Parallel Architectures
Lecture 2

CS149
11:00-12:15 TT
Herrin T195
Introduction

• Parallel programming for performance
  – Management of resources
  – Memory and control

• How are resources organized?
  – Characteristics of different classes of parallel architecture
  – Parallel performance implications
Today’s Topics

• Architecture principles
• Application parallelism
• Advanced pipelining
• Shared memory multiprocessor design
• Chip multiprocessors
• Graphics processing unit (GPU)
• Distributed memory multiprocessors
• Parallel performance analysis
Generic Uniprocessor Computer

- Processor
- Cache
- Memory
- I/O
- SRAM
- DRAM
1st Computer Architecture Principle

• Power limits performance
  - Cooling ability limits power density
  - Determines battery life (mobile phone)
  - Running cost and computing capacity of datacenter (Google)
  - End of voltage scaling ⇒ no power improvements with new technology nodes
**2nd Computer Architecture Principle**

- Smaller is faster $\Leftrightarrow$ smaller is lower power
  - Computation is cheaper than memory access
  - Small caches have lower access times than large caches
  - Cache access is faster than DRAM access

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time ratios</th>
<th>Power ratios</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Cache access</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>DRAM access</td>
<td>&gt;100</td>
<td>200</td>
</tr>
</tbody>
</table>
• Use transistor bounty from Moore’s Law to improve application performance by exploiting parallelism
Where is Application Parallelism?

- Process
- Task
- Loop
- Instruction

Grain Size

Parallel

Sequential

Programming Model

very fine  fine  medium  coarse  very coarse
1990s: Exploiting Instruction Level Parallelism (ILP)

Process
Task
Loop
Instruction

Grain Size

Advanced pipelining
Multiple instruction issue
Dynamic execution
Deep pipelines

Profs. Aiken/Olukotun  CS 149  Lecture 2
1990s: Exploiting Coarse Grain Parallelism

- Instruction
- Loop
- Task
- Process

Grain Size

very fine  fine  medium  coarse  very coarse

Advanced pipelining

Shared Memory Multiprocessor (SMP)
Sun Enterprise
Shared Memory

- SMP: shared memory multiprocessor
  - Hardware provides single physical address space for all processors
  - Synchronize shared variables using locks
  - Memory access time
    - UMA (uniform) vs. NUMA (nonuniform)
Shared Memory

- Memory access time
  - UMA (uniform) vs. NUMA (nonuniform)
  - Local vs. remote
  - Why build NUMA?
Cache Coherence Problem

- Want to cache shared data to reduce access time
- Problem
  - Two caches with inconsistent values

Diagram:

- Processors and caches connected via an interconnection network.
- Caches communicate with the processors above and memory below.
- Each cache contains a value: sum=1, sum=10, etc.

Profs. Aiken/Olukotun    CS 149 Lecture 2
Snoopy Cache Coherence

- Solution
  - Have caches “snoop” on other caches’ reads and writes
  - Writes: Invalidate other cached copies
  - Reads: Get latest version from other cache, if it has a “dirty” copy
  - Coherence kept on cache lines (multiple words)

Profs. Aiken/Olukotun    CS 149  Lecture 2
**Snoopy Cache Coherence 2**

- **Solution**
  - Have caches "snoop" on other caches' reads and writes
  - Writes: Invalidate other cached copies
  - Reads: Get latest version from other cache, if it has a "dirty" copy
  - Coherence kept on cache lines (multiple words)
Snoopy Cache Coherence 3

- Solution
  - Have caches “snoop” on other caches’ reads and writes
  - Writes: Invalidate other cached copies
  - Reads: Get latest version from other cache, if it has a “dirty” copy
    Coherence kept on cache lines (multiple words)
Snoopy Cache Coherence 4

- Solution
  - Have caches “snoop” on other caches’ reads and writes
  - Writes: Invalidate other cached copies
  - Reads: Get latest version from other cache, if it has a “dirty” copy

- Coherence kept on cache lines (multiple words)
Summary: Cache Coherence

- Snoopy cache coherence
  - Requires a broadcast network

- Write access to shared state creates implicit communication
  - Invalidations
  - Leads to cache misses (100s of cycles and more power)
  - Increases memory access time (more overhead)

- Important: coherence kept at cache line granularity
  - Can get communication when you don’t expect it ⇒ false sharing
Distributed Shared Memory

- Global address space ≠ cache coherence
  - Can’t cache shared data
- Cache coherent NUMA (CC-NUMA)
Synchronization Primitives

• Implementing locks
  - Need an “atomic” (indivisible) read and write instruction
  - E.g. “test-and-set” or “compare-and-swap”

• Modern CPUs use “load-linked” (LL) and “store-conditional” (SC) instruction pair
  - LL: load and mark cache line
  - SC: store and return true if cache line is still valid
  - If SC succeeds ⇒ no intervening write between LL and SC
  ⇒ LL and SC are atomic
1990s:
Exploiting ILP and Very Coarse Grain Parallelism

- Process
- Task
- Loop
- Instruction

Parallelism

Grain Size

very fine  fine  medium  coarse  very coarse

Advanced pipelining

Shared Memory Multiprocessor (SMP)
Sun Enterprise

Profs. Aiken/Olukotun   CS 149 Lecture 2
By End of 90s:
ILP Loses Steam and GHz Limited by Power
Thread-Level Parallelism

Chip Multiprocessor or Multicore

Grain Size

Process
Task
Loop
Instruction

very fine  fine  medium  coarse  very coarse

Chip Multiprocessor (CMP)
IBM Power4

Profs. Aiken/Olukotun   CS 149 Lecture 2
Exploit Medium and Fine Grain Thread-Level Parallelism

- Communication bandwidth and Latency improvements on chip
  - 10x-20x better latency
  - 100x better bandwidth
- Impact on parallel programming model
  - Lower overheads
  - Don’t worry as much about locality
  - Exploit fine-grained parallelism

Chip Multiprocessor (CMP)
IBM Power4

Grain Size

Profs. Aiken/Olukotun  CS 149 Lecture 2
Shared-Cache CMP (Multicore)

- Second generation
  - 2–4 cores
  - Reduce GHz
- IBM Power 4 & 5
- AMD Barcelona
- Intel Core-2 Duo
Multithreading

- Performing multiple threads of execution on a single processor
  - Context: replicate registers, PC, etc.
  - Fast switching between threads
  - Hide memory and function unit (FU) latency and keep pipeline busy ⇒ needs more memory bandwidth

- 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 (e.g., FU latency)
Simultaneous Multithreading

• In multiple-issue dynamically scheduled processor
  - Schedule instructions from multiple threads
  - Instructions from independent threads execute when function units are available

• Example: Intel Pentium-4 HT
  - Two threads: duplicated registers, shared function units and caches
  - Hyperthreading!
Multithreading Example
Shared-Cache Multithreaded CMP

- Third generation
  - Thread optimized
  - 8–16 cores
  - Simple cores
  - Lower power
  - “Many core”
- Sun Niagara
- Intel Larabee (?)
### General Purpose Multicore

<table>
<thead>
<tr>
<th>Year</th>
<th>Microprocessor</th>
<th>Proc/chip</th>
<th>Thread/proc</th>
<th>Thread/chip</th>
</tr>
</thead>
<tbody>
<tr>
<td>2004</td>
<td>IBM Power 5</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2005</td>
<td>Sun Niagara 1</td>
<td>8</td>
<td>4</td>
<td>32</td>
</tr>
<tr>
<td>2005</td>
<td>AMD Opteron</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2006</td>
<td>Intel Woodcrest</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2006</td>
<td>Intel Cloverton</td>
<td>4</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2007</td>
<td>AMD Barcelona</td>
<td>4</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2007</td>
<td>Sun Niagara 2</td>
<td>8</td>
<td>8</td>
<td>64</td>
</tr>
<tr>
<td>2008</td>
<td>Intel Nehalem</td>
<td>4</td>
<td>2</td>
<td>8</td>
</tr>
</tbody>
</table>

- IBM and Sun went multi-core first on the server side
- AMD/Intel now in core count race for laptops, desktops, and servers

**Cores are the new GHz!**

Profs. Aiken/Olukotun  CS 149  Lecture 2
The First Intel® Core™ Microarchitecture (Nehalem) Processor

Memory Controller

Core

Core

Queue

Core

Core

Shared L3 Cache

QPI: Intel® QuickPath Interconnect (Intel® QPI)

Profs. Aiken/Olukotun  CS 149  Lecture 2
Parallel Hardware Landscape: Instruction and Data Streams

- Flynn’s classification
  - Hardware view of dimensions of memory and control

<table>
<thead>
<tr>
<th>Instruction Streams</th>
<th>Data Streams</th>
<th>Data Streams</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>Single</td>
<td>Multiple</td>
</tr>
<tr>
<td><strong>SISD:</strong></td>
<td>Intel Pentium 4</td>
<td><strong>SIMD:</strong> SSE instructions of x86</td>
</tr>
<tr>
<td>Multiple</td>
<td><strong>MISD:</strong> No examples today</td>
<td><strong>MIMD:</strong> Intel Nehalem</td>
</tr>
</tbody>
</table>
SIMD

• Function Unit: Operate elementwise on vectors of data
  - E.g., MMX and SSE instructions in x86
    • Multiple data elements in 128-bit wide registers
    • \( A + B = A_0+B_0, A_1+B_1, A_2+B_2, A_3+B_3 \)

• Multiple Processors: All processors execute the same instruction at the same time
  - Each with different data address, etc.

• Simplifies synchronization
• Reduced instruction control hardware
• Works best for highly data-parallel applications
  - Graphics processing unit (GPU)
NVIDIA Tesla GPU

- 16 SM x 16 FLOP/SM = 256 FLOP/cycle @ 1.5 GHz

- • SIMD
  - 8 wide
  - • Highly multithreaded
  - • 24 x 32 = 768 contexts
Distributed Memory or Message Passing

- Each processor has a private physical address space
  - May be cheaper than shared memory
  - Forces application data to be partitioned

- Communication
  - Software sends/receives messages between processor
  - Synchronization is implicit in messages
Loosely Coupled Clusters

- Network of independent computers
  - Each has private memory and OS
  - Connected using I/O system
    - E.g., Ethernet/switch

- Suitable for applications with independent or very coarse granularity tasks
  - Low interconnect bandwidth
  - Web servers and scientific simulations

- High availability, scalable, affordable

- Massively Parallel Processor (MPP)
  - Supercomputer
  - Commodity processors
  - Much higher performance network ⇒ $$$$
Summary: Key Multiprocessor Questions

• How do parallel processors share data?
  - single address space: Symmetric MP (SMP) vs. NonUniform Memory Architecture (NUMA)
  - message passing: clusters, massively parallel processors (MPP)

• How do parallel processors coordinate?
  - synchronization (locks, semaphores)
  - built into send / receive primitives

• How are the processors interconnected?
Parallel Performance: Amdahl’s Law

• Sequential (overhead) component can limit speedup

• Example: 100 processors, 90× speedup?
  – \[ T_{\text{new}} = T_{\text{parallelizable}} / 100 + T_{\text{sequential}} \]
  
  – Speedup = \[ \frac{1}{(1 - F_{\text{parallelizable}}) + F_{\text{parallelizable}} / 100} \] = 90
  
  – Solving: \( F_{\text{parallelizable}} = 0.999 \)

• Need sequential part to be 0.1% of original time
Performance Scaling Example

- Workload: sum of 10 scalars, and $10 \times 10$ matrix sum
  - Speed up from 10 to 100 processors
- Single processor: $\text{Time} = (10 + 100) \times t_{\text{add}}$
- 10 processors
  - $\text{Time} = 10 \times t_{\text{add}} + \frac{100}{10} \times t_{\text{add}} = 20 \times t_{\text{add}}$
  - Speedup = $\frac{110}{20} = 5.5$ (55% of potential)
- 100 processors
  - $\text{Time} = 10 \times t_{\text{add}} + \frac{100}{100} \times t_{\text{add}} = 11 \times t_{\text{add}}$
  - Speedup = $\frac{110}{11} = 10$ (10% of potential)
- Assumes load can be balanced across processors
Performance Scaling Example (cont)

• What if matrix size is 100 × 100?
• Single processor: Time = (10 + 10000) × t_{add}
• 10 processors
  - Time = 10 × t_{add} + 10000/10 × t_{add} = 1010 × t_{add}
  - Speedup = 10010/1010 = 9.9 (99% of potential)
• 100 processors
  - Time = 10 × t_{add} + 10000/100 × t_{add} = 110 × t_{add}
  - Speedup = 10010/110 = 91 (91% of potential)
• Assuming load balanced
Strong vs Weak Scaling

• Strong scaling: problem size fixed
  – As in example
  – Faster time to solution

• Weak scaling: problem size proportional to number of processors
  – 10 processors, 10 × 10 matrix
    • Time = 20 × t_{add}
  – 100 processors, 32 × 32 matrix
    • Time = 10 × t_{add} + 1000/100 × t_{add} = 20 × t_{add}
  – Constant performance in this example
Concluding Remarks

• Goal: higher performance by using multiple processors
  - Need to target 100s of threads/contexts in near future
  - Need strong scaling

• Difficulties
  - Developing parallel software (CS149)
  - Devising appropriate architectures (CS315a)

• Reasons for optimism
  - Changing software and application environment
    • End of the free lunch
  - Chip multiprocessors and GPUs with lower latency, higher bandwidth interconnect