Simultaneous Multithreading

Hank Levy
Dept. of Computer Science and Engineering
University of Washington

Susan Eggers, Dean Tullsen
Jack Lo, Sujay Parekh (UW)

Joel Emer, Rebecca Stamm (DEC)

Talk Outline

- The Problem
- Simultaneous Multithreading and its Potential
- An Architecture for Simultaneous Multithreading
- Parallel Program Performance
- Conclusions

Problems and Opportunities

- Memory latencies are getting longer (relatively)
- Issue bandwidth is increasing (superscalar)
- Number of functional units is increasing
- BUT, processor utilization is **decreasing**

Alternatives to Parallelism

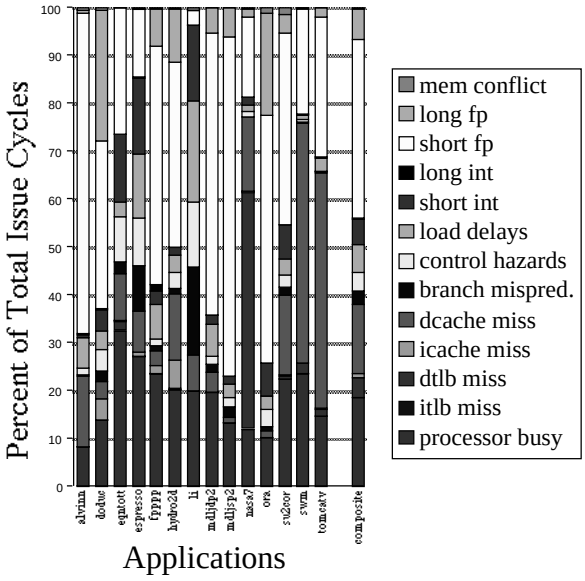
- Three current directions are:
  - superscalar processors: multiple instructions per cycle
  - fine-grained multithreading: context switch to a new thread every cycle
  - single-chip multiprocessors (several CPUs/chip)

Issue Width of Current Processors

Processor	Issue Rate
Pentium Pro	3
PowerPC 604	4
MIPS R10000	5
Alpha 21264	6
UltraSparc	4
HP PA-8000	4

Where Have All the Cycles Gone?

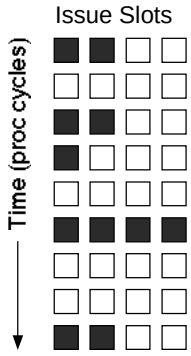
(simulated 8-issue Alpha 21164)



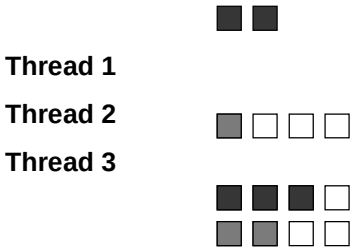
The Challenge

- How do we organize processors in the future to increase:
  - utilization of processor resources
  - overall performance of both sequential and parallel programs

Superscalar Execution



with Fine-Grain Multithreading



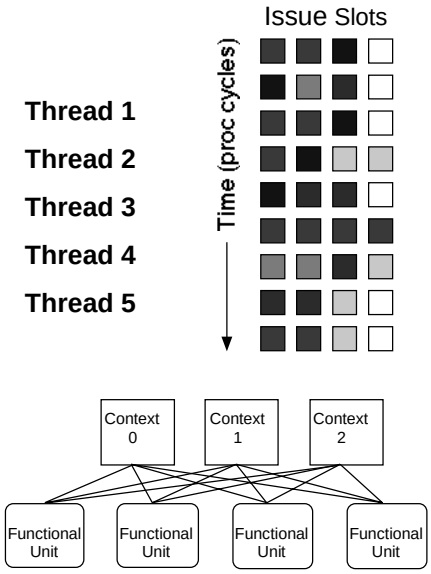
Talk Outline

- The Problem
- Simultaneous Multithreading and its Potential
- An Architecture for Simultaneous Multithreading
- Parallel Program Performance
- Conclusions

Simultaneous Multithreading

- A **Simultaneous Multithreading** (SMT) processor consists of:
  - multiple thread hardware contexts (register sets)
  - multiple functional units
  - multiple instruction issue
- In an SMT processor, multiple threads can issue to multiple function units in a single cycle.
- SMT combines the ILP capability of superscalars with the latency-hiding capability of multithreading.

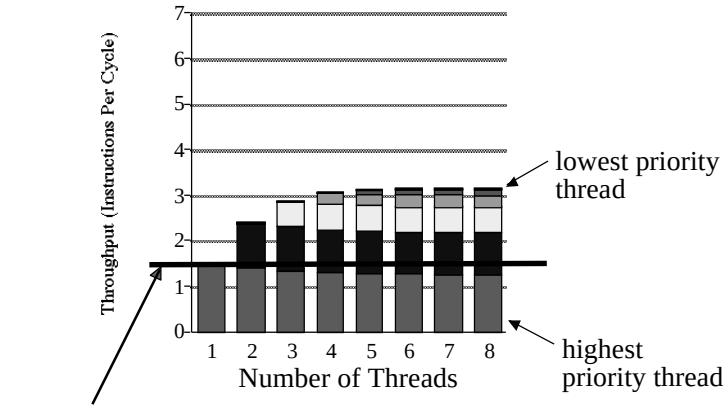
Simultaneous Multithreading (SMT)



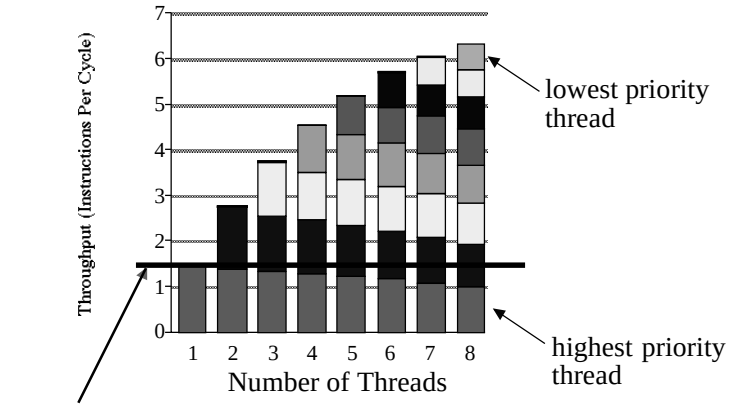
The Simulator

- We simulated a detailed SMT processor, based on Alpha 21164, augmented with:
  - wide superscalar and multithreading support
  - 8 instructions/cycle
  - 10 FUs (4 int, 2 FP, 3 load/store, 1 branch)
  - larger caches
  - improved branch prediction
  - 8 hardware contexts
- We generate code with Multiflow trace-scheduling compiler

Results: Superscalar and Fine-grain Multithreading



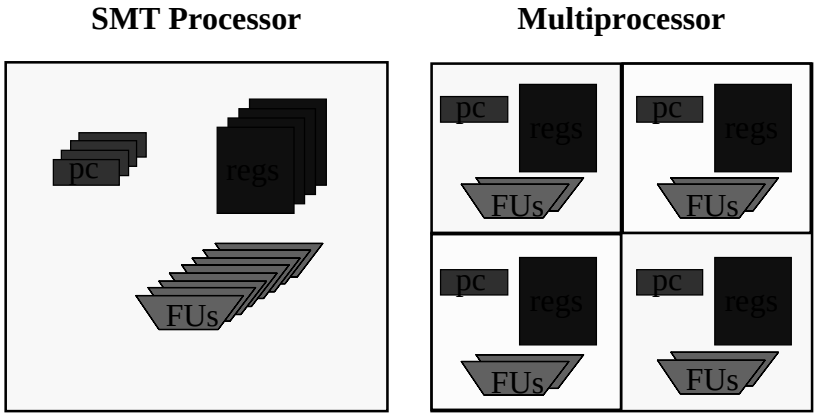
Results: Simultaneous Multithreading



Single-Chip Multiprocessing

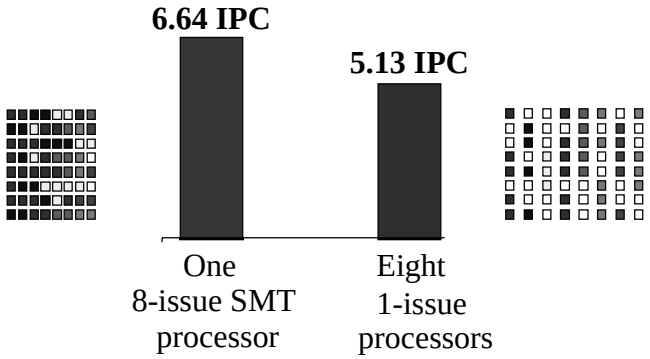
- Another way to use chip resources to build an MP on a chip, with each proc. a superscalar.
- This is similar to SMT in some ways:
  - both have multiple register sets
  - both have multiple functional units
  - both issue multiple instructions per cycle
- The difference is dynamic scheduling of resources.

Single-chip Multiprocessing



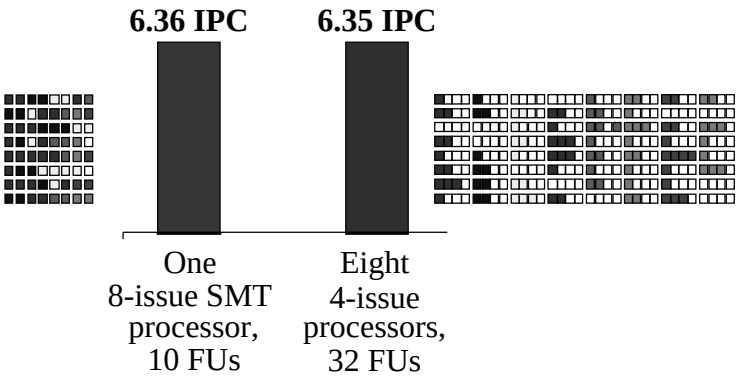
Simultaneous Multithreading vs. Single-chip Multiprocessing

Common Elements: 32 FUs, 8 register sets
total issue bandwidth of 8 IPC



Simultaneous Multithreading vs. Single-chip Multiprocessing

Common Elements: 8 register sets



SMT vs. SMP

- SMT requires fewer resources to achieve the same performance as MP.
- The advantage of SMT is greater if there are fewer threads.
- SMT is more flexible for design
  - MP can add only in units of full processors
  - in MP to add a FU, need to add 1 per processor
  - in SMT, we can benefit from the addition of 1 FU or 1 register set, i.e., there is finer granularity of design

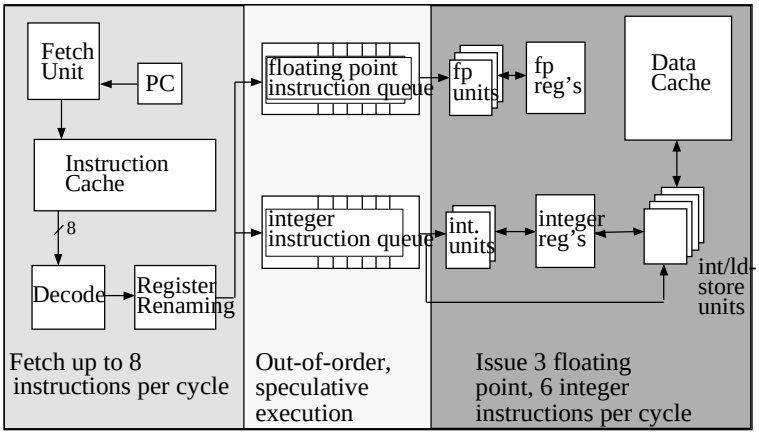
Talk Outline

- The Problem
- Simultaneous Multithreading and its Potential
- An Architecture for Simultaneous Multithreading
- Parallel Program Performance
- Conclusions

An SMT Architecture

- SMT has the potential to greatly increase processor utilization, but is this achievable?
- We set out to define a more detailed architecture with three primary goals:
  1. Minimize the impact on conventional superscalar design
  2. Achieve significant throughput gains with multiple threads
  3. Minimize the performance impact on a single thread executing alone

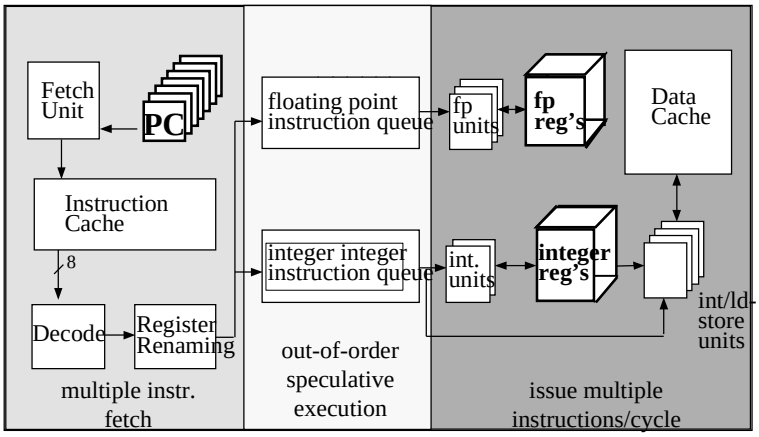
A Conventional Superscalar Architecture



From Superscalar to SMT

- There are several small changes need to support SMT:
  - multiple program counters
  - multiple return stacks
  - per-thread instruction retirement, trap and queue flush mechanisms
  - thread identifiers, e.g., with BTB entries, TLB entries
- There is one **big** item:
  - large register file: # threads * 32 + renaming regs

An SMT Architecture



Coping with a Large Register File

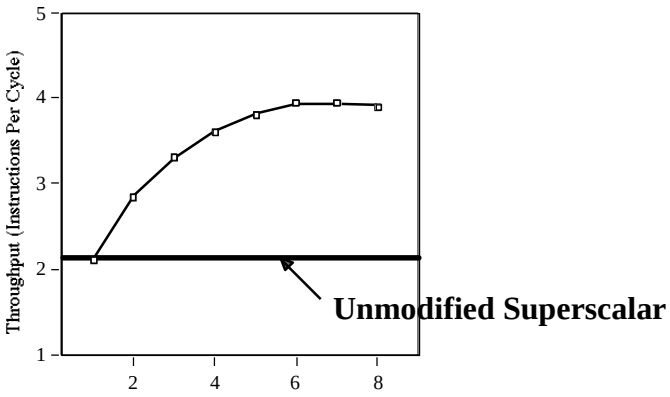
Fetch	Decode	Rename	Queue	Reg Read	Exec	Commit

Superscalar pipeline

Fetch	Decode	Rename	Queue	Reg Read	Reg Read	Exec	Reg Write	Commit

Pipeline modified for 2-cycle register access

Performance of the Base Design



Comments

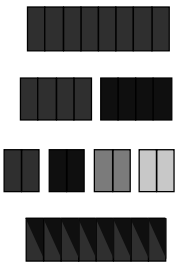
- These newer simulations assume a dynamically scheduled superscalar and SMT.
- They use the longer pipeline for SMT.
- They show improvements for SMT, but less than the potential predicted by our first experiments.
- How can we improve this architecture to help SMT's performance?

Improving Fetch Throughput

- Our analysis shows that for an SMT processor, the fetch unit is a problem and an opportunity.
- In particular, the fetch unit on an SMT has two significant advantages over a conventional architecture:
  1. Can fetch from multiple threads at once
  2. Can **choose** which threads to fetch

Partitioning the Fetch Unit

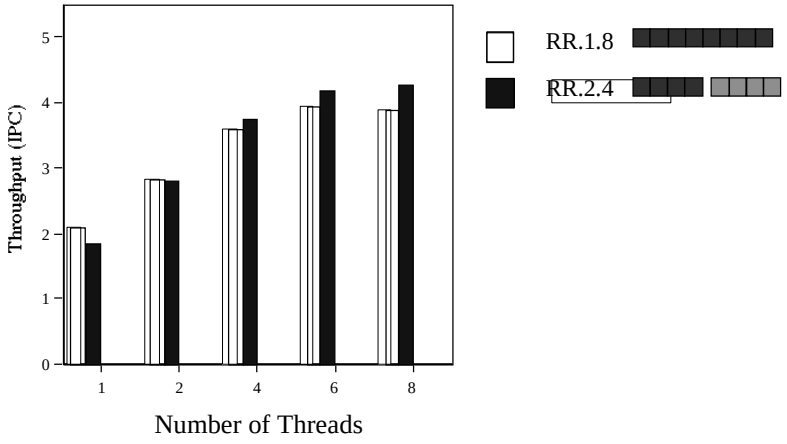
Policies: fetch up to

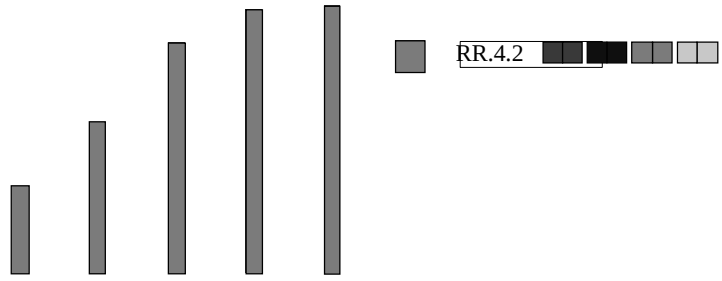


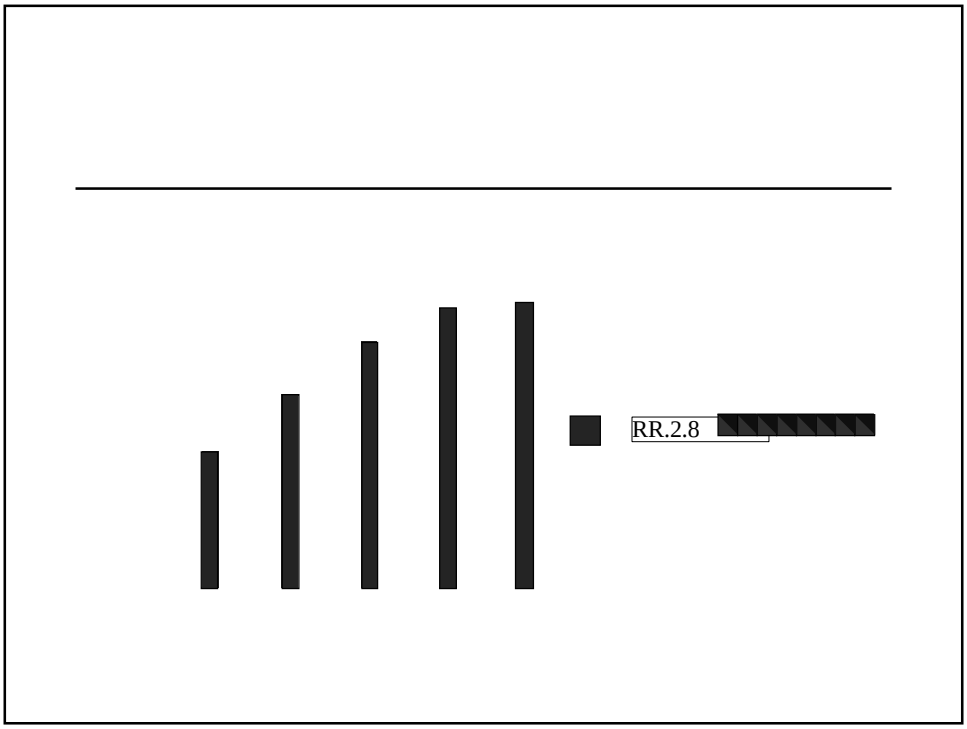
RR.1.8	**baseline**: 8 inst's from 1 thread
RR.2.4	4 inst's from each of 2 threads
RR.4.2	2 inst's from each of 4 threads
RR.2.8	8 inst's from each of 2 threads but no more than eight total

Round Robin thread priority

Partitioning the Fetch Unit





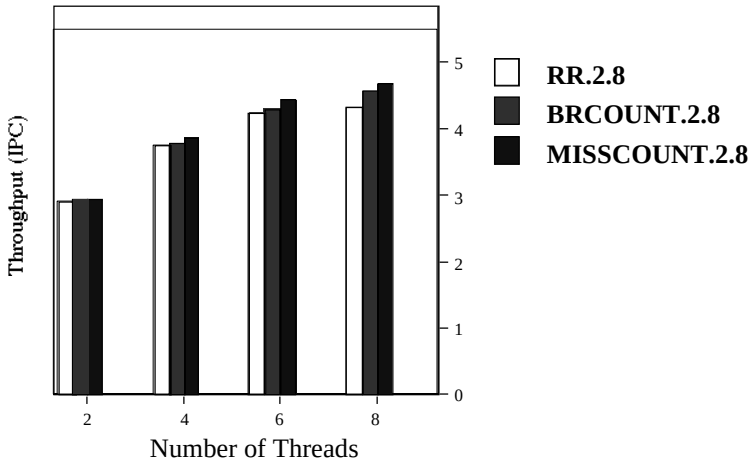


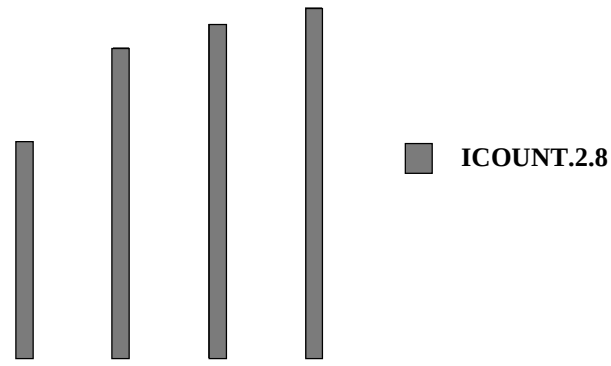
Exploiting Choice in the Fetch Unit

wrong-path	IQ full	
☐	☐	**RR** -- intelligent round-robin priority **(baseline)**
☑	☐	**BRCOUNT** -- priority to threads with fewest outstanding branches
☐	☑	**MISSCOUNT** -- priority to threads with fewest outstanding cache misses
☐	☑	**ICOUNT** -- priority to threads with fewest unissued instructions

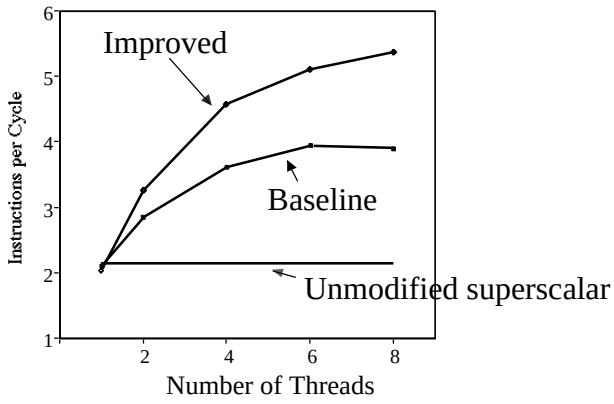
The Policies:

Exploiting Thread Choice in the Fetch Unit





Tuning the Base Architecture: How Far We've Come



Talk Outline

- The Problem
- Simultaneous Multithreading and its Potential
- An Architecture for Simultaneous Multithreading
- Parallel Program Performance
- Conclusions

SMT and Parallel Programs

- Our previous results looked at utilization for multiprogrammed workloads.
- How does SMT perform relative to small on-chip MPs for parallel programs?
- To evaluate this, we compared an SMT processor to:
  - a 2-processor 4-issue/processor MP
  - a 4-processor 2-issue/processor MP
- We simulate parallel programs of various types from the SPASH suite plus some others

SMT vs. SMP

- Our results show that SMT gets better speedup than SMP.
- The reason is the static partitioning of resources in an SMP. To show this explicitly, we measured the percentage of cycles in which 1 of the SMP processors ran out of a resource, and that same resource was available in the same cycle on another of the processors.

Converting TLP to IPL

- In general, there are two kinds of parallelism:
  - instruction-level parallelism (IPL)
  - thread-level parallelism (TLP)
- On an SMT processor, TLP is converted to ILP. The result is that the processor can perform well in the presence of only one or the other, or both. In contrast:
  - a superscaler will perform badly if IPL is limited
  - an MP performs badly if TLP is limited

SMT vs. SMP

- Simultaneous Multithreading and Shared-Memory Multiprocessing are not mutually exclusive.
- In the end, we expect to see multiprocessors where *each* processor is an SMT processor.

Talk Outline

- The Problem
- Simultaneous Multithreading and its Potential
- An Architecture for Simultaneous Multithreading
- Thread-Level vs. Instruction-Level Parallelism
- Conclusions

Conclusions

- This talk is too long...
- But, it shows that Simultaneous Multithreading has the potential to significantly improve utilization of future processors.
- SMT effectively hides latencies of many kinds.
- SMT provides superior speedup for parallel programs relative to small-scale single-chip MPs.
- An SMT architecture is achievable without enormous changes to current out-of-order superscalars.

For more info and papers:

http://www.cs.washington.edu/research/smt