Execution Characteristics of SPEC CPU2000 Benchmarks: Intel C++ vs. Microsoft VC++

Swathi Tanjore Gurumani Aleksandar Milenkovic
Electrical and Computer Engineering Department
University of Alabama in Huntsville
Huntsville, Alabama
{gurumas, milenka}@eng.uah.edu

ABSTRACT
Modern processors include features such as deep pipelining, multi-level cache hierarchy, branch predictors, out of order execution engine, and advanced floating point and multimedia units. To successfully exploit these features, architecture-aware compilers that can produce target-specific optimal codes for the applications are needed. Using the knowledge about the architectural features, the compilers can contribute to maximizing the application performance through effective pipeline scheduling, memory penalty minimization and path length reduction. A study of the execution characteristics for the binaries generated by the various compilers can provide insights about the effectiveness of the optimization options used in the compilers. The in-built performance monitoring hardware found in present day processors can be used to collect the performance metrics for the study of execution characteristics. In this paper, we compare the Intel C++ and Microsoft VC++ compilers by studying the execution characteristics of SPEC CPU 2000 benchmarks run on a Pentium IV processor. The benchmarks were compiled with identical optimization switches in both compilers and the performance metrics were collected using Intel's VTune Performance Analyzer. The analyses of results showed that the Intel C++ compiler performed better than VC++ for all considered applications and significantly better for computer visualization and graphical applications.

General Terms
Measurement, Performance, Experimentation

Keywords
Compiler optimizations, Performance Evaluation, SPEC CPU2000 benchmarks, Event-based sampling.

1. INTRODUCTION
The effect of compilers and compiler optimizations on application performance has been studied and analyzed for sometime now. Performance characteristics of an application are found to be dependent on the type of compiler options used [5]. Software developers depend on compiler optimizations to attain the required performance characteristics from current day applications, which are becoming more complex and bigger in size. The present day compilers are required to keep pace with upcoming processor technologies to exploit newer and more efficient optimization techniques. Compilers must be aware of computer architecture features like deep pipelining, multi-level cache hierarchy, instruction level parallelism, and branch prediction to provide optimal performance [3]. Architecture aware compilers exploit the hardware features and contribute to maximizing the application performance. Compiler/hardware interaction leads to performance improvements attained through path length reduction, efficient instruction selection, pipelining scheduling and memory penalty minimization [5].

Performance has been shown to improve by using processor specific optimizations. Performance gain of at least 10% was obtained by running Linpack executables compiled using Xeon specific optimizations over executables compiled using Pentium III processor optimizations [2]. Examples of branch optimization by architecture-aware compilers shown by Milenkovic et.al. [3] have proved the importance of compilers and optimizations with regard to application performance.

The best way to evaluate the compilers strength is to study the performance and execution characteristics of the executables generated by the compilers. The performance evaluation of these executables can be used to study the effectiveness of the various optimization switches that can be used in compilers. Most current day processors come with a built-in hardware to support performance monitoring, and these performance-monitoring counters can be calibrated to collect relevant event data. Performance evaluation has already been done for Pentium Pro processor using SPEC 2000 benchmarks [6] and Pentium II processor using Multimedia applications [1]. Pentium III and Pentium IV performances have been compared to emphasize processor specific optimizations in [2]. But these performance analyses did not include the comparison of compilers and were used to study the underlying architecture and to characterize workloads.

In this paper, we compare the Intel C++ and Microsoft VC++ compilers. The SPEC CPU2000 benchmarks were compiled with Intel C++ and Microsoft VC++ compilers and the execution and performance characteristics of the binaries were studied. The compiled benchmarks were run on a Pentium IV based system. Intel's VTune Analyzer was used to calibrate and collect data from the performance counters. Analyses show that the Intel compiler,

ACM SE'04, April 2–3, 2004, Huntsville, Alabama, USA.

by virtue of the knowledge of the processor's architecture, outperformed the Microsoft counterpart in most cases in performance. This proves that the processor itself does not determine the performance of the system. Though the system's processor and memory architecture contributes to the performance, the compiler used also plays a major role in the performance. A fair comparison of the compilers' contribution to performance can be obtained by running the executables generated by two different compilers on the same system, with the same processor, memory architecture, and operating system. Hence, the differences in the obtained execution characteristics are directly related to compilers.

The remainder of the paper is organized as follows. Section 2 gives an overview about Intel C++ and Microsoft VC++ compiler and about their optimization switches. Section 3 details the performance monitoring facilities available in the Pentium IV processor. Section 4 describes the experimental setup that was used to collect data. Section 5 presents the results obtained and also the observations and inferences that are made from the results.

2. COMPILERS USED

The information about the compilers is obtained directly from MSDN library for VC++ [9] and Intel C++ compiler's user guide [10]. Visual C++ 6.0 is the leading C++ compiler for 32-bit Microsoft Windows and has a number of features that aid in producing fast programs. The Microsoft VC++ compiler can perform all the general code optimizations with the help of compiler switches and pragma statements. The VC++ compiler performs copy propagation and dead store elimination, common subexpression elimination, register allocation, function inlining, loop optimizations, flow graph optimizations, peephole optimizations, scheduling, and stack packing. It does not do loop unrolling, although it does unroll loops for a few small special cases such as block memory moves. The switches that affect code generation are the /G options whereas switches that control optimization are the /O options. The compiler can target Pentium CPUs (/G5), Pentium Pro CPUs (/G6), or a "blend" of Pentium and Pentium Pro optimization options (/GB, the default). The default /GB switch is designed to produce good results, as it targets the most common current processor. The three most common compiler optimization switches are /Od, which disables all optimizations for debugging purposes, /O1, which minimizes code size, and /O2, which maximizes code speed. The /O2 option creates the fastest code and is the default setting for the release builds. It is equivalent to using the switches /Og /Oi /Ot /Oy /Ob1 /Gs /Gf /Gy. Table 2.1 gives the compiler options invoked with /O2 switch.

The Intel C++ compiler is designed to be easily interfaced with the Microsoft VC++ environment. The Intel C++ compiler optimizes performance for applications running on Intel architecture-based computers. Performance gains by using this compiler are a result of profile-guided optimization, prefetch instruction, and support for Streaming SIMD Extensions (SSE) and Streaming SIMD Extensions 2 (SSE2), automatic vectorizer, data prefetching, inter-procedural optimization and processor dispatch. The processor dispatch takes advantage of the latest Intel architecture features while maintaining object code compatibility with previous generations of Intel Pentium processors. Profile-guided Optimizations (PGO) let the compiler know which areas of an application are most frequently executed. By knowing these areas, the compiler is able to be more selective in optimizing the application. PGO helps by allocating registers using the profile information to optimize the location of spill code. Intel C++ compiler's /O2 optimization to favor speed enables the same switches that Visual C++ 6.0 enables with /O2.

Table 2.1 Compiler options invoked with /O2 switch

Switch	Function
/Og	Eliminates local and global common subexpressions, allows automatic register allocation, and allows loop optimization.
/Oi	Replaces certain function calls with inline function expansion.
/Ot	Favors faster executable files.
/Oy	Suppresses creation of frame pointers on the call stack.
/Ob	Controls inline expansion of functions
/Gs	Controls stack probes
/Gf	Copies identical strings into one location in the executable file
/Gy	Allows compiler to package individual functions in the form of packaged functions

3. PERFORMANCE MONITORING FEATURES OF PENTIUM IV PROCESSOR

The performance monitoring hardware in Pentium IV includes event detectors and event counters and support for Precise Event-Based Sampling (PEBS). It supports 48 event detectors and 18 event counters and this enables concurrent data collection of a large number of events. PEBS support is a significant performance-monitoring advantage provided by Pentium IV as previous processors supported only Imprecise Event-Based Sampling (IEBS). Event based sampling, whether precise or imprecise, gives a good estimation of the number of events. Event based sampling is an alternative to periodic or time based sampling, where samples are collected at regular time intervals. Events are measured only after a specified number of events have occurred. PEBS reduces the interference experienced by the monitoring system and allows collection of more samples for the same level of interference. The results showing the advantage of PEBS over IEBS, and more information about performance monitoring features of Pentium IV can be found in [8].

4. EXPERIMENTAL SETUP

SPEC CPU2000 benchmark suite [11] was selected to study the compilers. The benchmark suite portrays a wide variety of applications and can exhaustively test the abilities of the compiler. SPEC CPU2000 benchmarks measure the performance of the processor, memory and compiler used on the tested system and do not stress on I/O devices, networking and the operating system. The industry standard benchmarks portray real user applications and are computation intensive. Benchmarks from CINT2000 (integer benchmark suite) and CFP2000 (floating point benchmark suite) were used in this analysis. The reference input set provided by SPEC was used to provide inputs to the application. Table 4.1

gives the descriptions of the benchmarks used for this experimentation.

Table 4.1 Benchmarks Used

Name	Description
164.gzip (INT)	Data Compression written in C
176.gcc (INT)	C Programming Language Compiler
177.mesa (FP)	3-D Graphics Library written in C
181.mcf (INT)	Combinatorial Optimization written in C
186.crafty (INT)	Chess – Game Playing written in C
197.parser (INT)	Word Processing written in C
252.eon (INT)	Computer Visualization written in C++
253.perlbmk (INT)	PERL Programming Language written in C
254.gap (INT)	Group Theory, Interpreter written in C
255.vortex (INT)	Object Oriented database written in C

The source codes of the benchmark suite were first compiled using Microsoft Visual C++ and the Intel C++ compiler. The Visual C++ environment (IDE) was used to configure both compilers. The benchmarks were compiled with the /O2 optimization switch being enabled. By enabling the /O2 optimization switch, the compilers were used to produce a code for maximizing the speed of the application. As already shown, both compilers make use of the same options when the /O2 switch is used. The programmer was not allowed to interfere and fine tune the application, affecting its performance.

Intel's VTune performance analyzer v5.0 [4] was then used to collect the performance metrics using these compiler-generated executables. VTune supports simultaneous sampling of multiple events and real time display using counter monitors. It has a customizable user interface that can be configured to collect the required events. Though VTune supports both time based and event based sampling, VTune was configured to collect performance data using event based sampling to take advantage of Pentium IV's EBS feature. Since VTune has a low intrusion, the samples collected provide a closer representation of application's actual performance. VTune was configured to collect the following events: clockticks, instructions retired, loads retired, stores retired, branches retired, I level cache misses and mispredicted branches. The events were collected separately for Intel C++ and VC++ and later compared against each another.

5. RESULTS

Clockticks is the best measure for comparison when the /O2 optimization switch has been used. The applications have been compiled to maximize speed and we would expect the application to run in minimum amount of time. Figure 5.1 shows the normalized comparison of clockticks. It is clear that the number of clockticks is always less in the case of Intel compiler. There is a drastic decrease in the number of clockticks for the application 252.eon. It is interesting to note that 177.mesa and 252.eon are 3D graphics library and computer visualization applications, respectively and they show a clear reduction in clockticks when compiled with the Intel compiler. The other application, which shows a distinct difference, 186.crafty is the chess gaming program. Though the other benchmarks do not portray a very large difference, 186.crafty, is clear that the Intel compiler takes advantage of its architecture and optimizes better for graphics and visualization applications.

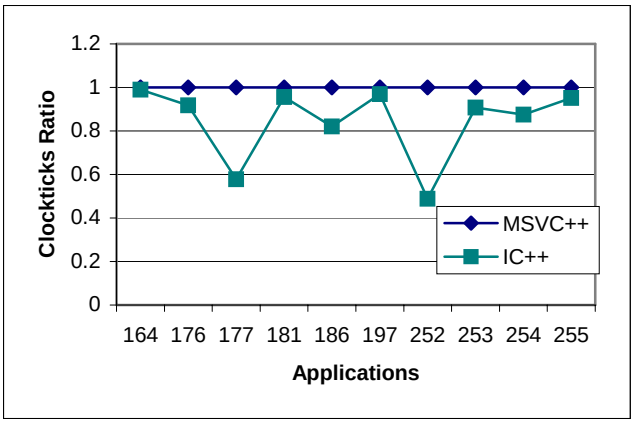

Figure 5.1 Comparison of Clockticks

Though the program size is not optimized in /O2, a comparison of the size of binaries generated by the compilers reveal that VC++ produced smaller executables. Table 5.1 gives the comparison of the size of the binaries produced by the compilers.

Table 5.1 Comparison of Binaries size

Benchmark	Code Size (in Bytes)	
	MSVC++	IC++
164.gzip	69,632	77,824
176.gcc	1,089,536	1,314,816
177.mesa	442,368	610,304
181.mcf	49,152	53,248
186.crafty	241,664	258,048
197.parser	118,784	131,072
252.eon	405,504	413,696
253.perlbmk	516,096	651,264
254.gap	356,352	413,696
255.vortex	417,792	454,656

It would be interesting to see the number of instructions for the same set of applications for which the clockticks were compared. Figure 5.2 shows the normalized number of instructions executed. The instruction counts are lower all applications, except for 197.parser, 253.perlbmk, and 254.gap which show a negligible increase in the number of instructions executed (<3%). Benchmarks 177.mesa and 252.eon have significantly reduced the number of instructions executed, for 21% and 15% respectively.

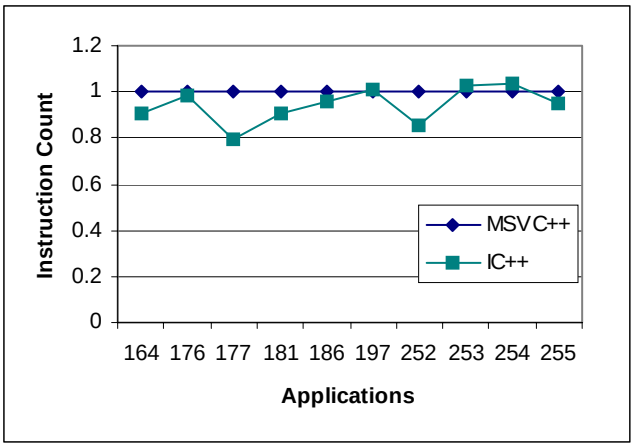

Figure 5.2 Comparison of Instruction Count

The general rule is that the most widely used instructions are the simple operations like load, store and branch. Together they contribute to about more than 50% of the overall instructions executed [7]. Hence, it was decided to study the distribution of these instructions in both cases. The breakdown of instructions was then studied to observe what kind of instructions made the difference in the comparison or if it was due to an overall decrease in all instructions. On average, there were about 35% load instructions, 20% stores and about 18% branch instructions in the considered applications. A more thorough comparison is provided in the following sections. Figure 5.3 compares the distribution of loads in the applications.

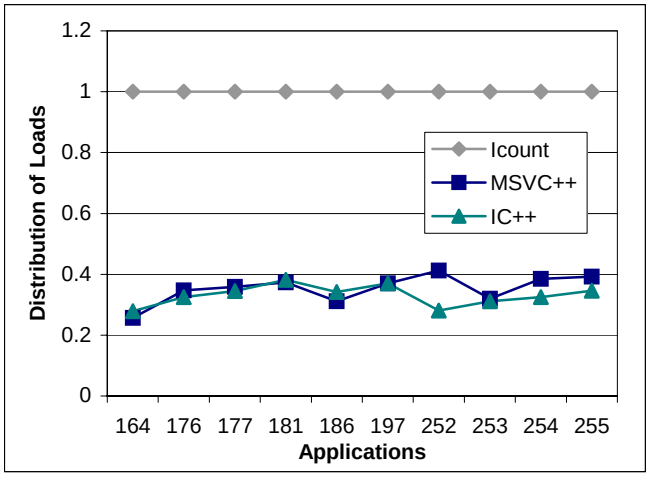

Figure 5.3 Comparison of Loads Retired

The distribution of loads corresponding to their respective instruction count shows that the percentage of load instructions is about the same for most applications except 252.eon, where there is about a 15% difference. It is obvious that VC++ would have more load instructions retired for all cases when the absolute numbers are compared.

The distribution of stores is shown in Figure 5.4. Apart from 252.eon, the percentage of distribution is identical for both compilers. It is clear that the Intel compiler has considerably reduced the number of memory accesses for the computer visualization application. Applications like 253.perlbmk and 254.gap, which had the instruction count on the higher side for the Intel compiler, also show a decrease in the percentage of loads and stores. It can be understood that the Intel compiler targets to decrease the number of memory accesses for all applications, decreasing the number of load and store instructions.

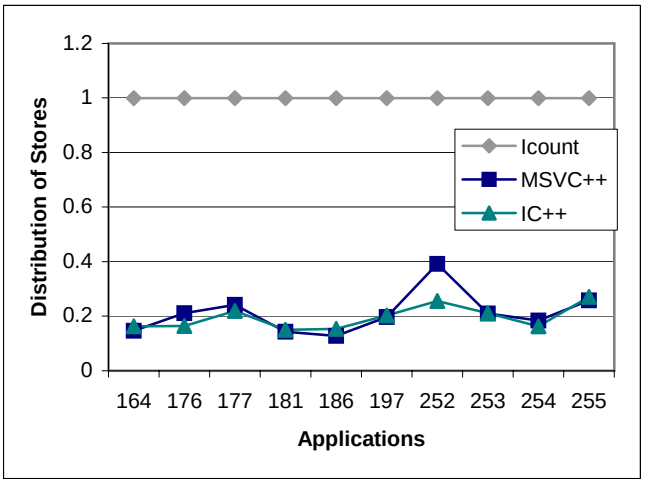

Figure 5.4 Comparison of Stores Retired

There is no visible difference between the two compilers in the distribution of branch instructions. Figure 5.5 shows the comparison. The Intel C++ compiler should be aware of the type of branch predictors used in the architecture. Hence the number of mispredicted branches was compared to check if this prior knowledge translates into fewer mispredictions.

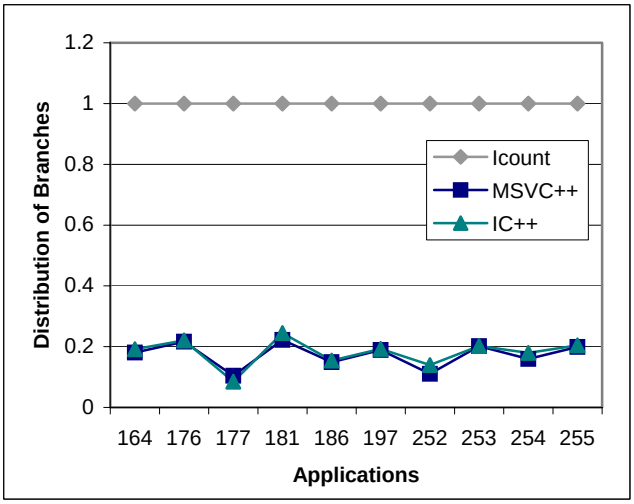

Figure 5.5 Comparison of Branches Retired

Figure 5.6 shows the comparison of mispredicted branches with respect to the overall branch instructions for each of the compilers. The ratio of mispredcited branches is less than 10% in both cases and there is not much of a difference between the two compilers.

Since the distribution of load, stores and branches were similar for most applications, a comparison was made of the other instructions that make up the total count. It is clear from Figure 5.7 that the other instructions contribute to about 35% of the total instructions. Also, 252.eon has a reasonably higher percentage of such

instructions for the Intel compiler, making up for the reduction in loads and stores. These other instructions could be multimedia specific instructions that the Intel compiler has taken advantage of and maximized the performance.

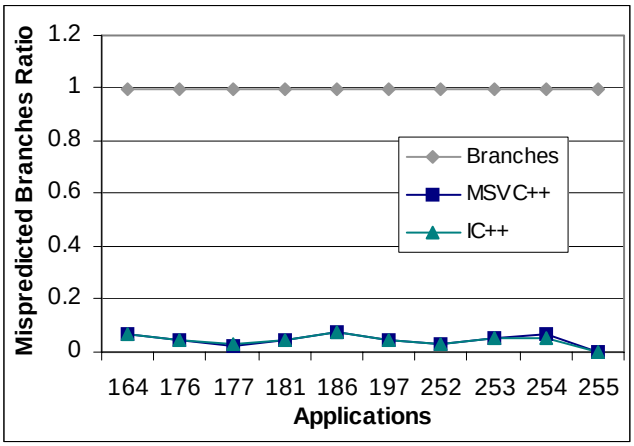

Figure 5.6 Comparison of Mispredicted Branches Retired

The same can be said about 177.mesa, though the difference in percentages of other instructions is not very pronounced. The I-level cache misses were also compared to study the effect of compilers on the memory characteristics of the applications. The comparisons of cache misses are given in Figure 5.8. Intel executables consistently had a fewer cache misses for the applications, as the number of references were also less. But there is no difference in the percentage of the cache misses in both cases. 181.mcf, which shows a high percentage of misses, is a large-scale minimum cost flow problem that is solved with a network simplex algorithm.

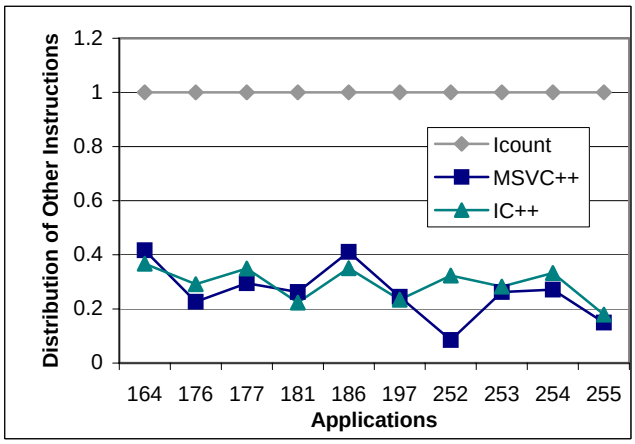

Figure 5.7 Comparison of other instructions

6. CONCLUSION

The execution characteristics of selected SPEC CPU2000 benchmarks were presented for the Microsoft VC++ and Intel C++ compilers. The comparison of clockticks showed that IC++ performed better than VC++ for all considered applications, though the degree of improvement was different for specific applications.

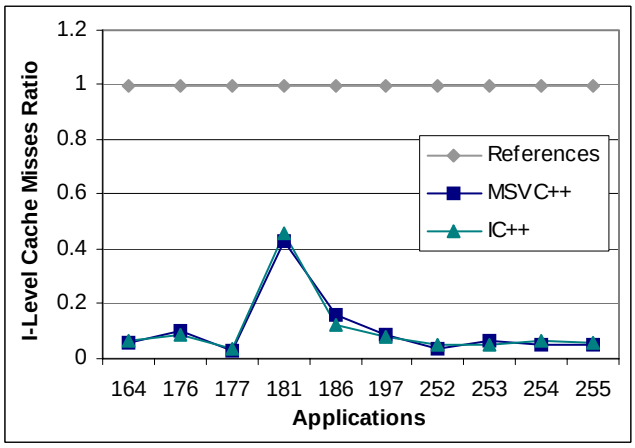

Figure 5.8 Comparison of I-level Cache Load Misses

The Intel C++ compiler performed considerably better than the Microsoft counterpart for applications that involved graphics libraries and computer visualization. Though the study of the instruction count reveals that VC++ generated fewer instructions for certain applications compared to IC++, it should be borne in mind that the applications were optimized for maximum speed. The distribution of load, store and branch instructions were almost similar for both compilers, though there is a difference in the absolute numbers. The number of loads and stores was reduced greatly in the Intel compiler for the computer visualization application. There was no distinct performance gain in branch prediction due to knowledge of branch predictors by IC++. The number of memory references was also reduced by IC++. The performance gain in the visualization and graphics library can be attributed to the effective exploitation of multimedia instructions by the Intel compiler. Collecting events related to this instruction set can testify this fact and give a better understanding of the breakdown of other instructions.

Study of the execution characteristics of applications, by compiling them with different compilers, gives an insight into their strength and weakness and also about the various compiler options that can be used with them. Repeating such analyses with different optimization switches can also be helpful in determining the effectiveness of compilers. Results from such analyses can also be used to design architecture-aware compilers and compilers targeting specific applications.

Microbenchmarks can be used in place of standard benchmarks, so that the user can have more control to test for different performance metrics. Use of microbenchmarks also gives better control to the programmer to test the compiler optimization switches.

This effort can be extended to study a variety of compilers and their optimization switches. Intel C++ compiler can be used to generate executables to be run on Pentium II and Pentium IV and this study can give better insight into processor-specific optimizations.

7. REFERENCES

[1] Talla, D., and John, L.K. Execution Characteristics of Multimedia Applications on a Pentium II Processor. In Proceedings of the IEEE International Conference on Performance, Computing and Communications, (IPCC '00), (February 20-22, 2000), 516-524.

[2] Mehis, A., Ali, R., and Radhakrishnan, R. Using Processor-Specific Optimizations to Maximize Performance on Dell Servers, Technical Article on PowerSolutions, August 2002. www.dell.com/powersolutions

[3] Milenkovic, M., Milenkovic, A., and Kulick, J. Microbenchmarks For Determining Branch Predictor Organization. To appear in Software Practice and Experience, Vol. 34, 2004.

[4] Intel VTune Performance Analyzer, www.intel.com/software/products/vtune/

[5] Stewart, K. E., and White, S.W. The Effects of Compiler Options on Application Performance. In Proceedings of IEEE International Conference on Computer Design: VLSI in Computers and Processors, (ICCD '94), (Oct 10-12, 1994), 340-343.

[6] Bhandarkar, D., and Ding, J. Performance Characterization of the Pentium Pro Processor. In Proceedings of the IEEE Third International Symposium on High Performance Computer Architecture, (Feb 1-5, 1997), 288-297.

[7] Hennessy, J., and Paterson, D. Computer Architecture: A Quantitative Approach. Morgan Kaufmann Publishers, San Mateo, CA, 2003.

[8] Sprunt, B. Pentium 4 Performance-Monitoring Features. In Proceedings of IEEE Micro, Vol: 22, Issue: 4, (July – August 2002), 72-82.

[9] MSDN Help for Microsoft VC++ Compiler

[10] Intel C++ Compiler User Guide,

www.intel.com/software/products/compilers/techtopics/ccug.htm

[11] SPEC CPU2000 Benchmark suite, http://www.spec.org/cpu2000/