Virtual Memory

- It's just another level in the cache/memory hierarchy
- *Virtual memory* is the name of the technique that allows us to view main memory as a cache of a larger memory space (on disk).

```plaintext
cpu
  \rightarrow caching
  \rightarrow cache
  \rightarrow caching
  \rightarrow memory
  \rightarrow virtual memory
```

disk

Virtual Memory

Provides
- performance (through the caching effect)
- protection
- ease of programming/compilation
- efficient use of memory

Virtual Memory mapping
Address translation via the page table

- virtual address
  - virtual page number
  - page offset

- page table reg
  - valid
  - physical page number
  - page offset

- physical address
  - physical page number
  - page offset

- all page mappings are in the page table, so hit/miss is determined solely by the valid bit (i.e., no tag)
  - so why is this fully associative???

Making Address Translation Fast

- A cache for address translations: translation lookaside buffer

Virtual Memory Key Points

- How does virtual memory provide:
  - protection?
  - sharing?
  - performance?
  - illusion of large main memory?

- Virtual Memory requires twice as many memory accesses, so we cache page table entries in the TLB.
- Three things can go wrong on a memory access: cache miss, TLB miss, page fault.
Multiprocessors and Multiprocessing

more is better?

Multiprocessors

- why would you want a multiprocessor?
- what things can it do well?
- What things can’t it do well?
- What things can it do that a bunch of computers can’t do?
- How much are you willing to pay?

Classifying Multiprocessors

- Interconnection Network
- Memory Topology
- Programming Model

Interconnection Network

- Bus
- Network
- pros/cons?
Memory Topology

- UMA (Uniform Memory Access)
- NUMA (Non-uniform Memory Access)
- pros/cons?

Programming Model

- Shared Memory -- every processor can name every address location
- Message Passing -- each processor can name only its local memory. Communication is through explicit messages (multicomputer).
- pros/cons?

Parallel Programming

Processor A
index = i++;

Processor B
index = i++;

- Shared-memory programming requires synchronization to provide mutual exclusion and prevent race conditions
  - locks (semaphores)
  - barriers

Multiprocessor Caches (Shared Memory)

- the problem -- cache coherency
- the solution?
Cache Coherency

- **write-update**
  - on each write, each cache holding that location updates its value
- **write-invalidate <= most common**
  - on each write, each cache holding that location invalidates the cache line.

- both schemes MUCH easier on a bus-based multiprocessor
- potentially requires a LOT of messages, but...

A good cache coherency protocol can avoid sending unnecessary (and expensive) invalidate or update messages.
- Allows each cache line to be in one of several *states*.
- MESI (Illinois)
  - modified
  - exclusive
  - shared
  - invalid

How do you know when an external read/write occurs?
- Snooping protocols
- Directory protocols

Snooping protocols
- Message-passing vs. Shared Memory
- Shared Memory is more intuitive, but creates problems for both the programmer (memory consistency, requiring synchronization) and the architect (cache coherency).
Simultaneous Multithreading

• Modern processors fail to utilize execution resources well.
• There is no single culprit.
• Attacking the problems one at a time (e.g., specific latency-tolerance solutions) always has limited effectiveness.
• However, a general latency-tolerance solution which can hide all sources of latency can have a large impact on performance.

Hardware Multithreading

Conventional Processor

Multithreaded Processor

Superscalar Execution with Fine-Grain Multithreading

Thread 1
Thread 2
Thread 3
Simultaneous Multithreading

Issue Slots

Thread 1
Thread 2
Thread 3
Thread 4
Thread 5

The Potential for SMT

Number of Threads

Throughput (Instructions per Cycle)

Simultaneous Multithreading
Fine-Grain Multithreading
Conventional Superscalar

Goals

We had three primary goals for this architecture:

1. Minimize the architectural impact on conventional superscalar design.
3. Achieve significant throughput gains with many threads.

A Conventional Superscalar Architecture

Fetch up to 8 instructions per cycle
Out-of-order, speculative execution
Issue 3 floating point, 6 integer instructions per cycle
**An SMT Architecture**

- Fetch up to 8 instructions per cycle
- Out-of-order, speculative execution
- Issue 3 floating point, 6 integer instructions per cycle

**Instruction Cache**

- Floating point instruction queue
- Integer instruction queue
- FP units
- Integer reg's
- Int/Id-store units

**Data Cache**

- FP reg's
- Int units

**Fetch Unit**

- PC
- Instruction Cache
- Decode
- Register Renaming

**Performance of the Naïve Design**

- Throughput (Instructions Per Cycle)

**Bottlenecks of the Baseline Architecture**

- Instruction queue full conditions (12-21% of cycles)
  - Lack of parallelism in the queue.
- Fetch throughput (4.2 instructions per cycle when queue not full)

**Improving Fetch Throughput**

- The fetch unit in an SMT architecture has two distinct advantages over a conventional architecture.
  1. Can fetch from multiple threads at once.
  2. Can choose which threads to fetch.
**Improved Fetch Performance**

- Fetching from 2 threads/cycle achieved most of the performance from multiple-thread fetch.
- Fetching from the thread(s) which have the fewest unissued instructions in-flight significantly increases parallelism and throughput.

**Improved Performance**

![Graph showing improved performance](image)

**This SMT Architecture, then:**

- Borrows heavily from conventional superscalar design.
- Minimizes the impact on single-thread performance
- Achieves significant throughput gains over the superscalar (2.5X, up to 5.4 IPC).

**Commercial SMT**

- Alpha 21464 (⊗)
- Clearwater Networks CNP810SP Network Services Processor
- Intel Pentium 4 “hyper-threading” processor.
- IBM Power 5 – 2 cores, 2 SMT threads/core
- Sun Niagara (2006) – 8 cores, 4 threads/core (SMT?)