Performance Evaluation of Multicore Processors

Fasiku A. I.¹, Oyinloye O. E.², Falaki S. O.³, Adewale O. S.⁴

¹ Computer Engineering Department, Ekiti State University, Ado - Ekiti, Ekiti State, Nigeria.
² Computer Science Department, Ekiti State University, Ado - Ekiti, Ekiti State, Nigeria.
³, ⁴ Computer Science Department, Federal University of Technology, Akure, Ondo State, Nigeria.

ABSTRACT

Multicore processors represent the latest significant development in microprocessor technology. Computer System Performance and Evaluation deal with the investigation of computer components (both hardware and software) with a view to establish the level of their performances. This research work carried out performance evaluation studies on AMD dual-core and Intel dual-core processor to know which of the processor has better execution time and throughput. The architecture of AMD and Intel duo-core processor were studied. SPEC CPU2006 benchmarks suite was used to measure the performance of AMD and Intel duo core processors. The overall execution and throughput time measurement of AMD and Intel duo core processors were reported and compared to each other. Results showed that the execution time of CQ56 Intel Pentium Dual-Core Processor is about 6.62% faster than AMD Turion II P520 Dual-Core Processor while the throughput of Intel Pentium Dual-Core Processor was found to be 1.06 times higher than AMD Turion (tm) II P520 Dual Core Processor. Therefore, Intel Pentium Dual-Core Processors exhibit better performance probably due to the following architectural features: faster core-to-core communication, dynamic cache sharing between cores and smaller size of level 2 cache.

Keywords: Intel Dual-Core Processor, AMD Dual-Core Processor, Computer Architecture and SPEC CPU2006

1. INTRODUCTION

Computer is a system which processes data according to a prescribed algorithm. It contains one or more programmable digital processors, also called Central Processing Units (CPU), memory for storing data, input and output devices. Modern microprocessors are among the most complex systems ever created by humans. Performance evaluation of a computing system consists of discovering and ascertaining the efficiency of computing systems, it may be, for example, the estimation of the performance of systems under construction, or monitoring that of existing one. It is the study and analysis of the flow of data and control information in computing systems (Adewale, 1997).

When we say one computer is faster than another, what we are actually saying is that a program runs in less time on one computer than the other. Computer users are actually interested in reducing response time or execution time and increase the throughput (the total amount of work done in a given time). In comparing design alternatives, it is often the practice to relate the performance of two different computers, say, X and Y. When computer X is faster than computer Y, this means that the response time or execution time is lower on X than on Y for the given task. Then, computer X is n times faster than computer Y.

Therefore,

\[ \frac{\text{Execution time of } Y}{\text{Execution time of } X} = n \]  

\[ n = \frac{\text{Performance of } Y}{\text{Performance of } X} \]  

(1.2)

The phrase "the throughput of X is 1.3 times higher than Y" signifies here that the number of tasks completed per unit time on computer X is 1.3 times the number completed on Y. The only consistent and reliable measure of performance is the execution time of real programs. (Patterson and Hennessy, 2007)

The performance of computer systems have been doubling every year for more than 40 years, a computer today with high performance processor is a thousand times more powerful than the one built in the early 70’s to mid 70’s. Technological improvements have been fairly steady, progress arising from better computer architectures has been consistent and sustaining the recent improvements in cost and performance will require continuous innovations in computer design (Patterson and Hennessy, 1996), (Falaki and Adewale, 2000). It is well-recognized fact that computer processors have increased in speed and decreased in cost at a tremendous rate for a very long time. This observation was first made popular by Gordon Moore in 1965, and is commonly referred to as the Moore’s Law. Specifically, Moore’s Law states that the advances in electronic manufacturing technology make it possible to double the number of transistors per unit area about every 12 to 18 months. It is this advancement that has fueled the phenomenal growth in computer speed and accessibility over more than five decades. Smaller transistors have made it possible to increase the number of transistors that can be applied to processor functions and reduce the distance signals must travel, allowing processor clock frequencies to soar. This simultaneously increases system performance and reduces system cost (Pase and Eckl, 2005).
Over the years, there has been a growing interest in designing microprocessor based on the notion of Instruction Level Parallelism (ILP). There are different approaches for exploiting ILP. One approach uses run – time scheduling to evaluate the data dependences and execute instructions concurrently. A microprocessor based on this technique is called a superscalar microprocessor. Another approach, commonly known as Very Long Instruction Word (VLIW) architecture, is entirely based on compile – time analysis to extract parallelism. Superscalar architectures and VLIW architectures both improve processor performance by increasing the CPI (Cycle Per Instruction) factor. These architectures exploit ILP by issuing more than one instruction per cycle. VLIW processors require a sophisticated compiler to extract ILP prior to execution, which results in code expansion. Superscalar architectures utilize dynamic scheduling to extract at run time, in addition to static scheduling (Adeoye, 2005). This lead to the development of Thread Level Parallelism (TLP), which explicitly represented by the use of multiple threads of execution that are inherently parallel.

Increasing the performance of single-threaded processors becomes increasingly difficult but, one way to enhance the performance of chip multiprocessors that has received considerable attention is the use of thread-level parallelism (Prabhu and Olukotun, 2006). If a high-performance processor is to realise its full potential, the compiler must re-order or schedule the object code at compile time. This scheduling creates groups of adjacent instructions that are independent and which therefore can be issued and executed in parallel at run time (Gordon, et. al. 1999). As high-performance uniprocessors have increased in complexity, further improvements in performance have become tremendously difficult to implement. The combined difficulties caused by increasing verification and validation times, wire delays, power dissipation and circuit unreliability have bolstered arguments against continuing to develop such highly centralized architectures. As a result, virtually every major manufacturer of high-performance processors has announced one or more chip multiprocessors (CMPs) (Prabhu and Olukotun, 2006).

In a multi-programmed environment, where the processors may be running independent tasks, each process is typically independent of other processes. A process is a segment of code that may be run independently; the state of the process contains all the information necessary to execute that program on a processor. It is also useful to be able to have multiple processors executing a single program and sharing the code and most of their address space. When multiple processes share code and data in this way, they are often called threads. Today, the term thread is often used in a casual way to refer to multiple loci of execution that may run on different processors, even when they do not share an address space. For example, a multi-threaded architecture actually allows the simultaneous execution of multiple processes, with potentially separate address spaces, as well as multiple threads that share the same address space (Patterson and Hennessy, 2007).

Therefore, a thread is a flow of control through a program with a single execution point. In order word, a thread is a separate process with its own instructions and data. A thread may represent a process that is part of a parallel program consisting of multiple processes, or it may represent an independent program on its own. Each thread has all the state (instructions, data, program counter (PC), register state, and so on) necessary to allow it to execute. Unlike instruction-level parallelism, which exploits implicit parallel operations within a loop or straight-line code segment, thread-level parallelism is explicitly represented by the use of multiple threads of execution that are inherently parallel. Although the amount of computation assigned to a thread, is called the grain size, is important in considering how to exploit thread-level parallelism efficiently, the important qualitative distinction from instruction-level parallelism is that thread-level parallelism is identified at a high level by the software system and that the threads consist of hundreds to millions of instructions that may be executed in parallel. This research work, we are try to compare the performance of AMD Turion II P520 Dual-Core Processor and Intel Pentium Dual-Core Processor, in relation to the overall execution time and throughput, of the two processors.

2. PROCESSOR PERFORMANCE EQUATION

Essentially all computers are constructed using a clock running at a constant rate. These discrete time events are called ticks, clock ticks, clock periods, clocks, cycles, or clock cycles. Computer designers refer to the time of a clock period by its duration (e.g., 1 ns) or by its rate (e.g., 1 GHz). CPU time for a program can then be expressed in two ways:

\[ \text{CPU time} = \text{CPU clock cycles for a program} \times \text{Clock cycle time} \]

Or

\[ \text{CPU time} = \frac{\text{CPU clock cycles for program}}{\text{Clock rate}} \]

In addition to the number of clock cycles needed to execute a program, we can also count the number of instructions executed; the instruction path length or instruction count (IC). If we know the number of clock cycles and the instruction count, we can calculate the average number of clock cycles per instruction (CPI). Though, it is easier to work with, and designers sometimes use instructions per clock (IPC), which is the inverse of CPI. CPI is computed as

\[ \text{CPI} = \frac{\text{CPU clock cycles for program}}{\text{Instruction count}} \]

This processor provides insight into different styles of instruction sets and implementations. By transposing instruction count in the above formula, clock cycles can be defined as IC x CPI. This allows us to use CPI in the execution time formula:

\[ \text{CPU time} = \text{Instruction count} \times \text{Cycles per instruction} \times \text{Clock cycle time} \]

Or

\[ \text{CPU time} = \frac{\text{Instruction count} \times \text{Clock cycle time}}{\text{Clock rate}} \]

Expanding the first formula into the units of measurement and inverting the clock rate shows how the pieces fit together:
As this formula demonstrates, processor performance is dependent upon three characteristics: clock cycle (or rate), clock cycles per instruction and instruction count (Patterson and Hennessy, 2007).

Unfortunately, it is difficult to change one parameter in complete isolation from others because the basic technologies involved in changing each characteristic are interdependent in the following ways:

i. Clock cycle time is based on Hardware technology and organization

ii. CPI is based on Organization and instruction set architecture

iii. Instruction count is based on Instruction set architecture and compiler technology

Therefore, many potential performance improvement techniques primarily improve one component of processor performance with small or predictable impacts on the other two. Sometimes it is useful in designing the processor to calculate the number of total processor clock cycles as:

$$\text{CPU clock cycles} = \sum_{i=1}^{n} IC_i \times CPI_i \quad \text{---------- (2.5)}$$

where, $IC_i$ - represents number of times instruction, $i$ - is executed in a program and $CPI_i$ - represents the average number of clocks per instruction for instruction $i$.

This form can be used to express CPU time as

$$\text{CPU time} = (\sum_{i=1}^{n} IC_i \times CPI_i) \times \text{Clock cycle time} \quad \text{--- (2.6)}$$

And overall CPI as

$$\text{CPI} = \frac{\sum_{i=1}^{n} IC_i \times CPI_i}{\text{Instruction count}} = \sum_{i=1}^{n} \frac{IC_i}{\text{Instruction count}} \times CPI_i \quad \text{--- (2.7)}$$

The latter form of the CPI calculation uses each individual $CPI_i$ and the fraction of occurrences of that instruction in a program (i.e., $IC_i \div \text{Instruction count}$). $CPI_i$ should be measured and not just calculated from a table in the back of a reference manual since it must include pipeline effects, cache misses, and any other memory system inefficiencies (Patterson and Hennessy, 2007).

3. BENCHMARKS SUITES

Benchmarks metrics to be used for performance evaluation have always been an interesting and a controversial issue. There has been a lot of improvement in benchmark suites since 1988. SPEC originally created a benchmark set focusing on processor performance (initially called SPEC89), which has evolved into its fifth generation: SPEC CPU2006, which follows SPEC2000, SPEC95, SPEC92, and SPEC89. SPEC CPU2006 consists of a set of 12 integer benchmarks (CINT2006) and 17 floating-point benchmarks (CFP2006) (SPEC, 2006). Before, computer performance evaluation have been with small benchmarks such as kernels extracted from applications (eg: Lawrence Livermore Loops), Dhrystone and Whetstone benchmarks, Linpack, Sorting, Sieve of Eratosthenes, 8-queens problem, Tower of Hanoi, etc. (John, 2003). The Standard Performance Evaluation Corporation (SPEC) consortium and the Transactions Processing Council (TPC) formed in 1980s have made available several benchmark suites and benchmarking guidelines to improve the quality of benchmarking.

The best choices of benchmarks to measure performance are real applications, such as a compiler, compiler is program that translates source code into object code. Attempts at running programs that are much simpler than a real application have led to performance pitfalls. Examples include:

a) Kernels, which are small, key pieces of real applications;

b) Toy programs, Toy benchmark are typically between 10 and 100 lines of code and produce a result the user already knows before running the toy program.

c) Synthetic benchmarks, which are fake programs invented to try to match the profile and behaviour of real applications, such as Dhrystone.

All these three are discredited today, usually because the compiler writer and architect can conspire to make the computer appear faster on these stand-in programs than on real applications (Patterson and Hennessy, 2007).

Another issue is the conditions under which the benchmarks are run. One way to improve the performance of a benchmark has been with benchmark specific flags; these flags often caused transformations that would be illegal on many programs or would slow down performance on others. To restrict this process and increase the significance of the results, benchmark developers often require the vendor to use one compiler and one set of flags for all the programs in the same language (C or FORTRAN). In addition to the question of compiler flags, another question is whether source code modifications are allowed. There are three different approaches to addressing this question:

a) No source code modifications are allowed.

b) Source code modifications are allowed, but are essentially impossible. For example, database benchmarks rely on standard database programs that are tens of millions lines of code. The database companies are highly unlikely to make changes to enhance the performance for one particular computer.
c) Source modifications are allowed, as long as the modified version produces the same output.

The key issues that benchmark designers face in deciding modification of the source is whether such modifications will reflect real practice and provide useful insight to users or whether such modifications simply reduce the accuracy of the benchmarks as predictors of real performance.

To overcome the danger of placing too many eggs in one basket, collections of benchmark applications, called benchmark suites, are a popular measure of performance of processors with a variety of applications. Of course, such suites are only as good as the constituent individual benchmarks. Nonetheless, a key advantage of such suites is that the weakness of any one of the benchmark is lessened by the presence of the other benchmarks. The goal of a benchmark suite is that it will characterize the relative performance of two computers, particularly for programs not in the suite that customers are likely to run. (SPEC, 2006)

4. ARCHITECTURE OF SINGLE CORE PROCESSOR

In 1945, mathematician John von Neumann, with the aid of J. Presper Eckert and John Mauchly, wrote a memo proposing the creation of an Electronic Discrete Variable Automatic Computer, more famously known as the EDVAC. In that paper, von Neumann suggested the stored-program model of computing. In the von Neumann architecture, the program is a sequence of instructions stored sequentially in the computer’s memory. The program’s instructions are executed one after the other in a linear, single-threaded fashion.

The last 30 years have seen the computer industry driven primarily by faster and faster uniprocessors, those days have come to a close. Emerging in their place are microprocessors containing multiple processor cores that are expected to exploit thread-level parallelism. Pentium 4 processor brand, refers to Intel's line of single-core desktop and laptop Central Processing Units (CPUs) introduced on November 20, 2000 at a speed rate of 1.5GHZ and shipped through August 8, 2008. They are the 7th-generation X86 micro architecture, called NetBurst, which was the company's latest design of single core processor designed since the introduction of P6 micro architecture of the Pentium Pro CPUs in 1995. NetBurst differed from the preceding P6 (Pentium III, II, etc.), featuring a very deep instruction pipeline to achieve very high clock speeds (up to 3.8 GHz). The first Pentium 4 cores, codenamed Willamette, were clocked from 1.3 GHz to 2 GHz, Pentium 4 CPUs introduced the SSE2 and, in the Prescott-based Pentium 4's, Streaming SIMD Extensions 3 (SSE3) instruction sets to accelerate calculations, transactions, media processing, 3D graphics, and games (Patterson and Hennessy, 2007). This processor requires balancing and tuning of many microarchitectural features that compete for processor die cost and for design and validation efforts. Figure 4.1 shows the basic Intel NetBurst microarchitecture of Pentium 4 shown in figure 4.1.

a) The in-order front end
b) The out-of-order execution engine

The in-order front end is part of the machine that fetches the instructions to be executed next in the program and prepares them to be used later in the machine pipeline. Its job is to supply a high-bandwidth stream of decoded instructions to the out-of-order execution core, which will do the actual completion of the instructions. The front end has highly accurate branch prediction logic that uses the past history of program execution to speculate where the program is going to execute next. The predicted instruction address, from this front-end branch prediction logic, is used to fetch instruction bytes from the Level 2 (L2) cache. These IA-32 instruction bytes are then decoded into basic operations called micro-operations (micro-operations) that the execution core is able to execute (Hinton et al., 2001).

The NetBurst microarchitecture has an advanced form of a Level 1 (L1) instruction cache called the Execution Trace Cache. Unlike conventional instruction caches, the Trace Cache sits between the instructions decode logic and the execution. In this location the Trace Cache is able to store the already decoded IA-32 instructions or micro-operations (uops). Storing already decoded instructions removes the IA-32 decoding from the main execution loop. Typically the instructions are decoded once and placed in the Trace Cache and then used repeatedly from there like a normal instruction cache on previous machines. The IA-32 instruction decoder is used only when the machine misses the Trace Cache and needs to go to the L2 cache to get and decode new IA-32 instruction bytes.

The front-end decoder translates each IA-32 instruction to a series of micro-operations (uops), which are similar to typical RISC instructions. The micro-operations are then executed by a dynamically scheduled speculative pipeline. A trace cache is a type of instruction cache that holds sequences of instructions to be executed including nonadjacent instructions separated by branches; a trace cache tries to exploit the temporal sequencing
of instruction execution rather than the spatial locality exploited in a normal cache.

The Pentium 4’s execution trace cache is a trace cache of microoperations, corresponding to the decoded IA-32 instruction stream. By filling the pipeline from the execution trace cache, the Pentium 4 avoids the need to redecode IA-32 instructions whenever the trace cache hits. Only on trace cache misses are IA-32 instructions fetched from the L2 cache and decoded to refill the execution trace cache. Up to three IA-32 instructions may be decoded and translated every cycle, generating up to six micro-operations; when a single IA-32 instruction requires more than three micro-operations, the micro-operations sequence is generated from the microcode ROM. The execution trace cache has its own branch target buffer, which predicts the outcome of micro-operations branches. After fetching from the execution trace cache, the micro-operations are executed by an out-of-order execution (Patterson and Hennessy, 2007).

b) Out-Of-Order Execution Logic

The out-of-order execution engine is where the instructions are prepared for execution. The out-of-order execution logic has several buffers that they use to smooth and re-order the flow of instructions to optimize performance as they go down the pipeline and get scheduled for execution. Instructions are aggressively reordered to allow them to execute as quickly as their input operands are ready. This out-of-order execution allows instructions in the program following delayed instructions to proceed around them as long as they do not depend on those delayed instructions. Out-of-order execution allows the execution resources such as the ALUs and the cache to be kept as busy as possible executing independent instructions that are ready to execute.

The retirement logic is what reorders the instructions, executed in an out-of-order manner, back to the original program order. This retirement logic receives the completion status of the executed instructions from the execution units and processes the results so that the proper architectural state is committed (or retired) according to the program order. The Pentium 4 processor can retire up to three micro-operations per clock cycle. This retirement logic ensures that exceptions occur only if the operation causing the exception is the oldest, non-retired operation in the machine. This logic also reports branch history information to the branch predictors at the front end of the machine so they can train with the latest known-good branch-history information (Hinton et al., 2001).

c) Integer and Floating-Point Execution Units

The execution units are where the instructions are actually executed. This section includes the register files that store the integer and floating-point data operand values that the instructions need to execute. The execution units include several types of integer and floating-point execution units that compute the results and also the L1 data cache that is used for most load and store operations.

d) Memory Subsystem

The memory subsystem includes the L2 cache and the system bus. The L2 cache stores both instructions and data that cannot fit in the Execution Trace Cache and the L1 data cache. The external system bus is connected to the backside of the second-level cache and is used to access main memory when the L2 cache has a cache miss, and to access the system I/O resources.

5. MULTICORE ARCHITECTURES

Computer architectures are approaching physical and technological barriers which make increasing the speed of a single core exceedingly difficult and economically infeasible. As a result, hardware architects have begun to design microprocessors with multiple processing cores that operate independently and share the same address space. It appears that the advent of these multi-core architectures will finally force a radical change in how applications are programmed. Specifically, developers must consider how to direct the collaboration of many concurrent threads of execution to solve a single problem. A multi-core processor is composed of two or more independent cores. One can describe it as an integrated circuit which has two or more individual processors called cores. Manufacturers typically integrate the cores into a single integrated circuit die (known as a chip multiprocessor or CMP). A dual-core processor contains two cores (Such as AMD Phenom II X2, AMD Turion II P520 Dual-Core, Intel Pentium Dual-Core and Intel Core Duo), a quad-core processor contains four cores (Such as the AMD Phenom II X4 and Intel 2010 core line, which includes 3 levels of quad core processors), and a hexa-core processor contains six cores (Such as the AMD Phenom II X6 or Intel Core i7 Extreme Edition 980X). A multi-core processor implement multiprocessing in a single physical package.

There are several variations in how cache and memory are approached in the new Chip Multiprocessors. The approaches of processor-to - processor communication vary among different implementations. The Chip Multiprocessors implementations from the major chip manufacturers on which each processor handle the Input/Output (I/O) bus and the Front Side Bus (FSB) differently. Though, most of these differences are not visible when looking strictly at the logical view of an application that is being designed to take advantage of a multicore architecture. Figure 5.1 illustrates three common architectural configurations that support multiprocessing (Hughes, 2008).

a) Configuration 1 in Figure 5.1 uses hyperthreading. Like Chip Multiprocessors, an hyperthreaded processor allows two or more threads to execute on a single chip. However, in a hyperthreaded package the multiple processors are logical instead of physical. There is some duplication of hardware but not enough to qualify a separate physical processor. So hyperthreading allows the processor to present itself to the operating system as complete multiple processors when in fact there is a single processor running multiple threads.
b) Configuration 2 in Figure 5.1 is the classic multiprocessor. In this case, each processor is on a separate chip with its own hardware.

c) Configuration 3 represents the current trend in multiprocessors. It provides complete processors on a single chip.

What important to remember is that each configuration presents itself to the developer as a set of two or more logical processors capable of executing multiple tasks concurrently. The challenge for system programmers, kernel programmers, and application developers is to know when and how to take advantage of these architectural configurations. The dual processor architecture used in this research belongs to the configuration 3 above.

6. PERFORMANCE MEASUREMENT AND DISCUSSION OF RESULTS

6.1 Analysis of the Benchmarks used

Benchmark programs used for measurement of system performance should be derived from how actual applications will execute. However, performance is often the result of combined characteristics of a given computer architecture and system hardware/software components in addition to microprocessor. Other factors such as operating system, compilers, libraries, memory design and input/output subsystem characteristics may also have impacts on the results and make comparisons difficult.

SPEC CPU2006 developed twelve integers and seventeen floating-point-intensive application programs used to analyse/measure the performance of computer system. The SPEC CPU2006 benchmark consists of two benchmark suites, which focuses on a different aspect of compute-intensive performance. CINT2006 measure and compare compute-intensive integer performance, while CFP2006 measures and compares compute-intensive floating-point performance. A “rate” version of each, which runs multiple instances of the benchmark to assess System Under Test (SUT) throughput. Though, for the purpose of this project we limit our performance measurement to CINT2006 suite i.e we ran only the CINT2006 SPECint_base benchmark, which comprises of 12 integer-compute-intensive codes; 9 in C and 3 in C++, shown below:

1. 400.perlbench (C)
2. 401.bzip2 (C)
3. 403.gcc (C)
4. 429.mcf (C)
5. 445.gobmk (C)
6. 456.hmmer (C)
7. 458.sjeng (C)
8. 462.libquantum (C)
9. 464.h264ref (C)
10. 471.omnetpp (C++)
11. 473.astar (C++)
12. 483.xalancbmk (C++)

These programs (benchmarks in the CINT2006 suite) were run on Hewlet-Packard (HP) Intel Pentium(R) Dual-Core Processor and Hewlet-Packard (HP) AMD Turion(tm) Premium Vision Dual-Core Processor to compare the execution time (speed) and throughput of the processors.

6.2 SPEC CPU2006 Configuration

SPEC’s standard instructions for building the CINT2006 executable were followed. On studying the best results for these benchmarks from the SPEC configuration files, linux32-i386-gcc42.cfg software was used. The benchmark required configuration was copied and modified to reflect the
appropriate system information about the computer system under test.

SPEC adopted a set of rules defining how SPEC CPU2006 benchmark suites must be built and run to produce base metrics. Base metrics must be produced by building all the benchmarks in the suite with a common set of optimizations. In addition to the general benchmark and base optimization rules must be adhered to strictly to produce a valid result.

With the release of SPEC CPU2006 suites, a set of tools based on GNU Make and Perl5 are supplied to build and run the benchmarks. To produce publication-quality results, these SPEC tools were used. This helps ensure reproduction of results by requiring that all individual benchmarks in the suites are run in the same way and that a configuration file is available that defines the optimizations used. The primary tool is called runspec (install.bat for Microsoft Windows and % . /install.sh for Linux or Unix). It is fully described in the runspec documentation on SPEC web site at http://www.spec.org/cpu2006/Docs/runspec.html.

To generate a valid result for tested computer system, the two systems comparing their performance must have the same or similar configuration. Therefore, the configuration of two System Under Test (SUT) for this project were shown in table 6.1. The systems are HP Pavilion dv5, AMD Turion II P520 Dual-Core Processor and HP Presario CQ56 Intel Pentium Dual-Core Processor.

<table>
<thead>
<tr>
<th>SYSTEM</th>
<th>HP PAVILION DV5, AMD TURION (TM) II P520 DUAL CORE</th>
<th>PRESARIO CQ56 INTEL PENTIUM DUAL-CORE PROCESSOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Vendor</td>
<td>AMD</td>
<td>Intel</td>
</tr>
<tr>
<td>Model</td>
<td>Pavilion dv 5 Notebook</td>
<td>Presario CQ56 Notebook</td>
</tr>
<tr>
<td>Name</td>
<td>Turion (tm) II P520</td>
<td>Pentium (R) T4500</td>
</tr>
<tr>
<td>Core Frequency (GHZ)</td>
<td>2.30GHz</td>
<td>2.30GHz</td>
</tr>
<tr>
<td>Number of Processor Package</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Number of Threads</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Processor Type</td>
<td>AMD Turion (tm) II P520 Dual Core Processor</td>
<td>Pentium (R) Dual-Core</td>
</tr>
<tr>
<td>Platform</td>
<td>BIOS Name and Version</td>
<td>HP BIOS F.03</td>
</tr>
<tr>
<td>BIOS Setting</td>
<td>Default</td>
<td>Default</td>
</tr>
<tr>
<td>Memory Module(s)</td>
<td>Vendor and Model Numbers</td>
<td>ECPIDA TAIWAN and EB121UE8B5SO-DJ-J</td>
</tr>
<tr>
<td>Type</td>
<td>PC3-10600S-9-10-F1</td>
<td>PC2-6400S-666-12</td>
</tr>
<tr>
<td>Size</td>
<td>4 GB</td>
<td>4 GB</td>
</tr>
<tr>
<td>Number of RAM Modules</td>
<td>2 X 2.048 MB</td>
<td>2 X 2.048 MB</td>
</tr>
<tr>
<td>Chip Organisation</td>
<td>Double-sided</td>
<td>Double-Sided</td>
</tr>
<tr>
<td>Hard Disk</td>
<td>Vendor and Model Numbers</td>
<td>Seagate and ST93015A</td>
</tr>
<tr>
<td>Type</td>
<td>Serial ATA-150 - 7200.0 rpm</td>
<td>SATA 3.0Gb/s TS55AA320</td>
</tr>
<tr>
<td>Size</td>
<td>320 GB</td>
<td>320 GB</td>
</tr>
<tr>
<td>Number of Disk in System</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Operating System</td>
<td>Name</td>
<td>Fedora 10.0</td>
</tr>
<tr>
<td>File System</td>
<td>EXT 3</td>
<td>EXT 3</td>
</tr>
<tr>
<td>Kernel</td>
<td>2.6.27.5-117.fc10.i686</td>
<td>2.6.27.5-117.fc10.i686</td>
</tr>
<tr>
<td>Language</td>
<td>English</td>
<td>English</td>
</tr>
</tbody>
</table>
6.3 Discussions of Results

The actual test results consist of the execution times, ratios for the individual benchmarks and the overall SPEC metric produced by running the benchmarks via the SPEC tools. It required the use of the SPEC tools and ensures that the results generated are based on benchmarks built, run, and validated according to the SPEC run rules. The measurement component for each SPEC CPU2006 suite generates SPECint_base2006. The execution times in seconds for each of the benchmarks in the CINT2006 suite are generated and the ratio of the System Under Test (SUT) are calculated. The SPECint_base2006 metrics are calculated as a Geometric Mean of the individual ratios, where each ratio is based on the median execution time from the three runs.

The throughput metrics are calculated based on the execution of benchmark binaries that are built using the same rules as binaries built for speed metrics. However, the tester may select the number of concurrent copies of each benchmark to be run. The same number of copies must be used for all benchmarks in a base test.

SPEC wrote nine of the applications in C and three in C++. A CINT2006 run and performs each of the twelve applications (tasks) three times and also reports the median for each. It also calculated the geometric mean of those 12 applications to produce an overall score i.e the throughput.

6.3.1 Result of Execution Time of the two Systems Under Test

The Execution time of HP Pavilion dv5, AMD Turion II P520 Dual-Core Processor generated from SPEC CINT2006 suite are shown in table 6.2 and the graph chart in figure 6.1, while table 6.3 and figure 6.2 showed the Execution time and graph generated for HP Presario CQ56 Intel Pentium Dual-Core Processor. Execution time is the time between the start and the completion of an event i.e the process of carrying out an instruction within a specific time in any computer system. The median value from the three result generated is underline and bold in the tables.
Table 6.3 The Execution times of HP Intel Pentium (R) Dual-Core Processor

Comparing the AMD Dual-Core and Intel Dual-Core Processor Execution Times

6.3.2 Comparing AMD and Intel Processor Execution Time

Table 6.2 and 6.3 showed the details results generated from running SPECint_base2006 on each system. The execution units depend on the number of benchmarks run in a given system under test. The median scores bold and underline in the table were picked from each system tested and compare the execution time as shown in the table 6.4. These values were used to compare the performance of the two systems tested and the charts layouts shown in figure 6.3 displays the difference in performance of the two processors tested. Table 6.5 and figure 6.4 show the geometric ratio of the two systems tested.

Table 6.4: AMD Dual-Core and Intel Dual-Core Processor Systems Execution times

Table 6.5: AMD Dual-Core and Intel Dual-Core Processor Geometric ratio

6.3.2 Comparing AMD and Intel Processor Execution Time

Table 6.2 and 6.3 showed the details results generated from running SPECint_base2006 on each system. The execution units depend on the number of benchmarks run in a given system under test. The median scores bold and underline in the table were picked from each system tested and compare the execution time as shown in the table 6.4. These values were used to compare the performance of the two systems tested and the charts layouts shown in figure 6.3 displays the difference in performance of the two processors tested. Table 6.5 and figure 6.4 show the geometric ratio of the two systems tested.
Comparing the AMD Dual-Core and Intel Dual-Core Processor Systems Execution times

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>AMD Turion Dual Core</th>
<th>Intel Pentium Dual Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 400.perlbench</td>
<td>647</td>
<td>661</td>
</tr>
<tr>
<td>2 401.bzip2</td>
<td>1225</td>
<td>1150</td>
</tr>
<tr>
<td>3 403gcc</td>
<td>726</td>
<td>674</td>
</tr>
<tr>
<td>4 429.mcf</td>
<td>892</td>
<td>654</td>
</tr>
<tr>
<td>5 445.gobmk</td>
<td>813</td>
<td>803</td>
</tr>
<tr>
<td>6 456.hmmer</td>
<td>1235</td>
<td>1236</td>
</tr>
<tr>
<td>7 458.sjeng</td>
<td>1011</td>
<td>942</td>
</tr>
<tr>
<td>8 462.libquantum</td>
<td>1462</td>
<td>1256</td>
</tr>
<tr>
<td>9 464.h264ref</td>
<td>1397</td>
<td>1338</td>
</tr>
<tr>
<td>10 471.omnetpp</td>
<td>690</td>
<td>714</td>
</tr>
<tr>
<td>11 473.astar</td>
<td>923</td>
<td>955</td>
</tr>
<tr>
<td>12 483.xalancbmk</td>
<td>626</td>
<td>541</td>
</tr>
<tr>
<td>Total Execution Time</td>
<td>11647</td>
<td>10924</td>
</tr>
</tbody>
</table>

Processor Execution Geometric ratio

The performance results of HP Pavilion dv5, AMD Turion II P520 Dual-Core Processor and HP Presario CQ56 Intel Pentium Dual-Core Processor generated from SPEC CINT2006 suite shown in table 6.4 and the total execution time for the processors are as follows:

Total Execution time of AMD Processor is 11647 seconds
Total Execution time of Intel Processor is 10924 seconds

Given,

\[ n = \frac{\text{Execution time of AMD Processor} (Ex)}{\text{Execution time of Intel Processor} (Ey)} \]  

To calculate the percentage of the processor execution time

\[ 1 + \frac{n}{100} = \frac{\text{Execution time of AMD Processor} (Ex)}{\text{Execution time of Intel Processor} (Ey)} \]  

Therefore,

\[ Ex(AMD \text{Processor}) = Ex (Intel \text{Processor}) + \frac{n}{100} \text{Ey(Intel Processor)} \]  

\[ 100 \left( \frac{Ex(AMD \text{Processor}) - Ex (Intel \text{Processor})}{Ey(Intel \text{Processor})} \right) = n \]  

\[ n = \frac{Ex(AMD \text{Processor}) - Ex (Intel \text{Processor})}{Ey(Intel \text{Processor})} \times 100 \]  

\[ n = \frac{11647 - 10924}{10924} \times 100 = 723 \times 100 = 6.6185\% \]  

Then,

\[ \text{SpeedUp} = \frac{11647}{10924} = 1.0662 \]

The above results confirm that HP Presario CQ56 Intel Pentium Dual-Core Processor is over 6.62% faster than HP Pavilion dv5, AMD Turion II P520 Dual-Core Processor on SPEC CINT2006 benchmarks.

Table 6.6 AMD Turion (tm) II P520 and Intel Pentium Dual Core Processor Throughput

<table>
<thead>
<tr>
<th>SYSTEM</th>
<th>SPECint_base_2006 Result</th>
<th>Overall Performance of Systems in percentage (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD Turion (tm) II P520 Dual Core Processor</td>
<td>10.9</td>
<td>48.44</td>
</tr>
<tr>
<td>Intel Pentium Dual-Core Processor</td>
<td>11.6</td>
<td>51.55</td>
</tr>
</tbody>
</table>

6.3.3 Result of the Throughput of the Two Systems under Test

Throughput is the total amount of work done in a given time, such as megabytes per second for a disk transfer i.e
output relative to input; the amount passing through a system from input to output (especially of a computer program over a period of time).

SPECint_base2006 performs three runs of each benchmark in the test suite and records the median, so the bold and underlined text indicates a median measurement as shown in table 6.2 and 6.3. The median score with the higher ratio perform better, while the processor with the higher scores of the overall system performance of SPECint_base2006 shown in figure 6.1 and 6.2 are better. Table 6.6 shown the throughput performance of the system under test, the results show that Intel Pentium Dual-Core Processor is 3.11% faster than AMD Turion (tm) II P520 Dual Core Processor, while the throughput of Intel Pentium Dual-Core Processor is 1.06 times higher than AMD Turion (tm) II P520 Dual Core Processor, figure 6.5 display the chart that clearly show the performance difference of the two processor.

![Comparison of AMD and Intel Processor Throughput](image)

**Figure 6.5: Comparison of AMD Turion (tm) II P520 and Intel Pentium Dual Core Processor Throughput**

7. CONCLUSION

Computer performance evaluation is a key technology for research in computer architecture. The continuous growth in complexity of computer systems is making this task increasingly complex. In general, the problem of developing effective performance evaluation techniques can be stated as finding the best trade-off between accuracy and speed. Multiple cores will provide easy benefits for multithreaded workloads, but many applications written for uniprocessors, will therefore not automatically benefit from CMP designs. While these applications could be parallelized into threads, they usually contain data and control flow dependences that render this a daunting task, especially for poorly-understood legacy code designed by previous programmers.

However, the requirements for successfully delivering hardware-enhanced threading and multicore processing capability go beyond critical silicon manufacturing capacity and technology. The promise of a better user experience also depends on software as well, unless we develop parallel user level applications, it will be difficult to harness the full power of multi core processor technology. The performance improvement to be gained from using some faster mode of execution is limited by the fraction of the time the faster mode can be used. The speedup that can be gained by using a particular feature is the factor of a parallel system as the ratio between the time taken by a single processor to solve a given problem to the time taken by a parallel system consisting of n processors to solve the same problem.

Intel Corporation and Advanced Micro Devices, Inc., have been leading in developing new technologies for the personal computer (PC). The two processors fierce rivalry and competition have been good for the consumer, resulting in constant innovation and lower prices. This thesis measures the performance of AMD dual core and Intel dual core processor, with the aids of SPEC CPU 2006 Benchmarks suite. The results showed that the execution time of HP Presario CQ56 Intel Pentium Dual-Core Processor is over 6.62% faster than HP Pavilion dv5, AMD Turion II P520 Dual-Core Processor while the throughput of Intel Pentium Dual-Core Processor is 1.06 times higher than AMD Turion (tm) II P520 Dual Core Processor. Intel Pentium Dual-Core Processor had the best performance due to faster core-to-core communication, dynamic cache sharing between cores, smaller size of level 2 cache, and run at lower core and bus frequencies.

REFERENCES


[5]. Anirban S., (2006); Dual Core Processors – A brief overview1, uploaded: April 12th 2006, anirbans@cs.ubc.ca retrieved on April, 2011.

[7]. Emer, J., (2005); “Microprocessor Evolution:4004 to Pentium-4” Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts.


[9]. Gordon S., Bruce C., Roger C., Richard P. and Fleur S. (1999); A superscalar architecture to exploit instruction level parallelism, University of Hertfordshire, Hatfield, Herts, AL10 9AB, UK.


[13]. John L. K. (2003); “Performance Evaluation: Techniques, Tools and Benchmarks”, Electrical and Computer Engineering Department, ENS 143, The University of Texas at Austin, Austin, TX 78712, ljohn@ece.utexas.edu, http://www.ece.utexas.edu/~ljohn


[15]. Pase D. M. and Eckl M. A. (2005); A Comparison of Single-Core and Dual-Core Opteron Processor Performance for HPC, IBM Systems and Technology Group, IBM xSeries Performance Development and Analysis, 3039 Cornwallis Road, Research Triangle Park, NC 27709-2195,USA.


[22]. Stefan R. (2004); Trends and Challenges in High-Performance Microprocessor Design, Published by Intel Corporation.
