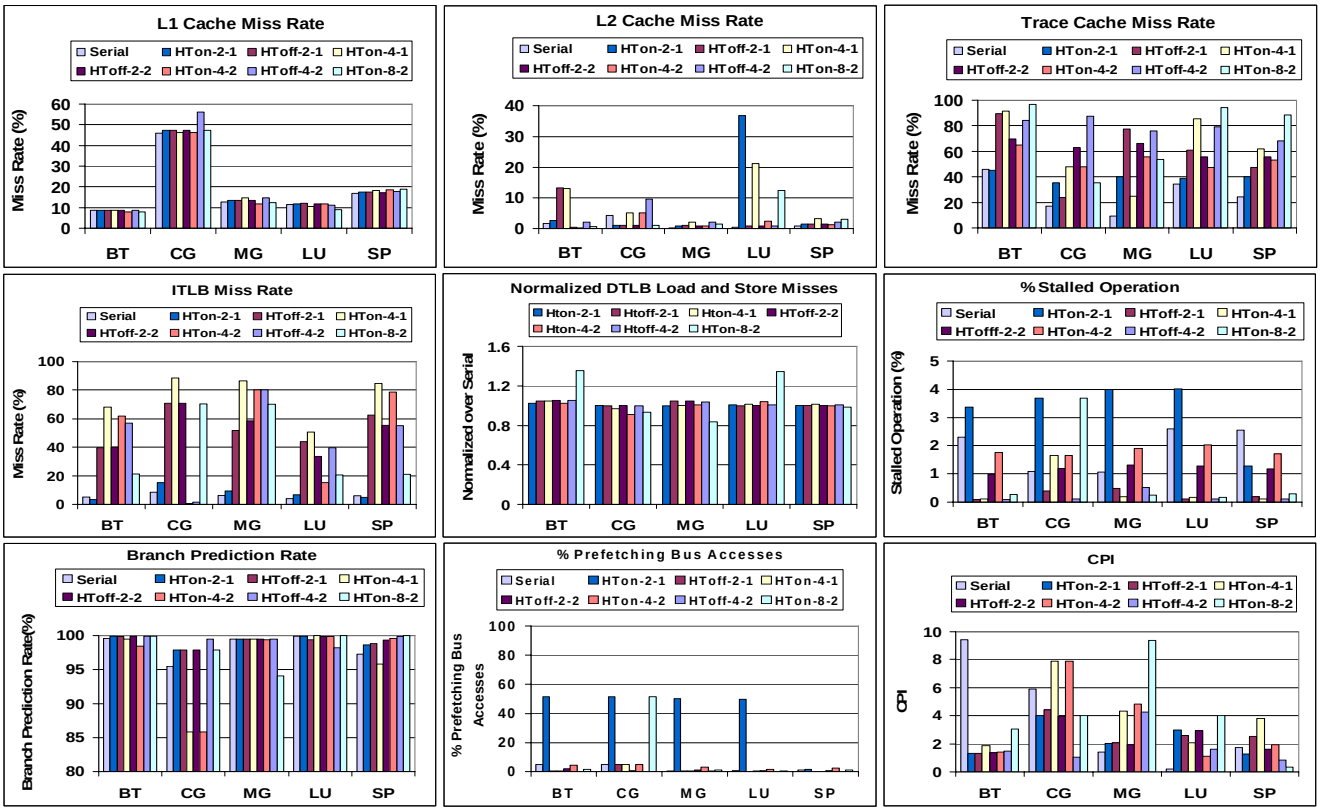

Figure 2. L1, L2, and trace cache miss rates, ITLB miss rate, DTLB load and store misses normalized to serial case, % of execution spent in a stalled state, branch prediction rate, % of prefetching bus accesses, and CPI of NAS benchmarks.

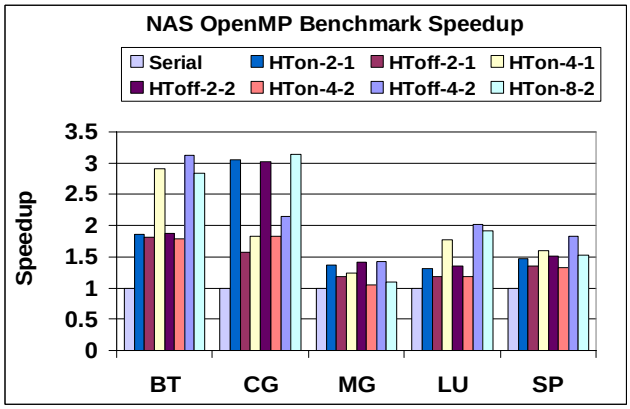

Figure 3. Speedup for NAS OpenMP applications.

When overall processor resources are increased to utilize two dual-core processors with HT enabled, the results are similar to previously observed HT-related behavior in that the overall effect reduces computational speed and results in a slowdown of approximately 6.7% versus HT_{off} [3]. Therefore, we can conclude that HT is of benefit when enabled for smaller numbers of processors (<4). However, the efficiency of HT with fewer physical processors has increased from previous observations most likely due to the improvements in memory bus speed.

Except for the CG benchmark, the performance of the HT_{on}-8-2 case is worse than the HT_{off}-4-2 case. To better understand the reasons behind this we examine the CG application in detail. In general, the HT_{on}-8-2 setup results in less total bus accesses than the HT_{off}-4-2 case, with an L1 cache miss rate of 47.1% versus 56.2% for the HT_{off}-4-2 case. This, coupled with an L2 cache miss rate of 1% versus 9.6% translates into a higher number of non-prefetching bus accesses from the HT_{off}-4-2 case. The HT_{off}-4-2 case has a lower CPI of 1.04 versus the HT_{on}-8-2's CPI of 4.02 that would imply that the performance of the HT_{off}-4-2 case should be superior. However, a large number of bus transactions in the HT_{on}-8-2 case are speculative prefetching (51.2% of all bus accesses) while the vast majority of bus transactions for the HT_{off}-4-2 case are not the result of prefetching. This leads to much more speculative execution in the HT_{on}-8-2 case, which is not accounted for in the CPI as it is counted as cycles per instruction committed. The trace cache performance of the HT_{on}-8-2 system is worse than the HT_{off}-4-2 case in all of the benchmarks, except for CG and MG, with the HTon-8-2 configuration having a major advantage of 35.6% miss rate versus the HT_{off}-4-2's miss rate of 87.3% for CG.

Table 2. Average speedup for architectures

SMT	CMP	CMT	SMP	SMT based SMP	CMP based SMP	CMT based SMP
1.81	1.42	1.87	1.83	1.43	2.11	2.10

4.2. Multithreaded, Multi-Program Results

The performance of the given architectural configurations is also of interest in a multi-program environment. The results shown in Figure 4 were collected using the same configurations as in the previous section, but utilized more than one concurrent program execution at a time to examine the ability of the architectures to handle complimentary and uncomplimentary workloads of multiple programs. Two NAS benchmarks were selected for this task: the FT benchmark, which is a Fourier transform operation requiring mostly computational resources and limited memory resources; and the CG benchmark, which requires significant memory resources.

Three separate tests were conducted, one with two copies of CG, one with two copies of FT and one with one copy of CG and one copy of FT. The maximum number of execution threads available to each system configuration was used, with the threads being distributed evenly between the executing programs.

4.2.1. Cache Performance. The L1 cache miss rates are relatively stable across the different configurations. However, the L2 cache miss rates show that the HT_{on}-2-1 configuration has difficulty achieving a high hit rate for the CG, as does the HT_{on}-8-2 configuration. In general, all of the HT_{on} configurations have a worse L2 miss rate than their HT_{off} equivalents, except for a few cases with the CG/CG workload.

The trace cache miss rate finds the HT_{off} configurations for both groups 2 and 3 are better than the HT_{on} configurations for both the CG/FT and CG/CG workloads, with the HT_{on} configurations having an advantage in the FT/FT workload. The advantage for the HT_{on} configurations for the FT/FT workload for group 2 is significant, but the advantage is even greater for the HT_{on} configuration of group 3. Finally, group 4 shows that there is no advantage to the HT_{on}-8-2 configuration in terms of trace cache misses for any of the workloads.

4.2.2. TLB Performance. The HT_{on} configurations suffer from excessive ITLB misses in groups 2 and 3 when running CG. In general, the CG and FT benchmarks benefit from running together versus their identical pairs. The HT_{on} configuration from group 3 also has difficulties with its DTLB for the FT/FT and CG/CG workloads. This implies that the execution units are potentially being starved of instructions, but further investigation into the amount of stalling that occurs due to these misses is required to determine the real effect of this finding.

4.2.3. Stall Cycles. The amount of total execution cycles spent in a stalled state for this multi-application workload is surprising. When running a complimentary workload (CG/FT) we can see that a significant amount of time is spent in a stalled state. From this, we can infer that the system is having a very difficult time providing the programs with the required resources, possibly switching the processors on which the programs are running frequently. This implies that there could potentially be significant gains in performance made if the exact causes of this stalled operation can be determined and rectified.

4.2.4. Branch Prediction. When examining the branch prediction rate we can see that the HT_{off} configurations from groups 2 and 3 are both worse than the HT_{on} configurations for all of the tests except for the CG/CG workload. Group 1 has relatively good branch prediction across the workloads, and group 4 shows that there is only a marginal difference between the branch prediction rates between the HT_{on}-8-2 and HT_{off}-4-2 configurations.

4.2.5. Bus Transactions. The prefetching activities being undertaken by each of the configurations reinforce the stalled operation results examined earlier in this section. The configurations spend a significant amount of time prefetching when running the CG/FT workload. From this we can infer that the source of the stalling is not due to memory bandwidth issues, but instead can be attributed to other factors such as pipeline flushes.

4.2.6. CPI. The CPI results for each workload indicate that despite the high number of execution cycles in which the systems are stalled for the CG/FT workload, the actual CPI of the HT_{on} configurations does not suffer significantly. With the exception of the CG/CG workload, the HT_{on} configurations for groups 2 and 3 are better than the HT_{off} configurations in terms of CPI. Group 4 shows that the HT_{on}-8-2 configuration is worse for all workloads than the HT_{off}-4-2 configuration.

4.2.7. Wall Clock Performance. The results clearly indicate that there is a tangible performance benefit to running compute bound and memory bound applications separately as the performance of both applications is better in such a configuration for most architectures. Surprisingly, the best performer of any configuration is HT_{on}-4-2, which is the fastest overall in two of the three configurations. The HT_{on}-8-2 configuration is the fastest for the CG/FT test but only by a small margin. This indicates that for multi-application workloads, some HT_{on} configurations can offer a significant performance benefit.

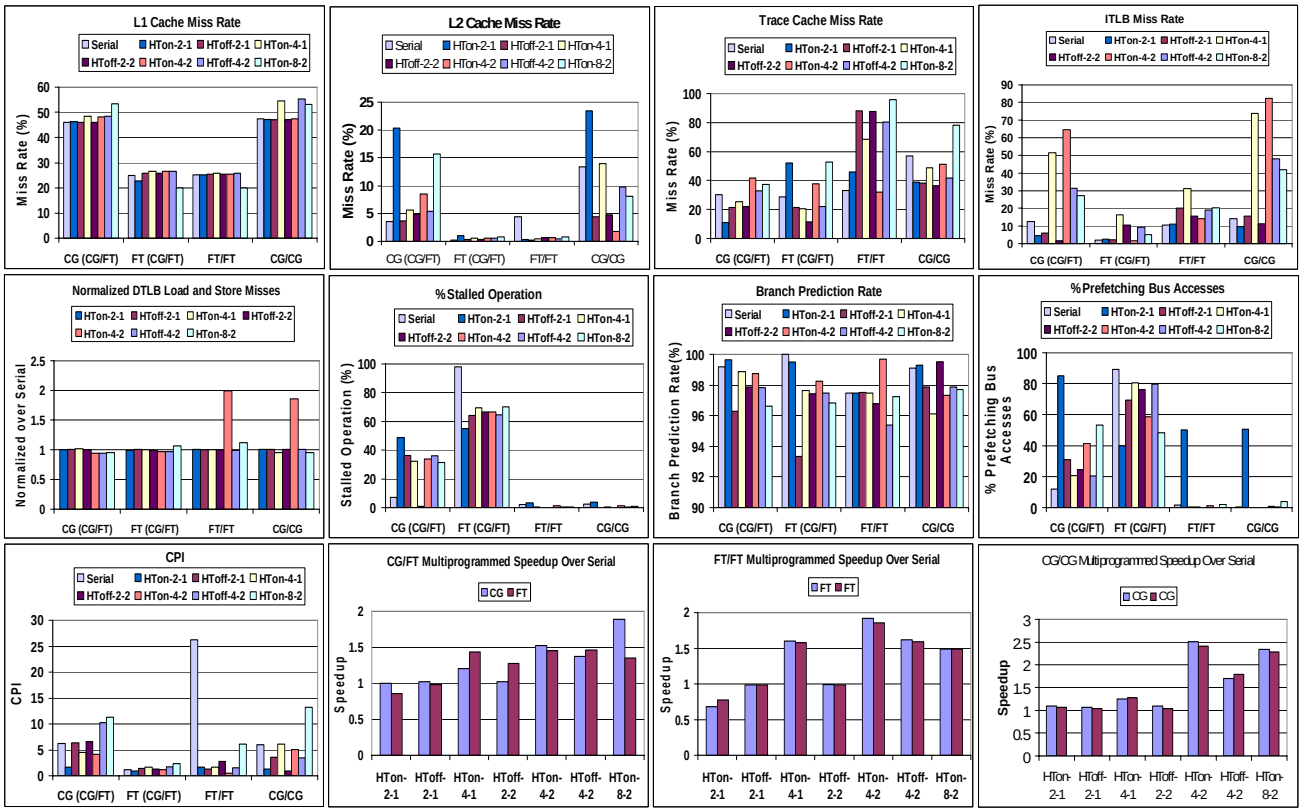

Figure 4. L1, L2, and trace cache miss rates, ITLB miss rate, DTLB load and store misses normalized to serial case, % of execution spent in a stalled state, branch prediction rate, % of prefetching bus accesses, and CPI and speedup of multi-program workloads.

4.3. Cross-Product Multi-Program Results

The configurations were tested using pairs of applications, and completed for all possible two-program configurations. The program pairs were run with enough evenly distributed threads as to fully load the architecture under test. The testing setup and methodology used for the cross-product pairs was identical to that used in section 4.2 with the exception of the larger set of applications that were used. The results are shown in a box and whiskers plot in Figure 5. The boxes in the figure represent the interquartile ranges of data (the 25th and 75th percentile of the data falls within the box), while the whiskers represent the maximum and minimum of the data.

From these results, we can conclude that the HT_{off}-4-2 (CMP-based SMP) architecture provides the overall best performance for the majority of program pairs across all of the benchmarking programs. However, for certain program pairs, the HT_{on} architectures can provide better overall performance. The performance of the CG benchmark when running with the BT benchmark on the HT_{on} architectures is significantly better than on the HT_{off} architectures, which accounts for the large whiskers on the CG results for the HT_{on} architectures. However, BT does not see significant speedup when run in conjunction with CG on HT_{on} architectures.

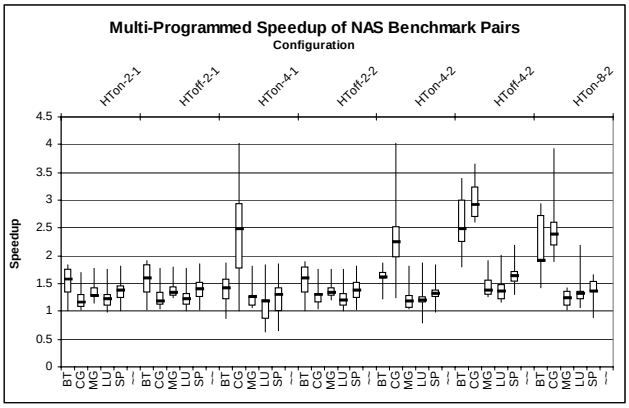

Figure 5. Speedup of NAS Benchmark Pairs.

5. Conclusions and Future Work

Multi-core systems have become the mainstream in processor design. Such processors present new challenges in maximizing performance especially when a number of them are configured in an SMP fashion. This paper sought

to identify the optimal configurations for running OpenMP applications and to discover the shared resources that might become a bottleneck to performance under the different hardware configurations available in chip multithreaded SMPs. By determining the potential causes of poor scalability or performance on different architectures we can adapt future operating system schedulers to achieve optimal performance when running OpenMP applications.

In this paper, we presented the performance of scientific applications from the NAS OpenMP suite on a range of system configurations on a 2-way dual-core Intel Xeon SMP. By collecting data from hardware performance counters, we analyzed the effect of HT on the various system configurations. When utilizing all of the available system resources, most applications suffer from the increasing number of L2 cache misses, and resource contention when both dual-core processors use HT.

Our results indicate that the most efficient architecture is a single dual-core processor with HT enabled, in terms of total computing power per system resources available. However, only one application enjoyed a performance gain due to HT on both dual-core processors. We discovered that the CMP-based SMP provides the best overall performance for the majority of program pairs across all of the benchmarking programs for the multi-program cases.

The decisions made by the scheduler are crucial to the performance of multithreading architectures. We are currently experimenting with other schedulers to improve the performance of OpenMP parallel applications on such hardware platforms. It is clear that the CPI, L1 and L2 cache miss ratios are not foolproof indicators of machine performance. Therefore, we intend to use the knowledge presented here to devise methods of measuring the overall efficiency of the system to achieve excellent performance while ensuring high machine throughput.

6. Acknowledgments

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

References

[1] J.R. Bulpin and I.A. Pratt. Multiprogramming performance of the Pentium 4 with Hyper-Threading. In *3rd Annual Workshop on Duplicating, Deconstruction and Debunking (WDDD 2004)*, pages 53–62, 2004.

[2] M. Curtis-Maury, T. Wang, C.D. Antonopoulos and D.S. Nikolopoulos. Integrating Multiple Forms of Multithreaded Execution on SMT Processors: A Quantitative Study with Scientific Workloads. In *2nd International Conference on the Quantitative Evaluation of Systems (QEST'05)*, 2005.

[3] R.E. Grant and A. Afsahi. Characterization of multi-threaded scientific workloads on simultaneous multithreading Intel processors. In *Workshop on Interaction between Operating System and Computer Architecture (IOSCA 2005)*, 2005.

[4] L. Hammond, B.A. Nayfeh, and K.A. Olukotun. A single-chip Multiprocessor. *IEEE Computer, 30(9):79-85*, 1997.

[5] Intel Corp., Intel Xeon processor. 2006.

[6] Intel Corp., Intel VTune performance analyzer, 2006.

[7] H. Jin, M. Frumkin, and J. Yan. The OpenMP implementation of NAS parallel benchmarks and its performance. NAS Technical report NAS-99-011, *NASA Ames Research Center*, 1999.

[8] R. Kalla, B. Sinharoy, and J.M. Tendler. IBM Power5 chip: a dual-core multithreaded processor. *IEEE Micro, 24(2):40-47*, 2004.

[9] C. Liao, Z. Liu, L. Huang, and B. Chapman. Evaluating OpenMP on chip multitreading platforms. In *1st International Workshop on OpenMP (IWOMP 2005)*, 2005.

[10] 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, 2002.

[11] C. McNairy and R. Bhatia. Montecito: A dual-core, dual-thread Itanium processor. *IEEE Micro, 25(2):10-20*, 2005.

[12] L.W. McVoy and C. Staelin. Lmbench: Portable tools for performance analysis. In *1996 USENIX Annual Technical Conference*, pages 279-294, 1996.

[13] OpenMP Architecture Review Board, OpenMP Specification Version 2.5, 2005.

[14] A. Snavely and D.M. Tullsen. Symbiotic job scheduling for a Simultaneous Multithreading Processor. In *9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'IX)*, pages 234-244, 2000.

[15] L. Spracklen and S.G. Abraham. Chip multithreading: opportunities and challenges. In *11th International Symposium on High-Performance Computer Architecture (HPCA-11)*, pages 248-252, 2005.

[16] Sun Corp., Sun UltraSPARC T1 Processor, 2006.

[17] N. Tuck and D.M. Tullsen. Initial observations of the simultaneous multithreading pentium 4 processor. In *12th International Conference on Parallel Architectures and Compilation Techniques (PACT 2003)*, pages 26-34, 2003.

[18] D.M. Tullsen, S.J. Eggers and H.M. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. *In 22nd Annual International Symposium on Computer Architecture (ISCA'95)*, pages 392-403, 1995.

[19] Y. Zhang and M. Voss. Runtime empirical selection of loop schedulers on Hyperthreaded SMPs. In *19th IEEE/ACM International Parallel and Distributed Processing Symposium (IPDPS 2005)*, 2005.

[20] W. Zhu, J.D. Cuvillo, and G.R. Gao. Performance characteristics of OpenMP constructs on a many-core-on-a-chip architecture. In *2nd International Workshop on OpenMP (IWOMP 2006)*, 2006.