CS184c: Computer Architecture [Parallel and Multithreaded]

Day 7: April 24, 2001
Threaded Abstract Machine (TAM)
Simultaneous Multi-Threading (SMT)

Reading

- Shared Memory
  - Focus: H&P Ch 8
    - At least read this…
  - Retrospectives
    - Valuable and short
  - ISCA papers
    - Good primary sources

Today

- TAM
- SMT

Threaded Abstract Machine

TAM

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

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?

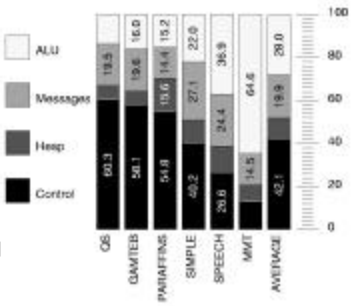
Frame Scheduling

- 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

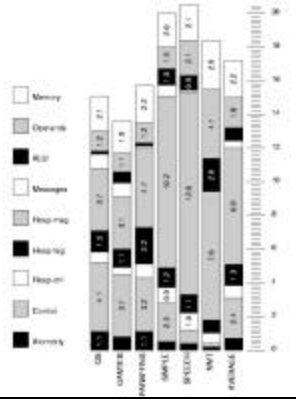
- Inlet Posts on non-running thread
  - 10-15 instructions
- Swap to next frame
  - 14 instructions
- Average thread cost 7 cycles
  - Constitutes 15-30% TL0 instr

Instruction Mix



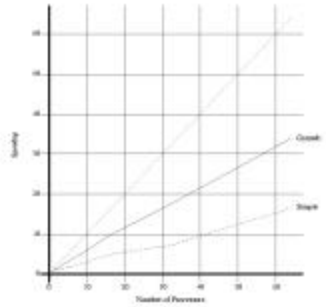
[Culler et. Al. JPDC, July 1993]

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 Instr. per Thread	2.6	3.2	3.1	5.5	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

Multithreaded Architectures

Problem

- Long latency of operations
  - Non-local memory fetch
  - Long latency operations (mpy, fp)
- Wastes processor cycles while stalled
- If processor stalls on return
  - Latency problem turns into a throughput (utilization) problem
  - CPU sits idle

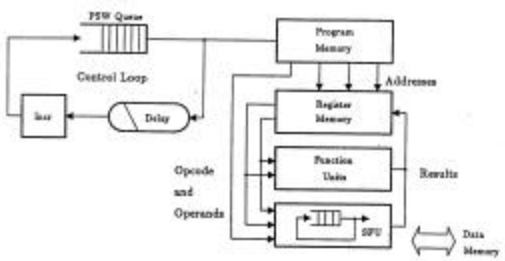
Idea

- Run something else useful while stalled
- In particular, another thread
  - Another PC
- Again, use parallelism to "tolerate" latency

HEP/μUnity/Tera

- Provide a number of contexts
  - Copies of register file…
- Number of contexts ≥ operation latency
  - Pipeline depth
  - Roundtrip time to main memory
- Run each round-robin

HEP Pipeline



[figure: Arvind+Innucci, DFVLR'87]

Strict Interleaved Threading

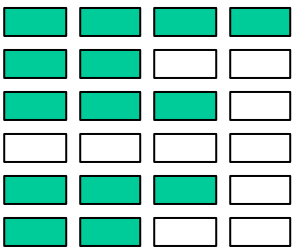
- Uses parallelism to get throughput
- Potentially poor single-threaded performance
  - Increases end-to-end latency of thread

SMT

Can we do both?

- Issue from multiple threads into pipeline
- No worse than (super)scalar on single thread
- More throughput with multiple threads
  - Fill in what would have been empty issue slots with instructions from different threads

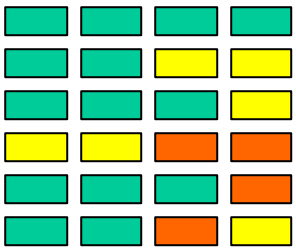
SuperScalar Inefficiency



Unused Slot

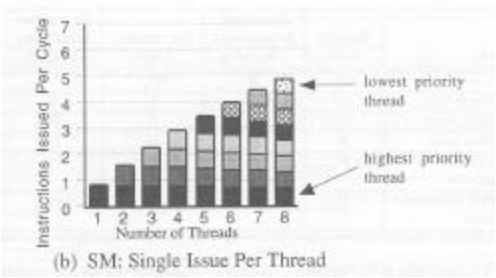
Recall: limited Scalar IPC

SMT Promise



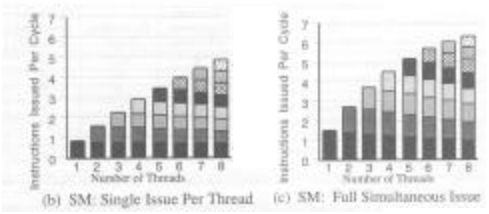
Fill in empty slots with other threads

SMT Estimates (ideal)



[Tullsen et. al. ISCA '95]

SMT Estimates (ideal)



[Tullsen et. al. ISCA '95]

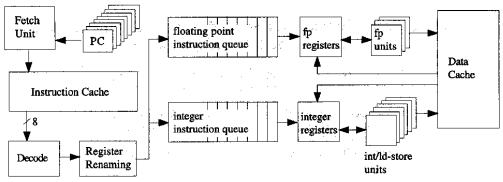
SMT uArch

- Observation: exploit register renaming
  - Get small modifications to existing superscalar architecture

Stopped Here

4/24/01

SMT uArch



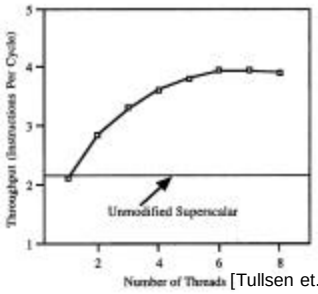
- N.B. remarkable thing is how similar superscalar core is

[Tullsen et. al. ISCA '96]

SMT uArch

- Changes:
  - Multiple PCs
  - Control to decide how to fetch from
  - Separate return stacks per thread
  - Per-thread reorder/commit/flush/trap
  - Thread id w/ BTB
  - Larger register file
    - More things outstanding

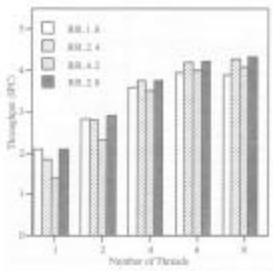
Performance



[Tullsen et. al. ISCA '96]

Optimizing: fetch freedom

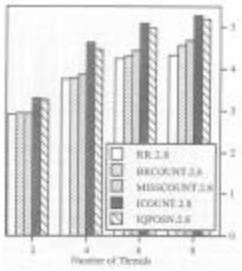
- RR=Round Robin
- RR.X.Y
  - X – threads do fetch in cycle
  - Y – instructions fetched/thread



[Tullsen et. al. ISCA '96]

Optimizing: Fetch Alg.

- ICOUNT – priority to thread w/ fewest pending instrs
- BRCOUNT
- MISSCOUNT
- IQPOSN – penalize threads w/ old instrs (at front of queues)

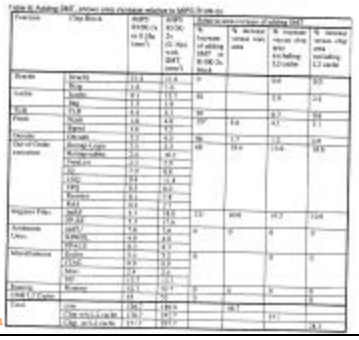


[Tullsen et. al. ISCA '96]

Throughput Improvement

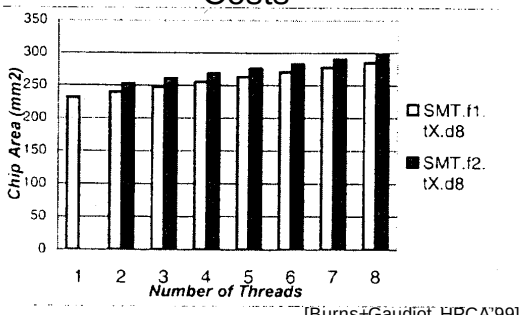
- 8-issue superscalar
  - Achieves little over 2 instructions per cycle
- Optimized SMT
  - Achieves 5.4 instructions per cycle on 8 threads
- 2.5x throughput increase

Costs



[Burns+Gaudiot HPCA'99]

Costs



[Burns+Gaudiot HPCA'99]

Not Done, yet…

- Conventional SMT formulation is for coarse-grained threads
- Combine SMT w/ TAM ?
  - Fill pipeline from multiple runnable threads in activation frame
  - ?multiple activation frames?
  - Eliminate thread switch overhead?

Thought?

- SMT reduce need for split-phase operations?

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