Future directions in the interaction of architectures and compilers in high-performance processor design.

Position paper, presented at the HICSS'97 Task Force of this title

Rudolf Eigenmann
Purdue University, School of Electrical and Computer Engineering

January 10, 1997

The interface between compilers and architectures has been an intrinsic part of every design of a compiler or a new architecture. Hence, the appropriate form of this interface may be considered an old and well-discussed question.

Are there any new issues? Certainly, as machines change, their instruction sets change, requiring compilers to respond with new code generation and new optimization techniques. By "instruction set" I mean the entire machine functionality that is at the compiler's disposal for driving a program execution. This instruction set is expected to change significantly with new generations of machines. Instructions may increasingly deal with latency hiding (multithreading, prefetch, cache-control), parallelism exploitation (at instruction level or coarse-grain), heterogeneous machine configurations, and a range of revolutionary issues introduced by future "petaflop" computers and their interfaces.

For the sake of this essay, however, I will not discuss the challenges created by such new functionality. Instead, I will discuss yet another challenge that we will have to meet: Although, we may succeed in creating compilation techniques that take advantage of the instruction sets offered by new architectures, we have begun to experience that it is no longer sufficient to accumulate all these advanced methods and tricks into one big compilation tool. While we could expect of early compilers that a fixed percentage of a machine's performance was consumed by the inefficiency of the compiler-generated code, this is no longer true. With the raise of parallel machines we have already begun to learn that the efficiency of compiled code can decrease as the architecture scales. Today, as the complexity of machines and the range of application problems keeps increasing, we are facing the situation that program efficiencies can reach very low levels. Our compilers, although exploiting architectural features well in some programs and contexts, may in fact decrease performance for just the range of problems that the eventual user is interested in.

As a consequence it is no longer so that architects can build new components, and that compiler writers can invent new optimization techniques, assuming that they will incrementally improve application performance. A combined compiler/architecture solutions is necessary. If it has been true all along that there has to be a match between machine, compiler and the range of applications that we are interested in, this design principle is now becoming of paramount importance. Strong interactions are called for between compiler writers and computer architects, but also between these two communities and the authors of the eventual applications.

How can we address this challenge? Generally, all of us have to become much more knowledgeable about the applications, the input data, the software, the hardware, and the environment of a program

execution. We have to become more aggressive in extracting information from these information sources and creating a knowledge base. Architectures and compilers can no longer afford to be naive in applying their techniques. Instead, we have to apply them more intelligently – driven by the knowledge base. Where static, compile-time knowledge is insufficient (and we have learned that this is often the case) we need to develop runtime query and optimization techniques. I will term this the "adaptive compiler/architecture interface".

What are the new roles of compilers and architectures in this adaptive scenario? The role of the compiler will be to use knowledge that becames available at program execution time and perform optimizations dynamically. In part, the compiler may need to speculate on certain optimizations, and verify them after the information has become available - with proper safe guards if the speculation fails. The architecture's role will be more than just providing the "instruction set". Runtime optimizations generally do runtime bookkeeping operations, which introduces overhead when done entirely in software. Hardware assistance is needed. Examples are fast range checks to monitor the index address of an array, data-dependence tests "in hardware", and fast comparisons of control parameters to test if the current execution context is the same as at the last subroutine entry.

Where are we on the path to these goals? We have developed a framework for adaptive and speculative compilation in the Polaris compiler. Polaris is an advanced program analysis infrastructure, including new techniques such as symbolic range analysis, demand-driven interprocedural analysis, and non-linear data dependence tests. Our framework for runtime optimization and speculative execution has demonstrated significant potential for improvements beyond what current compilers can achieve. It has also uncovered certain overheads, which will need combined compiler/architecture solutions.

Furthermore, in our work we have taken an active role in defining "significant computational applications" that can be used to drive and evaluate new software and hardware technology. In cooperation with the SPEC/High-Performance Group we are collecting and making available to the research community a set of industrially significant computational applications. This problem set will be an important basis for gathering the knowledge base referred to above.

In summary , the integrated design of compiler-architecture interfaces is no longer just a question of good system design; it has become a crucial performance issue. Architectures need to supply compilers with both static and dynamic information and provide fast implementations of expensive operations. Compilers need to use such facilities to carefully select appropriate optimizations and adapt them dynamically to the runtime context. We have to make an effort to understand important computational applications and how they can take advantage of advanced features offered by compilers and architectures. We have begun to tackle these issues in ongoing projects. However, significantly more work is necessary in order to ensure that the next generation of machines will become a successful one.