CSCI-UA.0480-010
Special Topics: Multicore Programming

Lecture 11
Heterogeneous Multicore

Christopher Mitchell, Ph.D.
cmitchell@cs.nyu.edu || http://z80.me
CPU Evolution

- Evolving single-core CPUs
  - Special purpose units
  - FPU, SSEs, etc

- Hybrid multi-core CPUs
  - Cell: 1 PowerPC core, 8 SPEs
  - AMD Fusion: CPU + GPU
  - Intel Sandy Bridge and later
## Classifying Multicore CPUs

<table>
<thead>
<tr>
<th></th>
<th>Same Cores</th>
<th>Different Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Different ISAs</strong></td>
<td>?</td>
<td>• Some SoCs&lt;br&gt;• Multicore/many core CPUs, GPUs</td>
</tr>
<tr>
<td><strong>Same ISA</strong></td>
<td>Traditional homogeneous multicore CPUs</td>
<td>Cores with different capabilities and functional units</td>
</tr>
</tbody>
</table>
Designing Heterogeneous Multicore CPUs

1. Design series of cores from scratch
2. Reuse series of previously-implemented CPU cores and change the interface
3. Hybrid approach
Why Make Heterogeneous Multicore CPUs?

- CPU design should balance throughput (multicore performance) with good single-threaded performance
  - SMT and CMP: have they achieved this?
- Heterogeneous multicore is more efficient
  - Match application to core(s) with capabilities
  - Provide better area-efficient coverage of workload demand
Matching Applications and Cores

• Different applications have different resource requirements
  • How to determine these requirements
  • Time-variant needs

• Gains from matching threads/processes to cores
  • Better performance
  • More efficient execution: lower power consumption
Matching Applications and Cores

- Applications with fewer, sophisticated threads
  - Traditional multicore
  - Latency-optimized cores
- Applications with more, simpler threads
  - Large number of throughput optimized cores
  - E.g.: GPUs
Two Challenges

Remember this from Lecture 3?

**Power**

**Performance**

- Efficiency
- Locality

Data movement costs more than computation.
Issues in Heterogeneous Multicore Processors

1. Scalability
2. Scheduling
3. Task switching cost
4. Task migration cost
5. Core-switching decisions
6. Memory model (shared memory?)
7. Coherence
8. ...

Real-World Heterogeneous Multicore CPUS
Real-World Heterogeneous Multicore CPUs

- Cell Processor: Playstation 3
- Espresso: Wii U
- AMD Fusion, later AMD APU
- Intel Sandy Bridge (and beyond)
- Core Fusion
Cell Processor: Playstation 3

- Each Cell chip:
  - One PowerPC core (PPE)
  - 8 Synergistic Processing Elements (SPEs)
  - On-chip memory controller, I/O, and interconnect

- Playstation 3
  - 1 Cell chip, 6 usable SPEs
  - 256MB DRAM
Cell Processor:
PowerPC Element (PPE)

- 3.2 GHz
- Dual issue, in-order
- 2-way multithreaded
- 512KB L2 cache
- No hardware prefetching
- SIMD + FMA
  - Fused multiply-add: good for graphics + physics
- Performance
  - Float: 25.6 GFLOPS
  - Double: 6.4 GFLOPS

- Purpose
  - Compatibility processor
    - Legacy code, libraries, etc
  - System functions
    - Initializing and managing SPEs
Cell Processor: Synergistic Processing Element (SPE)

- Dual-issue, in-order, VLIW-inspired SIMB processor
- 128x 128b registers
- 256KB local store (LS)
  - No explicit data or instruction caches
  - Runs programs directly from the local store
- Memory Flow Controller (MFC)
  - Programmable DMA engine
  - On-chip network interface

- Performance
  - Float: 25.6 GFLOPS
  - Double: 1.83 GFLOPS
  - LS: 51.2 GB/s
  - Interconnect: 25.6 GB/s
Cell Processor: Threading Model

• PPE runs programs on SPEs
  1. Create SPE context
  2. Load SPE program binary to SPE local store
  3. Create pthread to run it

• Simplified pthread contents
  spe_context_run(...);
  pthread_exit(...);

• Work split into repeated phases (BSP):
  • Code that runs on SPEs
  • Code that runs on PPE
  • Barriers
Cell Processor: Threading Model

- PPE
  - Create pthreads
    - SPE
      - SPE program
    - Barrier
    - SPE phase
      - SPE phase
    - Barrier
    - PPE phase
      - Barrier
    - Join pthreads
Cell Processor: Communication

- PPE communicates with SPEs via
  - Mailboxes (FIFO)
  - HW signals
  - DRAM
  - Direct access (DMA) to SPEs’ local stores
Espresso: Wii U

- Three 45nm PowerPC cores
- 1.24GHz
- Symmetric multiprocessing with MESI support
- 32-bit integer unit, 64-bit FPU
- Asymmetric L2 caches
- 6 execution units per core
AMD “Trinity” APU

• Two dual-core x86 units (4 cores total)
• 2MB L2 cache total (1MB local per dual-core unit)
• Radeon HD GPU, with speed scaling for power management (304-800MHz)
AMD “Trinity” APU

“TRINITY” APU

- DDR3 Controller
- UNB
- GMC
- L2 Cache
- Dual Core x86 Module
- Dual Core x86 Module
- GPU
- L2 Cache
- AMD HD Media Accelerator
- Dual Channel DDR3 Memory Controller
- Unified Northbridge
- Northern Islands AMD Radeon™ GPU
- Up to 4 “Piledriver” Cores with 2MB L2
- HDMI, DisplayPort 1.2, DVI controllers
- PCI Express I/O — 24 lanes, optional digital display interfaces
Intel: Sandy Bridge and Beyond

- First processor with on-board GPU
  - Predecessor had separate dies in same ceramic package
  - Advantage: lower memory latency
  - Blurry line between heterogeneous multicores and closely-integrated separate chips

- Shared L3 cache (even accessible to GPU)

- Multimedia applications
  - Advanced Vector Extensions (AVX)

- Power efficiency
Intel: Sandy Bridge and Beyond
Core Fusion: Research CPU

• Common pattern: applications have sequential and parallel phases
  • Modern CPUs do not handle this
  • Burden on programmer

• Die composition is immutable
  • Number of cores
  • Number of functional units per core
  • Amount of cache per core
Core Fusion: Research CPU

- Independent cores with independent L1 caches
- Cores can be merged together as needed
- Application control
  - FUSE and SPLIT operations
  - OS driver controls application’s ability to control core layout, as with other configurable hardware

Programming Heterogeneous Multicore CPUs
Programmers’ Burden

• Programmers reason about algorithms, correctness, and thread partitioning

• Programmers don’t reason about asymmetry
  • Asymmetry negatively affects applications
  • Unexpected or even unpredictable workload behavior

• Fixing asymmetry
  • Evaluate threads’ work partitioning
  • Automatic scheduling of asymmetric threads
Programmers’ Burden

• Assist the hardware
• Define as many threads as you can
• Know the OS and the hardware
  • How many cores? How heterogeneous?
  • How does OS scheduling work?
• Combine threads if necessary
  • Bind threads to cores if necessary
• Avoid load imbalance -> wasted waiting
  • Even with homogeneous multicore processors!
  • Lab 2-3 designs
Load Imbalance Sources

- Poor single-processor performance
  - Memory inefficiencies -> optimize for caching
- Too much parallelism overhead
  - Thread lifetime is too short
- Different amounts of per-core work
- Different resources per processor
Recognizing Load Imbalance

- Easiest: time spent at synchronization is high and uneven across cores
- Imbalance may change over time
- Insufficient parallelism often leads to load imbalance

Useful tools
- Tuning and Analysis Utility (Tau): http://www.cs.uoregon.edu/research/tau/home.php
- Parallel Application Programming Interface (PAPI): http://icl.cs.utk.edu/papi/
- Paradyn: http://pages.cs.wisc.edu/~paradyn/
Important Load Balancing Questions

• **Thread Costs**
  • Do all threads have equal costs?
  • If not, when are the costs known?

• **Thread Dependencies**
  • Can all threads be run in any order?
  • If not, when are the dependencies known?

• **Locality**
  • Does locality affect any threads’ performance?
  • If so, when is information about communication known?
Important Load Balancing Questions: Thread Cost

- Thread cost known before starting
  - Matrix multiplication
- Thread cost known when thread created
  - N-body
- Thread cost known when thread ends
  - Search
Important Load Balancing Questions: Thread Dependencies

• Threads can execute in any order
  • Dependency-free loops

• Threads have a known dependency graph
  • Matrix computations

• Threads have no known dependency graph
  • Search
Important Load Balancing Questions: Thread Communication

- Threads do not communicate
  - Pixel shaders
- Thread communication is predictable
  - PDE solver
- Thread communication is unpredictable
  - N-body physics simulation
Solutions

- Task queue
  - Centralized or distributed
- Work stealing
- Work pushing
- Offline scheduling
- Self-scheduling
- ...

Rhetorical Question

Are heterogeneous cores of different ISAs in one processor useful?
Conclusions

- Modern CPUs look increasingly like heterogeneous CPUs
  - Good for power and performance
  - Potential unpredictability
- Importance of load balancing in homogeneous and heterogeneous CPUs