Building A Cell BE SPE Backend For LLVM

B. Scott Michel, Ph.D.
High Performance Computing Section
Computer Systems Research Department

scottm@aero.org
Who, What and Where?

Where:

The Aerospace Corporation’s Computer Systems Research Department

What:

Cluster-based computing, FreeBSD kernel hacking, Grid computing, multicore processor architectures, …, and LLVM hacking

The Aerospace LLVM Project Team:

Scott Michel, Mark Thomas and Michael AuYeung
Why Add A New Backend To LLVM?

- Two multicore families: homogeneous and heterogeneous
  - Homogeneous: More execution units, more threads, software transactional memory, “It’s manageable!”
  - Heterogeneous: Cooperation between specialized and general-purpose processors, “It’s a nightmare!”
- Heterogeneous encompasses more than just Cell BE
  - General-purpose GPU computing
  - Reconfigurable computing (FPGAs, accelerators)
- Research Theme: Multicore programmability for mere mortals…
  - Resource allocation between elements
  - Reuse CellSPU approach to other heterogeneous platforms
- Step 1: Need to generate CellSPU assembly code…

Compiler hacking for fun and profit!!
Cell’s Characteristics

• Unified vector-scalar, 128 element register file
  – Register info TD defines 16-, 32- and 64-bit integer and floating point register classes
    – 64- and 128-bit integer support coming...
    – Makes writing Instruction Info TD easier: reuse same instruction in multiple contents

• Some 8-bit instructions, but not for math and logical ops (many v16i8 special cases)

• Interesting special patterns
  – ORC: Or with complement
  – SELB, select bits: (A & C) | (B & ~C)

• Used PPC/PPC-64 the starting template
How Far Along?

• 10 instruction groups to implement
  – Completed: load/store, constant formation, integer and logical, shift/rotate
  – Mostly complete: floating point
  – Remaining: compare/branch, hint-for-branch, control and synchronization
  – Dejagnu-based testing: “It looks like we’re generating the right code” (and spu-as accepts the code too!)
  – GCC implements many of these instructions as intrinsics
• ABI, Structures and unions: “best guess” based on ABI specs
• Need gcc 4.2/4.3 to adequately generate and test real code
Challenges

• Really need gcc 4.2 or 4.3 (compiler versions with Cell SPU support)
  – Generate the intrinsic calls that Cell SDKs support
  – Backporting to 4.0 is not an option: SPU’s “md” file uses newer features for which there are no backport paths

• Error messages from asserts are close to meaningless to llvm newcomers (been working on that incrementally)

• Instruction scheduling for SPE
  – Dual issue instruction queue, even-odd pipes: loads-stores must be on odd pipe, different functional units prefer specific pipe
  – Doesn’t fit well with current LLVM instruction scheduling pass… needs “whole function” scheduling

• Probably issues in current register allocation passes, but don’t know yet…
Research Roadmap

• Heterogeneous multicore programming is not for mere mortals
• Resource allocation is the underlying problem
  – It’s simple: Just identify the code that can be run on the CellSPU…
  – Steps 1 and 1a: Vectorizing and data orchestration
  – Step 2: message orchestration
    – Moving work units into and out of SPU’s local store memory
    – Identify additional dependent data, i.e., control state
• Connecting the components
  – GPGPU (CTM, CUDA), FPGA (VHDL) etc.,
  – More generally, communicating processes, e.g., OpenMPI
• How much support from the language is really needed?
  – Google is working on advise module… which could really help!
FAQ: When Can I Play With The Cell SPU Backend?

• Hopefully, we will have something to release by end of August
  – Code has to pass through Aerospace’s software review and release committee before we commit to CVS/SVN

• Michael AuYeung is migrating patches between llvm’s gcc 4.0 source and gcc 4.2
  – Almost finished last week with files beginning with “c”
Questions?