Compiler Support for GPUs:
Challenges, Obstacles, & Opportunities

or

Why doesn’t GCC generate good code for my GPU?

Keith D. Cooper
Department of Computer Science
Rice University
Houston, Texas

History

• First real compiler —Fortran I for the IBM 704 in 1957
  • Noted for generating code that was near to hand-coded quality
• Literature begins (in earnest) around 1959, 1960
• 45 years of research & development
• Peak of compiler effectiveness might have been 1980
  • Any compiler achieved 85% of peak on the VAX machines
  • Uniprocessor ||ism and growing memory latencies have made the task harder in the succeeding years
• Today, users of advanced processors see 5 to 15% of peak on real applications & 70% or more on benchmarks

When will compilers generate good code for GPUs?

Discussion assumes that we want good code
Roadmap

Challenges
• Architecture
• General purpose code
• Rate of change

Obstacles
• Compiler design & implementation
• GPU-specific issues
• Rate of change

Opportunities
• Potential sources for software (compiler) development
• Other strategies for success

Challenges: Architecture

What do GPUs look like?
• Specialized instructions
• Multiple pipelined functional units
• Ability to link multiple GPUs together for greater width
Challenges: Architecture

What do GPU’s look like?

- Specialized instructions
  - Possible problems with completeness
    - Double-precision floating-point numbers
    - Support for structured & random memory access
  - Clever schemes to use existing specialized operations
- Multiple pipelined functional units
  - Need exposed ILP to handle multiple units
  - Need vector parallelism for pipelines
- Ability to link multiple GPUs together for greater width
  - Automating multi-processor ||ism has been hard

Challenges: General Purpose Code

What is the goal?

- Microsoft Office?
- Mail filtering, web filtering, or web servers?
- Scientific codes?

⇒ Each of these is a different market & a distinct challenge

Compiler’s goal is to handle a broad range of program styles

- Great compilers tend to be more narrow than good compilers
- Bleeding edge compilers are often quite narrow
  - Cray Fortran compiler, HPF compiler, ...
  - Application outside window gets poor performance
Challenges: General Purpose Code

To succeed, compiler must

• Discover & expose sufficient instruction-level parallelism
• Find loop-style parallelism for vector/pipeline units & larger granularity parallelism for multi-GPU situations

Arrays, pointers, & objects all present obstacles to analysis

Alternative: Use a language designed for GPUs
+ Special features that map nicely onto GPU features
- General purpose applications are written in old languages
⇒ We will return to this idea later in the talk

Challenges: Rate of Change

New GPUs are introduced rapidly

• Marketplace expects new models every 6 to 12 months
• Pace of innovation is a benefit to industry & users
• User is insulated from change by well-designed interfaces
  ♦ Interfaces change slowly
  ♦ Stable target for application programmers

Excellent example of software engineering

However, ...

• Compilers deal with low-level detail
• Optimization, scheduling, & allocation need significant work as target machine changes (not well parameterized)
Roadmap

Challenges
• Architecture
• General purpose code
• Rate of change

Obstacles
• Compiler design & implementation
• GPU-specific issues
• Rate of change

Opportunities
• Potential sources for software (compiler) development
• Other strategies for success

Obstacles: Compiler Design & Implementation

Compiler structure is well understood

Diagram showing the structure of a compiler with Front End, Optimizer, and Back End.
Obstacles: Compiler Design & Implementation

Compiler structure is well understood

Front End ----> Optimizer ----> Back End

Language dependent
Largely target independent

Language, program, & target dependent
(Same problem in any compilation context)
Obstacles: Compiler Design & Implementation

Compiler structure is well understood

GPUs are not easy targets for code generation
• Need optimizations that find & expose parallelism
• Need rapidly retargetable back end technology to cope with rate of change
Obstacles: Compiler Design & Implementation

Specific technologies for success with GPUs

- Optimization for vector & parallel performance
  - Need data-dependence analysis
  - Need loop-restructuring transformations (based on }
  - Not typically found in open-source compilers

How hard is this stuff?

+ Today, you can buy a book that covers this material. Five years ago, you could not.
- Not widely taught or understood

Still, there are complications
Obstacles: Compiler Design & Implementation

Data-dependence analysis requires whole-program analysis
Exposing ||ism requires whole-program transformation

- Problems & algorithms are well understood
- Whole-program analysis requires access to the whole program
  - Serious obstacle to compiling general-purpose code
  - Object-only libraries
- Whole-program transformations create recompilation effects
  - Edits to one module force reoptimization of other modules
  - Requires analysis to track changes & their effects

Implementation is much more complex than GCC

Obstacles: Compiler Design & Implementation

Specific technologies for success with GPUs

- Instruction selection to deal with idiosyncratic ISAs
  - Tools to match huge pattern libraries against low-level code
  - Efficient & effective tools to use this technology
  - BURS technology is gaining users

Bad news

- Instruction selection & register allocation are not easily retargeted — hand coded, often with ad hoc heuristics
- Lack of tools to build robust tools that use best practices

Serious R&D effort is needed
Obstacles: GPU-specific Issues

Need detailed information about the target processors

- ISA, timing information, model dependent issues
- GPU community has discouraged this kind of targeted work
  - Instead, they encourage use of well-defined interfaces

Compiler writers need easy access to the truth about targets.

Obstacles: Finding Information about Targets

Google search finds online manuals for Itanium easily.

Similar results for Pentium, Power, MIPS, Sparc, ...

A query where "I'm feeling lucky" works.
Obstacles: Finding Information about Targets

Difficult to obtain information needed to develop a compiler for NVIDIA products

Marketing presentation

Obstacles : Finding Information about Targets

White paper, not manual (not much information that is useful for code generation)

Magazine reviews
Obstacles: Finding Information about Targets

Need detailed information about the target processors

- ISA, timing information, model dependent issues
- GPU community has discouraged this kind of targeted work
  - Instead, they encourage use of well-defined interfaces

Compiler writers need easy access to the truth about targets

Such information is not readily available

- Does not fit the business & technology model
- Declaring all those details ties the vendors’ hands

Obstacles: Rate of Change

Compiler technology lags processor design by 4 to 5 years

- Cray 1, i860, IA-64, …
- Processor lifetime is often less shorter than the compiler development cycle

Two components to this lag

- Development of new techniques to address target features
- Time to retarget, retune, and debug

The new product cycle in GPUs is too short to allow for effective development of optimizing compilers using our current techniques.
Roadmap

Challenges
• Architecture
• General purpose code
• Rate of change

Obstacles
• Compiler design & implementation
• GPU-specific issues
• Rate of change

Opportunities
• Potential sources for software (compiler) development
• Other strategies for success

Opportunities

Is there profit here?
• Sure. GPUs exist in most of our machines
• Gaming devices are pushing the state of the art
  ♦ Network interfaces + high-powered processors
  ♦ Is the next “surprise” supercomputer a pile of Playstation 3s?

Example: our object-level vectorizer (VIZER)
• Read, optimized, recoded Pentium object modules
• Found || loops & recoded them for SSE2 operations
• Speedups of near 4x on true || loops

Had to reconstruct type & data structure information
Opportunities

Where might we obtain a compiler for these GPUs?

• GPU manufacturers
• Compiler vendors
• Open source movement
• Academia

Unfortunately, production quality compilers do not grow on trees
Opportunities

Where might we obtain a compiler for these GPUs?

- **GPU manufacturers** — the natural source
  - They have the information
  - They would need to gather the expertise
  - Timeline: two to four years for a quality product

- **Compiler vendors** — a little faster path
  - They have the expertise
  - They would need detailed long term product plans from vendor
  - Big question: is there a market to justify the effort?

- **Open source movement**
- **Academia**

Opportunities

Where might we obtain a compiler for these GPUs?

- **GPU manufacturers**
- **Compiler vendors**
- **Open source movement** — get a compiler for free
  - GCC’s structure would make this difficult
  - ORC might be the best vehicle, but would need a new back end
  - Open source does well capturing well-understood technology
    - Less suited to bleeding-edge algorithms and techniques
    - Relies on good understanding in contributor community
  - Slower development cycle
  - Would need published details on target platform
- **Academia**
Opportunities

Where might we obtain a compiler for these GPUs?

- GPU manufacturers
- Compiler vendors
- Open source movement
- Academia
  - Need funding
  - Would need published details on target platform
  - Questions about usability of delivered product

Opportunities

The picture is bleak. Is it hopeless? No!

Several potential approaches

- Open source using ORC compiler
  - Interest someone with money in the project (Nat'l lab?)
  - ASCI Pixellated Mountain?
- Binary-level optimizer
  - Identify code that maps well onto GPU and recode it
  - Our VIZER project for Pentium showed that it can be done
- Library approach
  - Develop a library of useful domain-specific computations
  - Follow the vendor's current programming model
Opportunities
Library approach has several potential advantages

• Same freedom to manufacturers as DirectX interface
• Let experts program specific functions for efficiency

• Provide a standard interface for commonly-used functions
  ♦ Start with something widely used, such as BLAS or BLAS-3
  ♦ Use Thinking Machines’ strategy
  ♦ Issue updates as hardware changes
• Couple well defined, well implemented library with a DSL (e.g., Matlab) for a complete solution
  ♦ Preprocessor + library creates a DSL
  ♦ Matlab accelerator might be an attractive starting point

Summary + Conclusions
• Potential payoff is high
  ♦ Linking progress to GPU improvement curve is attractive
  ♦ Might lead to market opportunities in HPC
• Challenges
  ♦ GPUs might need additional features
  ♦ Software strategy must not interfere with business model
  ♦ Must handle rate of change & protect proprietary designs
• Plan of attack
  ♦ Open interfaces, proprietary implementations for domain-specific libraries
  ♦ Couple libraries with syntax to create useful DSLs
  ♦ And, of course, a couple of compiler R&D problems
History: Backup Data on Performance

CCS-3 Group at LANL conducted a detailed study of application performance on LANL’s ASCI Blue Mountain machine:

- 6,144 MIPS R10000 processors
- Carefully separated uniprocessor issues from \( \parallel \)ism

Measured uniprocessor performance:

- PARTISN achieved 13% of peak
- SAGE achieved 4% of peak
- LINPACKD achieved 70% of peak

\( \square \) Popular benchmark

\[ \text{Real codes} \]

---

1 Hoisie, Kerbyson, Pakin, Petrini, & Wasserman, Report at ASCI PI Meeting, San Diego, CA, USA, January 2003
Opportunities: The ORC Compiler

Began as version of MIPSPro Compiler for IA-64

Commercial quality compiler that is open source
- 3 front ends, 1 back end
- Supporting community focused on Itanium
- Has tools for ILP, MP ||ism, IDFA
- Lacks retargetable back end

Extra Slides start here
Abstract

Graphic processors are high-speed, domain-specific processors. Consumer demand for graphic interfaces has produced an intensely focused design effort for these devices, resulting in increasing processing power and decreasing prices. Simple economics compel us to examine the opportunities to use GPUs in general-purpose computation.

To make GPUs useful for general-purpose computation, we need

1. mechanisms to identify those portions of a computation that are well suited to execution on the GPU;
2. tools that produce high-quality code for those program segments and link the code into an executable form; and
3. tools to debug (both performance and correctness) the resulting programs.

This talk will survey the technical challenges that arise in attacking these problems and the available solutions to them. It will discuss the infrastructure issues that make addressing these problems difficult. It will suggest ways that the community can improve the likelihood of a successful attack on these problems.

The Opportunity for Change

The structure of compilers has not changed much since 1957

1. Front End, Middle Section (Optimizer), Back End
2. Series of filter-style passes
3. Fixed order of passes
Challenges: Architecture

What do GPUs look like?

- Specialized instructions
- Multiple pipelined functional units
- Ability to link multiple GPUs together for greater width

Compiler writers need detailed architectural specifications

- Not readily available on the net
- Protecting design is a competitive issue for vendor
- Strong disincentive for open-source compiler developers