Introduction to Parallel & Distributed Computing (review)

Spring 2024

Instructor: Guojie Luo

gluo@pku.edu.cn

Logistics

- **Final exam**
  - 第2周 周三 (2024.06.19) 上午 理教408
  - closed-book, closed-discussion exam
    - one A4-size single-sided cheat sheet allowed
    - one calculator allowed
  - Problems in English & Chinese
  - Answers in Chinese, English, or programming languages
- **Contact method in the next a few weeks**
  - ping TA or me in the wechat group

In this Lecture …

- **Review the (partial) contents that will be covered in the final**

Mean-Time-Between-Failures

- **Cheap nodes fail, especially when you have many**
  - **Example**
    - **Mean time between failures (MTBF) for 1 node = 3 years**
    - **MTBF for a raw 1000-node system ≈ 1 day**
  - **Solution: restrict programming model so you can efficiently "built-in" fault-tolerance (art)**

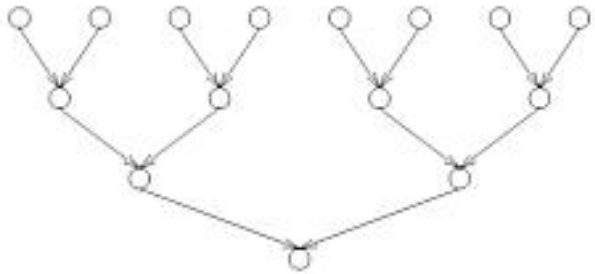
Source: Andy Konwinski, "Cloud Computing using MapReduce, Hadoop, Spark," ParLab Bootcamp 2012

Ways of Exploiting Parallelism

- Data decomposition
- Task decomposition
- Pipelining

Exercise: Task Decomposition

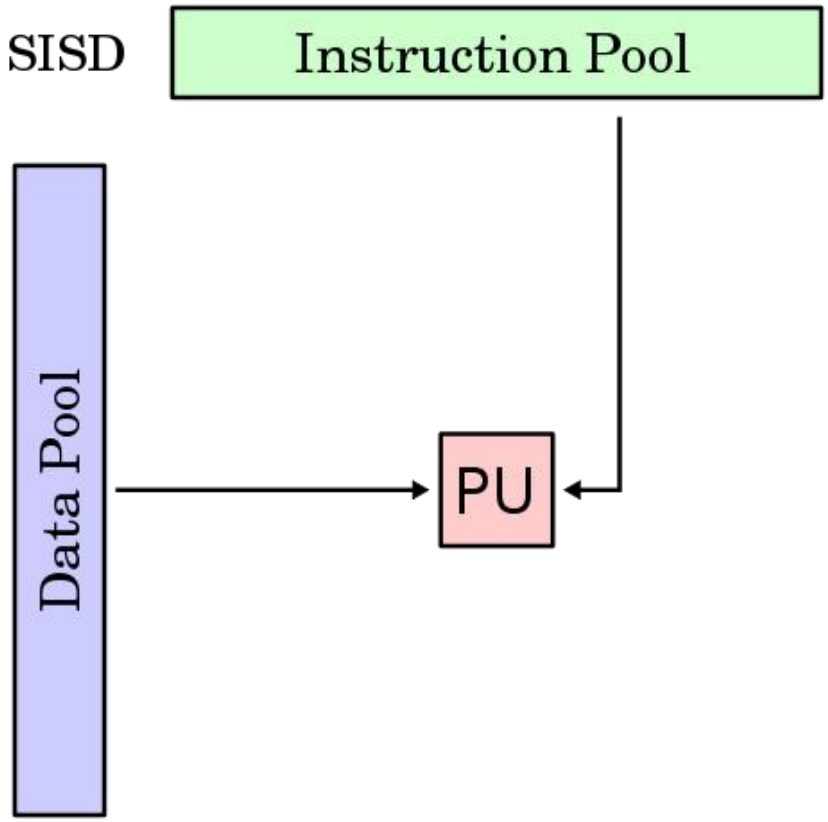
- For the task graph on the bottom right, determine
  - Maximum achievable speedup over one process assuming that an arbitrarily large number of processes is available.
  - The minimum number of processes needed to obtain the maximum possible speedup.
  - The maximum achievable speedup if the number of processes is limited to 2, 4, and 8.



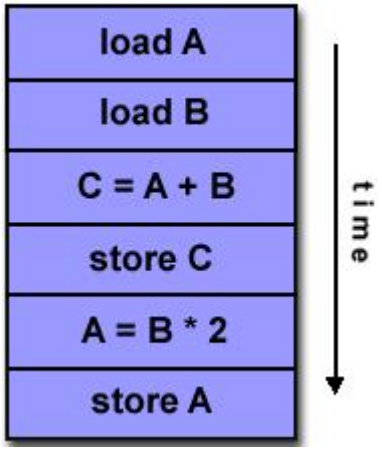
Parallel Architectures

- **Flynn's classical taxonomy**
  - **Single instruction, single data (SISD)**
  - **Multiple instruction, single data (MISD)**
  - **Single instruction, multiple data (SIMD)**
  - **Multiple instruction, multiple data (MIMD)**
    - **Shared-memory architecture**
    - **Distributed-memory architecture**
    - **Hybrid distributed-shared memory architecture**

SISD Architecture



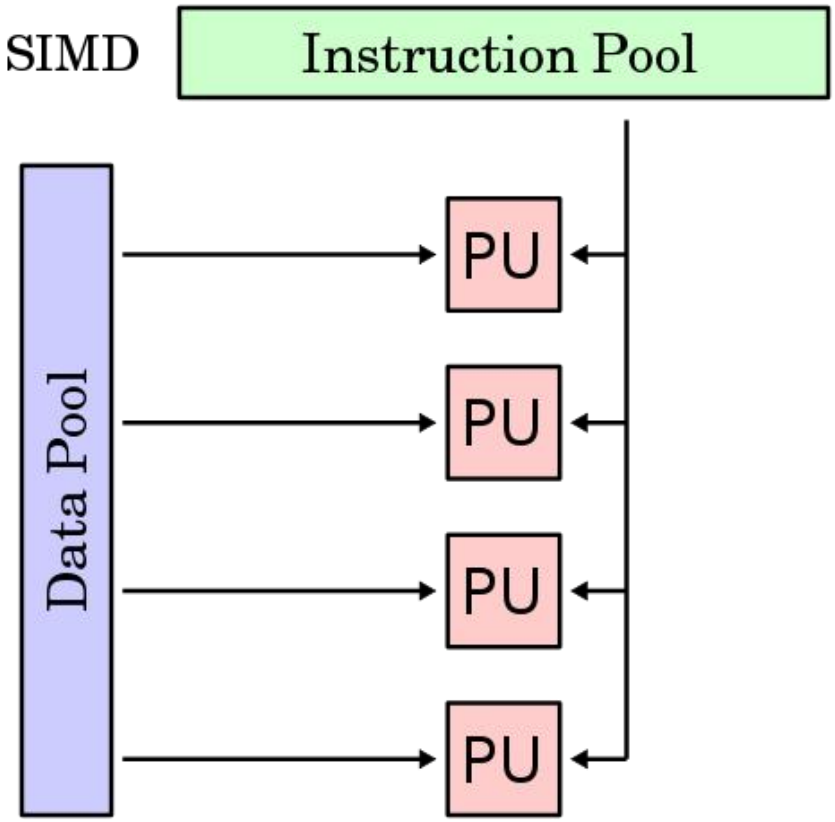
Example: single-core computers

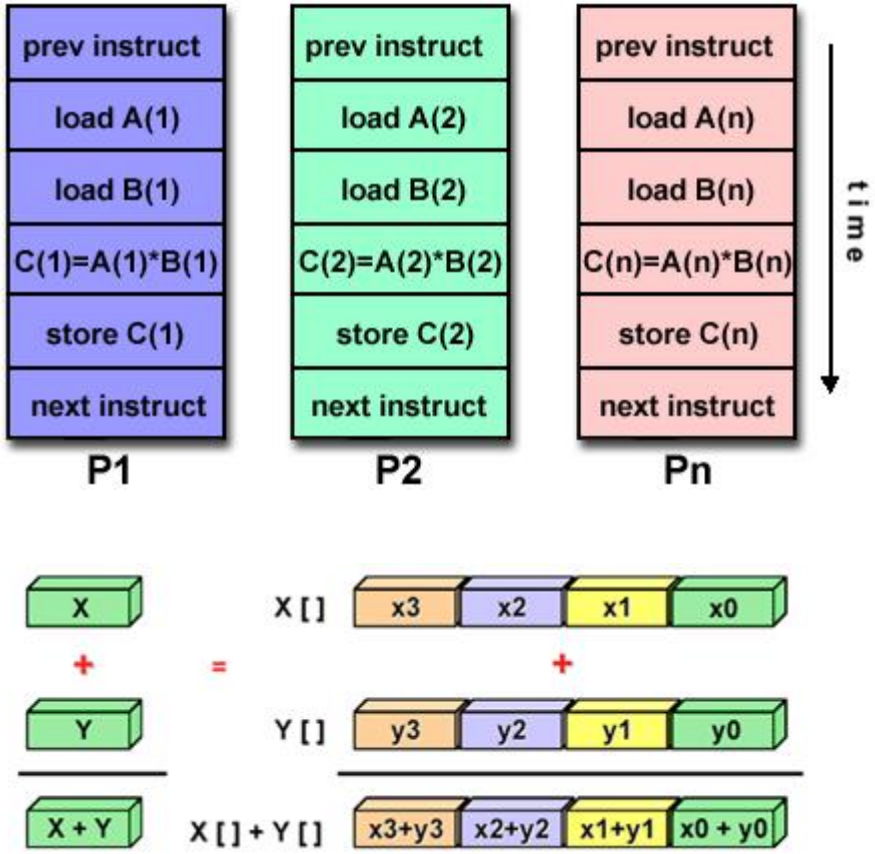


Source: http://en.wikipedia.org/wiki/Flynn's_taxonomy
Source: Blaise Barney (LLNL), Introduction to Parallel Computing

SIMD Architecture

Example: vector processors, GPUs

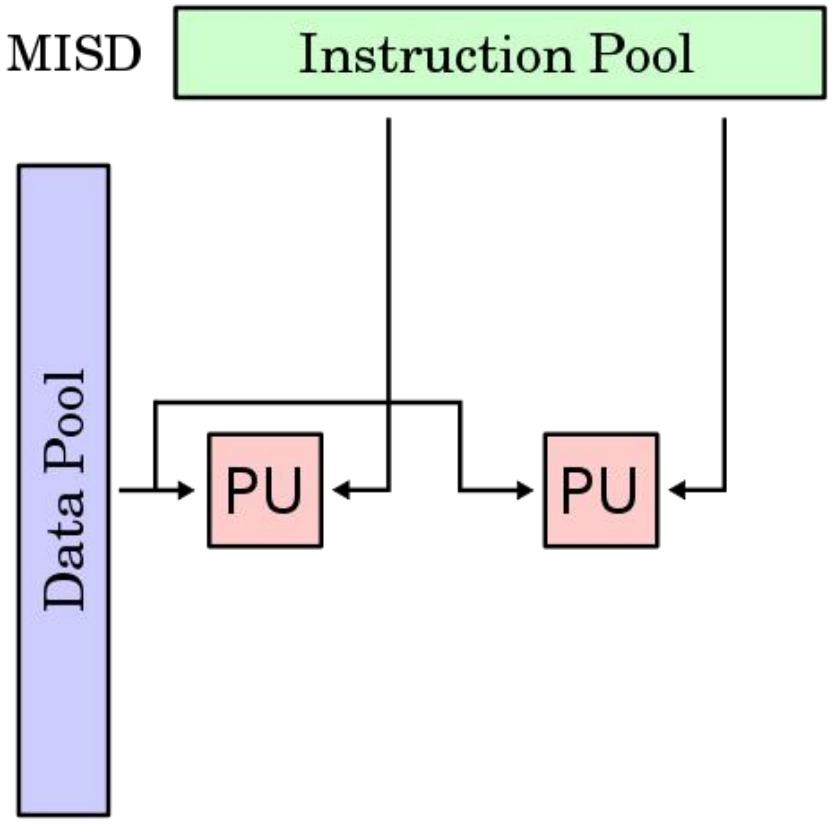


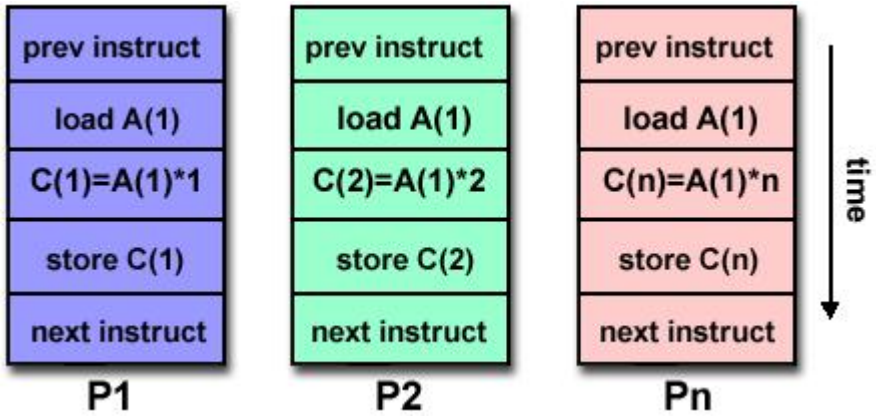


Source: http://en.wikipedia.org/wiki/Flynn's_taxonomy
Source: Blaise Barney (LLNL), Introduction to Parallel Computing

MISD Architecture

* no well-known systems fit

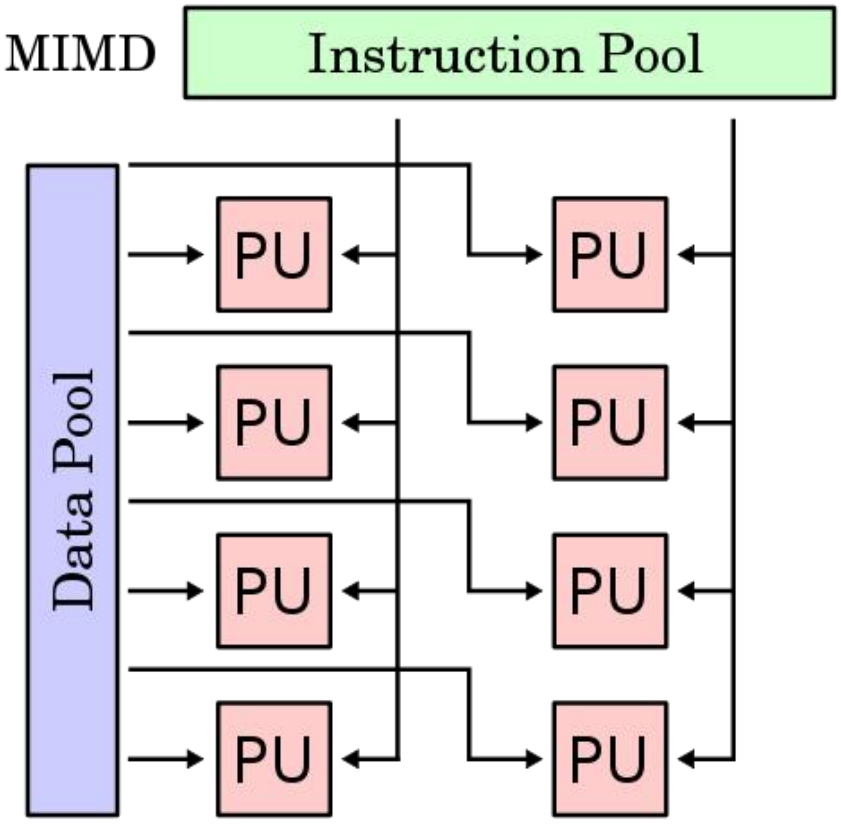


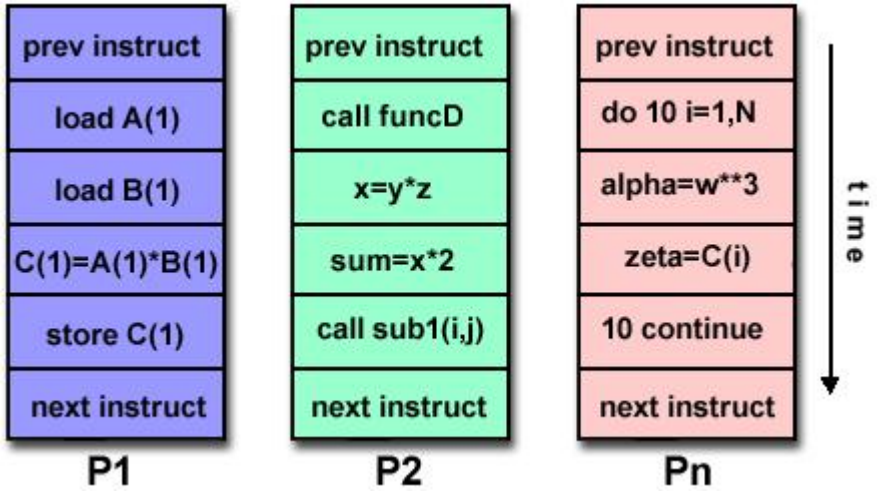


Source: http://en.wikipedia.org/wiki/Flynn's_taxonomy
Source: Blaise Barney (LLNL), Introduction to Parallel Computing

MIMD Architecture

Example: modern parallel systems

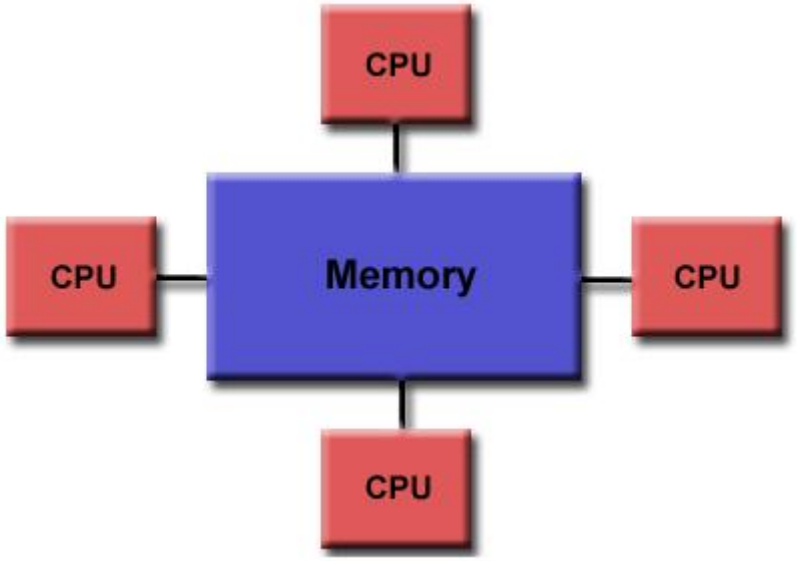




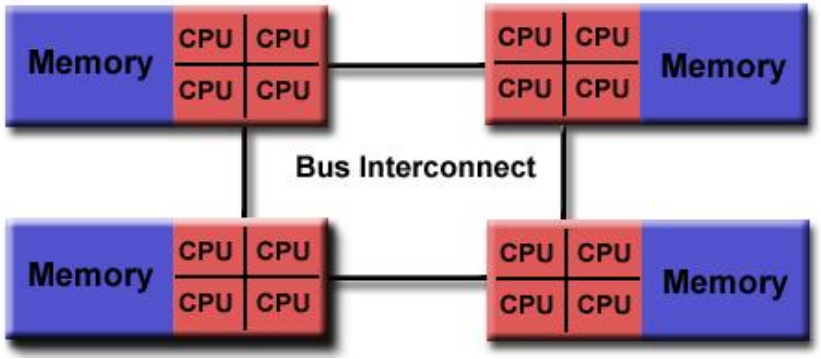
Source: http://en.wikipedia.org/wiki/Flynn's_taxonomy
Source: Blaise Barney (LLNL), Introduction to Parallel Computing

Shared-Memory Architecture

Symmetric Multiprocessors (SMP)

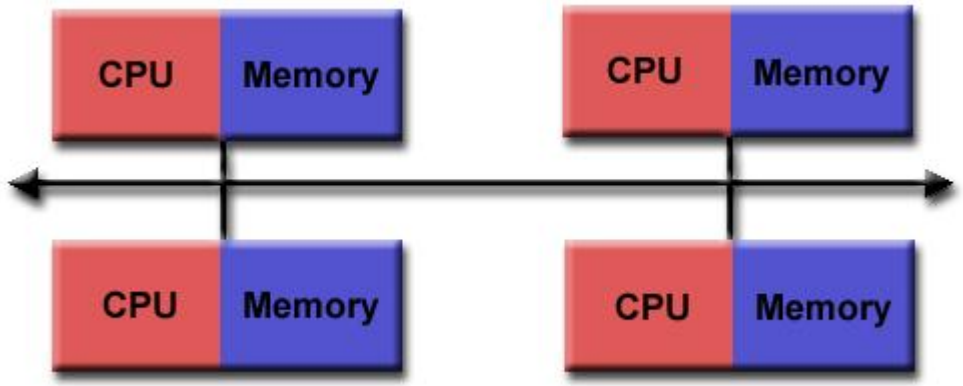


Nonuniform Memory Access (NUMA)



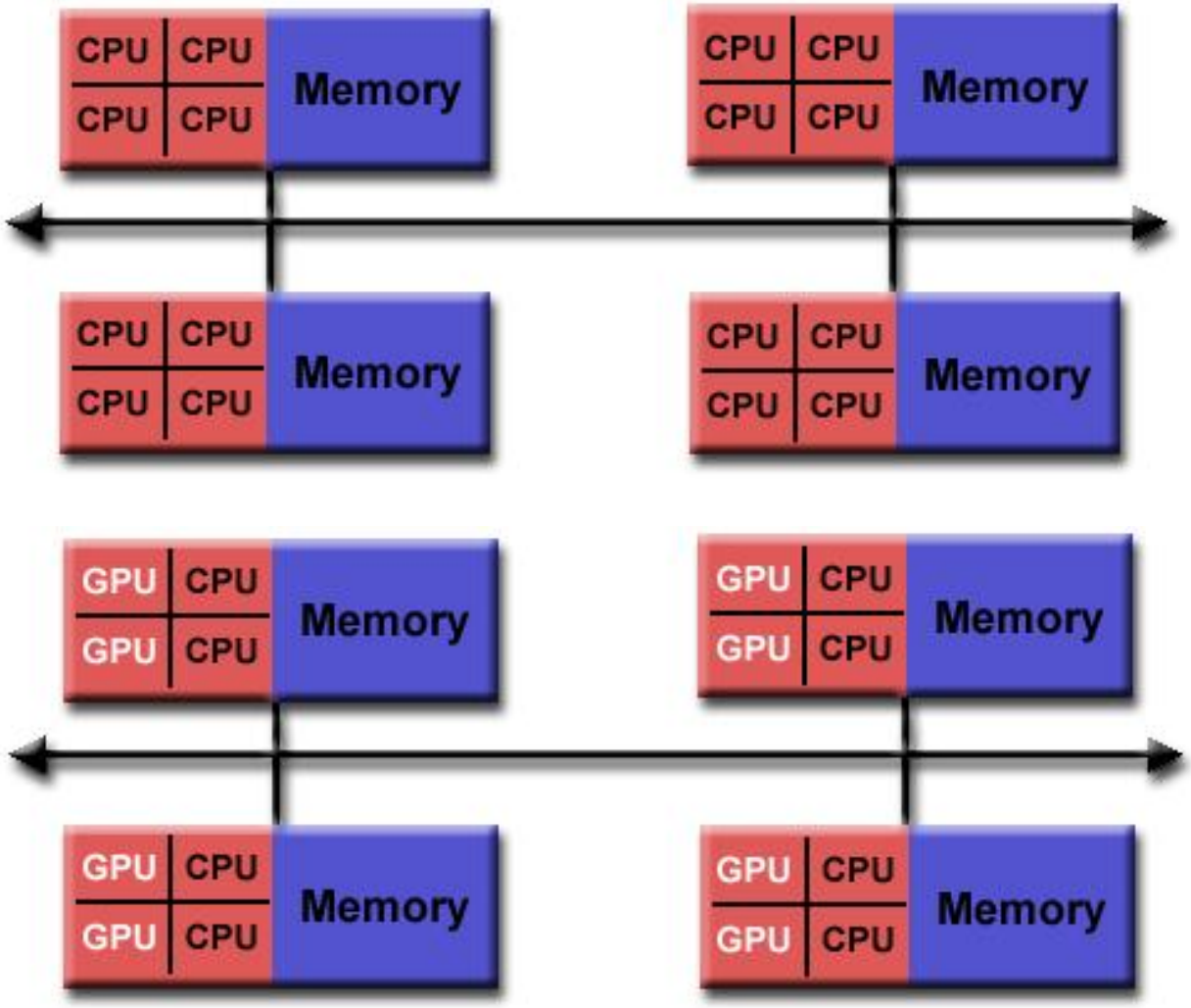
Source: Blaise Barney (LLNL), Introduction to Parallel Computing

Distributed-Memory Architecture



Source: Blaise Barney (LLNL), Introduction to Parallel Computing

Hybrid Architecture



Hybrid & Homogeneous

Hybrid & Heterogeneous

Source: Blaise Barney (LLNL), Introduction to Parallel Computing

Basic Concepts

- Synchronization
  - Barriers
  - Mutual exclusion
- Race condition
- Deadlock

Exercise: Race Condition

```
struct Node {
    struct Node* next;
    int data;
}
struct List {
    struct Node* head;
};
void AddHead (struct List* list, struct Node* node) {
    node->next = list->head;
    list->head = node;
}
```

Exercise: Deadlock

Thread 1

```
lock(lock_a);
a += 5;
lock(lock_b);
b += 7;
a += b;
unlock(lock_b);
a += 11;
unlock(lock_a);
```

Thread 2

```
lock(lock_b);
b += 5;
lock(lock_a);
a += 7;
a += b;
unlock(lock_a);
b += 11;
unlock(lock_b);
```

Analytical Model

- Amdahl's law
- Speedup
  - Superlinear speedup
  - Speedup anomalies
- Efficiency
- Isoefficiency
- Scalability

Exercise: Speedup

- For a problem size of interest, 10% of the operations of a parallel program are inside I/O functions that are executed on a single processor.
- What is the maximum speedup we could expect from executing a parallel version of this program on 10 processors?
  - if the I/O time can be *perfectly* hidden by computation
  - if the I/O time cannot be hidden *at all* by computation

Exercise: Superlinear Speedup

- Consider a 4-processor system with a distributed shared-address-space. Consider a simple cost model in which it takes 10 ns to access local cache, 100 ns to access local memory, and 250 ns to access remote memory.
  - A parallel program is running on this machine. The program is perfectly load balanced with 80% of all accesses going to local cache, 19% to local memory, and 1% to remote memory. What is the effective memory access time for this computation? If the computation is *memory bound*, what is the peak computation rate?
  - Now consider the same computation running on one processor. Here, the processor hits the cache 70% of the time and local memory 30% of the time. What is the effective peak computation rate for one processor? What is the fractional computation rate of a processor in a parallel configuration as compared to the serial configuration?

Exercise: Isoefficiency & Scalability

- **Let $n \geq f(p)$ denotes the isoefficiency relation of a parallel system and $M(n)$ denote the amount of memory required to store a program of input size n.**
- **Which of the following two parallel systems is more scalable?**
  - $f(p) = Cp$ **and** $M(n) = n^2$
  - $f(p) = C\sqrt{p}$ **and** $M(n) = n^2$

Loop Transformations

- Loop fission
- Loop fusion
- Loop inversion

Exercises: Loop Transformation (1/3)

```
float *a, *b;
for (int i = 1; i < N; i++) {
    if (b[i] > 0.0) a[i] = 2.0;
    else a[i] = 2.0 * fabs(b[i]);
    b[i] = a[i-1];
}
```

Exercises: Loop Transformation (2/3)

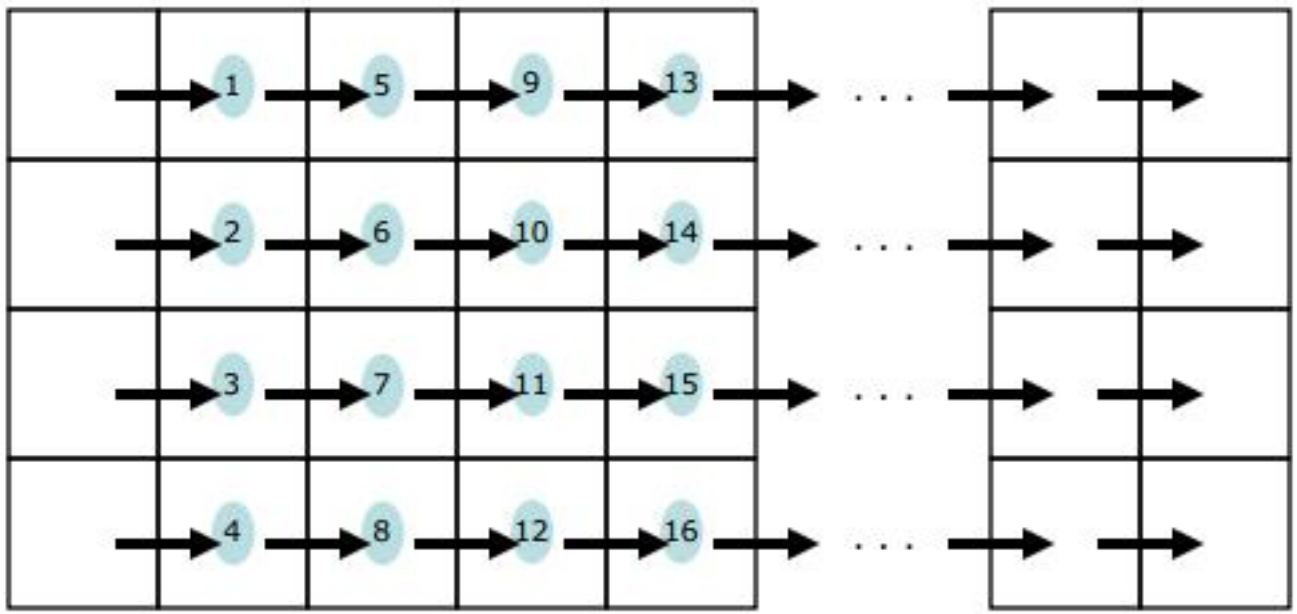
```
float *a, *b, x, y;

…
for (int i = 0; i < N; i++) a[i] = foo(i);
x = a[N-1] – a[0];
for (int i = 0; i < N; i++) b[i] = bar(a[i]);
y = x * b[0] / b[N-1];
```

- **Assume functions *foo* and *bar* are side-effect free**

Intel, "Introduction to Parallel Programming," 2009

Exercises: Loop Transformation (3/3)

```
for (int j =1; j < n; j++)
    for (int i = 0; i < m; i++)
        a[i][j] = 2 * a[i][j-1];
```



Intel, “Introduction to Parallel Programming,” 2009

A Few Parallel Algorithms

- Matrix-vector multiplication
- Parallel depth-first search
  - Work splitting
  - Load balancing: ARR, GRR, RP
- Parallel best-first search
  - Priority queue design
    - Centralized
    - Ring communication
    - Blackboard communication

A Few More Parallel Algorithms

- Parallel SSSP algorithm
- Parallel maximal independent set
- ...

Basic Communication Operations

Operation	MPI Name
One-to-all broadcast	MPI_Bcast
All-to-one reduction	MPI_Reduce
All-to-all broadcast	MPI_Allgather
All-to-all reduction	MPI_Reduce_scatter
All-reduce	MPI_Allreduce
Gather	MPI_Gather
Scatter	MPI_Scatter
All-to-all personalized	MPI_Alltoall

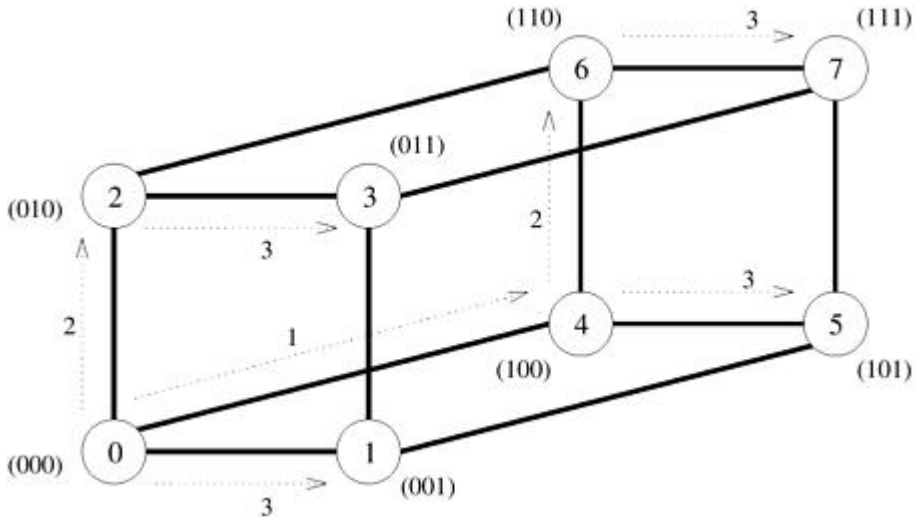
Exercise: One-to-All Broadcast (1/3)

- **Modify the following algorithm, so that it work for any number of processes, not just the power of 2.**
  - **(next slide)**
  - **A one-to-all broadcast procedure on a 2^d-node network when node 0 is the source of the broadcast.**
  - **The procedure is executed at all the nodes. At any node, the value of my_id is the label of that node.**
  - **Let X be the message to be broadcast, which initially resides at the source node 0.**
  - **The procedure performs d communication steps, one along each dimension of a hypothetical hypercube.**

Exercise: One-to-All Broadcast (2/3)

```
1.  procedure ONE_TO_ALL_BC(d, my_id, X)
2.  begin
3.    mask := 2^d - 1;              /* Set all d bits of mask to 1 */
4.    for i := d - 1 downto 0 do    /* Outer loop */
5.      mask := mask XOR 2^i;       /* Set bit i of mask to 0 */
6.      if (my_id AND mask) = 0 then /* If lower i bits of my_id are 0 */
7.        if (my_id AND 2^i) = 0 then
8.          msg_destination := my_id XOR 2^i;
9.          send X to msg_destination;
10.       else
11.         msg_source := my_id XOR 2^i;
12.         receive X from msg_source;
13.       endelse;
14.     endif;
15.   endfor;
16. end ONE_TO_ALL_BC
```

Exercise: One-to-All Broadcast (3/3)



Code Fixing (e.g., CUDA/HIP)

你的朋友用CUDA/HIP实现了矩阵-向量乘法，他的想法是把矩阵按行划分，每个work group计算每行的结果。他为每个work group设置了32个work item，其中第i个work item计算下标模32余i元素的乘法和加法。计算完成后，每行使用归约（reduction）来计算最终结果。

以下是他的主程序和kernel程序。

1) 程序在第15行、第23行、第111行、第112行缺少了代码。请帮他填写缺少的语句。

2) 语句填写后，程序仍然不能正确运行。请帮助他找出并修复错误。在进行修改时，i) 请注明你要修改的地方的行号，ii) 同时请注明错误的原因。

// 在主程序里，我们只列出代码片段。你可以假设此代码片段的前面（包括初始化）和后面的代码都是正确的。如果需要添加kernel代码，请标明所添加代码的前一行和后一行代码的行号。

Good luck!