Hera-JVM: Abstracting Processor Heterogeneity Behind a Virtual Machine

Ross McIlroy and Joe Sventek
University of Glasgow
Department of Computing Science
Heterogeneous Multi-Core Architectures

- CPUs are becoming increasingly Multi-Core
- Should these cores all be identical?
  - Specialise cores for particular workloads
  - Large core for sequential code, many small cores for parallel code
- Found in specialist niches currently
  - e.g. network processors (Intel IXP), games consoles (Cell)
- Likely to become more common
  - On-chip GPUs (AMD Fusion), Intel Larrabee
Developing for HMAs

Application Threads
Developing for HMAs

Main Arch Code
Secondary Arch Code

Application Threads
Developing for HMAs

Main Core

Main Arch Code

Secondary Arch Code

Secondary Cores
Developing for HMAs

Main Arch Code  Secondary Arch Code  Support Code

Main Core

Secondary Cores
Developing for HMAs

Main Core

Main Arch Code
Secondary Arch Code
Support Code

Secondary Cores
Developing for HMAs

Main Arch Code
Secondary Arch Code
Support Code

Main Core
Secondary Cores
Developing for HMAs

Main Core
- Main Arch Code
- Secondary Arch Code
- Support Code
- Libraries

Secondary Cores
- main.o
- secondary.o

Main Core
- Libraries

Secondary Cores
- Libraries
Hera-JVM

• Hide this heterogeneity from the application developer
  - Present the illusion of a homogeneous multi-threaded virtual machine
  - The same code will run on either core type

• Runtime system is aware of heterogeneous resources
  - Can transparently migrate threads between core types based upon this knowledge

• Provide portable application behaviour hints to enable runtime system to infer the application’s heterogeneity
  - Explicit Code Annotations
  - Static Code Analysis / Typing information
  - Runtime Monitoring / Profiling
Developing for Hera-JVM

- **Main Core**
  - Integer
  - Random Memory Access
  - Float

- **Secondary Cores**
  - Branching Code
  - Sequential Memory Access

**Application Threads**
Developing for Hera-JVM

Application Threads
- Integer
- Random Memory Access
- Float
- Branching Code
- Sequential Memory Access

Runtime System
- Main Core Costs
  - Int, Float
  - Rand
- Sec. Core Costs
  - Int, Float, Seq
  - Rand

Main Core  Secondary Cores
Developing for Hera-JVM

Application Threads
- Integer
- Random Memory Access
- Float
- Branching Code
- Sequential Memory Access

Runtime System
- Main Core Costs
- Rand
- Int, Float
- Sec. Core Costs
- Int, Float, Seq
- Rand

Main Core
Secondary Cores
Developing for Hera-JVM

Main Core

Integer
Random Memory Access
Float
Branching Code
Sequential Memory Access

Application Threads

Runtime System

Main Core Costs
Rand
Sec. Core Costs
Int, Float, Seq
Int, Float
Rand

Main Core
Secondary Cores
Developing for Hera-JVM
Developing for Hera-JVM

Application Threads

Integer
Random Memory Access
Float
Branching Code
Sequential Memory Access

Runtime System

Main Core Costs
Rand Int, Float
Sec. Core Costs
Int, Float, Seq
Rand

Main Core
Secondary Cores
Developing for Hera-JVM

Main Core

Secondary Cores

Integer

Random Memory Access

Float

Branching Code

Sequential Memory Access

Application Threads

Runtime System

Main Core Costs

Rand

Sec. Core Costs

Int, Float, Seq

Int, Float

Rand

Main Core

Secondary Cores
Developing for Hera-JVM

Main Core

Secondary Cores

Integer

Random Memory Access

Float

Branching Code

Sequential Memory Access

Application Threads

Runtime System

Main Core Costs

Rand

Sec. Core Costs

Int, Float, Seq

Int, Float

Rand

Main Core

Secondary Cores
Developing for Hera-JVM

Main Core

Integer

Random Memory Access

Float

Branching Code

Sequential Memory Access

Application Threads

Runtime System

Int, Float, Seq

Main Core Costs

Rand

Int, Float

Sec. Core Costs

Rand

Secondary Cores

Int, Float, Seq

Rand

Main Core Costs

Int, Float
Cell Processor
A JVM for Two Architectures

- Built upon JikesRVM
  - Java in Java
  - PowerPC and x86 support
A JVM for Two Architectures

• Built upon JikesRVM
  - Java in Java
  - PowerPC and x86 support
A JVM for Two Architectures

- Built upon JikesRVM
  - Java in Java
  - PowerPC and x86 support
A JVM for Two Architectures

- Built upon JikesRVM
  - Java in Java
  - PowerPC and x86 support
A JVM for Two Architectures

- Built upon JikesRVM
  - Java in Java
  - PowerPC and x86 support
A JVM for Two Architectures

- Built upon JikesRVM
  - Java in Java
  - PowerPC and x86 support
A JVM for Two Architectures

• Built upon JikesRVM
  - Java in Java
  - PowerPC and x86 support
A JVM for Two Architectures

- Built upon JikesRVM
  - Java in Java
  - PowerPC and x86 support
A JVM for Two Architectures

- Built upon JikesRVM
  - Java in Java
  - PowerPC and x86 support
Migration

- A thread can migrate between the PPE and SPE cores at any method invocation
  - Migration is triggered either by an explicit annotation or is signalled dynamically by the scheduler
  - Syscalls and native methods always migrate back to PPE

- **Migration from core type A to B:**
  - Thread “traps” to support code on core A, which saves arguments
  - Method JITed for core type B if required
  - Migration marker and migration support frame pushed onto stack
  - Thread placed on ready queue of core type B
SPE Local Memory

- Instead of a cache, SPEs have 256KB of explicitly accessible local memory.
- Main memory accessed through DMA using MFC (Memory Flow Controller).
- Setting up many small DMA transfers is costly.
Software Caching in a High Level Language

- Java bytecodes are typed, therefore, we have high level knowledge of what’s being cached
  - Cache an object completely when it is accessed
  - Cache arrays in 1KB blocks
- Java memory model only requires coherency operations at synchronisation points
- Methods are cached in their entirety when invoked
Hera-JVM Performance

Single Threaded
Hera-JVM Performance

Single Threaded

SPE v.s. PPE Speedup

- sm.sor
- Mandelbrot
- sm.sparse
- sm.lu
- sm.montecarlo
- MolDyn
- sm.fft
- MpegAudio
- Ray Tracer
- MonteCarlo
- Compress
Hera-JVM Performance

Multi-Threaded
(6 threads)

6 SPEs v.s. PPE Speedup

- sm.sor
- Mandelbrot
- sm.sparse
- sm.lu
- sm.montecarlo
- MoDy
- sm.fft
- MpegAudio
- Ray Tracer
- MonteCarlo
- Compress
Proportion of Execution Time by Operation

compress

mpegaudio

mandelbrot

0% 20% 40% 60% 80% 100%

Floating Point
Integer
Branch
Stack
Local Memory
Main Memory
Data Cache Hit-Rate

- compress
- mpegaudio
- mandelbrot

Graph showing the relationship between Data Cache Size (KB) and Data Hit Rate, with Performance (relative to 96KB default) as the y-axis.
Code Cache Hit-Rate

- compress
- mpegaudio
- mandelbrot

Performance (relative to 84KB default)

Method Hit Rate

Code Cache Size (KB)

88 80 72 64 56 48 40 32 24 16 8 0
Conclusion / Future Work

• Architectures are likely to become more heterogeneous

• This heterogeneity should be taken out of the hands of non-specialist programmers

• Instead, hide this heterogeneity from the programmer and provide abstractions to infer a program’s heterogeneity
  - E.g. code annotations, runtime monitoring, etc.

• Hera-JVM is a proof of concept of this approach
  - Overheads involved in hiding the heterogeneity are tolerable for most applications

• Next Stage: Fully integrate behaviour tagging with scheduling / migration decisions