Analysis and Improvement of Performance and Power Consumption of Chip Multi-Threading SMP Architectures

By: Ryan Eric Grant

A thesis submitted to the Department of Electrical and Computer Engineering in conformity with the requirements for the degree of Master of Science (Engineering)

Queen's University

Kingston, Ontario, Canada

August, 2007

Abstract

Emerging processor technologies are becoming commercially available that make multi-processor capabilities affordable for use in a large number of computer systems. Increasing power consumption by this next generation of processors is a growing concern as the cost of operating such systems continues to increase.

It is important to understand the characteristics of these emerging technologies in order to enhance their performance. By understanding the characteristics of high performance computing workloads on real systems, the overall efficiency with which such workloads are executed can be increased. In addition, it is important to determine the best trade-off between system performance and power consumption using the variety of system configurations that are possible with these new technologies.

This thesis seeks to provide a comprehensive presentation of the performance characteristics of several real commercially available simultaneous-multithreading multi-processor architectures and provide recommendations to improve overall system performance. As well, it will provide solutions to reduce the power consumption of such systems while minimizing the performance impact of these techniques on the system.

The results of the research conducted show that the new scheduler proposed in this thesis is capable of providing significant increases in efficiency for traditional and emerging multi-processor technologies. These findings are confirmed using real system performance and power measurements.

Acknowledgements

I would like to first thank my supervisor Dr. Ahmad Afsahi for his valuable feedback and support in conducting the research for this thesis and his assistance in assembling this document. Without his help this work would not have been possible.

Many thanks to the office staff of the ECE Department here at Queen's, particularly Ms. Debie Frasier and Ms. Bernice Ison for their invaluable help in administrative matters during the tenure of my Master's degree.

To my co-workers in the Parallel Processing Research Laboratory, Reza Zamani, Mohammad Rashti, and Ying Qian, thank you for your continued support over the last two years.

Finally, my deepest appreciation goes to my immediate and extended family for their support throughout the years. A special and heartfelt thanks goes to my loving and understanding wife Meagan without whom this would not be possible.

Table of Contents

List of Tables

List of Figures

Glossary

ADI – Alternating Direction Implicit

AMD – American Micro Devices Inc.

AMP – Asymmetric Multi-Processor

API – Application Programming Interface

BT – Block Tri-Diagonal

CFD – Computational Fluid Dynamics

CG – Conjugate Gradient

CMP – Chip Multi-Processor

CMT – Chip Multi-Threading

CPI – Cycles Per Instruction

CPU – Central Processing Unit

DDR – Double Data Rate

DVFS – Dynamic Voltage and Frequency Scaling

EP – Embarrassingly Parallel

EPI – Energy Per Instruction

FORTRAN – "IBM Mathematical FORmula TRANslating System"

FT – Fourier Transform

HPC – High Performance Computing

HT – Hyper-Threading

ITRS – International Technology Roadmap for Silicon Organization

LLP – Loop Level Parallelism

LU – Lower-Upper

MG – Multi-Grid

MPI – Message Passing Interface

OS – Operating System

SDRAM – Synchronous Dynamic Random Access Memory

SMP – Symmetric Multi-Processor

SMT – Simultaneous Multi-Threading

SP – Scalar Pentadiagonal

SPEC – Standard Performance Evaluation Corporation

TLP – Thread Level Parallelism

Chapter 1: Introduction

Recent developments in the computing industry have signaled a shift in microprocessor design philosophy towards the use of multiple processing cores. This allows a single device to execute multiple instructions simultaneously. The process by which multiple instructions are executed in parallel on multiple CPUs is referred to as parallel processing. There are two distinct types of parallelism, *instruction level parallelism* (ILP) and *thread level parallelism* (TLP). ILP exploits the potential of individual instructions to be executed simultaneously. ILP requires that the instructions of a process be executed simultaneously or out of order, and that code not be dependant on the previously executed instructions. In reality, ILP is of limited use because of the difficulties in coding algorithms in which there are limited amounts of inter-instruction dependence. As such, ILP can be very difficult to exploit because many sequential code sections occur in most algorithms, which lessens the overall effect that ILP can have upon runtimes.

TLP approaches the parallelization problem in a different way than ILP, by dividing processes into distinct threads of execution, which contain their own independent sequential instruction dependencies. This allows for algorithms to be coded in a more traditional manner and for parallelism to be more logically divided amongst separate threads. In addition, TLP is not as inherently limited in its upward parallelism potential as ILP. With ILP, there are only a finite number of instructions that can be executed in parallel in one time period, where with TLP, one can execute as many instructions as possible on each CPU during the same time period, given that there are enough threads to keep all of the processors busy. Because of the advantages of TLP and the larger potential benefits of exploiting it, the research and commercial sectors have taken an interest in TLP over ILP.

TLP can be taken advantage of using the concept of multi-threading. Traditionally, multi-threading has been used for multi-tasking, with a single CPU desktop computer running multiple processes in a time multiplexed execution that enables operating systems to perform multi-tasking. All major modern operating systems support multi-threading, including Microsoft's Windows operating system, Apple computer's OS X and

UNIX/Linux. Multi-threading can also be used in systems with multiple CPUs, as threads can execute simultaneously on each independent CPU. In addition to running multiple distinct processes, multi-threading can be used to divide a single computational task or process into several execution threads that can be run simultaneously for the purpose of using multiple CPUs to reduce process run times. Multi-processor systems such as *symmetric multi-processors* (SMP) which use shared memory have been available for many years. These systems use multiple traditional *central processing units* (CPUs) connected to a shared memory system.

Efforts to take advantage of TLP have been attempted using single-core multi-thread execution units commonly referred to as *simultaneous multi-threading* (SMT) [91] capable platforms. SMT equipped processors can simultaneously execute multiple threads by sharing processor resources and using a limited number of duplicate resources. This creates a significant amount of pressure on the memory system which limits the overall effectiveness of SMT technology. In order to increase the performance of systems, new designs are incorporating multiple complete independent processors on a single chip, eliminating the need for any sharing of actual chip architecture.

These new designs, called *chip multi-processors* (CMP) have been proposed for many years by the research community [35] but have only begun to appear on the commercial market in 2006. Both Intel Corp [40] and AMD [1] have released multi-core versions of their processors, with Intel Corp releasing a multi-core multi-threading CPU design. Sun Microsystems has taken this trend further with the UltraSparc T1 processor [86], offering six or eight cores on a single chip with each core capable of executing four threads simultaneously. These multi-core simultaneous multi-threading processors are referred to as c*hip multi-threading* (CMT) [83] processors.

The adoption of multi-processor systems in mainstream applications is reinvigorating interest in parallel computing as more systems move to take advantage of multiple cores and the performance of future computing systems relies increasingly on their ability to effectively parallelize complex tasks.

The corresponding increase in transistor density of new systems has also brought about significant challenges relating to power consumption and heat generation. The move to multiple cores has been motivated by the inability to increase individual core

clock speeds due to heat generation. The waste heat produced by processors is directly related to their power consumption. Therefore, there have been efforts made to reduce the power consumption of modern systems by introducing *dynamic voltage and frequency scaling* (DVFS) [22] as well as low level design revision in order to increase energy efficiency.

1.1 Motivation

With the trend toward massively parallel computing systems in the future, designers will conceivably continue to add additional cores and thread support in future designs. With this emerging trend, the analysis and evaluation of existing multi-threaded systems is important to better understand the performance benefits of parallel processing on commercially available platforms. In addition to performance concerns, the increasing power consumption of modern processors is of great concern to the research community as well as the commercial sector. Future forecasts of power consumption and thermal dissipation by the International Technology Roadmap for Silicon Organization (ITRS) [43] are as high as 100 W/cm^2 in 2010 and up to 250 W/cm^2 by 2020. This means that current design trends cannot continue to be effective as the forecasted power consumption for devices in the year 2020 is four times the allowable thermal generation density for future chip packages [43]. In fact, the amount of power consumed by servers throughout the world is estimated to rise by as much as 76% from 2005 to 2010 [50]. The forecasts for future server system power consumption for the next four years is detailed in Figure 1.1, with information taken from the 2006 ITRS Roadmap Report [43].

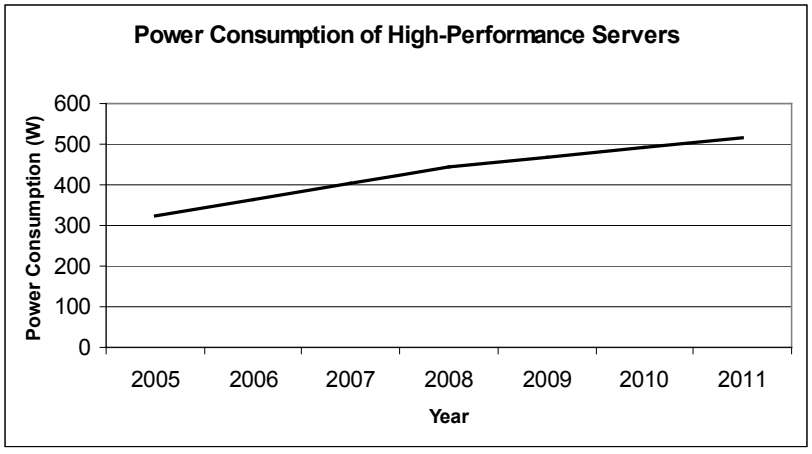


Figure 1.1: Server Power Consumption Forecasts

As such, it is important to understand the power consumption characteristics of existing multi-threading systems and determine the optimal performance/power savings trade-off point for system execution. The characterization data can then be used to fine tune existing systems through process scheduling as well as other software techniques such as compiler optimizations that can increase system performance and power consumption. This characterization information will also be of use to the semiconductor industry until the introduction of technologies that can possibly help to reduce the overall power consumption of integrated circuits, like emerging nano-technology. However, these technologies are not guaranteed to solve all of the current power consumption issues, and therefore work such as that presented in this thesis will continue to be of importance long into the future.

1.2 Contributions

The primary goal of this thesis is to garner some insight into the behaviour of emerging multi-threaded systems as they relate to power and performance, in order to help determine the best architecture for different types of system workloads. This information will aid the research community as well as the commercial sector in the utilization of such systems for real-world workloads. In addition, this thesis seeks to provide the basis for a software approach to better manage the power consumption of a simultaneous multi-threading processor system. It utilizes a program/profile independent approach designed to minimize the effect of operating system noise [89] for the purpose of performance/power savings. As such, it is extremely lightweight and useful in an *asymmetric multiprocessor* (AMP) [3] system that utilizes SMT. The resulting energy savings when utilizing this method equate into a significant cost savings when this method is applied to larger scientific processing systems, potentially saving hundreds of thousands of dollars in costs per year. Although the resultant energy savings and speedup may seem small at 5-15%, this is excellent in an HPC context as HPC applications provide only limited opportunities to increase performance or save energy as they typically utilize all of the processing systems resources for extended periods of intensive calculations. The idea of a protected CPU has been explored in a performance sense on older real-time systems [11] by using a protected CPU to process real-time tasks in order

to decrease system response time and ensure that tasks are completed before their deadline. The application of the protected CPU method in relation to non-real-time power/performance optimization and the use of SMT are to this author's knowledge, novel and unique. In addition, this thesis explores the effects of multi-application workloads and thread overloading on commercial CMP/SMT platforms. Each of these contributions are outlined below.

- An analysis of OpenMP [69] constructs and their effect on shared memory programs. This analysis was also conducted using different versions of the Linux kernel to evaluate the effect that OpenMP constructs have on the performance of the system using the traditional Linux scheduler and the recent O(1) scheduler.
- An in depth profiling of single-core multi-processor shared memory systems. The profiling of the NAS [44] and SPEC [82] benchmark suites running OpenMP on a shared memory system is a long and complex task. The collection of the profiling information takes a significant amount of time, with each benchmark run taking 20-32 hours and multiple runs required to capture and confirm the results for a single performance metric. Therefore, the collection of the dozens of metrics required in order to properly determine the effect of the various architectural elements on the system's performance is time consuming and produces an incredibly large volume of experimental data which must then be collated and analyzed. The collection and analysis of such data is of great use to the academic community as it is the basis from which further research can be performed.
- Profiling of chip-multithreaded SMP systems in multiple architectural configurations with multi-threaded single and multi-application workloads. This contribution is a more in depth profiling and analysis of multi-core systems in multiple configurations, (SMT enabled and SMT disabled) simulating systems of various sizes and determining the differences between execution on a single multi-core processor and

multiple multi-core processors in a shared memory system. Building upon the knowledge of previous profiles this contribution is significant in its scope and the lack of good profiling data available on modern multi-core shared memory systems.

- Development of a power/performance optimizing scheduling algorithm. The development of a scheduler to reduce the impact of operating system noise while reducing power consumption and increasing overall system performance is an excellent proof of concept for illustrating the potential benefits of intelligent scheduling in reducing the power consumption of systems while maintaining excellent performance.

1.3 Outline

This thesis is organized into seven chapters. The first is this introduction, followed by an in depth description of the background work in this area and the most important literature in the subject area. The third chapter describes in detail, the two major benchmarking suites for multi-processor shared memory systems that are used throughout this thesis. The fourth chapter explores the overhead incurred when running a parallelized application using the OpenMP shared memory interface API, and details the performance of multi-processor shared memory systems using simultaneous multi-threading. Chapter 5 continues the study of multi-processor systems by examining the performance of multi-core multi-processor systems with multi-threaded single and multi-application workloads, and the effect of thread overloading on such architectures. It concludes by investigating the effect of operating system noise on multi-core systems and laying the groundwork for the scheduler changes introduced in the next chapter. Chapter 6 details a study performed on the power usage of multi-processor systems and uses an alternative-scheduling algorithm that takes advantage of an asymmetric multi-processor system. Chapter 7 concludes the thesis with a brief summary and closing remarks as well as discussing future work to be done in this area.

Chapter 2: Background

To aid in understanding the material presented in this publication, the following background material provides an introduction to parallel computer architecture, multi-threading, the OpenMP [69] shared memory application programming interface (API), system characterization, operating system noise and power modeling of real systems.

2.1 Parallel Computing Architecture

Parallel computing can be accomplished using a variety of methods, each with its own advantages and disadvantages. Fundamentally, it is the usage of multiple CPUs working together on a given task for the purpose of decreasing the total running time of the task, or increasing the throughput of the system for a multi-program workload. The two basic classifications of parallel computing implementations are the shared memory multiprocessor and the message passing multiprocessor system. Shared memory processors can be of several types, the most prevalent being symmetric multiprocessors. Other types of shared memory systems include *non-uniform memory access* (NUMA) systems, which are most commonly *cache coherent non-uniform memory access systems* (ccNUMA) and *cache-only memory architecture* (COMA) systems. Multiple independent systems can be linked together to form a parallel processor system using methods like message passing, where information relating to the execution of a parallel program is passed between systems over a network using standardized message passing systems such as MPI [64]. By using such networked systems it is possible to create clusters of individual shared memory nodes in an attempt at increased performance. This thesis concerns a shared memory multiprocessor where all of the processing units are physically localized within a single system.

Traditional multi-processor systems require the use of multiple processor chips that are integrated onto a single motherboard. Each processor must share the available system memory. Typically this is done using a shared system bus. This system bus must then be arbitrated to ensure system consistency. There are a number of arbitration techniques that allow for pipelined bus accesses, and other techniques for a split-transaction bus that allows for delayed independent memory requests and replies.

In addition to bus arbitration, a multi-processor system must ensure cache consistency, as each processor typically has independent L1 and L2 caches. Cache consistency for shared memory multiprocessors is typically accomplished via snooping on the shared bus. Snooping involves additional circuitry for each microprocessor that monitors the bus for requests that correspond to data held in the local processors L1 or L2 cache. In the event that a request corresponds to a value in memory, the request is examined to determine the type of the request. A request for a read access to the data requires no action, while a request for a write to a location requires that the data be flushed from the cache. However, this is further complicated as the data held in the cache may be a more current version of the data than that held in memory, as the caches are typically a write-back cache instead of a write-through cache. Consequently, the processor may have to suppress the main memory response and respond to the request with the data in its cache. This will ensure data consistency and enable correct execution of a program when its threads are run across several processors in a system.

Multi-threading is an essential element in the use of parallel computing. It is a process by which a given task is broken down into separate executable units (threads) which can then be executed on several processing units at a given time to reduce the overall time of execution. Multi-threading using a time-division scheme has existed for many years. This method uses a single CPU and time-shares many threads, providing a multi-program environment.

Simultaneous multi-threading is a technology that allows more than one thread to run on a single processing core at one time. The goal of SMT is to increase the overall utilization of a CPU by using as many components of said CPU in a cycle as possible. However, numerous complex interactions among the shared resources may affect the performance of multi-threaded applications running on SMTs [33, 90]. This can create conflicts as multiple threads vie for the use of individual components and as such, the proper pairing of threads on an SMT is vital as complimentary threads can perform very well, while thread pairs that require the same processing resources in the same execution phases can cause slowdowns due to contention. *Hyper-Threading* (HT) [61] technology is an implementation of SMT on Intel processors. HT technology replicates essential CPU resources to allow the execution of two program threads at a time on the CPU.

Using HT technology, each physical processor is divided into two logical processors each of which has its own independent run-queue. These logical processors share the resources of the physical CPU including cache, execution units, translation look aside buffers, branch prediction unit and load and store buffers.

Processors are becoming available that have multiple processing cores integrated onto a single processor die. This technology called chip multi-processing [35] has been introduced by both AMD and Intel, with Intel also introducing a CMP that is HT enabled, creating a chip multi-threading architecture. The CMP and CMT architectures can then be integrated into a traditional SMP interconnect to allow multiple physical CMP or CMT chips to be integrated into a single system.

The amount of interconnectivity between the cores in a CMP can vary between implementations. The UltraSPARC IV's dual-cores [85] share only off-chip data paths, while the dual-cores in IBM Power5 [46] share L2 cache for faster inter-thread communications between the cores. The Intel Xeon [39] dual-core processors studied in this thesis are the first generation of dual-core processors from Intel and do not share an L2 cache, unlike the latest Intel Core [40] processors that have shared L2 caches.

2.1.1 Prior Art

This section reviews some of the most important publications in the field of parallel processing architecture. It also discusses the most recent publications that are related to the work presented in this thesis.

SMT processors have been studied in academia [91], and have appeared in mainstream processors from IBM such as the Power4 and Power5 processors [46], Intel's Core 2 [40] and Xeon [39] processors, and Sun Microsystems UltraSparc IV [85] and T1 [86] processors.

Tuck and Tullsen [90] analyzed the Intel Pentium 4 HT processor. They found that up to a 25% performance improvement on parallel applications could be achieved by using HT technology. They also confirmed Snavely et al.'s paper [81] by showing that the Pentium 4 Hyper-Threading technology demonstrates symbiotic behaviour due to cache and resource conflicts. Research at Intel has focused on Hyper-Threading with multimedia OpenMP applications [88]. This study found that enabling HT increased

performance, although it results in an increased 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.

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 [55]. They conclude that the necessity of doubling the number of processes for HT can degrade performance by increasing demand on the interconnect subsystems.

Recent work has also been done on CMP architectures and their emergence in the commercial sector. Work by De Vuyst et al. [21] and El-Moursy et al. [24] have examined the effect of scheduling on CMP architectures. They study the performance and energy optimization possibilities of such architecture. Researchers have also been examining scheduler performance and support in current operating systems [27, 67] as well as examining possible methods of increasing overall system throughput using CMPs [26]. Their investigations differ from the work in this thesis in that they use a simulated platform for their investigations, while we focus on real commercially available servers. Work relating to the cache contention and sharing issues brought about by chip multiprocessors have been studied, mainly focusing on cache sharing and partitioning [14] as well as the contention effects expected to occur in CMPs [13].

CMPs have also been examined with respect to their abilities executing single-threaded programs [93], particularly relating to the thread migration effects [16]. Researchers have also investigated the power performance of CMP architectures [56, 58, 77] although they also use simulations and only one of these papers examines the SMT/CMP architecture [56]. Other publications relating to CMT/CMP architectures have documented their evolution in the commercial sector [83]

2.2 Programming Shared Memory Parallel Applications

Programming of multi-threaded shared memory applications is normally done using one of two multi-threading APIs, OpenMP [69] or POSIX [38] threads. The POSIX threads API is based on the POSIX 1003.1 –1995 standard from ANSI and IEEE. It defines a set of thread creation and management operations as well as subroutines to

manage mutually exclusive code sections and communication between individual threads. POSIX threads are only available for coding in C, but wrappers make it possible to use POSIX threads in other languages such as FORTRAN with minimal associated overhead. However, programming with POSIX threads can be significantly more complicated than programming with OpenMP, as OpenMP does not require explicit thread creation, joining or destructions, and only uses relatively simple compiler directives for defining parallel sections of code. Therefore, OpenMP is typically the API of choice when developing shared memory applications.

The OpenMP API [69] is an industry standard interface that is used to develop shared memory parallel processing programs in Fortran, C and C++. It is a specification of compiler flags, environment variables and programming library functions that can be used to create parallelism on a system using a shared memory interface. OpenMP is an attempt to standardize shared memory parallel processing interfaces across a wide range of platforms. The predecessor to OpenMP, ANSI X3H5 [70] was never finalized and as a draft specification was never able to obtain true portability amongst different vendor's platforms due to implementation specific directives that were not portable between all of the system vendor's machines. OpenMP overcame this problem by revising the standard such that proprietary machine hardware implementations were not a prerequisite for any of the directives.

OpenMP defines a set of compiler directives which determine how threads are assigned portions of computational loops and directives used to control the execution of the threads, providing support for mutually exclusive sections of code and thread synchronization. The most used clauses of the OpenMP specification are the PARALLEL, DO/FOR, PARALLEL DO/FOR, BARRIER, SINGLE, CRITICAL, LOCK/UNLOCK, ORDERED and REDUCTION clauses. A description of each is detailed below [69].

- PARALLEL: Defines a region that should be executed by several threads. Each thread executes the code within the region unless it is flagged by an exclusion clause.

- DO/FOR: The DO/FOR clause is used inside of a PARALLEL region and defines loops that should be split up amongst the group of executing threads with each thread executing a portion of the iterations of the loop.
- PARALLEL DO/FOR: A shortcut clause that defines a PARALLEL region as well as starting a DO/FOR clause.
- BARRIER: A synchronization clause that causes threads to wait at the BARRIER until all threads have reached the BARRIER clause in code.
- SINGLE: The SINGLE clause prevents all of the executing threads in a PARALLEL region from executing the code within the SINGLE region. Only a single thread executes this code while the rest of the threads wait at the end of the region for the execution to complete, unless the NOWAIT option is specified.
- CRITICAL: CRITICAL clauses define regions that only one thread may execute at a time. Threads must wait at the beginning of a CRITICAL region if another thread is executing the code within the region.
- LOCK/UNLOCK: This clause is an alternative to using a CRITICAL region, and allows for mutual exclusion locking/unlocking of mutex type variables.
- ORDERED: ORDERED directives ensure that the code defined by the ORDERED region is executed in the same order as it would if the execution loops were run in a sequential manner.
- REDUCTION: This clause can be used within the PARALLEL region to define the method in which the variables used within the PARALLEL region are to be combined and returned when the PARALLEL region's execution is complete.

2.2.1 Prior Art

The two major programming languages used for parallel programming are C and high-performance FORTRAN. These languages support a combination of OpenMP and POSIX threads on many machine types and operating systems. FORTRAN was first

extended for parallelization in 1973 for the ILLIAC IV, extending its support to include a parallel do operation [65]. FORTRAN has remained an important language for parallel computing and has evolved into high performance FORTRAN (HPF), a language with parallel processing extensions. The High-performance FORTRAN specification [59] was released in 1993. The specifications were revised throughout the 1990s until the release of the current HPF 2.0 specifications in 1997.

The OpenMP API [20] standardized a shared memory programming interface for several computer languages. Researchers have developed compilers to take advantage of OpenMP instructions for clusters of SMPs [78]. Academia has also used OpenMP to develop several benchmarks for shared memory systems, the most prominent of them being the NAS OpenMP benchmark suite [44] and the SPEComp suite [6]. In addition to these major suites, other benchmarking programs such as EPCC [74] micro benchmarks and the LNLL benchmarks [54] evaluate the overhead incurred during OpenMP library calls.

Several publications have dealt with the parallelization of sequential code using OpenMP, including work by Couturier et al. [17] in which they parallelize a molecular dynamics program using OpenMP, in addition to work such as that by Renouf et al. [75] in which they use OpenMP to speed up the computation of the dynamics involved in large granular materials such as concrete or granular powders. OpenMP has also been used in conjunction with MPI to develop hybrid OpenMP+MPI programs for intra-node and inter-node parallelism for parallel jobs such as Monte Carlo simulations [80] and complex tasks such as high-resolution map visualization [15].

2.3 System and Application Characterization

Characterization of real systems can be a difficult task. Commercial systems are not released with their design files, or in depth technical details about their circuit layout. This creates challenges, as it is not possible to perform a low-level circuit analysis on such systems, and consequently, simulation of such systems is not accurate compared to the real run-time characterization of the systems. The characterization of such systems must therefore be accomplished using repeatable benchmark programs, and for the purposes of this thesis, they must be multi-threaded benchmarks.

The primary method of obtaining information about the performance of a system is a profiling tool like Vtune [41] that uses performance counters that are built into the CPU architecture. These hardware performance counters are system registers that can be set up to monitor a variety of relevant system events. These registers enable one to obtain very accurate counts of system events that are relevant to the systems performance, such as cache hit rates, the number of cycles spent actively executing the code, and a breakdown of the number of events that have occurred on each CPU, whether logical or physical in the system. This can lead to difficulty in determining the cause of some observed trends, as the amount of information that can be obtained from a real system is by its nature less than that which can be collected using simulation tools at the design file level. This is not to say that this method cannot provide a significant insight into the real operation of systems, only that the granularity of such observations can make it difficult to prove the exact reason behind performance abnormalities. The results obtained from testing on a real machine are devoid of any errors that could be caused by assumptions and shortcuts used in simulation software. Therefore, while the results of such real world tests can sometimes be difficult to explain, the results are an accurate representation of real world performance.

2.3.1 Prior Art

Several benchmarks have been developed to test the performance of real systems using OpenMP. Section 2.2.1 introduced the NAS, SPEC, EPCC and LNLL benchmark suites. We will discuss the most relevant and important publications that have dealt with shared memory systems and the characterization and evaluation of real systems using these benchmarks in this section.

Several research publications have been released dealing with the evaluation of shared memory systems using OpenMP. The evaluation of OpenMP construct overhead has been performed on large scale systems [9, 12, 28] as well as smaller scale SMPs [57]. Liao et al. [57] evaluated OpenMP on chip multithreading platforms, including the NAS and SPEC benchmarks for a Sun Fire V490 and a Dell Precision 450 workstation. They devised several experiments in order to get a better understanding of the behaviour of OpenMP on such architectures.

Evaluation of the OpenMP SPEC benchmark suite has been performed on the Fujitsu PrimePower 2000, SGI Origin 3800, and the HP PA-8700 systems by Saito et al. [76]. In 2001, Aslot et al. [5] evaluated the SPEC OpenMP benchmarks on a Sun Enterprise 450 SMP system. An evaluation of the SPEC OpenMP benchmarks was performed by Fredrickson et al. [28] in addition to the NAS OpenMP benchmark suite on a Sun Fire 15K SMP.

The NAS OpenMP benchmarks were first evaluated at the time that they were released by Jin et al.[44]. The NAS OpenMP benchmarks have been further evaluated in [79] on large scale SMP clusters in order to investigate the causes of performance variability. Maury et al. [19] evaluated a CMP/SMT hybrid using OpenMP on a 4-way SMT system and a simulated CMP and found the memory subsystem to be the primary performance bottleneck.

Several parallel programs such as HMMER have been evaluated in [84] where they evaluate the performance of bio-informatics programs. HMMER was also evaluated by Purkayastha et al. [73] on 4, 8 and 16-way Intel Xeon SMP nodes using OpenMP.

2.4 Scheduling and Operating System Noise

Job scheduling is a critical component in modern operating systems. An operating system's scheduler has the task of deciding which threads are assigned time on the CPU(s) of a system. Time periods are usually of fixed length, unless the process finishes before the end of its time slot. The Linux scheduler works by priority scheduling, assigning each thread a priority. The priorities of the threads are modified after each time slot, with threads that have not been executed rising in priority and threads that have just completed their time slot being of lower priority. The Linux scheduler in the 2.4.x kernels used a single process queue which held information about all of the active tasks and their priority. Each time a new task was to be assigned, the queue needed to be traversed. The scheduler in the 2.5.x and higher kernels was upgraded to the O(1) scheduler to avoid having to traverse the queue after each time interval. The new scheduler functions by having many queues, one for each priority. This avoids the need to traverse all of the active processes each time a scheduling decision must be made, only the highest priority queues need to be searched.

Operating system noise is a term describing the overhead caused by running the requisite daemons and services required by the operating system. This overhead can be a significant factor in determining the performance of high-intensity workloads. The actual time devoted to running operating system tasks is typically negligible but the swapping of such small jobs into a processor group that is running a full workload can cause a job imbalance which can degrade system performance by delaying some execution threads. This causes delays to all of the other related program threads when thread synchronization is necessary. This noise is most significant when the system is running a multi-threaded process that uses all of the processors in a shared memory system.

2.4.1 Prior Art

The work in [62, 94] has been the only work on devising algorithms for OpenMP loop scheduling [94], and thread pairing [62]. In [94], the authors focused on altering 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 [62] 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 a 7% to 28% improvement over the default Linux scheduler. Others have attempted to develop scheduling methods for improved performance on SMP machines [72] as well. Methods for scheduling on SMT systems [81] have also been introduced that attempt to take advantage of the symbiotic nature of some processes when running on an SMT system. Work by Nikolopoulos et al. [4, 68] has examined scheduling algorithms that can be used to improve performance by adaptively scheduling processes while taking into account bus bandwidth and memory pressure issues.

Previous research has shown that system noise may have a dramatic effect on high-performance computing systems [45, 71]. In [71], Petrini and his colleagues noticed that their application has superior performance when using three processors per node instead of the full four. Using a number of methodologies, they discovered that this is due to neither the MPI implementation nor the network, but the system noise including OS

daemons, kernel threads, and OS real-time interrupts, among other things. The effect of system noise on the performance of applications has also been verified in [45, 89].

2.5 System Power Consumption

The process of power modeling is a very in-depth process that can be accomplished using a wide range of techniques. Primarily, power modeling is done using software simulation based on the design files used to design the logic circuits in question, with programs such as HSPICE [87]. The estimation of the power consumption of a given system can be roughly determined using guidelines that have been found experimentally by measuring the current consumption of the system under various workloads. A common power simulation approach is the use of popular power simulation programs based on the Alpha family of processors such as Wattch [10]. Simulators such as Wattch use predetermined power consumption values to estimate the approximate energy consumption of each individual instruction. The power simulation program then interfaces with a simulator, like SimpleScalar [7]. The major problem with this approach is that it does not easily translate over to current real commercial systems. Power modeling of existing commercial systems whose design is not available to the public presents a much more difficult problem, as the models developed for such systems do not have enough information about the implementation of the device being modeled to make the models extremely accurate. The power simulation programs available rely on in depth circuit information that was available for processors such as the Alpha. Such detailed information is not available from Intel, which provided the CPUs used in this investigation. However, measurements from Intel Corp. for their CPUs [3] provide the basic metrics of power consumption that are used in some of the experiments detailed in this thesis.

This thesis avoids many of the problems associated with modeling power consumption by measuring the power consumption of real systems. This provides the most accurate method of studying power consumption as the power consumption measurements are within ±3% of the actual system power consumption. The power measurements were obtained using a digital multi-meter equipped with a data recorder. The measurements were taken from the power supply side of the machines, which means that the

measurements taken represent the real power consumption of the system including all of the power loss associated with inefficiencies in the power supply.

2.5.1 Power Metrics

The following metrics are used to study the power-performance characteristics of systems.

Performance: High-performance computing has always been concerned with performance. Performance of an application running on a system is given by wall-clock execution time, *D*.

Power: Theory [66] tells us that the power consumed by a CMOS processor, in watts, is equal to the activity factor of the system (percentage of gates that switch for each cycle) multiplied by the capacitance of the CPU times the voltage squared times the frequency. This is shown in Equation 2.1.

$$P = \alpha C V^2 f \qquad (2.1)$$

Note we have ignored the power expended due to short-circuit current, and the power loss from leakage current, as the dynamic power consumption, αCV^2f dominates in CMOS circuits.

Frequency is directly proportional to the supply voltage. Therefore, power is proportional to the cube of a changing frequency. However, historical data [3] suggests that power on modern processors is proportional to the square of the duty cycle.

Energy: Power is the consumption at a discrete point in time. Energy is the cost during the execution time, *D*, and is shown as:

$$E = \int_0^D P dt = P_{avg} \times D \qquad (2.2)$$

Power-performance efficiency: This metric allows choosing the operating point at which maximum energy saving can be achieved with acceptable performance

degradation. The energy-delay product is used to quantify the power-performance efficiency, as shown in Equation 2.3:

$$E.D = E \times D \qquad (2.3)$$

2.5.2 Prior Art

Several architectural proposals have been put forward in an attempt to reduce the power consumption of processors including proposals to enable the decay of information in processor caches [37], and a very interesting proposal to allow processor operation at very low voltages and corresponding error correction [25]. As this thesis is not concerned with making architectural changes to existing CPU designs, no more details relating to such proposals will be provided. Instead we will concentrate on methods of reducing power consumption that are done using software.

There have been quite a number of micro-architectural studies on the subject of energy reduction of modern day processors. These include dynamically tuning processor resources with adaptive processing [2], comparison of SMT and chip multiprocessing [56], and heterogeneous multi-core architectures [52, 53]. Most of this research has been done with single-threaded applications and through simulations, or analytical methods. This thesis concentrates on multi-threaded workloads on real systems.

There have been attempts to reduce overall power usage through the use of compiler optimizations [47] and also by controlling power management systems through compiler flags and simulation [92]. Recent attempts have used real-time information from hardware performance counters to attempt to schedule threads for performance and power consumption [18] on SMT and CMP systems. Additional work has focused on previous run profiling for creating databases that can suggest configurations for the system when running certain programs [60].

Dynamic voltage and frequency scaling is known as one of the most effective methods to reduce CPU power consumption, unfortunately at the expense of performance degradation. In fact, the semiconductor industry has recently introduced many energy saving technologies into their chips. The most successful of these have been *SpeedStep* technology from Intel as well as *PowerNow!* and *Cool'n'Quiet* from AMD. Several papers have been published on research utilizing such features to reduce power and

energy consumption, from devising a compiler algorithm for optimizing single-threaded programs for energy usage on laptops [36] to power and energy management techniques for servers and data centers [51]. Power consumption has also been studied for high-performance computing workloads in SMP and AMP servers [3, 8] and in high-performance clusters [29, 30].

The authors in [3, 8] describe the process of creating an AMP node from a commercial Intel SMP server. An AMP is a system that contains multiple different processors. The processors can be different architecures running at the same or different speeds, or be the same architecture but running at different speeds. In [3], Annavaram et al. analyze the *energy per instruction* (EPI) gains that can be obtained from using CPUs operating at different frequencies. In fact, they determined that by utilizing a setup that consists of one fast processor to run sequential code, assisted by three slower CPUs to run parallel code, one could reduce the overall EPI of a system while maintaining a higher speed than a normal SMP using the fixed power budget of one 2.0GHz Intel Xeon processor. They used the SPEComp benchmarks as well as several other applications in their study; however, their work did not address HT-enabled systems [3].

In [8], Balakrishnan et al. investigated the impact of performance asymmetry of different AMPs on commercial applications as well as SPEComp applications. For commercial applications, they observed significant performance instability. They were able to eliminate this for some applications by devising a new kernel scheduler ensuring faster cores never go idle before slower ones. For SPEComp scientific applications with tight coupling among different threads, they found stability, but with poor scalability as the slowest core forces faster ones to idle. To eliminate performance asymmetry, they changed the static OpenMP loop scheduling used to dynamic scheduling. However, this resulted in degraded performance. This work also did not address HT-enabled systems. Although, the work in [8] is only focused on performance asymmetry, they indicate AMP systems can be effective for power/performance efficiency.

2.6 Summary

Although the area of shared memory multi-processors is a mature field and the research available for such systems is comprehensive, the study of modern hybrid SMT

and CMP-based SMPs is still evolving. This thesis explores the characteristics of modern single and multi-core processors using real machines. The majority of previous work on similar topics has been performed using simulations and non-SMT architectures. The study of AMPs is still relatively new, and previous work has focused on fixed power budgets and performance stability. This thesis expands on this breadth of research by investigating the power and performance benefits that can be realized using an AMP architecture. The following chapters investigate the overhead caused by OpenMP constructs and then profiles single and multi-core processors running multi-threaded scientific benchmarks. The knowledge gained from the profiling of real system is then used to implement a new operating system scheduler that increases system performance while increasing energy efficiency.

Chapter 3: Application Specifications

This chapter details the benchmarking applications that are used in this thesis. The NAS OpenMP [44] and SPEC OpenMP [82] benchmarks are considered industry standards that are updated on a regular basis. The NAS benchmark suite is produced by the United States National Aeronautics and Space Administration (NASA). The NAS benchmark suite is available as an open source suite of applications coded in high performance FORTRAN. The SPEC OpenMP benchmarks were developed by the Standard Performance Evaluation Corporation in 2001. They are available as a commercial package and are coded using FORTRAN and C.

3.1 NAS Parallel Benchmarks

The NAS parallel benchmarks [44] are published by NASA and are based upon the Computational Fluid Dynamics (CFD) calculations that are used in scientific aerospace computations. The benchmarks consist of two types of application, 5 kernel applications replicate the behaviour of the main computational loops of CFD applications and 3 simulated computational fluid dynamics applications that replicate the data manipulation activity of real CFD applications. This thesis studies the behaviour of all 3 of the simulated CFD applications as well as 2 of the kernel applications. Each application is discussed in detail in this chapter, and although this information is not strictly necessary for understanding this thesis, it is included for the interested reader who wishes to have more insight as to the nature of the benchmarks utilized here.

3.1.1 NAS Simulated CFD Applications

3.1.1.1 BT

The BT benchmark is an application that solves a 3-D matrix of Navier-stokes equations [44]. The Navier-Stokes equations used are not in a fully compressed form. The benchmark uses an Alternating Direction Implicit (ADI) approximation factorization that produces equations that are decoupled from the x, y, and z directions which can then be solved directly [44]. The majority of the benchmark scales well, but some K

dimensional operations involve large memory strides, which makes this benchmark less scalable due to poor cache performance.

3.1.1.2 LU

The LU benchmark solves Navier-Stokes equations in a 3-D system using Symmetric Successive Over-Relaxation, by splitting the system into block lower and upper triangular systems [44]. This benchmark uses a pipelined computation where the first processor in the system provides information necessary for the execution of the thread run by the second processor and so on. This makes this benchmark especially susceptible to synchronization issues. The introduction of delay from one of the processors can create a ripple effect amongst the pipeline which significantly affects the performance of the benchmark. Consequently this benchmark is best suited to execution on systems with a small amount of additional system overhead.

3.1.1.3 SP

The SP benchmark uses a factorization to decouple the x, y and z directions and directly solves the resulting decoupled linear equations [44]. This benchmark is somewhat scalable. Its scalability is limited primarily due to the same cause as the scalability of BT, that memory operations in the K direction use large strides and cause a significant number of cache misses. Therefore, this benchmark is dependent on memory subsystem performance for good performance.

3.1.2 NAS Kernel Applications

3.1.2.1 MG

MG is a multi-grid solver for processing scalar Poisson equations. Because of the nature of this multi-grid solver it is an excellent intensive memory test as it requires a significant amount of data movement [44]. The benchmark projects a fine-grid onto a coarse grid, solves the coarse grid and projects the result onto a fine grid, then performs a smoothing operation. This process is repeated several times. Due to the size of the grids, this benchmark is affected by memory subsystem speeds.

3.1.2.2 CG

CG stands for conjugate gradient, which this test uses on a sparsely populated unstructured matrix that is meant to test the system's ability to perform operations on such matrices [44]. This is of course memory fetch intensive as the locations of data are distributed randomly, therefore reducing the effectiveness of pre-fetching. This benchmark scales well for small numbers of processors but begins to experience poor scaling between 8 and 16 processors.

3.1.3 SPEC OpenMP Benchmark Suite

The Standard Performance Evaluation Corporation (SPEC) [82] CPU2000 benchmark suite was adapted to run under OpenMP in 2001. Since that time some modifications have been made to the suite. The version of the suite used for testing in this publication was version 3.0, with the Purdue University source code fix for the art benchmark. This version is almost identical to the only recently released version 3.1 (the Purdue University modifications were officially included with version 3.1). Eleven benchmarks are provided with the suite, and six of those benchmarks are used in this paper, comprising five high performance FORTRAN benchmarks and one C benchmark. The benchmarks simulate the run-time behaviour of real scientific applications.

The *apsi* benchmark is a FORTRAN benchmark simulating an air pollution modeler [6]. It uses primarily floating point arithmetic. It models the effects of temperature, pressure and wind on pollution particles and tracks the predicted diffusion of pollutants through the atmosphere. The *art* benchmark is a C benchmark simulating image recognition and neural networks tasks [6]. An Adaptive Resonance Theory neural network is trained to recognize two objects, an airplane and a helicopter and then is used to find these objects in an image. The *fma3d* benchmark is a FORTRAN benchmark based on a crash modeling simulator [6]. The benchmark is floating point operation intensive as it uses finite element analysis to model the effect of sudden impacts on 3-D solids. The input files used model an explosive element of a cylinder. The *mgrid* benchmark is a FORTRAN benchmark that is a multi-grid solver [6]. This benchmark is a standard multi-grid solver, a technique that is used in a variety of areas. This

benchmark uses a constant coefficient equation on a cubical grid. The nature of multi-grid solvers makes them memory intensive as a large amount of data is moved during the computational process.

The *swim* benchmark is a FORTRAN benchmark for modeling shallow water [6]. It calculates the velocity vectors of a shallow water data representation consisting of a 1335x1335 matrix over a short time span. It uses a significant amount of double floating point arithmetic. Finally, the *wupwise* benchmark is a FORTRAN benchmark modeling quantum chromodynamics [6]. Quantum chromodynamics is the study of sub-atomic particles like quarks and gluons, and the interactions between them. Wupwise stands for the Wuppertal Wilson Fermion Solver, which is used to calculate the interactions between quarks in the area of fundamental physics. This is a very computationally intensive process.

Chapter 4: Workload Characteristics of SMT-Capable SMPs

In this chapter we examine the SMT implementation from Intel called Hyper-Threading. We examine the performance of HT when used in multi-threaded shared memory applications. 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.

This chapter first examines the performance of different OpenMP constructs on single-core dual CPU HT-based Intel Xeon servers running the RedHat's Linux kernels 2.4.22 and 2.6.9. It is found that the overhead of OpenMP constructs with HT enabled is significantly larger than when HT is off [33].

This chapter also presents an evaluation of scientific applications in the NAS OpenMP suite (version 3.2) [44], and SPEC OMPM2001 suite (version 3) [82] with kernel 2.6.9. It is interesting to discover the impact of SMT-based SMP systems on the performance of such applications. The effect of resource sharing within the processors on the overall system performance is evaluated by collecting data from the hardware performance counters. In addition, the architectural limitations of such a system are pinpointed by observing its cache performance [33].

This chapter is organized as follows. The experimental setup is described in section 4.1. In section 4.2, the performance results are analyzed. A summary of the findings is presented in section 4.3.

4.1 Experimental Setup

The experiments in this chapter were conducted on a single-core platform, a Dell PowerEdge 2850 server. The PowerEdge 2850 server from Dell has two single-core 2.8 GHz Intel Xeon EM64T processors, a 16KB shared execution trace cache, a 16KB L1 shared data cache, a 1MB shared and unified L2 cache, and 2GB of DDR2-SDRAM on a 800 MHz front side bus. The operating system is RedHat's Enterprise WS 4.1 Linux kernel version 2.6.9 and kernel 2.4.22. 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 equal to 3856 MB/s and 1855 MB/s, respectively.

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. The new scheduler 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 active tasks. This is accomplished by using multiple priority queues and a scheduling algorithm that is priority aware such that it can quickly decide between processes based upon which priority queue they reside in. The previous version of the scheduler was also priority aware, but used a unified process queue which necessitated stepping through the entire queue for each scheduling period in order to make its next scheduling decision.

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.

4.1.1 Terminology

Figure 4.1 and Table 4.1 describe the naming convention for specifying the configuration of the two machines. Figure 4.1 shows the naming of each of the processors in an SMT-capable SMP system for up to 2 processors.

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 (either 1 or 2).

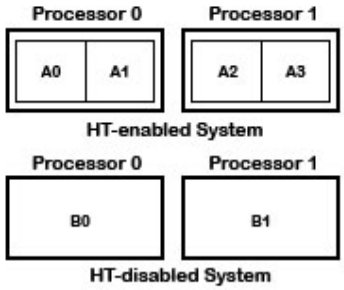


Figure 4.1: Processor numbering for a SMT system with a maximum of 2 processors

Table 4.1: Configuration Information

Terminology	Hardware Contexts	Corresponding Architecture
Serial	B0	Serial
HT_{on}-2-1	A0, A1	SMT
HT_{off}-2-2	B0, B1	2-way SMP
HT_{on}-4-2	A0, A1, A2, A3	SMT-capable 2- way SMP

4.1.2 Application Characterization

Application characteristics were gathered at run-time using the hardware performance counters available on the Intel Xeon processors. We collected data using the Intel VTune Performance Analyzer version 7.2 [41]. The Intel Fortran and C/C++ compilers (version 8.1) were used to build the benchmark applications.

Collecting profiles of real systems can be a very time consuming process. The number of performance counters available for use is extremely limited requiring multiple runs of a single application to collect all of the required profiling information. This is further exasperated by using large data set benchmarks which require a significant amount of time to finish execution. In order to collect the most comprehensive profile possible, the data collected must be collected for the entire runtime of the application, and all of the application runs must be confirmed by running the same tests multiple times. For example, in order to collect 13 performance metrics for a system configuration using the SPEC benchmark suite used in this thesis one has to run at least 26 iterations of the benchmark, 13 for collecting the data and at least 13 to confirm the data, not including runs which are not properly confirmed and must be thrown away. With each iteration of the SPEC benchmarks taking many hours, a single configuration profile can easily take an entire month of processing time to complete successfully.

4.2 Experimental Results and Analysis

This section describes the results and analysis of our experiments on the PowerEdge 2850 server. 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_{on}-4-2 and HT_{on}-2-1 cases. Results are shown for kernel 2.6.9, unless otherwise noted.

4.2.1 EPCC

Overhead due to synchronization and loop scheduling is an important factor in the performance of shared-memory parallel programs written in OpenMP. The EPCC OpenMP microbenchmarks (version 2.0) [74] were used to measure the overhead of synchronization and loop scheduling calls 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.

4.2.1.1 OpenMP Synchronization

The EPCC synchronization benchmark was used to find the overhead of OpenMP directives with varying a number of threads, with and without Hyper-Threading. Figure 4.2 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 enabled is significantly greater than the same number of threads on an HT-disabled system.

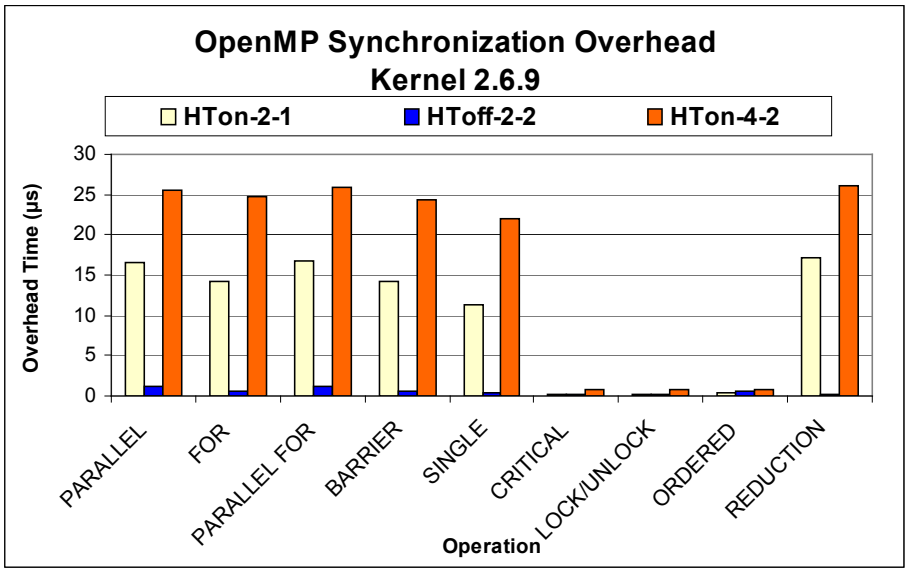


Figure 4.2: Overhead of OpenMP synchronization.

Figure 4.3 compares 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 HT-disabled case (not shown here). However, the Linux kernel 2.4.22 was found to be superior when using HT as shown in Figure 4.3. The difference between the 2.4.22 kernel overhead and the 2.6.9 kernel overhead is most likely the result of the extra processing required to assign the newly created threads to their respective priority queues in the 2.6.9 kernel. The 2.4.22 kernel does not have this overhead as all processes are simply inserted into a single process queue.

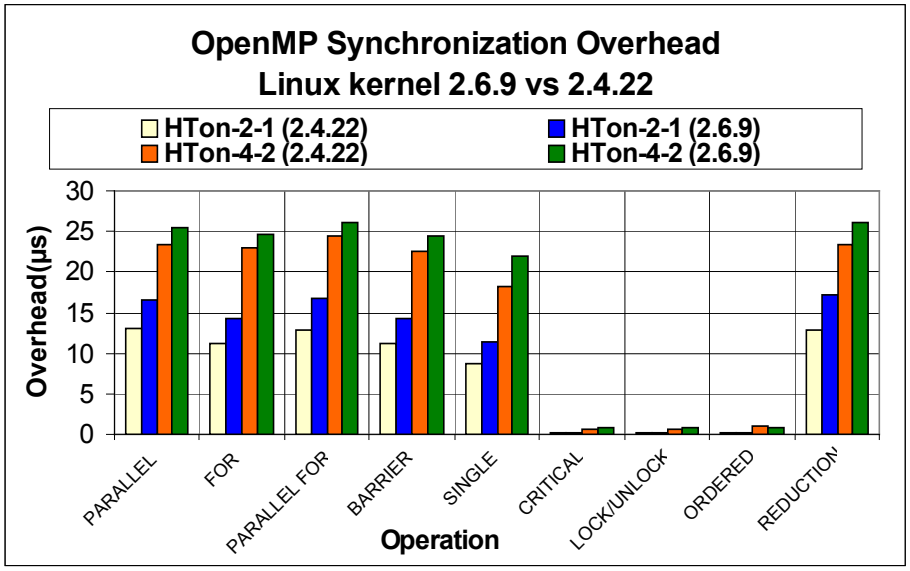


Figure 4.3: Kernel 2.6.9 vs. 2.4.22 impact on OpenMP synchronization overhead.

4.2.1.2 OpenMP Scheduling

OpenMP provides three options for scheduling loop iterations among threads: *STATIC*, *DYNAMIC*, and *GUIDED*. Figure 4.4 shows the loop scheduling overheads for HT_{off}-4-4 and HT_{on}-8-4. As can be seen, the overhead of different OpenMP loop scheduling schemes is 2.5 to 8 times larger for Hyper-Threading than for an HT-disabled system.

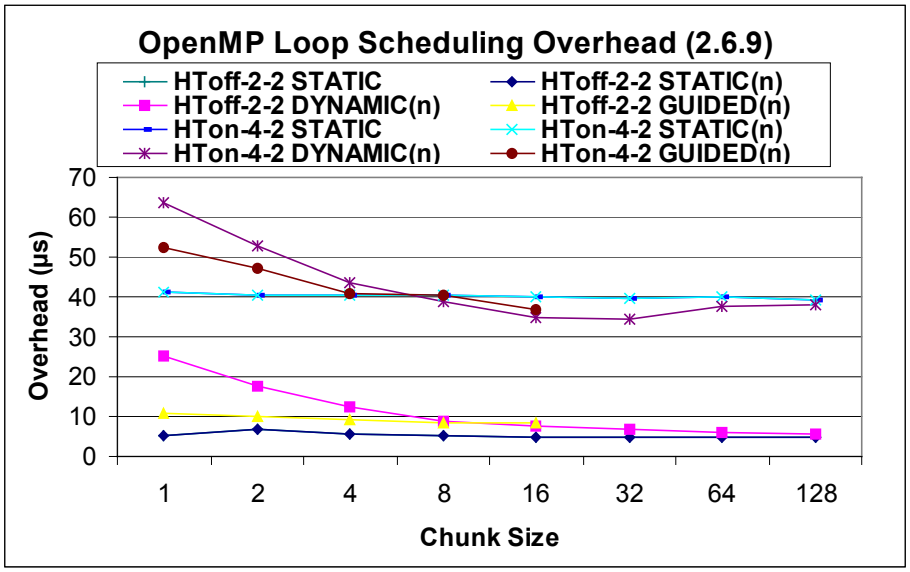


Figure 4.4: Overhead of OpenMP loop scheduling policies.

The *STATIC* scheduling is approximately as fast as *DYNAMIC* scheduling for larger groups, but of course does not have the benefit of dynamic load balancing. It is clear that if the workload is large enough that large chunks are possible, *DYNAMIC* scheduling with its low overhead but load balancing benefits is clearly the best choice. However, if the workload is dynamic and unbalanced as well as large, then *GUIDED* scheduling would be an excellent choice. 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 HT-enabled and HT-disabled cases, the overhead of *GUIDED(n)* is close to that of *DYNAMIC(n)*. Figure 4.5 compares the impact of two kernels on synchronization directives. Except for the some *Dynamic* strategies and one occurrence of the *Guided* policy, kernel 2.4.22 is better than the 2.6.9 kernel.

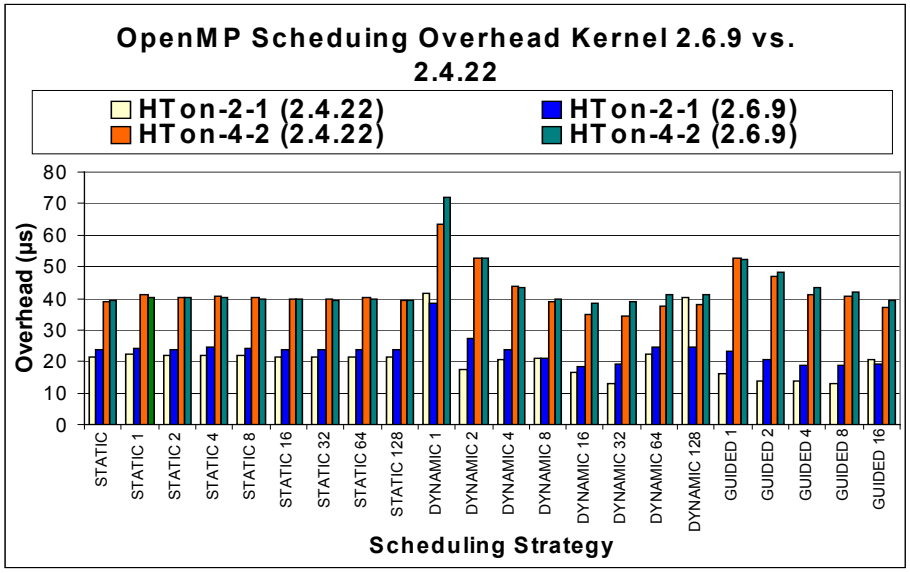


Figure 4.5: Kernel 2.6.9 vs. 2.4.22 impact on OpenMP loop scheduling policies.

4.2.2 Application Performance and Analysis

When determining the performance of an application, execution time is the most practical performance metric. In this section, we present the performance of NAS and SPEC applications on our PowerEdge 2850 platform under the 2.6.9 kernel. Figure 4.6 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-enabled. The second pair is for two physical processors.

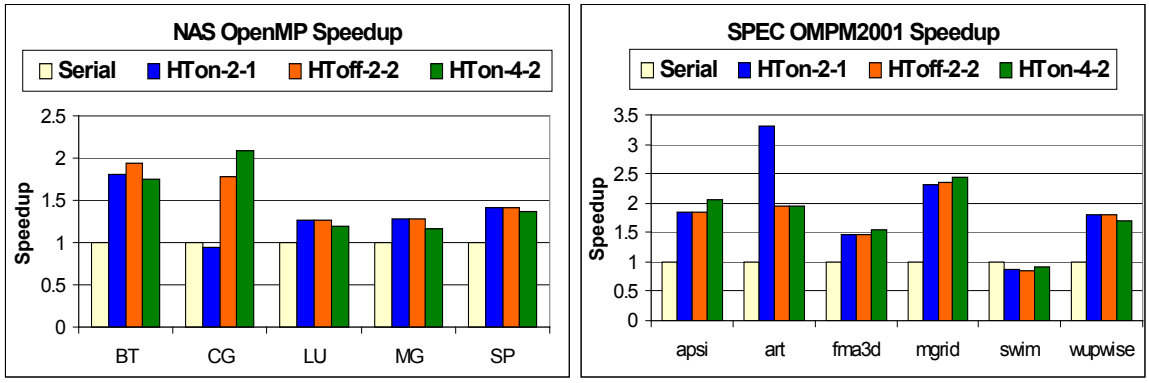


Figure 4.6: Speedup for NAS OpenMP and SPEC OMPM2001 applications under Kernel 2.6.9 relative to the serial case.

The goal of these tests 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, and wupwise benefit from having two threads in one-processor execution, they suffer somewhat in the two-processor execution. 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_{on}-2-1 not only outperforms the serial case, but also outperforms HT_{off}-2-2. 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. This lack of scalability has traditionally been the case for the OpenMP version of swim on platforms similar to the ones used here.

BT, CG, LU, MG, and SP attain an average speedup of 35%, 6%, 10%, 9%, and 19%, respectively, when HT is enabled. For the SPEC suite of applications, wupwise, mgrid, 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.

Table 4.2 summarizes the average performance gain when logical processors are enabled. It is evident that the HT implementation of SMT cannot provide a performance gain for 2-processor executions, at least for our multi-threaded applications.

Table 4.2: Average speedup gained by enabling HT for NAS and SPEC Parallel benchmarks.

Physical Processors	NAS OMP	SPEC OMP	Overall
1	34%	92%	63%
2	-2.2%	3.3%	0.55%

4.2.2.1 Trace Cache Analysis

In this section, we investigate the reasons behind the possible flaws of Hyper-Threading. 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 4.7 presents trace cache misses, while Figure 4.8 depicts trace cache delivery rate for these applications.

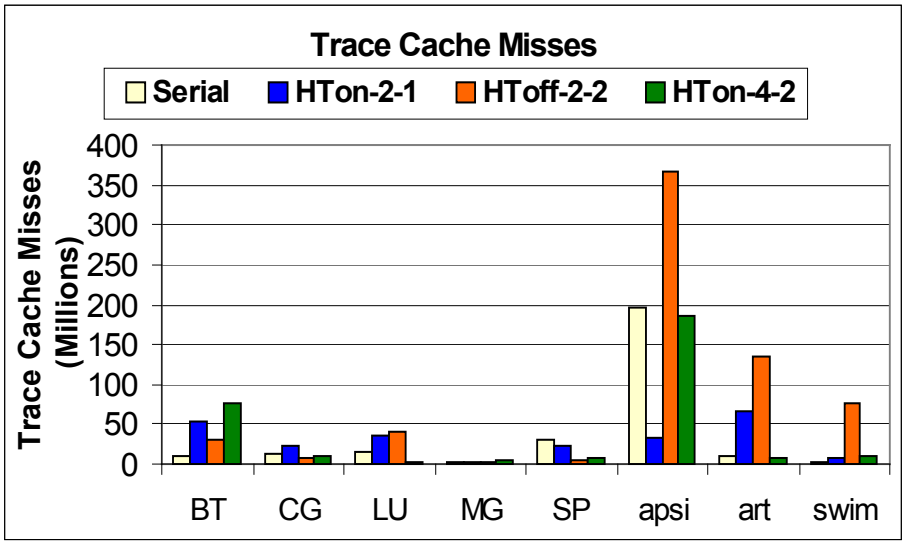


Figure 4.7: Trace cache misses

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, HT provides improvements over the single processor system.

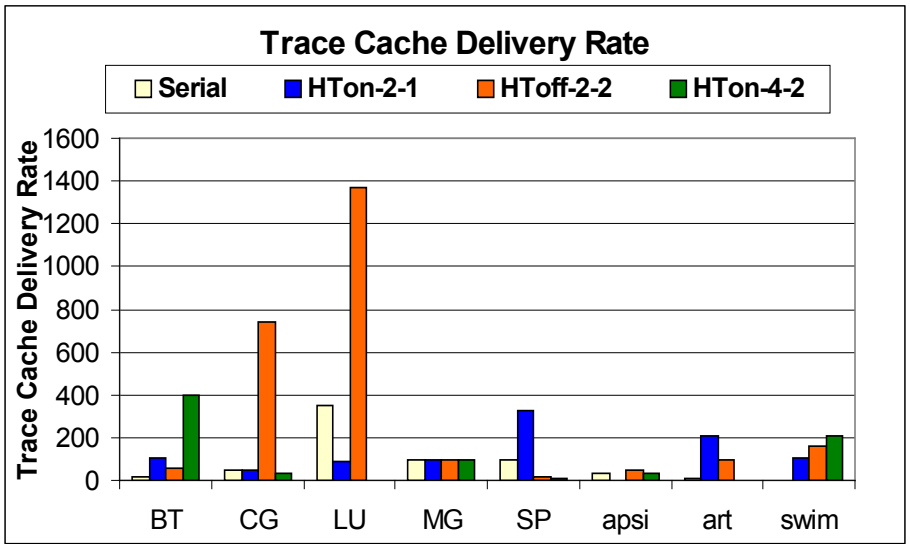


Figure 4.8: Trace cache delivery rate (trace cache fetches per 100 clock cycles)

It can be observed in Figure 4.7 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 4.6 to Figure 4.7 and Figure 4.8. For instance, when performance is significantly worse for the CG on HT_{on}-2-1 compared to the HT_{off}-2-2 and HT_{on}-4-2 cases, one can observe a marked increase in the trace cache misses from the HT_{on}-2-1 system. This also occurs for the SP benchmark, but it can be observed that SP's 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_{on}-4-2 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_{on}-2-1 case. For the case of diminishing performance at the HT_{on}-4-2 level, one can observe that the main performance bottleneck is the cache memory bandwidth. In Figure 4.9, the increased number of 2nd level cache

misses over the serial case corresponds to the decreased performance of the swim benchmark illustrated in Figure 4.6.

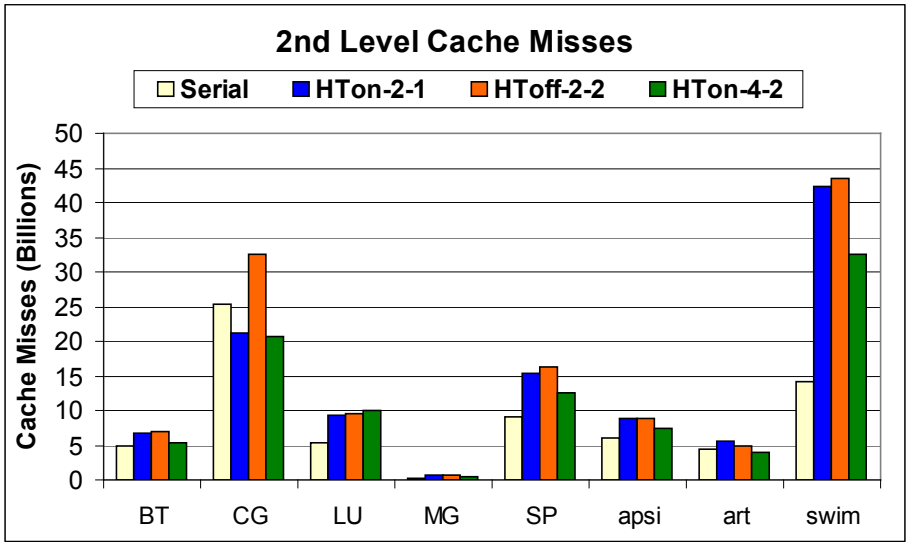


Figure 4.9: 2nd Level Cache Misses.

4.3 Summary

In this chapter the overhead due to OpenMP constructs has been examined, comparing the 2.4.22 Linux scheduler with the O(1) 2.6.9 Linux scheduler. It was found that overall the STATIC loop scheduling was a good choice for small chunk size applications while DYANMIC and GUIDED are the best choices as chunk size increases. It was also found that the 2.4.22 scheduler exhibited slightly less OpenMP overhead than the 2.6.9 scheduler. In addition, the implementation of SMT from Intel, Hyper-Threading was analyzed to determine its potential benefits in increasing system performance. It was determined that HT can only be of benefit for a small number of applications for a 2-way system, although it can be of benefit in a single processor system. The cause of this lack of speedup when enabling HT was identified as being a memory system limitation. The increased memory pressure caused by doubling the number of running threads (by turning on HT) causes an overall system slowdown that cannot be offset by the increase in the number of simultaneously executing threads.

Liao et al. [57] have previously studied OpenMP overhead and we find that our results are consistent with their findings on a Dell 450 Precision workstation. Their NAS and SPEComp benchmark results are fairly consistent with our findings. However, their

testing methodology involved using two physical processors running 2 threads for HT_{on}-2-1 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.

With the information that was gathered in this chapter, we can now focus on the most relevant metrics for determining the optimal configuration of modern multi-processor systems. The work illustrated here on single-core processors can be extended to multi-core processors within similar environments. Therefore, the prime metrics discussed in this chapter such as memory bandwidth and cache miss rates can be compared to newly emerging commercial multi-core processor systems. In the next chapter we examine multi-core systems in a similar yet more in depth method to determine if the traditional performance metrics are suitable for judging the performance of such new systems.

Chapter 5: Characterization and Analysis of Multi-Core SMPs

The placement of multiple cores on a single processor die is an emerging technology in today's commercial market. As the design of new processors with consistently higher clock speeds has slowed due to cooling concerns, the attention of industry has shifted from faster individual processors to groups of processors in order to increase the overall performance of CPUs. The emergence of chip multi-processing and the hybridization of this technology with SMT and SMP technology creates an interesting platform from which we can determine the optimal operating environment for these emerging technologies.

This chapter investigates dual-core two-way systems available with Intel's Hyper-Threading technology and examines the reasons behind the performance of such systems under various configurations and workloads [31].

The first generation of Intel Xeon CMT processors 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 multi-threaded SMPs, where a number of CMT processors are used in a conventional *n*-way SMP configuration. This can create further bottlenecks as each individual physical chip shares a memory controller between its two cores, and therefore the two physical chips require a method of ensuring memory consistency and bus arbitration while taking into account that there are two channels to main memory. As a result, the competition for shared resources is intense.

CMT-based SMPs present new challenges as well as new opportunities to maximize performance. Given a set of applications and a number of CMT processors there are ample opportunities to control how many threads, and which threads, to co-schedule on different cores or contexts in order to achieve performance. It is our intention in this chapter to identify the shared resources that might become a bottleneck to performance under different configurations.

Traditionally, OpenMP applications have been developed for flat SMP systems. With the availability of modern CMT processors, hybrid SMP configurations may perform

differently due to workload characteristics of applications. They also present new challenges to the systems software to accommodate inter- and intra- parallelism among threads. Greater demand on cache subsystem, increased bus transactions, and stall cycles may significantly affect the performance of OpenMP applications [69].

The first contribution of this chapter is the performance evaluation of scientific applications in the NAS OpenMP suite [31] in section 5.2. It is interesting to discover the impact of multi-core 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 attempt to pinpoint architectural limitations of such a system by observing its overall cache performance, *cycles per instruction* (CPI), branch prediction rates, bus transactions, ITLB and DTLB hit rates and number of stalled cycles. In section 5.3, an investigation is performed into the performance of the different configurations using a variety of multi-application workloads. The performance of the systems while overloaded with twice as many execution threads for a given benchmark as are available in hardware is then investigated in section 5.4. Finally, an investigation into the effects of operating system noise is detailed in section 5.5

5.1 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 with 12KB shared execution trace cache, and 16KB L1 shared data cache on each core. A 2x2MB L2 cache is allocated with one 2MB L2 cache for each core on a chip. There are 4GB of DDR-2 SDRAM on an 800 MHz Front Side Bus. The chip's core is the Paxville core, whose basic architecture is described in Figure 5.1. The operating system is Red Hat Linux Enterprise WS 4.1 with Kernel 2.6.9-11. Using LMbench [63] we have measured the L1, L2, and main memory latencies of the processor as 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 using the LMbench benchmark 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. The main memory read and write bandwidths when using two physical chips are 5.43 GB/s and 1.61 GB/s respectively. This emphasizes the utilization of the secondary memory access controller in the second chip as well as the dual channel memory architecture available. The decreased write bandwidth is most likely due to consistency requirements with respect to the two memory access controllers.

To test the system in a variety of configurations, some of the tests were run by masking off some of the available processors in the system such that they could not be used by our application threads. This enabled us to test the performance of the system using different thread distributions between the existing resources. The default Linux scheduler was used for assigning the individual threads amongst the specified processors.

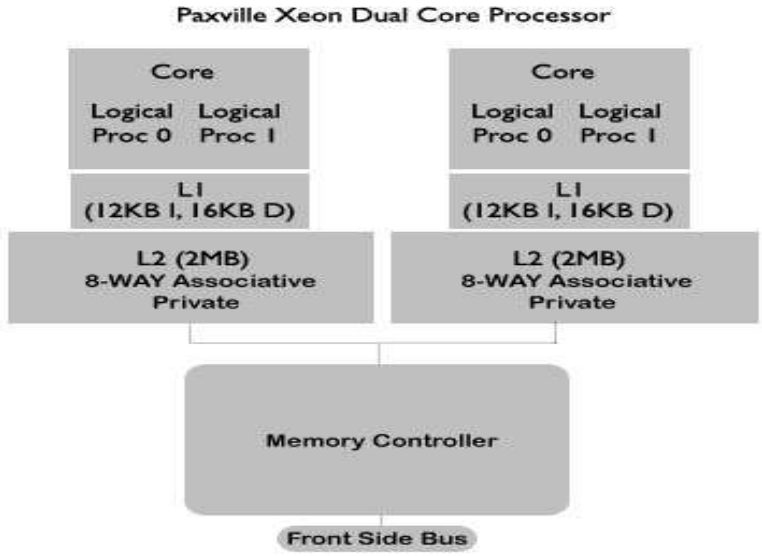


Figure 5.1: Intel Xeon Dual-Core Chip Layout

5.1.1 Terminology

The test results detailed in the next section show the results for the testing on a variety of different configurations. Our intention is to find the best set of architectures for each application under study running with a given number of threads. Figure 5.2 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.

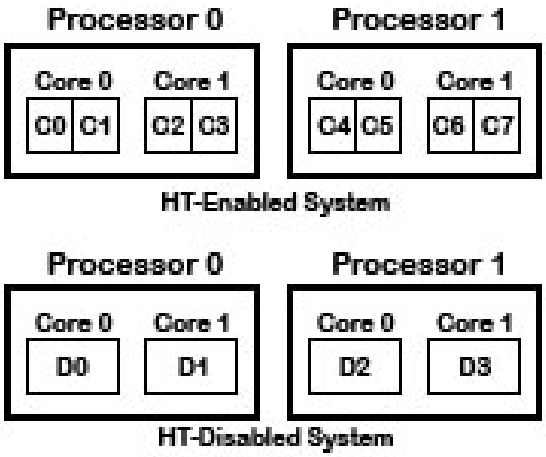


Figure 5.2: Processor numbering for 2-way Dual-core System

Table 5.1 shows the different configurations used in our study. The basic terminology used to describe these configurations is comprised of three parts (four parts for the overloaded cases). 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. The first three terms are identical to the system terminology in Chapter 4. For the overloaded tests, the fourth item is the number of hardware contexts available for use. There is no need for a fourth item for non-overloaded cases as there are as many hardware contexts available as the number of application threads.

Table 5.1: Configuration Information

Terminology	***Hardware Contexts***	***Corresponding Architecture***
Serial	D0	Serial Uni-processor
HT_{on} -2-1	C0, C1	SMT
HT_{off} -2-1	D0, D1	CMP
HT_{on} -4-1	C0, C1, C2, C3	CMT
HT_{off} -2-2	D0, D2	SMP
HT_{on} -4-2	C0, C1, C4, C5	SMT-based SMP
HT_{off} -4-2	D0, D1, D2, D3	CMP-based SMP
HT_{on} -8-2	C0, C1, C2, C3, C4, C5, C6, C7, C8	CMT-based SMP
HT_{on} -4-1-2	C0, C1	Overloaded SMT
HT_{off} -4-1-2	D0, D1	Overloaded CMP
HT_{off} -4-2-2	D0, D2	Overloaded SMP
HT_{on} -8-1-4	C0, C1, C2, C3	Overloaded CMT

5.2 Single Application Results

In order to present these results in a fair manner, the configurations must be divided into several groups, such that the configurations can be compared to configurations with similar amounts of resources. The HT_{on}-2-1 configuration must be examined separately from the rest of the data as it has by far the smallest amount of resources available to it. In this configuration it shares a trace cache between executing threads along with several functional units on the CPU, in addition to having only one 2MB L2 cache available for use.

The second group is much larger in that 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. This group utilizes all of the resources available in our platform with HT on and off. Sometimes, 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.

5.2.1 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 and L2 cache miss rates are presented in Figure 5.3(a) and 5.3(b). The first observation that can be made by examining the graphs, is that 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 excellent cache performance can be attributed to several factors, one being that there is relatively little contention for the cache resources, and the limited computational resources available mean that the memory bus is free for pre-fetching operations during execution.

The second group is similar to the third group in term of the 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 have fairly comparable memory performances, with the HT_{on}-8-2 having the advantage in terms of its trace cache performance for the CG benchmark, while the HT_{off}-4-2 configuration has an advantage for the L2 miss rate on the LU benchmark.

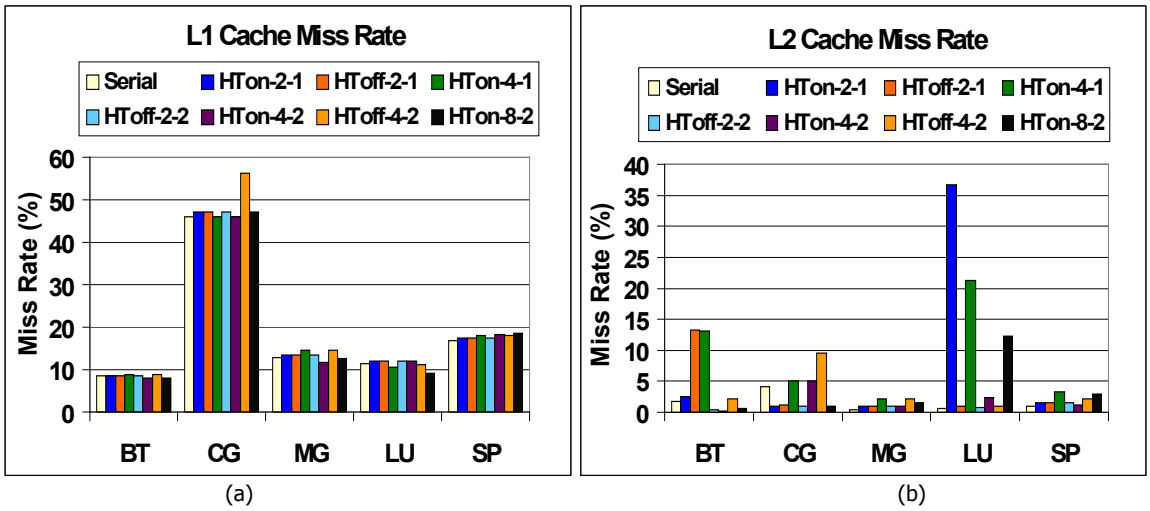


Figure 5.3: (a) L1 Cache Miss Rate and (b) L2 Cache Miss Rate

The effect of enabling HT on the different architectures leads to a 2.8% increase in L1 miss rate for group 1, a 0.01% increase on group 2, and a 3.4% and 11.0% decrease for groups 3 and 4, respectively. The average first level cache miss rate is 20.1% across all applications for all HT_{off} configurations, while the average first level cache miss rate for HT_{on} configurations is 17.9%. The CG benchmark has a small perturbation, most likely caused by interference with the executing computational loops.

The second level cache miss rates fluctuate between the different applications, but the changes in the LU benchmark are quite significant; therefore, the results detailed here neglect the LU miss rates. The effect of enabling HT on the different architectures leads to a 105.5% increase for group 1, a 55.9% increase on group 2, a 150.5% increase for group 3 and a 37.2% decrease for group 4. The average second level cache miss rate is 2.35% across all applications for all HT_{off} configurations, while the average second level cache miss rate for HT_{on} configurations is 5.35%.

The LU benchmark spans a large memory area throughout its execution and its memory access patterns require that previous computational results be present in memory for use in future computation, leading to a disadvantage for computationally limited configurations, as they cannot perform in large parallel batches, and old results are evicted from the cache before they are needed again. This leads to a much larger increase in L2 cache miss rate when enabling HT, with a 7260% increase for group 1, a 2121% increase for group 2, a 177% increase for group 3 and a 1242% increase for group 4.

The trace cache miss rate of each of the architectures is detailed in Figure 5.4. It shows that the enabling of HT has an effect on the trace cache miss rates of the different architectures, leading to a 101.5% increase for group 1, a 20.9% increase on group 2, a 12.9% decrease for group 3 and a 5.1% decrease for group 4.

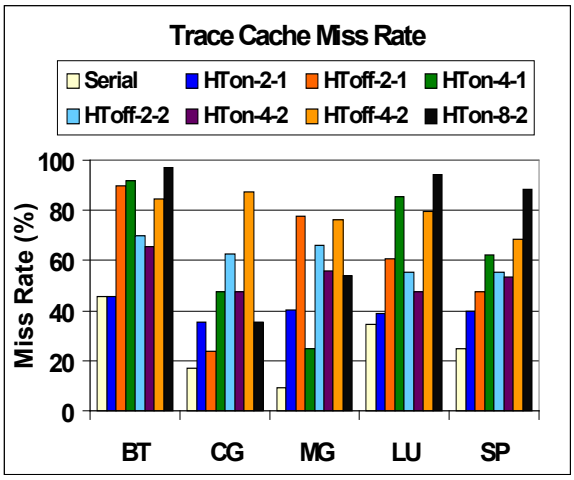


Figure 5.4: Trace Cache Miss Rates

The average trace cache miss rate is 59.5% across all applications for all HT_{off} configurations, while the average trace level cache miss rate for HT_{on} configurations is 56.8%. The high increase in trace cache miss rate for group 1 is most likely due to its

direct comparison to the serial case, where the trace cache of the Intel processor has a significant efficiency advantage over a multi-threaded workload, in which it must predict the behavior of several threads instead of a single active thread.

5.2.1.1 TLB Performance

By examining the Figures 5.5(a) and 5.5(b), we can see that the number of ITLB misses rises between the different groups. The number of ITLB misses increases as the complexity and resources of the architectures increase. This is true for group 1, group 2, and if CG and LU are excluded it is also true for group 3. For group 4 the ITLB misses see a decrease between the HT_{off} and HT_{on} configurations across all of the benchmarks except for CG. DTLB misses are relatively flat across all groups (except for HT_{on}-8-2 for BT and LU) indicating that the increases in complexity do not significantly impact the performance of the DTLB.

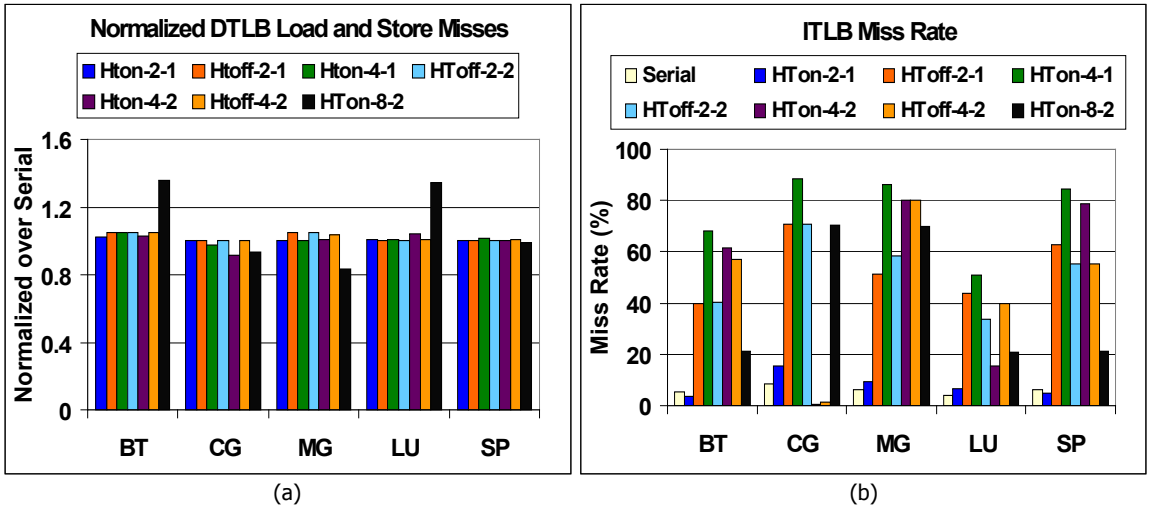


Figure 5.5: (a) DTLB Load and Store Misses Normalized to the Serial Case (b) ITLB Miss Rate

5.2.2 Stalled Operation

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 behaving the way that they do. Figure 5.6 illustrates the percentage of run time that the system spends in the halt state for a variety of architectures.

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 an indication of thread contention for shared resources in the cores. Group 2, 3 and 4 show similar patterns once again, with the HT_{on} configurations having more stalled cycles than the HT_{off} configurations. 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%.

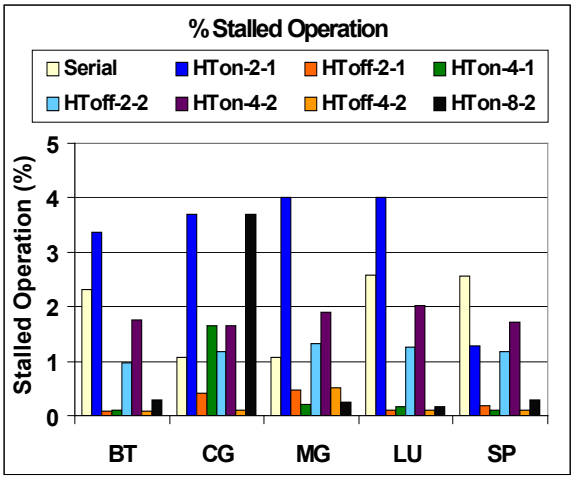


Figure 5.6: % of Total Execution Spent in a Stalled State

5.2.3 Branch Prediction

Figure 5.7 shows the branch prediction rate for each of the architectures for all five benchmarks. We can see that the branch prediction rates are excellent for almost all benchmarks across all configurations, with the exception of two of the HT_{on} configurations from groups 2 and 3 for CG and HT_{on}-8-2 for MG. The branch prediction rate does have an effect on the total number of stall cycles that an architecture incurs, as a mis-predicted branch can cause a memory order clear event and may affect the relevance of data in the trace cache. 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.

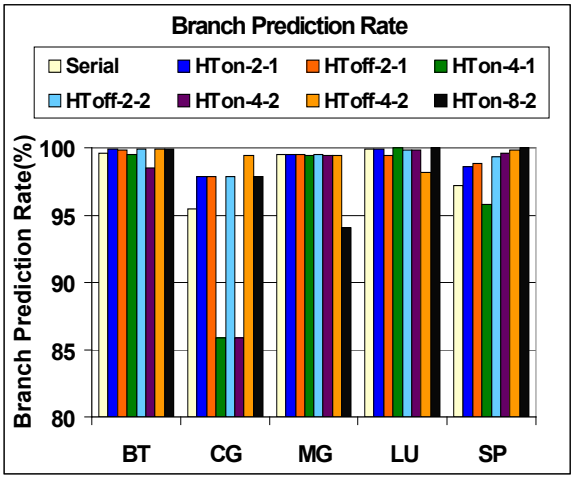


Figure 5.7: Branch Prediction Rate

5.2.4 Bus Transactions

When examining the bus transaction characteristics of the configurations presented in Figure 5.8, 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. Group 1 spends ~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.

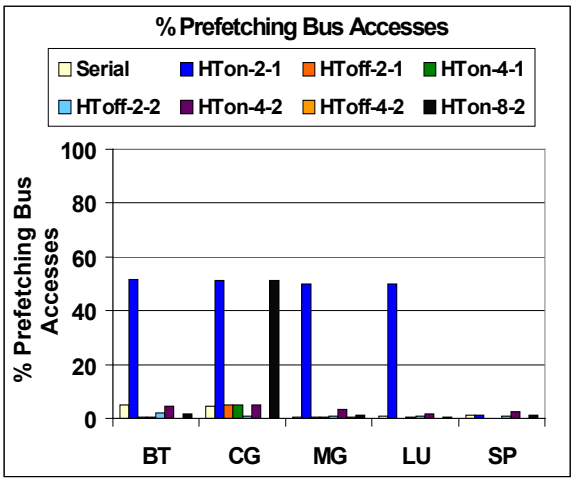


Figure 5.8: Pre-fetching Bus Accesses

5.2.5 Cycles Per Instruction

When examining the CPI of the different configurations presented in Figure 5.9, many of the observations made in the previous sections can be observed impacting the efficiency of the system. One can see a direct correlation between the results in the previous sections and the higher than average CPIs of some configurations when running

some of the benchmarks. 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 mis-prediction rate would cause many flushes of the execution pipeline and therefore reduce overall efficiency.

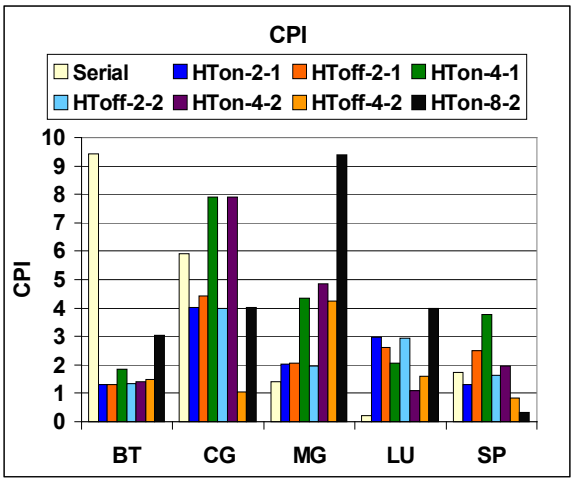


Figure 5.9: Cycles Per Instruction

5.2.6 Wall Clock Performance

The NAS benchmarks were run through a series of ten independent trials, with minimal variance between tests (<~1-5%). The results are detailed in Figure 5.10. The runtimes of the NAS benchmarks show an interesting trend that is new to the dual-core Intel Xeon architecture. Specifically, that the use of SMT on a CMT core can be extremely beneficial to the performance of a system. Of particular interest are the results for the 4 threaded HT_{on} case utilizing both cores on a dual-core chip in HT_{on} mode (HT_{on}-4-1). In this case the overall performance of a single SMT dual-core chip is comparable to the performance of two dual-core processors operating with HT_{off}. Despite the single HT_{on}-4-1 chip having half as many available computational resources, the appreciable slowdown over the dual processor dual-core SMP case is only 13.6%.

Overall, the HT_{off}-4-2 configuration has the best wall clock times, with the exception of CG, and has the highest average speedup across all of the applications. This follows

the same trend found for the HT_{on}-4-1 case, in that despite the fact that the CMT architecture has half as many available computational resources, the appreciable slowdown over the CMP-based SMP case is only 12.8%.

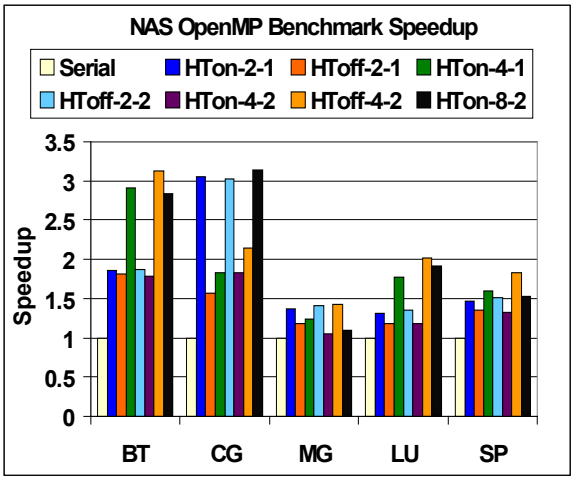


Figure 5.10: Speedup for NAS OpenMP applications.

When overall processor resources are increased to utilize two dual-core processors with HT_{on} (HT_{on}-8-2), the results are different to previously observed HT-related behavior in that the overall effect on performance is minimal, in contradiction to previous work on single-core HT architectures in chapter 4. However, it should be noted that in the case of applications in which there is a significant amount of data sharing, HT has excellent speedup (such as in the case of the application CG) which offsets its poorer performance on the other benchmarks.

The overall average speedup for group 1 for the HT-enabled architectures versus the HT-disabled case is 1.81 versus speedup of 1 for the serial case. The average speedup for group 2 was 1.86 for the HT_{on} case and 1.42 for the HT_{off} case. Groups 3 and 4 had average speedups of 1.43 and 2.10 for the HT_{on} architectures and speedups of 1.83 and 2.10 for the HT_{off} cases, respectively.

Except for the CG case, 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-pre-fetching 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 5.02 which would imply that the performance of the HT_{off}-4-2 case should be superior, but a large number of bus transactions in the HT_{on}-8-2 case are speculative pre-fetching (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 pre-fetching. 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 for all of the benchmarks with the exception of CG and MG, with the HT_{on}-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. The average speedup of each of the architectures is detailed in Table 5.2.

Table 5.2. Speedup for architectures

SMT (HT_{on}-2-1)	CMP (HT_{off}-2-1)	CMT (HT_{on}-4-1)	SMP (HT_{off}-2-2)	SMT based SMP (HT_{on}-4-2)	CMP based SMP (HT_{off}-4-2)	CMT based SMP (HT_{on}-8-2)
1.81	1.42	1.87	1.83	1.43	2.11	2.10

5.3 Multi-Application Results

The performance of the given architectural configurations is also of interest in a multi-programmed environment. The following results were collected using the same configurations as in section 5.2, 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. As such, these results are not directly comparable to those in the previous section. The goal of these multi-application tests is to determine the performance of the different available system configurations in a variety of multi-application workloads. All of the workloads used for testing fully load the system, using all of the system resources in a balanced way between the applications; for example, the HT_{off}-4-2 configuration uses four threads, two for the first application and two for the second application, while HT_{off}-2-1 uses two threads, one for the first application and one for the second application.

Two NAS benchmarks were selected for this task, the FT benchmark, which is a Fourier transform application requiring mostly computational resources and limited memory resources, and the CG benchmark, which requires significant memory resources.

Three separate tests were conducted; the first used a combination of CG and FT for a computationally demanding application paired with a memory intensive program. The second used two copies of FT to determine the system's performance with mostly computationally intense workloads, and the third used two copies of CG to determine the system's performance under a memory intensive workload. The maximum number of execution threads available to each system configuration was used, with the threads being distributed evenly between the executing programs. The threads were not tied to any specific processor, and the scheduler was free to relocate threads amongst the available processors.

5.3.1 Cache Performance

By studying the L1 and L2 cache miss rates of the different architectures presented in Figure 5.11(a) and 5.11(b) we can observe that the 1st level cache miss rates are relatively stable across the different configurations. However, the 2nd level cache miss rates show that the HT_{on}-2-1 configuration has difficulty achieving a high hit rate for the CG benchmark, as does the HT_{on}-8-2 configuration.

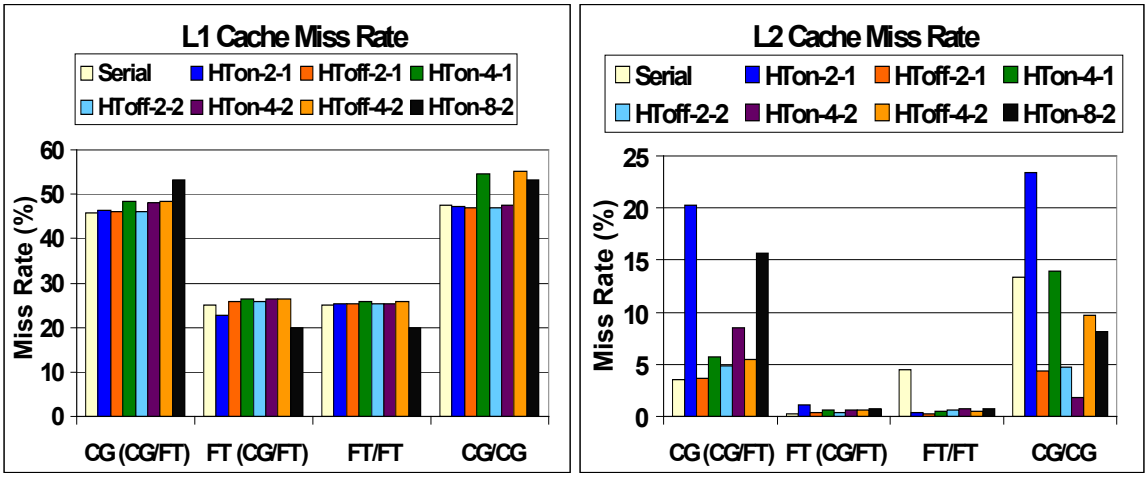


Figure 5.11: (a) L1 Cache Miss Rate and (b) L2 Cache Miss Rate

In general, all of the HT_{on} configurations have a worse L2 miss rate than their HT_{off} equivalents. This is most likely due to cache contention as the two programs cause cache evictions of important data belonging to the other program. This also explains why CG is affected to a greater degree by the system sharing than the FT benchmark as it uses a much larger data set and is more memory intensive than the FT benchmark.

The trace cache miss rates shown in Figure 5.12 illustrate that the HT_{off} configurations for both group 2 and group 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 fairly significant, but the advantage is even greater for the HT_{on} configuration in group 3. Finally, group 4 shows that there is no advantage to the HT_{on} configuration in terms of trace cache misses in any of the workload configurations.

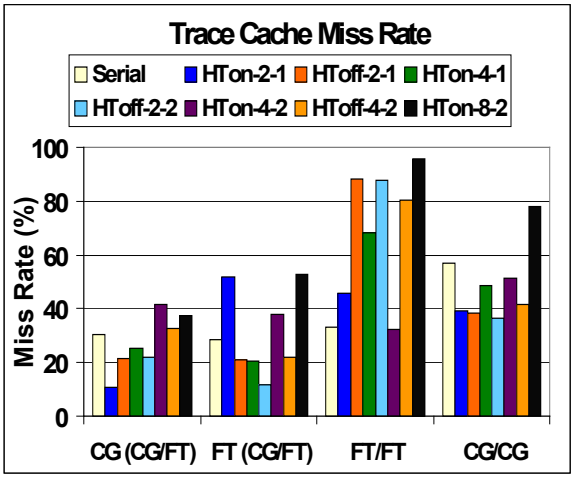


Figure 5.12: Trace Cache Miss Rate

5.3.1.1 TLB Performance

Figures 5.13(a) and 5.13(b) show that the HT_{on} configurations suffer from excessive ITLB misses in both groups 2 and 3 when running the CG benchmark.

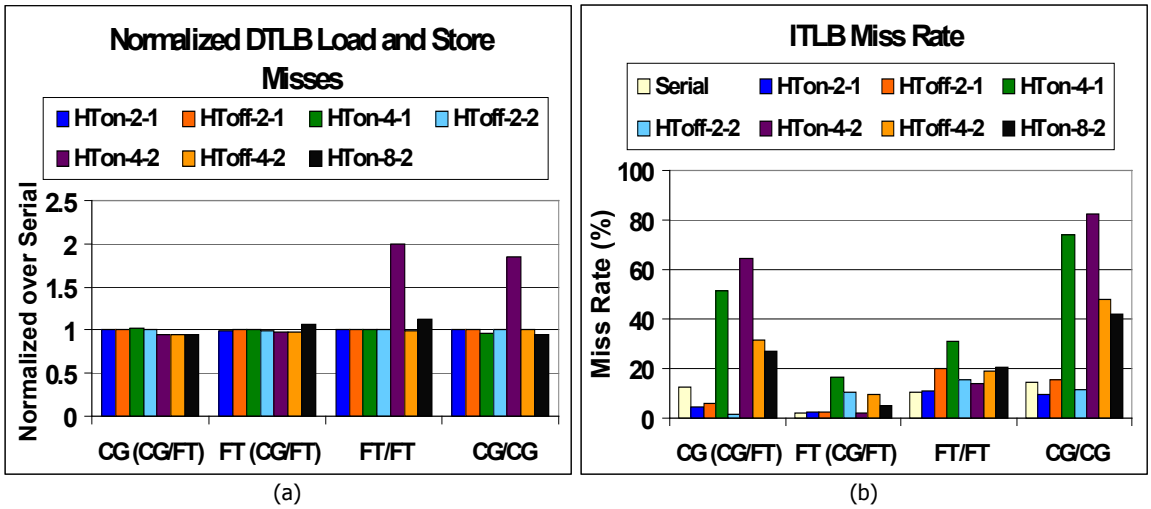


Figure 5.13: (a) DTLB Load and Store Misses Normalized to the Serial Case (b) ITLB Misses

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.

5.3.2 Stalled Operation

The percentages of the total run time that the system spends in a stalled state are presented in Figure 5.14. Upon examination, the amount of total execution cycles spent in a stalled state for this multi application workload is surprising. When running a supposedly 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.

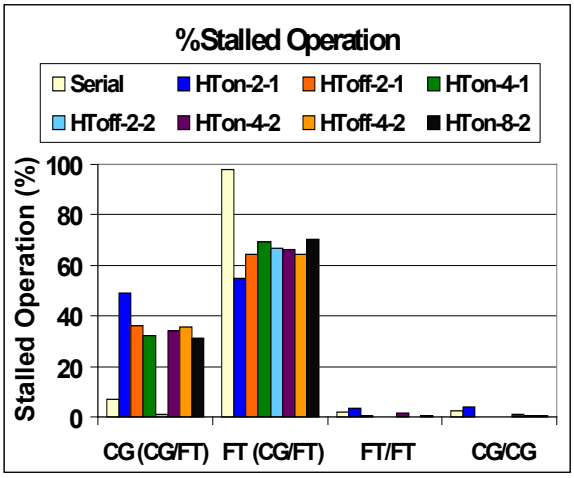


Figure 5.14: Percentage of Operation Time Spent Stalled

5.3.3 Branch Prediction

When examining the branch prediction rate detailed in Figure 5.15, 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} and HT_{off} configurations.

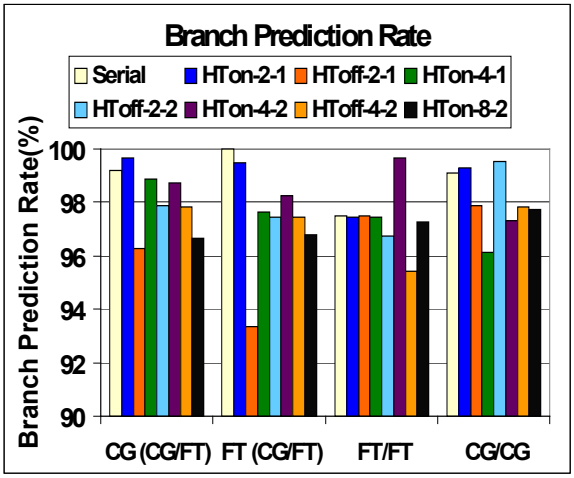


Figure 5.15: Branch Prediction Rate

5.3.4 Bus Transactions

The pre-fetching activities being undertaken by each of the configurations presented in Figure 5.16 reinforce the stalled operation results examined earlier in this section. The configurations spend a significant amount of time pre-fetching 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.

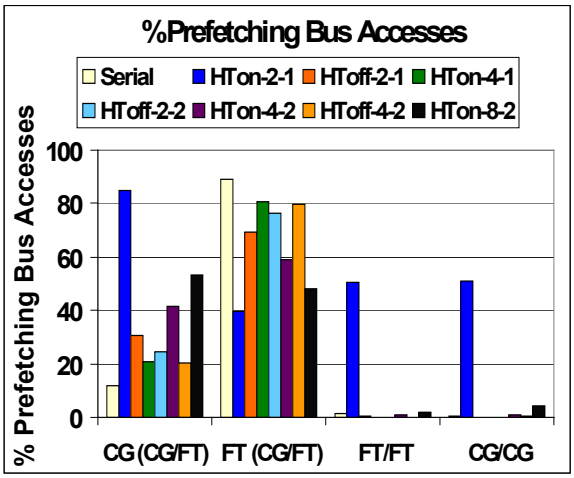


Figure 5.16: Percentage of Pre-fetching Bus Accesses of All Bus Accesses

5.3.5 Cycles Per Instruction

The CPI results in Figure 5.17 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} configuration is worse than the HT_{off} configuration for all workloads.

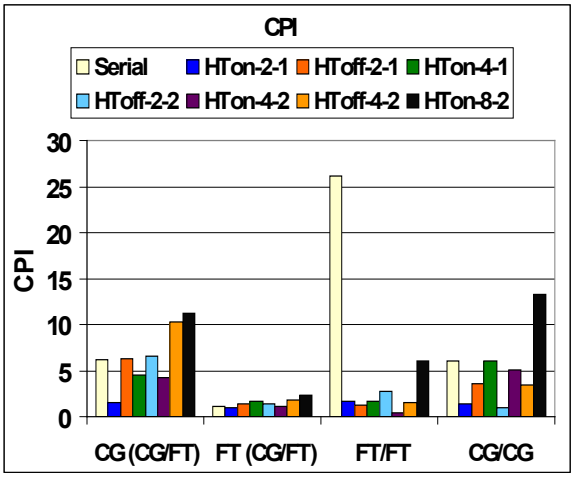


Figure 5.17: Cycles Per Instruction

5.3.6 Wall Clock Performance

The results examining multi-application performance are presented below in Figures 5.18(a), 5.18(b) and 5.19. The goal of these multi-application tests was to determine the performance of the different available system configurations in a variety of multi-application workloads. All of the workloads used for testing fully load the system, using all of the system resources in a balanced way between the applications, for example, the HT_{off}-4-2 configuration uses four threads, two for the first application and two for the second application, while HT_{off}-2-1 uses two threads, one for the first application and one for the second application. The applications were specifically chosen because CG is typically memory bound, while FT is compute bound. Thus, the performance numbers presented in the Figure 5.18(a) directly correlate to running half of a system with compute bound threads and half with memory bound threads. Figure 5.18(b) demonstrates the system performance for an entirely compute bound workload, while Figure 5.19 demonstrates the performance of a fully memory bound workload. The results clearly indicate that there is a tangible performance benefit to running compute bound and memory bound applications in parallel, as the performance of both applications is better in such a balanced environment than a system running alike applications.

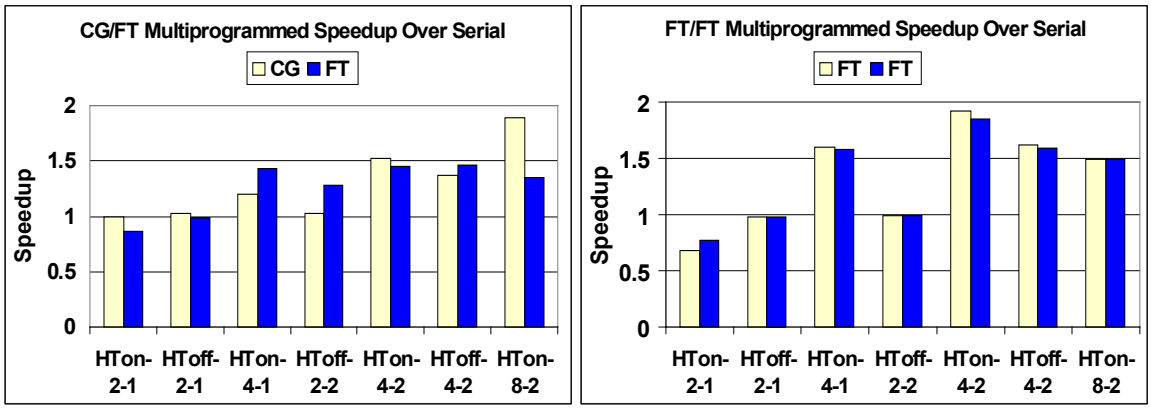


Figure 5.18: (a) CG/FT and (b)FT/FT Multi-Application Speedup

The speedups show that by a small margin, the applications enjoy running with themselves, but small variances occur that make it difficult to make strong generalized conclusions. The overall best performer of any configuration is HT_{on}-4-2, which is the fastest overall for 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, HT offers a tangible performance benefit, whether the system is balanced in its workload or not. In addition, it indicates that for multi-application workloads, clever scheduler design could achieve optimal performance of the system by utilizing HT and varying the number of threads for the active applications running on the system.

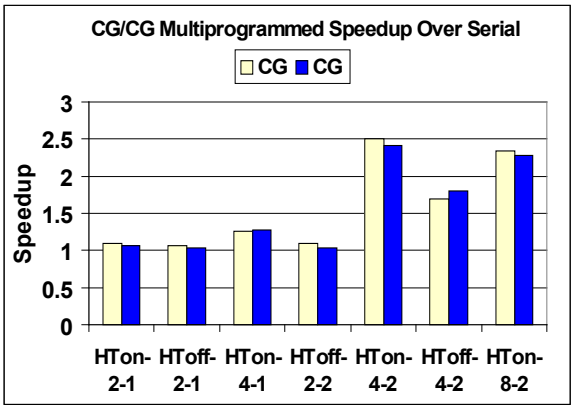


Figure 5.19: CG/CG Multi-Application Speedup

5.3.7 Cross-Product Multi-Program Results

The different architectural configurations were tested using a pair of applications, and completed for all possible two-program pairs in all configurations. The program pairs were run with enough evenly distributed threads as to fully load the architecture under

test. The results are shown in a box and whiskers plot in Figure 5.20. The boxes in the figure represent the 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.

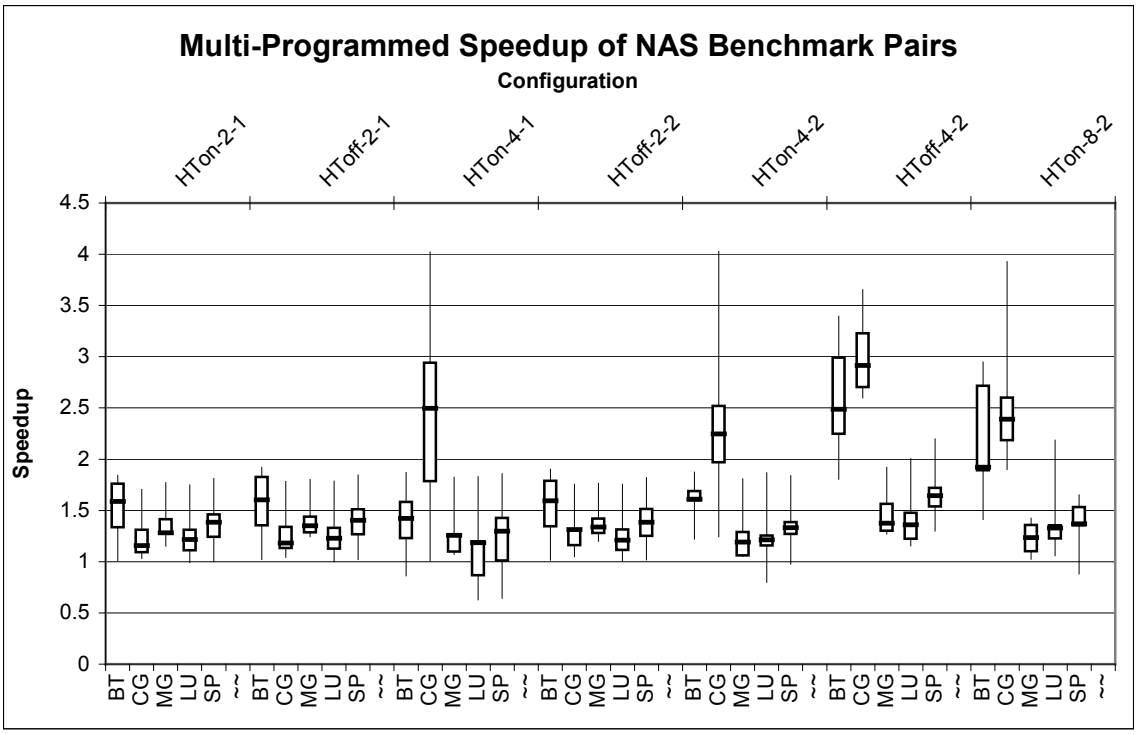


Figure 5.20: Multi-programmed speedup of pairs of NAS benchmarks for all architectures

5.4 Overloaded Configuration Analysis

The four configurations presented in Figures 5.21 to 5.28 represent balanced yet overloaded workloads for the processors. All four cases have twice as many threads as they do execution contexts in which to run them. The LU benchmark has been omitted

from these tests as it suffers from incredibly high runtimes when the system is overloaded.

5.4.1 Cache Performance

In Figure 5.21(a) and 5.21(b) we can see that the cache performance of the overloaded cases is excellent given their intensive workload. The L1 cache miss rates stay relatively stable across all of the configurations, with a small increase in the HT_{on}-8-1-4 case for CG. L2 cache miss rates are similar to those for the non-overloaded cases, rising by a significant but not overwhelming amount given the increased workload, with the majority of the increase in miss rate occurring from the SP benchmark.

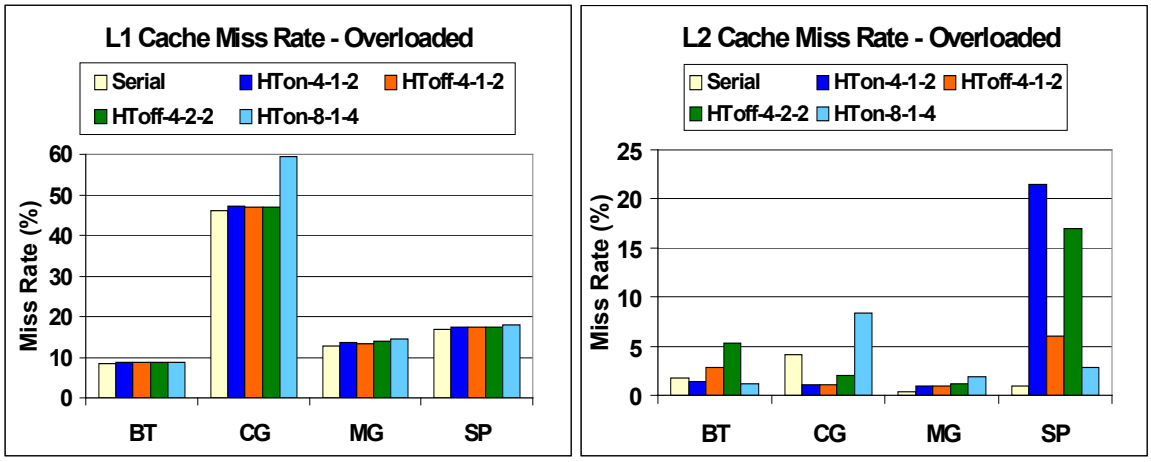


Figure 5.21: (a) 1st Level Cache Miss Rates for Overloaded Cases, (b) 2nd Level Cache Miss Rates for Overloaded Cases

The trace cache miss rates for the overloaded cases are shown in Figure 5.22. The best trace cache performance for the overloaded cases occurs with both of the HT_{on} configurations, both having much better cache miss rates than their HT_{off} alternatives. This illustrates that the HT_{on} configurations are benefiting from their shared trace cache. This means that the system is fairly well balanced in terms of program synchronization if the HT_{on} configurations are able to exploit their shared trace cache.

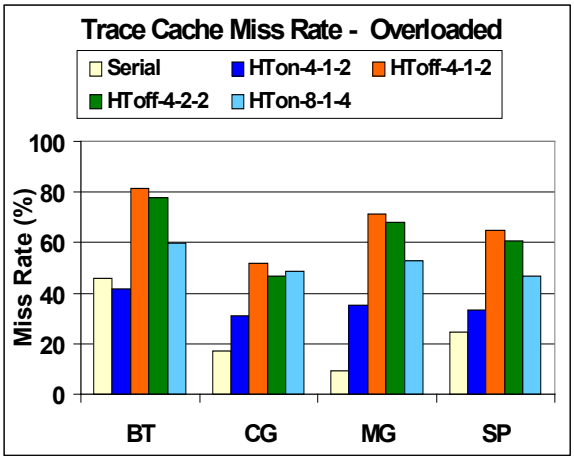


Figure 5.22: Trace Cache Miss Rates for Overloaded Cases

The DTLB miss performance of the overloaded cases in Figure 5.23(a) is similar to that for the non-overloaded cases in Figure 5.5, seeing no significant increase due to the thread overloading. The ITLB performance of the overloaded cases in Figure 5.23(b) shows an increase for most of the configurations with the exception HT_{on}-8-1-4, which sees a real benefit to its ITLB miss rate when overloaded.

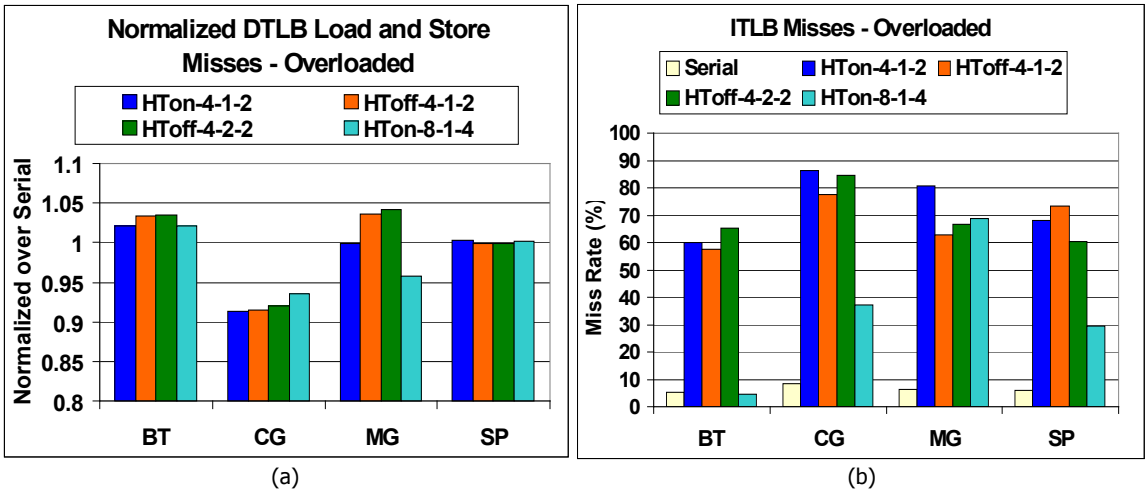


Figure 5.23: (a) DTLB Load and Store Misses and (b) ITLB Miss Rates for Overloaded Cases

5.4.2 Stalled Operation

The percentage of the total number of execution clockticks in which the system was stalled is presented in Figure 5.24. We can see that it is slightly higher for an overloaded configuration than for a non-overloaded configuration in Figure 5.6, rising by as much as 3.9% over the non-overloaded configuration, with HT_{on}-4-1-2 seeing the highest increases. This is to be expected as we overload the system, as it will require more

memory order clear instructions and in general require the flushing of the pipeline more often, especially when switching contexts to a new execution thread.

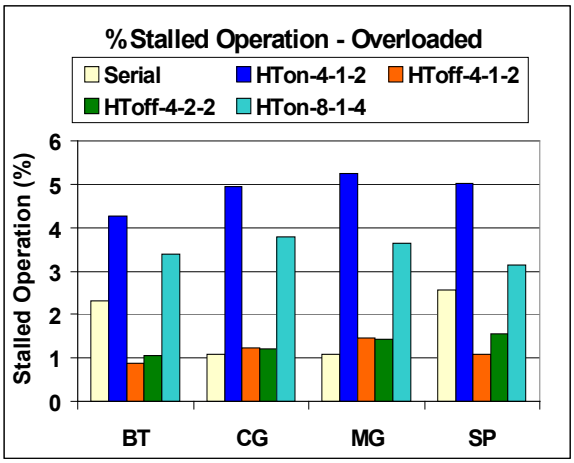


Figure 5.24: Percentage of Stalled Operation for Overloaded Cases

5.4.3 Branch Prediction

The branch prediction rate for the overloaded cases detailed in Figure 5.25 is excellent. The branch prediction problems that occurred with the HT_{on}-4-1 non-overloaded configuration (Figure 5.7) are no longer occurring and overall the configurations are doing well compared to their non-overloaded configurations.

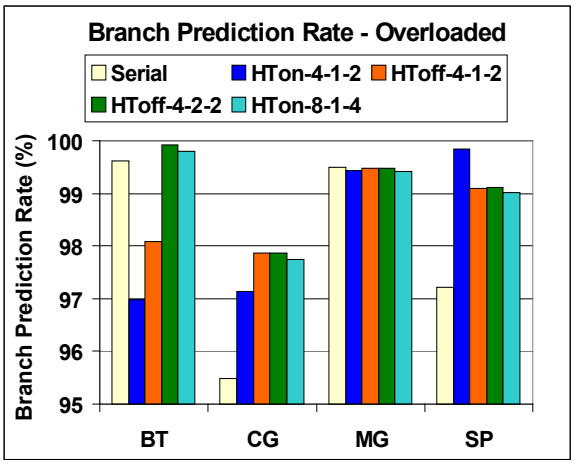


Figure 5.25: Branch Prediction Rate For Overloaded Configurations

5.4.4 Bus Transactions

The percentage of bus accesses that are for pre-fetching activities is shown in Figure 5.26. It can be seen that for both of our HT_{on} configurations, the limiting factor to performance appears to be the computation resources available to the system, while the

HT_{off} configurations are still limited by memory performance. This is keeping in line with the observations made for the configurations in their non-overloaded states as shown in Figure 5.8.

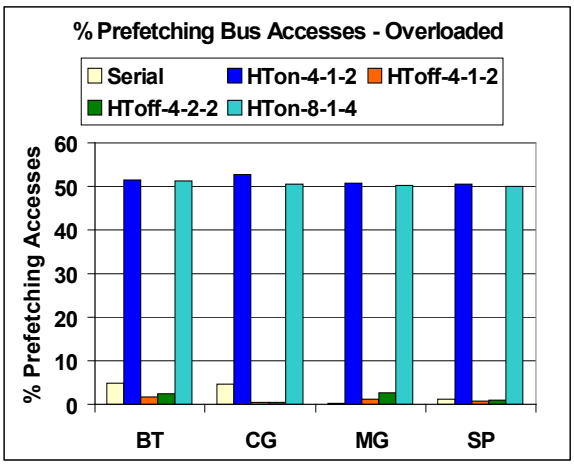


Figure 5.26: Percentage of Pre-fetching Bus Accesses For Overloaded Configurations

For the majority of the applications, this advantage in pre-fetching does not necessarily lead to a performance increase as the cache hit rates are still very comparable to the HT_{off} cases which rely on less overall pre-fetching activities as illustrated in Figure 5.8. This explains why the HT_{on} cases do not gain a significant performance advantage over the SMP cases, as the HT_{on} cases also have less total available cache memory to them as more threads share a smaller overall cache area.

5.4.5 Cycles Per Instruction

Comparing the CPI of the overloaded configurations in Figure 5.27 with the non-overloaded configurations in Figure 5.9, we find that the CPIs are not greatly affected by overloading. The overall impact for all of the cases is an average increase in CPI by 1.35%, with the best performance being a drop of 46.3% in CPI for the HT_{on}-8-1-4 configuration for the BT benchmark, and the worst being a CPI rise of 48.8% for the HT_{on}-4-1-2 configuration for the SP benchmark.

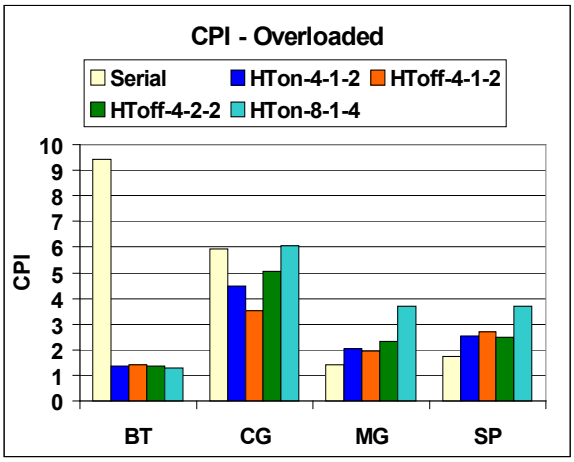


Figure 5.27: CPI For Overloaded Configurations

5.4.6 Wall Clock Performance

The behaviour of the HT for the overloaded case of HT_{on}-8-1-4, where 8 threads are executing on a single dual-core processor with HT shows interesting results. With the exception of BT, the HT_{on}-8-1-4 case is one of the fastest of all of the tested configurations. Overall, it results in a slowdown of only 7.6% versus the HT_{off}-4-2 case. This is 0.9% slower than the slowdown seen by the HT_{on}-4-1 configuration. However, when you consider the best case for each benchmark between the HT_{on}-8-1-4 and HT_{on}-4-1 the overall result is a speedup over the HT_{off}-4-2 case of 3.25%. This implies that there are future opportunities for performance better than that of a system with twice the computational resources by using intelligent scheduling techniques with HT-enabled on a single dual-core processor.

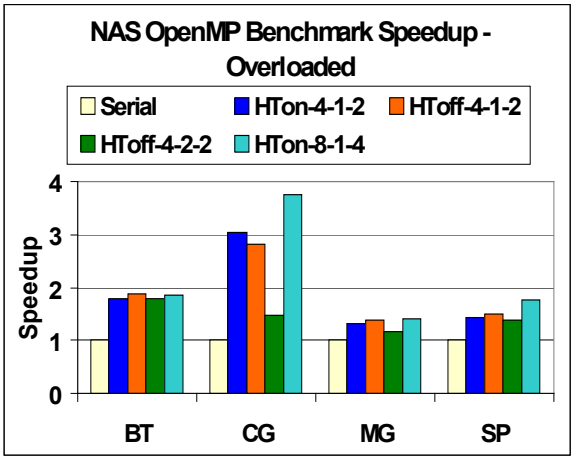


Figure 5.28: Overloaded NAS Benchmarks Speedup

It is interesting to examine the performance numbers from the CG benchmark, specifically the overloaded HT_{on}-8-1-4 case versus the HT_{off}-4-2 case. In general, this benchmark benefits from additional execution threads that are located on physically close cores. Interestingly, the best performance on this benchmark comes from the HT_{on}-8-1-4 case, even though the cache hit rates and trace cache hit rates are lower than other configurations. However, the overloaded cases have a slight advantage in terms of efficiency of bus usage over the non-overloaded cases, as the HT_{on}-8-1-4 case executes significantly more pre-fetch operations than do the non-overloaded cases. The percentage of operations that were pre-fetching on the HT_{on}-8-1-4 configuration was only 49.5% versus a 0.08% value for the HT_{off}-4-2 case, as illustrated in Figure 5.10. The HT_{on} configurations have an advantage in terms of executing more pre-fetching operations than the HT_{off} configurations throughout all of the tests, but in benchmarks, this pre-fetching is not beneficial to overall system performance. In the case of the CG benchmark, the pre-fetching significantly improves the performance of the application.

5.4.7 Overloaded Overhead

Table 5.3 provides a comparison of the overloaded cases with their architectural equivalent non-overloaded case from section 5.2.

Table 5.3. Percentage degradation for overloaded cases versus non-overloaded cases

	SMT	CMP	CMT	SMP
L1	31.7	-0.5	14.6	0.2
L2	317.8	-35.9	-38.8	574.6
Trace cache	-12.3	12.6	-8.2	-0.3
ITLB	96.5	1.2	-31.8	20.4
DTLB	-7.1	0.9	-5.3	-2.2
Stalled Operation	57.6	4.0	575.8	303.5
Branch Prediction	-0.6	0.9	4.0	-0.4
Bus Transactions	33.0	0.7	3510.6	-29.3
CPI	20.4	0.9	-17.8	-6.7
Speedup	-4.2	-24.8	-14.8	25.7

One can observe that overloading the architectures has beneficial effect on speedup, particularly CMP and CMT, when neglecting the LU benchmark. The SMT configuration has increased cache miss rates for all levels of cache and sees a large increase in CPI. The CMP architecture sees a drop in L1 and L2 cache and marginal increases for its TLBs, with only minor increases in stalls and CPI. CMT sees mostly increasing cache miss rates, but also has a significant increase in stalls and pre-fetching activities. SMP sees increases in cache miss rates and stalls, but enjoys higher pre-fetching rates and a lower CPI.

5.5 Effect of Operating System Noise

Operating systems have been shown to have an effect on the performance of intensive workloads in multi-processor systems [45, 71]. The overhead required to maintain OS services does not represent a large portion of the system's overall computational load, but the time spent providing system services can cause the greater computational workload to lose synchronicity which creates slowdowns, as the workload must wait at synchronization barriers. As the number of simultaneously executing threads increases so does the penalty that is incurred at barriers, as more threads wait for a small proportion of threads that are lagging behind the average threads in the workload. These effects have been observed in large cluster systems using MPI applications and real-time systems, so it is reasonable to investigate multi-core systems to determine the extent to which operating system noise affects the performance of such platforms running multi-threaded applications in OpenMP.

While there have been some effective techniques proposed in [45, 71] to reduce the impact of system noise, such as removing unnecessary OS daemons and kernel threads (or moving them to another processor), lowering tick rate, and co-scheduling, leaving one processor for OS tasks is still a simple, viable option to effectively separate system noise from the computation [89]. Meanwhile, past work on real-time processing with Linux schedulers [11] has found that reserving a CPU specifically to respond to real-time priority threads significantly decreases the latency for real-time threads as well as the interrupt response time. These solutions have all addressed the performance impact of OS noise, and as such it is pertinent to attempt such an approach with our system in an

attempt to increase performance. This is a useful approach as it helps alleviate unwanted cache evictions caused by OS threads that adversely impact our HPC applications.

5.5.1 Operating System Noise Effects on Single-Threaded Applications

In order to verify that operating system noise has an effect on smaller SMP systems a preliminary test of five NAS benchmark applications was run on a system using a single execution thread. The Linux processor affinity mask was used to ensure that processes were bound to a specific CPU. One test was performed with all threads in the system bound to a single processor (using the affinity mask), including the benchmark thread. A second test was performed on the same machine with the benchmark execution thread on a secondary processor while all operating system tasks were assigned to the primary processor. Each application was run several times in order to ensure accurate and reportable results. The results of these tests are detailed in Figures 5.29 and 5.30. Figure 5.29 illustrates the effect of operating system noise on the cache performance of the system. The effect of operating system noise on the cache performance of the LU benchmark is noteworthy, resulting in an increase in cache hit rate of 20.6%. The remainder of the applications show a minor improvement or no change in their cache hit rates.

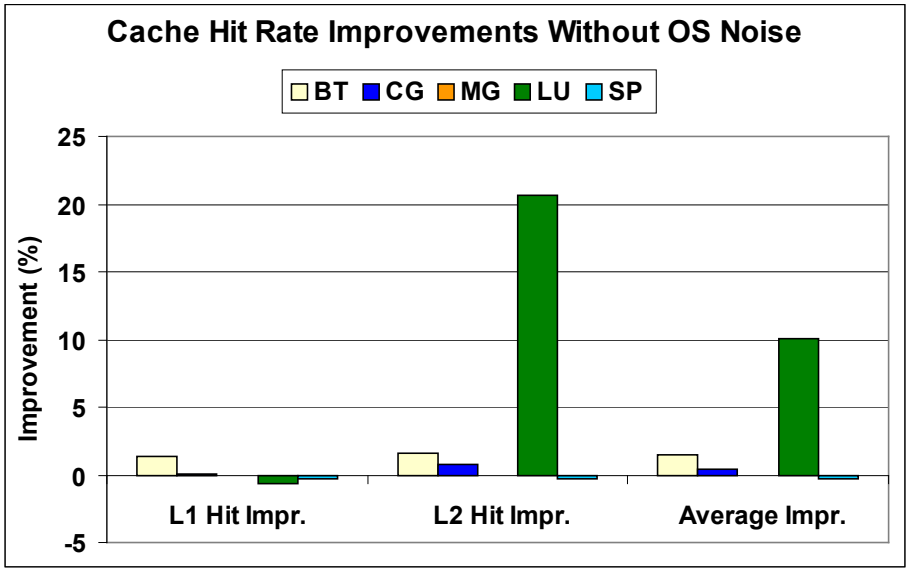


Figure 5.29: Improvement in cache hit rate without OS noise

The results in Figure 5.30 indicate that the system can experience up to a 9.1% performance impact due to operating system noise, and all applications show some

decrease in performance due to the noise. The LU benchmark has seen a decrease in runtime corresponding to its increased cache hit rate, as have BT and CG. SP and MG have also both seen a decrease in runtime when OS noise is removed, indicating that the technique is effective for the entire range of benchmark tests.

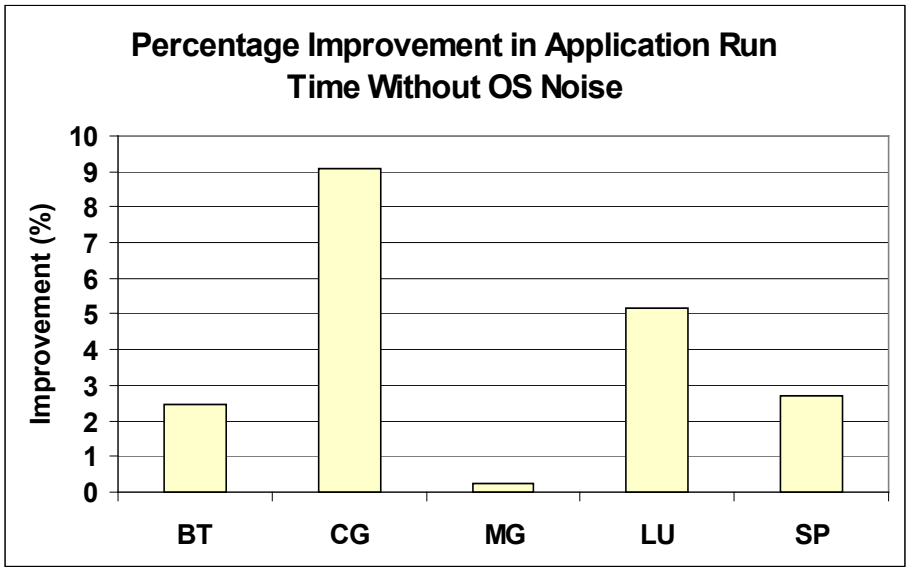


Figure 5.30: Effect of operating system noise on system performance

5.5.2 Operating System Noise Effect on Multi-threaded Applications

Given the findings of the serial case research, a comprehensive study of the system under many different configurations was performed. Operating system noise was isolated on the system by masking off a single processor (logical or physical depending on the configuration), and assigning OS tasks to that processor. The resulting data, shown in Figure 5.31, indicates that the effect of OS noise can be significant with modern multi-core processors. OS noise almost always results in an increase in L1 and L2 cache misses, seeing an average degradation of 1.07% for the L1 cache hit rate and 3.02% for the L2 cache across all configurations and applications of the NAS benchmarks.

The average improvement in cache hit rate for each configuration is shown in Figure 5.31 for the NAS benchmarks. The improvement percentages for each architecture correspond to the average improvement in L1 and L2 cache hit rates across all five of the NAS benchmarks. The increase of cache hit rates is of particular importance, as the bottleneck to performance for such applications is most typically the memory access latencies and memory bandwidth.

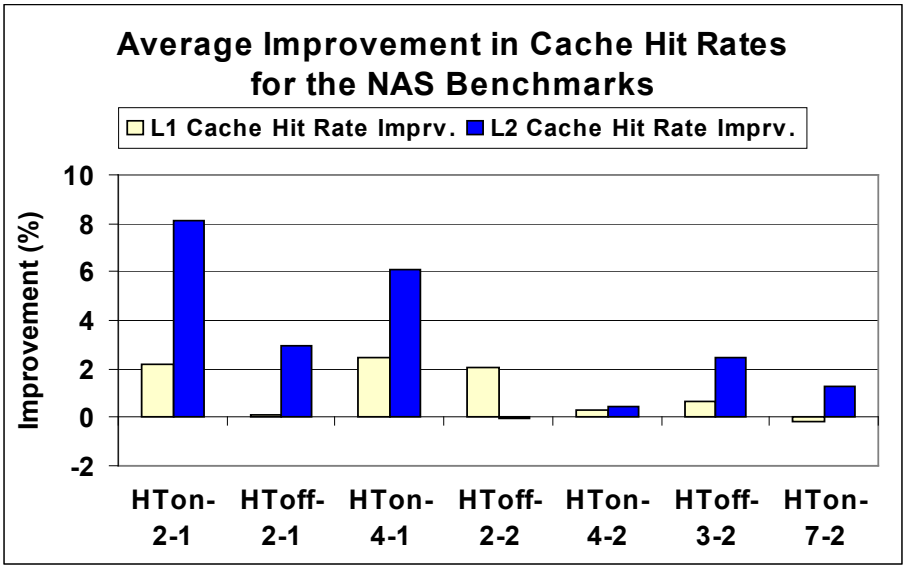


Figure 5.31: Improvement in Cache Hit Rate Without OS Noise

For each of the configurations, the trend is that the degradation of cache hit rates is inversely proportional to the total number of processors in the system. This is expected as the more processors that are in the system, the less overall impact occurs when a single processor is assigned to an OS task. This corresponds with a resulting decrease in application runtimes, particularly for programs that are significantly impacted by OS noise, resulting in significant gains in terms of wall clock execution time.

Two of the configurations presented here are special cases in that they were tested using one thread less than their operating system loaded counterparts. This was unavoidable for testing these configurations, as the OS noise requires a free processor to offload its overhead onto, and for the fully utilized system configurations, this is impossible. This means that although the HT_{on}-7-2 and HT_{off}-3-2 configurations may show a slowdown versus their operating system loaded partners, the systems operating without operating system noise have fewer overall resources available to them. Despite this handicap they still manage to have gains in their respective cache hit rates for the benchmarks. The HT_{on}-7-2 and HT_{off}-3-2 configurations show negligible improvement for their L1 cache hit rates, but have an improvement of 1.27 and 2.43% respectively for their L2 cache hit rates. Despite having fewer resources available to it, the HT_{on}-7-2 configuration sees an 8.3% improvement in runtimes over all of the applications, with only one application seeing an increase in runtime resulting in a slowdown of 3.64% for the BT benchmark. The HT_{off}-3-2 configuration sees a much larger increase in wall

clock execution times seeing an average slowdown of 16.5% across all of the applications, with only one application, CG, showing an improvement of 1.55%.

The effect of operating system noise on the runtime of such scientific applications is visible, with an average percentage decrease in wall clock times of 3.8% for all of the applications across all of the configurations. If the results for the two configurations that have less overall system resources available to them due to the isolation of operating system noise (HT_{off}-3-2 and HT_{on}-7-2) are removed from the group, the results improve to an average percentage decrease in wall clock time of 7.0%. All of the configurations (except for HT_{off}-3-2) see a decrease in wall clock run times, ranging from 0.3% to 13.5%. The improvements in runtimes are detailed in Figure 5.32.

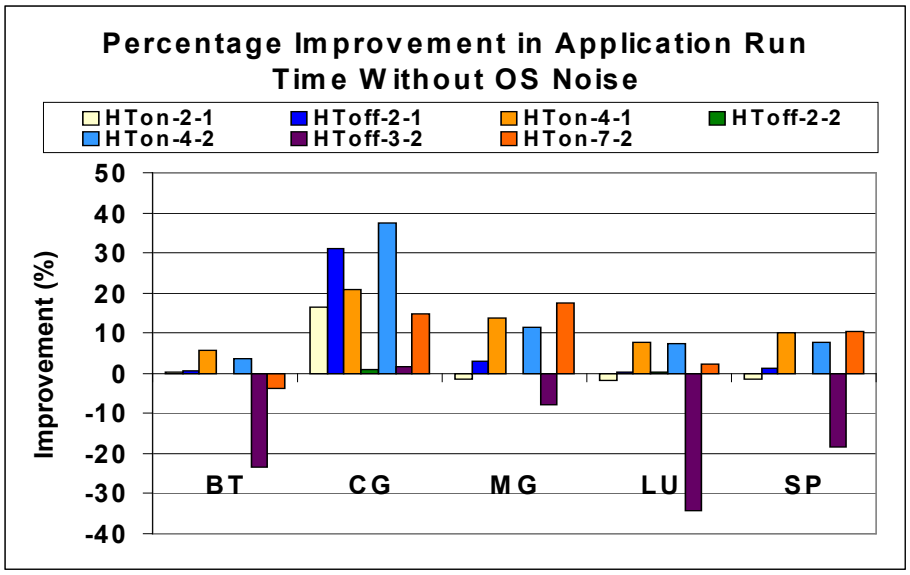


Figure 5.32: Application Run-time Improvement

5.6 Summary

In this chapter, we presented performance of scientific applications from the NAS OpenMP suite on a range of system configurations with kernel 2.6.9 on a 2-way dual-core Hyper-Threaded SMP. Our performance results indicate that the majority of applications could benefit from using a single dual-core processor with HT enabled, in terms of total computing power per system resources available. However, only one application enjoyed performance gain of due to HT on both dual-core processors.

By collecting data from hardware performance counters, we analyzed the effect of HT on the various system configurations as well as the effect of thread overloading on the

system. When utilizing all of the available system resources, most applications suffer from the increasing number of cache misses when both dual-core processors are enabled with HT enabled.

The decisions made by the scheduler are crucial to the performance of HT. With the optimization of the scheduler the performance of a single processor could be increased for scientific applications to almost the same performance level of a system with twice as many non-HT processors.

In addition, it has been determined that operating system noise may cause significant performance degradation and could be a potential source of optimization for small scale multi-core SMPs. The next chapter uses the findings of this chapter to propose a method of improving system performance, and takes advantage of an opportunity to reduce system power consumption at the same time.

Chapter 6: Power Management of Chip Multi-Threading SMPs

Power consumption is an important design constraint in modern day servers and high-performance server clusters. This chapter explores the power-performance efficiency of Hyper-Threaded AMP servers, and proposes a scheduling algorithm that can be used to reduce the overall power consumption of a server while maintaining a high level of performance. An AMP is a system that has a heterogeneous collection of processors. The processors can be of different types and/or operate at different speeds. The AMP presented here has identical processors, running at different speeds.

This chapter proposes a modification to the Linux scheduler as a method of potentially reducing the power consumption of a system, while producing less of a performance impact on the system than would have been otherwise achieved using the default process scheduler [32, 34]. Previous research has shown that system noise, including operating system (OS) interference with the application, has a dramatic effect on high-performance computing [71]. Using static clock throttling and processor affinity, we bind all OS activities to logical processor zero (or physical processor zero) that runs at a lower frequency than the rest of processors in the AMP. In order to sustain the performance for the parallel OpenMP threads, all other processors run at their maximum frequency.

The results in the previous chapter concerning the effect of operating system noise on system performance have been the motivation behind this method to offload system noise onto a single processor. This reduced noise should correspond to an increased performance of the user threads such that the impact of reserving the CPU for system tasks is minimized. In the event of a system load that does not correspond to a full load for a single processor, there exists an opportunity to reduce the frequency of the reserved CPU such that its load is as close to 100% as possible. This clock throttling of the reserved CPU has a power savings effect.

The rest of this chapter is organized as follows. In Section 6.1, we describe the experimental framework including the AMP setup. Section 6.2 examines the performance increase that is possible using the proposed scheduler over the default scheduler. Section 6.3 describes the real power consumption measurements for a dual-

core 2-way CMP/SMT hybrid that uses clock throttling. Finally, we predict the effect that true frequency scaling would have on the power consumption of a dual-core 2-way CMP/SMT hybrid system in section 6.4.

6.1 Experimental Framework

The experiments in section 6.2 & 6.3 were conducted on a dual-core Dell PowerEdge 2850 server. The specifications of this platform can be found in section 5.1.

6.1.1 AMP Setup

To evaluate the power-performance efficiency of AMP over SMP systems, we created static AMP configurations on our 2-way dual-core platform through clock throttling and affinity control. In clock throttling, one can set the duty cycle to one of the seven available levels. Clock throttling has a similar impact on performance as reducing the frequency [3], but is not as ideal a solution as true frequency scaling, which should further increase the potential energy savings of the approach detailed in this chapter. This effect is predicted in Section 6.3.

The Linux 2.6.9 kernel supports clock throttling through a sysfs interface with appropriate drivers. The system was configured to enable CPU frequency scaling using clock throttling. The p4-clockmod driver was built into the kernel and the standard sysfs interface was used. The frequency governor was set to user-space control, creating a static operating point for clock throttling. By static setup, we mean the duty cycle is set only once before the application run.

We have implemented a new Linux scheduler, to be called *power-saving scheduler* (PS-Scheduler), by modifying the Linux scheduler to reserve a single CPU that runs only kernel threads, leaving the rest of the CPUs in the system to execute all user threads at maximum frequency. This is accomplished by using the processor affinity properties available in the Linux 2.6.9 that allow processes to be bound to a specific set of processors, or an individual processor.

The available operating points for the 2-way dual-core platform were 2.8GHz, 2.4GHz, 2.1GHz, 1.8GHz, 1.5GHZ, 1.2GHz, 900MHz, 600MHz, and 300MHz. The CPU frequency of the first physical processor was adjusted throughout the available

operating points for the execution of system activities, while the rest of processors in the system remained at the highest available clock frequency to run the application threads. However, our experimentation with the application benchmarks revealed the performance of the AMPs decreased significantly when the clock speed was reduced to under two times the front side bus (main memory pathway) speed of the machine, so only results from the operating points above two times the front side bus of each machine are reported.

It should be mentioned that the duty cycle can only be set on a per physical processor basis on Intel multiprocessors. Therefore, in the case of an HT-enabled system, this creates an asymmetrical imbalance among the logical processors executing the user threads (the benchmarks). This can have a negative effect on the system performance.

In order to differentiate between the possible configurations of our platform, a naming convention similar to the one used in chapter 5 is presented in Table 6.1. Figures 6.1 is provided as a reference to help understand the different configurations. The system is identical to the system in chapter 5 with the expression of AMP to indicate that the system is operating as an AMP followed by its HT status, either on or off, followed by the number of threads used and finally the number of physical processors in use.

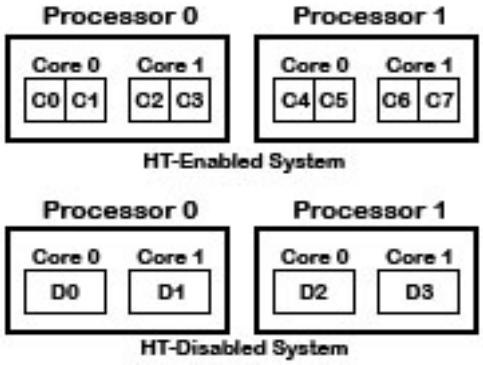


Figure 6.1: Processor numbering for 2-way dual-core system

Table 6.1: AMP Naming Convention

Terminology	*Hardware Contexts*	*Corresponding Architecture*
AMP-HT$_{on}$-1-1	C0 used for OS, C1	AMP-SMT
AMP-HT$_{off}$–1-1	D0 used for OS, D1	AMP-CMP
AMP-HT$_{on}$–3-1	C0 used for OS, C1, C2, C3	AMP-CMT
AMP-HT$_{off}$–1-2	D0 used for OS, D2	AMP-SMP
AMP-HT$_{on}$ –3-2	C0 used for OS, C1, C4, C5	AMP-SMT-based SMP
AMP-HT$_{off}$–3-2	D0 used for OS, D1, D2, D3	AMP-CMP-based SMP
AMP-HT$_{on}$ –7-2	C0 used for OS, C1, C2, C3, C4, C5, C6, C7, C8	AMP-CMT-based SMP

6.2 PS-Scheduler vs. the Default Linux Scheduler

The main motivation behind the proposed scheduler is to sustain an SMP's performance with the original O(1) scheduler, while providing energy savings. Our intention in this section is to see if the new scheduler performs on par with the default scheduler in both HT-enabled and HT-disabled configurations. The configurations in this section have all processors running at full clock speed, so they use the naming convention introduced in chapter 5 instead of that in section 6.1.

Figure 6.2 compares the baseline performance of the *PS-Scheduler* with the default scheduler for the SPEC benchmarks for both HT-disabled and HT-enabled systems. The performance of the *PS-Scheduler* compared to the default scheduler is promising, with almost all configurations and benchmarks seeing an improvement in performance. The HT_{on}-3-1 and HT_{on}-3-2 configurations show the best speedup of the configurations at an average of 14.4% and 16.2% respectively. The HT_{on}-1-1 configuration shows the worst speedup, with a slowdown of 11.8%. Only the HT_{on}-1-1 and HT_{off}-3-2 configurations show a decrease in performance with the new scheduler.

Overall, the performance gain of the *PS-Scheduler* of the HT-enabled configurations with the original scheduler ranges from –11.8% to +16.2%, with an average performance gain of 5.3%. For the HT-disabled case, the performance gain ranges from –9.1% to +4.4%, with an average performance gain of –0.9%.

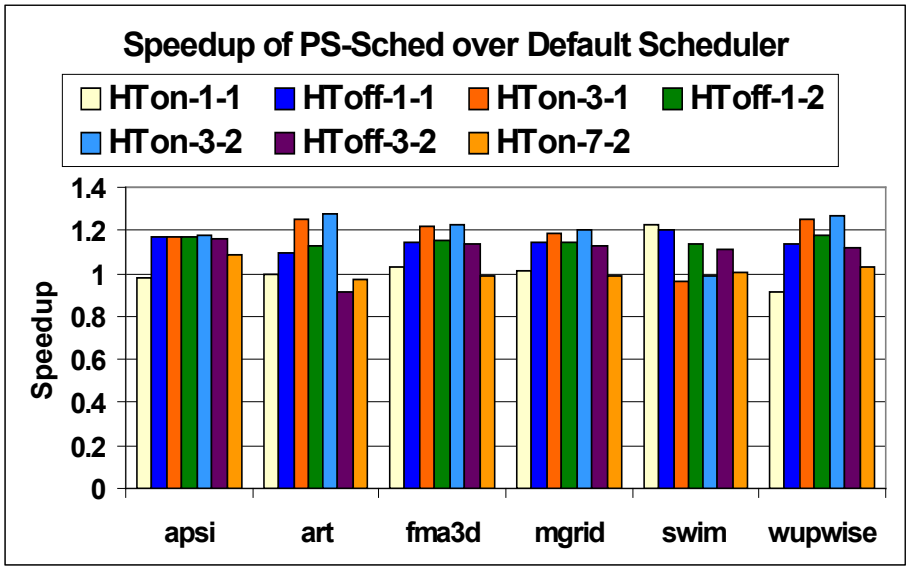


Figure 6.2: PS-Scheduler vs. default scheduler for SPEC benchmarks

6.3 Real Power Measurements

An experiment was setup in order to determine the actual power consumption of a real system utilizing the different schedulers. A Dell PowerEdge 2850 Dual-Core 2.8GHz SMP was used for the testing, whose details are described in section 5.1. The system was connected to a Keithley 2701 Digital Multimeter [49] with a Keithley 7700 data acquisition unit [48] installed. The overall power consumption of the system was measured using a resistive element attached to the power cable leading into the system. Due to the nature of the rack-mounted system, power measurements on the output of the power supply inside the machine were impractical. Therefore, the power consumption numbers presented here are affected by the power supply losses and take into account all components of the system fed by the main power supply.

The SPEComp benchmark suite was used for real-world power measurement due to its longer run times, which enabled us to obtain consistent stable power measurements. The power measurement equipment was validated through the use of a Wattsup EPS Pro power meter [23], and found to be within the acceptable error range of the two devices, with the Keithley meter having an error range of +/-1%.

6.3.1 Average Power Consumption

The average instantaneous power measurements for each of the configurations at the varying operating frequencies are presented in Figure 6.3 and Figure 6.4. The power consumption of the system using the new scheduler is surprising. In many cases, the new scheduler has higher instantaneous power consumption than the original scheduler. The highest average instantaneous power consumption of either scheduler occurs with the *PS-Scheduler* with a high of 337.4 W for the AMP-HT_{on}-7-2 configuration running mgrid at 2.8GHz. This compares to the maximum value for the default scheduler of 320.9 W for the HT_{on}-8-2 configuration running mgrid at 2.8GHz. However, the average power consumption for all of the operating frequencies and configurations across all of the applications is lower for the *PS-Scheduler* at 262.4 W versus the default scheduler's average of 272.1 W. Given that the *PS-Scheduler* has faster runtimes than the default

scheduler, the higher instantaneous power readings do not necessarily translate into worse overall energy efficiency, as will be discussed later in this chapter.

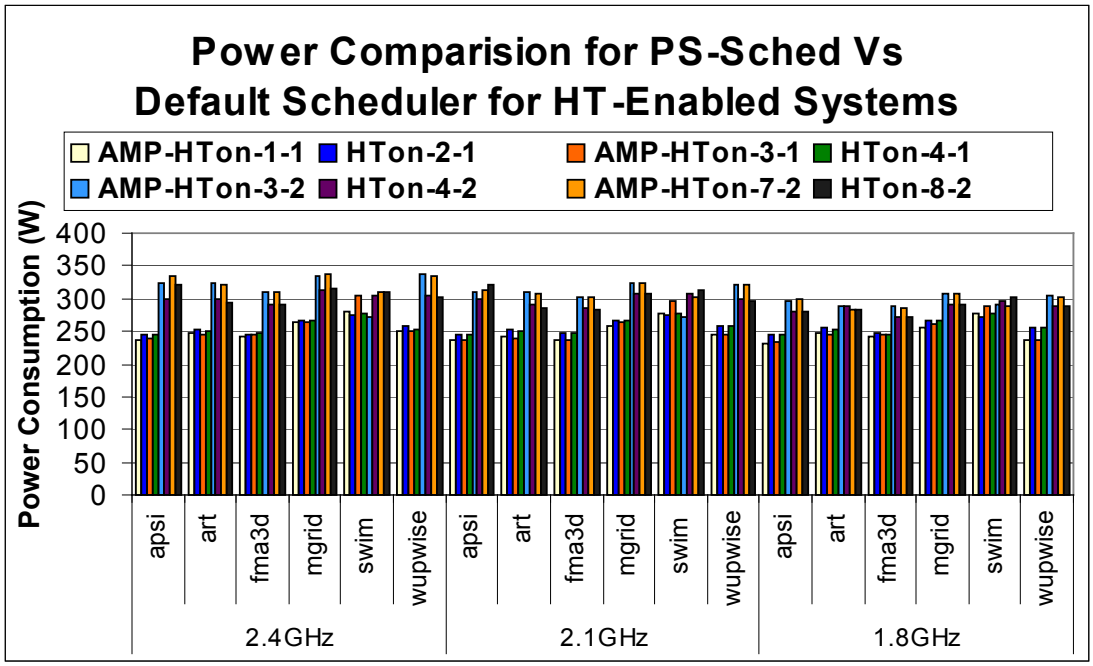


Figure 6.3: Average power consumption of HT-enabled configurations

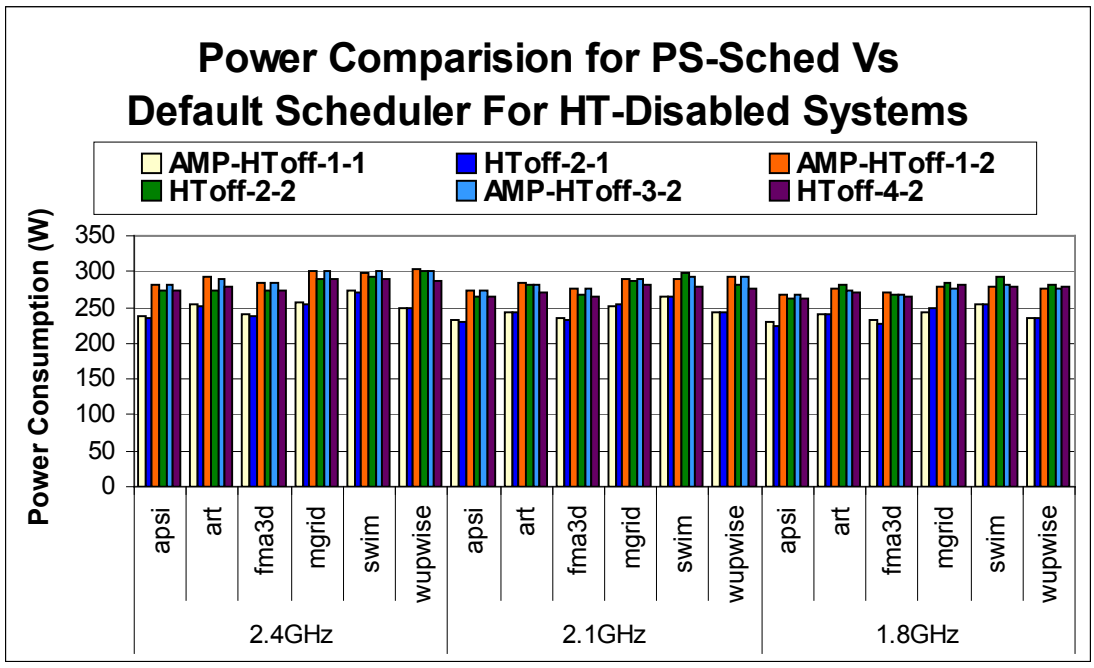


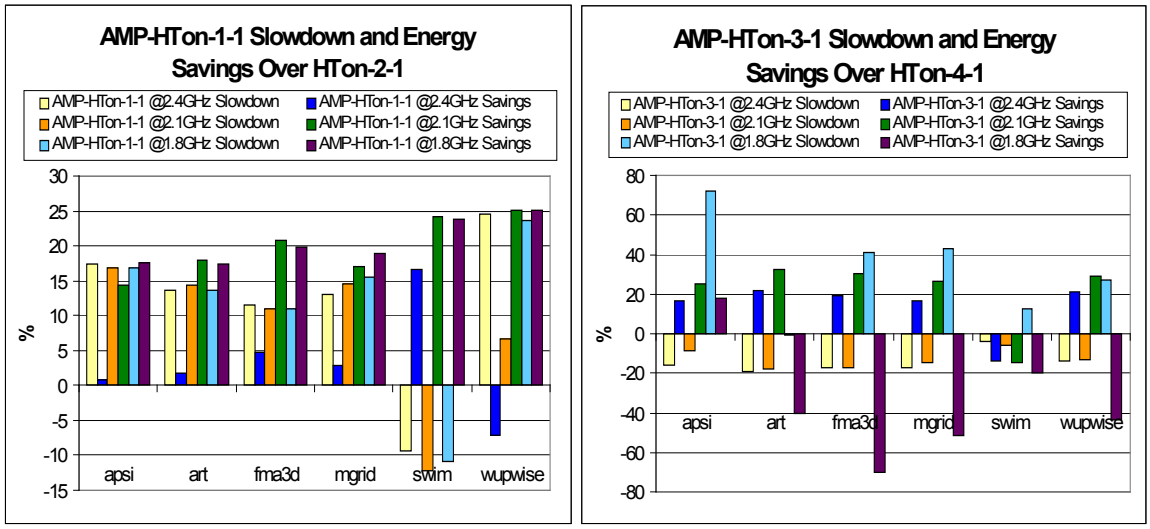
Figure 6.4: Average power consumption of HT-disabled configurations

6.3.2 Slowdown and Energy Savings

This section presents the actual slowdown in wall clock time that occurs when using the *PS-Scheduler* and the corresponding energy savings that occur. The AMP frequency

in the figures in this section correspond to the CPU frequency of the first physical processor in the system. The remaining physical processors are running at maximum frequency. In the case of the HT-enabled processors, it should be noted that the first two logical processors are scaled in frequency. Therefore, one logical CPU that is executing the user threads has a reduced frequency in addition to the reserved CPU.

The results presented in Figure 6.5(a) show that the AMP-HT$_{on}$-1-1 configuration on average does not perform well, with significant slowdowns occurring that yield almost a 1:1 relationship between slowdown and energy savings, with the exception of the swim benchmark which sees both a speedup and a reduction in corresponding energy usage. This is contrasted by the AMP-HT$_{on}$-3-1 results, where speedup occurs for both the 2.4 GHz and 2.1 GHz operating points, providing energy savings of between 18 to 32% while simultaneously reducing execution time for all applications with the exception of the swim benchmark.

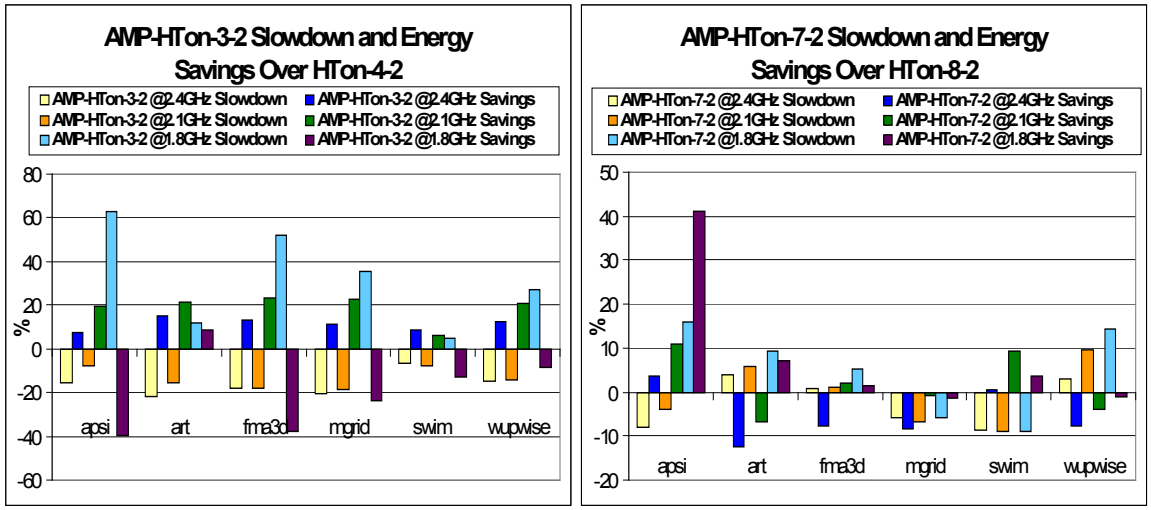


(a) (b)

Figure 6.5: Slowdown and energy savings for (a) AMP-HT$_{on}$-1-1 and (b) AMP-HT$_{on}$-3-1

The results for the AMP-HT$_{on}$-3-2 configuration presented in Figure 6.6 are similar in pattern to those of the AMP-HT$_{on}$-3-1 configuration with speedup occurring for the applications between 2.4 GHz and 2.1 GHz and good resultant energy savings, with the exception of the swim benchmark, which sees some marginal improvement for the first two operating frequencies. The AMP-HT$_{on}$-3-2 configuration does lag behind the AMP-HT$_{on}$-3-1 configuration in total speedup and as a result shows lower potential energy savings. The AMP-HT$_{on}$-7-2 configuration shows good results for the apsi benchmark,

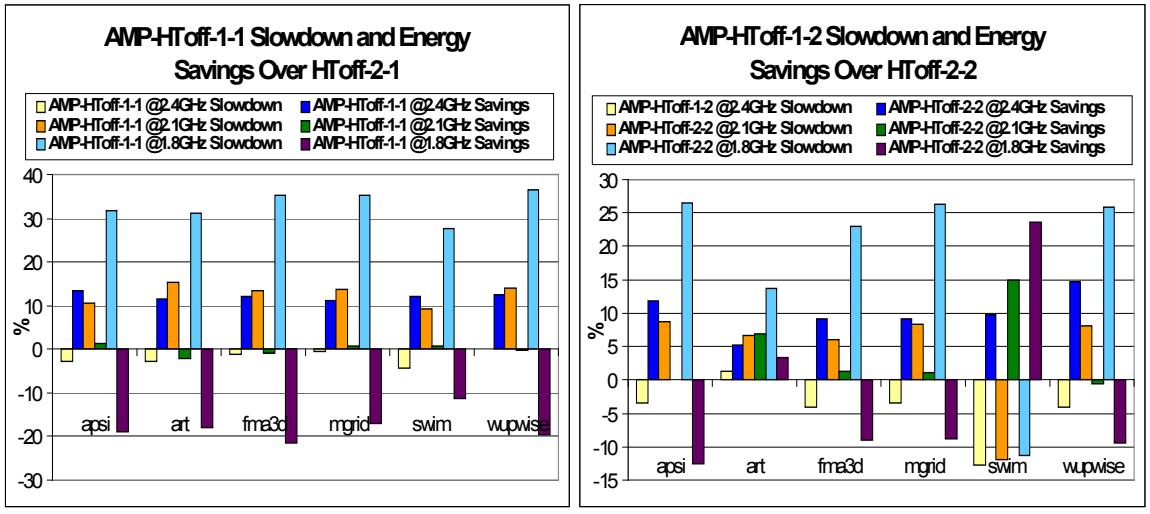
but the remaining benchmarks show a combination of slowdown and poor energy savings. The apsi, mgrid and swim benchmarks benefit from the *PS-Scheduler* in terms of runtime, but mgrid and swim fall behind in terms of energy consumption. The remainder of the applications suffer from both slowdown and increased energy consumption.



(a) (b)

Figure 6.6: Slowdown and Energy Savings for (a) AMP-HT$_{on}$-3-2 and (b) AMP-HT$_{on}$-7-2

The results for slowdown and energy savings for the HT-disabled architectures in Figure 6.7, show that the AMP-HT$_{off}$-1-1 and AMP-HT$_{off}$-1-2 configurations show good slowdown/savings for the 2.4 GHz operating point. With the exception of swim for the AMP-HT$_{off}$-1-2 configuration, the 2.1 GHz and 1.8 GHz operating points see too much slowdown to be able to realize any significant energy savings.



(a) (b)

Figure 6.7: Slowdown and energy savings for (a) AMP-HT$_{off}$-1-1 and (b) AMP-HT$_{off}$-1-2

The final architecture that was examined, AMP-HT_{off}-3-2, has its slowdown and energy savings illustrated in Figure 6.8. The results are varied, with apsi and mgrid showing excellent results, while art exhibits terrible slowdown and consequently poor energy savings numbers. The fma3d and wupwise applications show some energy savings are possible, but the resulting slowdown is slightly greater than the potential energy savings. The swim benchmark shows some speedup, and some potential for energy savings as well.

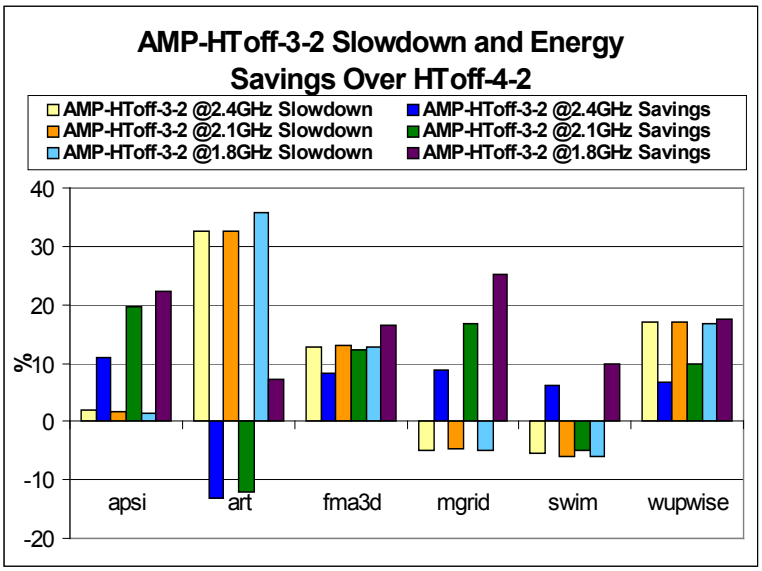


Figure 6.8: Slowdown and energy savings for AMP-HT_{off}-3-2

6.3.3 Energy-Delay Analysis

To compare the schedulers fairly one must also take into account the speed of each scheduler in addition to the energy savings that can be obtained. To this end, the energy-delay analysis is presented in Figures 6.9 to 6.14 for the *PS-Scheduler* normalized to the default scheduler. Energy-delay is a metric used to determine whether a trade-off between energy usage and the delay that is causes is beneficial or not. Energy-delay products of less than 1 indicate a beneficial trade-off.

The energy-delay of the HT_{on}-1-1 configuration is presented in Figure 6.9. We can observe that the energy-delay of the system operating at 2.4GHz is mediocre. However, the efficiency at the 2.1GHz and 1.8GHz operating points is excellent with all of the applications having energy delays below 1. Of course, this represents the significant energy savings that a single processor has over the baseline system with four physical

processors. As such the HT_{on}-1-1 configuration has good energy-delay products but is not very useful due to the loss of performance that it incurs.

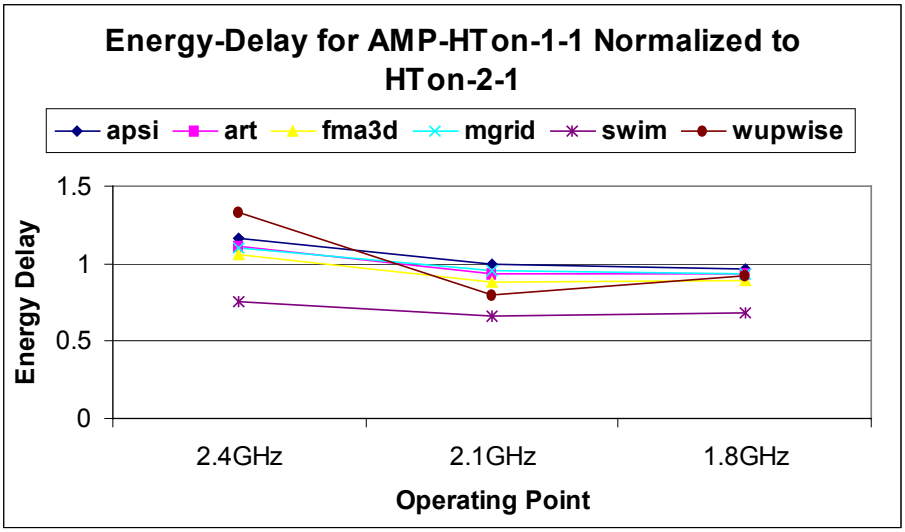


Figure 6.9: Energy delay for the PS-Scheduler in an HT_{on}-1-1 configuration

The energy delay figures for AMP-HT_{on}-3-1 configuration in Figure 6.10 are very promising. The AMP-HT_{on}-3-1 configuration has a majority of the applications with energy delays of less than 1 for both the 2.4 GHz and 2.1 GHz operating points, but shows a significant increase in energy delay for the 1.8 GHz operating point. Overall, its average energy delay for the 2.4GHz operating point is 0.75.

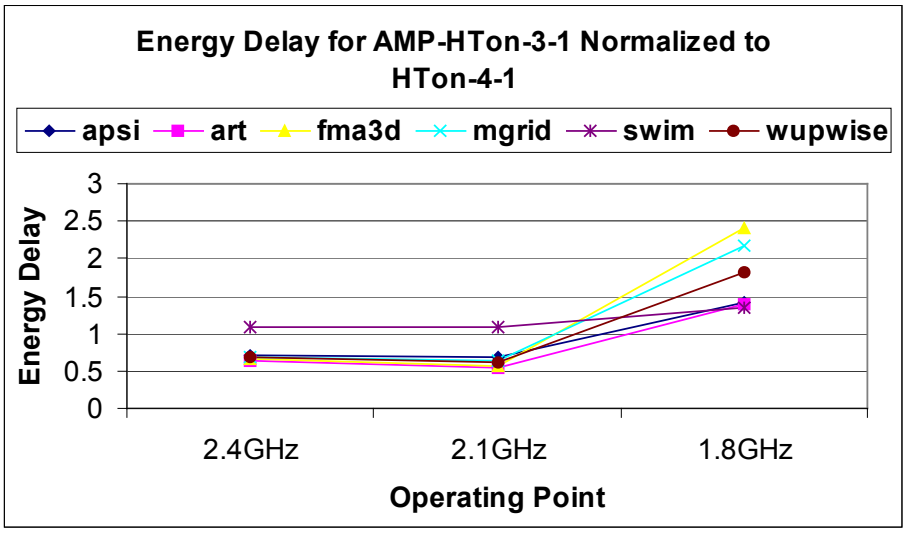


Figure 6.10: Energy delay for the PS-Scheduler in an HT_{on}-3-1 configuration

The AMP-HT_{on}-3-2 configuration's energy-delay products are illustrated in Figure 6.11. The AMP-HT_{on}-3-2 configuration is the best of all of the configurations, with none

of the applications having an energy-delay of above 1 for the first two operating points, and energy delays within the 0.6-0.8 range. In addition, its performance is excellent, giving it both good energy consumption and performance.

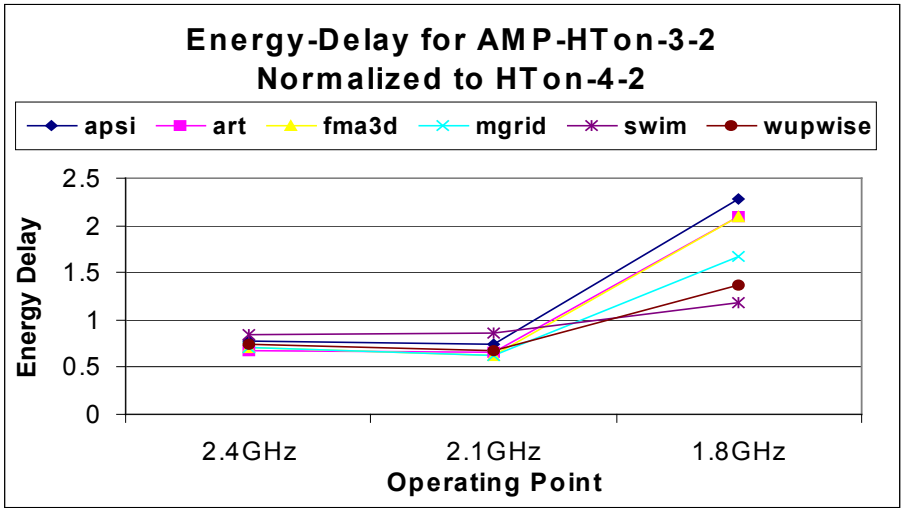


Figure 6.11: Energy-delay for the PS-Scheduler in an HT_{on}-3-2 configuration

For the AMP-HT_{on}-7-2 configuration presented in Figure 6.12, the majority of applications do not see an energy-delay of less than one until the operating point is lowered to 2.1GHz or lower. Half of the applications see a benefit to using the *PS-Scheduler*. The AMP-HTon-7-2 configuration with the *PS-Scheduler* can be beneficial but it is dependant on the type of applications which are being run.

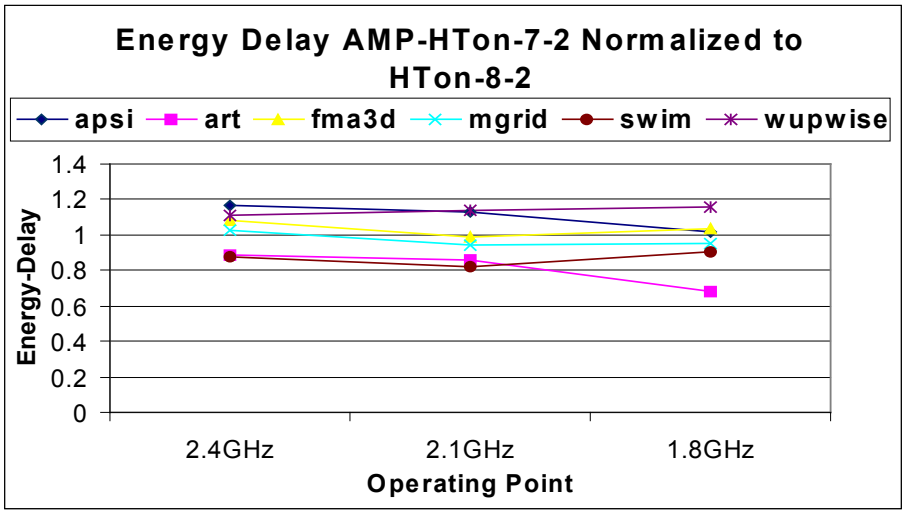


Figure 6.12: Energy-delay for the PS-Scheduler in an AMP-HT_{on}-7-2 configuration

The AMP-HT_{off}-1-1 configuration in Figure 6.13 shows that an energy delay of less than one can be achieved for the upper operating points with the *PS-Scheduler*, for all

applications. This behaviour abruptly ceases when the operating point is lowered below 2.4GHz. All of the applications have energy-delay products greater than 1 for all of the operating points below 2.4GHz.

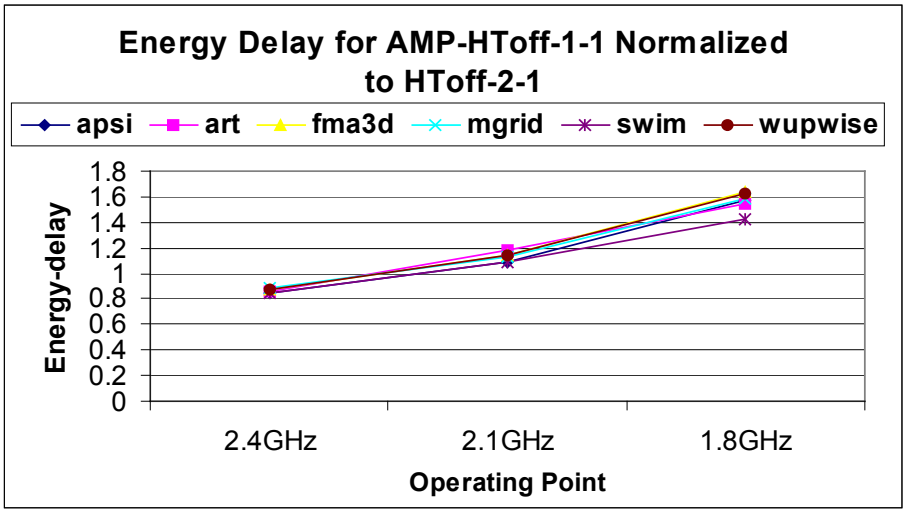


Figure 6.13: Energy-delay for the PS-Scheduler in an AMP-HT$_{off}$-1-1 configuration

The energy-delay products for the AMP-HT$_{off}$-1-2 configuration are shown in Figure 6.14. One can observe that the energy-delay products for the 2.4GHz operating point are all below 1, indicating that the *PS-Scheduler* can provide a significant increase in efficiency for all of the benchmarks in the suite. The efficacy of the *PS-Scheduler* technique quickly fades as the delay incurred at the lower operating frequencies outweighs any energy savings.

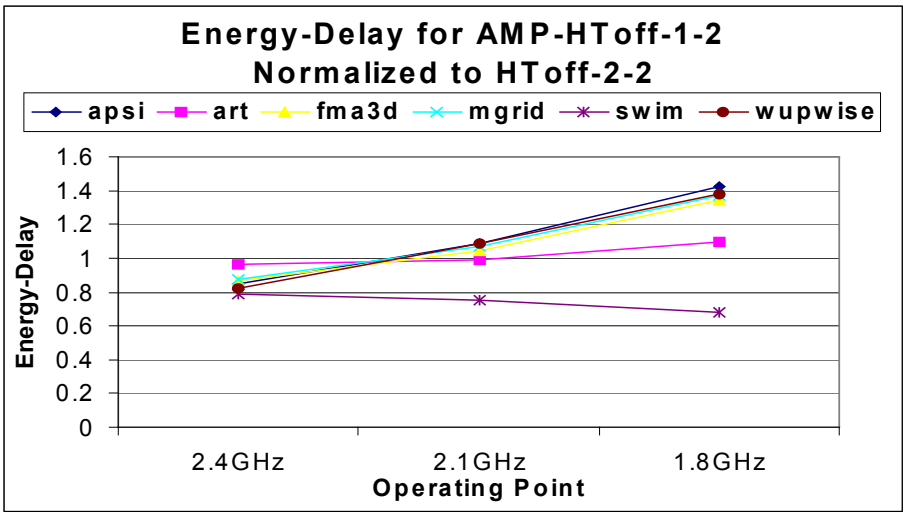


Figure 6.14: Energy-delay for the PS-Scheduler in an AMP-HT$_{off}$-2-1 configuration

We can see that the energy-delay of the *PS-Scheduler* is on the whole mediocre for the AMP-HT$_{off}$-3-2 case in Figure 6.15, with no significant overall savings. Although art,

mgrid and swim benefit from using the scheduler, the majority of applications do not. In fact, with the SPEC benchmarks, the system sees a general improvement of wall clock times but higher power consumption. This leads to poor energy-delay products as the increase in speed does not offset the increase in energy consumption.

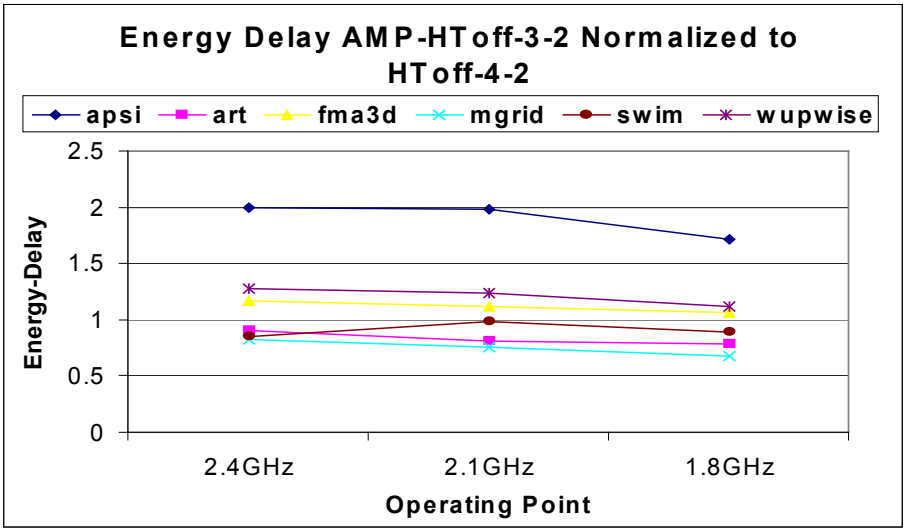


Figure 6.15: Energy-delay for the PS-Scheduler in an AMP-HT$_{off}$-3-2 configuration

6.4 AMP Power Consumption Predictions With Future Technology

Using a method similar to that in [3], we can estimate the effect that true frequency scaling would have on a dual-core system. The power consumption of a 2.8GHz dual-core Xeon processor is 135W when active and 32W when idle [42]. The authors in [3] provide a method based on historical data, where for a changing frequency, the power consumption is proportional to the square of the duty cycle. Therefore, if a 2-way 2.8GHz dual-core Intel Xeon processor system consumes 270W when highly active, the system at 2.4GHz would be expected to consume 198.3W ($270W \times (2.4/2.8)^2$) when highly active. This corresponds to an energy consumption of 49.575W per core. When CPUs are in an idle state the power consumption of the system is 64W, corresponding to 16W per core. Applying the same scaling as used for the active case, we determine that the idle energy consumption for a single core at 2.4GHz is 11.75W. Using the same methodology, one can easily find the highly active power consumption of an AMP with one CPU operating at 2.4GHz and the other three CPUs operating at 2.8GHz to be 252W. Knowing the approximate energy consumption of our AMP systems, when highly active

or idle, allows us to estimate the power consumption while executing the SPEC OpenMP benchmarks.

6.4.1 Slowdown and Energy Savings

The energy savings and slowdown due to frequency scaling of the system while executing the SPEC benchmarks are presented in Figures 6.16 to 6.18.

A prediction of the performance of the best configuration from the previous section is pertinent. Therefore, the results for the AMP-HT_{on}-3-2 configuration are presented in Figure 6.16. Every application demonstrates a performance increase for the 2.4GHz and 2.1GHz operating points. The resulting energy savings for the AMP-HT_{on}-3-2 configuration for the 2.4GHz and 2.1GHz operating points, on average, across all of the benchmarks are 27.6% and 27.8% respectively. This corresponds to an average speedup of 16.2% and 13.7% for the 2.4GHz and 2.1GHZ operating points. The 1.8GHz operating point sees a slowdown of 32.4% and an energy loss of 5.33%.

When comparing the frequency scaling estimates with the previously measured results in figure 6.6(a), we find that the energy savings using frequency scaling improve the results from 11.38%, 19.0% and –19.0% for the 2.4GHz, 2.1GHz and 1.8GHz operating points to 27.6%, 27.8% and –5.33%. Obviously, the use of true frequency scaling should significantly improve the effectiveness of the *PS-Scheduler* for this configuration.

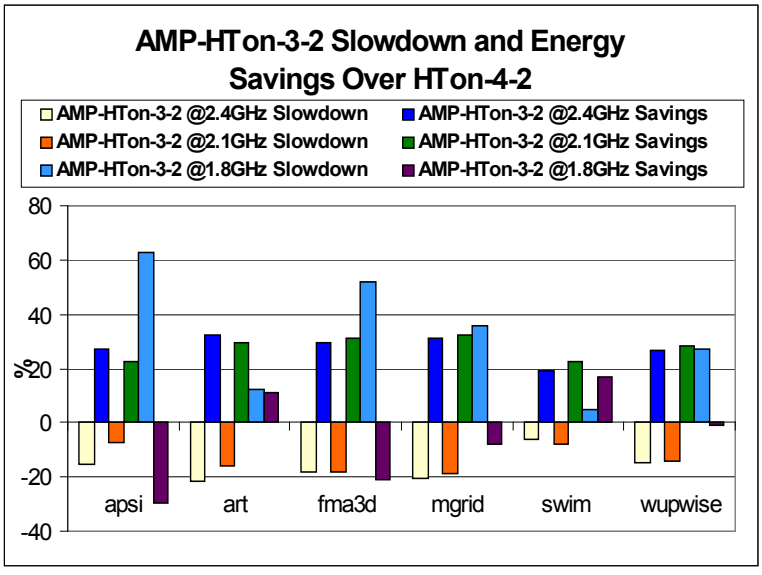


Figure 6.16: AMP slowdown and energy savings for SPEC benchmarks over HT_{on}-4-2

In the HT_{on}-7-2 case in Figure 6.17, we can observe that frequency scaling would improve the energy savings of the system by 21.3% on average over the *PS-Scheduler* results that were measured in Figure 6.6(b).

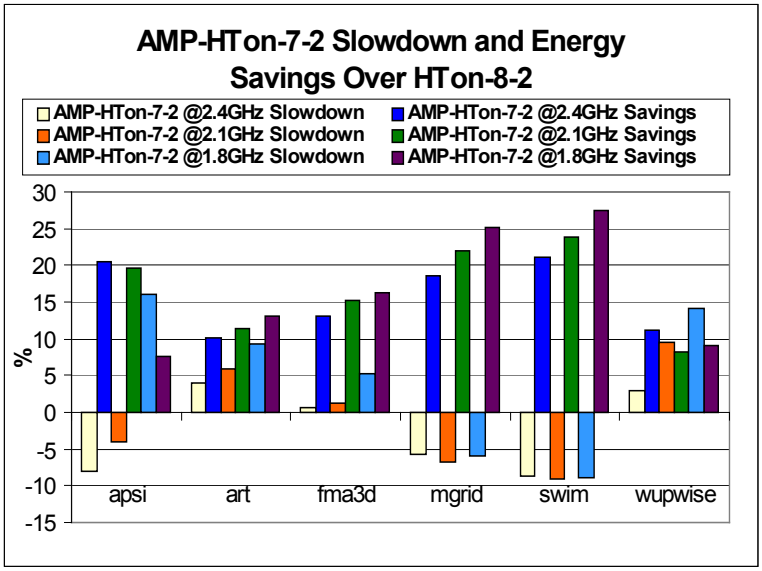


Figure 6.17: AMP slowdown and energy savings for SPEC benchmarks over HT_{on}-8-2

Overall, the average range of energy savings of the AMP-HT_{on}-7-2 configuration for the benchmarks is +8.13% to +27.5%. Swim, mgrid and apsi are the applications that benefit the most from the new scheduler, seeing an improvement in both performance and power consumption. However, all of the applications see an improvement in power consumption.

The slowdown and energy savings of the AMP-HT_{off}-3-2 system are presented in Figure 6.18. The apsi, mgird and swim benchmarks see the greatest benefit from the *PS-Scheduler* with all three benchmarks experiencing a decrease in runtime as well as energy savings. The swim benchmark exhibits the best behaviour showing an average speedup of 5.74% and energy savings of 26.5%. The swim benchmark is a memory intensive benchmark [33], and is well known for its poor scalability. Therefore, the energy savings can be attributed to two factors, the reduction of system noise and a smaller number of overall execution threads.

Overall, the AMP-HT_{off}-3-2 configuration has an average energy savings of 14.9% across all three frequencies and all of the benchmarks. It has an average slowdown of 9.1%. A comparison to the previous results in Figure 6.8 show that energy savings

improve by 8.84%, 7.79% and 0.26% for the 2.4GHz, 2.1GHz and 1.8GHz operating points respectively for a true frequency scaling system.

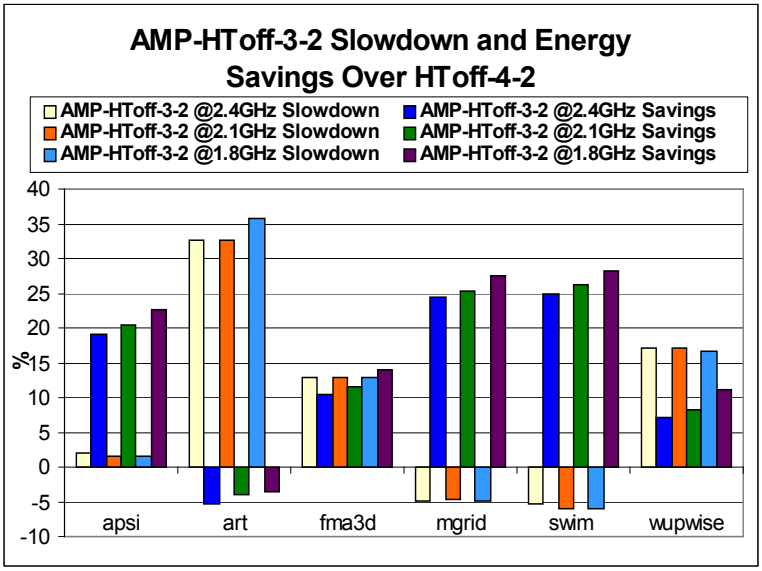


Figure 6.18: AMP slowdown and energy savings for SPEC benchmarks over HT_{off}-4-2

6.4.2 Energy-Delay Analysis

First, we examine the predicted energy-delay product of the most efficient configuration from section 6.3. The energy-delay products for the AMP-HT_{on}-3-2 configuration are presented in Figure 6.19.

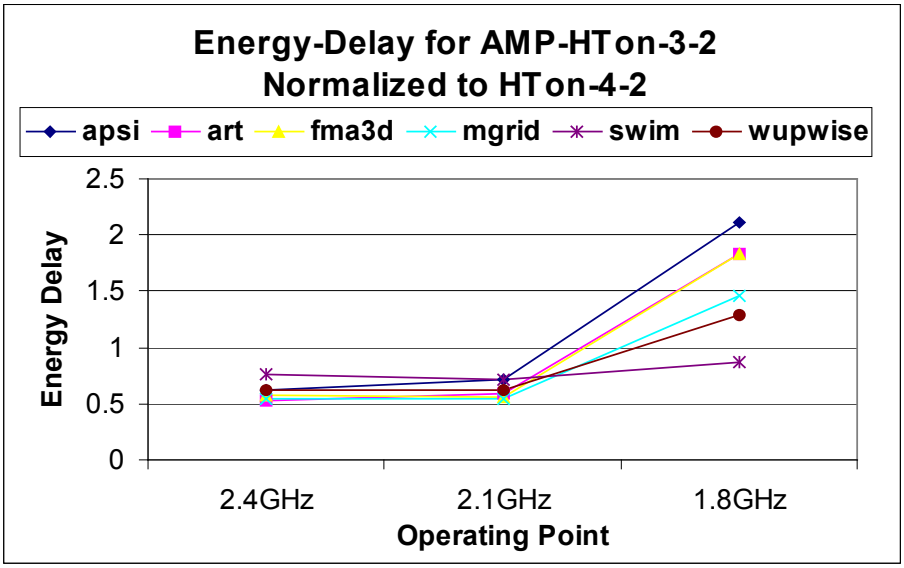


Figure 6.19: Normalized Energy-Delay for SPEC benchmarks for AMP-HT_{on}-3-2 over HT_{on}-4-2

When Figure 6.19 is compared with Figure 6.11 we can see that the energy-delay products of the predictive case are reduced. This leads to a new average energy-delay of 0.61 for the 2.4GHz operating point over the measured average energy-delay of 0.74. Overall, the performance of the PS-Scheduler with a system configured as an AMP-HT_{on}-3-2 is excellent and the energy savings are substantial.

Figure 6.20 presents the energy-delay of the AMP running the *PS-Scheduler* normalized to the original scheduler for the HT_{on}-8-4 case. An energy-delay of less than one shows that the savings in energy consumption outpace the corresponding increase in execution speed. Comparing Figure 6.20 with Figure 6.12 we can observe that energy-delays have dropped by approximately 0.2 for the applications. The average energy-delay for the 2.4GHz operating point has dropped from 1.0 to 0.82. Both the 2.4GHz and 2.1GHz operating points now have all of the SPEC applications with energy-delays of below 1. This indicates that the future applications of the *PS-Scheduler* are expected to improve as technology advances.

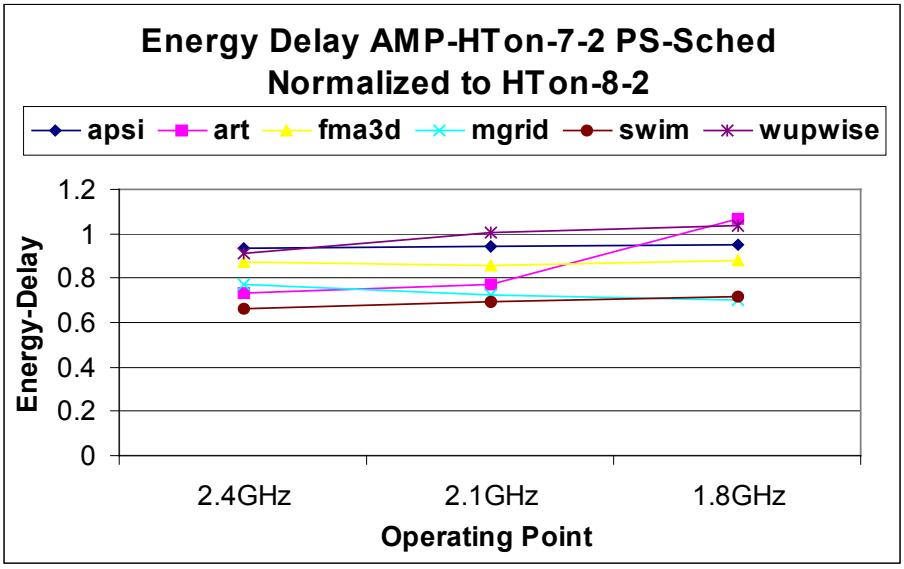


Figure 6.20: Normalized Energy-Delay for SPEC benchmarks for AMP-HT_{on}-7-2 over HT_{on}-8-2

The results of the energy-delay analysis of the AMP-HT_{off}-3-2 configuration are presented in Figure 6.21. When comparing Figure 6.21 with Figure 6.15 we find that the energy-delays have improved but a number of applications still have energy delay products significantly higher than 1. However, art, mgrid and swim have shown a further improvement to their efficiency with the *PS-Scheduler* and receive a significant benefit

from utilizing it. From this we can conclude that the *PS-Scheduler* has a potential for use on non-HT systems for some applications.

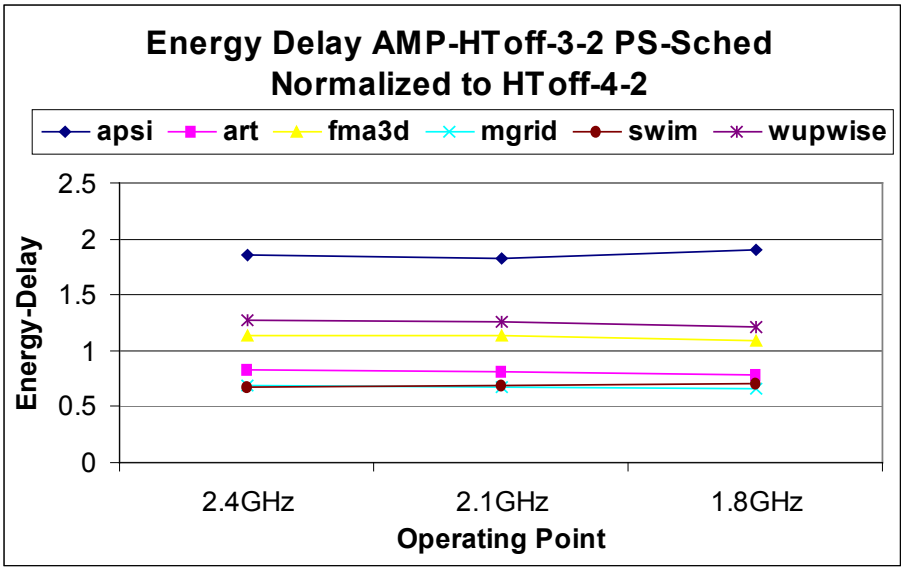


Figure 6.21: Normalized Energy-Delay for SPEC benchmarks for AMP-HT$_{off}$-3-2 over HT$_{off}$-4-2

6.4 Summary

In this chapter, the lessons learned from the previous chapters were used to design a new operating system task scheduler in an attempt to increase system performance and save energy.

The power consumption of a real dual-core system was measured. It was found that performance improvements could be made using the new scheduler on dual-core systems. In addition, total consumed energy could be reduced despite a higher instantaneous power usage using the new scheduler. The best platforms for such a scheduler were determined to be the AMP-HT$_{on}$-3-1, AMP-HT$_{on}$-3-2 and AMP-HT$_{off}$-2-2 architectures.

The modified scheduler was then tested and its energy usage was estimated on several different computing platforms to determine the power consumption of an identical system with true frequency scaling. These results indicated that the new scheduler could achieve good energy conservation on systems with very little performance impact.

The new scheduler does a good job at reducing the energy consumption of AMPs running the SPEComp applications while having a minimal impact on the performance of the system. The performance results using real power measurements indicate on average

15.6% energy savings and 6.1% slowdown for the HT-disabled case, and 7.1% energy savings and 4.8% slowdown for the HT-enabled case across all applications studied in this chapter.

Chapter 7: Conclusions and Future Work

This thesis has explored the behaviour of single-core and dual-core symmetric multiprocessor systems using the OpenMP interface and the effect of Intel's Hyper-Threading technology. The overhead incurred by using the OpenMP API was examined and it was determined that the 2.6.9 Linux kernel causes more OpenMP overhead than the 2.4.22 kernel for SMT architectures. The behaviour of single-core architectures, in relation to their performance executing well-known high profile scientific benchmarking suites, has been examined in detail. It was found that the memory system and CPU caches are the performance bottlenecks of the systems. The trace cache performance was found to be a source of performance degradation as well.

This thesis has performed an in depth exploration of the performance characteristics of multi-core processors equipped with SMT capabilities and investigated the effect of overloaded workloads on system performance. From this investigation, it seems clear that the optimal platform for high performance computing on such systems lies in utilizing the SMT features available in the dual-core architecture. This thesis has demonstrated that the performance of SMT technologies in multi-core processor designs can perform as well or better than SMP or CMP architectures by showing that a single multi-core processor with SMT capabilities can closely match the performance of a SMP or CMP machine with twice as many processors. Therefore we can conclude that HT technology can be of use in the HPC domain, and in fact can provide tangible efficiency benefits over the non-HT alternatives when used in architectures that are composed of a single dual-core CPU or two single-core CPUs. However, the benefits of utilizing HT have been negligible when the number of logical cores in the system rises above four. By operating the systems with workloads creating two times as many threads as the number of available execution contexts, it has been shown that significant performance improvements can be achieved for some applications.

The effect of operating system noise on such systems was explored, and found to be a potential area of improvement for such SMP/SMT systems. These findings initiated research into methods of reducing operating system noise, and optimizing systems to increase performance as well as investigating the potential power savings that could be

realized using such schemes. In addition, the possible power consumption benefit of SMT technologies has been explored, particularly as it relates to AMPs. It has been shown that by reserving a CPU in a multi-processor system, both performance gains and power savings are possible for real systems. This was demonstrated by measuring the power consumption of a real 2-way dual core system. From this experimental data, the power consumption of a true frequency scaling system was predicted, showing that the energy savings of the proposed scheduler could be increased significantly when such scaling techniques are available for Xeon processors.

Therefore, we can conclude that significant performance benefits can be realized, in addition to power savings, over the traditional SMP/CMP architectures. CMP/CMT hybrid technology can be leveraged to provide performance levels equivalent to those of systems with twice as many computational resources. CMT technology is a promising new architecture that when utilized correctly will be able to provide great benefit to the HPC scientific community in terms of both power and performance. Coupled with the use of the scheduling method described in this thesis, the effect that such architectures can have within the HPC community is a positive one. The increase in overall machine efficiency that can be achieved using such approaches offers the possibility of increased system throughput at a lower operating cost than previous generations of systems.

7.1 Future Work

In the future, this work can be improved upon by adapting the scheduler to take advantage of online performance monitor counter data. This would allow an AMP to dynamically adjust itself according to the current processor workload. In addition, using the power monitor data collection system developed for this thesis, it is technically possible to integrate a real time power consumption reporting system that can report the system's power consumption during system operation and this data can be used to further refine the efficiency of the scheduler. The online power system reporting could also be integrated into an API for use within the programs themselves to make them power aware. All of these future research options should further increase the efficiency of the scheduler developed in this thesis, making the technique more beneficial for executing scientific applications.

The *PS-Scheduler* could be extended to work with emerging quad-core processors and tested with a larger group of scientific applications. It could also be tested and developed for use with commercial applications, particularly in a data center context. The concepts behind the *PS-Scheduler* could also be applied to processor design, creating a slow low power processing core specifically to handle OS activity.

References

[1] Advanced Micro Devices. AMD athlon X2 dual core. Available: http://www.amd.com/us-en/Processors/ProductInformation/0,,30_118_9485_13041,00.html

[2] D. H. Albonesi, R. Balasubramonian, S. G. Dropsbo, S. Dwarkadas, F. G. Friedman, M. C. Huang, V. Kursun, G. Magklis, M. L. Scott, G. Semeraro, P. Bose, A. Buyuktosunoglu, P. W. Cook and S. E. Schuster, "Dynamically tuning processor resources with adaptive processing," *IEEE Computer,* vol. 36, pp. 49-58, 2003.

[3] M. Annavaram, E. Grochowski and J. Shen, "Mitigating amdahl's law through EPI throttling," in *ISCA '05: Proceedings of the 32nd Annual International Symposium on Computer Architecture,* 2005, pp. 298-309.

[4] C. D. Antonopoulos, D. S. Nikolopoulos and T. S. Papatheodorou, "Scheduling algorithms with bus bandwidth considerations for SMPs," in *ICPP '03: Proceedings of the International Conference on Parallel Processing,* 2003, pp. 547-554.

[5] V. Aslot and R. Eigenmann, "Performance characteristics of the SPEC OMP2001 benchmarks," in *EWOMP '01: Proceedings of the European Workshop on OpenMP,* 2001,

[6] V. Aslot, M. Domeika, R. Eigenmann, G. Gaertner, W. B. Jones and B. Parady, "SPEComp: A new benchmark suite for measuring parallel computer performance," in *WOMPAT '01: Proceedings of the Workshop on OpenMP Applications and Tools,* 2001, pp. 10.

[7] T. Austin, E. Larson and D. Ernst, "SimpleScalar: an infrastructure for computer system modeling," *IEEE Computer,* vol. 35, pp. 59-67, 2002.

[8] S. Balakrishnan, R. Rajwar, M. Upton and K. Lai, "The impact of performance asymmetry in emerging multicore architectures," in *ISCA '05: Proceedings of the 32nd Annual International Symposium on Computer Architecture,* 2005, pp. 506-517.

[9] R. Berrendorf and G. Nieken, "Performance characteristics for OpenMP constructs on different parallel computer architectures," *Concurrency Practice and Experience,* vol. 12, pp. 1261-1273, 2000.

[10] D. Brooks, V. Tiwari and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," in *ISCA '00: Proceedings of the 27th International Symposium on Computer Architecture,* 2000, pp. 83-94.

[11] S. Brosky. Shielded CPUs: Real-time performance in standard linux. Available: http://www.linuxjournal.com

[12] J. M. Bull, "Measuring synchronisation and scheduling overheads in OpenMP," in *EWOMP '99: Proceedings of First European Workshop on OpenMP,* 1999, pp. 99-105.

[13] D. Chandra, F. Guo, S. Kim and Y. Solihin, "Predicting inter-thread cache contention on a chip multi-processor architecture," in *HPCA '05: Proceedings of the 11th International Symposium on High-Performance Computer Architecture,* 2005, pp. 340-351.

[14] J. Chang and G. S. Sohi, "Cooperative Caching for Chip Multiprocessors," *IEEE Network,* vol. 1, pp. L1D,

[15] L. Chen, I. Fujishiro and K. Nakajima, "Parallel performance optimization of large-scale unstructured data visualization for the earth simulator," in *EGPGV '02: Proceedings of the Fourth Eurographics Workshop on Parallel Graphics and Visualization,* 2002, pp. 133-140.

[16] T. Constantinou, Y. Sazeides, P. Michaud, D. Fetis and A. Seznec, "Performance implications of single thread migration on a chip multi-core," *ACM SIGARCH Computer Architecture News,* vol. 33, pp. 80-91, 2005.

[17] R. Couturier and C. Chipot, "Parallel molecular dynamics using OpenMP on a shared memory machine," *Comput. Phys. Commun.,* vol. 124, pp. 49-59, Jan. 2000.

[18] M. Curtis-Maury, J. Dzierwa, D. Antonopoulos and D. S. Nikolopoulos, "Online strategies for high-performance power-aware thread execution on emerging multiprocessors," in *HP-PAC '06: 2nd Worksop on High-Performance, Power-Aware Computing in the Proceedings of the 21st International Parallel and Distributed Processing Symposium,* 2006,

[19] M. Curtis-Maury, X. Ding, C. D. Antonopoulos and D. S. Nikolopoulos, "An evaluation of OpenMP on current and emerging Multithreaded/Multicore processors," in *IWOMP '05: Proceedings of the International Workshop on OpenMP,* 2005,

[20] L. Dagum and R. Menon, "OpenMP: an industry standard API for shared-memory programming," *IEEE Computational Science and Engineering,* vol. 5, pp. 46-55, 1998.

[21] M. De Vuyst, R. Kumar and D. M. Tullsen, "Exploiting unbalanced thread scheduling for energy and performance on a CMP of SMT processors," in *IPDPS '06: Proceedings of the 20th International Parallel and Distributed Processing Symposium,* 2006, pp. 10.

[22] M. J. DeLuca and M. A. Rivas, "Computing system with selective operating voltage and bus speed," U.S.A. 5086501, 1992, 1989.

[23] Electronic Educational Devices. (2007, Jan.). Wattsup pro EPS power meter. *2007(May 1),* pp. 2. Available: https://www.doubleed.com/watts_up__es.pdf

[24] A. El-Moursy, R. Garg, D. H. Albonesi and S. Dwarkadas, "Compatible phase co-scheduling on a CMP of multi-threaded processors," in *IPDPS '06: Proceedings of the 20th International Parallel and Distributed Processing Symposium,* 2006, pp. 10.

[25] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner and T. Mudge, "Razor: A low-power pipeline based on circuit-level timing speculation," in *MICRO 36: Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture,* 2003, pp. 7.

[26] A. Fedorova, M. Seltzer, C. Small and D. Nussbaum, "Throughput-Oriented Scheduling On Chip Multithreading Systems," *Technical Report TR-17-04, Harvard University, August,* 2004.

[27] A. Fedorova, C. Small, D. Nussbaum and M. Seltzer, "Chip multithreading systems need a new operating system scheduler," in *EW '04: Proceedings of the 11th Workshop on ACM SIGOPS European Workshop: Beyond the PC,* 2004, pp. 9.

[28] N. R. Fredrickson, A. Afsahi and Y. Qian, "Performance characteristics of openMP constructs, and application benchmarks on a large symmetric multiprocessor," in *ICS '03: Proceedings of the 17th Annual International Conference on Supercomputing,* 2003, pp. 140-149.

[29] V. W. Freeh and D. K. Lowenthal, "Using multiple energy gears in MPI programs on a power-scalable cluster," in *PPoPP '05: Proceedings of the Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming,* 2005, pp. 164-173.

[30] R. Ge, X. Feng and K. W. Cameron, "Improvement of power-performance efficiency for high-end computing," in *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium,* 2005, pp. 8.

[31] R. E. Grant and A. Afsahi, "A comprehensive analysis of multithreaded OpenMP applications on dual-core intel xeon SMPs," in *MTAAP '07: Workshop on Multithreaded Architectures and Applications in the Proceedings of the 21St International Parallel and Distributed Processing Symposium (IPDPS 2007),* 2007, pp. 365.

[32] R. E. Grant and A. Afsahi. Power-performance efficiency of asymmetric multiprocessors for multi-threaded scientific applications. Presented at *HPPAC '06: 2nd Workshop on High-Performance, Power-Aware Computing* in the Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS 2006).

[33] R. E. Grant and A. Afsahi, "Characterization of multi-threaded scientific workloads on simultaneous multithreading intel processors," in *IOSCA '05: Proceedings of the Workshop on Interaction between Operating System and Computer Architecture,* 2005, pp. 13-19.

[34] R. E. Grant and A. Afsahi, "Improving Power and Performance of Chip-Multiprocessors by Exploiting the Effects of Operating System Noise," To be submitted.

[35] L. Hammond, B. A. Nayfeh and K. Olukotun, "A Single-Chip Multiprocessor," *IEEE Computer,* vol. 30, pp. 79-85, 1997.

[36] C. Hsu and U. Kremer, "The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction," in *PLDI '03: Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation,* 2003, pp. 38-48.

[37] Z. Hu, S. Kaxiras and M. Martonosi, "Let caches decay: reducing leakage energy via exploitation of cache generational behavior," *ACM Transactions on Computer Systems,* vol. 20, pp. 161-190, 2002.

[38] IEEE, "Information Technology-Portable Operating System Interface (POSIX) Standard," vol. IEEE Std. 1003.1: 2001, 2001.

[39] Intel Corp. (2006, Intel xeon processor. [http://www.intel.com/products/processor/xeon/index.htm]. *2006*Available: http://www.intel.com/products/processor/xeon/index.htm

[40] Intel Corp. Intel core 2 processor family. Available: http://www.intel.com/products/processor/core2/index.htm

[41] Intel Corp. (2007, Intel VTune performance analyzer. Available: http://www.intel.com/software/products/vtune

[42] Intel Corp. (2005, Oct.). Dual-core intel xeon processor 2.8GHz datasheet. [[online]]. Available: http://download.intel.com/design/Xeon/datashts/30915801.pdf

[43] ITRS. International technology roadmap for silicon organization. Available: www.itrs.net

[44] H. Jin, M. Frumkin and J. Yan. (1999, Oct.). The OpenMP implementation of NAS parallel benchmarks and its performance. NASA Ames Research Center, U.S.A.

[45] T. Jones, S. Dawson, R. Neely, W. Tuel, L. Brenner, J. Fier, R. Blackmore, P. Caffrey, B. Maskell, P. Tomlinson and M. Roberts, "Improving the scalability of parallel jobs by adding parallel awareness to the operating system," in *SC '03: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing,* 2003, pp. 10.

[46] R. Kalla, B. Sinharoy and J. M. Tendler, "IBM Power5 chip: a dual-core multithreaded processor," *IEEE Micro,* vol. 24, pp. 40-47, 2004.

[47] M. Kandemir, N. Vijaykrishnan, M. J. Irwin and W. Ye, "Influence of compiler optimizations on system power," in *DAC '00: Proceedings of the 37th Design Automation Conference,* 2000, pp. 304-307.

[48] Keithley Instruments Inc. (2007, Digital multimeters and data Acquisition/Switching systems - 7700. *2007(April 20, 2007),* Available: http://www.keithley.com/products/dmm/?mn=7700

[49] Keithley Instruments Inc. (2007, Digital multimeters and data Acquisition/Switching systems - 2701. *2007(April 20, 2007),* Available: http://www.keithley.com/products/dmm/?mn=2701

[50] J. G. Koomey. (2005, Jan.). Estimating total power consumption by servers in the U.S. and the world. Available: http://enterprise.amd.com/Downloads/svrpwrusecompletefinal.pdf

[51] R. Kotla, S. Ghiasi, T. Keller and F. Rawson, "Scheduling processor voltage and frequency in server and cluster systems," in *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium,* 2005, pp. 8.

[52] R. Kumar, K. Farkas, N. Jouppi, P. Ranganathan and D. M. Tullsen, "Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction," in *MICRO '36: Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture,* 2003, pp. 81.

[53] R. Kumar, D. M. Tullsen, N. P. Jouppi and P. Ranganathan, "Heterogeneous chip multiprocessors," *IEEE Computer,* vol. 38, pp. 32-38, 2005.

[54] Lawrence Livermore National Laboratory. LLNL OpenMP benchmarks. Available: www.llnl.gov/CASC/RTS Report/openmp perf.html

[55] T. Leng, R. Ali, J. Hsieh, V. Mashayekhi and R. Rooholamini, "An empirical study of hyper-threading in high performance computing clusters," in *The Proceedings of the Third LCI International Conference on Linux Clusters: The HPC Revolution,* 2002,

[56] Y. Li, K. Skadron, D. Brooks and Z. Hu, "Performance, energy, and thermal considerations for SMT and CMP architectures," in *HPCA '05: Proceedings of the 11th International Symposium on High-Performance Computer Architecture,* 2005, pp. 71-82.

[57] C. Liao, Z. Liu, L. Huang and B. Chapman, "Evaluating OpenMP on chip MultiThreading platforms," in *IWOMP '05: Proceedings of the First International Workshop on OpenMP,* 2005,

[58] C. Liu, A. Sivasubramaniam, M. Kandemir and M. J. Irwin, "Exploiting barriers to optimize power consumption of CMPs," in *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium,* 2005, pp. 5a-5a.

[59] D. B. Loveman, "High Performance Fortran," *IEEE Parallel & Distributed Technology: Systems & Technology,* vol. 1, pp. 25-42, 1993.

[60] G. Magklis, M. L. Scott, G. Semeraro, D. H. Albonesi and S. Dropsho, "Profile-based dynamic voltage and frequency scaling for a multiple clock domain

microprocessor," in *ISCA '03: Proceedings of the 30th Annual International Symposium on Computer Architecture,* 2003, pp. 14-25.

[61] D. T. Marr, F. Binns, D. L. Hill, G. Hinton, D. A. Koufaty, J. A. Miller and M. Upton. (Feb. 2002, Hyper-threading technology architecture and microarchitecture. *Intel Technology Journal 6(1),*

[62] R. L. McGregor, C. D. Antonopoulos and D. S. Nikolopoulos, "Scheduling algorithms for effective thread pairing on hybrid multiprocessors," in *IPDPS '05: Proceedingsof the 19th IEEE International Parallel and Distributed Processing Symposium,* 2005, pp. 28a.

[63] L. W. McVoy and C. Staelin, "Lmbench: Portable tools for performance analysis," in *USENIX Annual Technical Conference,* 1996, pp. 279-294.

[64] Message Passing Interface Forum. (1997, MPI, A message passing interface standard. *1.2*

[65] R. E. Millstein, "Control structures in Illiac IV Fortran," *Communications of the ACM,* vol. 16, pp. 621-627, 1973.

[66] T. Mudge, "Power: A First Class Design Constraint," *IEEE Computer,* vol. 34, pp. 52-52-57, Apr. 2001.

[67] D. Nellans, R. Balasubramonian and E. Brunvand, "A case for increased operating system support in chip multi-processors," in *P=ac2 '05: Proceedings of the 2nd IBM Watson Conference on Interaction between Architecture, Circuits, and Compilers,* 2005,

[68] D. S. Nikolopoulos and C. D. Polychronopoulos, "Adaptive scheduling under memory pressure on multiprogrammed SMPs," in *IPDPS '02: Proceedings of the International Parallel and Distributed Processing Symposium,* 2002, pp. 11-16.

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

[70] Openmp.org. Open MP - simple, portable, scalable SMP programming. Available: www.openmp.org

[71] 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," in *SC '03: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing,* 2003, pp. 55.

[72] E. D. Polychronopoulos, D. S. Nikolopoulos, T. S. Papatheodorou, X. Martorell, J. Labarta and N. Navarro, "An efficient kernel-level scheduling methodology for multiprogrammed shared memory multiprocessors," in *PDCS '99: Proceedings of the 12th International Conference on Parallel and Distributed Computing Systems,* pp. 148-155.

[73] A. Purkayastha, C. Guiang, K. Schulz, T. Minyard, K. Milfeld, W. Barth, P. Hurley and J. Boisseau, "Performance characteristics of dual-processor HPC cluster nodes based on 64-bit commodity processors," in *IISWC '05: Proceedings of the IEEE International Symposium on Workload Characterization,* 2005, pp. 87-98.

[74] F. J. L. Reid and J. M. Bull, "OpenMP microbenchmarks version 2.0," in *EWOMP '04: Proceedings of 6th European Workshop on OpenMP,* 2004,

[75] M. Renouf, F. Dubois and P. Alart, "A parallel version of the non smooth contact dynamics algorithm applied to the simulation of granular media," *J. Comput. Appl. Math.,* vol. 168, pp. 375-382, 2004.

[76] H. Saito, G. Gaertner, W. Jones, R. Eigenmann, H. Iwashita, R. Lieberman, M. van Waveren and B. Whitney, "Large system performance of SPEC OMP2001 benchmarks," in *WOMPAT '02: Workshop on OpenMP: Experiences and Implementation in the Proceedings of the International Workshop on OpenMP (IWOMP),* 2002,

[77] R. Sasanka, S. V. Adve, Y. Chen and E. Debes, "The energy efficiency of CMP vs. SMT for multimedia workloads," in *ICS '04: Proceedings of the 18th Annual International Conference on Supercomputing,* 2004, pp. 196-206.

[78] M. Sato, S. Satoh, K. Kusano and Y. Tanaka, "Design of OpenMP compiler for an SMP cluster," in *EWOMP '99: Proceedings of the European Workshop on OpenMP,* 1999, pp. 32–39.

[79] D. Skinner and W. Kramer, "Understanding the causes of performance variability in HPC workloads," in *IISWC '05: Proceedings of the IEEE International Symposium on Workload Characterization,* 2005, pp. 137-149.

[80] L. Smith and P. Kent, "Development and performance of a mixed OpenMP/MPI quantum Monte Carlo code," *Concurrency Practice and Experience,* vol. 12, pp. 1121-1129, 2000.

[81] A. Snavely, D. M. Tullsen and G. Voelker, "Symbiotic jobscheduling with priorities for a simultaneous multithreading processor," in *The Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems,* 2002, pp. 66-76.

[82] SPEC. (2005, SPEC OMP benchmark suite. Available: http://www.spec.org/omp/.

[83] L. Spracklen and S. G. Abraham, "Chip multithreading: Opportunities and challenges," in *HPCA '05: Proceedings of the International Symposium on High-Performance Computer Architecture,* 2005, pp. 248-252.

[84] U. Srinivasan, P. S. Chen, Q. Diao, C. C. Lim, E. Li, Y. Chen, R. Ju and Y. Zhang, "Characterization and analysis of HMMER and SVM-RFE parallel bioinformatics applications," in *IISWC '05: Proceedings of the IEEE International Symposium on Workload Characterization,* 2005, pp. 87-98.

[85] Sun Microsystems. (2006, UltraSPARC IV. *2006*Available: http://www.sun.com/processors/UltraSPARC-IVplus/index.xml

[86] Sun Microsystems. T1 processor. Available: http://www.sun.com/processors/UltraSPARC-T1/index.xml

[87] Synopsys. HSPICE - circuit simulation software. Available: http://www.synopsys.com/products/mixedsignal/hspice/hspice.html

[88] X. Tian, Y. 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 *IPDPS '03: Proceedings of the International Parallel and Distributed Processing Symposium,* 2003, pp. 8.

[89] D. Tsafrir, Y. Etsion, D. G. Feitelson and S. Kirkpatrick, "System noise, OS clock ticks, and fine-grained parallel applications," in *ICS '05: Proceedings of the 19th Annual International Conference on Supercomputing,* 2005, pp. 303-312.

[90] N. Tuck and D. M. Tullsen, "Initial observations of the simultaneous multithreading pentium 4 processor," in *PACT '03: Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques,* 2003, pp. 26-34.

[91] D. M. Tullsen, S. J. Eggers and H. M. Levy, "Simultaneous multithreading: Maximizing on-chip parallelism," in *ISCA '95: Proceedings of the 22nd Annual International Symposium on Computer Architecture,* 1995, pp. 392-403.

[92] O. S. Unsal, I. Koren, C. M. Krishna and C. A. Moritz, "Cool-Fetch: Compiler-Enabled Power-Aware Fetch Throttling," *IEEE Computer Architecture Letters,* vol. 1, pp. 100-103, 2002.

[93] N. Vachharajani, M. Iyer, C. Ashok, M. Vachharajani, D. I. August and D. Connors, "Chip multi-processor scalability for single-threaded applications," *ACM SIGARCH Computer Architecture News,* vol. 33, pp. 44-53, 2005.

[94] Y. Zhang, M. Burcea, V. Cheng, R. Ho, V. Cheng and M. Voss, "An adaptive OpenMP loop scheduler for hyperthreaded SMPs," in *IPDPS '04: Proceedings of 16th International Conference on Parallel and Distributed Computing Systems,* 2004,