Compiler needs to drive the ISA

Arrvindh Shriraman

Simon Fraser University, Jan 1

with input from
Sandhya Dwarkadas, Michael Scott, Engin Ipek, and Chen Ding

UNIVERSITY of ROCHESTER

Special thanks to feedback from NSF ACAR II Panel, “What next in Instruction-Level-Parallelism research?”
Brief Introduction

- I am a computer architect
  - you guys probably know more about compilers!

- While many collaborators, opinions and mistakes are my own

- Non Goal : What should the architecture stack should look like in the future ?

- Goal : get compiler designers to actively participate in defining HW-SW interface
Outline

- Inefficiency of current microprocessors
  - Energy wall
  - RISC ISA
  - Where does energy go?

- Big and Morphable ISA

- Heterogeneous execution substrate
## Energy Wall

<table>
<thead>
<tr>
<th>Year</th>
<th>Total Si</th>
<th>Power /Si</th>
<th>Usable Si</th>
</tr>
</thead>
<tbody>
<tr>
<td>2008</td>
<td>1</td>
<td>1</td>
<td>100%</td>
</tr>
<tr>
<td>2014</td>
<td>4</td>
<td>1</td>
<td>25%</td>
</tr>
<tr>
<td>2020</td>
<td>16</td>
<td>0.6</td>
<td>4%</td>
</tr>
</tbody>
</table>

- 45nm for 2008
- 22nm for 2014
- 11nm for 2020
RISC Instructions

- Simple by design
  - focus on pipelineability and latency
- Instructions encode little work
  - no parallelism within instructions
  - Instruction-parallelism reaching its limits

Classic study: Tejas and Smith
RISC Instructions (contd..)

- OO HW
  - needs to scale resources quadratically
  - needs to reduce misses and branch penalty quadratically

- Pipelining small instructions is inefficient
  - fetch consumes 4x more energy than execute

Energy allocation:
- Fetch: 38%
- Pipeline: 31%
- Exe: 6%
- D$: 15%
- Reg.: 10%
Current ISAs - Do we need them?

- Tension between hardware and software

- Software allowed to provide only limited info on data types and operation semantics

- Not necessarily useful for hardware either
  - energy is the #1 constraint, not area or clock
  - HW needs to work hard to mine parallelism
  - does not support energy-efficient functional units
Agenda

Need to perform more work per instruction

ISA needs app-specific information

Hardware needs to cut-down overheads
Outline

- Inefficiency of current microprocessors

- Dynamic Morphable ISA
  - Decoupled Multi-level ISA
  - Big Instructions
  - Application-based ISA extensions
  - Challenges

- Heterogeneous execution substrate
De-coupled ISAs

- Two-levels of ISA abstraction
  - App ISA for customization, more info, and portability
  - HW ISA for platform-specific opt. and accelerators

- Always-present virtual machine
  - can use rich SW program information
  - can use feedback from HW for dynamic optimization
- **Big Instructions**
  - compound instructions that encode more work

- **Can capture relationship between operations**
  - provides information about operation parallelism
  - enables hardware to exploit parallelism

**SSE, AVX**

D1  |  D2  |  D3  |  D4

**VLIW, EPIC**

D1  |  D2  |  D3  |  D4

**Austin TRIPS**

D1  -> D2  -> D3
Application ISA

- **Custom instructions**
  - library runtimes, C++ Boost, MATLAB, R, Sphinx
  - generate application-specific instructions

- **Static instructions**
  - many applications have a small hot code-regions
  - 70% code shared between applications

- **Dynamic instructions**
  - enable software to add new instructions to ISA
  - 90% coverage with 40,000 static instruction*

* GreenDroid - HotChips 2010
Research Agenda (1/2)

- What is the basic application ISA
  - X86, LLVM, RISC, Java bytecode, Nvidia PTX?

- How to specify and modify application ISA?
  - pragmas, macro language?
  - generate from hot path?

- How to enable software to make use of new ISA?
  - compiler generators? Is that so crazy or feasible?
  - how to alter the backend of a compiler at runtime?
Research Agenda (2/2)

- Reconsider Instruction semantics
  - do we need precise exceptions at the instruction level?
  - atomicity of which operation guaranteed?
  - major impact on hardware complexity
Example Designs

- **IBM System 38**
  - Application ISA : TMI
  - Implementation ISA : IBM AS-400 descendants

- **Transmeta CMS (ahead of its time ?)**
  - Application ISA : x86 for compatibility
  - Implementation ISA: VLIW for energy efficiency

- **Nvidia CUDA**
  - Application ISA : PTX
  - Implementation ISA : Nvidia GPUs
Outline

- Inefficiency of current microprocessors
- Dynamic Morphable ISA
- Heterogeneous execution substrates
  - Architecture
  - Program state management
  - Examples
Accelerator-based architecture

- Dedicated specialized hardware to improve performance under same energy

- Granularity
  - 80s, floating point unit, co-processors
  - 90s-00s: Vector units.
  - Future: GPU, DSP, and more...
## New Challenge: Program State

### Multicore CPU (e.g., Power6)
- Hardware caching
- Fine-grained sharing
- Minimal programmer effort
- SW cannot directly optimize

### Accelerators (e.g., GPU)
- Scratchpad memories
- Coarse-grain sharing
- SW can optimize
- Weak memory models

### Latency-optimized
- small reg files (<128 bytes)
- large caches
  - per L1 = 64KB
  - L2 = 4MB
  - 90% of on-chip mem.

### Throughput-optimized
- large reg files (2MB)
- small cache
  - all L1s = L2 = 768KB
  - 70% of on-chip mem.

90% of on-chip mem. hardware controlled

70% of on-chip mem. software controlled
Program state challenges

- Where to stage data required by big instructions?
  - registers, on-chip cache, memory
  - what about bandwidth?

- SW runtime to manage on-chip caches?
  - similar to GC, except for physical storage,
  - on-chip memory shared by many tasks
  - need to organize private and shared-data

- Managing program state in SW too hard?
  - can we afford not to, HW too reactive and unoptimal
New Challenges: Task Mapping

**Task**: Bag of instructions

- How to choose which tasks to run on accelerator?
- How to choose accelerator?

Adaptive Task Mapping [Micro 2010]
Example systems

- **Focus**: design time chip generator for embedded applications are reasonably stable
  - profile application and create HW for hot-code

- **Industry**: Tensilica Systems
  - new language to specify extension units and instructions
  - automate integration of special-purpose HW w/ CPU
  - generates compiler backend for new instructions

- **Academia**: Conservation Cores [UCSD]
  - generated HW from profiled hot-code in program
  - cold-code on CPU; hot-code on specialized HW
Summary

- Decouple ISA
  - Applications
    - HLLs can communicate more information
    - enable more dynamic optimizations
    - allow for dynamic extensions for app-customization
  - Software code translator
  - Hardware ISA
  - Hardware

- Application ISA
  - Heterogeneous accelerator cores
    - new compute units driven by software libraries
    - SW needs to manage on-chip program-state