Performance Characterization of SPEC CPU2006 Benchmarks on Intel Core 2 Duo Processor

Tribuvan Kumar Prakash and Lu Peng, *Member, IEEE Computer Society*

***Abstract*—As the processor architectures are evolving, it is very important to develop appropriate benchmarks that are used to measure their performance. Also, it is very important to design appropriate compilers that can optimally utilize the new features of the evolving processors. For this we need to have a complete insight on the performance characteristics and the impact of compilers on performance characteristics of the benchmarks. In this paper, we first report performance characterization of SPEC CPU2006 suite on Intel Core 2 Duo processor which represents an emerging popular computing platform. Second, we compare the effects of two widely used C++ compilers: Intel C++ and Microsoft VC++ compilers. Performance characteristics include Instruction per cycle (IPC), run time, cache miss rate and branch miss rate are measured and reported. Our results showed that Intel Compiler has better performance than Microsoft VC++ compilers for a majority of SPEC CPU2006 C/C++ programs running on Intel Core 2 Duo Processor.**

***Index Terms*— SPEC CPU2006, Intel Core 2 Duo, Intel C++ Compiler, Microsoft VC++ Compiler.**

***Area of Interests*: 5.5 Computer Architecture**

I. Introduction

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 [9] is one such benchmark that has intensive workloads based on real applications and is the successor of the SPEC CPU2000 benchmark [9]. Also, the need of appropriate compilers to keep up with those advanced architectures to maximize the performance has evoked interests in researchers to understand the impact of compilers on performance characteristics.

Manuscript received Oct. 12, 2007. This work is supported in part by the Louisiana Board of Regents grants NSF (2006)-Pfund-80 and LEQSF (2006-09)-RD-A-10, the Louisiana State University and an ORAU Ralph E. Powe Junior Faculty Enhancement Award.

Tribuvan Kumar Prakash is with Realization Technologies, Inc., San Jose, CA 95113 USA (email: tribuvan@gmail.com).

Lu Peng, is with the Electrical and Computer Engineering Department, Louisiana State University, Baton Rouge, LA 70803 USA (phone: 1-225-578-5535, fax: 1-225-578-5200, email: lpeng@lsu.edu).

This paper presents a detailed analysis of the SPEC CPU2006 benchmark running on Intel Core 2 duo processor [4] and emphasizes on its workload characteristics and memory system behavior. We compare the CPU2006 and CPU2000 benchmarks with respect to performance bottlenecks by using the Intel VTune performance analyzer [5] for the entire program execution. Also, the various performance aspects of two popularly used C/C++ compilers: Intel C++ 9.1 [3] and Microsoft Visual C++ 2005 [7] are compared.

The Intel C++ compiler 9.1 for Windows provides advanced optimization features that maximize performance for applications running on the latest Intel processors, including Chip Multi-processors (CMP). Key features of Intel C++ 9.1 compiler include multi-threaded application support, multi-core development support, Microsoft Visual Studio 200X integration and advanced optimization like Interprocedural Optimization (IPO), Profile-guided Optimization (PGO), Automatic Vectorizer and High-Level Optimization (HLO) [3]. On the other hand, Microsoft Visual C++ 2005 [7] is an integrated development environment (IDE) product developed by Microsoft. It has features such as syntax highlighting, IntelliSense (a coding auto completion feature) and advanced debugging functionality. It includes MFC (Microsoft Foundation Classes) 8.0 and support for the C++/CLI language and OpenMP.

According to our measurements, CPU2006 benchmarks have larger input dataset and longer execution time than those of CPU2000. Our results also show that apart from architectural features, compilers also have high impact on performance. For some application such as *hmmer* and *h264ref*, Intel C++ shows its superiority in performance over Microsoft VC++ compiler. In addition, it also shows better microarchitecture performance in L2 cache miss rate and branch miss rate for most of programs because of its specific optimizations on Intel Core architecture. However, its larger dynamic instruction counts compromises this effect for some floating programs such as *lbm*.

The remainder of this paper is organized as follows. Section II describes the methodology. Section III reports the performance characterization of SPEC CPU2006 and CPU2000 on Intel Core 2 Duo processor. Section IV details the comparison of performance characteristics for Intel C++ 9.1 and Microsoft Visual C++ 2005 compilers on SPEC CPU2006. Section V describes the related work. Lastly, section VI gives a brief conclusion obtained from our analysis.

TABLE I
SYSTEM SPECIFICATION

CPU	Intel Core 2 Duo E6400 (2 x 2.13GHz)
Technology	65nm
Transistors	291 Millions
Hyperthreading	No
Branch Predictor	Combined three types of predictors - global, bi-modal and loop detectors.
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.
L1 TLB size	Instructions: 128 entries Data: 256 entries
Memory	2GB (1GB x 2) DDR2 533MHz
FSB	1066MHz Data Rate 64-bit
FSB bandwidth	8.5GB/s
HD Interface	SATA 375MB/s

II. METHODOLOGY

We installed Microsoft Visual C++ 2005 (also known as VC++ 8) and Intel C++ compiler 9.1 on 32 bit Windows XP with SP2 operating system running on Intel Core 2 Duo E6400 processor with 2.13GHz. The specification of Intel Core 2 Duo machine is shown in table 1.

For performance characterization of SPEC CPU2006 benchmark suite, all the integer and floating point programs were considered. The details of the applications in the benchmark suite can be found in [9]. We also made a comparison with SPEC CPU2000 C/C++ programs. Microsoft Visual C++ 2005 and Intel FORTRAN Compiler 9.1 were used to compile most of the applications under consideration except for *libquantum, xalancbmk*, *calculix, povray, tonto, wrf* and *zeusmp* due to compilation problems. Therefore, we compiled these programs using the Intel C++ 9.1 compiler.

After that, a subset of C/C++ SPEC CPU2006 benchmark suite was used to analyze the performance characteristics of the two compilers under consideration. We use the fastest speed compilation flags for both compilers. For the Microsoft VC++ compiler, we set "-O2", while for the Intel C++ compiler we set "-fast" which is equal to "-O3 –ipo -xP" [3].

All benchmark applications were analyzed using Intel(R) VTune(TM) Performance Analyzer 8.0.1. 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. Event based sampling was selected for monitoring. We measured microarchitecture events such as L1D cache miss, L2 cache misses, DTLB misses, Instruction per Cycle (IPC), branch misprediction, etc.

III. CHARACTERIZATION OF SPEC CPU2006 BENCHMARK

Compared with CPU2000 programs, CPU2006 benchmarks have larger input dataset and longer execution time. According to our measurement, the execution time for CPU2000 programs ranges from 56-170 seconds while those for CPU2006 benchmarks ranges from 563-1590 seconds on the Intel Core 2 Duo system.

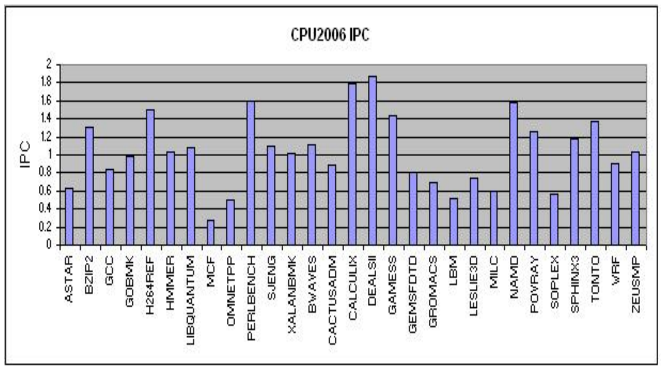

Figure 1(a)

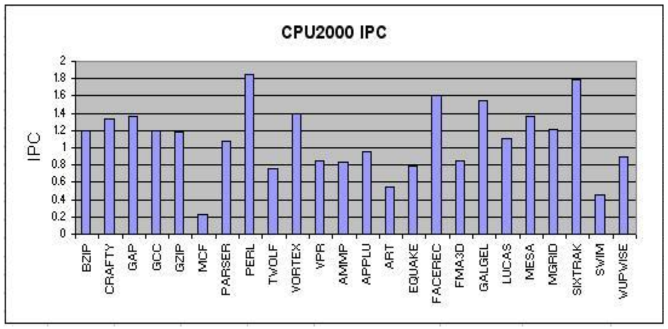

Figure 1(b)

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

Figure 1(a) and Figure 1(b) depict the Instruction per Cycle (IPC) of CPU2006 and CPU2000 respectively. The average IPC for CPU2006 and CPU2000 benchmarks were measured at 0.97 and 1.1 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.

Figure 2(a) and Figure 2(b) represent the instruction retired 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 *gcc*, *libquantum*, *mcf*, *perlbench*, *sjeng*, *xalancbmk* and *gamess* have high percentage of branch instructions. On the other hand, 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.

Higher percentage of load and store instructions retired or higher percentage of branches do not necessary indicate the presence of more 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

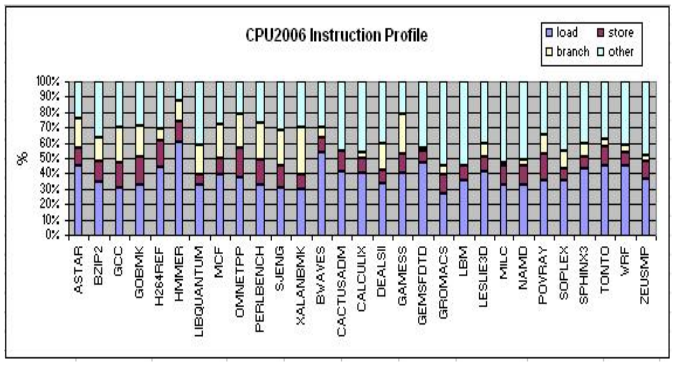

Figure 2(a)

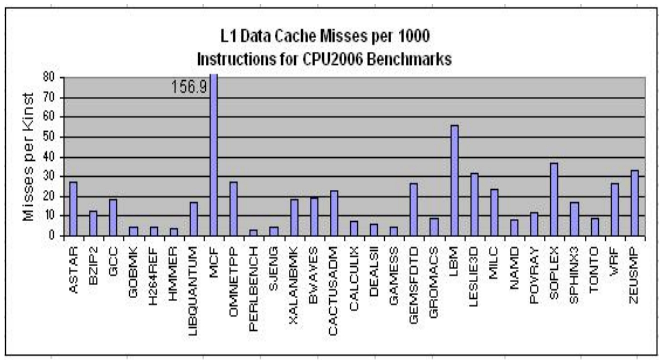

Figure 3(a)

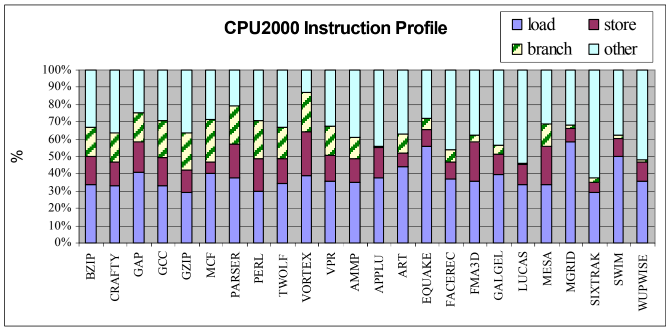

Figure 2(b)

Figure2 (a) Instruction Profile of SPEC CPU2006 Benchmark; (b) Instruction Profile of SPEC CPU2000 Benchmark

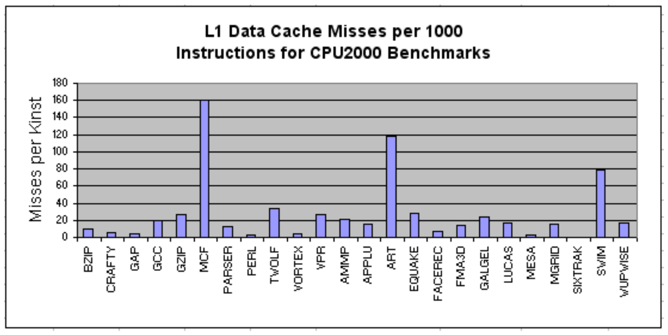

Figure 3(b)

Figure 3 (a) L1 D Cache Misses Per 1000 Instruction of SPEC CPU2006 Benchmarks; (b) L1 D Cache Misses Per 1000 Instruction of SPEC CPU2000 Benchmarks

these benchmarks, L1 data cache misses per 1000 instructions, L2 cache misses per 1000 instructions and branch misprediction per 1000 instructions were measured and analyzed.

Figure 3(a) and 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 L1D cache misses per 1000 instructions for cpu2006 and cpu2000 benchmark set under consideration was found to be 22.5 and 27 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.

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 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.

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* and *gromacs* are $3.3*10^{-5}$ and $6.2*10^{-5}$ respectively.

Figure 5(a) and 5(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 4.1 and 2.6 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].

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 6(a). A macro “*TEST_FLAG_SWEEP*” is responsible for 21% of overall L2 cache misses. The definition of *TEST_FLAG_SWEEP* is shown in Figure 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

```
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 4 Code Sample of MCF

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

(a)

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

(b)

Figure 6 Code Sample of LBM

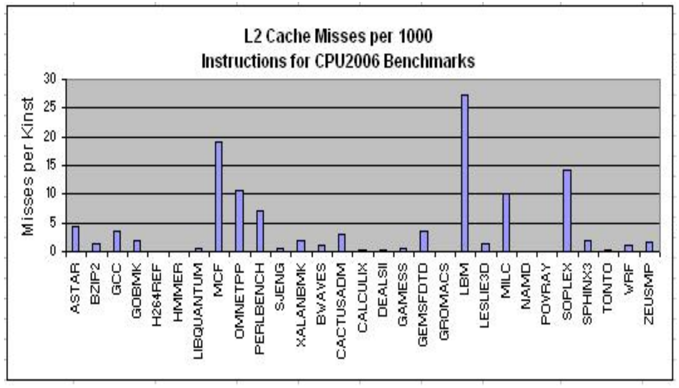

Figure 5(a)

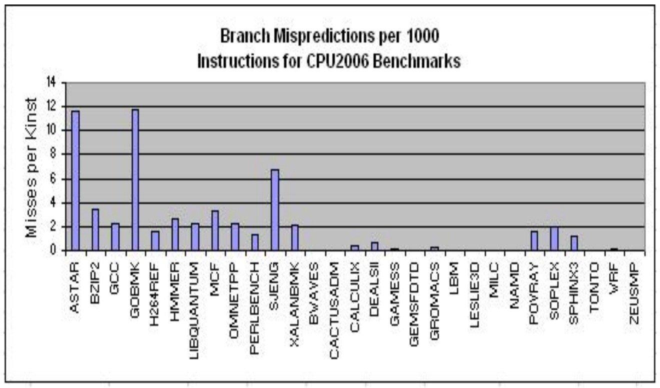

Figure 7(a)

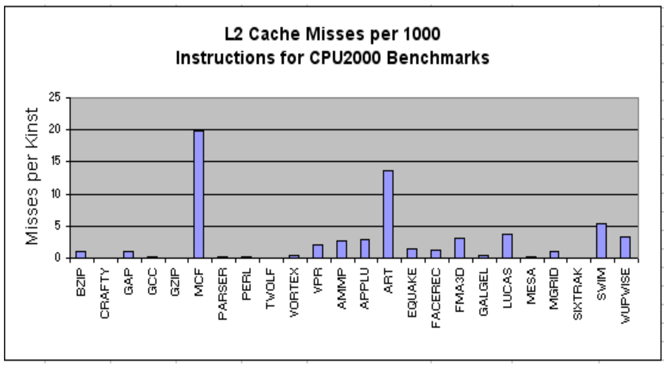

Figure 5(b)

Figure 5(a) L2 Cache Misses Per 1000 Instructions of SPEC CPU2006 Benchmarks; (b) L2 Cache Misses Per 1000 Instruction of SPEC CPU2000 Benchmarks

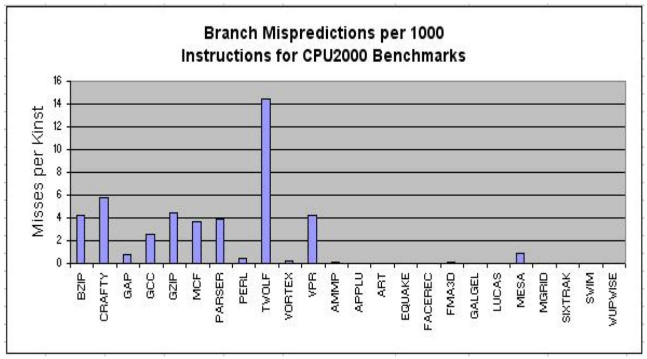

Figure 7(b)

Figure 7(a) Branch Misprediction Per 1000 Instruction of SPEC CPU2006 Benchmarks; (b) Branch Misprediction Per 1000 Instruction of SPEC CPU2000 Benchmarks

lbm has very large data footprint which results in high stress on L2 cache. For *mcf*, *Primal_bea_mpp* (33.4%) and *re-fresh_poten-tial* (20.2%) are two major functions resulting in L2 cache misses. Intensive pointer chasing is responsible for this.

Figure 7(a) and 7(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 0.38 and 0.08 respectively.

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.

IV. Microsoft VC++ Vs. Intel C++

In this section, we compared compiler effects on SPEC CPU2006. We first compared static code size and dynamic instruction counts. Table 2 lists static code size of binaries generated by both compilers. In general, we observed that Intel C++ binaries are larger than those generated by the Microsoft VC++ compiler. Figure 8 shows the profile of Instruction Retired comparison between Microsoft VC++ and Intel C++. The vertical axis represents the absolute number of in-

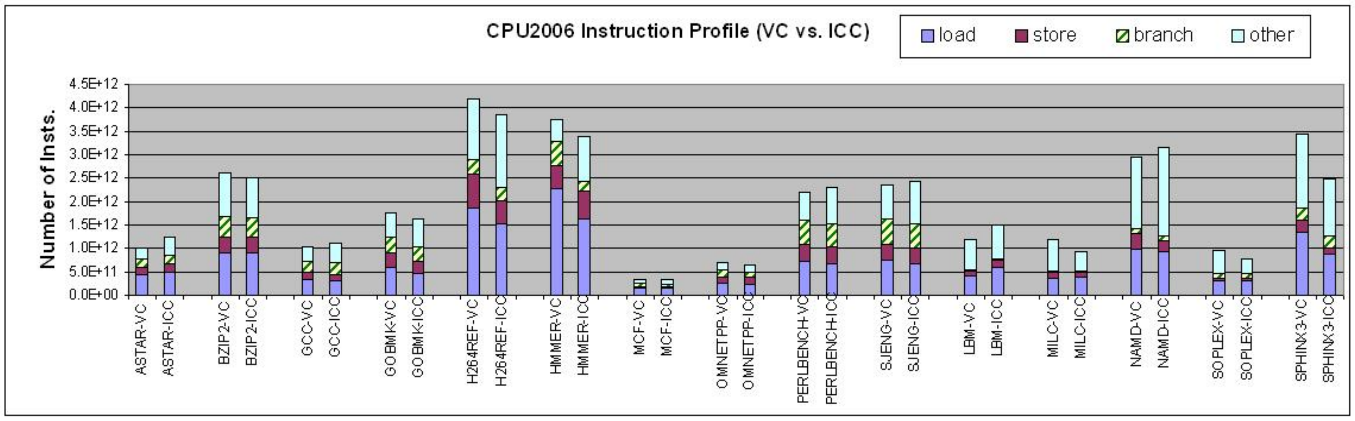

Figure 8. CPU2006 Instruction Retired Profile (VC vs. ICC)

structions brake down by types. A few observations can be made:

(1) For 9 out of 15 programs, dynamic instructions retired for Intel C++ binaries are smaller than those generated by the Microsoft VC++ compiler though the former have larger static code size.

(2) The percentage of load and store instructions is lower in most cases for binaries generated from Intel C++ compiler compared to that of Microsoft VC++ binaries. Hence, Intel C++ compiler reduces the number of memory accesses comparatively.

(3) The percentage of branch instructions is closely same for both Intel C++ and Microsoft VC++ binaries. Other instructions consist of various integer and floating point instructions which on an average comprise for approximately 37% and 32% of the overall instructions, for Intel C++ binaries and Microsoft C++ binaries respectively.

We then compared the normalized runtime for Intel C++ and Microsoft VC++ compilers running SPEC CPU2006 benchmarks. For normalization, the runtime of Microsoft VC++ was considered to be the base runtime. Figure 9 shows the normalized runtime for Intel C++ and Microsoft VC++ compilers. From the figure, it is evident that the runtime for most of the applications are very close. However, for applications *hmmer* and *h264ref* there is a drastic decrease in runtime while running with Intel C++ compiler. Microsoft VC++ shows improvement in runtime for floating programs *lbm*, *soplex* and *sphinx3*.

TABLE II
STATIC CODE SIZE (IN BYTES) OF BINARIES GENERATED BY MICROSOFT VC++ AND INTEL C++

Name / Bytes	VC++	IC++
ASTAR	126976	163840
BZIP2	122880	163840
GCC	2744320	3788800
GOBMK	3190784	3792896
H264REF	552960	1294336
HMMER	237568	323584
MCF	90112	106496
OMNETPP	724992	1286144
PERLBENCH	978944	1536000
SJENG	188416	266240
LBM	102400	102400
MILC	180224	323584
NAMD	356352	561152
SOPLEX	409600	1093632
SPHINX3	262144	393216

To better understand the performance impact of compilers, we compared various performance matrics. We analyzed the L1D cache misses per 1000 instructions, L2 cache misses per 1000 instructions and branch misprediction per 1000 instructions for binaries generated by the Intel C++ and Microsoft VC++ compiler. Figure 10 shows the comparison of L1D cache misses per 1000 instructions. From this figure, the total number of L1D cache misses rate is almost the same for both compliers except for *sphinx3* and *soplex*. The L1 data cache rate gap between Intel C++ and Microsoft VC++ is responsible for the execution time difference for these two programs.

Figure 11 shows the comparison of L2 cache misses per 1000 instructions for both compilers. The figure shows that there was considerable improvement in L2 cache misses rate for memory intensive applications such as *mcf*, *lbm, perlbench* and *soplex* in the case of Intel C++ compiler compared to that of Microsoft VC++ compiler. From this figure, we can conclude that Intel C++ compiler, which utilizes more features of Intel Core 2 Duo processor, has better memory performance than that of Microsoft VC++.

Figure 12 shows the branch misprediction rate. From this figure, it can be observed that *astar*, *h264ref*, *hmmer* and *omnetpp* show improvement in branch misprediction rate when running with Intel C++ compiler compared to that with Microsoft VC++ compiler. Other programs show similar behaviors.

In general, we find that Intel C++ compiler shows superior performance for *hammer* and *h264ref*. In addition, it also shows better microarchitecture performance in L2 cache miss rate and branch miss rate for most of programs. However, its larger dynamic instruction counts compromises this effect for some floating programs such as *lbm*.

V. RELATED WORK

Researchers in computer architecture area show strong interests in performance characterization of CPU2006. Sarah et al [1] reported the performance characterization of SPEC

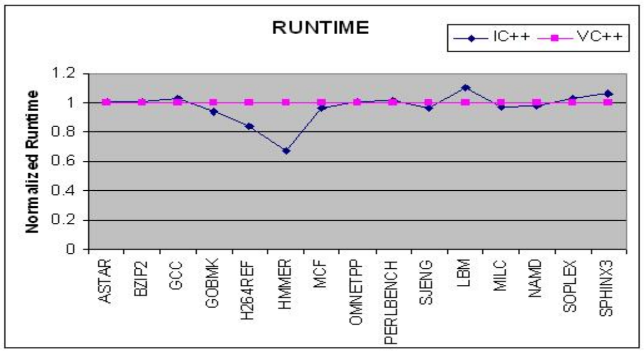

Figure 9. Runtime Comparison

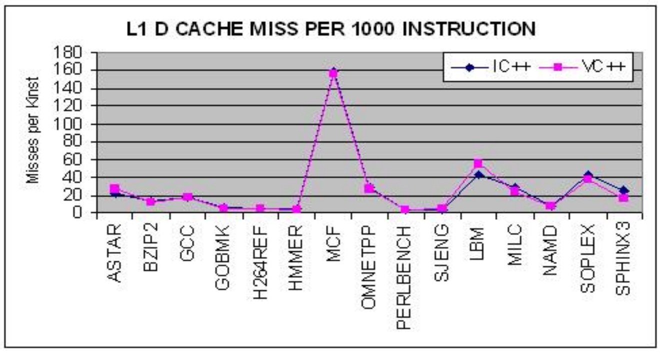

Figure 10. Comparison of L1D Cache Miss Per 1000 Instructions

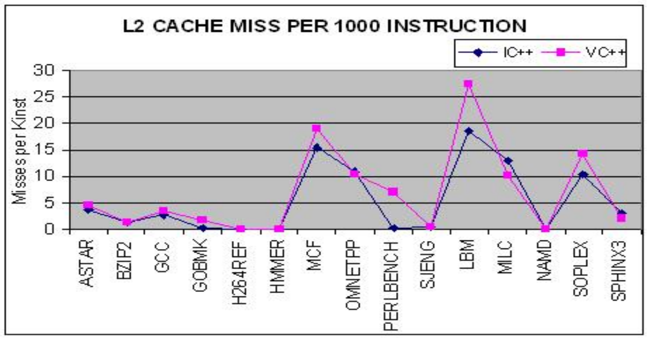

Figure 11. Comparison of L2 Cache Miss Per 1000 Instructions

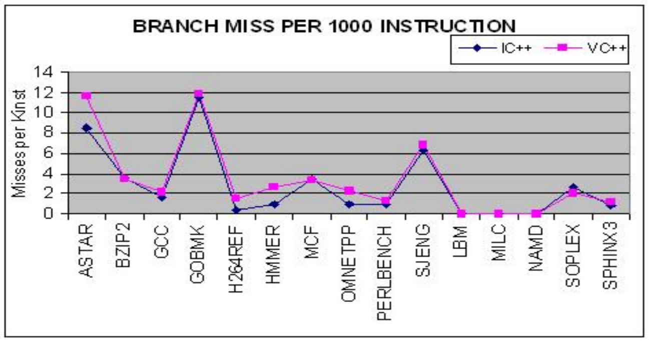

Figure 12. Comparison of Branch Mis-prediction Per 1000 Instruction

CPU2006 and analyzed the impact of “Macro fusion” and “Micro-op fusion” of the Woodcrest processor. These results parallel our own upon which this paper is based. Ye et al [10] compared CPU2006 integer benchmark binaries in 64-bit and 32-bit formats on an x86-64 architecture based processor.

The effect of compilers and compiler optimizations on application performance has been studied and analyzed for a long time. Gurumani and Milenkovic studied the execution characteristics of Visual C++ 6.0 and Intel C++ on Pentium 4 processor using SPEC CPU2000 benchmark suite in [2]. They concluded that Intel C++ compilers performed better for graphics and visualization applications.

Compared with software simulation, using Intel VTune performance analyzer and performance counters in real processors is a fast and feasible way to characterizing emerging workloads. There are a few recent works analyzing Bioinformatics and Data Mining workload [6][8] by performance counters and VTune analyzer.

VI. Conclusion

In this paper, we analyzed the emerging CPU2006 on Intel Core 2 Duo processor. According to our measurements, CPU2006 benchmarks have larger input dataset and longer execution time than those of CPU2000. Our results also show that apart from architectural features, compilers also have high impact on performance. For some application such as *hammer* and *h264ref*, Intel C++ shows its superiority in performance over Microsoft VC++ compiler. In addition, it also shows better performance in L2 cache miss rate and branch miss rate for most of programs because of its specific optimizations on Intel Core architecture. However, its larger dynamic instruction counts compromises this effect for some floating programs such as *lbm*.

References

[1] S. Bird, A. Phansalkar, L K. John, A. Mericașand and R. Indukuru, “Performance Characterization of SPEC CPU Benchmarks on Intel's Core Microarchitecture based processor”, in Proceedings of 2007 SPEC Benchmark Workshop, Jan 2007.

[2] S. T. Gurumani and A. Milenkovic, “Execution Characteristics of SPEC CPU2000 Benchmarks:Intel C++ vs. Microsoft VC++”, in Proceedings of the 42nd ACM annual southeast regional conference, 2004.

[3] Intel, Intel C++ Compiler 9.1 for Windows, http://cache.www.intel.com/cd/00/00/28/48/284831_284831.pdf

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

[5] Intel, Intel VTune Performance Analyzer, http://www.intel.com/cd /software/products/asmona/eng/vtune/239144.htm

[6] Y. Li, T. Li, T. Kahveci, and J. Fortes. Workload characterization of bioinformatic applications. In Proceedings of IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), 2005.

[7] Microsoft, 32-bit Optimizations and Command-Line Switches, http://msdn.microsoft.com/vstudio/tour/vs2005_guided_tour/VS2005pro /Framework/CPlus32BitOptimization.htm

[8] B. Ozisikyilmaz, R. Narayanan, J. Zambreno, G. Memik, A. Choudhary, An Architectural Characterization Study of Data Mining and Bioinformatics Workloads, in Proceedings of IEEE International Symposium on Workload Characterization, Oct. 2006.

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

[10] D.Ye, J. Ray, C. Harle and D. Kaeli, Performance Characterization of SPEC CPU2006 Integer Benchmarks on x86-64 Architecture, in Proceedings of IEEE International Symposium on Workload Characterization, Oct. 2006.

[11] H. Zhou and T. M. Conte, “Enhancing memory level parallelism via recovery-free value prediction,” Proceedings of the 17th annual international conference on Supercomputing (ICS), Jun. 2003.