THREAD LEVEL PARALLELISM

Mahdi Nazm Bojnordi
Assistant Professor
School of Computing
University of Utah
Overview

☐ Announcement
  - Homework 7 is due on 12/02

☐ This lecture
  - Thread level parallelism (TLP)
  - Parallel architectures for exploiting TLP
    - Hardware multithreading
    - Symmetric multiprocessors
    - Chip multiprocessing
Recall: Flynn’s Taxonomy

- Forms of computer architectures

- Instruction Stream
  - Single
    - Single-Instruction, Single Data (SISD) uniprocessors
  - Multiple
    - Multiple-Instruction, Single Data (MISD) systolic arrays

- Single Instruction, Multiple Data (SIMD) vector processors

- Multiple Instruction, Multiple Data (MIMD) multiprocessors
Basics of Threads

- **Thread** is a single sequential flow of control within a program including instructions and state
  - Register state is called **thread context**

- A program may be single- or multi-threaded
  - Single-threaded program can handle one task at any time

- **Multitasking** is performed by modern operating systems to load the context of a new thread while the old thread’s context is written back to memory
Thread Level Parallelism (TLP)

- Users prefer to execute multiple applications
  - Piping applications in Linux
    - `gunzip -c foo.gz | grep bar | perl some-script.pl`
  - Your favorite applications while working in office
    - Music player, web browser, terminal, etc.

- Many applications are amenable to parallelism
  - Explicitly multi-threaded programs
    - Pthreaded applications
  - Parallel languages and libraries
    - Java, C#, OpenMP
Thread Level Parallel Architectures

- Architectures for exploiting thread-level parallelism

**Hardware Multithreading**

- Multiple threads run on the same processor pipeline
- Multithreading levels
  - Coarse grained multithreading (CGMT)
  - Fine grained multithreading (FGMT)
  - Simultaneous multithreading (SMT)

**Multiprocessing**

- Different threads run on different processors
- Two general types
  - Symmetric multiprocessors (SMP)
    - Single CPU per chip
  - Chip Multiprocessors (CMP)
    - Multiple CPUs per chip
Observation: CPU become idle due to latency of memory operations, dependent instructions, and branch resolution

Key idea: utilize idle resources to improve performance
- Support multiple thread contexts in a single processor
- Exploit thread level parallelism

Challenge: the energy and performance costs of context switching
Coarse Grained Multithreading

- Single thread runs until a costly stall—e.g. last level cache miss
- Another thread starts during stall for first
  - Pipeline fill time requires several cycles!
- At any time, only one thread is in the pipeline
- Does not cover short stalls
- Needs hardware support
  - PC and register file for each thread
Coarse Grained Multithreading

- Superscalar vs. CGMT

<table>
<thead>
<tr>
<th>Conventional Superscalar</th>
<th>Coarse Grained Multithreading</th>
</tr>
</thead>
<tbody>
<tr>
<td>FU1</td>
<td>FUs</td>
</tr>
<tr>
<td>FU2</td>
<td>FUs</td>
</tr>
<tr>
<td>FU3</td>
<td>FUs</td>
</tr>
<tr>
<td>FU4</td>
<td>FUs</td>
</tr>
</tbody>
</table>

FU: Functional Unit

- Conventional Superscalar: Each FU is used for a single instruction.
- CGMT: Multiple FUs can execute instructions simultaneously.
Fine Grain Multithreading

- Two or more threads interleave instructions
  - Round-robin fashion
  - Skip stalled threads
- Needs hardware support
  - Separate PC and register file for each thread
  - Hardware to control alternating pattern
- Naturally hides delays
  - Data hazards, Cache misses
  - Pipeline runs with rare stalls
- Does not make full use of multi-issue architecture
Fine Grained Multithreading

- **CGMT vs. FGMT**

<table>
<thead>
<tr>
<th>FU1</th>
<th>FU2</th>
<th>FU3</th>
<th>FU4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FU1</th>
<th>FU2</th>
<th>FU3</th>
<th>FU4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FU1</th>
<th>FU2</th>
<th>FU3</th>
<th>FU4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FU1</th>
<th>FU2</th>
<th>FU3</th>
<th>FU4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Coarse Grained Multithreading

Fine Grained Multithreading
Simultaneous Multithreading

- Instructions from multiple threads issued on same cycle
  - Uses register renaming and dynamic scheduling facility of multi-issue architecture

- Needs more hardware support
  - Register files, PC’s for each thread
  - Temporary result registers before commit
  - Support to sort out which threads get results from which instructions

- Maximizes utilization of execution units
Simultaneous Multithreading

- FGMT vs. SMT

Fine Grained Multithreading

Simultaneous Multithreading
Symmetric Multiprocessors

- Multiple CPU chips share the same memory
- From the OS’s point of view
  - All of the CPUs have equal compute capabilities
  - The main memory is equally accessible by the CPU chips
- OS runs every thread on a CPU
- Every CPU has its own power distribution and cooling system
Chip Multiprocessors

- Can be viewed as a simple SMP on single chip
- CPUs are now called cores
  - One thread per core
- Shared higher level caches
  - Typically the last level
  - Lower latency
  - Improved bandwidth
- Not necessarily homogenous cores!
Why Chip Multiprocessing?

- CMP exploits parallelism at lower costs than SMP
  - A single interface to the main memory
  - Only one CPU socket is required on the motherboard
- CMP requires less off-chip communication
  - Lower power and energy consumption
  - Better performance due to improved AMAT
- CMP better employs the additional transistors that are made available based on the Moore’s law
  - More cores rather than more complicated pipelines
Efficiency of Chip Multiprocessing

- Ideally, \( n \) cores provide \( nx \) performance
- Example: design an ideal dual-processor
  - **Goal**: provide the same performance as uniprocessor

<table>
<thead>
<tr>
<th></th>
<th>Uniprocessor</th>
<th>Dual-processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>Execution Time</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Dynamic Power</td>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>Dynamic Energy</td>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>Energy Efficiency</td>
<td>1</td>
<td>?</td>
</tr>
</tbody>
</table>
Efficiency of Chip Multiprocessing

- Ideally, \( n \) cores provide \( nx \) performance
- Example: design an ideal dual-processor
  - **Goal**: provide the same performance as uniprocessor

<table>
<thead>
<tr>
<th></th>
<th>Uniprocessor</th>
<th>Dual-processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>Execution Time</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Dynamic Power</td>
<td>1</td>
<td>2x0.125</td>
</tr>
<tr>
<td>Dynamic Energy</td>
<td>1</td>
<td>2x0.125</td>
</tr>
<tr>
<td>Energy Efficiency</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

\[ f \propto V \& P \propto V^3 \rightarrow V_{\text{dual}} = 0.5V_{\text{uni}} \rightarrow P_{\text{dual}} = 2 \times 0.125 P_{\text{uni}} \]