Agenda

• Quiz 2
• Go over PS2
Nehalem-EX Xeon 7560

- 2010
- octo-core
- 2.3 Billion transistors
- 45 nm
- 2.26 GHz
- 24 MB L3 cache (on-chip!!)
- 130 Watts (TDP)
- $3,692 (per die, when bought in 1k units)
  - (naturally slightly slower versions with smaller L3 caches go from $900–1400)
Quiz 2 on Monday!
- closed book
- will look just like pset

Turn in lab2 by Monday to receive credit (if not turned in already in)
Quiz 2 – Possible Topics

- Memory Hierarchies
  - caches
    - direct-mapped, set-associative
    - types of misses
    - AMAT
    - critical paths, physical design

- Virtual Memory
  - address translation, protection
  - TLBs
    - pages, page tables, page walking
  - aliasing

- software interplay
  - analyze software behavior
  - optimizing for memory/VM
Quiz 2 – How to study?

• fully understand pset #2
• go over past tests
• go over past psets
• Out of six points
• (but is worth same amount overall as pset#1)
PSet 2 – Problem 1
Can the 3rd and 4th stages be combined?
- yes, just insert NOP
- we aren’t changing state, so it’s okay

What about D$ tag check, write-back?
- technically, yes
- in reality, pretty difficult to get right
  • could write incorrect value into register file
  • need to devise way to undo this
  • e.g., can you handle LD r1, 0(r1)?
PSet 2 – Problem 2.A

Figure 2. R10000 block diagram (a) and pipeline timing diagram (b). The block diagram shows pipeline stages left to right to correspond to pipeline timing.

note: reg-read happens at end of cycle, reg-write happens at beginning of cycle
- Can the 3rd and 4th stages be combined?
  – yes, just insert NOP
  – we aren’t changing state, so it’s okay
- What about D$ tag check, write-back?
  – technically, yes
  – in reality, pretty difficult to get right
    - could write incorrect value into register file
    - need to devise way to undo this
    - e.g., can you handle LD r1, 0(r1)?
What about writes? Does this work?

- need to ensure we don’t write to memory if tag check misses

- solutions:
  - add fourth stage (but then ST,LD structural hazard)
  - delayed write-buffer (much better solution, but now loads must also check the write-buffer)
PSet 2 – Problem 2.C

- Does this work for set-associative instruction cache?
  - cycle time hit, improved miss rate
  - no, tag check required to get data out of the correct way

- Does this work for set-associative data cache?
  - yes, we don’t use data until after tag check
  - (not that this is a good idea. in particular, you can’t begin to drive data out until tag check completed)
• Branches resolved in Execute...
• What is the branch delay?
  – 3 cycles
Bypassing speculated loads...
How can an interlock in Decode handle this?
  – stall if dependent on load in EXE, DAD, DAA

How many cycles is the load–use delay? (assuming hit)
  – 3 cycles
• Bypassing speculated loads...
• Instead let’s speculatively execute. What state do we need to store to restart the pipeline on a misspeculation?
  – track the PC of the bad load
  – restart pipeline when data comes back
  – (it’s basically an exception!)
• Load delay is now how many cycles?
  – 2 cycles
The matrix $A$ is stored contiguously in memory in row-major order. Row major order means that elements in the same row of the matrix are adjacent in memory as shown in the following memory layout:

$$A[i][j] \text{ resides in memory location } [4 \times (64 \times i + j)]$$

Memory Location:

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>252</th>
<th>256</th>
<th>$4 \times (64 \times 127 + 63)$</th>
</tr>
</thead>
</table>
### Loop A

```plaintext
sum = 0;
for (i = 0; i < 128; i++)
    for (j = 0; j < 64; j++)
        sum += A[i][j];
```

### Loop B

```plaintext
sum = 0;
for (j = 0; j < 64; j++)
    for (i = 0; i < 128; i++)
        sum += A[i][j];
```

### Table

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>252</td>
<td>256</td>
<td>4*(64*127+63)</td>
</tr>
</tbody>
</table>

- 4KB direct-mapped, 8-word lines (32B)
- How many lines?
  - 128 lines
- 64*128 = 8192 elements, or 32768 bytes (32KB)
- Loop A misses:
  - **1024** (notice the 4KB capacity is irrelevant)
- Loop B misses:
  - **8192** (100% misses, cache is too small to save us)
### PSet 2 – Problem 3.B

<table>
<thead>
<tr>
<th>Loop A</th>
<th>Loop B</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sum = 0;</code></td>
<td><code>sum = 0;</code></td>
</tr>
<tr>
<td><code>for (i = 0; i &lt; 128; i++)</code></td>
<td><code>for (j = 0; j &lt; 64; j++)</code></td>
</tr>
<tr>
<td><code>for (j = 0; j &lt; 64; j++)</code></td>
<td><code>for (i = 0; i &lt; 128; i++)</code></td>
</tr>
<tr>
<td><code>sum += A[i][j];</code></td>
<td><code>sum += A[i][j];</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>252</th>
<th>256</th>
<th>4*(64*127+63)</th>
</tr>
</thead>
</table>

- 4KB direct-mapped, 8-word lines (32B), 128 lines
- 64*128 = 8192 elements, or 32768 bytes (32KB)
- Cache size required?
  - 1 cache line
- Cache size required?
  - 1024 cache lines
Loop A

sum = 0;
for (i = 0; i < 128; i++)
    for (j = 0; j < 64; j++)
        sum += A[i][j];

Loop B

sum = 0;
for (j = 0; j < 64; j++)
    for (i = 0; i < 128; i++)
        sum += A[i][j];

- 4KB fully-assoc, 8-word lines (32B), 128 lines, FIFO
- 64*128 = 8192 elements, or 32768 bytes (32KB)
- Loop A misses?
  - 1024 (still only compulsory, can’t fix this)
- Loop B misses?
  - 1024 (only compulsory, we can fit an entire column of matrix A (8x128 elements)
PSet 2 – Problem 4–Critical Path?

tag decoder -> tag read -> comparator -> 2-in AND -> buffer driver -> data output driver

240+330+500+50+200+300 = 1620ps, or 1.62 ns

Friday, March 4, 2011
PSet 2 – Problem 4

tag check = 1320ps

final answer (through utags): 1350ps, or 1.35 ns.

data up to data output
driver = 930ps

microtag to buffer
driver = 1050ps
• AMAT?
• 95% hit rate
• miss penalty = 20ns
  – AMAT = hit_time + miss_rate*miss_penalty
  – AMAT = hit_time + (0.05)*(20ns)
  – AMAT = hit_time + 1ns
  – AMAT_old = 1.62ns + 1ns = 2.62ns
  – AMAT_new = 1.35ns + 1ns = 2.35ns
• new constraint:
  – all microtags in set must be unique!
• What type of miss does this affect?
  – conflict miss
• How does miss_rate compare to 4-way?
  – worse... more conflict misses
• How does miss_rate compare to direct-mapped?
  – at least as good (better)... similiar microtags in a given set would already alias and kick each other out
Problem:
- an alias will see a HIT (physical tag already present), **BUT** it will not see its microtag
-> data_out will float and the CPU gets incorrect data
PSet 2 – Problem 4.E – Fix?
Solution
- if tag check hits, but utag misses, then you have an alias
- evict first alias
• 2x associativity (halves # of sets)
  – compulsory misses
    • no effect
  – conflict misses
    • reduces
    • more places to put in a set
  – capacity misses
    • no effect
    • capacity unchanged
• halving line size (halves capacity)
  – compulsory misses
    • increases
    • less “prefetching” data on same line
    • less able to exploit spatial locality
  – conflict misses
    • no effect
  – capacity misses
    • increase
    • capacity has been halved
double number of sets (halves associativity)
  - compulsory misses
    • no effect
  - conflict misses
    • increase
    • less associativity
  - capacity misses
    • no effect
• adding prefetching
  – compulsory misses
    • decreases
    • good prefetcher brings in data before we need it
  – conflict misses
    • increase
    • prefetched data could pollute the cache
  – capacity misses
    • increase
    • prefetched data could pollute the cache
• double associativity (halves # of sets)
  – hit time
    • increases
    • sets decrease → larger tags
    • more tags to check, more ways to mux out
  – miss rate
    • decrease
    • less conflict misses
  – miss penalty
    • no effect
    • dominated by the other parts of the memory hierarchy
PSet 2 – Problem 6

• halving line size (halves capacity)
  – hit time
    • decreases
    • cache becomes physically smaller (x2), so quicker access probably dominates larger tag check (by 1 bit)
  – miss rate
    • increases
    • smaller capacity, less spatial locality
  – miss penalty
    • decreases
    • uses less bandwidth out to memory
double # of sets (halves associativity)
  - hit time
    - decreases
    - less logic getting in the way, smaller tags
  - miss rate
    - increases
    - more conflict misses
  - miss penalty
    - no effect
    - this is dominated by outer memory heirarchy
• adding prefetching
  – hit time
    • no effect
    • isn’t on critical path for the hit
  – miss rate
    • decrease
    • this is the purpose of a prefetcher!
    • bring in data a head of time
  – miss penalty
    • decreases
    • prefetch could be inflight when miss occurs
Questions?