CS184c: Computer Architecture [Parallel and Multithreaded]

Day 1: April 3, 2001

Overview and Message Passing

Today

- This Class
- Why/Overview
- Message Passing

CS184 Sequence

- A - structure and organization
  - raw components, building blocks
  - design space
- B - single threaded architecture
  - emphasis on abstractions and optimizations including quantification
- C - multithreaded architecture

CS184b

"Architecture"

- "attributes of a system as seen by the programmer"
- "conceptual structure and functional behavior"
- Defines the visible interface between the hardware and software
- Defines the semantics of the program (machine code)

CS184b

Conventional, Single-Threaded Abstraction

- Single, large, flat memory
- sequential, control-flow execution
- instruction-by-instruction sequential execution
- atomic instructions
- single-thread “owns” entire machine
- byte addressability
- unbounded memory, call depth

This Term

- Different models of computation
  - different microarchitectures
- Big Difference: Parallelism
  - previously **model** was sequential
- Mostly:
  - Multiple Program Counters
  - threads of control

CS184a

Architecture Instruction Taxonomy

Control Threads (PCs)	*pinsts* per Control Thread	Instruction Depth	Granularity	Architecture/Examples
0	0	0	n/a	Hardwired Functional Unit (*e.g.* ECC/EDC Unit, FP MPY)
			1	FPGA
	n	1	w	Reconfigurable ALUs
1	1	c	$n_v \cdot 1$	Bitwise SIMD
			w	Traditional Processors
			$n_v \cdot w$	Vector Processors
	n	c	1	DPGA
		8	16	PADDI
		c	w	VLIW
m	n	1	1	HSRA/SCORE
	1	c	$n_v \cdot w$	MSIMD
m	1	c	1	VEGA
		8	16	PADDI-2
		c	w	MIMD (traditional)

Why?

- Why do we need a different model?
  - Different architecture?

Why?

- Density:
  - Superscalars scaling super-linear with increasing instructions/cycle
  - cost from maintaining sequential model
    - dependence analysis
    - renaming/reordering
    - single memory/RF access
  - VLIW lack of model/scalability problem
- Maybe there's a better way?

Consider

- Two network data ports
  - states: idle, first-datum, receiving, closing
  - data arrival uncorrelated between ports

CS184a

Instruction Control

- If FSMs advance orthogonally
  - (really independent control)
  - context depth => product of states
    - for full partition
  - *I.e.* w/ single controller (PC)
    - must create product FSM
    - which may lead to state explosion
      - N FSMs, with S states => S^N product states
  - This example:
    - 4 states, 2 FSMs => 16 state composite FSM

Why?

- Scalablity
  - compose more capable machine from building blocks
  - compose from modular building blocks
    - multiple chips

Why?

- Expose/exploit parallelism better
  - saw non-local parallelism when looking at IPC
  - saw need for large memory to exploit

Models?

- Message Passing (week 1)
- Dataflow (week 2)
- Shared Memory (week 3)
- Data Parallel (week 4)
- Multithreaded (week 5)
- Interface Special and Heterogeneous functional units (week 6)

Additional Key Issues

- How Interconnect? (week 7-8)
- Cope with defects and Faults? (week 9)

Message Passing

Message Passing

- Simple extension to Models
  - Compute Model
  - Programming Model
  - Architecture
- Low-level

Message Passing Model

- Collection of sequential processes
- Processes may communicate with each other (messages)
  - send
  - receive
- Each process runs sequentially
  - has own address space
- Abstraction is each process gets own processor

Programming for MP

- Have a sequential language
  - C, C++, Fortran, lisp…
- Add primitives (system calls)
  - send
  - receive
  - spawn

Architecture for MP

- Sequential Architecture for processing node
  - add network interfaces
  - process have own address space
- Add network connecting
- …minimally sufficient...

MP Architecture Virtualization

- Processes virtualize nodes
  - size independent/scalable
- Virtual connections between processes
  - placement independent communication

MP Example and Performance Issues

N-Body Problem

- Compute pairwise gravitational forces
- Integrate positions

Coding

- // params position, mass….
- F=0
- For I = 1 to N
  - send my params to p[body[I]]
  - get params from p[body[I]]
  - F+=force(my params, params)
- Update pos, velocity
- Repeat

Performance

- Body Work ~= cN
- Cycle work ~= cN^2
- Ideal Np processors: cN^2/N_p

Performance Sequential

- Body work:
  - read N values
  - compute N force updates
  - compute pos/vel from F and params
- c=t(read value) + t(compute force)

Performance MP

- Body work:
  - send N messages
  - receive N messages
  - compute N force updates
  - compute pos/vel from F and params
- c=t(send message) + t(receive message) + t(compute force)

Send/receive

- t(receive)
  - wait on message delivery
  - swap to kernel
  - copy data
  - return to process
- t(send)
  - similar
- t(send), t(receive) >> t(read value)

Sequential vs. MP

- $T_{seq} = c_{seq} N^2$
- $T_{mp}=c_{mp}N^2/N_p$
- Speedup = $T_{seq}/T_{mp} = c_{seq} \times N_p / c_{mp}$
- Assuming no waiting:
  - c_{seq} / c_{mp} ~= t(read value) / (t(send) + t(rcv))

Waiting?

- Shared bus interconnect:
  - wait O(N) time for N sends (receives) across the machine
- Non-blocking interconnect:
  - wait L(net) time after message send to receive
  - if insufficient parallelism
    - latency dominate performance

Dertouzous Latency Bound

- Speedup Upper Bound
  - processes / Latency

Waiting: data availability

- Also wait for data to be sent

Coding/Waiting

- For I = 1 to N
  - send my params to p[body[I]]
  - get params from p[body[I]]
  - F+=force(my params, params)
- How long processsor I wait for first datum?
  - Parallelism profile?

More Parallelism

- For I = 1 to N
  - send my params to p[body[I]]
- For I = 1 to N
  - get params from p[body[I]]
  - F+=force(my params, params)

Queuing?

- For I = 1 to N
  - send my params to p[body[I]]
  - get params from p[body[I]]
  - F+=force(my params, params)
- No queuing?
- Queuing?

Dispatching

- Multiple processes on node
- Who to run?
  - Can a receive block waiting?

Dispatching

- Abstraction is each process gets own processor
- If receive blocks (holds processor)
  - may prevent another process from running upon which it depends
- Consider 2-body problem on 1 node

Seitz Coding

- [see reading]

MP Issues

Expensive Communication

- Process to process communication goes through operating system
  - system call, process switch
  - exit processor, network, enter processor
  - system call, processes switch
- Milliseconds?
  - Thousands of cycles...

Why OS involved?

- Protection/Isolation
  - can this process send/receive with this other process?
- Translation
  - where does this message need to go?
- Scheduling
  - who can/should run now?

Issues

- Process Placement
  - locality
  - load balancing
- Cost for excessive parallelism
  - *E.g.* N-body on N_p < N processor ?
- Message hygiene
  - ordering, single delivery, buffering
- Deadlock
  - user introduce, system introduce

Low-Level Model

- Places burden on user [too much]
  - decompose problem explicitly
    - sequential chunk size not abstract
    - scale weakness in architecture
  - guarantee correctness in face of non-determinism
  - placement/load-balancing
    - in some systems
- Gives considerable explicit control

Low-Level Primitives

- Has the necessary primitives for multiprocessor cooperation
- Maybe an appropriate compiler target?
  - Architecture model, but not programming/compute model?

Announcements

- Note: CS25 next Monday/Tuesday
  - Seitz speaking on Tuesday
  - Dally speaking on Monday
  - (also Mead)
  - […even DeHon… :-)]
- Changing schedule (…already…)
  - Network Interface bumped up to next Mon.
    - von Eicken et. Al., Active Messages
    - Henry and Joerg, Tightly couple P-NI

Big Ideas

- Value of Architectural Abstraction
- Sequential abstraction
  - limits implementation freedom
  - requires large cost to support
    - semantic mismatch between model and execution
- Parallel models expose more opportunities

Big Ideas

- MP has minimal primitives
  - appropriate low-level model
  - too raw/primitive for user model
- Communication essential component
  - can be expensive
  - doing well is necessary to get good performance (come out ahead)
  - watch OS cost...