Spam Emails on the Rise

New studies show spam emails have increased by 50% worldwide. Worse yet, much of the spamming is being done by criminals, so laws may not help.

Outline

• Last time was about how to exploit parallelism from the software point of view

• Today is about how to implement this in hardware

• Some parallel hardware techniques

• A couple current examples

• Won’t cover out-of-order execution, since too complicated
Introduction

- Given many threads (somehow generated by software), how do we implement this in hardware?

- Recall the performance equation:
  
  \[
  \text{Execution Time} = \text{(Inst. Count)} \times \text{(CPI)} \times \text{(Cycle Time)}
  \]

- Hardware Parallelism improves
  
  - **Instruction Count** - If the equation is applied to each CPU, each CPU needs to do less
  
  - **CPI** - If the equation is applied to system as a whole, more is done per cycle
  
  - **Cycle Time** - Will probably be made worse in process
Disclaimers

• Please don’t let today’s material confuse what you have already learned about CPU’s and pipelining

• When *programmer* is mentioned today, it means whoever is generating the assembly code (so it is probably a compiler)

• Many of the concepts described today are *difficult* to implement, so if it sounds easy, think of possible hazards
Superscalar

• Add more functional units or pipelines to CPU

• Directly reduces CPI by doing more per cycle

• Consider what if we:
  • Added another ALU
  • Added 2 more read ports to the RegFile
  • Added 1 more write port to the RegFile
Simple Superscalar MIPS CPU

- Can now do 2 instruction in 1 cycle!
Simple Superscalar MIPS CPU (cont.)

• Considerations
  • ISA now has to be changed
  • Forwarding for pipelining now harder

• Limitations
  • Programmer must explicitly generate parallel code
  • Improvement only if other instructions can fill slots
  • Doesn’t scale well
Single Instruction Multiple Data (SIMD)

- Often done in a vector form, so all data has the same operation applied to it

- Example: AltiVec (like SSE)
  - 128bit registers can hold:
    - 4 floats, 4 ints, 8 shorts, 16 chars, etc.
  - Processes whole vector
Superscalar in Practice

• ISA’s have extensions for these vector operations

• One thread, that has parallelism internally

• Performance improvement depends on program and programmer being able to fully utilize all slots

• Can be parts other than ALU (like load)

• Usefulness will be more apparent when combined with other parallel techniques
Thread Review

- A *Thread* is a single stream of instructions
  - It has its own registers, PC, etc.
  - Threads from the same process operate in the same virtual address space
  - Are an easy way to describe/think about parallelism

- A single CPU can execute many threads by *Time Division Multiplexing*
Multithreading

• Multithreading is running multiple threads through the same hardware

• Could we do *Time Division Multiplexing* better in hardware?

• Consider if we gave the OS the abstraction of having 4 physical CPU’s that share memory and each execute one thread, but we did it all on 1 physical CPU?
Static Multithreading Example

Introduced in 1964 by Seymour Cray

Appears to be 4 CPU’s at 1/4 clock
Static Multithreading Example Analyzed

• Results:
  • 4 Threads running in hardware
  • Pipeline hazards reduced
    ▪ No more need to forward
    ▪ No control issues
    ▪ Less structural hazards
  • Depends on being able to fully generate 4 threads evenly
    ▪ Example if 1 Thread does 75% of the work
      – Utilization = (% time run)(% work done)
      = (.25)(.75) + (.75)(.25) = .375
      = 37.5%
Dynamic Multithreading

• Adds flexibility in choosing time to switch thread

• Simultaneous Multithreading (SMT)
  • Called Hyperthreading by Intel
  • Run multiple threads at the same time
  • Just allocate functional units when available
  • Superscalar helps with this
### Dynamic Multithreading Example

#### One thread, 8 units

<table>
<thead>
<tr>
<th>Cycle</th>
<th>M</th>
<th>M</th>
<th>FX</th>
<th>FX</th>
<th>FP</th>
<th>FP</th>
<th>BR</th>
<th>CC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Two threads, 8 units

<table>
<thead>
<tr>
<th>Cycle</th>
<th>M</th>
<th>M</th>
<th>FX</th>
<th>FX</th>
<th>FP</th>
<th>FP</th>
<th>BR</th>
<th>CC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Multicore

• Put multiple CPU’s on the same die

• Why is this better than multiple dies?
  • Smaller
  • Cheaper
  • Closer, so lower inter-processor latency
  • Can share a L2 Cache (details)
  • Less power

• Cost of multicore: complexity and slower single-thread execution
Multicore Example (IBM Power5)

Core #1

Core #2

Shared Stuff
Administrivia

• Proj4 due tonight at 11:59pm

• Proj1 - Check newsgroup for posting about Proj1 regrades, you *may* want one

• Lab tomorrow will have 2 surveys

• Come to class Friday for the HKN course survey
## Upcoming Calendar

<table>
<thead>
<tr>
<th>Week #</th>
<th>Mon</th>
<th>Wed</th>
<th>Thu Lab</th>
<th>Fri</th>
</tr>
</thead>
<tbody>
<tr>
<td>#15</td>
<td><strong>Parallel Computing in Software</strong></td>
<td><strong>Parallel Computing in Hardware (Scott)</strong></td>
<td>I/O Networking &amp; 61C Feedback Survey</td>
<td>LAST CLASS</td>
</tr>
<tr>
<td></td>
<td><strong>This Week</strong></td>
<td></td>
<td></td>
<td>Summary, Review, &amp; HKN Evals</td>
</tr>
<tr>
<td>#16</td>
<td></td>
<td></td>
<td><strong>FINAL EXAM</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Sun 2pm] Review 10 Evans</td>
<td></td>
<td><strong>THU 12-14 @ 12:30pm-3:30pm 234 Hearst Gym</strong></td>
<td></td>
</tr>
</tbody>
</table>

### Final exam

Same rules as Midterm, except you get 2 double-sided handwritten review sheets (1 from your midterm, 1 new one) + green sheet [Don’t bring backpacks]
Real World Example 1: Cell Processor

• Multicore, and more....
Real World Example 1: Cell Processor

- 9 Cores (1PPE, 8SPE) at 3.2GHz

- **Power Processing Element (PPE)**
  - Supervises all activities, allocates work
  - Is multithreaded (2 threads)

- **Synergystic Processing Element (SPE)**
  - Where work gets done
  - Very Superscalar
  - No Cache, only **Local Store**
Real World Example 1: Cell Processor

• **Great** for other multimedia applications such as HDTV, cameras, etc...

• Really **dependent** on programmer use SPE's and Local Store to get the most out of it
Real World Example 2: Niagara Processor

- **Multithreaded and Multicore**
- 32 Threads (8 cores, 4 threads each) at 1.2GHz
- Designed for **low power**
- Has simpler pipelines to fit more on
- Maximizes thread level parallelism
- Project Blackbox
Real World Example 2: Niagara Processor

- Each thread runs slower (1.2GHz), and there is less number crunching ability (no FP unit), but tons of threads

- This is great for webservers, where there are typically many simple requests, and many data stalls

- Can beat “faster” and more expensive CPU’s, while using less power
1. The **majority of PS3’s processing power** comes from the **Cell** processor

2. A computer that **has max utilization** can get more done multithreaded

3. Current multicore techniques **can scale well** to many (32+) cores
Peer Instruction Answer

1. All PS3 is 2.18TFLOPS, Cell is only 204GFLOPS (GPU can do a lot…) FALSE

2. No more functional power FALSE

3. Share memory and caches huge barrier. Why Cell has Local Store FALSE

1. The **majority of PS3’s** processing power comes from the **Cell** processor

2. A computer that **has max utilization** can get more done multithreaded

3. Current multicore techniques **can scale well** to many (32+) cores
Summary

• **Superscalar**: More functional units
• **Multithread**: Multiple threads executing on same CPU
• **Multicore**: Multiple CPU’s on the same die

• The gains from all these parallel hardware techniques relies **heavily** on the programmer being able to map their task well to multiple threads

• Hit up **CS150, CS152, CS162** and **wikipedia** for more info