The Making of a Compiler for the Intel® Itanium™ Processor

Carole Dulong, Intel Compiler Lab, Intel Corporation
Priti Shrivastav, Intel Compiler Lab, Intel Corporation
Azita Refah, Intel Compiler Lab, Intel Corporation

Index words: compiler, Intel® Itanium™ processor, SDK, SoftSDV, OSV, ISV

ABSTRACT

Intel has developed an Itanium compiler that compiles code written in C/C++ or Fortran languages and generates the assembly instructions for the Itanium™ architecture on Windows NT* and UNIX* platforms. This paper describes how the Intel Itanium compiler was designed and developed in the absence of hardware. The project started more than five years before we had any hardware, and the work on simulators presented many challenges. We discuss three different aspects of the Itanium compiler: the compiler performance analysis, which focused on demonstrating the Intel® Itanium™ processor (ITP) performance on the Spec benchmarks; enabling Operating System Vendors (OSVs) such as IBM to port their Operating System (OS) to the Itanium architecture; and Independent Software Vendor (ISV) enabling work, which focused on enabling ISVs to port real applications on the Itanium architecture.

For each area we describe the methodology used in the absence of hardware, the results achieved, and the lessons learned.

GOALS OF THE INTEL ITANIUM COMPILER

The project goals for the Intel Itanium compiler have evolved over time. Initially the goal was to create a reference compiler, which would demonstrate the Itanium processor performance on a few key benchmarks. A few years into the project, the strategy was changed, and we decided to develop a robust product compiler that would not only demonstrate performance, but also enable Independent Software Vendors (ISVs) and Operating System Vendors (OSVs) to port their applications and Operating Systems (OS) to the new Itanium architecture. Making the Intel compiler a product would give it credibility among our other compiler vendors, and it would create an incentive in the compiler industry to achieve as high performance as possible on this new architecture.

We knew from the beginning that many ISVs would not want to switch vendors, so it was important to incite the other compiler vendors to expend their resources on the performance aspect of the Itanium compilers.

Time Line

It was well understood from the beginning of the Itanium project that compiler technology was a key ingredient of the project. Therefore, the compiler design was started at the same time as the architecture definition and the chip design: i.e., in late 1994.

Key Phases and Milestones

The key phases and milestones for the project were as follows:

- 1994 – 1997: Design
- 1996 – 1999: Development
- Q4, 1996: Windows NT boots to command prompt with Intel's compiler in two months.
- Q3, 1997: First Non-NT OSV Engagement

*Other names and brands may be claimed as the property of others.

1® Intel is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

™ Itanium is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

2* Other names and brands may be claimed as the property of others.

- Q2, 1999: Monterey64 boots with Intel's Compiler on the simulator
- 1999 – 2001: Performance tuning on simulator, then on hardware
- Q1, 2000: First access to hardware

Software Development Kits (SDKs) were released to firmware (FW) developers, ISVs, and OSVs very early in the project:

- Q1, 1996: SDK 0.1 first FW release
- Q4, 1997: SDK 0.3 64 bit analyzer (for ISVs to make their application 64-bit clean)
- Q2 - Q4 1998: SDK0.4 and SDK 0.5, enabling of OS and FW development
- Q1, 1999 SDK0.6 release criteria included Intel assembler and building of NT OS 4.0
- Q3, 1999: SDK1.7 first release with strong emphasis on applications. Release criteria applications included 3D Studio Max (C/C+), Games and Nag F90 (Fortran)
- Q1, 2000 SDK2.0 last simulator-based release
- Q3, 2000 SDK5.0 first HW-based release, enabling large application vendors such as Ansys, Nastran, Oracle, SQL, and Mentor Graphics

Tools

When the Itanium compiler and OS development started, a simulation environment was defined to run programs on a functional simulator called Gambit. The environment, called application mode, was used to test the very first Itanium compiler by providing basic runtime capabilities including file I/O and memory management. The compiler was used to get the Windows* and System V UNIX OSs up and running on the functional simulator. The OS support was limited to loader, OS kernel, and basic OS functionality.

A Software Development Vehicle (SDV) was designed to include OS drivers and a loader to start an OS on the functional simulator under a debugger. The tool was named SoftSDV [2]. It was widely used to debug and test the OS, compilers, and 64-bit applications. A complete Windows OS and System V UNIX OS along with runtime support was developed and run on SoftSDV (see Figure 1).

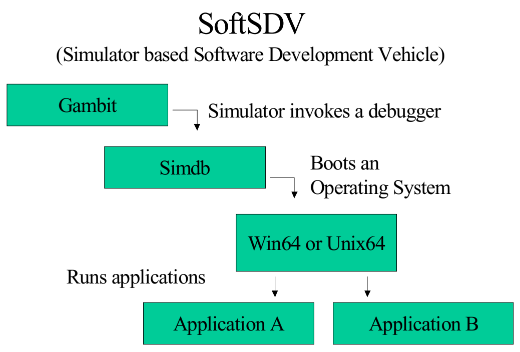
SoftSDV
(Simulator based Software Development Vehicle)
Gambit
Simulator invokes a debugger
Simdb
Boots an Operating System
Win64 or Unix64
Runs applications
Application A
Application B

Figure 1: SoftSDV

A performance simulator called Emerald was also developed. It was run in application mode, on top of Gambit. For performance analysis of the compiler on Emerald, we mostly used the SPEC CPU95* benchmarks. To run these benchmarks on the simulator in less than 24 hours per benchmark, we had to use the lite inputs, a subset of the reference inputs that was borrowed from HP. Results from Emerald include cycle count and micro-architecture statistics such as cache hit rates, branch mispredict, and instruction distribution. However, it is important for a compiler to know where events such as cache misses and branch mispredicts happen in order to prevent them. A special version of Vtune™, called Vtune++, gathered event information, including the Instruction Pointer (IP) from the Emerald simulator. It used the core GUI functions of Vtune, but could provide information at the bundle level, such as cache misses or time-based IP sampling

PERFORMANCE ANALYSIS

The compiler performance analysis has several goals:

- *Performance projection.*
- *Guiding and validating the optimization development.* Analysis of compiler-generated code shows where and what optimizations are needed. Performance analysis is also important to check if optimizations are doing what they are supposed to do, after they have been developed.
- *Performance regression testing.* Interactions between different optimizations can be complex, and

* Other names and brands may be claimed as the property of others.

™ Vtune is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

regression testing is needed to ensure that optimizations keep working as the compiler evolves.

The SPEC CPU95 benchmarks were used for performance analysis and projections. By the end of 1997, the compiler reached the long awaited level of 30 SPECint95*. At that time, the chip design went through two die diets in a row, and the compiler performance stayed at the 30 SPECint95 level for more than six months. New compiler optimizations were making up for lost silicon performance. Floating-point performance (shown with squares in the graph) was not affected as much by the die diets as was integer performance. However, the compiler team developed a new High-Level Optimizer (HLO) to take advantage of the Itanium™ architecture, but the HLO did not come on line before the first half of 1999 [1]. Figure 2 shows the performance improvements over time on SPEC CPU95 with the Intel Itanium compiler.

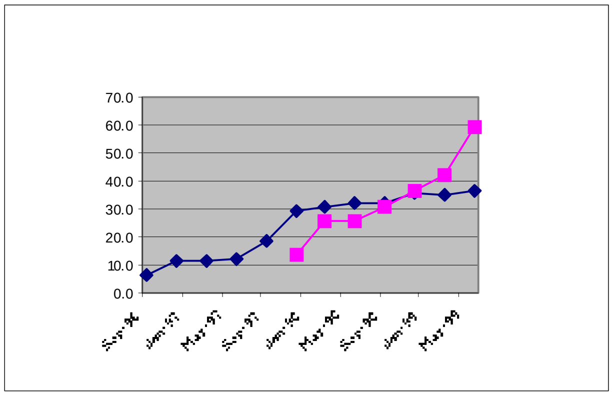

Figure 2: SPEC CPU95* performance over time

The chart in Figure 3 shows the difference between simulator-predicted performance and actual hardware performance. A negative number means that the simulator was optimistic and that the hardware did not perform as fast as the simulator. SPECint95 performance was predicted to be 13% faster than actual performance. The difference was partially due to the use of lite inputs on the simulator, which did not always correlate well with the reference inputs used on hardware. The other source of difference was inaccuracies in the simulator.

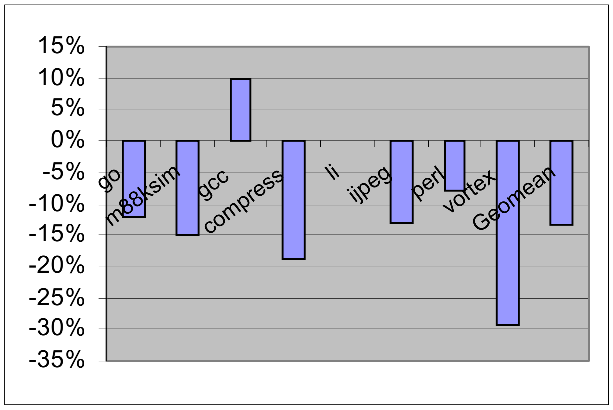

Figure 3: Simulator accuracy on SPECint95*

Overall, the floating-point performance projection was more accurate than the integer projection, see Figure 4; since it came within 3% of the hardware performance. However, the simulator was very inaccurate on benchmarks that were memory intensive. The memory model of the simulator was fairly simplistic and did not model overlapping requests to memory. We were fortunate to have two benchmarks grossly underestimated, and two benchmarks grossly overestimated. The errors compensated each other and gave an overall good prediction.

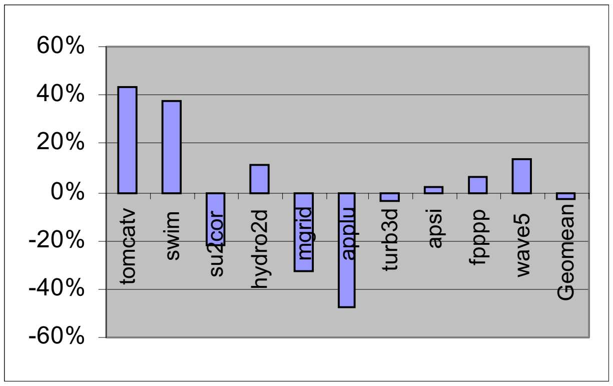

Figure 4: Simulator accuracy on SPECfp95*

ENABLING AN OPERATING SYSTEM VENDOR

The first enabling Software Development Kit (SDK) shipped in November 1997 to multiple Operating System Vendors (OSVs). The goal of the SDK was to provide OSVs with the tools necessary to port their Operating Systems (OSs) to the Itanium architecture. The initial

* Other names and brands may be claimed as the property of others.

™ Itanium is a trademark of Intel Corporation or its subsidiaries in the United States and other countries.

® Intel is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

™ Itanium is a trademark of Intel Corporation or its subsidiaries in the United States and other countries.

contents of the Enabling SDK were the compiler, the assembler, and a functional simulator called Gambit. These tools were all UNIX-based. But in September 1998, the simulator no longer supported the UNIX platforms; it only supported Windows platforms. Every one working on a port to the Itanium architecture had to have a Windows-based system to be able to continue. This was a hard sell to non-NT OSVs.

During the life of the Enabling SDK project (10/97 to 12/99), there were seven major releases, occurring approximately once per quarter.

The Monterey Story

Although the Enabling SDK was provided to several OSVs [3], Gemini64 (the Operating System provided by SCO) was our reference platform. We held weekly meetings to discuss status, progress, plans, and of course, compiler bugs or needed features.

In October 1998, the compiler teams engagement with IBM accelerated. IBM had already decided to port their AIX* Operating System using the Intel compiler. At the same time the announcement was made, SCO, IBM, and Sequent decided to collaborate on their OS effort, and they started the Monterey Operating System project. This OS was designed to take advantage of strengths in each code base.

Intel compilers had never before been used to port operating systems, so the testing done in the compiler group did not always catch problems seen while porting the Monterey OS. We were working closely with the OS team and providing them pre-release compilers to test the kernel. After a few releases, the compiler team decided to make the OS build a part of the compiler release criteria.

The main challenge on the Enabling SDK Project was that there were no constant factors:

- Aggressive performance-driven optimizations were under development, making the compiler unstable at times. A compiler simply meant for OSV-enabling would have been more conservative in its optimization work.
- The OS was also under constant change and development.
- Development was underway on the simulator, where the changes for new External Architecture Specification (EAS) revisions were incorporated.

[*] Other names and brands may be claimed as the property of others.

Because of all the changes, debugging was a challenge.

Testing the UNIX SDK was another challenge:

- There was no runtime environment available on UNIX. We had written a minimal set of runtime functions to interface with SoftSDV for our testing purposes. Each OS vendor had a different runtime environment.
- Our UNIX tool chain did not include a full-feature linker. Again OSVs were using their linker, which often behaved differently from what we were using.
- Not having a UNIX-based SoftSDV forced us to change our testing tools to work across platforms.

In Q3 1999, the bring-up plan gave all OS vendors a two-week period to bring up their OS on the Itanium Hardware in Intel Dupont. Monterey was scheduled for the week starting September 14. It took Monterey less than 24 hours to bring up the kernel on SDVs. According to IBM, this was the fastest they ever booted an OS.

The Monterey binary that booted on Itanium hardware was built with the Electron Compiler with optimizations enabled. No workarounds were required! This was quite a success story for the compiler and SoftSDV. [http://aix5l.ihost.com/innovations/index.shtml]

The presence of compilers for the porting effort along with simulators and other tools (enabling SDK) are what made the difference when hardware was available.

PRE-SILICON COMPILER TESTING AND VALIDATION METHODOLOGY

The Intel Itanium compilers were tested in two different environments to cover the C/C++ and Fortran compiler language functionality and to test optimizations and code generation, based on Itanium.

Testing Environments

Compiler testing was divided into two parts: Application-Mode testing on the functional simulator Gambit, and System Mode testing using SoftSDV on the Win64[*] Operating System (OS).

[*] Other names and brands may be claimed as the property of others.

® Intel is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

™ Itanium is a trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Application-Mode Testing: Compiler language conformance tests, coverage for optimizations using C/C++, and Fortran compiler test suites were mostly done in application mode. This resulted in fast turnaround on test results and overnight test runs. However, application mode testing was limited to basic runtime file I/O, and memory management. It could not be used to exercise the C/C++ language features, which required a complete Operating System (OS) runtime environment. It also could not test recovery code, since Gambit did not simulate page faults.

SoftSDV-Testing and Debugging Environment: The C/C++ functionality in the areas of structured exception handling, C++ exception handling, runtime type information, and C++ I/O required a complete OS runtime support. SoftSDV was the only way to exercise these C++ functionalities and the compiler and operating system stack unwinding support. Unalign and NAT exceptions specific to Itanium are handled by the OS, and so the compiler-generated code for speculation can only be tested with the OS. We relied on SoftSDV to have comprehensive test coverage with the Itanium compilers.

Compiler tests running on a simulated Win64 operating system had a slow turnaround time. This environment enabled a SoftSDV cross-testing environment for the compilers. Tests were compiled on the host system; SoftSDV then booted the operating system on the simulator and executed the compiler-generated executable.

Debugging programs on SoftSDV was very challenging. The debugger and OS were still in the development phase. Moreover, the OS/debugger interface was not very stable. Several times we ran into OS problems as we debugged compiler issues. Compiler debug engineers had to understand the interaction between the debugger and the OS. They also had to learn the OS interface for trapping the exceptions. The compiler functionalities, which were the hardest to debug, were structured exception handling and C++ exception handling.

A good example of this would be that we would set a breakpoint to trap an exception raised by the an exception-handling test case, and we would end up in the OS exception dispatcher, having trouble in unwinding. We would then enable OS breakpoints to debug the unwinding problems, to uncover the compiler or OS stack unwinding issues.

Netbatch Tools

To improve the test coverage for the compiler, a networked batch-testing software was integrated with the compiler test tool, widely known as Netbatch. With the Netbatch compiler, testing was completed in 16 to 18 hours. Netbatch allowed regression testing to be run nightly. This allowed fast turnaround on the bugs introduced daily.

Configuration: The test tools were modified to use the Netbatch APIs. The client systems known as Garcons were set up with SoftSDV tools, a Win64 OS, and drivers. The compiler-testing tool called TC, is written in Perl and uses the Netbatch capability to distribute compiler tests across all the machines in the Netbatch pool. TC works in the SoftSDV environment and in application mode.

Challenges with changing OS and SoftSDV: OS development was still in progress, and SoftSDV was constantly being modified for additional support for new drivers. Every upgrade for a new version of the OS and SoftSDV required changes to be made to the testing environment and testing tools, such as TC. We had to re-install the new versions of SoftSDV, the OS, and related tools on all Netbatch client systems before testing the compiler in the new environment.

SDK Development and Test Cycle

Intel compiler and OS teams worked closely with Microsoft, HP, and SCO to define Software Conventions and Runtime Architecture for the Itanium architecture. An NT Application Binary Interface (ABI) was defined to support the Windows NT operating system features for Itanium. We also defined a System V UNIX Processor-specific ABI for the Itanium architecture. The conventions, and OS ABI, formed a norm for the OS and for tool development on NT and UNIX platforms. On the NT platform, a complete Software Development Kit (SDK) was being developed, see Figure 5. It included the OS, SoftSDV, and the development tools: compilers, the assembler, the linker, and OS runtime libraries

[*] Other names and brands may be claimed as the property of others.

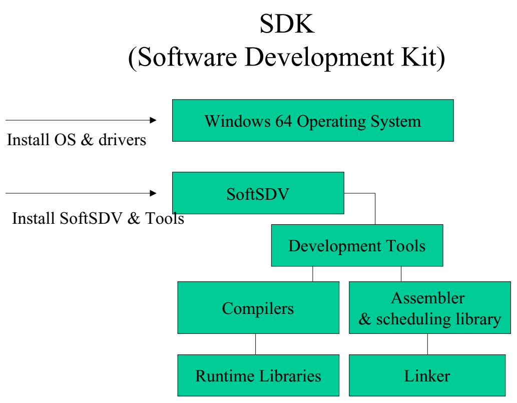

Figure 5: Software Development Kit

Microsoft worked with the Intel linker team to implement the ABI support and extended their executable file format to include 64-bit virtual addresses. The new file format was defined as an extension to the existing PE32 and was called PE32+. The compiler and assembler implemented the support for the PE32+ and the NT ABI. For example, support for function pointers known as "plabel," unwind tables, exception handling tables, and various 64-bit relocations were added to the tools and to the OS runtime. The SoftSDV included firmware and drivers, which were compatible with the Windows NT ABI. Figure 6 describes the dependencies within the SDK components. These dependencies made it hard to stabilize the SDK across engineering groups. All the OS and compiler ABI compatibility issues were tested on the simulator to be silicon ready. The pre-silicon releases enabled porting applications to 64-bit, so they would be ready to run on the Itanium hardware.

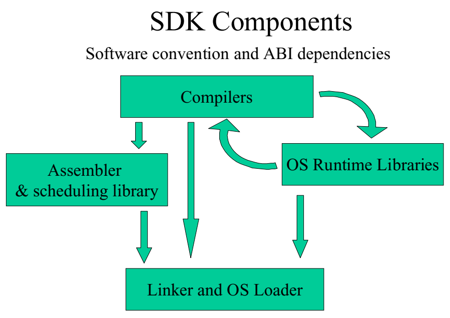
SDK Components
Software convention and ABI dependencies
Compilers
Assembler & scheduling library
OS Runtime Libraries
Linker and OS Loader

Figure 6: Dependencies within SDK components

Compiler Language Extensions for the Itanium Architecture

We defined extensions to the Windows NT C compiler to support the Win64 ABI. Among them were the following:

- pragma data_seg to define data as short or long was introduced in the Microsoft and Intel C compilers
- __ptr64 and __ptr32 to support 64-bit and 32-bit pointer types
- several compiler intrinsics were implemented to support the NT operating system functionality requirements in place of inline assembly

SoftSDV was used to test the mixed 32-bit and 64-bit pointer support by forcing programs to be loaded at higher than 2**32 bit address.

CONCLUSION

Developing a compiler in a virtual environment presented many challenges, but we were able to do everything that we would have done on hardware: performance projections, nightly regression testing, and Operating System Vendor (OSV) and Independent Software Vendor (ISV) enabling. Since the Intel Itanium processor was the first of a new architecture, no other methodology was possible, but it had a high cost in people to develop and maintain all these complex tools. We also had to limit the scope of performance analysis and stability testing to accommodate the simulator speed. We have learned that hardware is needed at least six months before the first release of a compiler product in order to analyze the performance and stability of real applications, and not only benchmarks and test suites.

ACKNOWLEDGMENTS

We acknowledge the work of all members of the Intel Compiler Lab. Special thanks to Tom Joyce, Bill Savage, Suresh Rao, and Steve Skedzielewski for their thorough reviews.

REFERENCES

[1] Carole Dulong, Rakesh Krishnaiyer, Dattatraya Kulkarni, Daniel Lavery ,Wei Li, John Ng, and David Sehr, "**An Overview of the Intel IA-64 Compiler**," *Intel Technology Journal,* Q4, 1999.

® Intel is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

™ Itanium is a trademark of Intel Corporation or its subsidiaries in the United States and other countries.

[2] Richard Uhlig, Roman Fishtein, Oren Gershon, Israel Hirsh, and Hong Wang, "**SoftSDV: A Pre-silicon Software Development Environment for the IA-64 Architecture**," *Intel Technology Journal*, Q4, 1999.

[3] Kathy Carver, Chuck Fleckenstein, Joshua LeVasseur, and Stephan Zeisset "**Porting Operating System Kernels to the IA-64 Architecture for Pre-silicon Validation Purposes**," *Intel Technology Journal*, Q4, 1999.

AUTHORS' BIOGRAPHIES

Carole Dulong has been with Intel for over eleven years. She is a principal engineer and staff member of the Intel Compiler Lab. Prior to joining Intel's Microcomputer Software Laboratory, she was with the Itanium architecture group, where she headed the Itanium multimedia architecture definition and the Itanium experimental compiler development. Her e-mail is carole.dulong@intel.com

Priti Shrivastav joined Intel's Itanium compiler code generator group in September 1995. She moved on to the Enabling SDK and Development Group as the technical lead in 1996. For three years she was the Itanium Compiler Evaluation team manager and now co-manages the IA-32 and IPF Product Compiler team within the Intel Compiler Lab. Priti received her Masters degree in Computer Science from Ohio University in 1983. Her e-mail is priti.shrivastav@intel.com.

Azita Refah has been with Intel for over eleven years. She leads the Linux development team in the Intel Compiler Lab. She graduated from Northeastern University (in Boston) with a B.S. degree in Computer Science. Her e-mail is azita.refah@intel.com