Work Scheduling on Heterogeneous Systems

Gabriele Keller (supervisor) Edward Pierzchalski

University of New South Wales

e.pierzchalski@unsw.edu.au

February 19, 2015

Overview

Why Heterogeneous Programming

- People do math, simulations, and stream processing
  - We can parallelise these!
- CPUs are becoming tiny graphics cards
- GPUs are becoming tiny CPU swarms

Why Not Heterogeneous Programming

- Distributing work is fiddly (thesis worthy!)
- Heterogeneous code is difficult, ugly, and imperative
- Hard to compose solutions
  - If I can write an optimised GPU kernel for maps, and another for prefix sums, can I 'fuse' them together while maintaining performance?

Accelerate: An Array DSL

- A deeply embedded domain-specific language (the domain is array computation)
- Embedded in Haskell
  - Lots of fancy type shenanigans to enforce semantics
- Solves the 'difficult, ugly, and imperative' problem
- Doesn't solve the 'distributing work' problem
  - Until recently!

Data Fission (by Newton, Holk, and McDonell)

- Added two nodes to Accelerate: `split` and `concat`
- Arrays are fissioned after operator fusion optimisations
- Initial algorithm: fission arrays into constant number of fragments, allocate each fragment to a device

Can we do better?

Kinds of Scheduling

- Static Scheduling:
  - Arrays are fissioned into constant number of work fragments
  - Fragments are scheduled on devices according to data dependency
- Dynamic Scheduling:
  - Use runtime information to choose fragmentation and device scheduling

What Has Been Tried

- Machine learning on source code (Grewe and O'Boyle)
- Binary analysis of array indexing (Lee et. al.)
- Statically split and allocate (Newton, Holk, and McDonell)
- Runtime performance analysis (Wang et. al.)

Speedy Dynamic Scheduling

- Devices perform differently under different workloads
- Use run-time profiling
- Do too little and you can't exploit device/workload differences
- Do too much and you get swamped by synchronisation overheads (both time and memory)
- Try something in-between!

Main Goals

- Implement dynamic fissioning, dynamic allocation
  - New `accelerate-backend-kit` already set up groundwork
  - Compare scheduling algorithms
  - Investigate interaction of fission and fusion
- Small-array optimisations
- Benchmarking

Extensions

- Improve array concatenation
- LLVM support for operations
- Device affinity for dynamic allocation algorithm

Question Time!