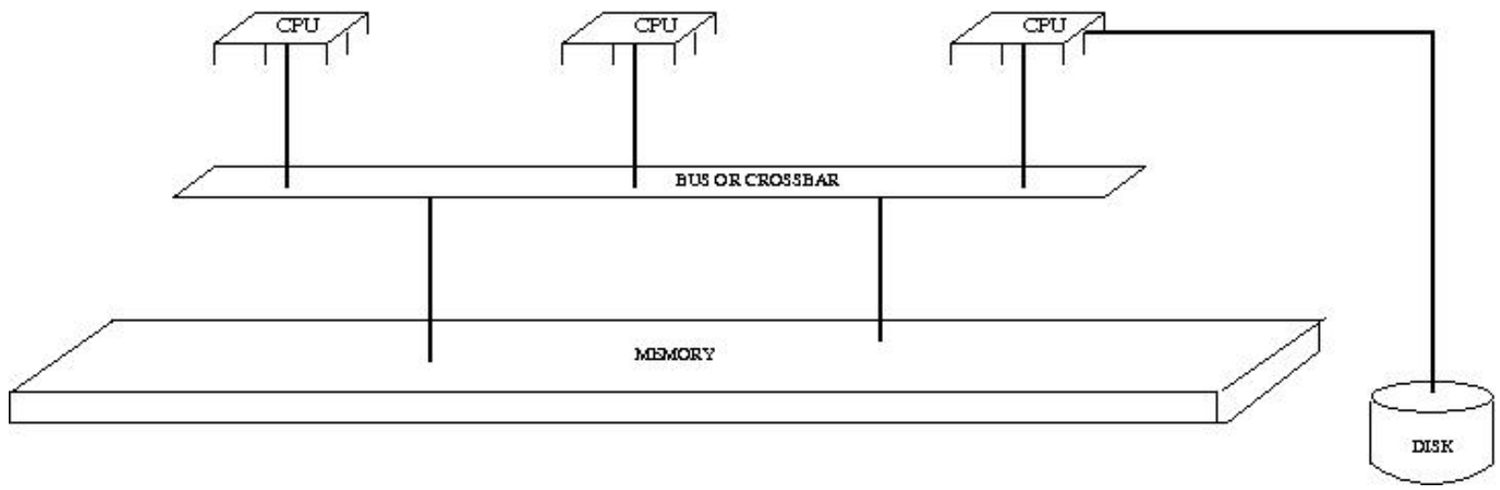
Introduction to Parallel Programming

Shared-Memory Processing

Each processor can access the entire data space

- Pro's
  - Easier to program
  - Amenable to automatic parallelism
  - Can be used to run large memory serial programs
- Con's
  - Expensive
  - Difficult to implement on the hardware level
  - Processor count limited by contention/coherency (currently around 512)
  - Watch out for "NU" part of "NUMA"

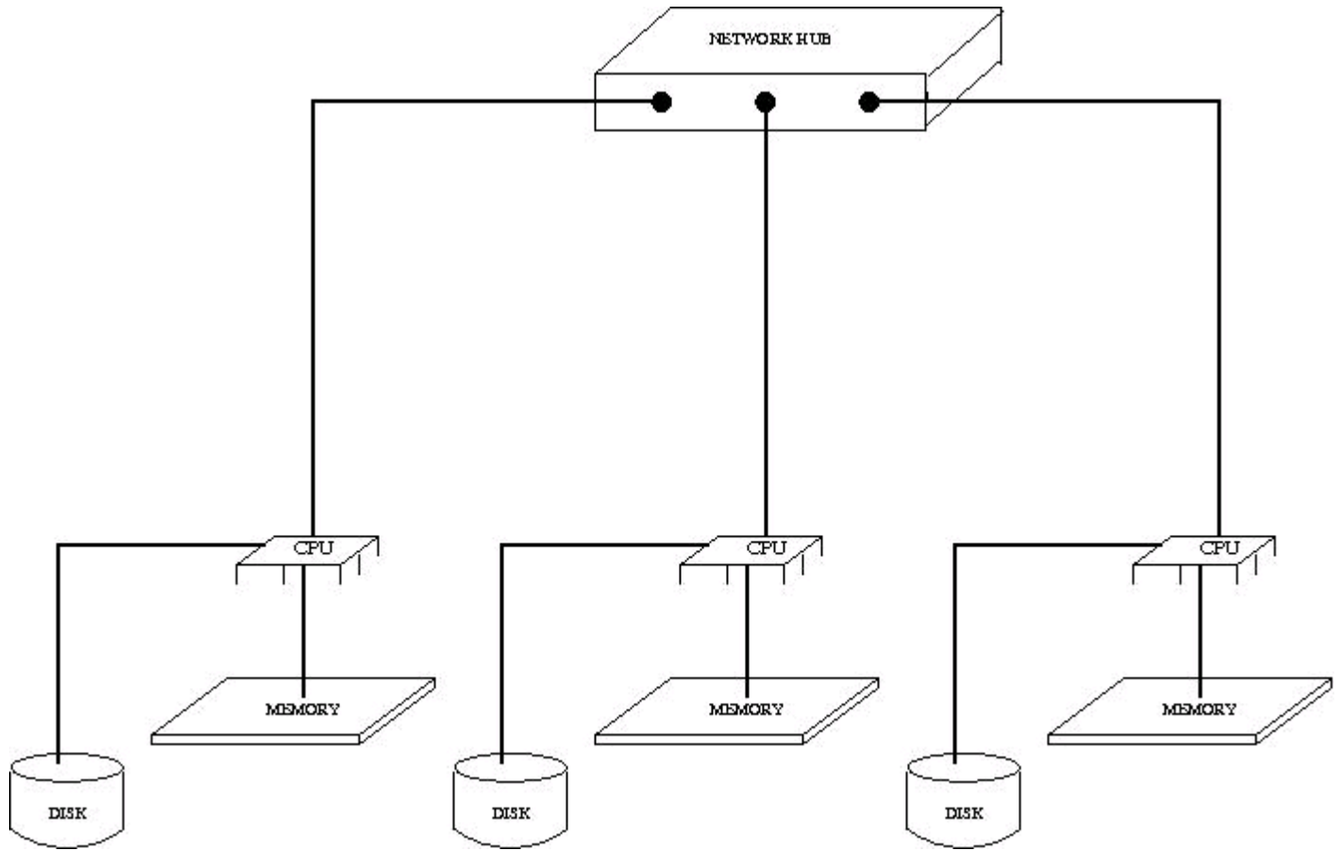


Distributed – Memory Machines

- Each node in the computer has a locally addressable memory space
- The computers are connected together via some high-speed network
  - Infiniband, Myrinet, Giganet, etc..

- Pros
  - Really large machines
  - Size limited only by gross physical considerations:
    - Room size
    - Cable lengths (10's of meters)
    - Power/cooling capacity
    - Money!
  - Cheaper to build and run
- Cons
  - Harder to program

    Data Locality



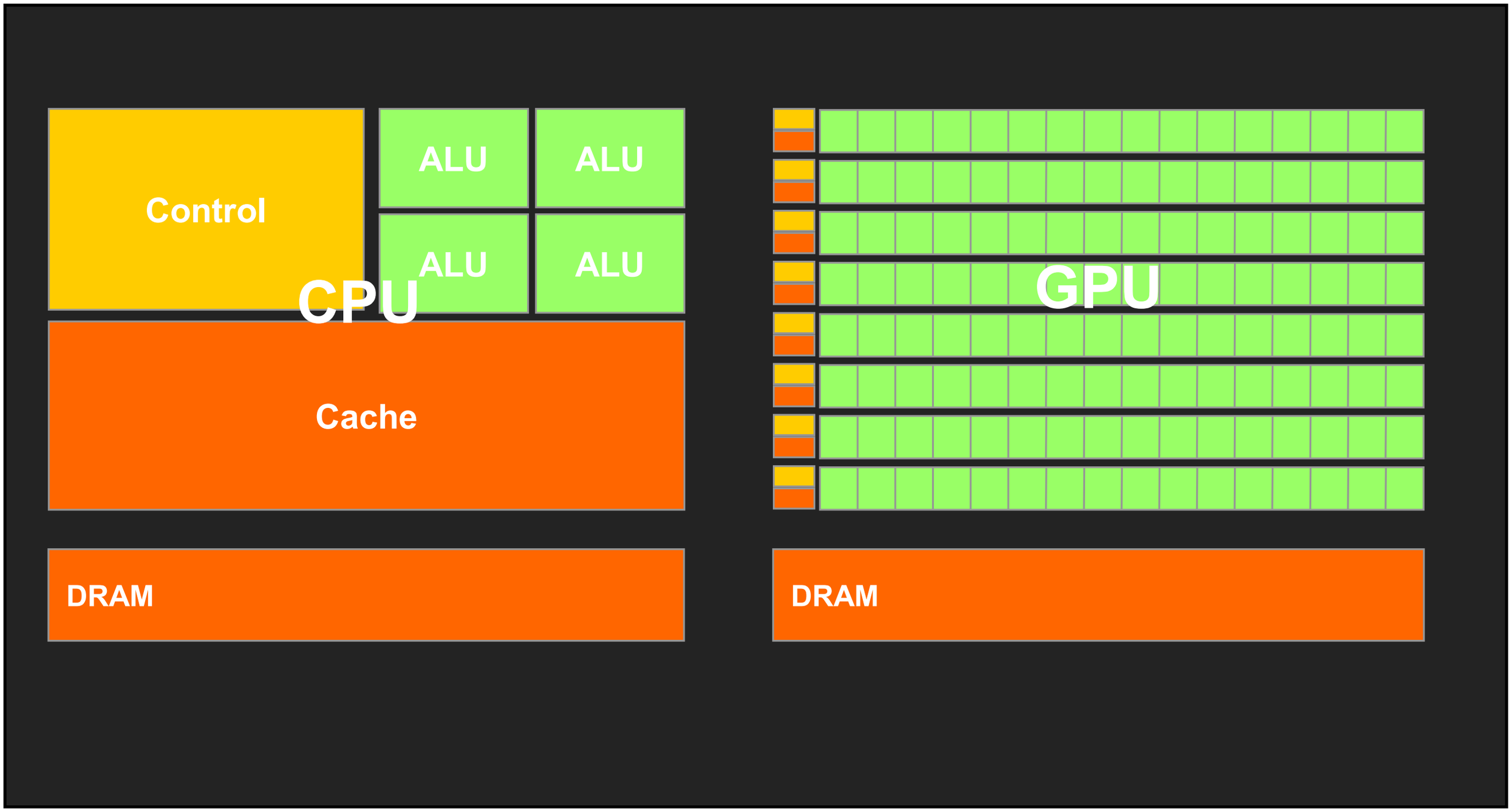
MPPs (Massively Parallel Processors)

Distributed memory at largest scale. Often shared memory at lower hierarchies.

- IBM BlueGene/L (LLNL)
  - 131,072 700 Mhz processors
  - 256 MB or RAM per processor
  - Balanced compute speed with interconnect

- Red Storm (Sandia National Labs)
  - 12,960 Dual Core 2.4 Ghz Opterons
  - 4 GB of RAM per processor
  - Proprietary SeaStar interconnect

Comparison of CPU vs GPU Architecture



Source: Prof. Wen-mei W. Hwu UIUC

GPU CPU Analogy

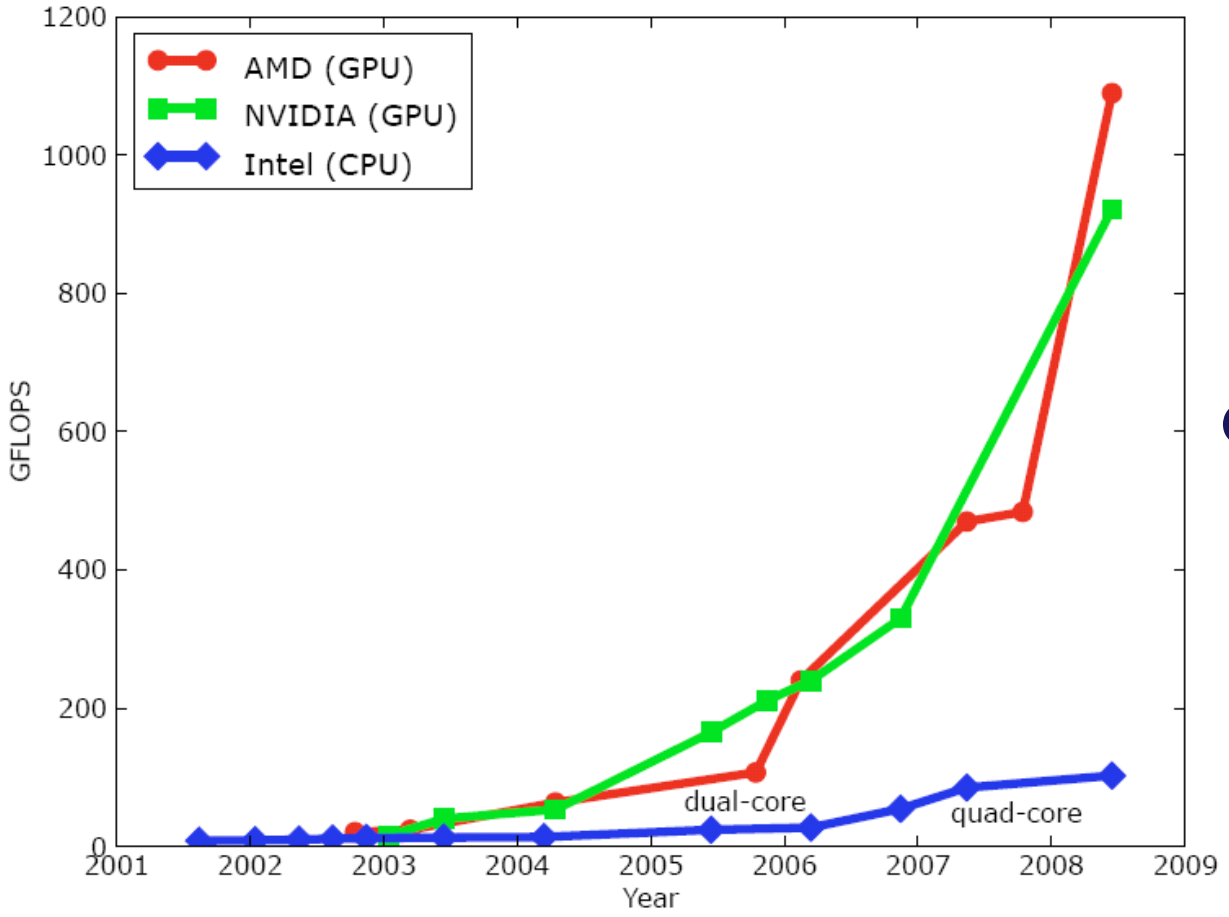
GPU CPU

It is more effective to deliver Pizza's through light duty scooters rather than big truck. Similarly effective to use several lightweight GPU processors for parallel tasks.

GPU Performance

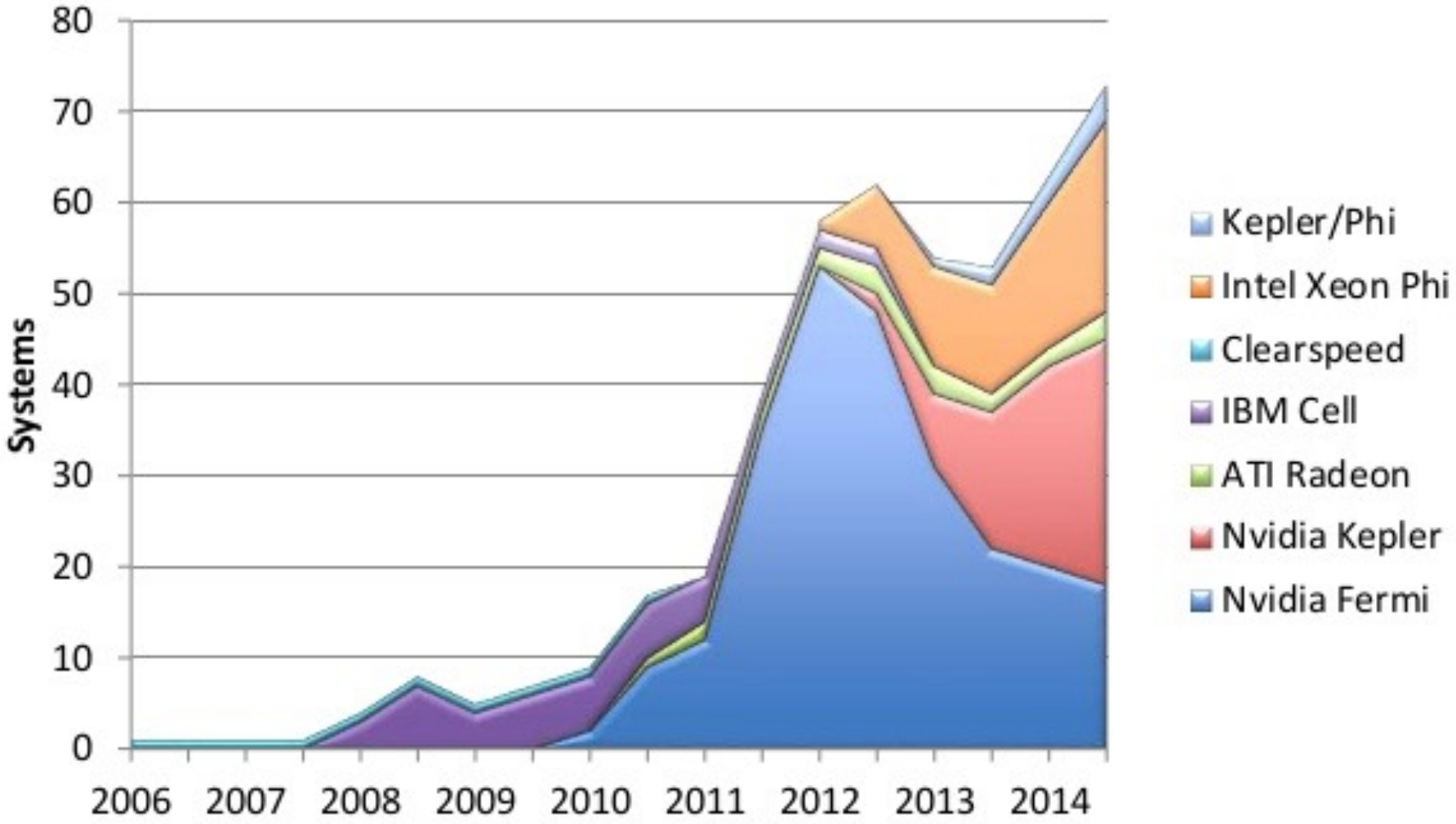
Peak performance increase

- Calculation ~ 1 TFlop on Desktop
- Memory Bandwidth ~ 150 GB/s



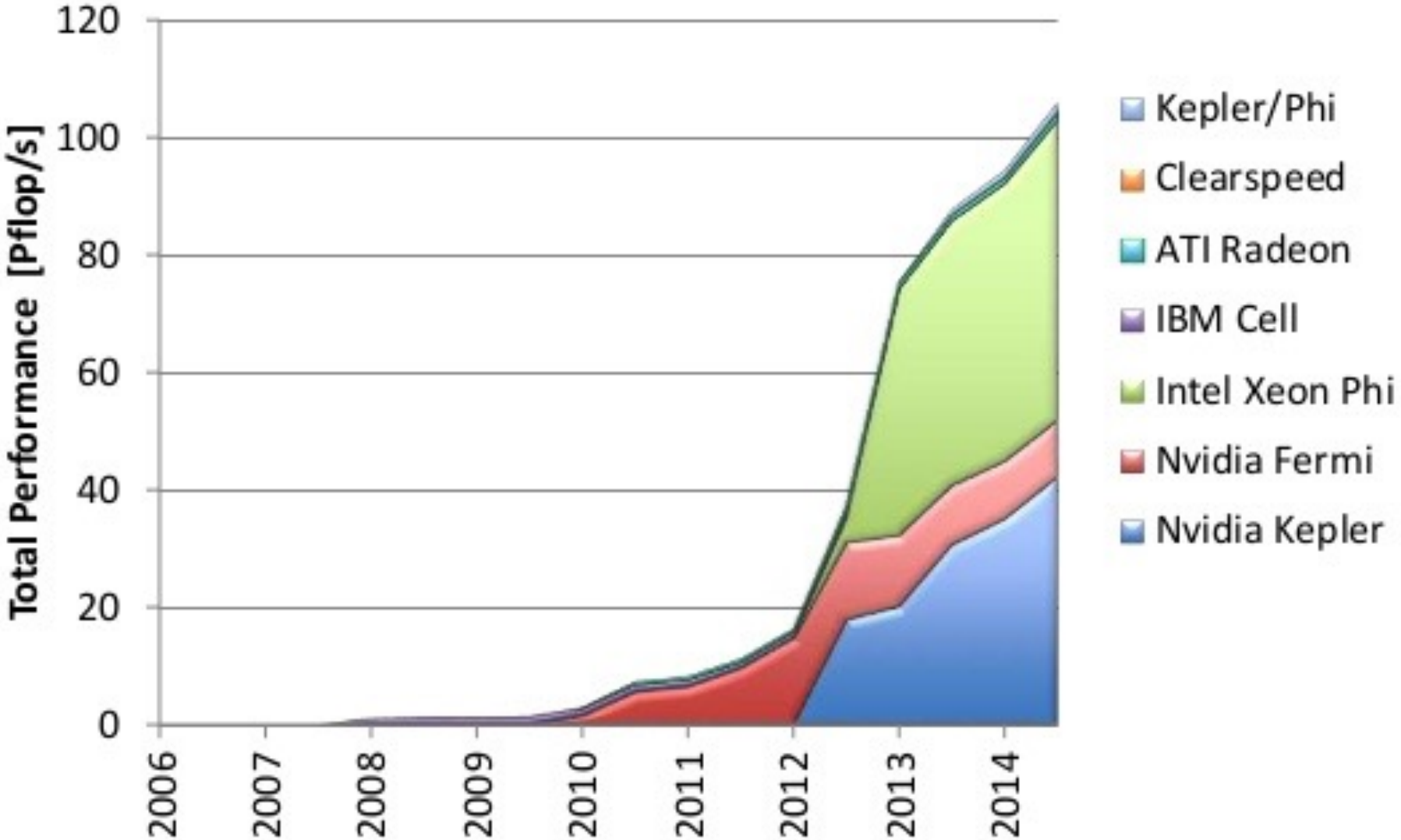
Courtesy: John Owens

ACCELERATORS



source: top500.org

PERFORMANCE OF ACCELERATORS

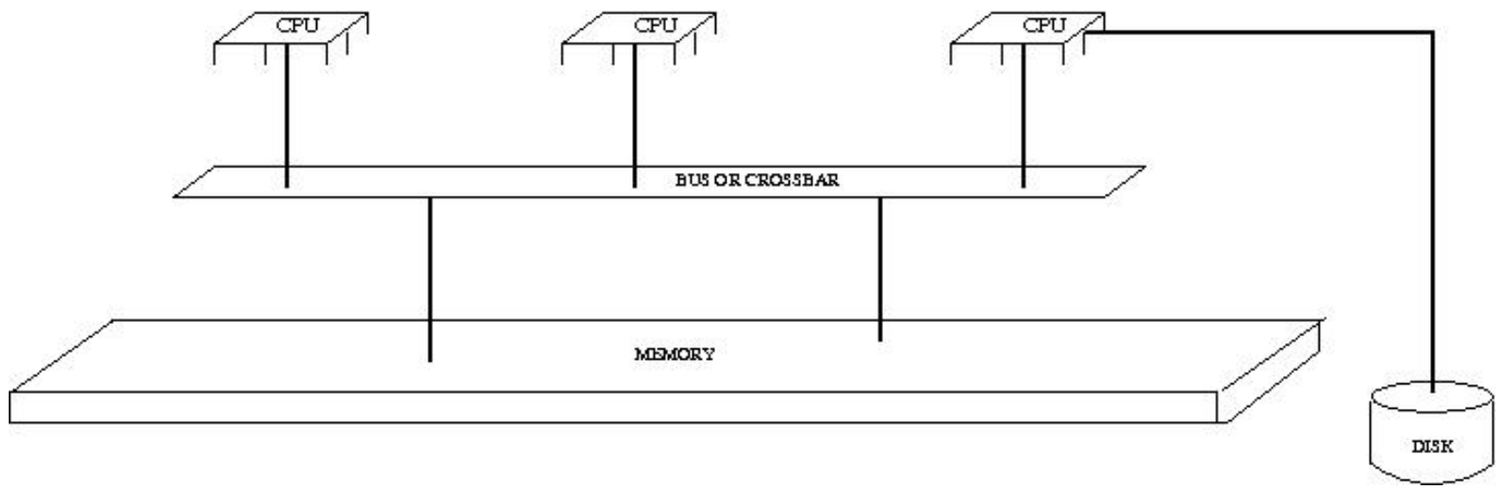


source: top500.org

Shared-Memory Processing

Each processor can access the entire data space

- Pro's
  - Easier to program
  - Amenable to automatic parallelism
  - Can be used to run large memory serial programs
- Con's
  - Expensive
  - Difficult to implement on the hardware level
  - Processor count limited by contention/coherency (currently around 512)
  - Watch out for "NU" part of "NUMA"



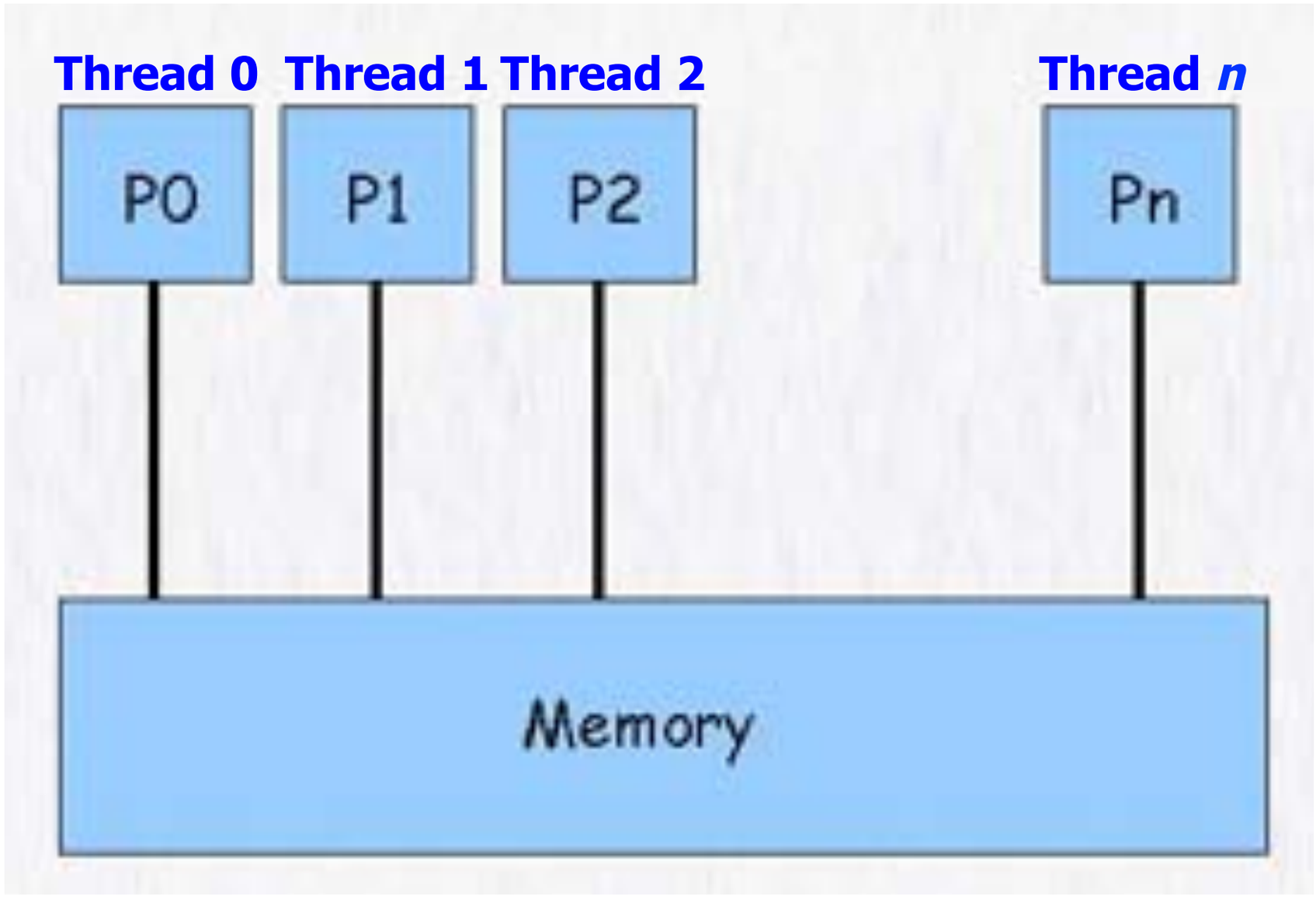
Introduction to OpenMP

Introduction

- OpenMP is designed for shared memory systems.
- OpenMP is easy to use
  - achieve parallelism through compiler directives
  - or the occasional function call
- OpenMP is a “quick and dirty” way of parallelizing a program.
- OpenMP is usually used on existing serial programs to achieve moderate parallelism with relatively little effort

Computational Threads

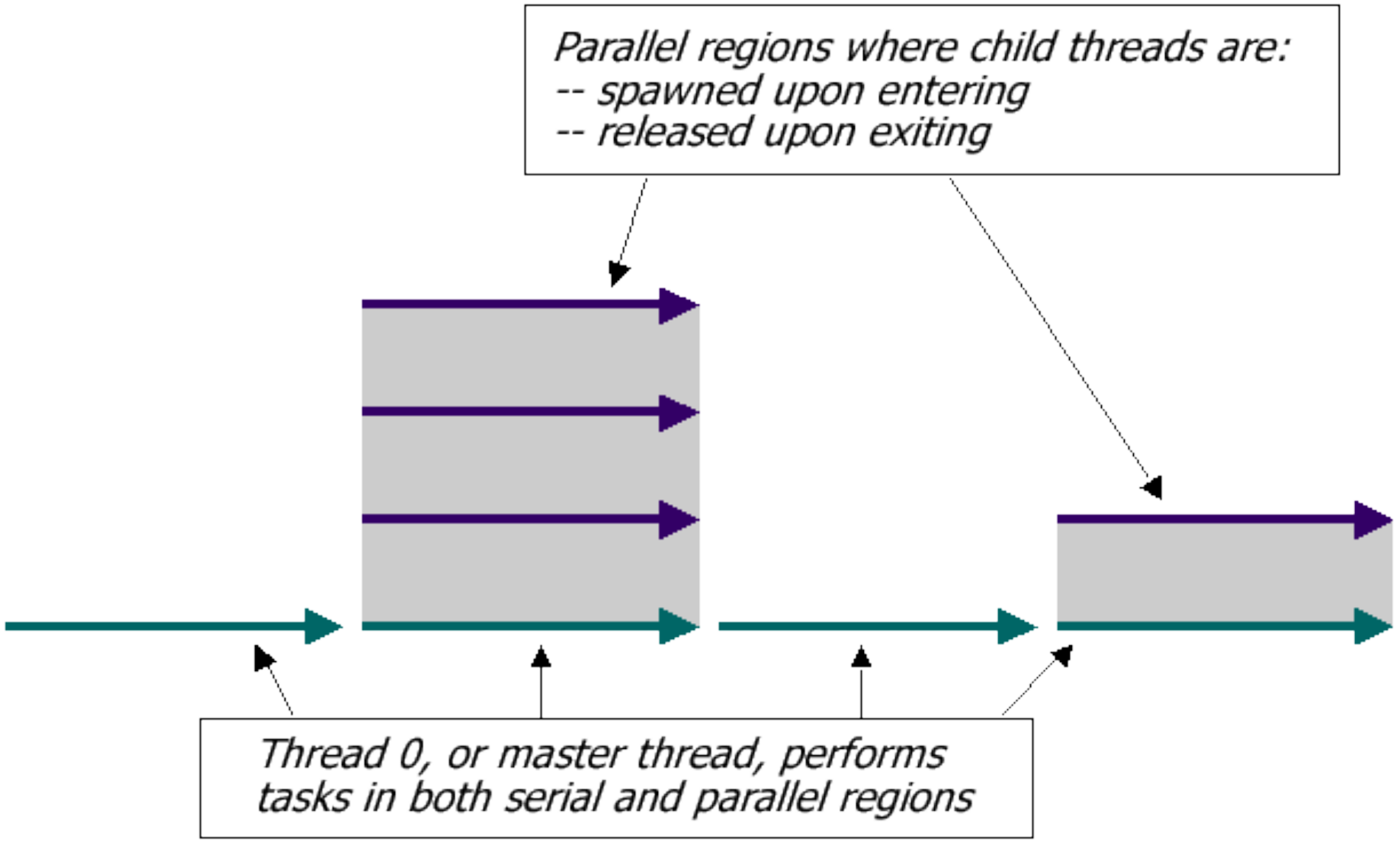
- Each processor has one thread assigned to it
- Each thread runs one copy of your program



OpenMP Execution Model

- In OpenMP, execution begins only on the master thread. Child threads are spawned and released as needed.
  - Threads are spawned when program enters a parallel region.
  - Threads are released when program exits a parallel region

OpenMP Execution Model



Parallel Region Example:
For loop

Fortran:

```
!$omp parallel do
do i = 1, n
 a(i) = b(i) + c(i)
enddo
```

C/C++:

```
#pragma omp parallel for
for(i=1; i<=n; i++)
   a[i] = b[i] + c[i];
```

This comment or pragma tells openmp compiler to spawn threads *and* distribute work among those threads

These actions are combined here but they can be specified separately between the threads

Pros of OpenMP

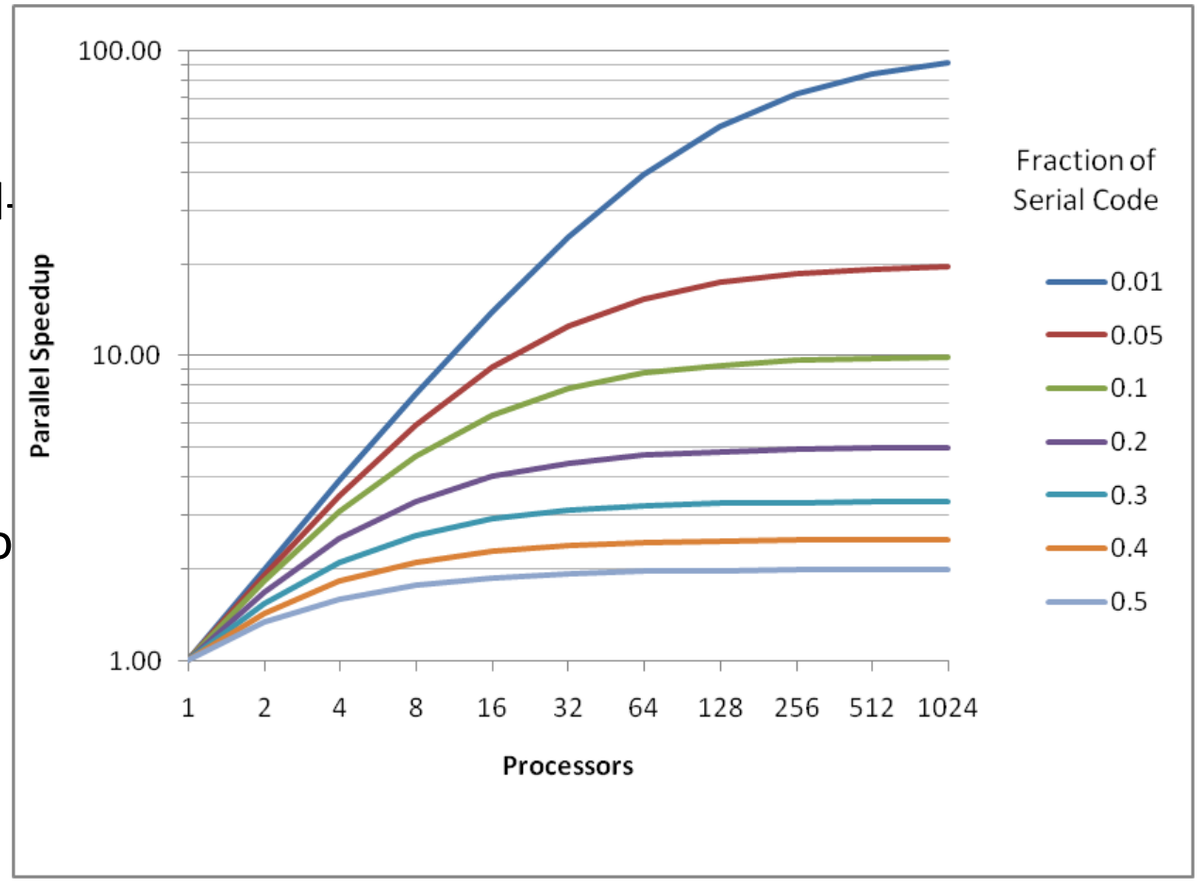
- Because it takes advantage of shared memory, the programmer does not need to worry (that much) about data placement
- Programming model is “serial-like” and thus conceptually simpler than message passing
- Compiler directives are generally simple and easy to use
- Legacy serial code does not need to be rewritten

Cons of OpenMP

- Codes can only be run in shared memory environments!
  - In general, shared memory machines beyond ~8 CPUs are much more expensive than distributed memory ones, so finding a shared memory system to run on may be difficult
- Compiler must support OpenMP
  - whereas MPI can be installed anywhere
  - However, gcc 4.2 now supports OpenMP

Cons of OpenMP

- In general, only moderate speedups can be achieved.
  - Because OpenMP codes tend to have serial-only portions, Amdahl's Law prohibits substantial speedups



- Amdahl's Law:

  F = Fraction of serial execution time that cannot be parallelized

  N = Number of processors

$$\text{Execution time} = \frac{1}{F + (1 - F)/N}$$

If you have big loops that dominate execution time, these are ideal targets for OpenMP

Compiling and Running OpenMP

- True64: **-mp**
- SGI IRIX: **-mp**
- IBM AIX: **-qsmp=omp**
- Portland Group: **-mp**
- Intel: **-openmp**
- gcc (4.2) **-fopenmp**

Compiling and Running OpenMP

- OMP_NUM_THREADS environment variable sets the number of processors the OpenMP program will have at its disposal.
- Example script

```
#!/bin/tcsh
setenv OMP_NUM_THREADS 4
mycode < my.in > my.out
```