PERFORMANCE ANALYSIS OF INTEL CORE 2 DUO PROCESSOR

A Thesis
Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering

In
The Department of Electrical and Computer Engineering

By
Tribuvan Kumar Prakash
Bachelor of Engineering in Electronics and Communication Engineering
Visveswaraiah Technological University, Karnataka, 2004.
August 2007

Acknowledgements

I would like to express my gratitude to my advisor, Dr. Lu Peng for his guidance, and constant motivation towards the completion of this thesis. His technical advice and suggestions helped me to overcome hurdles and kept me enthusiastic and made this work a wonderful learning experience.

I would like to thank my committee members Dr. David Koppelman and Dr. Suresh Rai for taking time out of their busy schedule and agreeing to be a part of my committee. I would like to also thank them for their valuable feedback.

I would like to thank the faculty members and Shirley and Tonya of the Department of Electrical Engineering, for all the support and making my study at Louisiana State University a pleasant experience.

I would like to thank my parents and sister without whom I would not have made it to this point. I would like to thank my friends Srinath Sitaraman and Balachandran Ramadas for their help while collecting data. I would also like to thank my roommates & friends here at LSU and back home for all the love and unending support.

Table of Contents

List of Tables iv

List of Figures v

Abstract vi

1. Introduction 1
   1.1 Overview 1
   1.2 Architecture of Intel Core 2 Duo 3

2. Performance Analysis of SPEC CPU Benchmarks Running on Intel's Core 2 Duo Processor 7
   2.1 Overview 7
   2.2 Methodology 7
   2.3 Measurement Results 9
      2.3.1 IPC and Instruction Profile 9
      2.3.2 L1 D-Cache Misses 11
      2.3.3 L2 Cache Misses 13
      2.3.4 Branch Misprediction 15

3. Performance Comparison of Dual Core Processor Using Microbenchmarks 17
   3.1 Overview 17
   3.2 Architecture of Dual-Core Processors 18
      3.2.1 Intel Pentium D 830 18
      3.2.2 AMD Athlon 64X2 19
      3.2.3 Processor Comparison 20
   3.3 Methodology 21
   3.4 Memory Bandwidth and Latency Measurements 23

4. Performance Comparison of Dual Core Processors Using Multiprogrammed and Multithreaded Benchmarks 31
   4.1 Overview 31
   4.2 Methodology 31
   4.3 Multiprogrammed Workload Measurements 33
   4.4 Multithreaded Program Behavior 36

5. Related Work 39

6. Conclusion 41

References 43

Vita 46

List of Tables

Table 1.1 Specification of Intel Core 2 Duo machine. 6

Table 2.1 SPEC CPU20006 Integer Benchmark 8

Table 2.2 SPEC CPU20006 Floating Point Benchmark 8

Table 3.1 Specifications of the selected processors 21

Table 3.2 Memory operations from *Lmbench* 22

Table 3.3 Kernel operations of the *STREAM* and *STREAM2* benchmarks 23

Table 4.1 Input parameters of the selected multithreaded workloads 33

List of Figures

Figure 1-1 Block Diagram of Intel Core 2 Duo Processor 4

Figure 1-2 Block Diagram of Intel Core Micro-architecture's IP Prefetcher 5

Figure 2-1 IPC of SPEC Benchmarks 10

Figure 2-2 Instruction Profile of SPEC Benchmarks 11

Figure 2-3 L1-D Cache Misses per 1000 instructions of SPEC Benchmarks 12

Figure 2-4 Sample Code of MCF Benchmark 13

Figure 2-5 L2 Cache Misses per 1000 instructions of SPEC Benchmarks 14

Figure 2-6 Sample Code of LBM Benchmark 15

Figure 2-7 Branch Mispredicted Per 1000 Instructions of SPEC Benchmarks 16

Figure 3-1 Block Diagram of Pentium D Processor 19

Figure 3-2 Block Diagram of AMD Athlon 64x2 Processor 20

Figure 3-3 Memory bandwidth collected from the *lmbench* suite (1 or 2 copies). 25

Figure 3-4 Memory load latency collected from the *lmbench* suite (1 or 2 copies) 27

Figure 3-5 Memory bandwidth and latency collected from the *STREAM* and *STREAM2* benchmarks (1 or 2 copies) 29

Figure 4-1 SPEC CPU2000 and CPU2006 benchmarks execution time 34

Figure 4-2 Multi-programmed speedup of mixed SPEC CPU 2000/2006 benchmarks… 35

Figure 4-3 (a) Execution time for 1-thread version of selected multithreaded programs. 36

Figure 4-4 Throughput of SPEC*jbb2005* running with 1 to 8 warehouses 38

Abstract

With the emergence of thread level parallelism as a more efficient method of improving processor performance, Chip Multiprocessor (CMP) technology is being more widely used in developing processor architectures. Also, the widening gap between CPU and memory speed has evoked the interest of researchers to understand performance of memory hierarchical architectures. As part of this research, performance characteristic studies were carried out on the Intel Core 2 Duo, a dual core power efficient processor, using a variety of new generation benchmarks. This study provides a detailed analysis of the memory hierarchy performance and the performance scalability between single and dual core processors. The behavior of SPEC CPU2006 benchmarks running on Intel Core 2 Duo processor is also explained. Lastly, the overall execution time and throughput measurement using both multi-programmed and multi-threaded workloads for the Intel Core 2 Duo processor is reported and compared to that of the Intel Pentium D and AMD Athlon 64X2 processors. Results showed that the Intel Core 2 Duo had the best performance for a variety of workloads due to its advanced micro-architectural features such as the shared L2 cache, fast cache to cache communication and smart memory access.

1. Introduction

1.1 Overview

This thesis work analyzes the performance characteristics of major architectural developments employed in Intel Core 2 Duo E6400 processor with 2.13GHz [15]. Intel Core 2 Duo is a high performance and power efficient dual core Chip-Multiprocessor (CMP). CMP embeds multiple processor cores into a single die to exploit thread-level parallelism for achieving higher overall chip-level Instruction-Per-Cycle (IPC) [4] [14] [15] [21]. In a multi-core, multithreaded processor chip, thread-level parallelism combined with increased clock frequency exerts a higher demand for on-chip and off-chip memory bandwidth causing longer average memory access delays. There has been great interest shown by researchers to understand the underlying reasons that cause these bottlenecks in processors.

The advances in circuit integration technology and inevitability of thread level parallelism over instruction level parallelism for performance efficiency has made Chip-Multiprocessor (CMP) or multi-core technology the mainstream in CPU designs. With the evolution of processor architectures over time, the benchmarks used to measure the performance of these high performance processors have also continued to evolve. Many single and multi threaded benchmarks have been defined and developed to stress the processor units to its maximum limit. Standard Performance Evaluation Corporation (SPEC) is one of the non profit organizations that have been developing benchmarks to meet the requirements of these dynamic processor architectures for nearly a decade. SPEC CPU2006 is a single-threaded compute-intensive benchmark developed by SPEC using C, C++ and FORTRAN programming language. To understand the performance of

multi-core processors completely it is equally important to understand their behavior while running multi threaded applications. *SPEC JBB2005, lmbench, bioperf* and *splash2* are some of the most popularly used multithreaded benchmarks for this purpose.

This thesis work focuses mainly on workload characteristics, memory system behavior and multi-thread interaction of the benchmarks. This work also seeks to report performance measurement on Intel Core 2 Duo E6400 with 2.13GHz [15] and compare the results with Intel Pentium D 830 with 3.0GHz [19] and AMD Athlon 64X2 4400+ with 2.2GHz [2]. In contrast to existing performance evaluations [13] [26] [27] that usually provide overall execution time and throughput, this work emphasizes on the memory hierarchy performance. It reports the measured memory access latency and bandwidth as well as cache-to-cache communication delays. It also examines the performance scalability between single and dual cores on the three tested processors.

Summarized below are a few interesting findings based on experiments conducted as part of this research:

- SPEC CPU2006 running on Core 2 Duo exerts less pressure on the L1 cache compared to SPEC CPU2000 benchmarks. However, CPU2006 benchmarks have larger data sets and longer execution times resulting in comparatively high stress on L2 cache.
- The cache to cache latency of Core 2 Duo was measured to be 33ns. Core 2 Duo has high memory bandwidth and low latency as a result of on-chip access to the other L1 cache and the presence of aggressive memory dependence predictors. . Its shared L2 generates less off-chip traffic than the other two.

- Due to its shared L2 cache access the execution time of all single threaded workloads are fast and range from 56-1500 seconds for Core 2 Duo. The average multi-programmed speedup for CPU2006 and CPU2000 benchmarks was measured at 1.76 and 1.77 respectively which is lower than the ideal speedup of 2. The Core 2 Duo's speed-ups are constrained due to its ability to use the entire L2 cache.

1.2 Architecture of Intel Core 2 Duo

The Intel Core 2 Duo E6400 (Figure 1.1) processor supports CMP and belongs to the Intel's mobile core family. It is implemented by using two Intel's Core architecture on a single die. The design of Intel Core 2 Duo E6400 is chosen to maximize performance and minimize power consumption [18]. It emphasizes mainly on cache efficiency and does not stress on the clock frequency for high power efficiency. Although clocking at a slower rate than most of its competitors, shorter stages and wider issuing pipeline compensates the performance with higher IPC's. In addition, the Core 2 Duo processor has more ALU units [13]. The five main features of Intel Core 2 Duo contributing towards its high performance are:

• Intel's Wide Dynamic Execution

• Intel's Advanced Digital Media Boost

• Intel's Intelligent Power Capability

• Intel's Advanced Smart Cache

• Intel's Smart Memory Access

Core 2 Duo employs Intel's Advanced Smart Cache which is a shared L2 cache to increase the effective on-chip cache capacity. Upon a miss from the core's L1 cache, the

shared L2 and the L1 of the other core are looked up in parallel before sending the request to the memory [18]**.** The cache block located in the other L1 cache can be fetched without off-chip traffic. Both memory controller and FSB are still located off-chip. The off-chip memory controller can adapt the new DRAM technology with the cost of longer memory access latency. Intel Advanced Smart Cache provides a peak transfer rate of 96 GB/sec (at 3 GHz frequency) [*17*].

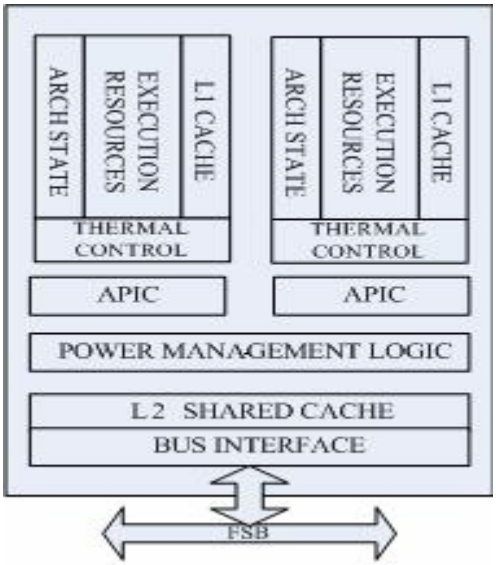


Figure 1-1 Block Diagram of Intel Core 2 Duo Processor

Core 2 Duo employs aggressive memory dependence predictors for memory disambiguation. A load instruction is allowed to be executed before an early store instruction with an unknown address. It also implements a macro-fusion technology to combine multiple micro-operations.

Another important aspect to alleviate cache miss penalty is data prefetching. According to the hardware specifications, the Intel Core 2 Duo includes a stride prefetcher on its L1 data cache [17] and a next line prefetcher on its L2 cache [13]. The Intel Core micro-architecture includes in each processing core two prefetchers to the Level 1 data cache and the traditional prefetcher to the Level 1 instruction cache. In

addition it includes two prefetchers associated with the Level 2 cache and shared between the cores. In total, there are eight prefetchers per dual core processor [17].The L2 prefetcher can be triggered after detecting consecutive line requests twice.

The stride prefetcher on L1 cache is also known as Instruction Pointer-Based (IP) prefetcher to level 1 data cache (Figure 1.2). The IP prefetcher builds a history for each *load* using the *load instruction pointer* and keeps it in the IP history array. The address of the next load is predicted using a constant stride calculated from the entries in the history array [17]. The history array consists of the following fields.

- 12 un-translated bits of last demand address
- 13 bits of last stride data (12 bits of positive or negative stride with the 13th bit the sign)
- 2 bits of history state machine
- 6 bits of last prefetched address—used to avoid redundant prefetch requests.

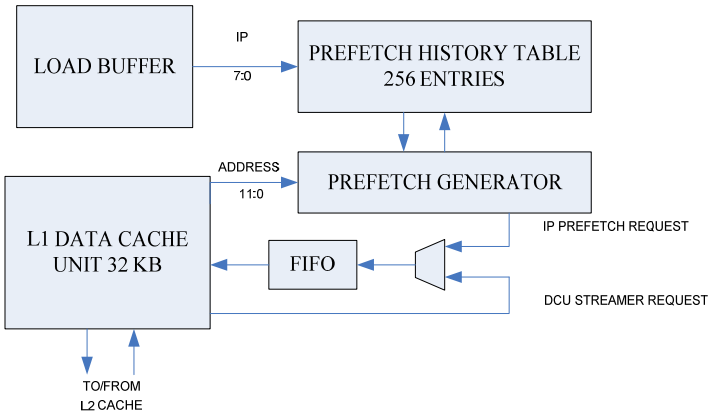


Figure 1-2 Block Diagram of Intel Core Micro-architecture's IP Prefetcher

The IP prefetcher then generates a prefetch request to L1 cache for the predicted address. This request for prefetch enters a FIFO and waits for its turn. When the request is encountered a lookup for that line is done in the L1 cache and the fill buffer unit. If the

prefetch hits either the L1 cache or the fill buffer, the request is dropped. Otherwise a read request to the corresponding line is sent to L2 cache.

Other important features involve support for new SIMD instructions called Supplemental Streaming SIMD Extension 3, coupled with better power saving technologies. Table 1.1 specifies the CPU specification of the Intel Core 2 Duo machine used for carrying out the experiments. It has separate 32 KB L1 instruction and data caches per core. A 2MB L2 cache is shared by two cores. Both L1 and L2 caches are 8-way set associative and have 64-byte lines.

Table 1.1 Specification of Intel Core 2 Duo machine.

CPU	Intel Core 2 Duo E6400 (2 x 2.13GHz)
Technology	65nm
Transistors	291 Millions
Hyperthreading	No
L1 Cache	Code and Data: 32 KB X 2, 8 way, 64–byte cache line size, write-back
L2 Cache	2MB shared cache (2MB x 1), 8-way, 64-byte line size, non-inclusive with L1 cache.
Memory	2GB (1GB x 2) DDR2 533MHz
FSB	1066MHz Data Rate 64-bit
FSB bandwidth	8.5GB/s
HD Interface	SATA 375MB/s

The remainder of this work is organized as follows. Chapter 2 analyzes SPEC CPU2006 benchmark using variety of performance results obtained from Intel(R) VTune(TM) Performance Analyzer 8.0.1 and compares it with SPEC CPU2000 benchmarks. Chapter 3 compares memory latency and hierarchy of three dual core processors using micro-benchmarks. Chapter 4 discusses the performance measurement results for three dual core processors using single threaded, multi-programmed and multithreaded workloads. Chapter 5 describes related work. Finally, chapter 6 explains the brief conclusion obtained.

2. Performance Analysis of SPEC CPU Benchmarks Running on Intel's Core 2 Duo Processor

2.1 Overview

With the evolution of processor architecture over time, benchmarks that were used to measure the performance of these processors are not as useful today as they were before due to their inability to stress the new architectures to their maximum capacity in terms of clock cycles, cache, main memory and I/O bandwidth. Hence new and improved benchmarks need to be developed and used. The SPEC CPU2006 is one such benchmark that has intensive workloads based on real applications and is a successor of the SPEC CPU2000 benchmark.

This section presents a detailed analysis of the SPEC CPU2006 benchmark running on the Core 2 duo processor discussed earlier and emphasizes on its workload characteristics and memory system behavior. Also, the cpu2006 and cpu2000 benchmarks are compared with respect to performance bottlenecks by using the v-tune performance analyzer for the entire program execution.

2.2 Methodology

The SPEC CPU2006 has 29 benchmarks with 12 integer and 17 floating point programs. For our experiments, all the integer programs and a subset of 10 floating point programs were considered. The details of these benchmark programs are shown in Tables 2.1 and 2.2.

All experiments were run on systems with 32 bit Windows XP SP2 operating system and Intel Core 2 Duo processors, as explained in Chapter 1. The Intel(R) VTune(TM) Performance Analyzer 8.0.1 was used to analyze all benchmarks for their

Table 2.1 SPEC CPU20006 Integer Benchmark

Integer Benchmark	Language	Description
Astar	C++	Path-Finding Algorithm
Bzip2	C	Compression
Gcc	C	C Compiler
Gobmk	C	Artificial Intelligence: go
H264ref	C	Video Compression
Hmmer	C	Search Gene Sequence
Libquantum	C	Physics: Quantum Computing
Mcf	C	Combinatorial Optimization
Omnetpp	C++	Discrete Event Simulation
Perlbench	C	PERL Programming Language
Sjeng	C	Artificial Intelligence: Chess
Xalancbmk	C++	XML Processing

Table 2.2 SPEC CPU20006 Floating Point Benchmark

Floating Point benchmarks	Language	Description
Bwaves	Fortran	Fluid Dynamics
Gamess	Fortran	Quantum Chemistry
Milc	C	Physics: Quantum Chromodynamics
Gromacs	C/Fortran	Biochemistry/Molecular Dynamics
CactusADM	C/Fortran	Physics / General Relativity
Leslie3d	Fortran	Fluid Dynamics
Soplex	C++	Linear Programming, Optimization
GemsFDTD	Fortran	Computational Electromagnetics
Lbm	C	Fluid Dynamics
Sphinx3	C	Speech recognition

complete run time [20]. At a given time, Intel(R) VTune(TM) Performance Analyzer 8.0.1 can measure only certain definite number of events, depending upon the configuration; hence, several complete runs were made to measure all the events. All

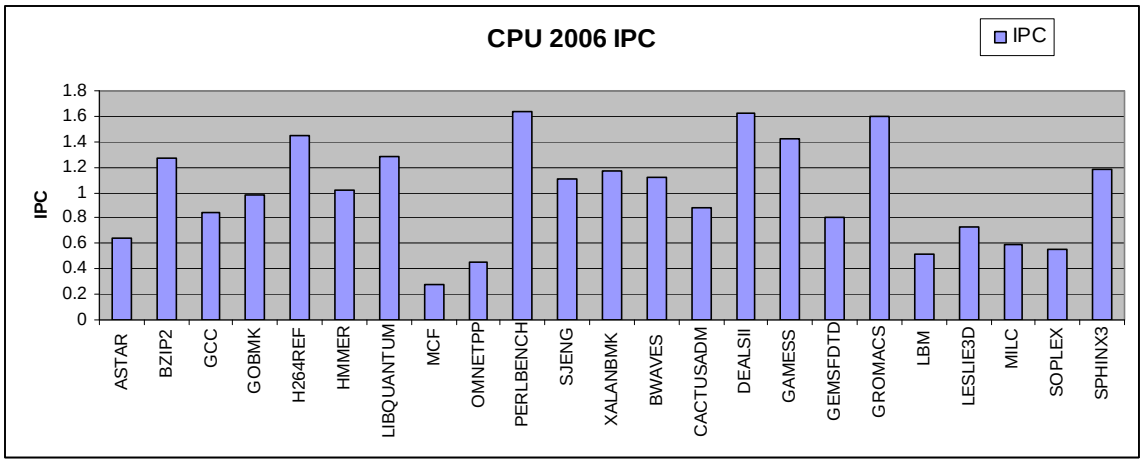
benchmarks were compiled using Microsoft Visual C/C++ 2005 and Intel® FORTRAN Compiler 9.1. We used the fastest speed compilation flags i.e. for the Microsoft VC++ compiler, we set “-O2”.

2.3 Measurement Results

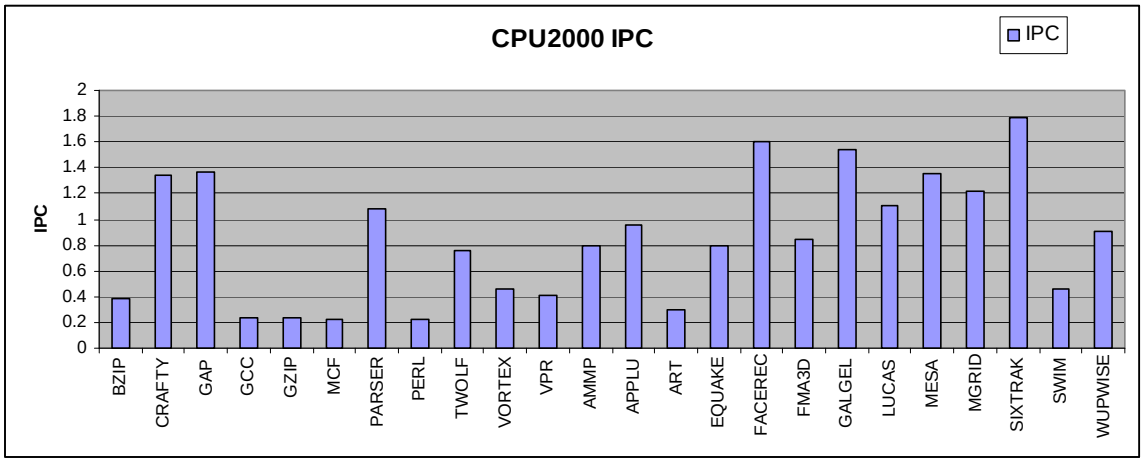
2.3.1 IPC and Instruction Profile

Figure 2.1(a) and Figure 2.1(b) represent the IPC of CPU2006 and CPU2000 respectively. The average IPC’s for CPU2006 and CPU2000 benchmarks were measured at 1.006 and 0.85 respectively. From the figures it can be observed that *mcf, omnetpp* and *lbm* have low IPC among CPU2006 benchmarks, while *mcf, art* and *swim* have low IPC among the CPU2000 benchmarks. It is interesting to understand the causes of performance bottlenecks among these benchmarks and to do so the instruction profiles of these benchmarks were analyzed.

Figure 2.2(a) and Figure 2.2(b) represent the instruction profile of CPU2006 and CPU2000 respectively. It is evident from the figure that a very high percentage of instructions retired consist of loads and stores. CPU2006 benchmarks like *h264ref, hmmer, bwaves*, lesli3d and *GemsFDTD* have comparatively high percentage of loads while *astar, bzip2, gcc, gobmk, libquantum, mcf, omnetpp, perlbench, sjeng, xalancbmk* and *gamess* have high percentage of branch instructions. On the contrary CPU2000 benchmarks like *gap, parser, vortex, applu, equake, fma3d, mgrid* and *swim* have comparatively high percentage of loads while almost all integer programs have high percentage of branch instructions.



(a)

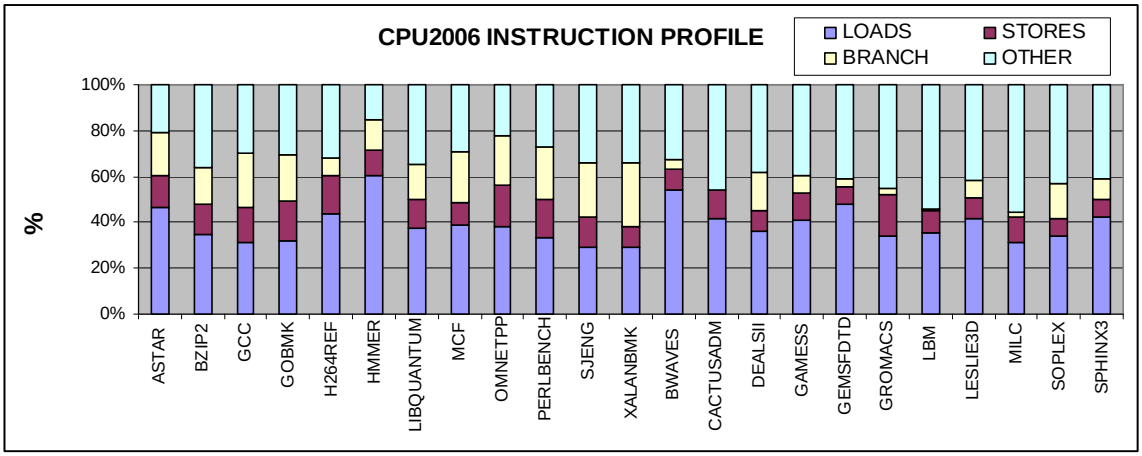


(b)

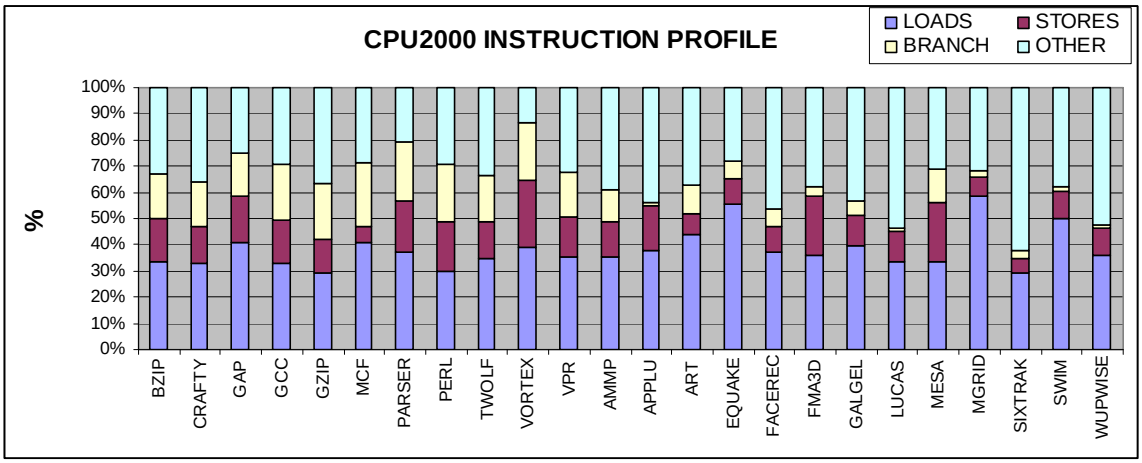
Figure 2-1 (a) IPC of SPEC CPU2006 Benchmarks
(b) IPC of SPEC CPU2000 Benchmarks

However, higher percentage of load and store instructions retired or higher percentage of branches do not indicate presence of better bottlenecks. For example *h264ref* and *perlbench* have high percentage of load, store and branch instructions, but they also have comparatively high IPC. Similarly among CPU2000 benchmarks *crafty*, *parser* and *perl* have high percentage of load, store and branch instruction and have better IPC. To get a better understanding of the bottlenecks of these benchmarks, L1 cache misses, L2 cache misses and branch instruction mis-predicted were measured and

analyzed. The higher the measured rates the better is the bottleneck produced by the respective benchmark.



(a)



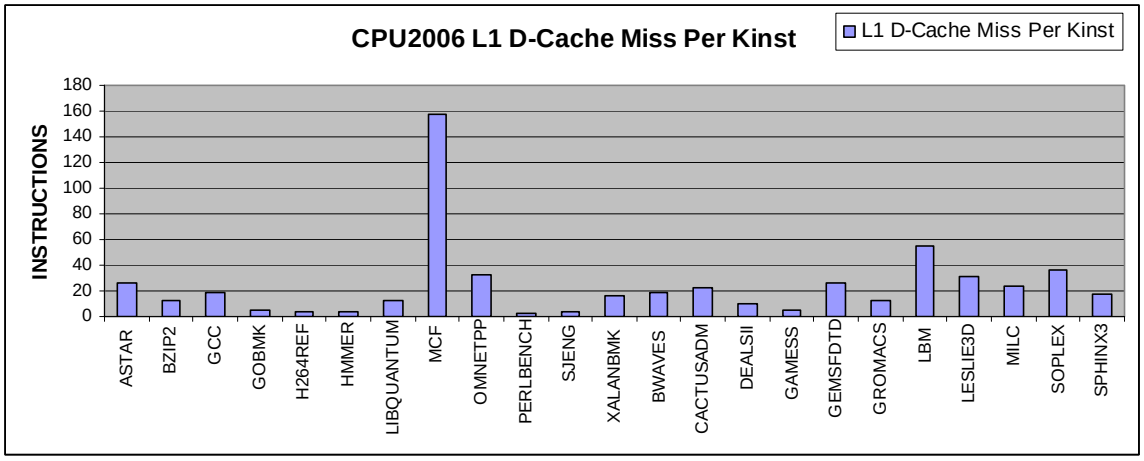
(b)

Figure 2-2 (a) Instruction Profile of SPEC CPU2006 Benchmarks
(b) Instruction Profile of SPEC CPU2000 Benchmarks

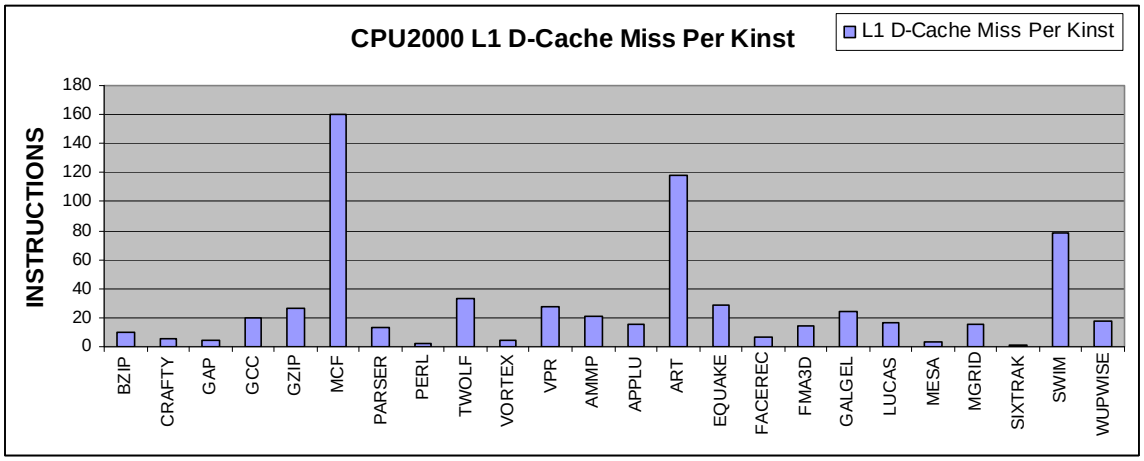
2.3.2 L1 D-Cache Misses

Figure 2.3(a) and 2.3(b) indicates the L1 cache misses per 1000 instructions of CPU2006 and CPU2000 benchmarks. The results show that there is no significant improvement in CPU2006 than CPU2000 with respect to stressing the L1 cache. The average L1-D cache misses per 1000 instructions for cpu2006 and cpu2000 benchmark set under consideration was found to be 24.2 and 27.8 respectively. The *mcf* benchmark

has highest L1 cache misses per 1000 instructions in both CPU2000 and CPU2006 benchmarks. This is one of the significant reasons for its low IPC.



(a)



(b)

Figure 2-3 (a) L1-D Cache Misses per 1000 instructions of SPEC CPU2006 Benchmarks
(b) L1-D Cache Misses per 1000 instructions of SPEC CPU2000 Benchmarks

Mcf is a memory intensive integer benchmark written in C language. Code analysis using Intel(R) VTune(TM) Performance Analyzer 8.0.1 shows that the key functions responsible for stressing the various processor units are primal_bea_mpp and refresh_potential. Primal_bea_mpp (72.6%) and refresh_potential (12.8%) together are responsible for 85% of the overall L1 data cache miss events.

A code sample of primal_bea_mpp function is shown in Figure 2.4. The function traverses an array of pointer (denoted by arc_t) to a set of structures. For each structure traversed, it optimizes the routines used for massive communication. In the code under consideration, pointer chasing in line 6 is responsible for more than 50% of overall L1D cache misses for the whole program. Similar result for mcf in CPU2000 was also found in previous work [11]. Apart from mcf, lbm have comparatively significant L1 cache misses rate in CPU2006 and mcf, art and swim have comparatively significant L1 cache misses rate in CPU2000.

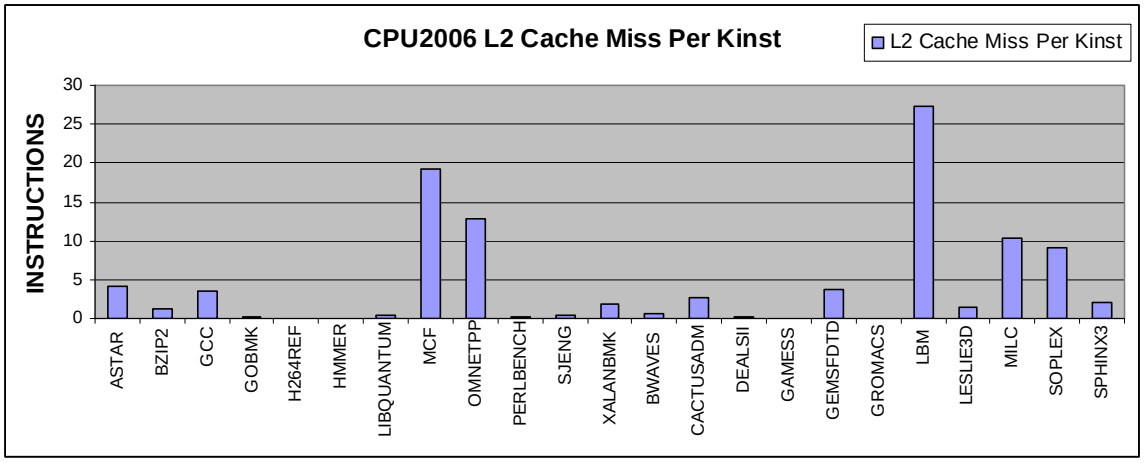
```
Ln1   NEXT: /* price next group */
Ln2    arc = arcs + group_pos;
Ln3    for( ; arc < stop_arcs; arc += nr_group ){
Ln4     if( arc->ident > BASIC ){
Ln5          /* red_cost = bea_compute_red_cost( arc ); */
Ln6          red_cost = (arc->cost - arc->tail->potential +
                         arc->head->potential);
Ln7           if( bea_is_dual_infeasible( arc, red_cost ) ){
  :              ....
Ln8              perm[basket_size]->cost = red_cost;
Ln9              perm[basket_size]->abs_cost = ABS(red_cost);
                }
          }
        }
```

Figure 2-4 Sample Code of MCF Benchmark

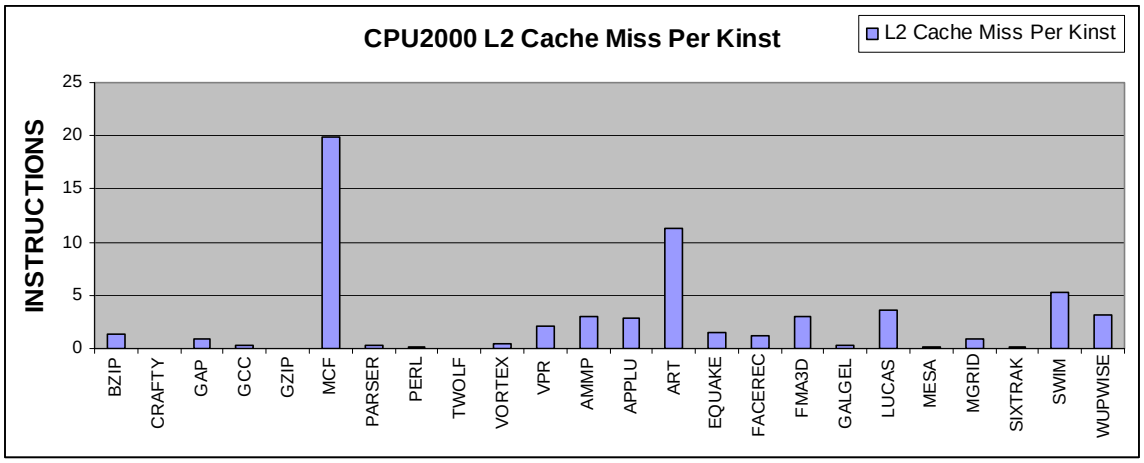
2.3.3 L2 Cache Misses

Figure 2.4(a) and 2.4(b) represent the L2 cache misses per 1000 instructions of CPU2006 and CPU2000 SPEC benchmarks respectively. The average L2 cache misses per 1000 instructions for CPU2006 and CPU2000 benchmarks under consideration was found to be 4.4 and 2.5 respectively. *Lbm* has the highest L2 cache misses which attributes for its low IPC. *Lbm* (Lattice Boltzmann Method) is a floating point based benchmark written in C language. It is used in the field of fluid dynamics to simulate the behavior of fluids in 3D. *Lbm* has two steps of accessing memory, namely i) streaming

step , in which values are derived from neighboring cells and ii) linear memory access to read the cell values (collide-stream) and write the values to the cell (stream-collide) [9].



(a)



(b)

Figure 2-5 (a) L2 Cache Misses per 1000 instructions of SPEC CPU2006 Benchmarks
(b) L2 Cache Misses per 1000 instructions of SPEC CPU2000 Benchmarks

Code analysis reveals that LBM_performStreamCollide function used to write the values to the cell is responsible for 99.98% of the overall L2 cache miss events. A code sample of the same function is shown in Figure 2.6. A macro “TEST_FLAG_SWEEP” is responsible for 21% of overall L2 cache misses. The definition of TEST_FLAG_SWEEP is shown in Figure 2.6(b). The pointer *MAGIC_CAST dynamically accesses memory accesses over 400MB of data which is much larger than the available L2 cache size

(2MB), resulting in very high L2 cache misses. Hence it can be concluded that lbm has very large data footprint which results in high stress on L2 cache. For mcf, Primal_bea_mpp (33.4%) and refresh_poten-tial (20.2%) are two major functions resulting in L2 cache misses. Intensive pointer chasing is responsible for this.

```
Ln1   if( TEST_FLAG_SWEEP( srcGrid, ACCEL )){
Ln2   …
  :   …
  :   }
```

(a)

```
#define TEST_FLAG_SWEEP(srcGrid,f)
        ((*MAGIC_CAST(LOCAL(srcGrid, FLAGS))) & (f))
```

(b)

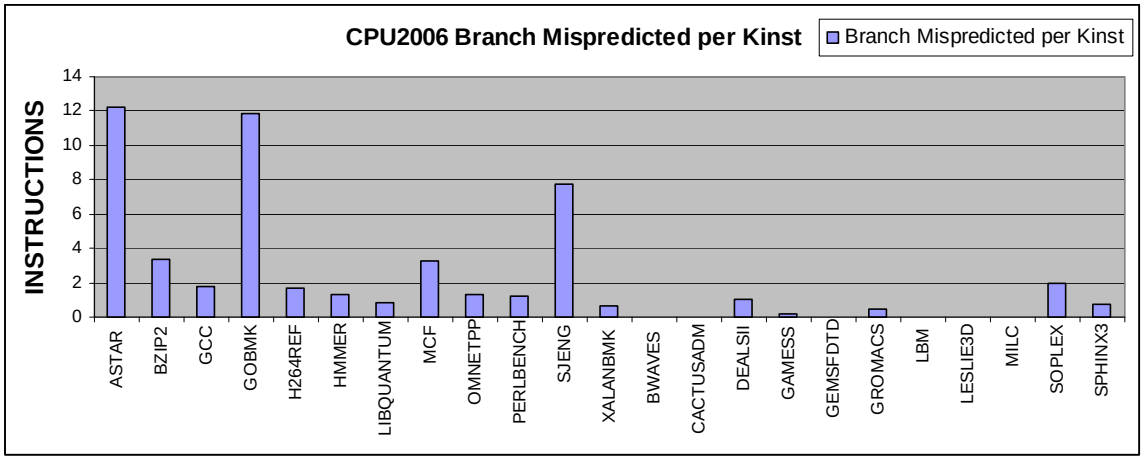
Figure 2-6 Sample Cde of LBM Benchmark

2.3.4 Branch Misprediction

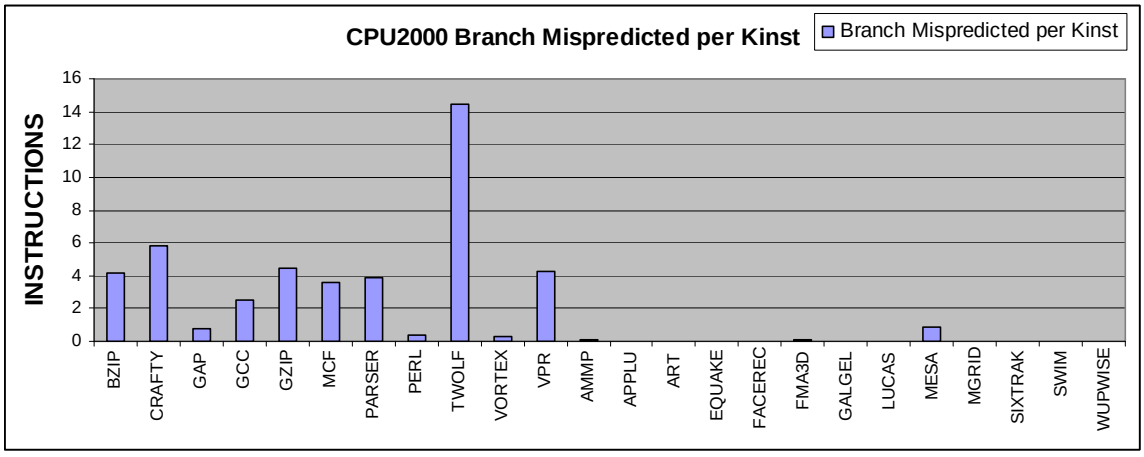
Figure 2.5(a) and 2.5(b) represents the branch mispredicted per 1000 instructions of CPU2006 and CPU2000 SPEC benchmarks. CPU2006 benchmarks have comparatively higher branch misprediction than CPU2000 benchmark and almost all floating point benchmarks under consideration have negligible branch misprediction comparatively. The average branch mispredicted per 1000 instructions for CPU2006 and CPU2000 integer benchmarks were measured as 4.2 and 4.0 respectively and the average branch misprediction per 1000 instructions for CPU2006 and CPU2000 floating point benchmarks were measured as 0.4 and 0.08 respectively.

We also measured L1 DTLB misses for SPEC CPU2006. Only a few programs have L1 DTLB miss rates equal to or larger than 1%. They are astar (1%), mcf (6%), omnetpp (1%) and cactusADM (2%). Some programs have very small L1 DTLB miss rate, for example, the miss rates for hammer, gromacs are 3.3*10-5 and 6.2*10-5 respectively. Other interesting results include *hmmer* and *h264ref* that has very high

percentage of loads and store but have negligible L1 and L2 cache misses per 1000 instructions. This is likely because *hmmer* and *h264ref* exhibit high locality of data set which favors the hardware prefetcher.



(a)



(b)

Figure 2-7 (a) Branch Mispredicted Per 1000 Instructions of SPEC CPU2006 Benchmarks; (b) Branch Mispredicted Per 1000 Instructions of SPEC CPU2000 Benchmarks

Thus from the results analyzed so far we can conclude that the cpu2006 benchmarks have larger data sets and requires longer execution time than its predecessor CPU2000 benchmarks.

3. Performance Comparison of Dual Core Processor Using Microbenchmarks

3.1 Overview

In this section performance measurement results of three dual core desktop processors: Intel Core 2 Duo E6400 with 2.13GHz [15], Intel Pentium D 830 with 3.0GHz [19] and AMD Athlon 64X2 4400+ with 2.2GHz [2] are analyzed and compared. The results in this section of work done emphasizes mainly on memory hierarchy and cache-to-cache communication delays of the three processors under consideration.

There are several key design choices for the memory subsystem of the three processors. All three have private L1 caches with different sizes. At the next level, the Intel Core 2 Duo processor adapts a shared L2 cache design, called *Intel Advanced Smart Cache* for the dual cores [17]. The shared L2 approach provides a larger cache capacity by eliminating data replications. It also permits naturally sharing of cache space among multiple cores. When only one core is active, the entire shared L2 can be allocated to the single active core. However, the downside for the shared L2 cache is that it suffers longer hit latency and may encounter competitions of its shared cache resources. Both the Intel Pentium D and the AMD Athlon 64X2 have a private L2 cache for each core, enabling fast L2 accesses, but restricting any capacity sharing among the two cores.

The shared L2 cache in the Core 2 Duo eliminates on-chip L2-level cache coherence. Furthermore, it resolves coherence of the two core's L1 caches internally within the chip for fast access to the L1 cache of the other core. The Pentium D uses an off-chip Front-Side Bus (FSB) for inter-core communications. Basically, the Pentium D is basically a technology remap of the Pentium 4 Symmetric Multiprocessor (SMP) that requires accessing the FSB for maintaining cache coherence. AMD Athlon 64X2 uses a

Hyper-Transport interconnect technology for faster inter-chip communication. Given an additional ownership state in the Athlon 64X2, cache coherence between the two cores can be accomplished without off-chip traffic. In addition, the Athlon 64X2 has an on-die memory controller to reduce memory access latency.

To examine memory bandwidth and latency, we used *lmbench* [33], a suite of memory measurement benchmarks. *Lmbench* attempts to measure the most commonly found performance bottlenecks in a wide range of system applications. These bottlenecks can be identified, isolated, and reproduced in a set of small micro-benchmarks, which measure system latency and bandwidth of data movement among the processor, memory, network, file system, and disk. In addition, we also ran *STREAM* [24] and *STREAM2* [25] recreated by using *lmbench*'s timing harness. They are kernel benchmarks measuring memory bandwidth and latency during several common vector operations such as matrix addition, copy of matrix, etc. We also used a small *lockless* program [29] to measure the cache-to-cache latency of the three processors. The *lockless* program records the duration of ping-pong procedures of a small token bouncing between two caches to get the average cache-to-cache latency.

3.2 Architecture of Dual-Core Processors

3.2.1 Intel Pentium D 830

The Pentium D 830 (Figure 3.1) glues two Pentium 4 cores together and connects them with the memory controller through the north-bridge. The off-chip memory controller provides flexibility to support the newest DRAM with the cost of longer memory access latency. The MESI coherence protocol from Pentium SMP is adapted in Pentium D that requires a memory update in order to change a modified block to shared.

The systems interconnect for processors remains through the Front-Side Bus (FSB). To accommodate the memory update, the FSB is located off-chip that increases the latency for maintaining cache coherence.

The Pentium D's hardware prefetcher allows stride-based prefetches beyond the adjacent lines. In addition, it attempts to trigger multiple prefetches for staying 256 bytes ahead of current data access locations [16]. The advanced prefetching in Pentium D enables more overlapping of cache misses.

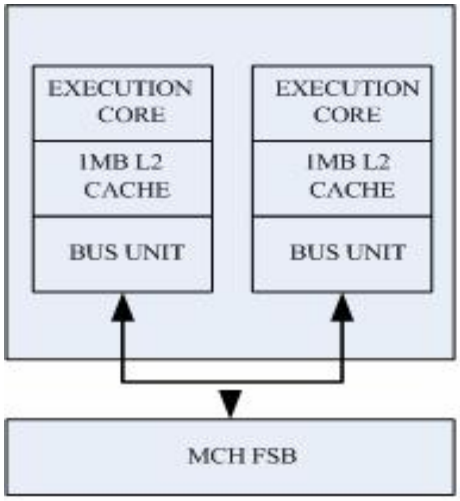


Figure 3-1 Block Diagram of Pentium D Processor

3.2.2 AMD Athlon 64X2

The Athlon 64X2 (Figure 3.2) is designed specifically for multiple cores in a single chip (Figure 1(c)). Similar to the Pentium D processor, it also employs private L2 caches. However, both L2 caches share a system request queue, which connects with an on-die memory controller and a Hyper-Transport. The Hyper-Transport removes system bottlenecks by reducing the number of buses required in a system. It provides significantly more bandwidth than current PCI technology [3]. The system request queue serves as an internal interconnection between the two cores without involvements of an external bus. The Athlon 64X2 processor employs MOESI protocol, which adds an

"Ownership" state to enable blocks to be shared on both cores without the need to keep the memory copy updated.

The Athlon 64X2 has a next line hardware prefetcher. However, accessing data in increments larger than 64 bytes may fail to trigger the hardware prefetcher [5].

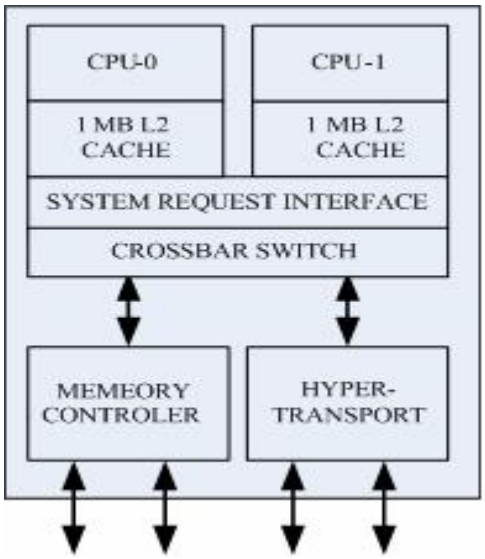


Figure 3-2 Block Diagram of AMD Athlon 64x2 Processor

3.2.3 Processor Comparison

Table 3.1 lists the specifications of the three processors experimented in this paper. There are no Hyper-threading settings on any of these processors. The Intel Core 2 Duo E6400 has separate 32 KB L1 instruction and data caches per core. A 2MB L2 cache is shared by two cores. Both L1 and L2 caches are 8-way set associative and have 64-byte lines. The Pentium D processor has a Trace Cache which stores 12Kuops. It is also equipped with a write-through, 8-way 16KB L1 data cache with a private 8-way 1MB L2 cache. The Athlon 64X2 processor's L1 data and instruction cache are 2-way 64KB with a private 16-way 1MB L2 cache for each core. Athlon 64X2's L1 and L2 caches in each core is exclusive. All three machines have the same size L2 caches and Memory. The Core 2 Duo and the Pentium D are equipped with DDR2 DRAM using advanced memory

controllers in their chipsets. The Athlon 64X2 has a DDR on-die memory controller. All three machines have 2GB memory. The FSB of the Core 2 Duo is clocked at 1066MHz with bandwidth up to 8.5GB/s. The FSB of the Pentium D operates at 800MHz and provides up to 6.4GB/sec bandwidth. The Athlon 64X2 has a 2GHz I/O Hyper-Transport with bandwidth up to 8GB/s. Bandwidth of hard drive interface for the three machines are 375MB/s, 150MB/s and 300MB/s respectively. Because of our experiments are all in-memory benchmarks, difference in hard drives should have little impact.

Table 3.1 Specifications of the selected processors

CPU	Intel Core 2 Duo E6400 (2 x 2.13GHz)	Intel Pentium D 830 (2 x 3.00GHz)	AMD Athlon64 4400+ (2 x 2.20GHz)
Technology	65nm	90nm	90nm
Transistors	291 Millions	230 Millions	230 Millions
Hyperthreading	No	No	No
L1 Cache	Code and Data: 32 KB X 2, 8 way, 64–byte cache line size, write-back	Trace cache: 12Kuops X 2, data: 16KB X 2, 8-way, 64-byte line size, write-through	Code and data: 64KB X 2, 2-way, 64-byte cache line size, write-back
L2 Cache	2MB shared cache (2MB x 1), 8-way, 64-byte line size, non-inclusive with L1 cache.	2MB private cache (1MB x 2), 8-way, 64-byte line size, inclusive with L1 cache.	2MB private cache (1MB x 2), 16-way, 64-byte line size, exclusive with L1 cache.
Memory	2GB (1GB x 2) DDR2 533MHz	2GB(512MBx4) DDR2 533MHz	2GB(1GB x 2) DDR 400MHz
FSB	1066MHz Data Rate 64-bit	800MHz Data Rate 64-bit	HyperTransport 16bit up/down 2GHz Data Rate (up+down)
FSB bandwidth	8.5GB/s	6.4GB/s	8GB/s
HD Interface	SATA 375MB/s	SATA 150MB/s	SATA 300MB/s

3.3 Methodology

We installed SUSE linux 10.1 with kernel 2.6.16-smp on all three machines. We used maximum level GCC optimization to compile the C/C++ benchmarks of *lmbench* and *lockless program*. We used *lmbench* suite running on the three machines to measure

bandwidth and latency of memory hierarchy. *Lmbench* attempts to measure performance bottlenecks in a wide range of system applications. These bottlenecks have been identified, isolated, and reproduced in a set of small micro-benchmarks, which measure system latency and bandwidth of data movement among the processor, memory, network, file system, and disk.

Table 3.2 Memory operations from *Lmbench*

Operation	Description
Libc bcopy unaligned	measuring how fast the processor can copy data blocks when data segments are not aligned with pages using a system call bcopy().
Libc bcopy aligned	measuring how fast the processor can copy data blocks when data segments are aligned with pages using a system call bcopy().
Memory bzero	measuring how fast the processor can reset memory blocks using a system call bzero().
Unrolled bcopy unaligned	measuring how fast the system can copy data blocks without using bcopy(), when data segments are not aligned with pages.
Memory read	measuring the time to read every 4 byte word from memory
Memory write	measuring the time to write every 4 byte word to memory

In our experiments, we focus on the memory subsystem and measure memory bandwidth and latency with various operations [33]. Table 3.2 lists the operations used to test memory bandwidth and their meanings. We can run variable stride accesses to get average memory read latency. In addition, we ran multi-copies *lmbench*, one on each core to test the memory hierarchy system. We also ran *STREAM* [24] and *STREAM2* [25] recreated by using *lmbench*'s timing harness. They are simple vector kernel benchmarks measuring memory bandwidth. Each version has four common vector operations as listed in Table 3.3. Average memory latencies for these operations are also reported.

Table 3.3 Kernel operations of the *STREAM* and *STREAM2* benchmarks

Set	Kernel	Operation
STREAM	copy	c[i]=a[i]
STREAM	scale	b[i] = scalar * c[i]
STREAM	add	c[i] = a[i] + b[i]
STREAM	triad	a[i] = b[i] + scalar *
STREAM2	fill	a[i] = q
STREAM2	copy	a[i] = b[i]
STREAM2	daxpy	a[i] = a[i] + q * b[i]
STREAM2	sum	sum = sum + a[i]

We measured the cache-to-cache latency using a small lockless program [29]. It doesn't employ expensive read-modify-write atomic instructions. Instead, it maintains a lockless counter for each thread. The c-code of each thread is as follows.

```
*pPong = 0;
  for (i = 0; i < NITER; ++i)
  {
      while (*pPing < i);
      *pPong = i+1;
  }
```

Each thread increases its own counter pPong and keeps reading the peer's counter by checking pPing. The counter pPong is in a different cache line from the counter pPing. A counter pPong can be increased by one only after verifying the update of the peer's counter. This generates a heavy read-write sharing between the two cores and produces a Ping-Pong procedure between the two caches. The average cache-to-cache latency is measured by repeating the procedure.

3.4 Memory Bandwidth and Latency Measurements

We used the lockless program described in section 3.3 to measure the dual-core cache-to-cache latency. The average cache-to-cache latency of Core 2 Duo, Pentium D, and Athlon 64X2 are 33ns, 133ns and 68ns respectively. Core 2 Duo resolves L1 cache

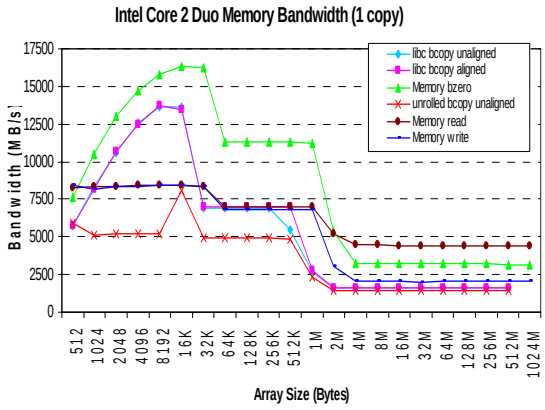
coherence within the chip and enables the fastest cache-to-cache transfer. Pentium D requires external FSB for cache-to-cache transfer. Athlon 64X2's on-chip system request inter-face and the MOESI protocol permits fast cache-to-cache communication.

We ran the bandwidth and latency test programs present in the *lmbench* suite. Figure 3.3 shows memory bandwidth for many operations from *lmbench*. Figure 3.3(a), 3.3(c) and 3.3 (e) present data collected while running one copy of *lmbench* on the three machines. Several observations can be made:

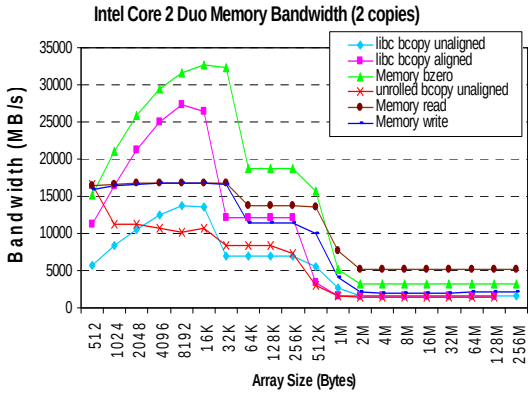
(1) In general, Core 2 Duo and Athlon 64 X2 have better bandwidth than that of Pentium D. Only exception is that Pentium D shows the best *memory read* bandwidth when the array size is less than 1MB. The shared cache of Core 2 Duo demands longer access latency though providing larger effective capacity. For Athlon 64X2, because the equipped DRAM has lower bandwidth, its *memory read* bandwidth is lower than that of Pentium D when memory bus is not saturated. The *memory read* bandwidth for the three machines drops when the array size is larger than 32KB, 16KB and 64KB respectively. These reflect the sizes of their L1 cache. When the array size is larger than 2MB, 1MB and 1MB for the respective three systems, we can see another dropping, reflecting their L2 cache sizes.

(2) The *memory bzero* operation shows different behaviors: when the array size is larger than their L1 data cache sizes, i.e., 32KB for Core 2 Duo and 64KB for Athlon 64X2, the memory bandwidth drops sharply. This is not true for Pentium D. The L1 cache of Core 2 Duo and Athlon 64X2 employ a write-back policy while the L1 cache of Pentium D uses a write-through policy. When the array size is smaller than their L1 data cache sizes, the write-back policy updates the L2 cache less frequently than the write-

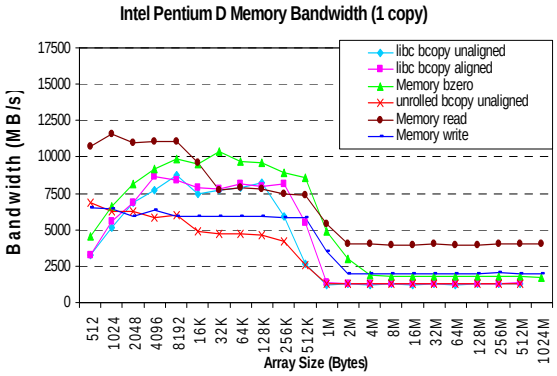
through policy, leading to higher bandwidth. However, when the array size is larger than their L1 data cache sizes, the write-back policy does not have any advantage as indicated



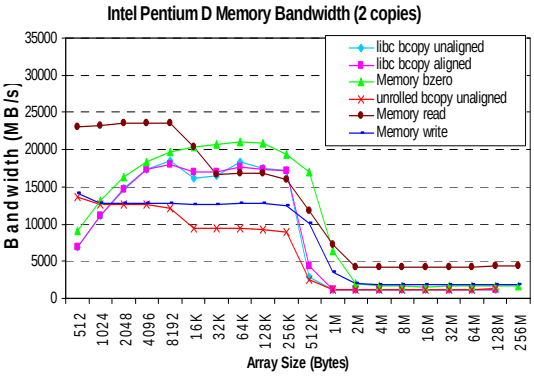
(a)



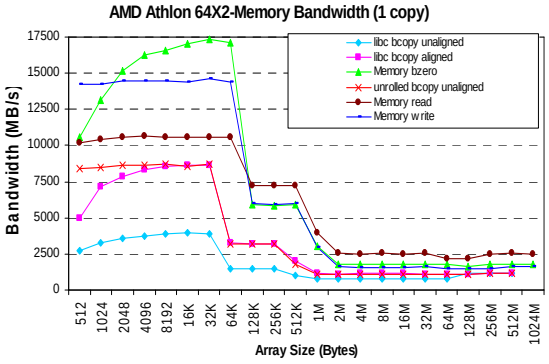
(b)



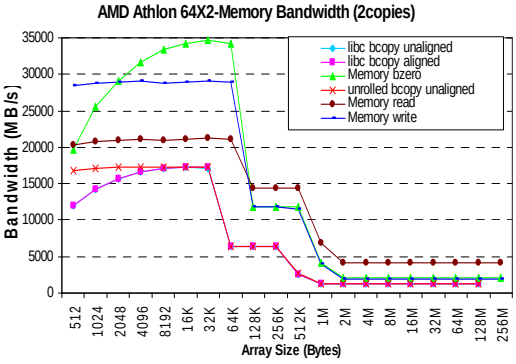
(c)



(d)



(e)



(f)

Figure 3-3 Memory bandwidth collected from the *lmbench* suite (1 or 2 copies).

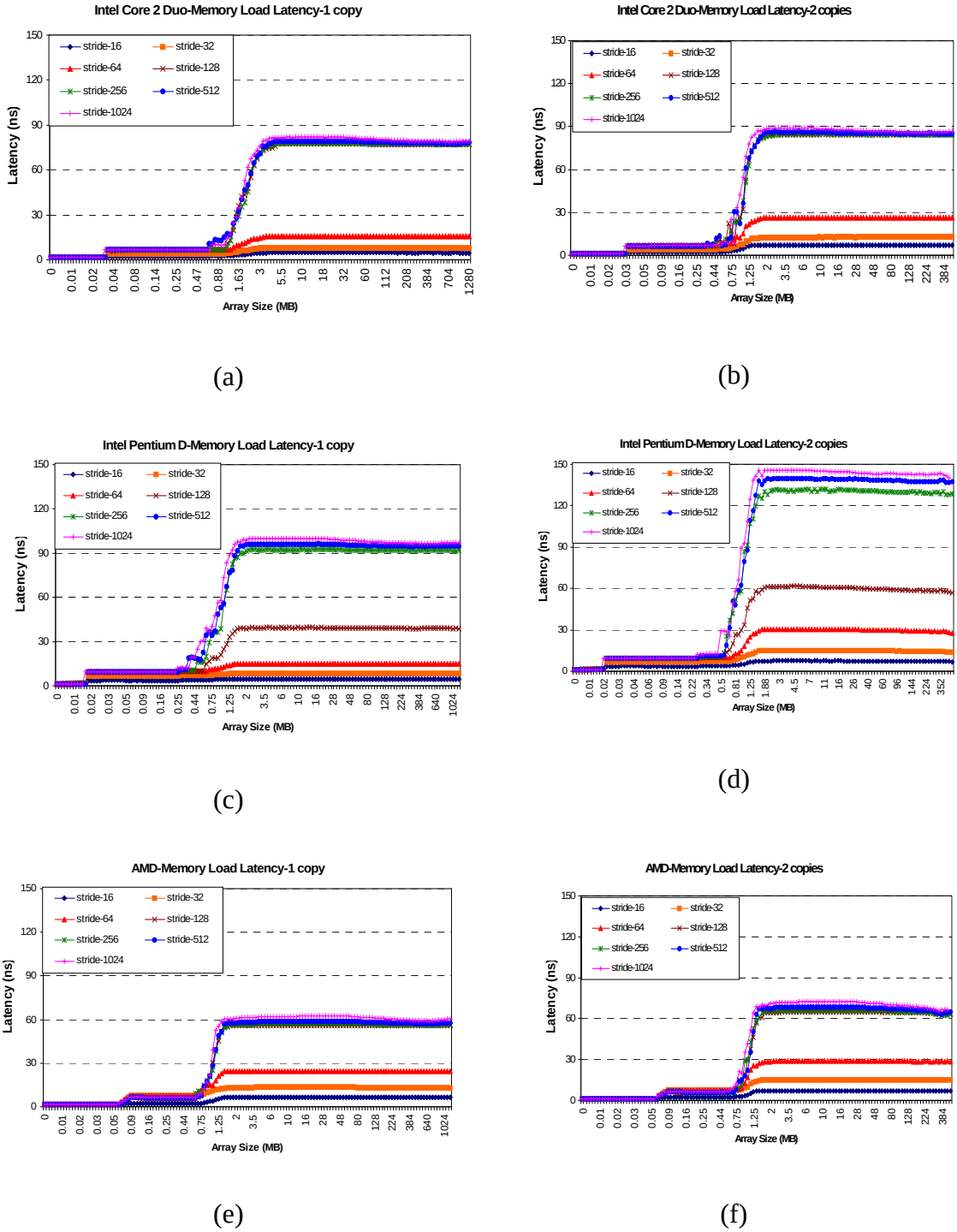
by the sharp decline of the bandwidth.

(3) For Athlon 64X2, *libc bcopy unaligned* and *libc bcopy aligned* show a big difference while alignment does not have much difference for Core 2 Duo and Pentium D. 'Aligned' here means the memory segments are aligned to the page boundary. The operation *bcopy* could be optimized if the segments are page aligned. In Figure 3.3(a), 3.3 (c) and 3.3 (e), Core 2 Duo and Pentium D show optimizations for *unaligned bcopy* access while Athlon 64X2 does not.

Figure 3.3 (b), 3.3 (d) and 3.3 (f) plot the bandwidth while running two copies of *lmbench* on three machines. The scale of the vertical axis of these three figures is doubled compared to their one-copy counterparts. We can observe that memory bandwidth of Pentium D and Athlon 64X2 are almost doubled for all operations. Core 2 Duo has increased bandwidth, but not doubled. This is because of the access contention when two *lmbench* copies compete with the shared cache. When the array size is larger than its L2 cache size 2MB, Athlon 64X2 provides almost doubled bandwidth for two-copy *lmbench memory read* operation compared with its one-copy counterpart. Athlon 64X2 benefits from its on-die memory controller and separate I/O Hyper-Transport. Intel Core 2 Duo and Pentium D processors suffer FSB bandwidth saturation when the array size exceeds the L2 capacity.

We tested memory load latency for multiple sizes of stride access and random access for all the three machines. Figure 3.4(a), 3.4 (c) and 3.4 (e) depict the memory load latency lines of the three machines running with one copy of *lmbench*. Several observations can be made: (1) For Core 2 Duo, latencies for all configurations jump after the array size is larger than 2 MB while for Pentium D and Athlon 64X2 latencies for all

the configurations jump after the array size is larger than 1MB. This relates to the L2 cache sizes of the measured machines. (2) As described in Section 2, when hardware



(a) (b)

(c) (d)

(e) (f)

Figure 3-4 Memory load latency collected from the *lmbench* suite (1 or 2 copies)

prefetchers on all machines work, the memory bus bottleneck will not be reflected. When the stride size is equal to 128 bytes, Pentium D still benefits partially from its hardware prefetcher but the L2 prefetchers of Core 2 Duo and Athlon 64X2 is not triggered. This leads to better performance for Pentium D. (3) When the stride size is large than 128 bytes, all hardware prefetchers don't take effect. Multiple L2 cache misses put pressures onto the memory buses. Athlon 64X2's on-die memory controller and separate I/O HyperTransport show the advantage. Pentium D's memory latency have a large jump for these operations but Athlon 64X2's latency almost keeps unchanged.

We increased pressure on memory hierarchy by running 2 copies of *lmbench* simultaneously. Figure 3.4(b), 3.4(d) and 3.4(f) show memory latencies of two *lmbench* copies. We found that Core 2 Duo and Athlon 64X2 show a slight increase in the latencies for stride sizes larger than 128 bytes while Pentium D's latencies in those situations increases a lot. Core 2 Duo benefits from its shared cache, which generates lower external traffic and its faster FSB while Athlon 64X2 take the advantage of on-chip memory controller and separate I/O Hyper-Transport. However, Pentium D's latencies jump due to suffering from memory bus saturation.

We also ran the *STREAM* and *STREAM2* benchmarks implemented in *lmbench* to measure memory bandwidth and latency of eight kernel operations. Figure 3.5(a) shows memory bandwidth of *STREAM* and *STREAM2* operations when running with a single copy of each operation. We made two observations. First, the *add* operation in the *STREAM* suite shows much higher bandwidth than other operations. After examining the related assembly code, we found that the *add* operation is a loop of *c[i] = a[i] + b[i]*, which can easily take advantage of the SSE2 packet operations. Other operations such as

copy and *fill* do not use SSE2 instructions and therefore do not show much difference. *Triad* and *daxpy* have longer delay and lower bandwidth for each step because of multiplication. Performance of the operation *sum* was hurt because of its inter-loop dependence: *s += a[i]*. Second, Intel Core 2 Duo shows the best bandwidth for all operations because of L1 data prefetchers and the faster Front Side Bus.

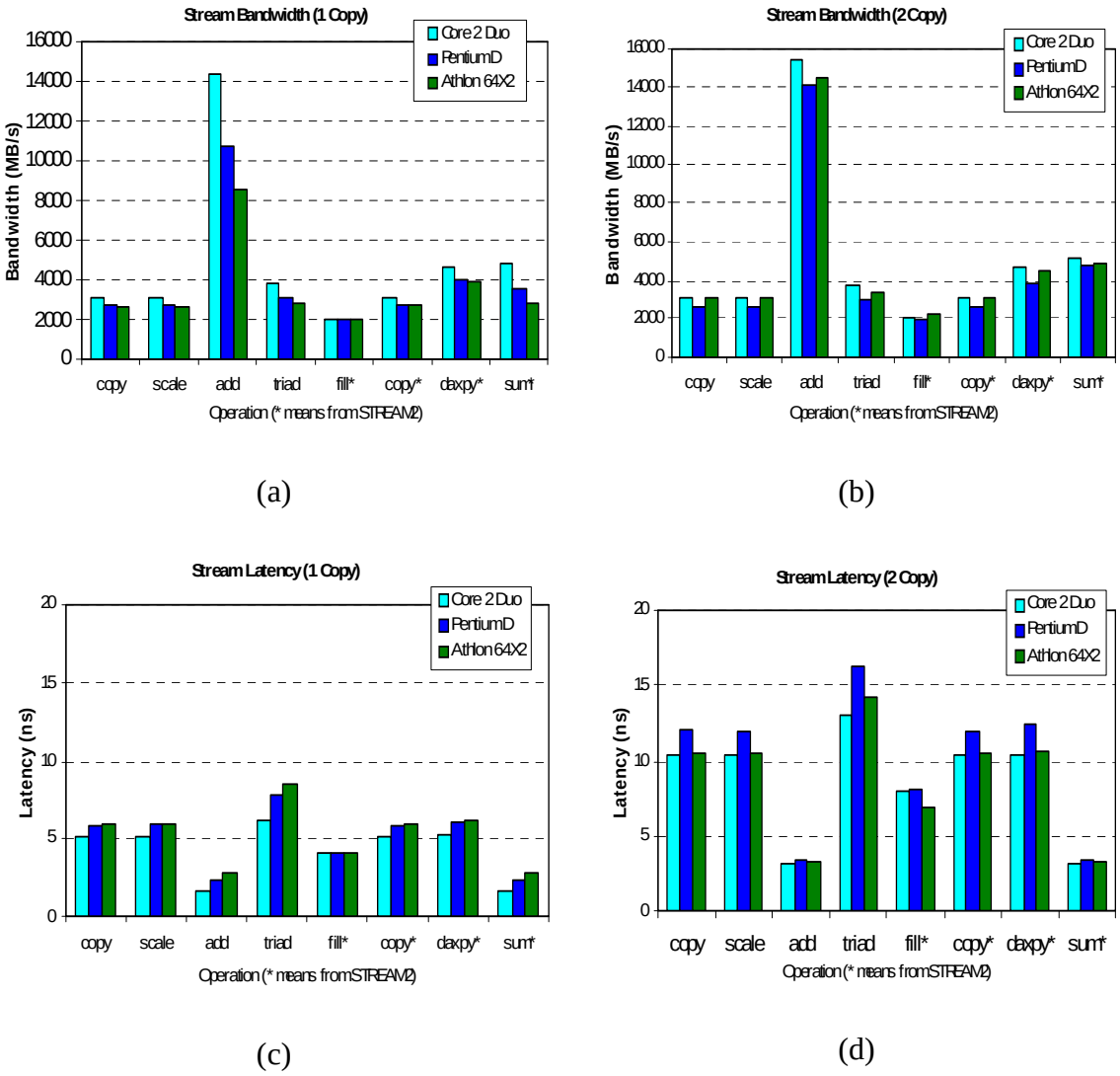


Figure 3-5 Memory bandwidth and latency collected from the *STREAM* and *STREAM2* benchmarks (1 or 2 copies)

Figure 3.5(b) depicts memory bandwidth when running with 2 copies of each operation in *STREAM / STREAM2*, one on each core. From this figure, we can see that Core 2 Duo and Athlon 64X2 have better bandwidth than that of Pentium D. This is due to the fact that Pentium D's FSB is saturated when running two copies of each operation. Athlon 64X2 benefits from its on-die memory controller and separate HyperTransport for I/O although its main memory DDR bandwidth is worse than that of Pentium D. Core 2 duo benefits from the presence of its L1 data prefetchers and the faster FSB. Figure 3.5(c) and 3.5(d) show the memory latencies for the three machines. Similar to the bandwidth figures, memory latency of Core 2 Duo and Pentium D are shorter than that of Athlon 64X2 when a single copy of the *STREAM/STREAM2* benchmark is running. Apparently, the shorter latency from on-die memory controller does not pay off in comparison with an off-die controller with better DRAM technology. However, while running the 2-copy version, memory latency of Pentium D is higher than the other two.

4. Performance Comparison of Dual Core Processors Using Multiprogrammed and Multithreaded Benchmarks

4.1 Overview

This section emphasizes on comparing the performance measurement results of three dual core desktop processors, explained in chapter 3 : Intel Core 2 Duo E6400 with 2.13GHz [15], Intel Pentium D 830 with 3.0GHz [19] and AMD Athlon 64X2 4400+ with 2.2GHz [2] using multi-programmed and multi threaded benchmarks.

To evaluate the architecture performance a mixture of single threaded and multi-programmed benchmarks are used. A set of single-threaded workloads is run on the three systems to determine the dual-core speedups over a single core. For single-thread programs, we experiment a subset of mixed SPEC CPU2000 and SPEC CPU2006 benchmarks [31]. To examine the scalability of single and dual cores, we run a set of single- and multi- threaded workloads on the three systems. For multi-threaded workloads, we select blastp and *hmmpfam* from the *BioPerf* suites [6], SPEC*jbb2005* [32], as well as a subset of *SPLASH2* [22].

4.2 Methodology

Similar to the methodology used in chapter 3 we used SUSE linux 10.1 with kernel 2.6.16-smp on all three machines for all our experiments in this section. We used maximum level GCC optimization to compile all the C/C++ benchmarks including SPEC CPU2000, SPEC CPU2006, *SPLASH2* and blastp and *hmmpfam* from *BioPerf*. SPEC *jbb2005* was compiled using SUN JDK 1.5.0.

For multiprogrammed workloads, the cross-product of mixed SPEC CPU2000/2006 benchmarks were run on the three machines to examine the dual-core

speedups over a single core. All the SPEC CPU2000/2006 programs were run with their respective ref inputs. In our simulations, when two programs were run together, we guaranteed that each program was repeated at least four times. The shorter programs may run more than four iterations until the longer program completes its four full iterations. We discarded the results obtained in the first run and used the average execution time and other metrics from the remainder three repeated runs to determine the speedups. We calculated the dual-core speedup for multiprogrammed workloads similarly to that used in [25]. Firstly, the single program's running time were collected individually and were considered as the base runtime. Secondly, the average execution time of each workload when run simultaneously was re-corded. Then, the dual-core speedup of each workload is calculated by finding the ratio of average run time when run individually (single core) by the average run-time when run together (dual core). Finally, we add the speedups of the two programs run together to obtain the dual-core speedup. For example, if the speedups of two programs are 0.8 and 0.9 when run simultaneously, the respective dual-core speedup will be 1.7.

We used the same procedure for homogeneous multi-threaded workloads including blastp and *hmmpfam* from the *BioPerf* suites, a subset of *SPLASH2*, as well as SPEC*jbb2005*. The *BioPerf* suite has emerging Bio-informatics programs. *SPLASH2* is a widely used scientific workload suite. SPEC*jbb2005* is a java based business database program. Table 4.1 lists the input parameters of the multithreaded workloads used. We ran each of these workloads long enough to compensate overheads of sequential portions of the workloads.

Table 4.1 Input parameters of the selected multithreaded workloads

Workload	Input parameters
blastp	Swissprot database, large input
hmmpfam	Large input
barnes	1048576 bodies
fmm	524288 particles
ocean-continuous	2050 X 2050 grid
fft	2^24 total complex data points transformed
lu-continuous	4096 X 4096 node matrix
lu-non-continuous	4096 X 4096 node matrix
radix	134217728 keys to sort
SPEC*jbb2005*	Default ramp up time 30s, measurement time 240s, from 1 to 8 warehouses

4.3 Multiprogrammed Workload Measurements

We measured execution time of a subset of SPEC CPU2000 and CPU 2006 benchmarks running on the three systems. In figure 5(a) and 5(c), the Core 2 Duo processor runs fastest for almost all workloads, especially for memory intensive workloads *art* and *mcf.*

Core 2 Duo has a wider pipeline, more functional units, and a shared L2 cache that provides bigger cache for single thread. Athlon 64X2 shows the best performance for *ammp*, whose working set is large, resulting in large amount of L2 cache misses for all three machines. Athlon 64X2 benefits from its faster on-chip memory controller.

Figure 4.1(b) and 4.1(d) depict average execution time of each workload when mixed with another program in the same suite. There is an execution time increasing for each workload. For memory bounded programs *art*, *mcf* and *ammp*, execution time increasing is large while CPU bounded workloads such as *crafty*, *mesa*, *perl* and *sjeng* show a little increasing.

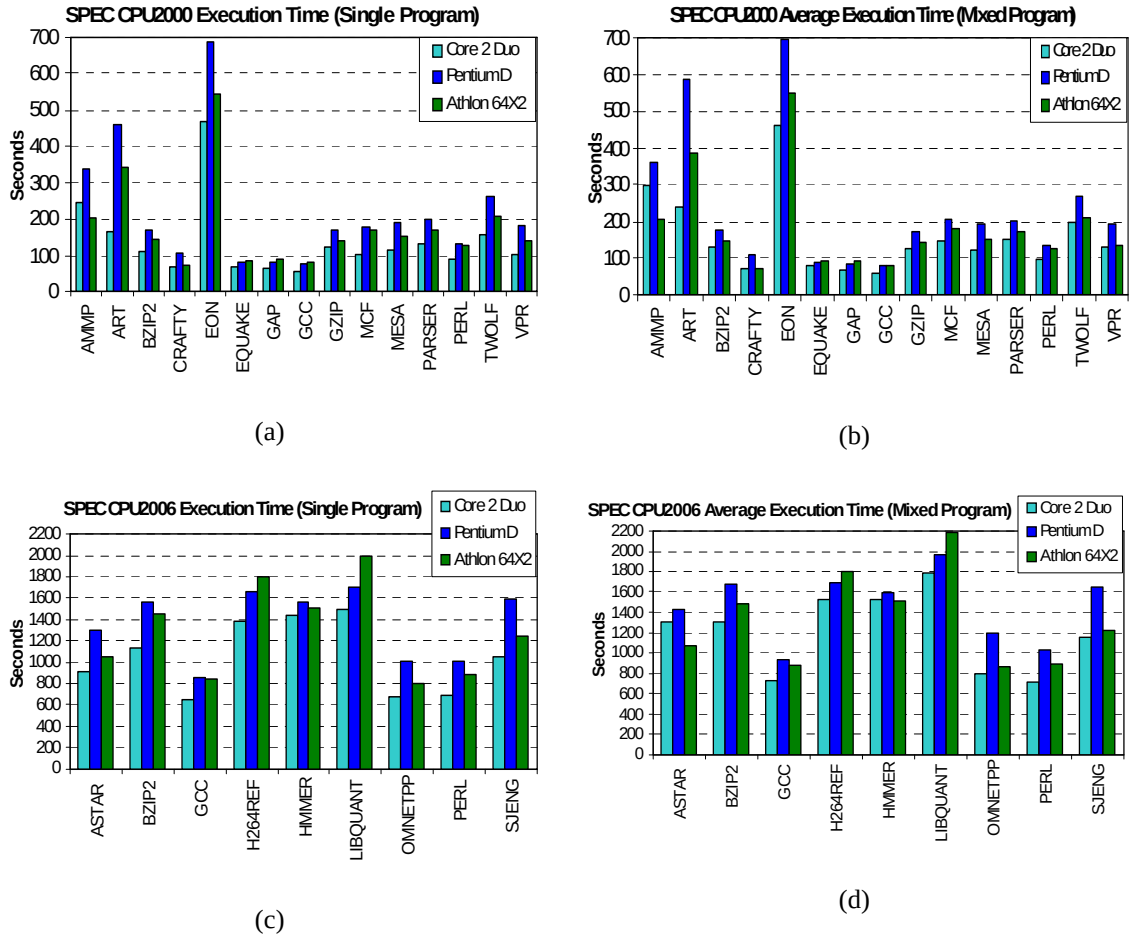


Figure 4-1 SPEC CPU2000 and CPU2006 benchmarks execution time

The multi-programmed speedup of the cross-product of mixed SPEC CPU2000 and CPU2006 programs for the three machines are given in the Figure 4.2, where C2D, PNT and ATH denote the measured Core 2 Duo, Pentium D, and Athlon 64X2 respectively. From Figure 4, we can see that Athlon 64X2 achieves the best speedup 2.0 for all the workloads. *Crafty*, *eon*, *mesa* in CPU 2000 and *perl* in CPU2006 have the best speedup when run simultaneously with other programs because they are CPU bounded instead of memory bounded programs which have comparatively very low L1 Data cache misses and hence do not conflict with the other program when running together. On the other hand, in most cases, *art* shows the worst speedup because it is a memory bounded

program. Its intensive L2 cache misses occupy the shared memory bus and block another program's execution. In the extreme case, when an instance of *art* was run against another *art*, the speedups were 0.82, 1.11 and 1.36 for Core 2 Duo, Pentium D and Athlon 64X2. Other memory bounded programs, *ammp* and *mcf*, present similar behaviors.

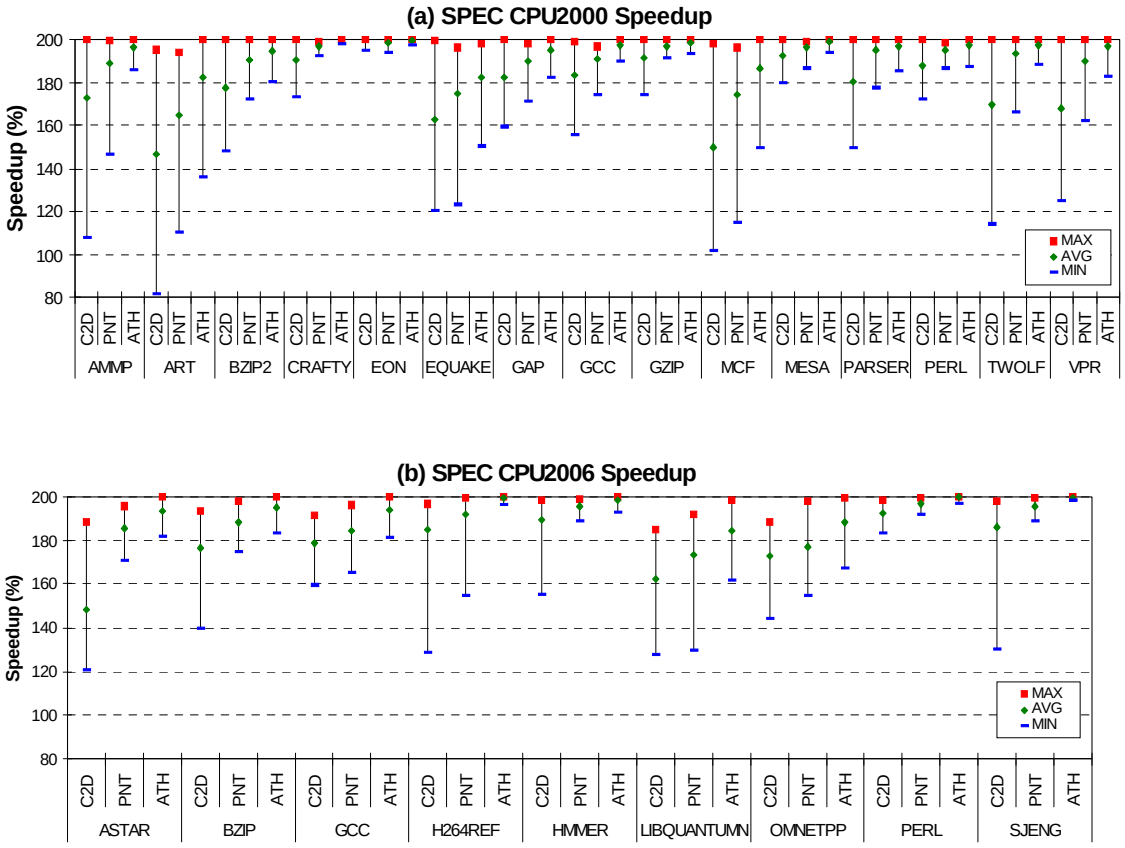
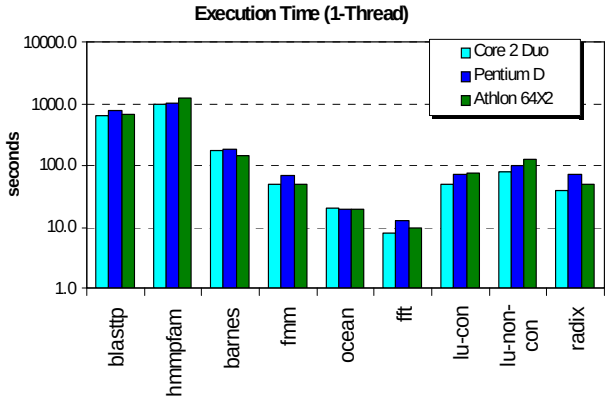


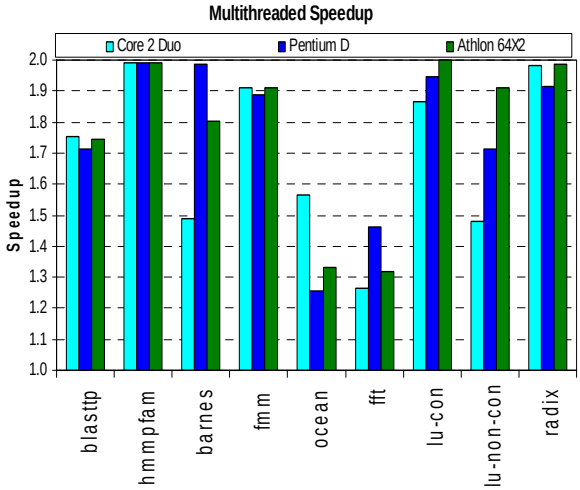
Figure 4-2 Multi-programmed speedup of mixed SPEC CPU 2000/2006 benchmarks.

Comparing the three machines, the multi-programmed Athlon 64X2 outperforms those of Core 2 Duo and Pentium D for almost all workload mixes. It is interesting to note that even though Core 2 Duo has better running time than the other two machines, the overall speedup is lesser. The reason again is due to its L2 shared cache.

4.4 Multithreaded Program Behavior



(a)



(b)

Figure 4-3 (a) Execution time for 1-thread version of selected multithreaded programs (b) Speedup for 2-thread version of selected multithreaded programs

The multithreaded program execution time and performance speedup of the three systems is presented. We selected blastp and *hmmpfam* from the *BioPerf* suite and a set of the *SPLASH2* workloads. Figure 4.3(a) and 4.3(b) illustrates execution time of single thread version of the programs and the speedup when running with 2-thread version. In general, Core 2 Duo and Athlon 64X2 do not show performance advantages on bioinformatics and scientific workloads because of less data communication between two

cores. Similar results were also reported on Multimedia programs [13]. Core 2 Duo shows the best speedup for *ocean* due to a large amount of cache-to-cache transfers [22]. We also verified this by Intel(R) VTune(TM) Performance Analyzer 8.0.1 [20]. According to the VTune measurements, the ratio of cache-to-cache transfers (snoop) to total loads and stores committed is over 15% which is the highest for these workloads. Pentium D shows the best speed up for *barnes* because of the low cache miss rate [22]. According to our measurement in chapter 3, the Pentium D processor shows the best *memory read* bandwidth when the array size is small. Bioinformatics workloads have high speedups for all three machines due to their small working sets [6].

Workloads with larger data sharing such as database programs benefit more from the cache-to-cache latency difference. We tested SPEC*jbb2005* on all the three machines. The throughput for different numbers of warehouses is shown in Figure 4.4. The throughput reaches its peak point when the number of warehouses equals to 2 due to the dual cores. In all cases, Core 2 Duo shows the best throughput due to its faster FSB and other memory design features. Scalability-wise, the throughput for 2 warehouses of Pentium D and Athlon 64X2 systems are, 1.78 and 1.88 of that for 1 warehouse respectively. The longer cache-to-cache latency in Pentium D accounts for the difference with Athlon 64X2. For the Core 2 Duo system the throughput for 2 warehouses is 1.71 times of that for 1 warehouse. The throughput ratio of Core 2 Duo's 2 warehouses version over 1 warehouse is relatively low because of the competence of its shared L2 cache.

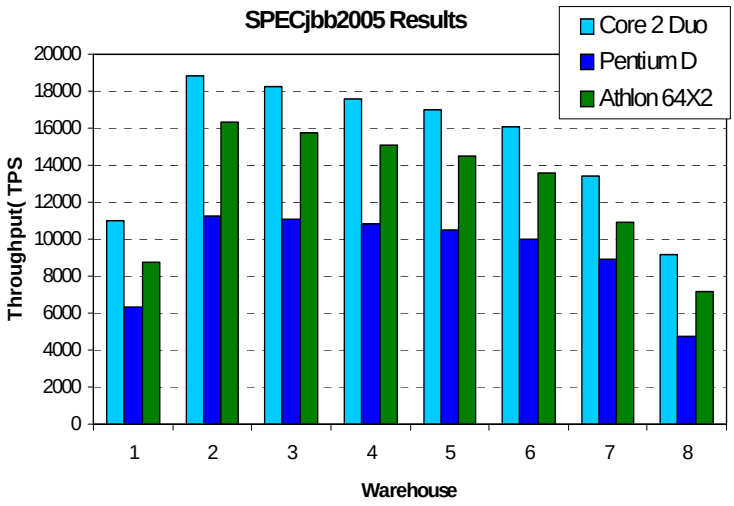


Figure 4-4 Throughput of SPEC*jbb2005* running with 1 to 8 warehouses.

From above studies, Core 2 Duo shows its performance advantages for workloads, such as ocean and SPEC*jbb2005*, with high data sharing. Basically, Athlon 64X2 doesn't show performance advantage over Pentium D for bioinformatics and scientific workloads though it has faster cache-to-cache data transfer

5. Related Work

Much of the early reports on performance of processors present the performance metrics such as running time and throughput without detailed analysis [13][26][27]. In this paper, we focus on the performance analysis of Intel Core 2 Duo processor and emphasize on the underlying reasons to better understand the design tradeoffs and causes for the performance bottlenecks.

Chip Multiprocessor (CMP) or multi-core technology was first reported in [14]. Companies such as IBM and SUN applied it on their server processors [22] [34]. In 2005, Intel announced to shelve its plan in pursuing higher frequency and instead switch to building multi-core processors [19]. Similarly, AMD also made the same decision about the same time [4].

Bekerman and Mendelson [12] used CPI metrics to analyze a Pentium based system. The CPI was broken into its basic constituents and the effect of various architectural features for different software environment was closely examined. Tuck and Tullsen [36] studied thread interactions on an Intel Pentium 4 Hyper-threading processor. They used multi-programmed and multithreaded workloads to measure speedup and synchronization and communication throughput. Bulpin and Pratt [9] measured an SMT processor with consideration about fairness between threads. They also showed the performance gap between SMP and Hyper-threaded SMT for multi-programmed workloads.

In [6], Yue Li, Tao Li, Kahveci and Fortes proposed and studied the workload charcacterization of a benchmark suite based on bioinformatics. To understand the impacts and implications of bioinformatics workloads on the microprocessor designs,

they compared the characteristics of bioinformatics workloads with the characteristics of SPEC 2000 integer benchmarks running under similar i/o environment. In a study carried by Birdj , Phansalkarj , John , Mericasa and Indukuru [8]; the behavior of CPU2006 benchmarks on the Intel's Woodcrest processor based on the Core microarchitecture. They analyzed the impact of new feature called "Macro-fusion" that reduces run time on the SPEC Benchmarks. They found that macro fusion shows significant improvement in performance for integer benchmarks.

There are several recent proposals to study the issues of CMP cache fairness and partitioning. In [38], the authors proposed and evaluated five different metrics such as shared cache miss rates, which can be correlated to execution time, used for CMP fairness and proposed static and dynamic caches partitioning algorithms that optimize fairness. This dynamic algorithm can help the operating system thread scheduling and to avoid thread thrashing. CMP cache capacity and latency optimization has recently been studied extensively. In [37], a victim replication scheme based on a shared cache to reduce latency was proposed. It tried to keep victims of L1 caches into the local L2 cache slices. In case of future reuses of the replicated victims, access latency would be reduced.

Peng, Peir, Prakash, Chen and Koppelman [30] studied memory performance scalability of Intel's and AMD's various dual core processor by reporting performance measurement results on three dual core desktop processors: Intel Core 2 Duo, Intel Pentium D and AMD Athlon 64X2. These measurements are parallel to my work done upon which the thesis is based.

6. Conclusion

In this paper, selected SPEC CPU2006 benchmarks and the memory hierarchy of selected Intel and AMD dual-core processors are analyzed. The IPC, instruction profile, L1 data cache miss rate, L2 cache miss rate and branch misprediction rate for the various SPEC CPU20006 benchmarks were first analyzed. Based on the analysis of CPU2006 benchmarks it was confirmed that CPU2006 benchmarks have large data set and take longer running time than CPU2000 benchmarks, resulting in stressing the processor architectures at full throttle.

The memory bandwidth and latency of Core 2 Duo, Pentium D and Athlon 64X2 was measured using *lmbench*. In general, Core 2 Duo and Athlon 64X2 have better memory bandwidth than that of Pentium D. Only exception is that Pentium D shows the best *memory read* bandwidth with small array size.

The individual execution time and the average execution time of each application when mixed with other programs on the dual cores of SPEC CPU2000 and CPU2006 were measured. In general, Core 2 Duo runs fastest for all single and mixed applications except for *ammp*. It was observed that memory intensive workloads such as *art*, *mcf* and *ammp* have worse speedups. From the measured cache-to-cache latencies it was determined that Core 2 Duo has the shortest cache-to-cache latencies. This generic memory performance behavior is consistent with the performance measurement results of multithreaded workloads such as SPECjbb with heavy data sharing between the two cores.

The Core 2 Duo has two distinct advantages: (1) faster core-to-core communication and (2) dynamic cache sharing between cores. Faster core-to-core

communication makes the Core 2 Dual the best for multi-threaded workloads with heavily data sharing or communication. However, to manage cache resource efficiently is a challenge especially when two cores have very different demands for caches. In summary, for the best performance and scalability, the following are important factors: (1) fast cache-to-cache communication, (2) large L2 or shared capacity, (3) fast L2 access delay, and (4) fair resource (cache) sharing. Three processors that we studied have shown benefits of some of them, but not all of them.

References

[1]. AMD, AMD Athlon 64X2 Dual-Core Processor Model Number and Feature Comparisons,http://www.amd.com/usen/Processors/ProductInformation/0,,30_118_9485 13041 %5E13076,00.html.

[2]. AMD, AMD Athlon 64X2 Dual-Core Product Data Sheet, http://www.amd.com/us-en/assets/content_type/ white_papers_and_tech_docs/33425.pdf.

[3]. AMD, AMD HyperTransport Technology, http://www.amd.com/us-en/Processors/DevelopWithAMD/ 0,,30_2252_2353,00.html.

[4]. AMD, Multi-core Processors: The Next Evolution in Computing, http://multicore.amd.com/WhitePapers/Multi-Core_Processors_WhitePaper.pdf, 2005.

[5]. AMD, Software Optimization Guide for AMD64 Proc-essors, Chap. 5, Page 105, www.amd.com/us-en/assets/ content_type/white_papers_and_tech_docs/25112.PDF.

[6]. D. Bader, Y. Li, T. Li, V. Sachdeva, *BioPerf*: A Bench-mark Suite to Evaluate High-Performance Computer Ar-chitecture on Bioinformatics Applications, in Proceedings of the 2005 IEEE International Symposium on Workload Characterization, Oct. 2005.

[7]. B. M. Beckmann and D. A. Wood, “Managing Wire Delay in Large Chip Multiprocessor Caches,” In Proceedings of the 37th International Symposium on Microarchitecture, pages 319-330, Dec. 2004.

[8]. Sarah Birdj , Aashish Phansalkarj , Lizy K. Johnj , Alex Mericasa and Rajeev Indukuru, “Performance Characterization of SPEC CPU Benchmarks on Intel's Core Microarchitecture based processor”, in the proceedings of 2007 SPEC Benchmark Workshop, Jan 2007.

[9]. J. R. Bulpin and I. A. Pratt, Multiprogramming Performance of the Pentium 4 with Hyper-threading, In Pro-ceedings of Third Annual Workshop on Duplicating, Deconstructing, and Debunking (WDDD), Jun. 2004.

[10]. D. Chandra, F. Guo, S. Kim, and Y. Solihin, “Predicting the Inter-thread Cache Contention on a Chip Multiproc-essor Architecture,” In Proceedings of the 11th International Symposium on High Performance Computer Ar-chitecture, pages 340-351, Feb. 2005.

[11]. J. Chang and G. S. Sohi. Cooperative Caching for Chip Multi-processors. In Proceedings of the 33nd Annual International Symposium on Computer Architecture, June 2006.

[12]. Michael Bekerman, Avi Mendelson, A Performance Analysis of Pentium Processor Systems, in Proceedings of the 1995 IEEE Micro, Oct 1995

[13]. F. Delattre and M. Prieur, Intel Core 2 Duo – Test, http://www.behardware.com/articles/623-16/intel-core-2-duo-test.html.

[14]. L. Hammond, B. A. Nayfeh and K. Olukotun, A Single-Chip Multiprocessor, IEEE Computer, Sep. 1997.

[15]. Intel, Announcing Intel Core 2 Processor Family Brand, http://www.intel.com/products/processor/core2/index.htm

[16]. Intel, IA-32 Intel Architecture Optimization Reference Manual, Chap. 6, Page 6-4,http://www.intel.com/design/ pentium4/manuals/248966.htm.

[17]. Intel, Inside Intel Core Microarchitecture and Smart Memory Access. http://download.intel.com/technology/ architecture/sma.pdf.

[18]. Intel, CMP Implementation in Systems Based on the Intel Core Duo Processor, http://www.intel.com/technology/itj/2006/voume10issue02/art02_CMP_Implementation/p03_implementation.htm.

[19]. Intel, Intel Pentium D Processor Product Information, http://www.intel.com/products /processor/pentium_d/.

[20]. Intel, Intel VTune Performance Analyzers, http://www.intel.com/cd/software/products/asmo-na/eng/vtune/239144.htm

[21]. J. M. Tendler, S. Dodson, S. Fields, H. Le, and B. Sin-haroy, “IBM eserver Power4 System Microarchitecture,” IBM White Paper, Oct. 2001.

[22]. S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta,” The *SPLASH-2* Programs: Characterization and Metho-dological Considerations”, in Proceedings of the 22nd Annual International Symposium on Computer Architecture (ISCA), pages 24-36, Jun. 1995.

[23]. N. Tuck and D. M. Tullsen.” Initial observations of the simultaneous multithreading Pentium 4 processor”,in Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques (PACT), pages 26-34, Sep. 2003.

[24]. J. D. McCalpin, Sustainable memory bandwidth in cur-rent high performance computers. Technical report, Sili-con Graphics, Oct. 1995.

[25]. J. D. McCalpin, The stream2 homepage. http://www.cs.virginia.edu/stream/stream2.

[26]. A. Mitrofanov, Dual-core processors, http://www.digital-daily.com/cpu/dualcore-cpu/index.htm.

[27]. www.motherboards.org, AMD Vesus Intel Battle of the Dual-Core CPUs, http://www.motherboards.Org/ re-views/ hardware/1513_2.html.

[28]. V. Romanchenko, Intel processors today and tomorrow, http://www.digital-daily.com/cpu/intel-roadmap/.

[29]. Michael S., How can we measure cache-to-cache trans-fer speed? http://www.aceshardware. com/forums/read_post.jsp? id=20676&forumid=2.

[30]. L. Peng, J-K. Peir, T. K. Prakash, Y-K. Chen, D. Koppelman, "Memory Performance and Scalability of Intel's and AMD's Dual-Core Processors: A Case Study," In Proceedings of *26th IEEE International Performance Computing and Communications Conference (IPCCC)*, New Orleans, LA, Apr. 2007.

[31]. SPEC, SPEC CPU2000 and CPU2006, http://www.spec.org/

[32]. SPEC, SPEC*jbb2005*, http://www.spec.org/*jbb2005*/

[33]. C. Staelin. *lmbench* --- an extensible micro-benchmark suite. HPL-2004-213. Dec. 2004, http://www.hpl.hp. com/techreports/2004/HPL-2004-213.pdf.

[34]. Sun Microsystems, "Sun's 64-bit Gemini Chip," Sun-flash, 66(4), Aug. 2003.

[35] Patrick Lester," A* Pathfinding for Beginners", July 18, 2005, http://www.gamedev.net/reference/articles/article2003.asp

[36] Jan Treibig, Simon Hausmann, Ulrich Rüde, "Performance analysis of the Lattice Boltzmann Method on x86-64 Architectures" in the proceeding of 18th Symposium Simulationstechnique ASIM 2005 Proceedings ,Sept 2005.

[37]. M. Zhang, K. Asanovic, "Victim Replication: Maximiz-ing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors," In Proceedings of the 32nd Interna-tional Symposium on Computer Architecture, pages 336-345, Jun. 2005.

[38] S. Kim, D. Chandra, Y. Solihin, "Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture," In Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques, pages 111-122, Sep. 2004.

Vita

Tribuvan Kumar Prakash was born in Bangalore, in the state of Karnataka, India. He graduated from High School in April 2000 with first class. In the Fall of 2000, he enrolled in the department of Electronics and Telecommunications at the Vemanna Institute of Technology (affiliated to Visweswariah Technical University, Karnataka) and graduated with a first class distinction in spring 2004 with a Bachelor of Engineering degree.

He then joined the Department of Electrical and Computer Engineering at Louisiana State University, Baton Rouge, to complete his master's, in the Fall of 2004. He worked with Unisys as an intern for the summer and fall semester of 2006. He will be graduating in August 2007 with the degree of Master of Science in Electrical Engineering.