CS252 Lecture Notes
Multithreaded Architectures

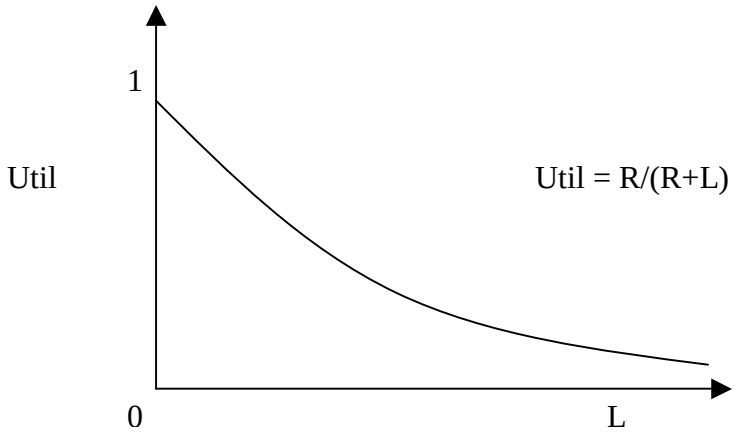
Concept
Tolerate or mask long and often unpredictable latency operations by switching to another context, which is able to do useful work.

Situation Today – Why is this topic relevant?
- ILP has been exhausted which means thread level parallelism must be utilized
- The gap between processor performance and memory performance is still large
- There is ample real-estate for implementation
- More applications are being written with the use of threads and multitasking is ubiquitous
- Multiprocessors are more common
- Network latency is analogous to memory latency
- Complex scheduling is already being done in hardware

Classical Problem
60's and 70's
- I/O latency prompted multitasking
- IBM mainframes
- Multitasking
- I/O processors
- Caches within disk controllers

Requirements of Multithreading
- Storage need to hold multiple context's PC, registers, status word, etc.
- Coordination to match an event with a saved context
- A way to switch contexts
- Long latency operations must use resources not in use

To visualize the effect of latency on processor utilization, let R be the run length to a long latency event, let L be the amount of latency then:



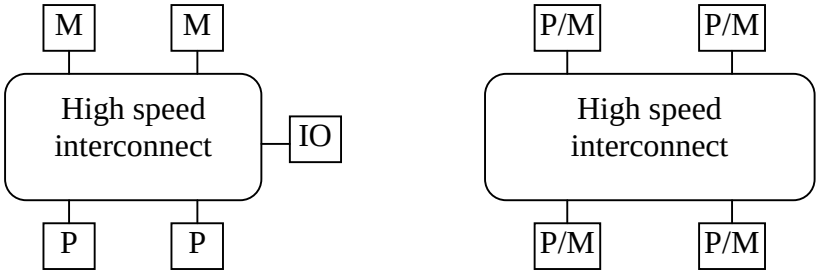
80's

Problem was revisited due to the advent of graphics workstations

- Xerox Alto, TI Explorer
- Concurrent processes are interleaved to allow for the workstations to be more responsive.
- These processes could drive or monitor display, input, file system, network, user processing
- Process switch was slow so the subsystems were microprogrammed to support multiple contexts

Scalable Multiprocessor

- Dance hall – a shared interconnect with memory on one side and processors on the other.
- Or processors may have local memory



How do the processors communicate?

Shared Memory

- Potential long latency on every load
- Cache coherency becomes an issue
- Examples include NYU's Ultracomputer, IBM's RP3, BBN's Butterfly, MIT's Alewife, and later Stanford's Dash.
- Synchronization occurs through share variables, locks, flags, and semaphores.

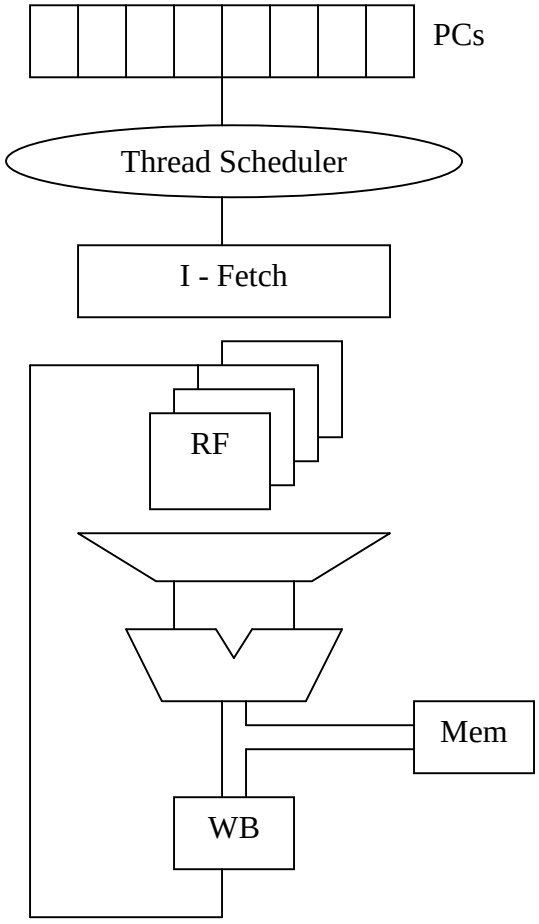
Message Passing

- Programmer deals with latency. This enables them to minimize the number of messages, while maximizing the size, and this scheme allows for delay minimization by sending a message so that it reaches the receiver at the time it expects it.
- Examples include Intel's PSC and Paragon, Caltech's Cosmic Cube, and Thinking Machines' CM-5
- Synchronization occurs through send and receive

Cycle-by-Cycle Interleaved Multithreading

Burton Smith

- Currently chief scientist at Cray
- Denelcor HEP1 (1982), HEP2
- Horizon, which was never built
- Tera, MTA



Features of this architecture

- An instruction from a different context is launched at each clock cycle
- No interlocks or bypasses thanks to a non-blocking pipeline

Optimizations:

- Leaving context state in proc (PC, register #, status)
- Assigning tags to remote request and then matching it on completion

Additional optimizations:

- A full/empty bit on every memory word allowing for automatic and efficient synchronization. This obviates the need for semaphores, locks, etc. and it may reduce polling time done by the processor by moving that job to a controller.

Challenges with this approach

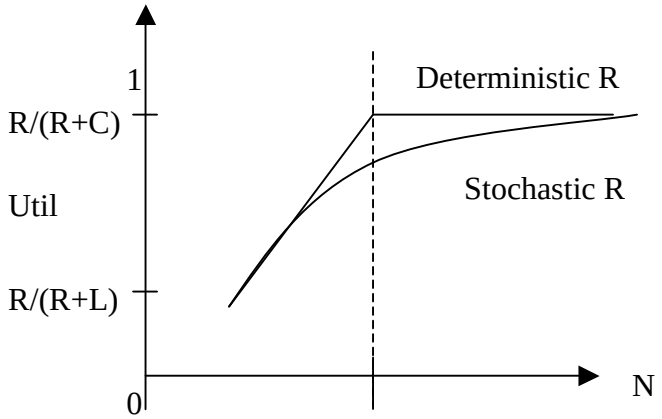
- Instruction bandwidth
- Since instructions are being grabbed from many different contexts, instruction locality is degraded and the I-cache miss rate rises.
- Register file access time increases due to the fact that the regfile had to significantly increase in size to accommodate many separate contexts. In fact, the HEP and Tera use SRAM to implement the regfile, which means longer access times. Some of this may be alleviated through increasing the pipeline depth at the cost of additional latency.
- Single thread performance is significantly degraded since the context is forced to switch to a new thread even if none are available.
- Insufficient pipelining (bandwidth bottleneck)
- Unpipelined FP unit – must stall or reflect up into thread scheduler
- Very high bandwidth network, which is fast and wide
- Retries on load empty or store full

Improving Single Thread Performance

- Do more operations per instruction (VLIW)
- Allow multiple instructions to issue into pipeline from each context. This could lead to pipeline hazards, so other safe instructions could be interleaved into the execution. For Horizon & Tera the compiler detects such data dependencies and the hardware enforces it by switching to another context if detected. This is implemented by inserting into each instruction a field which indicates its minimum number of independent successors over all possible control flows.
- Switch on load
- Switch on miss
- Switching on load or miss will increase the context switch time. Consider:

Type of Switch	R	C
Cycle	1	0
Load	5-10	1-2
Miss	5-100	5-10 (+ hit time)

- Max Utilization = R/(R+C)
- Where does saturation occur?



Nsat = L/(R+C) + 1

Cautions

- Pipeline bottlenecks are more apparent with effective multithreading. For example, an unpipelined FP unit needs to reflect reservation up to the thread scheduler
- Architected delay slots complicate multithreaded control logic. For example, an exception occurs on instructions in branch delay slot, while fetch and exec are at branch target
- Register file delay and bandwidth

Other Concepts

- Tagged Memory is another concept that is often circulated in which memory is thought of as a set of objects instead of homogenous bits. Examples of this are lisp machines, data-flow machines, and J-machines.
- Physical vs Virtual Parallelism