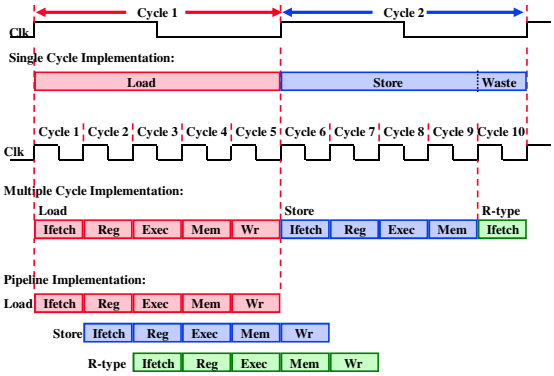
Pipelining, Superscalar, Multiprocessors

CPS 104

Admin

- Homework 6 due Wed
- Projects due Wed
- Final Tuesday April 28, 7pm-10pm
- Review (Jie will have one to work problems)

Single Cycle, Multiple Cycle, vs. Pipeline



Pipelining Summary

- Most modern processors use pipelining
- Pentium 4 has 24 (35) stage pipeline!
- Intel Core 2 duo has 14 stages
- Alpha 21164 has 7 stages
- Pipelining creates more headaches for exceptions, etc…
- Pipelining augmented with superscalar capabilities

Superscalar Processors

- Key idea: execute more than one instruction per cycle
- Pipelining exploits parallelism in the "stages" of instruction execution
- Superscalar exploits parallelism of independent instructions
- Example Code:

```
sub $2,  $1, $3
and $12, $3, $5
or  $13, $6, $2
add $3,  $3, $2
sw  $15, 100($2)
```

- Superscalar Execution

```
sub $2,  $1, $3        and $12, $3, $5
or  $13, $6, $2        add $3,  $3, $2
sw  $15, 100($2)
```


Superscalar Processors

- Key Challenge: Finding the independent instructions
  - Instruction level parallelism (ILP)
- Option 1: Compiler
  - Static scheduling (Alpha 21064, 21164; UltraSPARC I, II; Pentium)
- Option 2: Hardware
  - Dynamic Scheduling (Alpha 21264; PowerPC; Pentium Pro, 3, 4)
  - Out-of-order instruction processing

Instruction Level Parallelism

- Problems:
  - Program structure: branch every 4-8 instructions
  - Limited number of registers
- Static scheduling: compiler must find and move instructions from other basic blocks
- Dynamic scheduling: Hardware creates a big "window" to look for independent instructions
  - Must know branch directions before branch is executed!
  - Determines true dependencies.
- Example Code:

```
sub $2,  $1, $3
and $12, $3, $2
or  $2,  $6, $4
add $3,  $3, $2
sw  $15, 100($2)
```


Exposing Instruction Level Parallelism

- Branch prediction
  - Hardware can remember if branch was taken
  - Next time it sees the branch it uses this to predict outcome
- Register renaming
  - Indirection! The CS solution to almost everything
  - During decode, map register name to real register location
  - New location allocated when new value is written to reg.
- Example Code:

```
sub $2,  $1, $3     # writes $2 = $p1
and $12, $3, $2     # reads $p1
or  $2,  $6, $4     # writes $2 = $p3
add $3,  $3, $2     # reads $p3
sw  $15, 100($2)    # reads $p3
```


CPU design Summary

- Disadvantages of the Single Cycle Processor
  - Long cycle time
  - Cycle time is too long for all instructions except the Load
- Multiple Clock Cycle Processor:
  - Divide the instructions into smaller steps
  - Execute each step (instead of the entire instruction) in one cycle
- Pipeline Processor:
  - Natural enhancement of the multiple clock cycle processor
  - Each functional unit can only be used once per instruction
  - If a instruction is going to use a functional unit:
    - it must use it at the same stage as all other instructions
  - Pipeline Control:
    - Each stage's control signal depends ONLY on the instruction that is currently in that stage

Additional Notes

- All Modern CPUs use pipelines.
- Many CPUs have 8-12 pipeline stages.
- The latest generation processors (Pentium-4, PowerPC G4, SUN's UltraSPARC) use multiple pipelines to get higher speed (Superscalar design).
- The course: CPS220: Advanced Computer Architecture I covers Superscalar processors.
- Now, Parallel Architectures…

What is Parallel Computer Architecture?

- A Parallel Computer is a collection of processing elements that cooperate to solve large problems fast
  - how large a collection?
  - how powerful are the elements?
  - how does it scale up?
  - how do they cooperate and communicate?
  - how is data transmitted between processors?
  - what are the primitive abstractions?
  - how does it all translate to performance?

Parallel Computation: Why and Why Not?

- Pros
  - Performance
  - Cost-effectiveness (commodity parts)
  - Smooth upgrade path
  - Fault Tolerance
- Cons
  - Difficult to parallelize applications
  - Requires automatic parallelization or parallel program development
  - Software! AAHHHH!

Simple Problem

```
for i = 1 to N
        A[i] = (A[i] + B[i]) * C[i]
        sum = sum + A[i]
```

- Split the loops
  - Independent iterations

```
for i = 1 to N
        A[i] = (A[i] + B[i]) * C[i]
for i = 1 to N
        sum = sum + A[i]
```

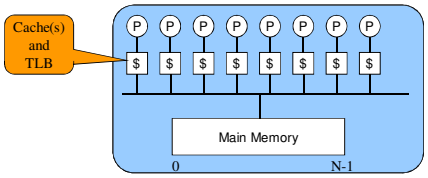

Parallel Programming

- Parallel software is the problem
- Need to get significant performance improvement
  - Otherwise, just use a faster uniprocessor, since it's easier!
- Difficulties
  - Partitioning
  - Coordination
  - Communications overhead

Amdahl's Law

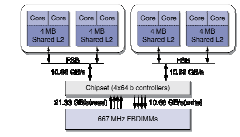
- Sequential part can limit speedup
- Example: 100 processors, 90× speedup?
  - $T_{new} = T_{parallelizable}/100 + T_{sequential}$
  - $$\text{Speedup} = \frac{1}{(1-F_{parallelizable}) + F_{parallelizable}/100} = 90$$
  - Solving: $F_{parallelizable} = 0.999$
- Need sequential part to be 0.1% of original time

Small Scale Shared Memory Multiprocessors

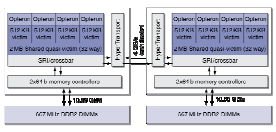


- Small number of processors connected to one shared memory
- Memory is equidistant from all processors (UMA)
- Kernel can run on any processor (symmetric MP)
- Intel dual/quad Pentium
- Multicore

Four Example Systems

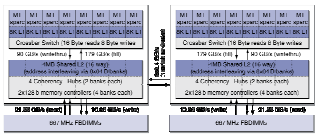


2 × quad-core
Intel Xeon e5345
(Clovertown)

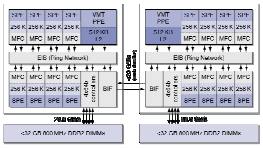


2 × quad-core
AMD Opteron X4 2356
(Barcelona)

Four Example Systems

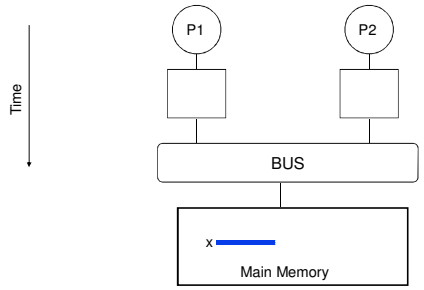


2 × oct-core
Sun UltraSPARC
T2 5140 (Niagara 2)

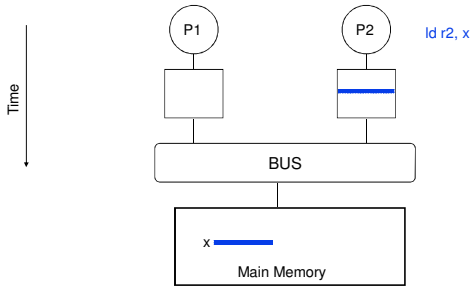


2 × oct-core
IBM Cell QS20

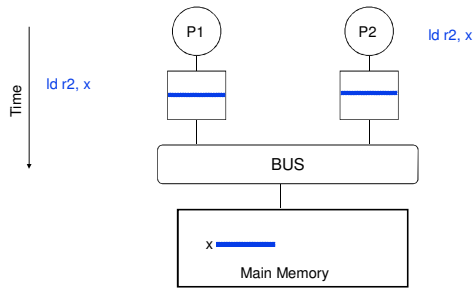
Cache Coherence Problem (Initial State)



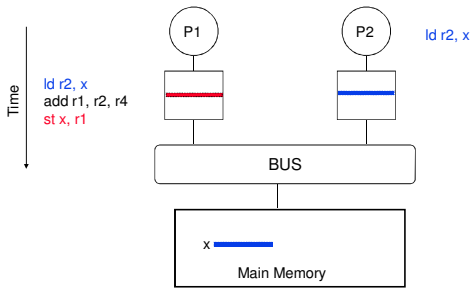
Cache Coherence Problem (Step 1)



Cache Coherence Problem (Step 2)

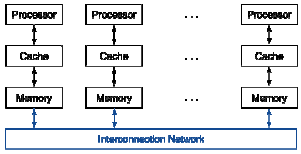


Cache Coherence Problem (Step 3)



Message Passing

- Each processor has private physical address space
- Hardware sends/receives messages between processors



Loosely Coupled Clusters

- Network of independent computers
  - Each has private memory and OS
  - Connected using I/O system
    - E.g., Ethernet/switch, Internet
- Suitable for applications with independent tasks
  - Web servers, databases, simulations, …
- High availability, scalable, affordable
- Problems
  - Administration cost (prefer virtual machines)
  - Low interconnect bandwidth
    - c.f. processor/memory bandwidth on an SMP

Grid Computing

- Separate computers interconnected by long-haul networks
  - E.g., Internet connections
  - Work units farmed out, results sent back
- Can make use of idle time on PCs
  - E.g., SETI@home, World Community Grid

Multithreading

- Performing multiple threads of execution in parallel
  - Replicate registers, PC, etc.
  - Fast switching between threads
- Fine-grain multithreading
  - Switch threads after each cycle
  - Interleave instruction execution
  - If one thread stalls, others are executed
- Coarse-grain multithreading
  - Only switch on long stall (e.g., L2-cache miss)
  - Simplifies hardware, but doesn't hide short stalls (eg, data hazards)