

Research Statement

Timothy J. Harvey

Background

Compilers offer a continuing variety of both engineering and theoretic challenges that shows no sign of abating, because the markets that drive compilation technology are themselves constantly changing.

Continually shifting architectures are the most obvious example of this phenomenon. Architectural history shows us moving from single processor, memory-to-memory CISC processors to deep memory hierarchies with multiple functional units and an almost unlimited array of restrictions and peculiarities. To utilize these architectures efficiently and keep software speed in line with the increases in hardware clock rates has taken and will continue to take new compiler research and development.

Along with architectural changes has come application evolution – we know that different kinds of applications offer different opportunities and challenges for optimization. We have also seen the rise of applications that require just-in-time compilation, which has required a considerably different focus than that required for applications such as scientific programming.

The evolution of compiler technology itself serves as a feedback loop to drive further research. An obvious example of this is the development of static single-assignment form in the early nineties, which opened up new avenues of optimization and analysis as well as fueling research on the restructuring of already known optimizations.

Opportunities

While the opportunities for research are numerous, here is a short list of those that I am interested in pursuing now:

- Register allocation – because of the critical importance of efficiently utilizing the memory hierarchy, the task of the register allocator continues to be a pressing issue, and every architecture is rife with idiosyncratic restrictions, such as registers subsets on Sparc and various DSP architectures, or variable numbers of registers on an architecture like the Itanium, etc. Add to this the fact that even with a simple architecture, the allocator is simultaneously tackling multiple NP-complete problems, and I believe that there is still much work to be done in this area.
- Adaptive compilation – our engineering approach to the problem of understanding adaptive compilation is in contrast to the more theoretical approach of other researchers, and as each evolves, we will need to understand the way the different methods can be leveraged off of each other. Certainly, we are still on the leading edge of the curve of research in this area.

- Legacy code disassembly/reassembly – the problem of porting compiled code to updated architectures is an important one, both for applications like the developing grid computing environment, and for legacy codes which may no longer be compilable (for example, if the source code has been lost). This work will require an entirely new infrastructure to rebuild some kind of coherent data-flow analysis and transform the code as needed.
- A holistic view of data-flow analysis – the literature on data-flow analysis includes many different methods, such as interval analysis and iterative analysis. Each of these has strengths and weaknesses, but within a fairly broad subset of possible inputs, they produce the same results. The implication is that they really aren't all that different from each other, and it may be possible to derive a single theoretic explanation of data-flow analysis from the many flavors.
- Code obfuscation – both industry and the government have a similar interest (albeit different goals) in hiding the function and design of code. While much work has been done in this area, I think that that the antagonistic nature of the obfuscators and the would-be decoders will continue to generate opportunities.
- Randomized code generation – an offshoot of my dissertation, in which we generated randomized control-flow graphs, was an intriguing notion that we could fill in the graphs with randomized code sequences that would be indistinguishable from real-world codes. This would have a commercial value, in that compiler companies could use it to generate unlimited amounts of code to stress-test their own compilers, but it would also be an interesting tool in the effort to obfuscate code.

Research

The above sketches of ideas have both engineering and theoretical requirements. They can, in many cases, be broken into smaller pieces and attacked from multiple approaches. Because of this, they can be adapted to either graduate or undergraduate researchers. In the twelve years that I have worked for Dr. Cooper, we have never had too few students for the work we want to do – I always have a pet project that I'd like an undergraduate to look at, and the most enjoyable part of graduate work is taking on a new idea or looking at an old idea in a new, unspoiled way.