1-2 Introduction

Lecture 1-2 Introduction

What is this story of parallel computing, multicore, multiprocessing, multi-anything?

Effect of Moore's law

Performance increase per year:

- 1986 - 2002 → 50% performance increase
- Since 2002 → 20% performance increase

Power Density

Power needed per chip is:

- Power consumed: for the processor to do the calculations needed to execute your programs.
- Power dissipated: in the form of temperature.

Multicore Processors Save Power

$$\text{Power} = C * V^2 * F \qquad \text{Performance} = \text{Cores} * F$$

(C = capacitance V= voltage F = frequency)

Let's have two cores

$$\text{Power} = 2*C * V^2 * F \qquad \text{Performance} = 2*\text{Cores} * F$$

But decrease frequency by 50% (Note that $V \alpha F$)

$$\text{Power} = 2*C * V^2/4 * F/2 \qquad \text{Performance} = 2*\text{Cores} * F/2$$

$$\text{Power} = C * V^2/4 * F \qquad \text{Performance} = \text{Cores} * F$$

- Using the transistors per chip to build more cores + lowering the frequency (i.e. clock) of each core → better performance with lower power
- More cores with slower frequency is better than one big fat core with higher frequency, if we can make good use of them.

Parallel computing: using multiple processors (or cores) in parallel to solve problems more quickly than with a single processor/core.

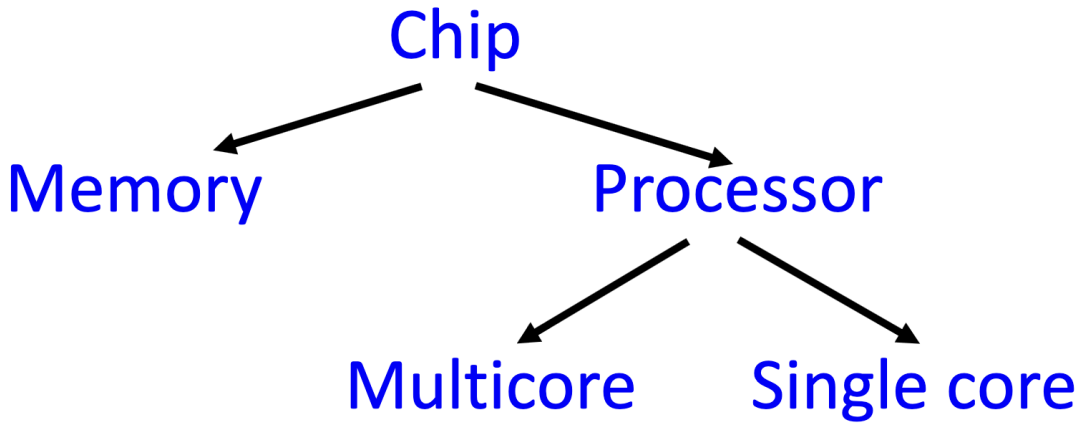
Examples of parallel machines:

A cluster computer that contains multiple PCs combined together with a high speed network

A shared memory multiprocessor (SMP) by connecting multiple processors to a single memory system

A Chip Multi-Processor (i.e. multicore) (CMP) contains multiple processors (called cores) on a single chip

Some terms: (core = CPU)



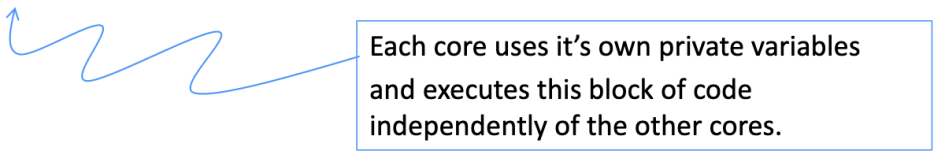
Example: Compute n values and add them together.

Serial solution:

```
sum = 0;
for (i = 0; i < n; i++) {
   x = Compute_next_value(. . .);
   sum += x;
}
```

We have p cores, p much smaller than n.
Each core performs a partial sum of approximately n/p values.

```
my_sum = 0;
my_first_i = . . . ;
my_last_i = . . . ;
for (my_i = my_first_i; my_i < my_last_i; my_i++) {
   my_x = Compute_next_value( . . .);
   my_sum += my_x;
}
```



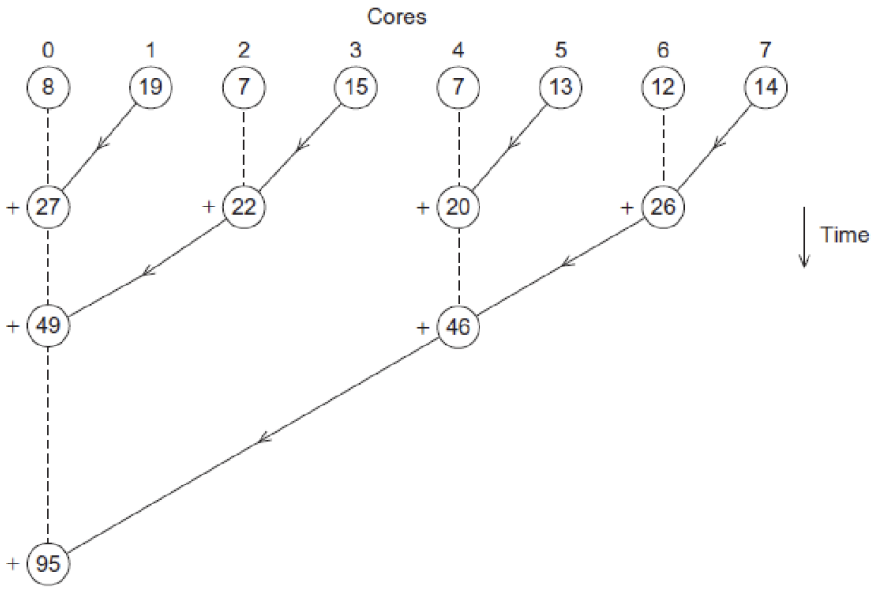
Once all the cores are done computing their private my_sum, they form a global sum by sending results to a designated "master" core which adds the final result.

```
if (I'm the master core) {
   sum = my_x;
   for each core other than myself {
      receive value from core;
      sum += value;
   }
} else {
   send my_x to the master;
}
```

Better parallel algorithm

Don't make the master core do all the work. Share it among the other cores.

- Pair the cores so that core 0 adds its result with core 1's result. Core 2 adds its result with core 3's result, etc.
- Work with odd and even numbered pairs of cores.
- Repeat the process now with only the evenly ranked cores. Core 0 adds result from core 2. Core 4 adds the result from core 6, etc.
- Now cores divisible by 4 repeat the process, and so forth, until core 0 has the final result



Analysis

- In the first version, the master core performs 7 receives and 7 additions.
- In the second version, the master core performs 3 receives and 3 additions.

Two Ways Of Thinking and one Strategy!

Strategy: Partitioning!

Two ways of thinking:

- Task-parallelism
- Data-parallelism