CS184c: Computer Architecture [Parallel and Multithreaded]

Day 6: April 19, 2001

Dataflow

Talks

- Ron Weiss
  - Cellular Computation and Communication using Engineered Genetic Regulatory Networks
  - [(Bio) Cellular Message Passing ☺]
  - Today 4pm
  - Beckman Institute Auditorium

Reading

- More dynamic schedule revision
- Multithreading next
  - HEP historical/intro Full-Empty Bits
    - May skip
  - Tullsen… simultaneous multithread…
    - Please read
  - Burns … Quantifying SMT
    - Please read

Today

- Fine-Grained Threading
- Split-Phase
- Futures / Full-Empty Bits / I-structures
- TAM

Fine-Grained Threading

- Familiar with multiple threads of control
  - Multiple PCs
- Difference in power / weight
  - Costly to switch / associated state
  - What can do in each thread
- Power
  - Exposing parallelism
  - Hiding latency

TAM Fine-Grained Threading

- Activation Frame – block of memory associated with a procedure or loop body
- Thread – piece of straightline code that does not block or branch (single entry)
- Inlet – lightweight thread for handling inputs

Analogies

- Activation Frame ~ Stack Frame
  - Heap allocated
- Procedure Call ~ Frame Allocation
  - Multiple allocation creates parallelism
- Thread ~ basic block
- Start/fork ~ branch
  - Multiple spawn creates local parallelism
- Switch ~ conditional branch

Fine-grained Threading

- Computational model with explicit parallelism, synchronization

Split-Phase Operations

- Separate request and response side of operation
  - Idea: tolerate long latency operations
- Contrast with waiting on response

Canonical Example: Memory Fetch

- Conventional
  - Perform read
  - Stall waiting on reply
- Optimizations
  - Prefetch memory
  - Then access later
- Goal: separate request and response

Split-Phase Memory

- Send memory fetch request
  - Have reply to different thread
- Next thread enabled on reply
- Go off and run rest of this thread (other threads) between request and reply

Prefetch vs. Split-Phase

- Prefetch in sequential ISA
  - Must guess delay
  - Can request before need
  - …but have to pick how many instructions to place between request and response

Split-Phase Communication

- Also for non-rendezvous communication
  - Buffering
- Overlaps computation with communication
- Hide latency with parallelism

Key Element of DF Control

- Synchronization on Data Presence
- Construct:
  - Futures
  - Full-empty bits
  - I-structures

Future

- Future is a promise
- An indication that a value will be computed
  - And a handle for getting a handle on it
- Sometimes used as program construct

Future

- Future computation immediately returns a future
- Future is a handle/pointer to result
- (define (dot a b)
  - (cons (future (* (first a) (first b)))
    - (dot (rest a) (rest b))))

Fib

- (define (fib n)
  - (if (< n 2) 1 (+ (future (fib (- n 1)))
  - (future (fib (- n 2))))))

Strict/non-strict

- Strict operation requires the **value** of some variable
  - E.g. add, multiply
- Non-strict operation only needs the handle for a value
  - E.g. cons, procedure call
    - (anything that just passes the handle off)

Futures

- Safe with functional routines
  - Create dataflow
- Can introduce non-determinacy with side-effecting routines
  - Not clear when operation completes

Future/Side-Effect hazard

- (define (decrement! a b)
  - (set! a (- a b)))
- (print (* (future (decrement! c d))
- (future (decrement! d 2))))

Full/Empty bit

- Tag on data indicates data presence
  - E.g. tag in memory, RF
  - Like Scoreboard present bit
- When computation allocated, set to empty
  - E.g. operation issued into pipe, future call made
- When computation completes
  - Future computation completes
  - Data written into slot (register, memory)
  - Bit set to full
- On data access
  - Bit full, strict operation can get value
  - Bit empty, strict operation block on completion

I-Structure

- Array/object with full-empty bits on each field
- Allocated empty
- Fill in value as compute
- Strict access on empty
  - Queue requester in structure
  - Send value to requester when written and becomes full

I-Structure

- Allows efficient “functional” updates to aggregate structures
- Can pass around pointers to objects
- Preserve ordering/determinacy
- E.g. arrays

Threaded Abstract Machine

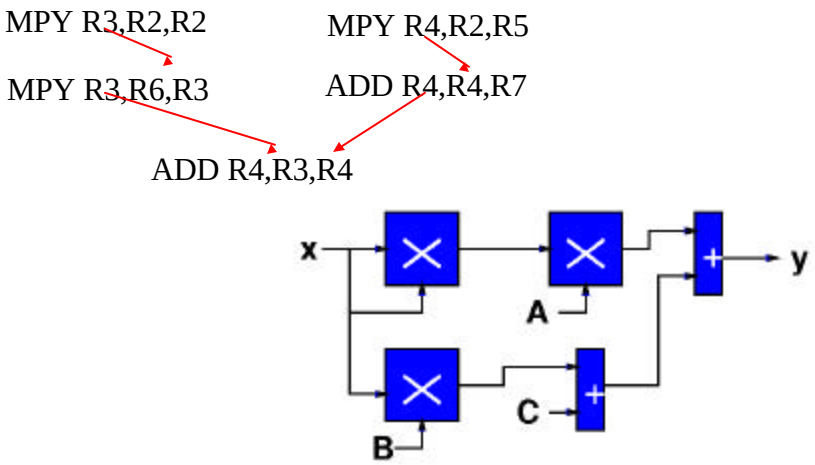
TAM

- Parallel Assembly Language
- Fine-Grained Threading
- Hybrid Dataflow
- Scheduling Hierarchy

Pure Dataflow

- Every operation is dataflow enabled
- Good
  - Exposes maximum parallelism
  - Tolerant to arbitrary delays
- Bad
  - Synchronization on event costly
    - More costly than straightline code
    - Space and time
  - Exposes non-useful parallelism

Old Example
~~Task Has Parallelism~~



Hybrid Dataflow

- Use straightline/control flow
  - When successor known
  - When more efficient
- Basic blocks (fine-grained threads)
  - Think of as coarser-grained DF objects
  - Collect up inputs
  - Run basic block like conv. RISC basic-block (known non-blocking)

Great Project

- Model and assess control flow versus dataflow
- General formulation
  - Whole problem could be run control or data flow
  - Pick when to use control vs. dataflow
  - Base on
    - Cost of synchronization
    - Model of delay predictability
    - Minimize expected runtime

Stopped Here

4/19/01

TL0 Model

- Activition Frame (like stack frame)
  - Variables
  - Synchronization
  - Thread stack (continuation vectors)
- Heap Storage
  - I-structures

TL0 Ops

- RISC-like ALU Ops
- FORK
- SWITCH
- STOP
- POST
- FALLOC
- FFREE
- SWAP

Scheduling Hierarchy

- Intra-frame
  - Related threads in same frame
  - Frame runs on single processor
  - Schedule together, exploit locality
    - (cache, maybe regs)
- Inter-frame
  - Only swap when exhaust work in current frame

Intra-Frame Scheduling

- Simple (local) stack of pending threads
- Fork places new PC on stack
- STOP pops next PC off stack
- Stack initialized with code to exit activation frame
  - Including schedule next frame
  - Save live registers

TL0/CM5 Intra-frame

- Fork on thread
  - Fall through 0 inst
  - Unsynch branch 3 inst
  - Successful synch 4 inst
  - Unsuccessful synch 8 inst
- Push thread onto LCV 3-6 inst

Fib Example

- [look at how this turns into TL0 code]

Multiprocessor Parallelism

- Comes from frame allocations
- Runtime policy where allocate frames
  - Maybe use work stealing?

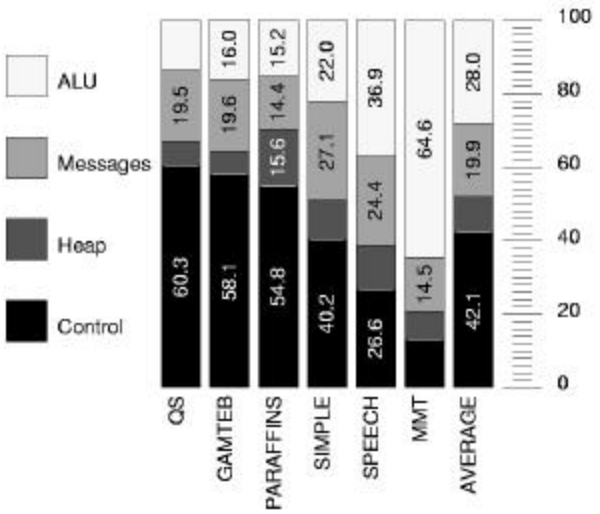
Frame Scheduling

- Inlets to non-active frames initiate pending thread stack (RCV)
- First inlet may place frame on processor's runable frame queue
- SWAP instruction picks next frame branches to its enter thread

CM5 Frame Scheduling Costs

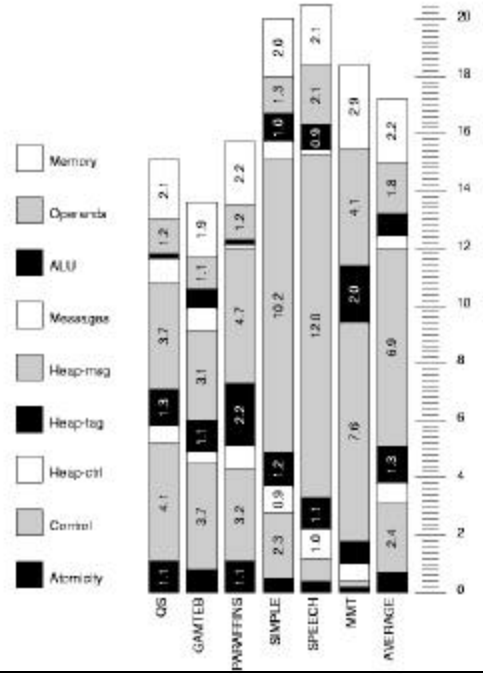
- Inlet Posts on non-running thread
  - 10-15 instructions
- Swap to next frame
  - 14 instructions
- Average thread cost 7 cycles
  - Constitutes 15-30% TL0 instr

Instruction Mix



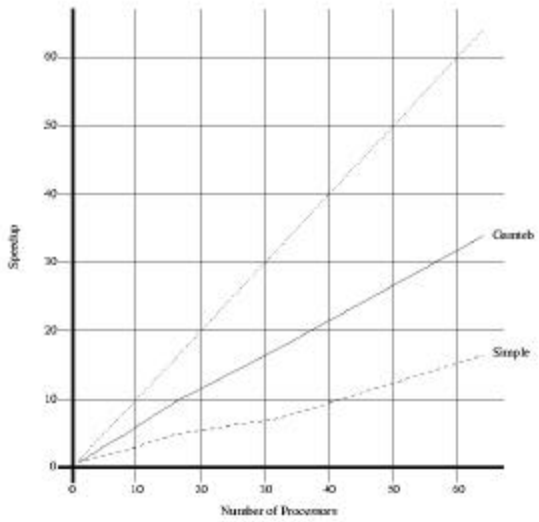
[Culler et. Al. JPDC, July 1993]

Cycle Breakdown



[Culler et. Al. JPDC, July 1993]

Speedup Example



[Culler et. Al. JPDC, July 1993]

Thread Stats

- Thread lengths 3—17
- Threads run per “quantum” 7—530

	QS	Gamteb	Paraffins	Simple	Speech	MMT
Ave TL0 Insts. per Thread	2.6	3.2	3.1	5.3	6.3	17.6
Threads per Quanta	11.5	13.5	215.5	7.5	16.7	530.0
RCV Size when Scheduled	1.1	1.6	1.3	1.4	1.0	1.6
Threads forked during Quantum	8.8	10.2	168.4	4.1	11.7	406.6
Threads posted during Quantum	1.5	1.6	45.7	1.9	4.0	121.9
Quanta per Invocation	4.1	3.4	2.7	4.8	21.7	3.4

Table 9: Dynamic scheduling characteristics under TAM for two programs on a 64 processor CM-5

[Culler et. Al. JPDC, July 1993]

Great Project

- Develop optimized μArch for TAM
  - Hardware support/architecture for single-cycle thread-switch/post

Big Ideas

- Primitives
  - Parallel Assembly Language
  - Threads for control
  - Synchronization (post, full-empty)
- Latency Hiding
  - Threads, split-phase operation
- Exploit Locality
  - Create locality
    - Scheduling quanta