Virtual Machines and Chip Multiprocessors

The JAMAICA project
Ian Rogers
The Advanced Processor Technologies group
http://www.cs.man.ac.uk/apt
Presentation outline:

• The JAMAICA Chip Multiprocessor architecture:
  – overview
  – a core and more cores
  – work distribution
  – future directions

• Software support:
  – static compiler tools
  – the Jikes RVM
  – loop optimisations
  – PearColator
  – future directions

• Summary
Overview of the JAMAICA architecture

- Java Machine and Integrated Circuit Architecture (JAMAICA)
- Design goals:
  - how to use a billion transistor budget?
  - targeted for Java programs
  - high-level initial simulation, for flexibility and speed
  - extends the work on the VAULT architecture (uniprocessor multiple contexts)
- Features:
  - 2 tier architecture of CPU nodes and groups of nodes
  - split transaction cache coherent bus protocol
  - thread scheduling and work distribution support unit
  - heap allocated register windows
  - context switch on cache miss
  - simulated in C
Combining the strengths of UMIST and The Victoria University of Manchester

A core

Alpha based instruction set

Token ring interface

5 stage MIPS based pipeline

Heap allocated registers and context management
Combining the strengths of UMIST and The Victoria University of Manchester

Grouping cores

Simulation prototype interfaces with memory here
Work distribution

- Idle threads distribute tokens on a separate token ring bus
- Executing context on a core requests to ship work to an idle context or core and context
- Taking a token from ring grants the use of a particular context
- Shipping of work between cores occurs over data bus
- Gives lightweight thread creation
- When token is redistributed, work has been completed
- Thread unit monitors for completion of forked work
Future directions

- Remove register windows
  - doesn’t adhere to our initial KISS philosophy
- Floating point support
- Groups of groups of cores
  - effect of tiers and bus configuration on the architecture
- Parallel and distributable simulator
- Consider improving instruction level parallelism (ILP)
  - VLIW is appealing
    - trade off between ILP and thread level parallelism (TLP) on chip
- Different work distribution interfaces
- Support for speculative execution (slide 8)
- Virtual memory system (slide 9)
Future directions: speculative execution

- **Aim**: increase number of parallel threads
- **Range of speculative and non-speculative execution states**
  - tree rooted at non-speculative state with branches for every spawned speculative context
  - speculative contexts may spawn more speculative contexts
- **If speculation goes wrong squash speculative state**
  - throw away values in cache or a buffer
- **Detect speculation problems**:  
  - in software: when a value isn’t that expected explicitly squash
  - in hardware: when an address is loaded by a speculative context, ensure that stores to the same address from a less speculative context cause a squash
- **Problems with creating speculative threads and avoiding excessive squashing**
- **Mechanism may aid virtual machines, e.g. handling of unaligned memory accesses**
Future directions: virtual memory system

- **Aims:**
  - allow the emulation of multiple architectures or execution of virtual machines on one underlying architecture
  - keep hardware simple

- **Prototype design:**
  - extend virtual address bus to contain a virtual machine identifier
    - value 0 reserved to access physical memory
  - loads, stores and instruction fetches are tagged with an identifier value either from the instruction itself or from a special purpose register
  - the MMU is simply a translation buffer that if a miss occurs raises a software exception (it also provides read/write/modified information)
  - software controls the contents of the translation buffer
    - emulates segments and paging for legacy architectures
    - provides full virtual address space for virtual machines
Software support for the JAMAICA architecture

- Static tools
  - C compiler – based on Princeton’s LCC
  - jtrans – Java class file to assembler
  - javar – modified to generate jtrans parallel constructs
  - sim-idbg – interactive debugger
The Jikes RVM

- JVM written in Java
- Support for IA32, PowerPC and Jamaica
- Baseline (quick) and optimizing compilers
- Adaptive optimization and feedback system
- Extended array SSA form sub-stages in HIR and LIR optimization

Jikes RVM optimizing compiler

Front End:
- Bytecode to HIR
- Optimization of HIR
- Optimized HIR

Back End:
- HIR to LIR
- Optimization of LIR
- Optimized LIR
- LIR to MIR
- Optimization of MIR
- Optimized MIR
- Final Assembly
- Binary Code

Profile information
BURES grammar
Machine description and parameters
Hardware parameters

HIR = High-level Intermediate Representation
LIR = Low-level Intermediate Representation
MIR = Machine-specific Intermediate Representation
BURES = Bottom-Up Rewrite System
Loop optimisations

- Jikes RVM is good at removing dependencies:
  - extended array SSA form models scalar and heap dependencies, SSA means all dependencies are true dependencies
  - pi nodes used to rename a variable following a comparison, to remove redundant compares
- Wrote simple loop parallelisation optimisation that, to work, ignored dependencies (java runtime style)
- Observation that loop parallelisation was being limited by bound and null check instructions that defined and used exception states
- Writing new loop optimisation framework:
  - annotated LST nodes describe the loop structure to the optimisations
  - moving bound and null checks out of loops using explicit tests
    - so far has yielded 3% speed up on certain SpecJVM benchmarks
  - migrating parallelisation code to this framework
  - future possibilities including cache and speculative compiler optimisations
PearColator

- PowerPC Dynamic Binary Translator (DBT) written (in Java) to use the Jikes RVM optimising compiler
  - Trace based
    - trace length depends on optimisation level
    - traces start and stop at procedure calls and returns
  - HIR code can be converted to Java bytecode
  - Lazy flag optimisations
  - Adaptive/parallel recompilation
  - Virtual memory system accurately emulated
    - Translator not visible in translated code’s memory space
    - Slow page then value look up
      - Loop optimise page look ups in hot loops
      - Hardware optimisation for Jamaica architecture
Future directions

- Java OS – preliminary work created a version of the Jikes RVM which will boot the JNode class library and device drivers
- AI algorithms to drive optimisation framework
  - Current optimisation phases have complex inter-relationships that behave different depending on the code they’re optimising
- Integration of compiler and hardware systems:
  - Virtual memory architecture and PearColator
  - Loop optimisation and speculation support
Summary

- Silicon density has already brought about CMP systems
- The JAMAICA project is refining prototype CMP systems that rely on software virtual machine support
- Simpler hardware, due to software support, increases flexibility and the amount of available parallel resources
- Knowledge of the hardware only known by the virtual machine
  - portable infrastructure
  - virtualised CPU system
- Hardware and software assist to extract more, possibly speculative, parallelism and to improve scheduling