inst.eecs.berkeley.edu/~cs61c

CS61CL : Machine Structures

Lecture #15 – Parallelism

2009-8-12

www.xkcd.com/619

Paul Pearce, TA

Background: Threads

- A *Thread* stands for "thread of execution", it is a single stream of instructions
  - A program can split, or fork itself into separate threads, which can (in theory) execute simultaneously.
  - It has its own registers, PC, etc.
  - Threads from the same process operate in the same virtual address space
    - switching threads is faster than switching processes!
  - Are an easy way to describe/think about parallelism
- A single CPU can execute many threads by *timeslicing*

Introduction to Hardware Parallelism

- Given many threads (somehow generated by software), how do we implement this in hardware?
- "Iron Law" of Processor Performance

Execution Time = (Inst. Count)(CPI)(Cycle Time)

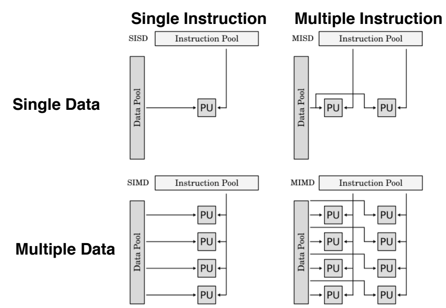
- Hardware Parallelism improves:
  - Instruction Count - If the equation is applied to each CPU, each CPU needs to do less
  - CPI - If the equation is applied to system as a whole, more is done per cycle
  - Cycle Time - Will probably be made worse in process

Disclaimers

- Please don't let today's material confuse what you have already learned about CPU's and pipelining
- When *programmer* is mentioned today, it means whoever is generating the assembly code (so it is probably a compiler)
- Many of the concepts described today are *difficult* to implement, so if it sounds easy, think of possible hazards

Flynn's Taxonomy

- Classifications of parallelism types

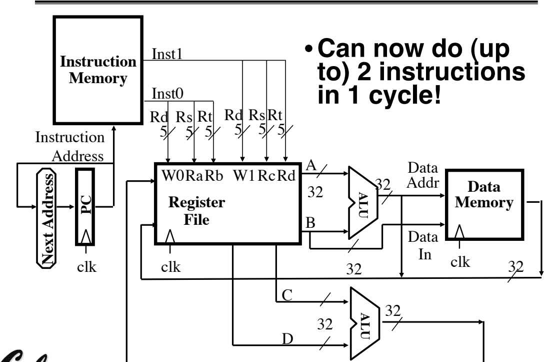


www.wikipedia.org

Superscalar

- Add more functional units or pipelines to CPU
- Directly reduces CPI by doing more per cycle
- Consider what if we:
  - Added another ALU
  - Added 2 more read ports to the RegFile
  - Added 1 more write port to the RegFile

Simple Superscalar MIPS CPU



- Can now do (up to) 2 instructions in 1 cycle!

Simple Superscalar MIPS CPU (cont.)

- Considerations
  - ISA now has to be changed
  - Forwarding for pipelining now *harder*
- Limitations
  - Programmer must explicitly generate parallel code OR require even more complex hardware for scheduling
  - Improvement only if other instructions can fill slots
  - Doesn't scale well

Superscalar in Practice

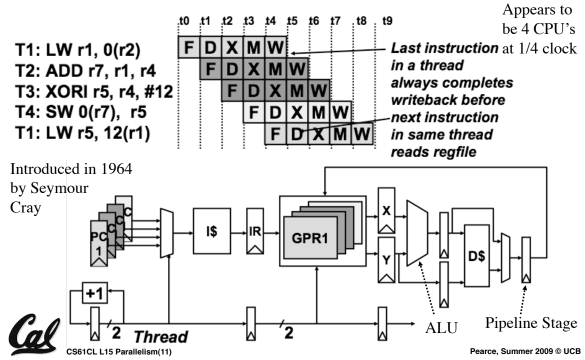
- Performance improvement depends on program and programmer being able to fully utilize all slots
- Can be parts other than ALU (like load)
- Usefulness will be more apparent when combined with other parallel techniques
- Other techniques, such as vectored data

Multithreading

- Multithreading is running multiple threads through the same hardware
- Could we do *Timeslicing* better in hardware?
- Consider if we gave the OS the abstraction of having 4 physical CPU's that share memory and each executes one thread, but we did it all on 1 physical CPU?

Static Multithreading Example

Interleave 4 threads, T1-T4, on non-bypassed 5-stage pipe



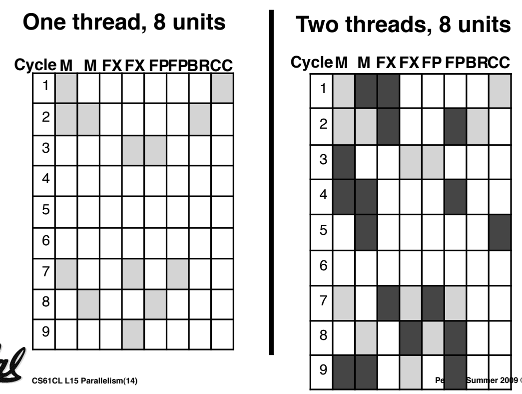
Static Multithreading Example Analyzed

- Results:
  - 4 Threads running in hardware
  - Pipeline hazards reduced
    - No more need to forward
    - No control issues
    - Less structural hazards
  - Depends on being able to fully generate 4 threads evenly
    - Example if 1 Thread does 75% of the work
      - Utilization = (% time run)(% work done)
        = (.25)(.75) + (.75)(.25) = .375
        = 37.5%

Dynamic Multithreading

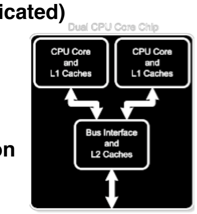
- Adds flexibility in choosing time to switch thread
- Simultaneous Multithreading (SMT)
  - Called Hyperthreading by Intel
  - Run multiple threads at the same time
  - Just allocate functional units when available
  - Superscalar helps with this

Dynamic Multithreading Example

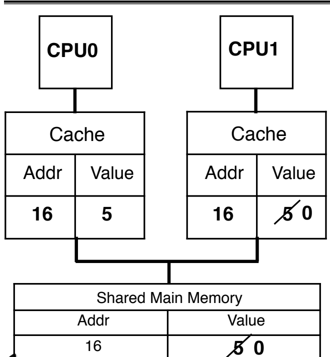


Multicore

- Put multiple CPU's on the same die
- Why is this better than multiple dies?
  - Smaller, Cheaper
  - Closer, so lower inter-processor latency
  - Can share a L2 Cache (complicated)
  - Less power
- Cost of multicore:
  - Complexity
  - Slower single-thread execution



Two CPUs, two caches, shared DRAM ...



CPU0:

```
LW R2, 16(R0)
```

CPU1:

```
LW R2, 16(R0)
```

CPU1:

```
SW R0,16(R0)
```

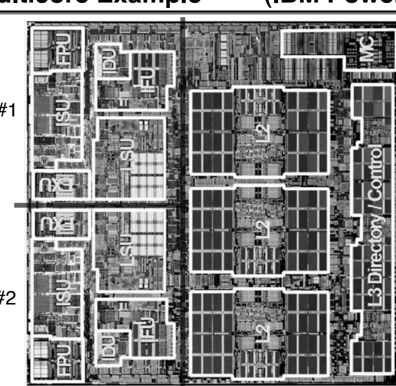
View of memory no longer "coherent".

Loads of location 16 from CPU0 and CPU1 see different values!

Write-through caches

Multicore Example (IBM Power5)

Core #1



Core #2

Shared Stuff

Administrivia

- Absolutely nothing else due!
  - You survived, congratulations….
    - Now study for for your final tomorrow!
- Final Exam: Tomorrow, 8/13, 9am-12. 277 Cory (this room)
- Final Exam Review: Right after this lecture!
- Sleep! We won't be answering any questions late into the night in an effort to get you guys to go to bed early! If you don't sleep, you won't do well.

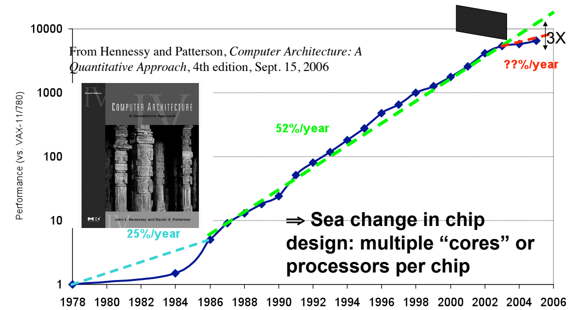
High Level Message

- Everything is changing
- Old conventional wisdom is out
- We desperately need new approach to HW and SW based on parallelism since industry has bet its future that parallelism works
- Need to create a "watering hole" to bring everyone together to quickly find that solution
  - architects, language designers, application experts, numerical analysts, algorithm designers, programmers, …

Conventional Wisdom (CW) in Computer Architecture

1. Old CW: Power is free, but transistors expensive
   - New CW: Power wall Power expensive, transistors "free"
     - Can put more transistors on a chip than have power to turn on
2. Old CW: Multiplies slow, but loads fast
   - New CW: Memory wall Loads slow, multiplies fast
     - 200 clocks to DRAM, but even FP multiplies only 4 clocks
3. Old CW: More ILP via compiler / architecture innovation
   - Branch prediction, speculation, Out-of-order execution, VLIW, …
   - New CW: ILP wall Diminishing returns on more ILP
4. Old CW: 2X CPU Performance every 18 months
   - New CW is Power Wall + Memory Wall + ILP Wall = Brick Wall

Uniprocessor Performance (SPECint)



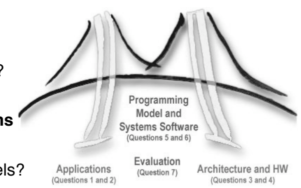
- VAX : 25%/year 1978 to 1986
- RISC + x86: 52%/year 1986 to 2002
- RISC + x86: ??%/year 2002 to present

Need a New Approach

- Berkeley researchers from many backgrounds met between February 2005 and December 2006 to discuss parallelism
  - Circuit design, computer architecture, massively parallel computing, computer-aided design, embedded hardware and software, programming languages, compilers, scientific programming, and numerical analysis
- Krste Asanovic, Ras Bodik, Jim Demmel, John Kubiatowicz, Edward Lee, George Necula, Kurt Keutzer, Dave Patterson, Koshik Sen, John Shalf, Kathy Yelick + others
- Tried to learn from successes in embedded and high performance computing
- Led to 7 Questions to frame parallel research

7 Questions for Parallelism

- Applications:
  1. What are the apps?
  2. What are kernels of apps?
- Hardware:
  3. What are HW building blocks?
  4. How to connect them?
- Programming Model & Systems Software:
  5. How to describe apps & kernels?
  6. How to program the HW?
- Evaluation:
  7. How to measure success?



(Inspired by a view of the Golden Gate Bridge from Berkeley)

Hardware Tower: What are the problems?

- Power limits leading edge chip designs
  - Intel Tejas Pentium 4 cancelled due to power issues
- Yield on leading edge processes dropping dramatically
  - IBM quotes yields of 10 – 20% on 8-processor Cell
- Design/validation leading edge chip is becoming unmanageable
  - Verification teams > design teams on leading edge processors

HW Solution: Small is Beautiful

- Expect modestly pipelined (5- to 9-stage) CPUs, FPUs, vector, Single Inst Multiple Data (SIMD) Processing Elements (PEs)
  - Small cores not much slower than large cores
- Parallel is energy efficient path to performance: $P \approx V^2$
  - Lower threshold and supply voltages lowers energy per op
- Redundant processors can improve chip yield
  - Cisco Metro 188 CPUs + 4 spares; Sun Niagara sells 6 or 8 CPUs
- Small, regular processing elements easier to verify
- One size fits all? Heterogeneous processors?

Number of Cores/Socket

- We need revolution, not evolution
- Software or architecture alone can't fix parallel programming problem, need innovations in both
- "Multicore" 2X cores per generation: 2, 4, 8, …
- "Manycore" 100s is highest performance per unit area, and per Watt, then 2X per generation: 64, 128, 256, 512, 1024 …
- Multicore architectures & Programming Models good for 2 to 32 cores won't evolve to Manycore systems of 1000's of processors
  ⇒ Desperately need HW/SW models that work for Manycore or will run out of steam (as ILP ran out of steam at 4 instructions)

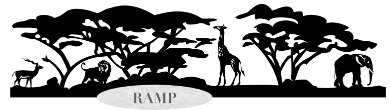
Measuring Success: What are the problems?

1. ≈ Only companies can build HW, and it takes years
2. Software people don't start working hard until hardware arrives
   - 3 months after HW arrives, SW people list everything that must be fixed, then we all wait 4 years for next iteration of HW/SW
3. How get 1000 CPU systems in hands of researchers to innovate in timely fashion on in algorithms, compilers, languages, OS, architectures, … ?
4. Can avoid waiting years between HW/SW iterations?

Build Academic Manycore from FPGAs

- As ≈ 16 CPUs will fit in Field Programmable Gate Array (FPGA), 1000-CPU system from ≈ 64 FPGAs?
  - 8 32-bit simple "soft core" RISC at 100MHz in 2004 (Virtex-II)
  - FPGA generations every 1.5 yrs; ≈ 2X CPUs, ≈ 1.2X clock rate
- HW research community does logic design ("gate shareware") to create out-of-the-box, Manycore
  - E.g., 1000 processor, standard ISA binary-compatible, 64-bit, cache-coherent supercomputer @ ≈ 150 MHz/CPU in 2007
  - RAMPants: 10 faculty at Berkeley, CMU, MIT, Stanford, Texas, and Washington
- "Research Accelerator for Multiple Processors" as a vehicle to attract many to parallel challenge

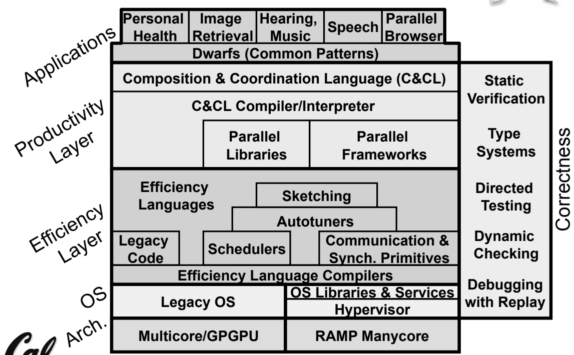
Multiprocessing Watering Hole



Parallel file system Dataflow language/computer Data center in a box
Fault insertion to check dependability Router design Compile to FPGA
Flight Data Recorder Security enhancements Transactional Memory
Internet in a box 128-bit Floating Point Libraries Parallel languages

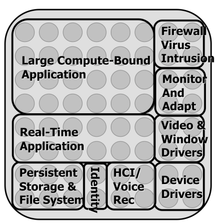
- Killer app: ≈ All CS Research, Advanced Development
- RAMP attracts many communities to shared artifact ⇒ Cross-disciplinary interactions
- RAMP as next Standard Research/AD Platform? (e.g., VAX/BSD Unix in 1980s)

ParLab Research Overview



Tessellation: The ParLab OS

- Key Concept: Space-Time Partitioning
  - Resources (cores, memory, cache, etc) are divided into discrete units which are isolated from one another
  - These divisions are able to change over time, but with time slices (we think) larger than what is currently done for processes today
  - Performance and resource guarantees are associated with partitions. This is called Quality of Service (QoS)
  - OS written completely from scratch. Coding began Jan 09.



What I do on Tessellation

- Remote System Calls (RSCs)
  - System Calls are functions that transfer control to the kernel to perform privileged operations
  - Tessellation doesn't have Disk / File System support, so we package up file related System Calls and send them over some medium (serial / Ethernet) to a remote machine for processing and return the result
- PCI / Ethernet / IOAPIC Support
  - Wrote a basic PCI bus parser, and Ethernet driver. This gives Tessellation the ability to perform basic network communication. RSCs currently run over this medium
- Standalone TCP/IP Stack Integration
  - Responsible for integrating a third-party TCP/IP stack into OS as a user space library running inside of a partition – the first example of our partitioning model
- Interrupt Routing
  - Wrote the system that allows device interrupts to be routed to specific cores or groups of cores. Random side note: AHHHH x86 is ugly! Be grateful for MIPS!

How to get involved

- We've talked in-depth about a few research projects. The purpose of this was to give you a brief overview of some of the great projects being worked on here at Cal by undergraduates just like you
- I'm an undergraduate transfer. I sat in the very seats you were in Spring 08. I began work on Tessellation by simply asking my CS162 Professor if he had a project he needed help with
- How to get involved:
  - Attend lecture and office hours, get to know the instructors
  - Have conversations with professors, ask them what they are working on, and if they need help (the answer will likely be yes)
  - Not sure who to talk too? Check out these great resources. These programs have lists of projects looking for undergraduates. You can get units, and in some cases money!
    - http://research.berkeley.edu/urap/
    - http://coe.berkeley.edu/students/current-undergraduates/student-research/uro/

Summary

- Superscalar: More functional units
- Multithread: Multiple threads executing on same CPU
- Multicore: Multiple CPU's on the same die
- The gains from all these parallel hardware techniques relies heavily on the programmer being able to map their task well to multiple threads
- Research projects need your help!

Reasons for Optimism towards Parallel Revolution this time

- End of sequential microprocessor/faster clock rates
  - No looming sequential juggernaut to kill parallel revolution
- SW & HW industries fully committed to parallelism
  - End of lazy Programming Era
- Moore's Law continues, so soon can put 1000s of simple cores on an economical chip
- Open Source Software movement means that SW stack can evolve more quickly than in past
- RAMP as vehicle to ramp up parallel research
- Tessellation as a way to manage and utilize new manycore hardware

Credits

- Thanks to the following people and possibly others for these slides:
  - Krste Asanovic
  - Scott Beamer
  - Albert Chae
  - Dan Garcia
  - John Kubiatowicz

Up next…..

Review time with Josh and James!