# *Tritanium:* Augmenting the Trimaran Compiler Infrastructure To Support IA-64 Code Generation

Yogesh Chobe, Bhagi Narahari, Rahul Simha
Department of Computer Science
The George Washington University
Washington DC 20052
{ylchobe, narahari, simha}@seas.gwu.edu

Weng-Fai Wong
Department of Computer Science
National University of Singapore
3, Science Drive 2
Singapore 117543
wongwf@comp.nus.edu.sg

## ABSTRACT

This paper describes Tritanium, an on-going project to enhance compiler optimization tools for Explicitly Parallel Instruction Computing (EPIC) architectures, in particular, the Intel's IA-64 Itanium processor [1]. The project we describe involves augmenting a popular research compiler suite, the Trimaran compiler infrastructure [3], with the capability to generate IA-64 code so that its well-documented capabilities can be used to explore compiler optimizations by testing the resulting code directly on the IA-64 processor. We have validated the tool with preliminary SPEC-based results that show performance comparable with gcc and sgicc; this suggests that the tool can be used for exploring and testing compiler optimizations to improve performance.

## 1. INTRODUCTION

In this paper, we describe the Tritanium (Trimaran for Itanium) project: an effort to enhance the Trimaran research compiler to produce code for the IA-64 EPIC processor. Since EPIC processors rely strongly on the compiler to extract instruction-level parallelism [8], an infrastructure that provides the ability to experiment with compiler optimizations helps compiler-writers save time in algorithm development. The Trimaran compiler suite is one such popular research compiler that allows experimentation at three levels: front-end compiler techniques (for example, loop unrolling, program transformations, etc), back-end resource optimizations (for example, scheduling and register allocation), and architectural exploration (for example, changing the number of registers, etc) [3]. Unfortunately, the architecture for which Trimaran was developed (a precursor to IA-64) was not realized in hardware and remains inside Trimaran's simulator. Therefore, it is difficult to obtain realistic measurements. We believe the compiler community, in particular, the growing Trimaran community, would benefit from the ability to have the tools generate code directly for the IA-64.

The importance of tools such as Trimaran and Tritanium is well-established. EPIC architectures, including the IA-64, provide hardware features that have the potential to exploit high levels of Instruction Level Parallelism (ILP). Because EPIC processors rely heavily on the compiler to extract parallelism and perform instruction scheduling, the compiler is now the focus of experimentation in optimization strategies. Since experimentation requires easy access to the internals of the compiler and the ability to measure performance at various stages of the compiler process, Trimaran also consists of a full suite of analysis and optimization modules, as well as a graph-based intermediate language. Optimizations and analysis modules can be easily added, changed or bypassed, thus facilitating highly targeted compiler optimization research.

This paper describes the design issues involved in extending and modifying the Trimaran Infrastructure to generate code for the Itanium processor. These include using the hardware-description langugage of Trimaran (MDES) to support Itanium, providing a code-generator and handling issues specific to IA-64 in the compiler front-end. The performance of the Tritanium is being evaluated by comparing its performance to that provided by gcc and sgicc on a number of benchmarks. Current results include some of the SPEC benchmarks with work going on to make the compiler robust enough for more. These results indicate that the Tritanium infrastructure produces code whose performance is comparable to that produced by gcc and sgicc.

## 2. TRIMARAN - A BRIEF DESCRIPTION

Trimaran consists of the following components.

- An architecture description language called MDES [5][6] to describe the target processor.
- A compiler front-end for the C programming language, called IMPACT, which performs parsing, type checking and a large suite of high-level (i.e. machine independent) classical and ILP optimizations.
- A compiler back-end, called Elcor, parameterized by a machine description, that performs instruction scheduling, register allocation, and machine-dependent optimizations.

- An extensible IR (intermediate program representation) that has both an internal and textual representation (called Rebel), with conversion routines between the two. This IR supports modern compiler techniques by representing control flow, data and control dependence, and many other attributes.
- A cycle-level simulator of the HPL-PD architecture, which is configurable by a machine description and provides run-time information on execution time, branch frequencies, and resource utilization. This information can be used for profile-driven optimizations as well as to provide validation of new optimizations.
- An integrated Graphical User Interface (GUI) for configuring and running the Trimaran system. Included in the GUI are tools for the graphical visualization of the program intermediate representation and of the performance results.

Trimaran allows various optimization modules to be inserted into the optimization path; thus providing researchers with an infrastructure with which to test compiler optimization algorithms on the HPL-PD architecture. For example, researchers can test a new register allocation algorithm by replacing the built-in register allocation module with their algorithm. The proposed Tritanium infrastructure, as a result of it being developed on top of the Trimaran infrastructure, provides the same flexibility and thus allows researchers to test their optimization algorithms on the Itanium/IA-64 architecture.

## 3. IA-64 OVERIEW

The Intel Itanium is an IA-64 EPIC architecture with a 64 bit processor supporting most of the EPIC features provided in HPL-PD [1] [7]. However, there are some major differences between the two architectures. For example, Itanium does not provide a division instruction and has a different register stack engine. Besides this, other differences include the 32 bit addressing of HPLPD and the 64 bit addressing of the IA-64 processor. In summary, the IA-64 supports features such as two system environments (32 and 64 bit modes), hardware support for speculation (control and data), predication, and rotating registers. In addition to predication, the compiler can use branch predict instructions which can communicate early indication of target address. IA-64 avoids the unnecessary spilling and filling of registers at procedure call and return interfaces through compiler controlled register renaming. Compiler optimizations which utilize these EPIC features can be developed and tested on Trimaran for the HPL-PD and on the Itanium processor using our proposed augmentation- Tritanium.

## 4. DESIGN AND IMPLEMENTATION OF TRITANIUM- AN OVERVIEW

Tritanium has been implemented by writing a back end code generator module that accepts Elcor output and writes out the IA-64 assembly file. The tasks involved in the implementation of Tritanium also required changes to other existing modules in Trimaran as listed below:

- Creating a hardware description of IA-64 in MDES. The Trimaran Infrastructure uses MDES, a high-level machine description language, to describe the target processor. The IA-64 had to be described as a first step towards retargeting the compiler for IA-64. The number of registers, number of functional units and their latencies had to be described. Operations that had characteristics different from HPL PD were described in the modified Elcor file along with some changes to follow the runtime conventions of IA-64.
- Changes to IMPACT (the front-end) to support the runtime conventions of IA-64 and branch semantics. Impact is the front end compiler in Trimaran that performs machine independent optimizations prior to resource allocation. Changes were made to the instruction selection module to generate code that can be mapped to the IA-64 processor. The runtime conventions of the IA-64 for parameter passing also had to be built into this stage.
- Supporting the rotating registers of the IA-64 in Elcor (the back-end). HPL-PD assumes a separate set of rotating registers from regular registers. This is not true for the IA-64. Hence in the post-processing phase of modulo scheduling, the current predicate registers are saved before rotating. In Elcor, two registers are considered equal even if one is rotating and other is not. However, this is not the case in IA-64 and therefore affects the correctness of dependency inferences and the correctness of scheduling. Hence changes were made to distinguish between them.
- Supporting instructions that don't have hardware implementation on the IA-64. We included code to replace instructions such as DIV and MOD, which do not have hardware implemention on the IA-64 by generating the appropriate routines to perform such operations at the instruction selection phase in IMPACT.
- The IA-64 code generator module. The output generated from Elcor is similar to IA-64 code, but is still in the syntactical representation of HPL-PD. This output was transformed to the IA-64 assembly file by the Code generator module. Essentially, the code generator module maps the HPL-PD instructions to the equivalent IA-64 instructions.

## 5. VALIDATION OF TRITANIUM

The Tritanium system was validated initially using the NUE [2] (Native User Environment) Simulator from HP and later using an Itanium SDV machine provided by Intel Corporation. The following Benchmarks / Test Suites were used while developing Tritanium to test the implementation validity of the compiler.

1. Trimaran Test Suite: The test programs that came with Trimaran were the initial targets for the compiler. This suite includes small programs that test different aspects of the compiler and various aspects of the language.
2. SPEC Benchmarks: Spec (The Standard Performance Evaluation Corporation) is the publisher of benchmark suites that aim to provide a standardized set of relevant benchmarks that can be applied to obtain a performance evaluation of architectures.

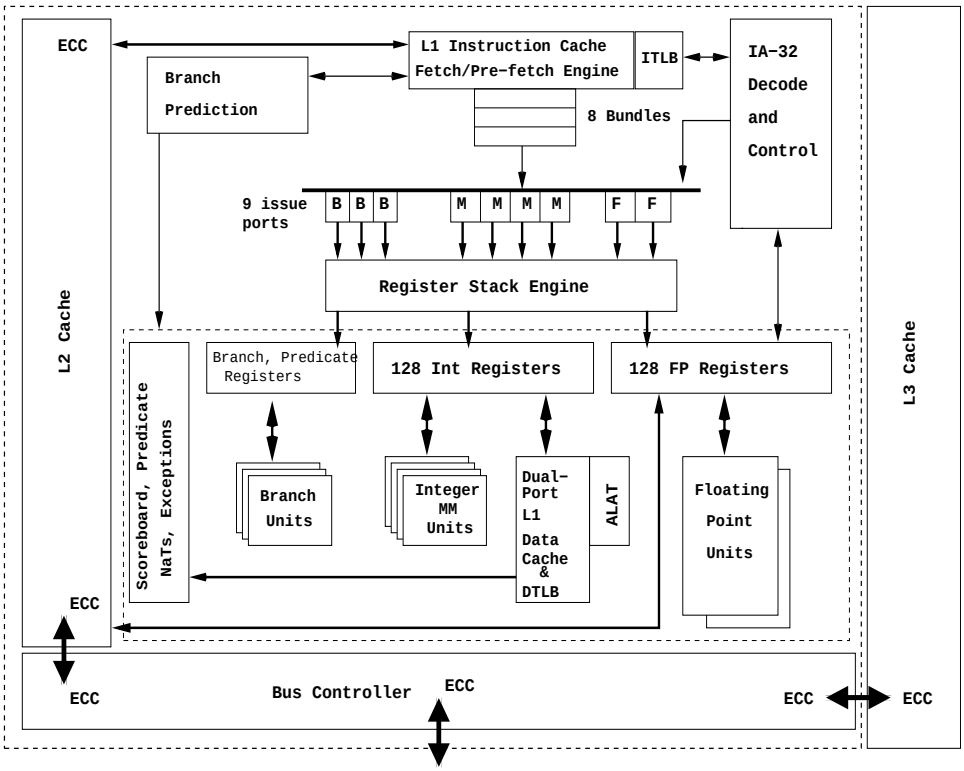


**Figure 1: Block Diagram of Itanium Processor**

3. MediaBench: MediaBench is a suite of benchmarks from UCLA, which is provided as a tool for evaluating and synthesizing multimedia, and communication systems.

4. DIS: The Data Intensive Systems Benchmark Suite developed by the Atlantic Aerospace Electronics Corporation and the Boeing company and ERIM Int. Inc.

5. Olden: The benchmark suite from Princeton which was used to test Olden, which is the language and runtime system for parallelizing programs using dynamic data structures.

Currently, we have not completed testing of all the programs in the benchmarks suites but anticipate completion incrementally as the infrastructure is released. The list of currently validated benchmarks will be available at the Tritanium website [4]. The performance of the code generated by the Tritanium infrastructure (using the optimization modules currently available in Trimaran) for some test programs is presented in the table below. As the table shows, the performance of Tritanium is comparable to gcc with the current implementation, which has not yet been tuned for performance. With further work on tuning, which will include some basic required additions like support for superblock and hyper block region formation, due to which we expect an improvement in performance.

For the performance table above, we compiled the programs using gcc with the -O3 option and sgicc with the -O3 option. These options correspond to the full range of optimizations

**Table 1: Performance Comparison of Tritanium**

| Test Program | Tritanium | gcc | sgicc |
|---|---|---|---|
| 052.alvinn | 115 | 117 | 100 |
| 022.li | 102 | 100 | 101 |
| 026.compress | 112 | 110 | 100 |
| 124.m88ksim | 108 | 116 | 100 |

provided by these compilers - for example, sgicc in the -O3 option forms hyperblock regions. In constrast, the Tritanium compiler was used with the basic block region formation option. At the time of running the tests, the superblock and hyperblock region formation techniques were not stable enough (for IA-64 code) to be included in the tests. The numbers in the table are based on the execution times on the Intel Itanium SDV. The entries in the table show the relative performance of the compilers, on each benchmark application, and not the absolute execution time of the programs. The numbers in the table were obtained by dividing all the execution times with the least execution time of that program - the lowest execution time was almost always obtained with sgicc and thus the sgicc performance has been normalized to 100%.

## 6. CONCLUSIONS AND FUTURE WORK

The Tritanium infrastructure will allow compiler researchers to develop, test, and validate ILP optimization techniques by examining performance of their techniques on the Itanium processor. Currently, we have been able to compile a number of small programs and some applications from var-

ious benchmark suites using the base set of optimizations provided in Trimaran. The generated code, inspite of using some conservative optimizations, shows performance comparable to other compilers like gcc. We expect to support the full range of region formation methods, such as superblock and hyperblock, soon. In addition, enhancements to Tritanium will include some basic required additions such as support for compiling varargs functions, and additions of some unimplemented instructions. While the initial phase of Tritanium development involved generating correct code and making the compiler compliant with the standard benchmarks, the next phase involves performance tuning of the compiler.

## 7. ACKNOWLEDGMENTS

The authors would like to thank Intel Corp for the Itanium SDV machine given to The George Washington University. The authors would also like to thank Aakash Kambuj, formarly at National University of Singapore for his contributions in the initial implementation of Tritanium.

## 8. REFERENCES

[1] The itanium architecture manuals. *http://developer.intel.com/design/itanium/manuals/index.htm.*

[2] Nue, ia-64 simulator. *www.software.hp.com/ia64linux.*

[3] Trimaran web site. *www.trimaran.org.*

[4] Tritanium web site. *www.seas.gwu.edu/ťri and www.seas.gwu.edu/ÿlchobe/tritanium.html.*

[5] S. Aditya, V. Kathail, and B. R. Rau. Elcor's machine description system: Version3.0. *HPL Technical Report HPL-98-128. Hewlett-Packard Laboratories*, July 1998.

[6] J. C. Gyllenhaal, W. m. W. Hwu, and B. R. Rau. Hmdes version 2.0 specification. *Technical Report IMPACT-96-3. University of Illinois at Urbana-Champaign*, 1996.

[7] V. Kathail, M. Schlansker, and B. R. Rau. Hpl-pd architecture specification:version 1.1. *HPL Technical Report HPL-93-80 (R.1). Hewlett-Packard Laboratories*, February 1994.

[8] M. Schlansker and B. R. Rau. Epic: An architecture for instruction-level parallel processors. *HPL Technical Report HPL-1999-111. Hewlett-Packard Laboratories*, February 2000.