ECE 5315 Multiprocessor-Based System Design

- ECE Technical Elective
- 3 cr
- Instructor: Dr. Taek M Kwon
- Computer Usage: PC, .Net
- Prerequisites: ECE 2325, ECE4305
Assessment

- Projects+HW  20%
- Midterm  35%
- Final    40%
- Attendance 5%
Text Book

*Scalable Parallel Computer Architecture*, by David Culler and Jaswinder Singh, Morgan Kaufmann, 1999
Course Objectives

- Basic concepts of scalability
- Parallel computer models
- Performance metrics
- Modern microprocessor design
- Shared memory multiprocessors
- Distributed memory multiprocessors with latency tolerance
- Cache coherence, consistency
- Multithreading and synchronization
Notations

- **Bit** = b, 5 bits or 5b
- **Byte** = B, 4 Bytes = 4B = 16b
- Small and large numbers

<table>
<thead>
<tr>
<th>Notation</th>
<th>Symbol</th>
<th>Unit</th>
<th>Exponent</th>
</tr>
</thead>
<tbody>
<tr>
<td>nano</td>
<td>n</td>
<td>One billions</td>
<td>$10^{-9}$</td>
</tr>
<tr>
<td>pico</td>
<td>p</td>
<td>One trillions</td>
<td>$10^{-12}$</td>
</tr>
<tr>
<td>femto</td>
<td>f</td>
<td>One quadrillionth</td>
<td>$10^{-15}$</td>
</tr>
<tr>
<td>atta</td>
<td>a</td>
<td>One quintillionth</td>
<td>$10^{-18}$</td>
</tr>
<tr>
<td>giga</td>
<td>G</td>
<td>Billion</td>
<td>$10^9$</td>
</tr>
<tr>
<td>tera</td>
<td>T</td>
<td>Trillion</td>
<td>$10^{12}$</td>
</tr>
<tr>
<td>peta</td>
<td>P</td>
<td>Quadrillion</td>
<td>$10^{15}$</td>
</tr>
<tr>
<td>exa</td>
<td>E</td>
<td>Quintillion</td>
<td>$10^{18}$</td>
</tr>
</tbody>
</table>
## Computer Generations

<table>
<thead>
<tr>
<th>Generation</th>
<th>Technology</th>
<th>Software, OS</th>
<th>Example System</th>
</tr>
</thead>
<tbody>
<tr>
<td>First 1946-1956</td>
<td>Vacuum tubes, relay memory, single bit CPU</td>
<td>Machine, assembly language, no sub</td>
<td>ENIAC, IBM 701, Princeton IAS</td>
</tr>
<tr>
<td>Second 1956-1967</td>
<td>Transistors, core memory, I/O channels, FP</td>
<td>Algol, Fortran, Batch proc OS</td>
<td>IBM 7030, CDC1604, Univac LARC</td>
</tr>
<tr>
<td>Third 1967-1978</td>
<td>ICs, pipelined CPU, microprogramed controller</td>
<td>C, multiprogramming, time sharing OS</td>
<td>PDP-11</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IBM 360/370</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CDC 6600</td>
</tr>
<tr>
<td>Fourth 1978-1989</td>
<td>VLSI, solid-state memory, multi, vector processor</td>
<td>Symmetric, multiproc, parallel compiler</td>
<td>IBM PC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>VAX 9000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Cray X/MP</td>
</tr>
<tr>
<td>Fifth 1990-present</td>
<td>ULSI, scalable computers, clusters</td>
<td>Java, multithreading, distributed OS</td>
<td>IBM SP2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SGI Origin 2000</td>
</tr>
</tbody>
</table>
Scalability

• A computer is called **scalable** if it can *scale* up to accommodate increasing demand or *scale* down to reduce cost

• **Functionality and Performance**: increase of computing power to *n times* when the system resource is improved *n times*

• **Scaling in Cost**: scale up *n times* costs no more than *n* or *n log n* times

• **Compatibility**: scale up should not cause loss of compatibility
Scalable Parallel Computer Architecture: Shared Nothing

Shared-nothing architecture

C=Cache, D=Disk, M=Memory, NIC=Network Interface Circuit, P=Processor
Scalable Parallel Computer Architecture: Shared Disk

Shared-disk architecture

C=Cache, D=Disk, M=Memory, NIC=Network Interface Circuit, P=Processor
Scalable Parallel Computer Architecture: Shared Memory

Shared-memory architecture

C=Cache, D=Disk, M=Memory, NIC=Network Interface Circuit, P=Processor
Dimensions of Scalability

- **Resource Scalability**: gaining higher performance by increasing resources such as machine size (# of processors), storage (cache, main mem, disks), software

- **Application Scalability**: the same program should run with proportionally better performance on a scaled up system

- **Technology Scalability**: adaptability to changes in technology
Flynn’s classification

- SISD: single-instruction single-data stream
- SIMD: single-instruction multiple-data stream
- MIMD: multiple-instruction multiple-data stream
- SPMD: single-program multiple-data stream
Synchrony

- **Synchronous at instruction level**: tightly synchronized, PRAM
- **Asynchronous**: each process executes at its own pace, MIMD
- **Bulk Synchronous Parallel (BSP)**: synchronize at every superstep
- **Loosely Synchronous**: synchronize at divided phases
Interaction Mechanisms

- **Share Variables**: interaction through shared variables, PRAM
- **Message Passing**: multiprocessor, multicomputer
Address Spaces

- **Single Address Space**: all memory locations reside in a single address space from the programmer’s point of view, PRAM
- **Multiple Address Space**: each processor has its own address space, multicomputer
- **Uniform Memory Access (UMA)**
- **Non-uniform Memory Access (NUMA)**
- **Local Memory, Remote Memory**
Memory Models

- **Exclusive Read Exclusive Write (EREW):** A memory cell can be read or written by at most one processor.
- **Concurrent Read Exclusive Write (CREW):**
- **Concurrent Read Concurrent Write (CRCW):** Multiple processors can both read from or write to the same memory location.
Def: Atomic Operation

- **Indivisible:** once it starts, it cannot be interrupted in the middle.
- **Finite:** once it starts, it will finish in a finite amount of time
Performance Attributes

- Machine size: $n$
- Clock rate: $f$, MHz
- Workload: $W$, MFlops
- Sequential execution time: $T_1$, sec
- Parallel execution time: $T_n$, sec
- Speed: $P_n = W/T_n$, Mflops/s
- Speedup: $S_n = T_1/T_n$
- Efficiency: $E_n = S_n/n$
- Utilization: $U_n = P_n/nP_{peak}$
- Startup time, us,
- Asymptotic bandwidth, MB/s
Abstract Machine Model: PRAM

- Parallel Random Access Machine (PRAM)
  - Machine size $n$ can be arbitrarily large
  - A cycle is a basic time step
  - Within a cycle, each processor executes exactly one instruction
  - All processors are synchronized at each cycle
  - Synchronization overhead is assumed to be zero
  - Communication is done through shared variables
  - Communication overhead is assumed to be zero
  - An instruction can be any random-access machine instruction (fetch one or two words from memory as operands, perform an ALU operation, store the result back in memory)
Many details of real system are ignored

Unrealistic assumption

But simplicity makes it an excellent model for developing parallel algorithms

Many parallel algorithms developed with the use of PRAM turn out to be practical

Still lacks the properties of real-life parallel computers → BSP
Bulk Synchronous Model (BSP)

—Proposed by Leslie Valiant, Harvard University
—To overcome the shortcomings of the PRAM while keeping simplicity
—Consists of a set of n processor/memory pairs
—MIMD
BSP

- Basic time step = cycle
- In each step, process executes the computation operation in at most $w$ cycle
- $g$: $h$ relation coefficient, $gh$ cycles for communication
- A barrier forces processes to wait so that all processes have to finish the current superstep before any of them can begin next superstep
- $/\!\!/$: barrier synchronization
- Loosely synchronous at the superstep
BSP (2)

- Within each superstep, different processes execute asynchronously at their own pace
- Synchronization by shared variable or message passing
- A processor can access not only its own memory but also any remote memory in another node
- Single address space
- Within each superstep, each computation uses only data in its local memory → computations are independent of other processors
**BSP (2)**

- The same memory location cannot be read or written by multiple processes
- All memory or communication operations in a superstep must be completed before any operation of the next step → sequential memory consistency
- Allows overlapping of the computation, communication, and synchronization within a superstep
Phase Parallel Model

- **Parallelism phase**: overhead work involved in process management, e.g. process creation, grouping
- **Computation phase**
- **Interaction phase**: communication, synchronization, aggregation
BSP Vector Multiplication Example for 8 processors (1)

• Superstep 1
  — Computation: \( w = \frac{2N}{8} \) cycles per processor (mul & sum)
  — Communication: Processors 0, 2, 4, 6 send their sums to processors 1, 3, 5, 7 (\( g = 1 \))
  — Barrier Synchronization (\( l = 1 \))

• Superstep 2
  — Computation: Processors 1, 3, 5, 7 each perform one addition (\( w = 1 \))
  — Communication: Processors 1 and 5 send their sums to processors 3 and 7 (\( g = 1 \))
  — Barrier Synchronization (\( l = 1 \))
BSP Vector Multiplication Example for 8 processors (2)

• Superstep 3
  — Computation: Processors 3 and 7 each perform one addition (w=1)
  — Communication: Processors 3 sends its sum to processors 7 (g=1)
  — Barrier Synchronization (l=1)

• Superstep 4
  — Computation: Processors 7 performs one addition (w=1)

• Total execution time = \( 2N/8 + 3g + 3l + 3 \)
Physical Machine Models

- Parallel vector processor (PVP)
- Symmetric multiprocess (SMP)
- Massively parallel processor (MPP)
- Distributed shared memory machine
- Cluster of workstations (COW)
Parallel Vector Processor (PVP)

- Cray C-90, Cray T-90, NEC SX-4 super computers
- A small number of powerful custom designed vector processors
- High-bandwidth custom designed cross-bar network that connects a number of shared memory modules
- Uses a large number of vector registers and an instruction buffer
Symmetric Multiprocessor (SMP)

- IBM R50, SGI Power Challenge, DEC Alpha server 8400
- Uses commodity microprocessors with on-chip cache
- Shared memory through a high speed snoopy bus or cross bar
- Used in database and on-line transaction systems
- **Symmetric**: every processor has equal access to the shared memory, I/O devices, OS services
Massively Parallel Processor (MPP)

- Cray T3D, T3E
- Used for applications with high available parallelism
- Scientific computing, engineering simulation, signal processing, astronomy, environmental simulation
- Commodity processors in processing nodes
- Distributed memory over processing nodes
- High communication bandwidth
- Asynchronous MIMD with message-passing
- Nodes are tightly coupled
Distributed Shared Machines (DSM)

- Stanford DASH architecture
- Memory is physically distributed among different nodes, but the system hardware and software create an illusion of a single address space to application users.
Cluster of Computers (COW)

- Each node is a complete workstation minus peripherals (monitor, keyboard, mouse,...) → headless workstation
- Nodes are connected through a commodity network, e.g., Ethernet, FDDI, ATM switch, etc
- Loosely coupled to I/O bus in a node
- Local disk
- A complete OS resides in each node
- **Single-system image**: a single computing resource
- High availability: the cluster still functions after a node failure, local disk failure, a local OS failure
- Scalable performance
# NOW performance comparison

<table>
<thead>
<tr>
<th>System Config</th>
<th>ODE (s)</th>
<th>Transport (s)</th>
<th>Input/Output (s)</th>
<th>Total (s)</th>
<th>Cost ($M)</th>
<th>Mflops/s per $M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cray C90</td>
<td>7</td>
<td>4</td>
<td>16</td>
<td>27</td>
<td>30</td>
<td>4.4</td>
</tr>
<tr>
<td>Intel Paragon</td>
<td>12</td>
<td>24</td>
<td>10</td>
<td>46</td>
<td>10</td>
<td>78</td>
</tr>
<tr>
<td>NOW</td>
<td>4</td>
<td>23,340</td>
<td>4030</td>
<td>27,347</td>
<td>4</td>
<td>0.32</td>
</tr>
<tr>
<td>NOW + ATM</td>
<td>4</td>
<td>192</td>
<td>2015</td>
<td>2211</td>
<td>5</td>
<td>3.3</td>
</tr>
<tr>
<td>NOW + ATM +PIO</td>
<td>4</td>
<td>192</td>
<td>10</td>
<td>205</td>
<td>5</td>
<td>35</td>
</tr>
<tr>
<td>NOW + ATM +PIO+AM</td>
<td>4</td>
<td>8</td>
<td>10</td>
<td>21</td>
<td>5</td>
<td>342</td>
</tr>
</tbody>
</table>
Scalable Design Principles (1)

• Principle of independence
  — Design that leads to independence of components as much as possible
  — Upgrading of one component should not require upgrading of remaining components

— Specific Independence Examples
  • Algorithm should be independent of architecture
  • Application should be independent of platform
  • Programming language should be independent of the machine
  • Nodes should be independent of network
Scalable Design Principles (2)

- Principle of balanced design
  - Design to minimize any performance bottleneck by avoiding unbalanced system design
  - Avoid single point of failure
  - Degradation factors: load imbalance, parallel overhead, communication start-up overhead, per-byte communication overhead
  - Try to limit degradation by each overhead less than 50%
Scalable Design Principles (3)

- **Overdesign**
  - Design features by anticipating future scale up
  - Allows smooth migration
  - Memory space: 32-b computers $\rightarrow$ 4GB address space, 64-b computers $2^{64} = 11.8 \times 10^{19}$ B. 64-b UNIX is easier to migrate
  - Bad example: 8086/8088 $\rightarrow$ 640-KB DOS; 286, 386, 486, Pentium $\rightarrow$ high memory, expanded memory, extended memory
  - Reduces total development and production cost

- **Backward compatibility**
  - Weed out obsolete features. Overdesign to anticipate future improvements and backward compatibility