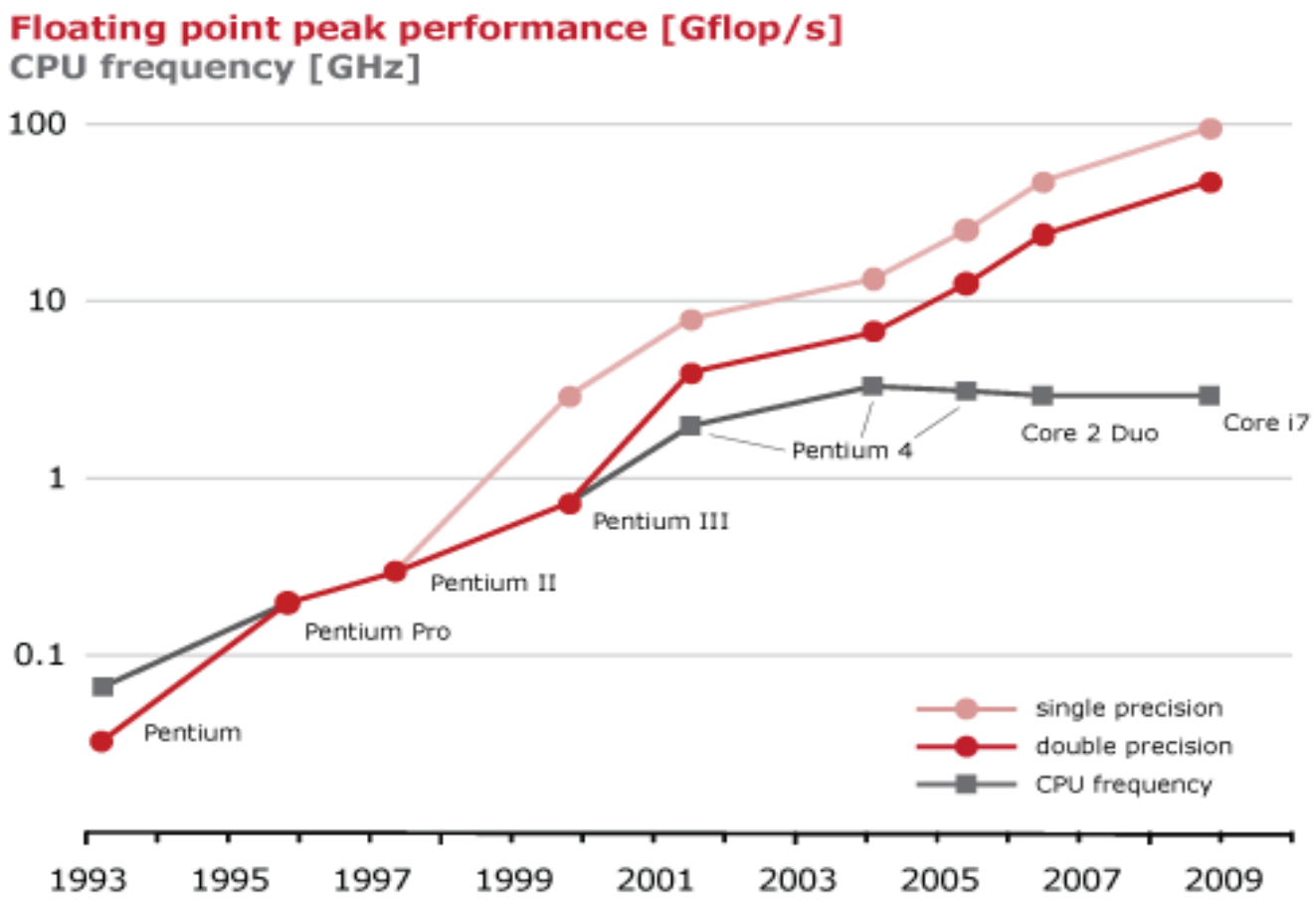
Lab:
Introduction to Parallelism

"*The era of sequential computing must give way to a new era in which parallelism is at the forefront*"

- National Research Council (2011)

Clients: Intel microprocessor performance



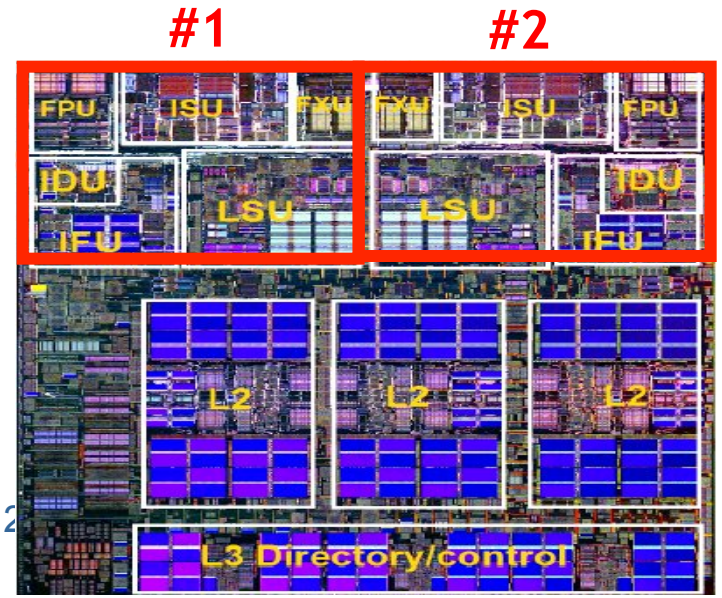
Knights Ferry
MIC co-processor

(Graph from Markus Püschel, ETH)

Why parallelism now?

If frequency does not increase, how are we going to build processor chips that run faster than previous generations?

Increase the number of cores

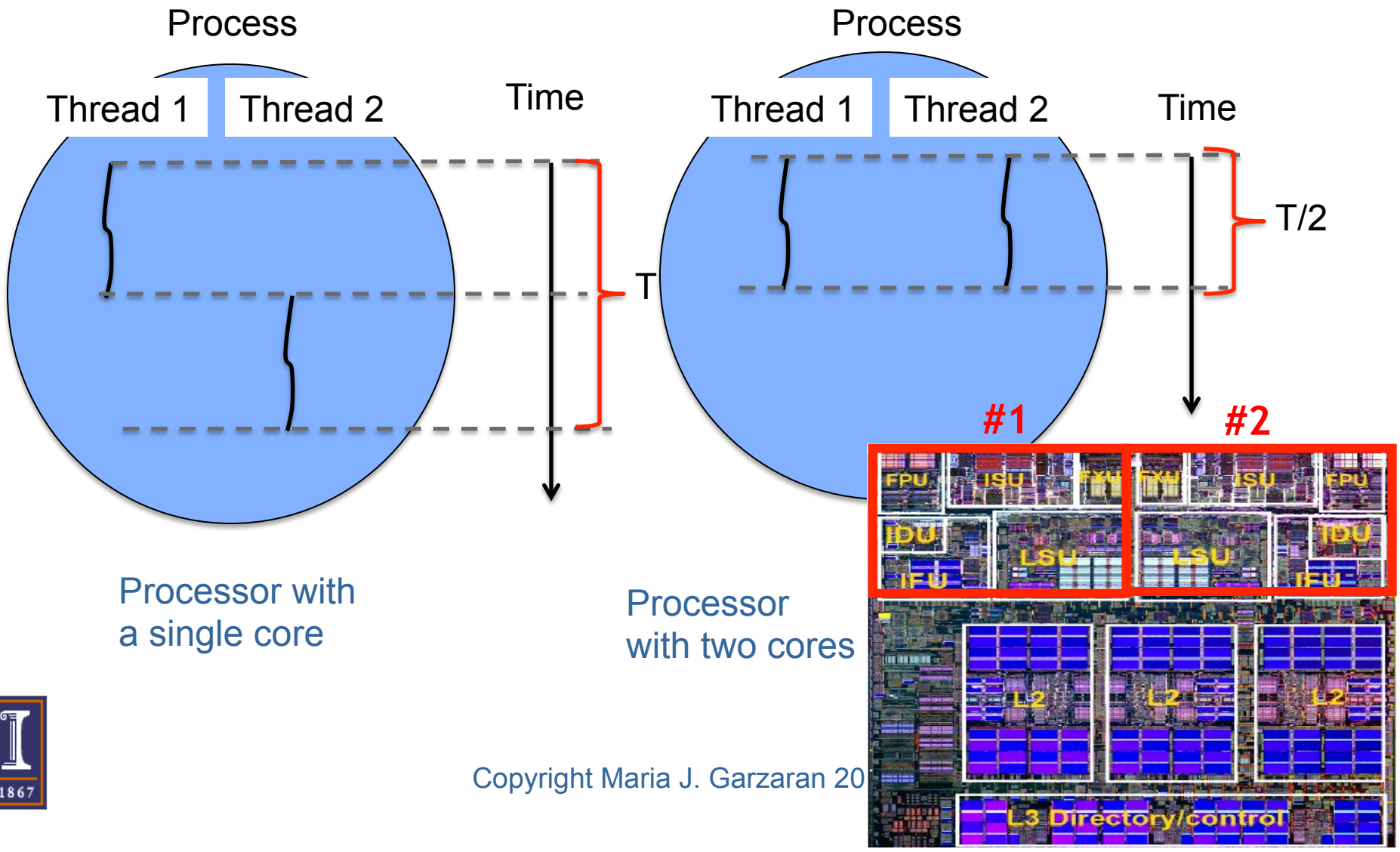


Why parallelism now?

"*Parallel computing is a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently (in parallel)*"

From Wikipedia

How can we execute a program in parallel?

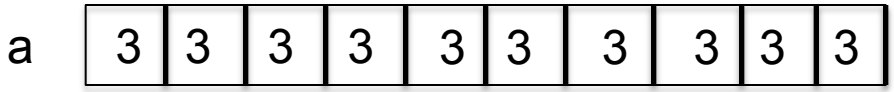


Processor with a single core

Processor with two cores

How can we execute a program in parallel?

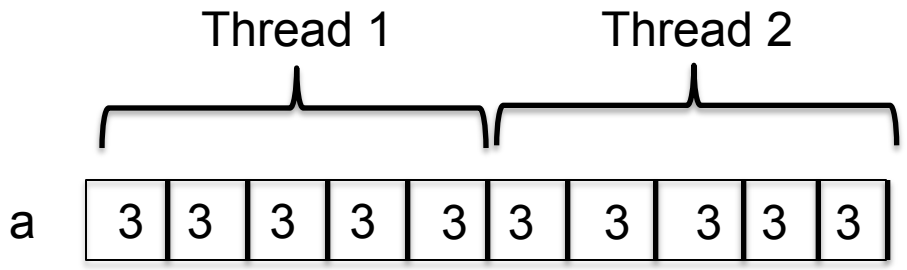
```
for (int i=0; i<10; i++)
    a[i] = a[i] + 1;
```



Sequential execution

How can we execute a program in parallel?

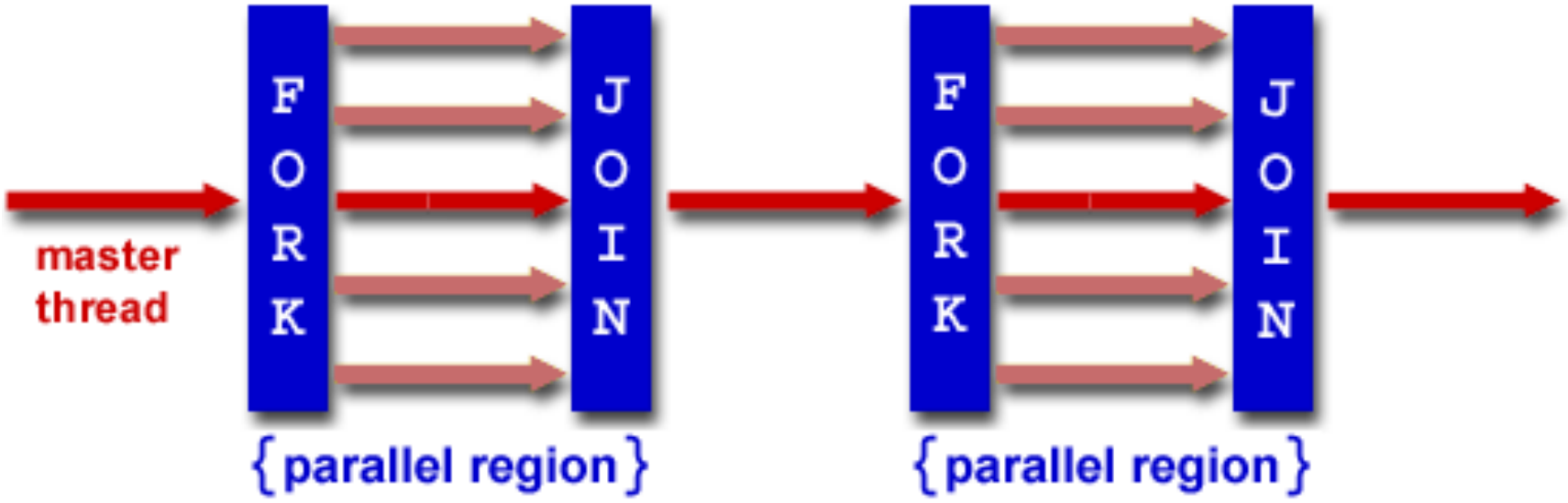
```
for (int i=0; i<10; i++)
    a[i] = a[i] + 1;
```



Parallel execution

OpenMP

- OpenMP is a collection of compiler directives, library routines, and environment variables that can be used to specify shared memory parallelism
- Based on the fork/join model



Source: Craig Zilles CS232

OpenMP parallel construct

- parallel construct starts parallel execution

```
#include <iostream>
using namespace std;
#include <omp.h>

int main() {
#pragma omp parallel
      cout << "Hello" << endl;
}
```

- Compiles with:
  - icc –openmp hello.cpp –o hello

  or
  - g++ –fopenmp hello.cpp –o hello

- A sample run with 4 threads:

[user@linux4 ~]./hello

Hello
Hello
Hello
Hello

OpenMP parallel construct

```
#include <iostream>
using namespace std;
#include <omp.h>

int main() {
   omp_set_num_threads(5);
   #pragma omp parallel
   cout << "Hello" << endl;
}
```

```
[xx@linux4 ~]$./hello
hellohellohello
hello

hello
```

OpenMP parallel for construct

`for` loops are often parallelized, so we have a construct just for them!

(Why are `for` loops often parallelized?)

```
#pragma omp parallel for
for (int i=0; i<n; i++)
    A[i] = B[i] + C[i];
```

If 2 threads execute this loop,
- thread 1 executes iterations [0 .. (n/2)-1]
- thread 2 executes iterations [(n/2) .. n-1]

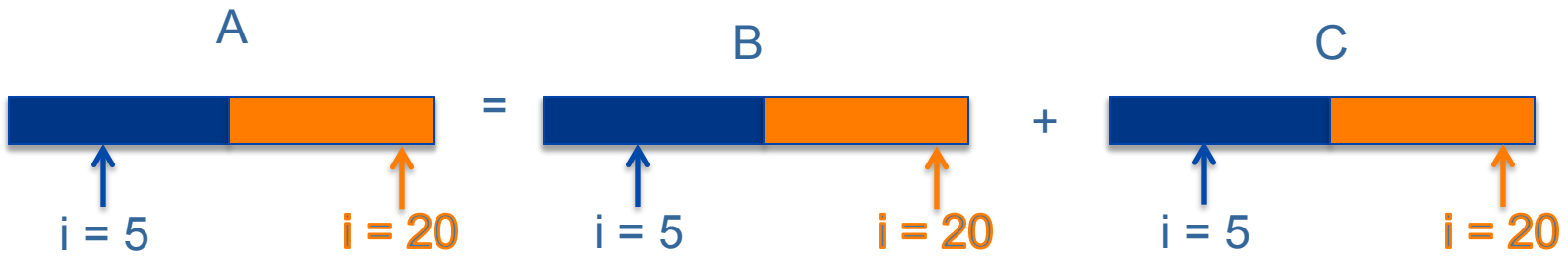
OpenMP shared and private variables

A simple example:

```
double * A = new double[n];
double * B = new double[n];
double * C = new double[n];

#pragma omp parallel for
for (int i=0; i<n; i++)
    A[i] = B[i] + C[i];
```

Variables declared inside a parallel `for` are private to each thread.



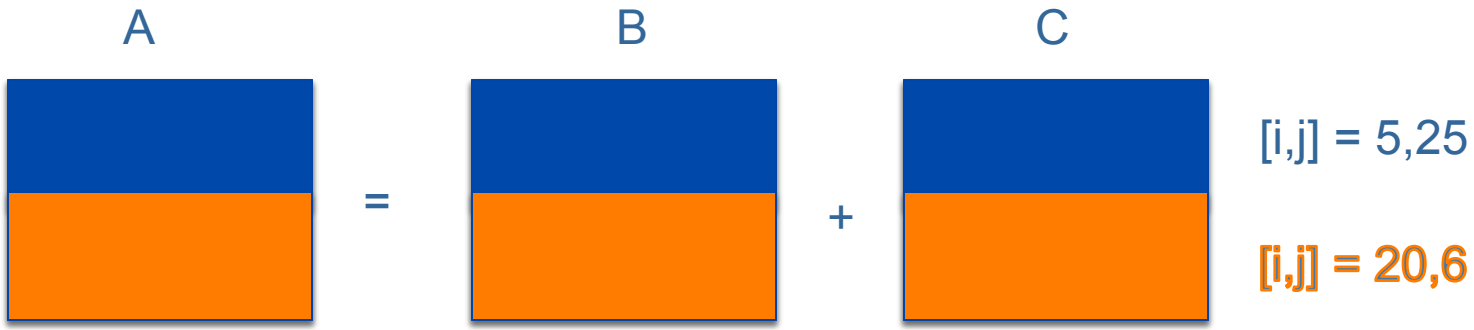
OpenMP shared and private variables

Another example:

```
double **A = Allocate2DArray< double >(n,m);
double **B = Allocate2DArray< double >(n,m);
double **C = Allocate2DArray< double >(n,m);

#pragma omp parallel for
for (int i=0; i<n; i++)
   for (int j=0; j<m; j++)
       A[i][j] = B[i][j] + C[i][j];
```

We parallelize the outer loop.



Speedup

How do we measure improvement from parallelization?

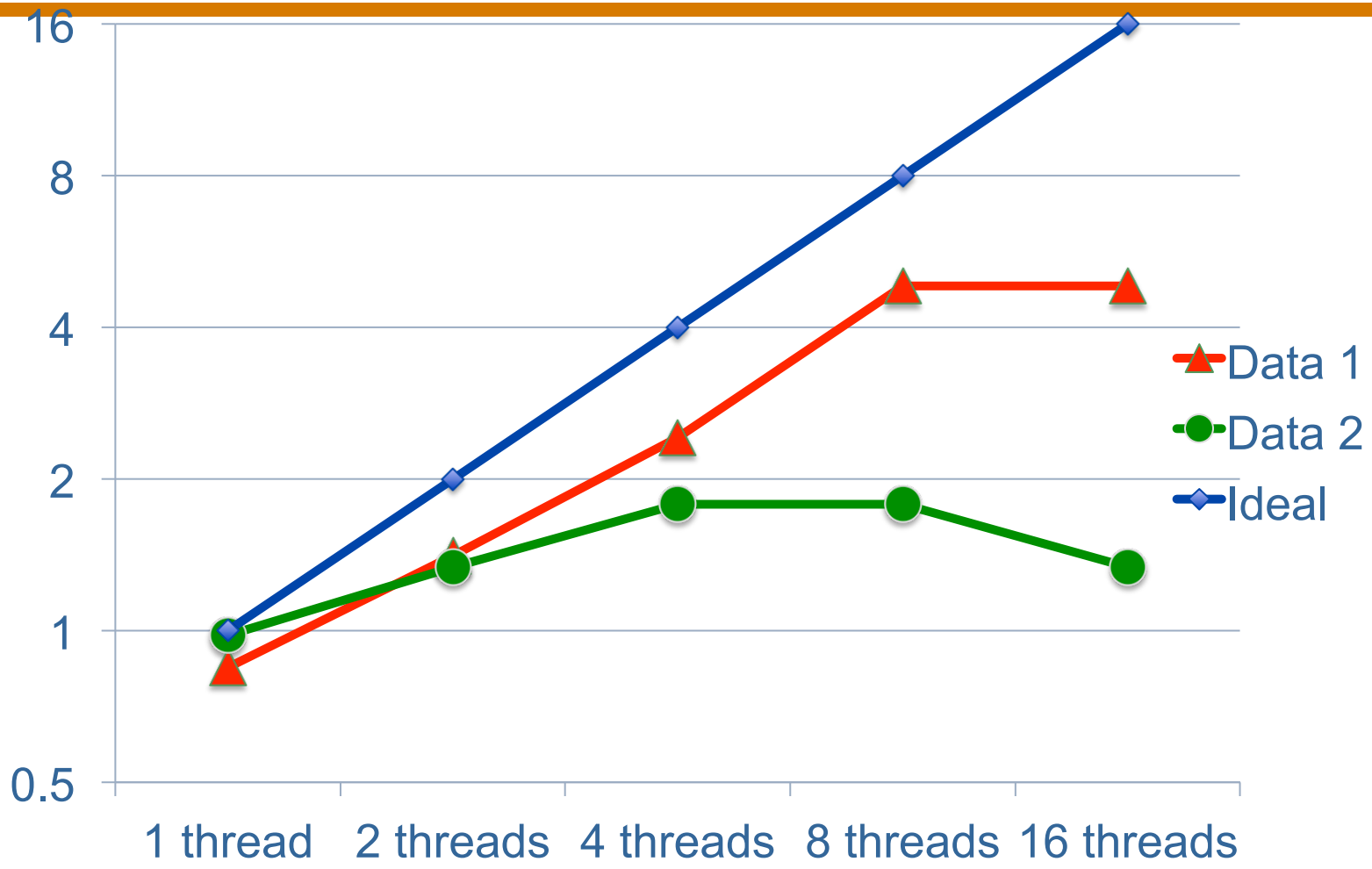
$$\text{Speedup} = \frac{\text{Time for best serial execution}}{\text{Time for version with p physical threads}}$$

Example 1:
- Time serial = 18 seconds
- Time parallel with 2 threads = 10 seconds
- Speedup = 18 / 10 = 1.8

Example 2:
- Time serial = 18 seconds
- Time parallel with 8 threads = 10 seconds
- Speedup = 18 / 10 = 1.8

Example 3:
- Time serial = 18 seconds
- Time parallel with 4 threads = 2 seconds
- Speedup = 18 / 2 = 9

Speedup



Amdahl's Law

- Suppose a program takes 1 unit of time to execute serially.
- Part of the program is inherently serial (cannot be parallelized)
  - s = fraction of time spent in serial portion
  - (1- s) = remaining parallelizable portion