Introduction

CSCI 4850/5850 High-Performance Computing

Spring 2018

Tae-Hyuk (Ted) Ahn
Department of Computer Science
Program of Bioinformatics and Computational Biology
Saint Louis University
What is Parallel Computing?

- Parallel computing is the use of multiple processing entities in combination to solve a single problem.
- It is not always obvious that a parallel algorithm has benefits, unless we want to do things …
  - faster: doing the same amount of work in less time
  - bigger: doing more work in the same amount of time
- Both of these reasons can be argued to produce better results, which is the only meaningful outcome of program parallelization.
Levels of Parallelism of Processors

- Microprocessors enhance performance of a sequential program by extracting parallelism from an instruction stream (called instruction-level parallelism).
  - Instruction-level parallelism (ILP), simultaneous execution of multiple instructions. Good for vector operations.

- Multiprocessors enhance performance of an explicitly parallel program by running multiple threads in parallel (called thread-level parallelism).
  - Thread-level parallelism (TLP), concurrent execution of multiple threads.

- TLP provides parallelism at a much larger granularity compared to ILP.

- In multiprocessors ILP and TLP work together.
Moore’s Law and Dennard Scaling

● An observation made in 1965 by Intel co-founder:
  - The number of transistors in a dense integrated circuit doubles approximately every two years.
  - It has been used as a standard in semiconductor industry (a self-fulfilling prophecy).

● Fueled by Dennard scaling:
  - The amount of power required to run the transistors in a specific unit volume stays constant despite increasing their number, such that the voltage and current scale with length.
  - In abstract, as transistors get smaller, power density stays constant.
  - Yet, this observation is no longer becoming valid as transistors are growing very small. The scaling of voltage and current with length is reaching its limits, since transistor gates have become too thin, affecting their structural integrity, and currents are starting to leak.
Moore’s Law by Intel

Mastering Moore’s Law

Intel’s progress in packing more transistors on mainstream microprocessor chips

<table>
<thead>
<tr>
<th>Year</th>
<th>Transistor Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1971</td>
<td>1 million</td>
</tr>
<tr>
<td>1980</td>
<td>10,000</td>
</tr>
<tr>
<td>1990</td>
<td>1,000,000</td>
</tr>
<tr>
<td>2000</td>
<td>100,000</td>
</tr>
<tr>
<td>2010</td>
<td>10 billion</td>
</tr>
</tbody>
</table>

*Upgraded versions of prior models

Source: Intel

THE WALL STREET JOURNAL.

en.wikipedia.org
Why haven’t CPU clock speed increased in the last few years?

- The most important parts in a processor are the transistors, the electronic devices that act as switches in order to construct logical gates.

- So, if we feed our processor with one input signal per second and the processor performs our operations error-free, we say that the processor is clocked at 1 Hz.

- Looking at it the other way around, a processor with a clock speed of 3 GHz allows us to feed it with 3 billion operations per second, and we can still expect it to perform as predicted.

- Why do we see so few Intel® processors over 3.7 GHz in the last few years?

- The reason is heat dissipation and power consumption.

- The faster we switch the transistors on and off, the more heat will be generated. Without proper cooling, they might fail and be destroyed.
New Trend of HPC

- Two major trends:
  - Through the application of multicore computing. Processor architects focus on throughput, not clock speed, to improve performance.
  - Access to widely available graphic processing units (GPUs) for general processing.
Flynn’s Taxonomy

- First proposed by Michael J. Flynn in 1966, Flynn's taxonomy is a specific classification of parallel computer architectures that are based on the number of concurrent instruction (single or multiple) and data streams (single or multiple) available in the architecture. The four categories in Flynn's taxonomy are the following:
  - (SISD) single instruction, single data
  - (SIMD) single instruction, multiple data
  - (MISD) multiple instruction, single data
  - (MIMD) multiple instruction, multiple data
  - (SIMT) Single instruction, multiple threads
Flynn’s Taxonomy

- **SISD**
  - Typical thread
- **SIMD**
  - Vector processors
  - GPUs
- **MISD**
  - For fault tolerance
- **MIMD**
  - Cluster of computers
  - Supercomputers

Source: en.wikipedia.org
Shared Memory System

- All processors share a single address space.
- Communication is implicit: write and read operations on shared variables.
- Simple programming model: no data distribution among processors.
- Limited scalability (memory contention).
Shared Memory System

- Symmetric Multiprocessor (SMP): memory access latency is the same for all processors.
- Also called Uniform Memory Access (UMA).
- Non-Uniform Memory Access (NUMA):
  - Different access times to memory modules.
  - Processor caches mitigate latency.
  - Improved scalability.

Distributed Memory System

- Each processor has its own private memory.
- Communication is explicit through message passing.
- Involved programming model: data distribution.
- Good scalability.
Titan Supercomputer CPU (Computer Node)

- Each Titan compute node contains (1) AMD Opteron™ 6274 (Interlagos) CPU and NVDIA K20 GPU.
- Each NUMA node contains a die's L3 cache and its (4) compute units (8 cores).
- Each compute unit contains (2) integer cores (and their L1 cache), a shared floating point scheduler, and shared L2 cache.
Parallelizing Compilers

- Parallelization as an optimization.
- A compiler may automatically parallelize a sequential code.
- Big quest during the 90s:
  - General parallelization is too hard for a compiler.
  - Figuring out all data dependences is impossible at compile time.
  - Relative success on more restrictive languages (CoArray Fortran, High Performance Fortran).
  - Vectorization of loops has become mainstream.
- The programmer has to be involved in the parallelization of the code.
Parallel Programming

- Considered harder than sequential programming.
- Training in computer science is based on von Neumann architecture (sequential).

Considerations in parallel programming:
- Communication, sending data between processors.
- Synchronization, coordinating access to shared resources.
- Load balancing, distributing computation evenly across processor set.

Parallel machine:
- Contains a set of n processors, numbered from 0 to n-1.
- Each processor has a unique ID.
Parallel Sum
Parallel Languages

- Concurrent programming languages, libraries, APIs, and parallel programming models (such as algorithmic skeletons) have been created for programming parallel computers.

- These can generally be divided into classes based on the assumptions they make about the underlying memory architecture - shared memory, distributed memory, or shared distributed memory.
  - Shared memory programming languages communicate by manipulating shared memory variables.
  - Distributed memory uses message passing.
  - POSIX Threads and OpenMP are two of most widely used shared memory APIs, whereas Message Passing Interface (MPI) is the most widely used message-passing system API.
  - CUDA and OpenACC for NVIDIA accelerators.
Which is the best language for parallel programming?

- Fortran
- C (C++)
- Others: Java, Python, MATLAB, R
What topics will be covered?

- Introduction (modern processors, trend of HPC, basic optimization techniques for serial code).
- Parallelization (parallel computers, parallel concept, scalability, and more).
- Shared-memory parallelism with OpenMP.
- Distributed-memory parallelism with MPI.
- New HPC techniques (MATLAB PCT, GPU with OpenACC, Cloud Computing with Apache Spark).
Question?

- Question?
Architecture

CSCI 4850/5850 High-Performance Computing

Spring 2018

Tae-Hyuk (Ted) Ahn
Department of Computer Science
Program of Bioinformatics and Computational Biology
Saint Louis University
Learning Objectives

- Understand the basic computer architecture and levels of parallelism.
- Evaluate technologies used to augment hearing loss.
John von Neumann

- Contributions to mathematics, economics, computer science, and statistics.
- Member of Manhattan Project and Institute for Advanced Study.
- Proposed a design for a digital computer (EDVAC) in 1945 that later became the von Neumann model.
- Introduced cellular automata.
- Designed merge sort algorithm.

www.wikipedia.org
The von Neumann Architecture

- The “classical” von Neumann architecture consists of main memory, a central processing unit (CPU) and an interconnection between the memory and the CPU.
- Arithmetic/Logic unit (ALU): executes the instruction.
- Main memory consists of a collection of locations, each of which is capable of storing both instructions and data.
- Every location consists of an address, which is used to access the location and the contents of the location—the instructions or data stored in the location.
The von Neumann Architecture

- The control unit is responsible for deciding which instructions in a program should be executed.
- The **ALU** is responsible for executing the actual instructions.
- Data in the CPU and information about the state of an executing program are stored in special, very fast storage called **registers**.
- Instructions and data are transferred between the CPU and memory via the interconnect, the bus.
- When data or instructions are transferred from memory to the CPU, we sometimes say the data or instructions are **fetched** or **read** from memory.
Processes, Multitasking, and Threads

- When a user runs a program, the operating system creates a **process** — an instance of a computer program that is being executed.

- **Multitasking**: the operating system provides support for the apparent simultaneous execution of multiple programs.

- **Threading** provides a mechanism for programmers to divide their programs into more or less independent tasks with the property that when one thread is blocked another thread can be run.

- Both processes and threads are independent sequences of execution. The typical difference is that threads (of the same process) run in a shared memory space, while processes run in separate memory spaces.
Instruction

- CPU instructions are numbers stored in memory.
- The instructions are specific to the CPU architectures.
- Basic operation: read instruction from memory, decode/crack/crunch it, execute, write-back, advance.
CISC vs RISC

- **Complex Instruction Set Computing (CISC)**
  - Large number of instruction sets
  - The primary goal of CISC architecture is to complete a task in as few lines of assembly as possible.
  - EX) A*B CISC instruction: `MULT A, B`

- **Reduced Instruction Set Computing (RISC)**
  - Lesser set of instructions
  - RISC processors only use simple instructions that can be executed within one clock cycle.
  - EX) A*B RISC instruction:```
    LOAD R1, A
    LOAD R2, B
    PROD A, B
    STORE R3, A```
## CISC vs RISC

<table>
<thead>
<tr>
<th>CISC</th>
<th>RISC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emphasis on hardware</td>
<td>Emphasis on software</td>
</tr>
<tr>
<td>Includes multi-clock complex instructions</td>
<td>Single-clock, reduced instruction only</td>
</tr>
<tr>
<td>Memory-to-memory: &quot;LOAD&quot; and &quot;STORE&quot;</td>
<td>Register to register: &quot;LOAD&quot; and &quot;STORE&quot;</td>
</tr>
<tr>
<td>incorporated in instructions</td>
<td>are independent instructions</td>
</tr>
<tr>
<td>Small code sizes, high cycles per second</td>
<td>Low cycles per second, large code sizes</td>
</tr>
<tr>
<td>Transistors used for storing complex</td>
<td>Spends more transistors on memory registers</td>
</tr>
<tr>
<td>instructions</td>
<td></td>
</tr>
</tbody>
</table>

x86 is a CISC architecture. The number of instructions is a big factor as all CISC architectures with all more instructions. Furthermore as instructions are complex in CISC they can take >1 cycle to complete, where as in RISC they should be single cycle.
Now we are into the **post-RISC era**, where processors have the advantages of both RISC and CISC architecture.

The gap between RISC and CISC has blurred significantly.

Intel’s Pentium Core 2 Duo processor can execute more than one CISC instruction per **clock cycle** due to increased processing speed.

This speed advantage would enable CISC instructions to be pipelined. On the other hand, RISC instructions are also becoming complex (CISC-like) to take advantage of increased processing speed.

So at present, classifying a processor as RISC or CISC is almost impossible, because their instructions sets all look similar.
Clock and Registers

- Clock: oscillates in a cycle with constant frequency. It determines the speed of the CPU.

  ![Clock Diagram]

- **Clock cycle**: discrete time for event (e.g., 0.25 ns)
- **Clock rate**: the inverse of the clock period (e.g., 4 GHz)

- Registers:
  - Program Counter (PC): stores the address of the next instruction to be executed.
  - Instruction Register (IR): stores the current instruction code.
An instruction has go through the following stages:
- Instruction Fetch (IF)
- Instruction Decode (ID)
- Instruction Execution (EX)
- Memory Read/Write (MEM)
- Result Writeback (WB)

Let us assume each stage takes one clock cycle.
Assembly Line – Instruction Level Parallelism

<table>
<thead>
<tr>
<th>i</th>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>MEM</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td></td>
<td>IF</td>
<td>ID</td>
<td></td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td></td>
<td>IF</td>
<td>ID</td>
<td></td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td></td>
<td>IF</td>
<td>ID</td>
<td></td>
<td>MEM</td>
<td>WB</td>
</tr>
</tbody>
</table>

The diagram illustrates the concept of instruction-level parallelism in an assembly line scenario. Each row represents a stage in the pipeline (IF: Instruction Fetch, ID: Instruction Decode, EX: Execute, MEM: Memory Access, WB: Write Back), showing how different tasks can be executed in parallel to improve efficiency.
Data Dependences

- Instruction level parallelism is limited by *data dependences*

\[
i_1. R2 <- R1 + R3 \\
i_2. R4 <- R2 + R3
\]

\[(w*x) * (y*z)\]
Data Dependences

- **Hazards** are problems with the instruction pipeline in CPU microarchitectures when the next instruction cannot execute in the following clock cycle.

- Instruction level parallelism is limited by *data dependences*.

- Types of dependences:
  - **RAW**: read-after-write
    
    \[
    \begin{align*}
    i_1 & : R2 \leftarrow R1 + R3 \\
    i_2 & : R4 \leftarrow R2 + R3
    \end{align*}
    \]
    
    (i2 tries to read a source before i1 writes to it)

  - **WAR**: write-after-read
    
    \[
    \begin{align*}
    i_1 & : R4 \leftarrow R1 + R5 \\
    i_2 & : R5 \leftarrow R1 + R2
    \end{align*}
    \]
    
    (i2 tries to write a destination before it is read by i1)

  - **WAW**: write-after-write
    
    \[
    \begin{align*}
    i_1 & : R2 \leftarrow R4 + R7 \\
    i_2 & : R2 \leftarrow R1 + R3
    \end{align*}
    \]
    
    (i2 tries to write an operand before it is written by i1)
Memory Hierarchy

Storage systems are organized in a hierarchy:
- Speed
- Cost
- Volatility

faster

volatile

cheaper per byte
## Performance of Storage

- Registers are fast, typically one clock cycle to access data.
- Cache access takes tens of cycles.
- Memory access takes hundreds of cycles.
- Movement between levels of storage hierarchy can be explicit or implicit.

<table>
<thead>
<tr>
<th>Level</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>registers</td>
<td>cache</td>
<td>main memory</td>
<td>disk storage</td>
</tr>
<tr>
<td>Typical size (KB)</td>
<td>&lt; 1</td>
<td>&gt; 16</td>
<td>&gt; 16</td>
<td>&gt; 100</td>
</tr>
<tr>
<td>Implementation technology</td>
<td>custom memory with multiple ports, CMOS</td>
<td>on-chip or off-chip CMOS SRAM</td>
<td>CMOS DRAM</td>
<td>magnetic disk</td>
</tr>
<tr>
<td>Access time (ns)</td>
<td>0.25 – 0.5</td>
<td>0.5 – 25</td>
<td>80 – 250</td>
<td>5,000,000</td>
</tr>
<tr>
<td>Bandwidth (MB/sec)</td>
<td>20,000 – 100,000</td>
<td>5000 – 10,000</td>
<td>1000 – 5000</td>
<td>20 – 150</td>
</tr>
<tr>
<td>Managed by</td>
<td>compiler</td>
<td>hardware</td>
<td>operating system</td>
<td>operating system</td>
</tr>
<tr>
<td>Backed by</td>
<td>cache</td>
<td>main memory</td>
<td>disk</td>
<td>CD or tape</td>
</tr>
</tbody>
</table>
Memory Wall

- **Latency** is the delay from input into a system to desired outcome.
- Access to registers is orders of magnitude faster than access to memory.
- Solutions:
  - Caching.
  - Prefetching.
  - Multithreading.
Memory Access

- A logical address is translated into a physical address in virtual memory using a page table
  - The translation lookaside buffer (TLB) is an efficient on-chip address translation cache
  - Memory is divided into pages
  - Virtual memory systems store pages in memory (RAM) and on disk
    - Page fault: page is fetched from disk

- L1 caches (on chip)
  - I-cache stores instructions
  - D-cache stores data

- L2 cache (E-cache, on/off chip)
  - Is typically unified: stores instructions and data
Latency and Bandwidth

- The two most important terms related to performance for memory subsystems and for networks:

  - **Latency:**
    - How long does it take to retrieve a word of memory?
    - Units are generally nanoseconds (milliseconds for network latency) or clock periods (CP)
    - Sometimes address are predictable: compiler will schedule the fetch. Predictable code is good!

  - **Bandwidth:**
    - What data rate can be sustained once the message is started?
    - Units are B/sec (MB/sec, GB/sec, etc.)
Caching

- A faster and smaller memory that only stores the most popular values.

- Principle of locality:
  - Spatial: if value X is accessed, its neighboring values will be likely to be accessed in the near future.
  - Temporal: if value X is accessed, it will be likely to be accessed in the near future.
Cache Mappings

- Fully associative: a line can be placed anywhere in the cache.

- Direct mapped: a line has a unique place in the cache.

- \( n \)-way set associative: a line has \( n \) locations where it can be placed in the cache.
Multicore Architecture

Single core

Dual core
Shared Memory Multiprocessor

- Processors access *shared memory* via a common switch, e.g. a *bus*
  - Problem: a single bus results in a *bottleneck*

- Shared memory has a *single address space*
  - Access to memory is restricted to one processor at a time
  - This limits the speedup and scalability with respect to the number of processors
Shared Memory Multiprocessor with Local Cache

- Add local cache to improve performance
- Problem: how to ensure cache coherence?
A cache coherence protocol ensures that processors obtain newly altered data when shared data is modified by another processor.

Because caches operate on cache lines, more data than the shared object alone can be effected, which may lead to false sharing.
Distributed Memory Multicomputer

- Massively parallel processor (MPP) systems with $P > 1000$
- Communication via message passing
- Nonuniform memory access (NUMA)
- Network topologies
  - Mesh
  - Hypercube
  - Cross-bar switch
Topology

- How the components are connected.

- Important properties
  - **Diameter**: maximum distance between any two nodes in the network (hop count, or # of links).
  - **Nodal degree**: how many links connect to each node.
  - **Bisection bandwidth**: The smallest bandwidth between half of the nodes to another half of the nodes.

- A good topology: **small diameter, small nodal degree, large bisection bandwidth**.
Interconnection Network

(a) Ring

(b) 2D mesh

(c) 2D torus

(d) Hypercube (4-cube)
Mesh Topology

- Network of $P$ nodes has mesh size $\sqrt{P} \times \sqrt{P}$
- Diameter $2 \times (\sqrt{P} - 1)$
- Torus network wraps the ends
  - Diameter $\sqrt{P} - 1$
Hypercube Topology

- $d$-dimensional hypercube has $P = 2^d$ nodes
- Diameter is $d = \log_2 P$
- Node addressing is simple
  - Node number of nearest neighbor node differs in one bit
  - Routing algorithm flips bits to determine possible paths, e.g. from node 001 to 111 has two shortest paths
    - 001 → 011 → 111
    - 001 → 101 → 111

\begin{itemize}
  \item $d=2$
  \item $d=3$
  \item $d=4$
\end{itemize}