Introduction to High Performance Computing

Gregory G. Howes
Department of Physics and Astronomy
University of Iowa

PHYS 5905: Numerical Simulation of Plasmas
Department of Physics and Astronomy
University of Iowa
Spring 2019

Thank you

THE UNIVERSITY OF IOWA

This presentation borrows heavily from information freely available on the web by Ian Foster and Blaise Barney (see references)

Outline

- Introduction
- Thinking in Parallel
- Parallel Computer Architectures
- Parallel Programming Models
- References

Introduction

THE UNIVERSITY OF IOWA

Disclaimer: High Performance Computing (HPC) is valuable to a variety of applications over a very wide range of fields. Many of my examples will come from the world of physics, but I will try to present them in a general sense

Why Use Parallel Computing?

- Single processor speeds are reaching their ultimate limits

- Multi-core processors and multiple processors are the most promising paths to performance improvements

Definition of a parallel computer:

A set of independent processors that can work cooperatively to solve a problem.

Introduction

THE UNIVERSITY OF IOWA

The March towards Petascale Computing

- Computing performance is defined in terms of FLoating-point OPerations per Second (FLOPS)

GigaFLOP $1\ \mathrm{GF} = 10^9\ \mathrm{FLOPS}$

TeraFLOP $1\ \mathrm{TF} = 10^{12}\ \mathrm{FLOPS}$

PetaFLOP $1\ \mathrm{PF} = 10^{15}\ \mathrm{FLOPS}$

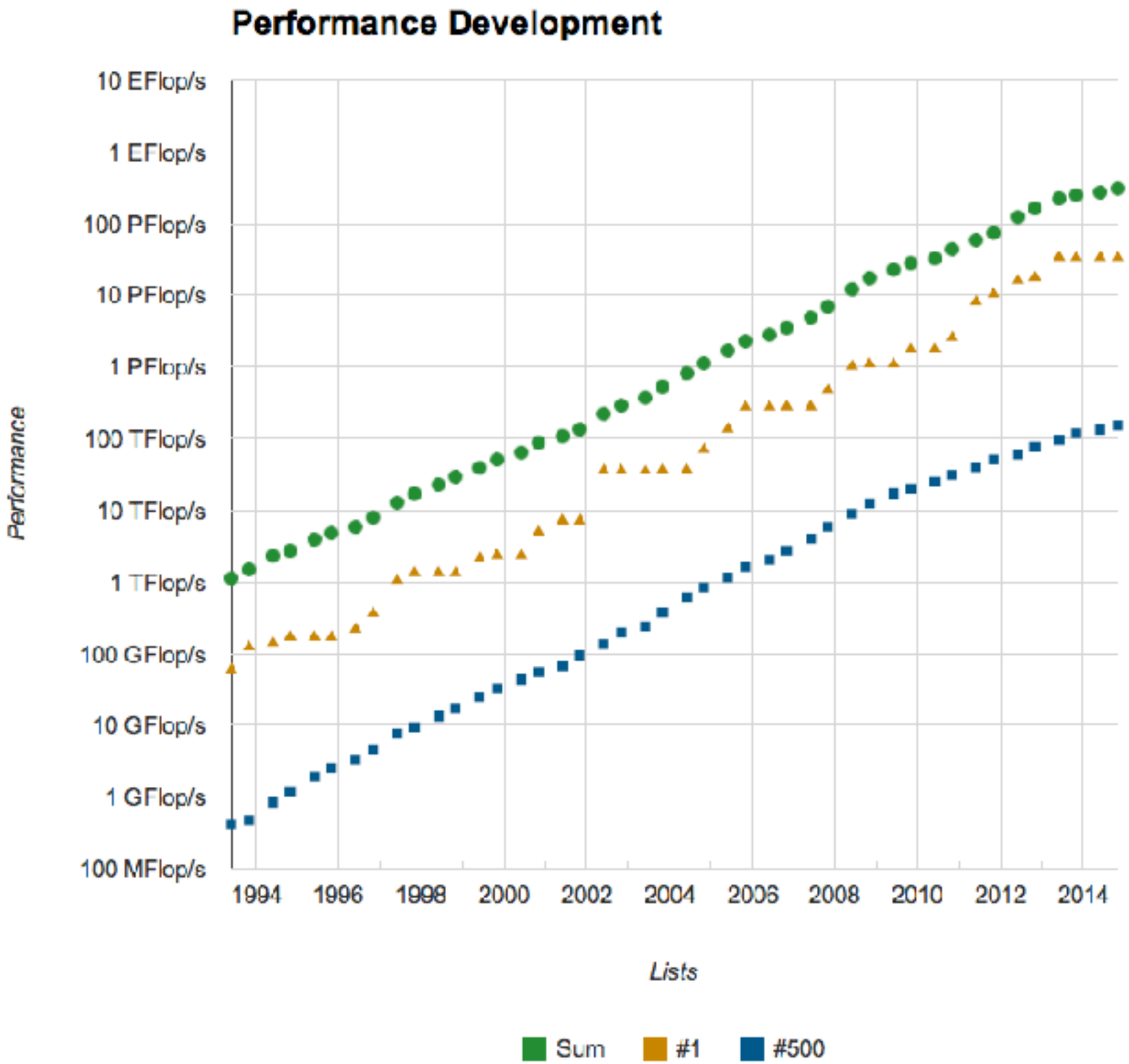
- Petascale computing also refers to extremely large data sets

PetaByte $1\ \mathrm{PB} = 10^{15}\ \mathrm{Bytes}$

Introduction

THE UNIVERSITY OF IOWA

Performance improves by factor of ~10 every 4 years!



Outline

- Introduction
- Thinking in Parallel
- Parallel Computer Architectures
- Parallel Programming Models
- References

Thinking in Parallel

THE UNIVERSITY OF IOWA

DEFINITION Concurrency: The property of a parallel algorithm that a number of operations can be performed by separate processors at the same time.

Concurrency is the key concept in the design of parallel algorithms:

- Requires a different way of looking at the strategy to solve a problem
- May require a very different approach from a serial program to achieve high efficiency

Thinking in Parallel

THE UNIVERSITY OF IOWA

DEFINITION Scalability: The ability of a parallel algorithm to demonstrate a speedup proportional to the number of processors used.

DEFINITION Speedup: The ratio of the serial wallclock time to the parallel wallclock time required for execution.

$$S = \frac{\text{wallclock time}_{serial}}{\text{wallclock time}_{parallel}}$$

• An algorithm that has good scalability will take half the time with double the number of processors

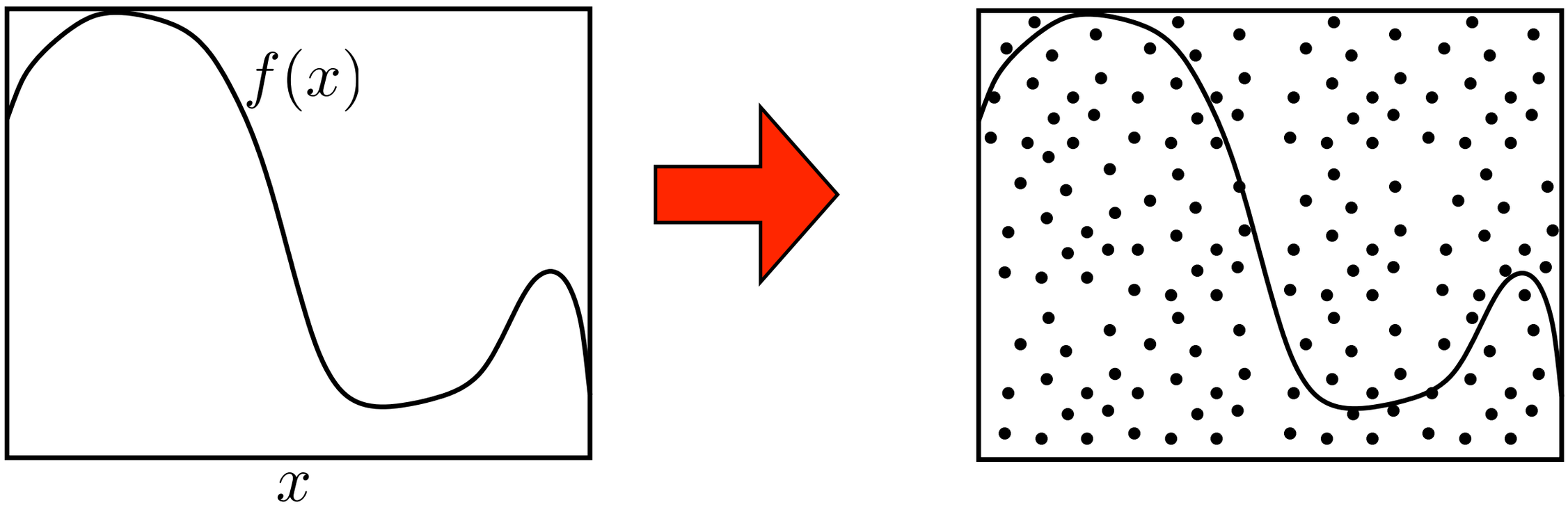
• Parallel Overhead, the time required to coordinate parallel tasks and communicate information between processors, degrades scalability.

Example: Numerical Integration

THE UNIVERSITY OF IOWA

Numerical Integration: Monte Carlo Method

- Choose N points within the box of total area A
- Determine the number of points n falling below $f(x)$
- Integral value is $I = \frac{n}{N} A$



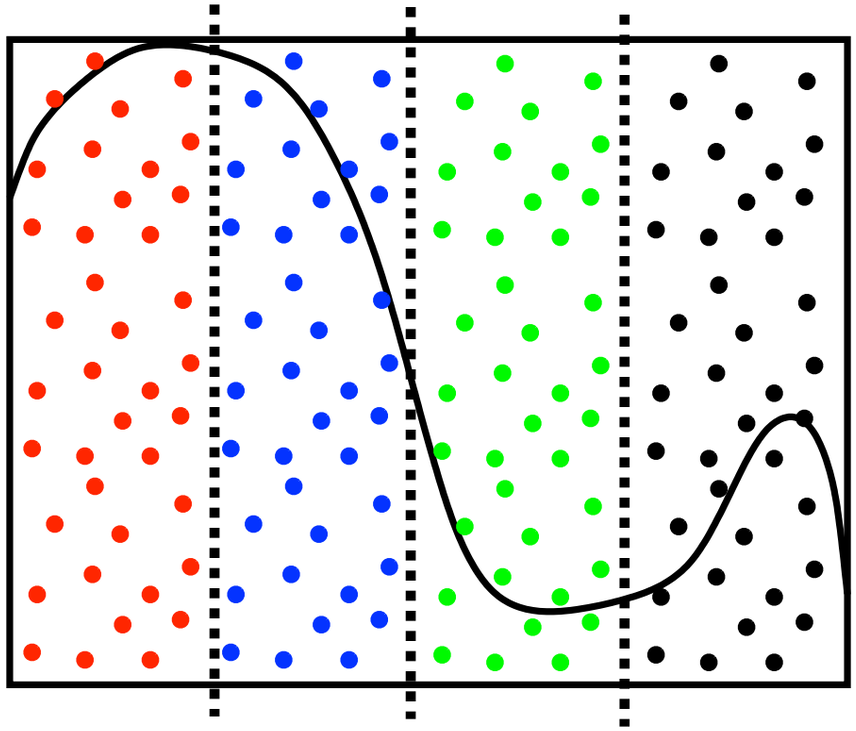
How do we do this computation in parallel?

Example: Numerical Integration

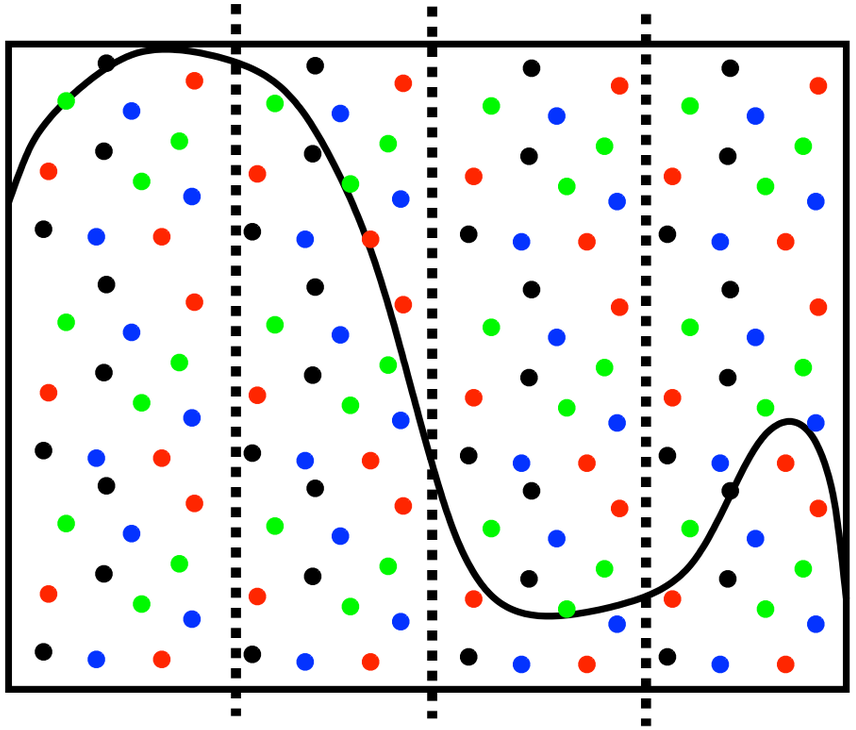
THE UNIVERSITY OF IOWA

Strategies for Parallel Computation of the Numerical Integral:

1) Give different ranges of x to different processors and sum results



2) Give $N/4$ points to each processor and sum results



Example: Fibonacci Series

The Fibonacci series is defined by:

$$f(k+2) = f(k+1) + f(k) \quad \text{with } f(1) = f(2) = 1$$

The Fibonacci series is therefore $(1, 1, 2, 3, 5, 8, 13, 21, \ldots)$

The Fibonacci series can be calculated using the loop

```
f(1)=1
f(2)=1
do i=3, N
      f(i)=f(i-1)+f(i-2)
enddo
```

How do we do this computation in parallel?

This calculation cannot be made parallel.

- We cannot calculate $f(k+2)$ until we have $f(k+1)$ and $f(k)$

- This is an example of data dependence that results in a non-parallelizable problem

Example: Protein Folding

THE UNIVERSITY OF IOWA

- Protein folding problems involve a large number of independent calculations that do not depend on data from other calculations
- Concurrent calculations with no dependence on the data from other calculations are termed Embarrassingly Parallel
- These embarrassingly parallel problems are ideal for solution by HPC methods, and can realize nearly ideal concurrency and scalability

Unique Problems Require Unique Solutions

THE UNIVERSITY OF IOWA

- Each scientific or mathematical problem will, in general, require a unique strategy for efficient parallelization

Thus, each of you may require a different parallel implementation of your numerical problem to achieve good performance.

- Flexibility in the way a problem is solved is beneficial to finding a parallel algorithm that yields a good parallel scaling.

- Often, one has to employ substantial creativity in the way a parallel algorithm is implemented to achieve good scalability.

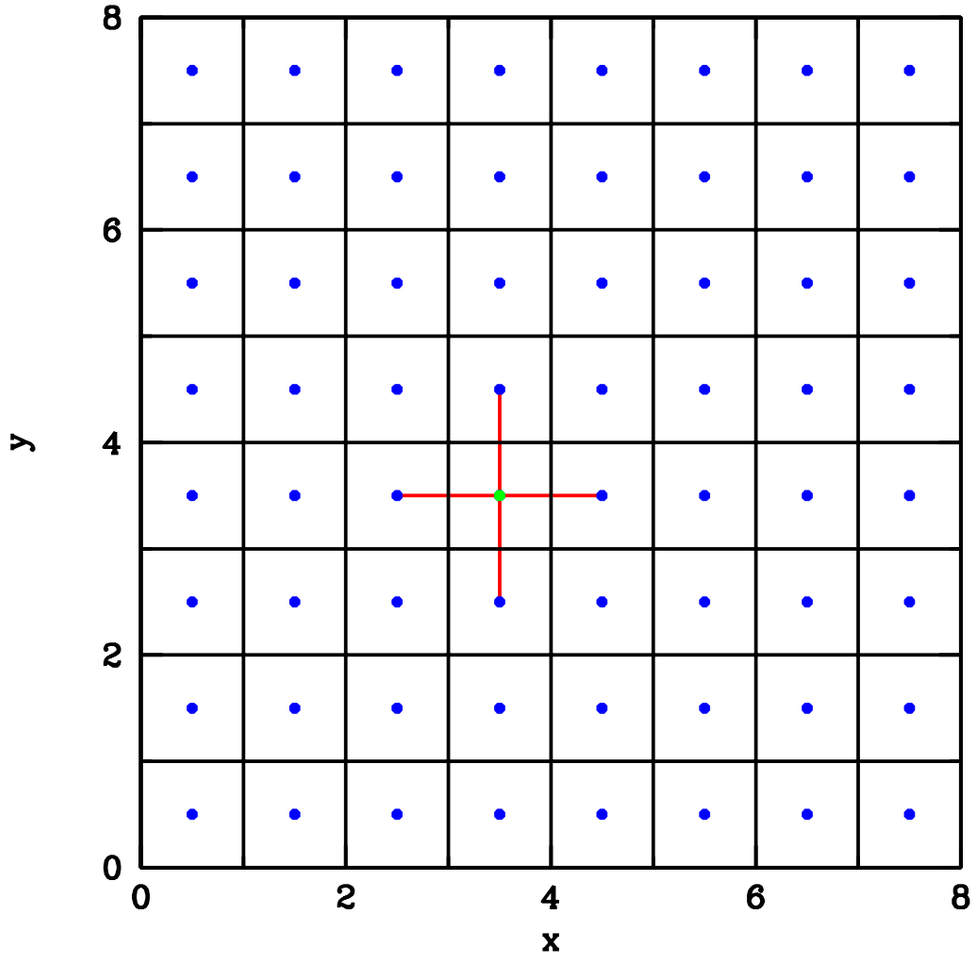
Understand the Dependencies

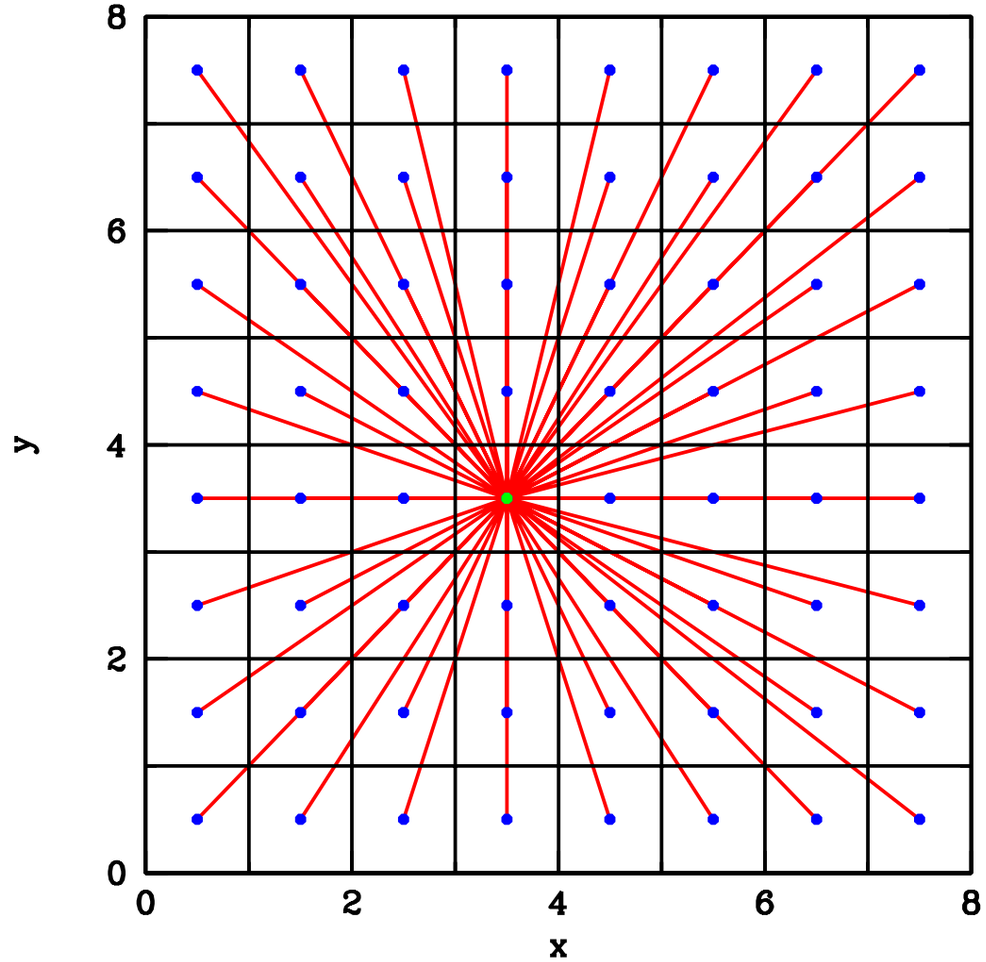
THE UNIVERSITY OF IOWA

- One must understand all aspects of the problem to be solved, in particular the possible dependencies of the data.

- It is important to understand fully all parts of a serial code that you wish to parallelize.

Example: Pressure Forces (Local) vs. Gravitational Forces (Global)





Rule of Thumb

When designing a parallel algorithm, always remember:

Computation is FAST

Communication is SLOW

Input/Output (I/O) is INCREDIBLY SLOW

Other Issues

In addition to concurrency and scalability, there are a number of other important factors in the design of parallel algorithms:

Locality

Granularity

Modularity

Flexibility

Load balancing

We'll learn about these when we discuss the design of parallel algorithms.

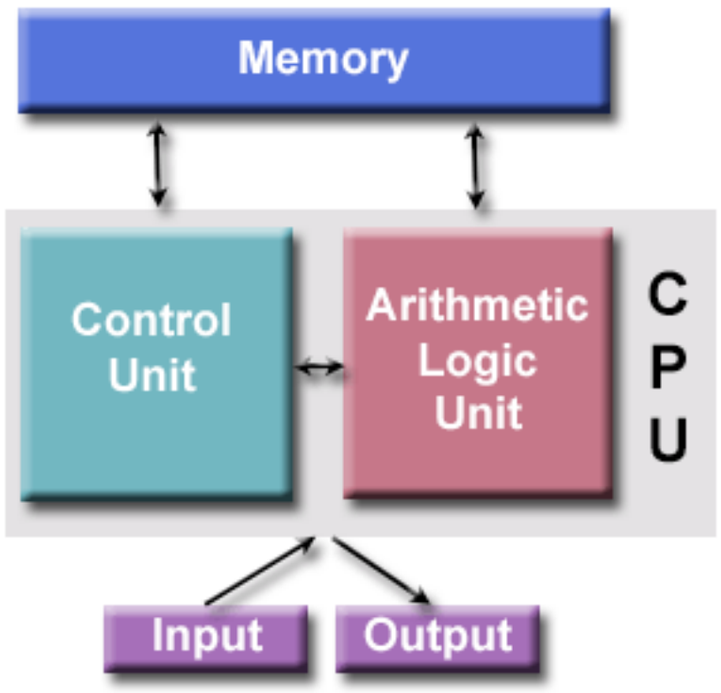
Outline

- Introduction
- Thinking in Parallel
- Parallel Computer Architectures
- Parallel Programming Models
- References

The Von Neumann Architecture

THE UNIVERSITY OF IOWA

Virtually all computers follow this basic design



- Memory stores both instructions and data
- Control unit fetches instructions from memory, decodes instructions, and then sequentially performs operations to perform programmed task
- Arithmetic Unit performs mathematical operations
- Input/Output is interface to the user

Flynn's Taxonomy

THE UNIVERSITY OF IOWA

SISD **Single Instruction, Single Data**	**SIMD** **Single Instruction, Multiple Data**
MISD **Multiple Instruction, Single Data**	**MIMD** **Multiple Instruction, Multiple Data**

- SISD: This is a standard serial computer: one set of instructions, one data stream

- SIMD: All units execute same instructions on different data streams (vector)
  - Useful for specialized problems, such as graphics/image processing
  - Old Vector Supercomputers worked this way, also moderns GPUs

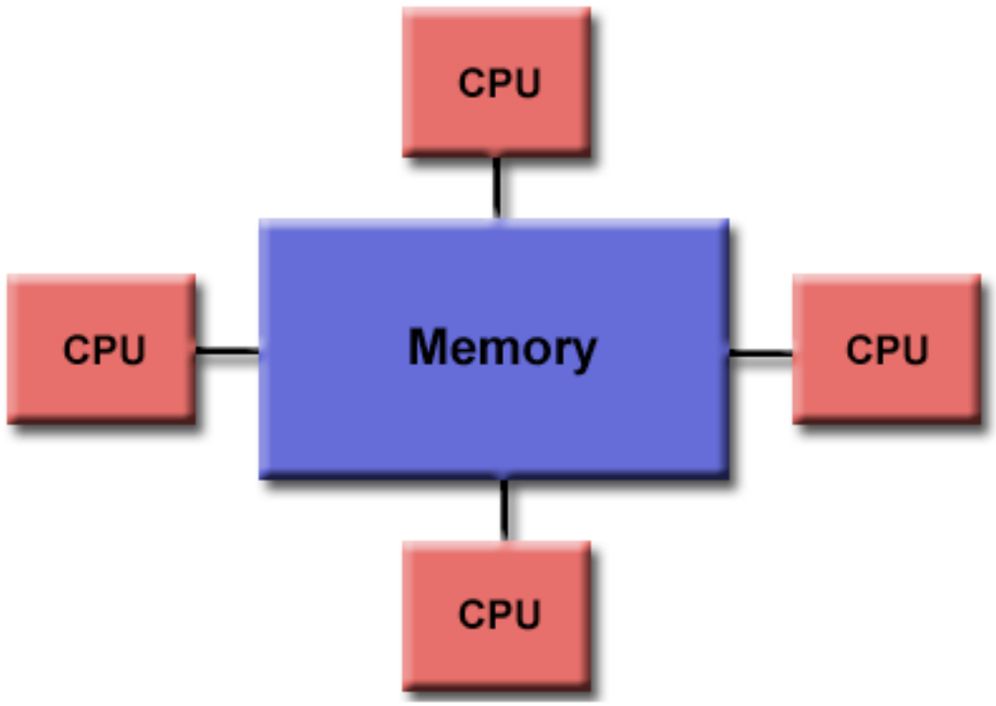
- MISD: Single data stream operated on by different sets of instructions, not generally used for parallel computers

- MIMD: Most common parallel computer, each processor can execute different instructions on different data streams
  -Often constructed of many SIMD subcomponents

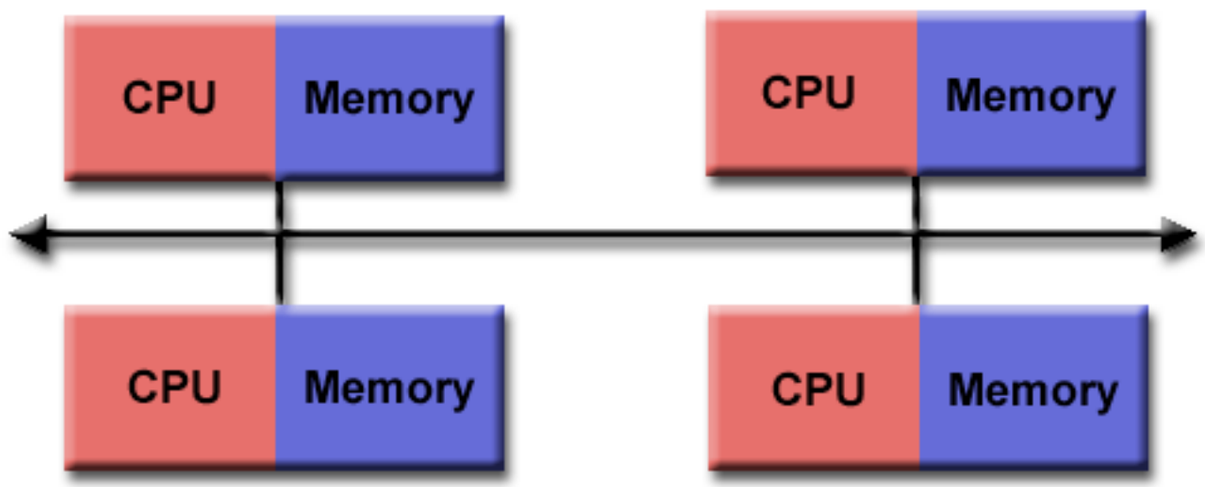
Parallel Computer Memory Architectures

THE UNIVERSITY OF IOWA

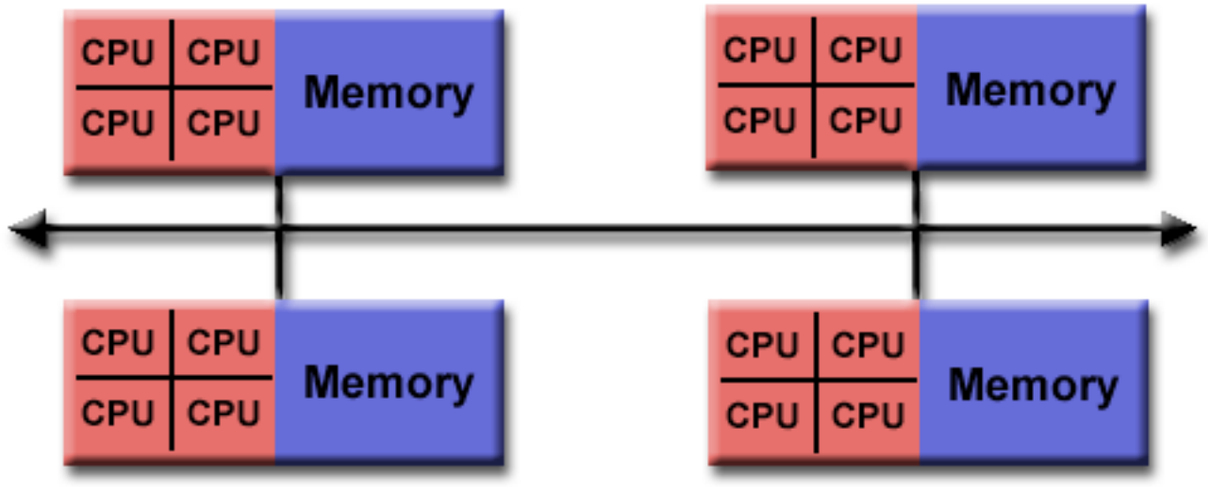
Shared Memory



Distributed Memory



Hybrid Distributed Shared Memory



Relation to Parallel Programming Models

THE UNIVERSITY OF IOWA

• OpenMP: Multi-threaded calculations occur within shared-memory components of systems, with different threads working on the same data.

• MPI: Based on a distributed-memory model, data associated with another processor must be communicated over the network connection.

• GPUs: Graphics Processing Units (GPUs) incorporate many (hundreds) of computing cores with single Control Unit, so this is a shared-memory model.

• Processors vs. Cores: Most common parallel computer, each processor can execute different instructions on different data streams

  -Often constructed of many SIMD subcomponents

Outline

- Introduction
- Thinking in Parallel
- Parallel Computer Architectures
- Parallel Programming Models
- References

Parallel Programming Models

THE UNIVERSITY OF IOWA

- Embarrassingly Parallel
- Master/Slave
- Threads
- Message Passing
- Single Program-Multiple Data (SPMD)
  vs. Multiple Program-Multiple Data (MPMD)
- Other Parallel Implementations: GPUs and CUDA

Embarrassingly Parallel

THE UNIVERSITY OF IOWA

- Refers to an approach that involves solving many similar but independent tasks simultaneously

- Little to no coordination (and thus no communication) between tasks

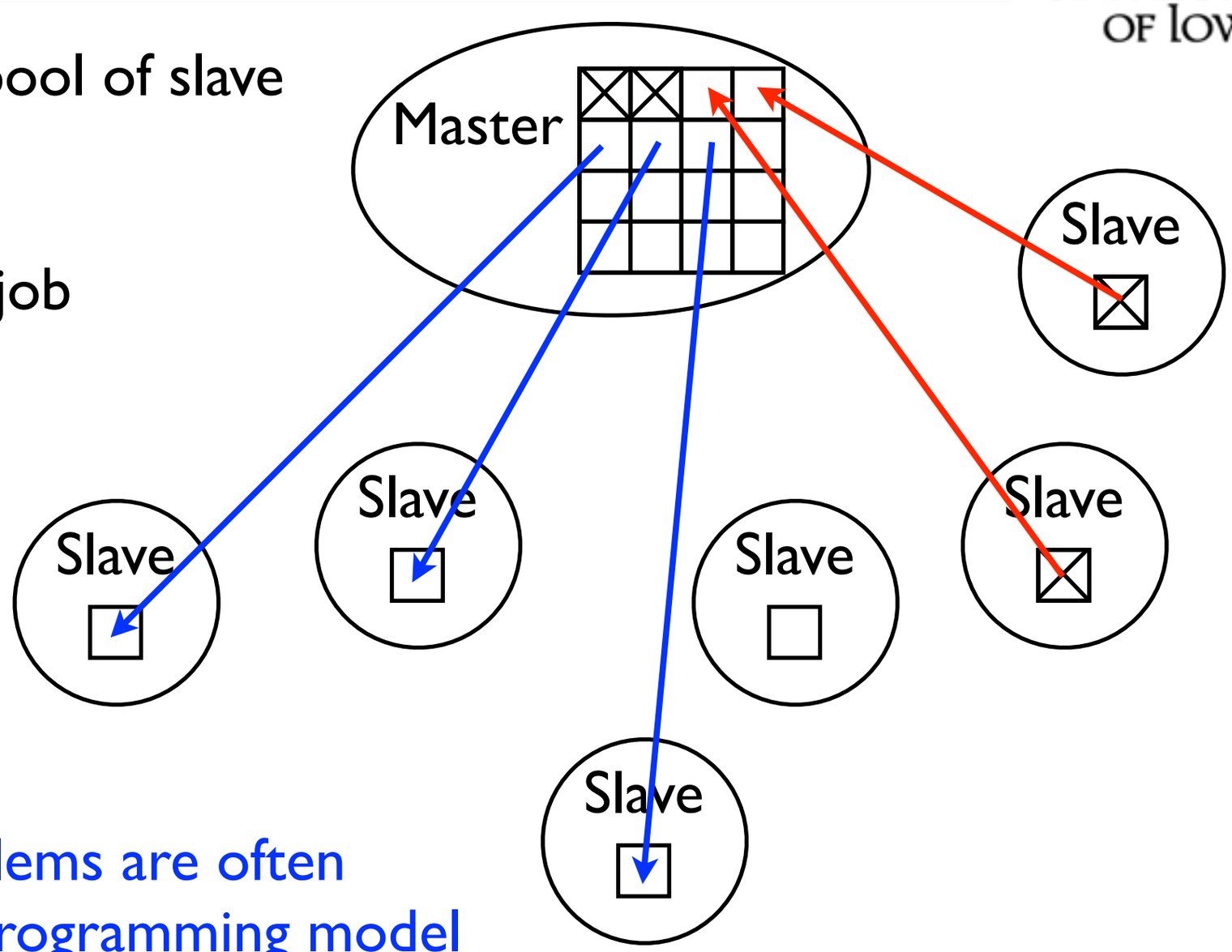
- Each task can be a simple serial program

- This is the "easiest" type of problem to implement in a parallel manner. Essentially requires automatically coordinating many independent calculations and possibly collating the results.

- Examples:
  - Computer Graphics and Image Processing
  - Protein Folding Calculations in Biology
  - Geographic Land Management Simulations in Geography
  - Data Mining in numerous fields
  - Event simulation and reconstruction in Particle Physics

Master/Slave

- Master Task assigns jobs to pool of slave tasks
- Each slave task performs its job independently
- When completed, each slave returns its results to the master, awaiting a new job



- Emabarrasingly parallel problems are often well suited to this parallel programming model

Multi-Threading

THE UNIVERSITY OF IOWA

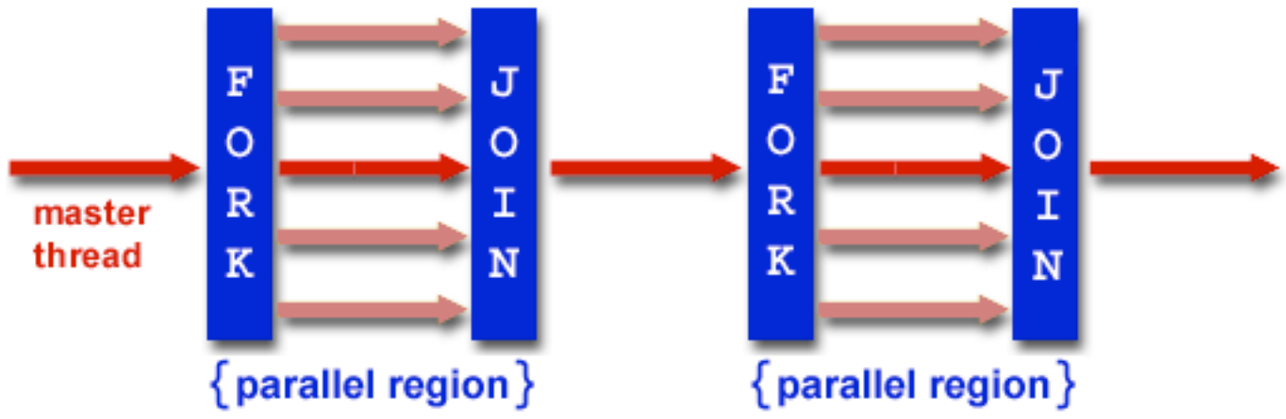
• Threading involves a single process that can have multiple, concurrent execution paths

• Works in a shared memory architecture

• Most common implementation is OpenMP (Open Multi-Processing)

```
serial code
        .
        .
        .

!$OMP PARALLEL DO
do i = 1,N
  A(i)=B(i)+C(i)
enddo
!$OMP END PARALLEL DO
        .
        .
        .
serial code
```



• Relatively easy to make inner loops of a serial code parallel and achieve substantial speedups with modern multi-core processors

Message Passing

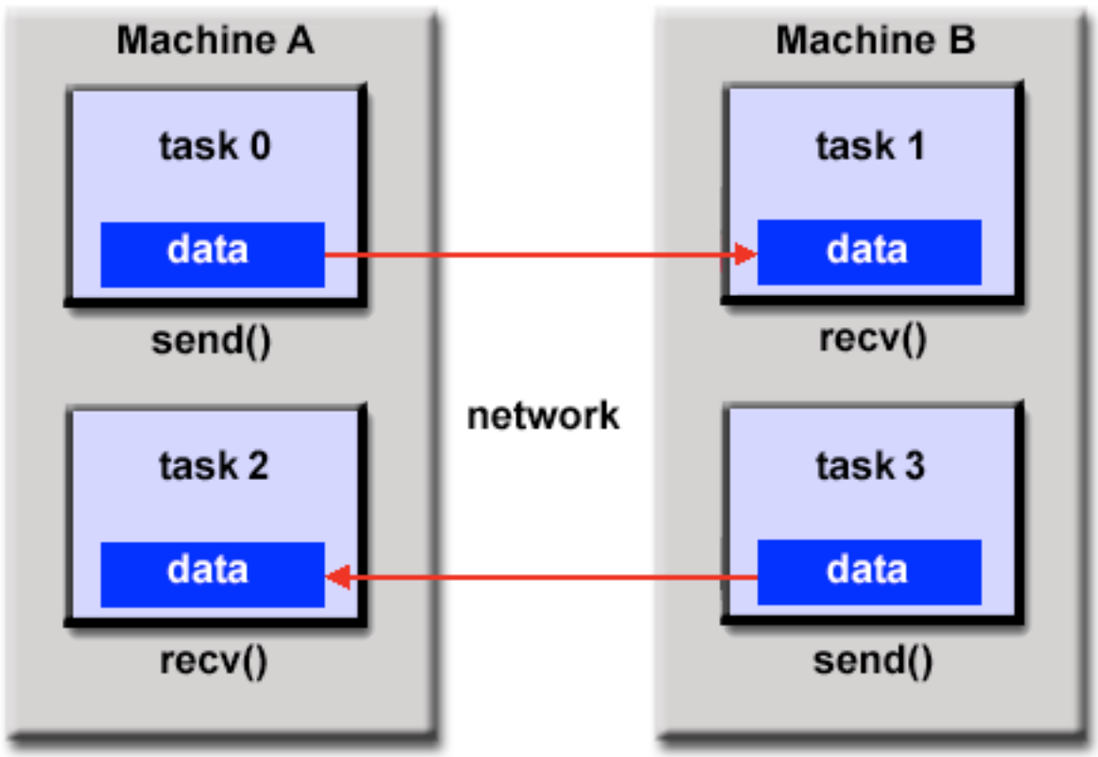
THE UNIVERSITY OF IOWA

- The most widely used model for parallel programming

- Message Passing Interface (MPI) is the most widely used implementation

- A set of tasks have their own local memory during the computation (distributed-memory, but can also be used on shared-memory machines)

- Tasks exchange data by sending and receiving messages, requires programmer to coordinate explicitly all sends and receives.



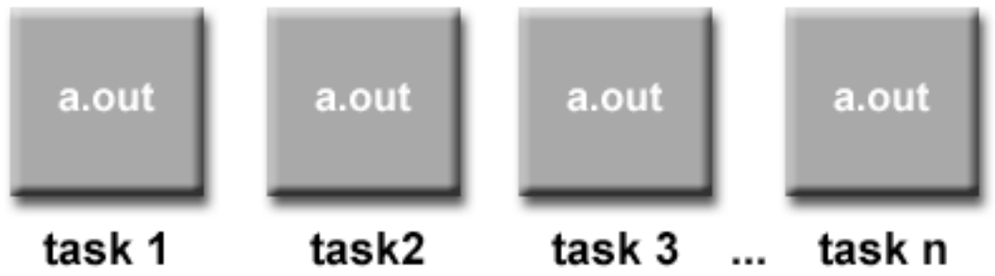
- One aim of this course will focus on the use of MPI to write parallel programs.

SPMD vs. MPMD

THE UNIVERSITY OF IOWA

Single Program-Multiple Data (SPMD)

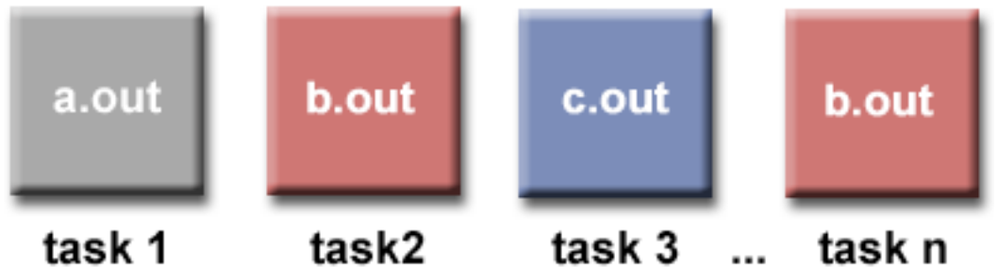
- A single program executes on all tasks simultaneously



- At a single point in time, different tasks may be executing the same or different instructions (logic allows different tasks to execute different parts of the code)

Multiple Program-Multiple Data (MPMD)

- Each task may be executing the same or different programs than other tasks

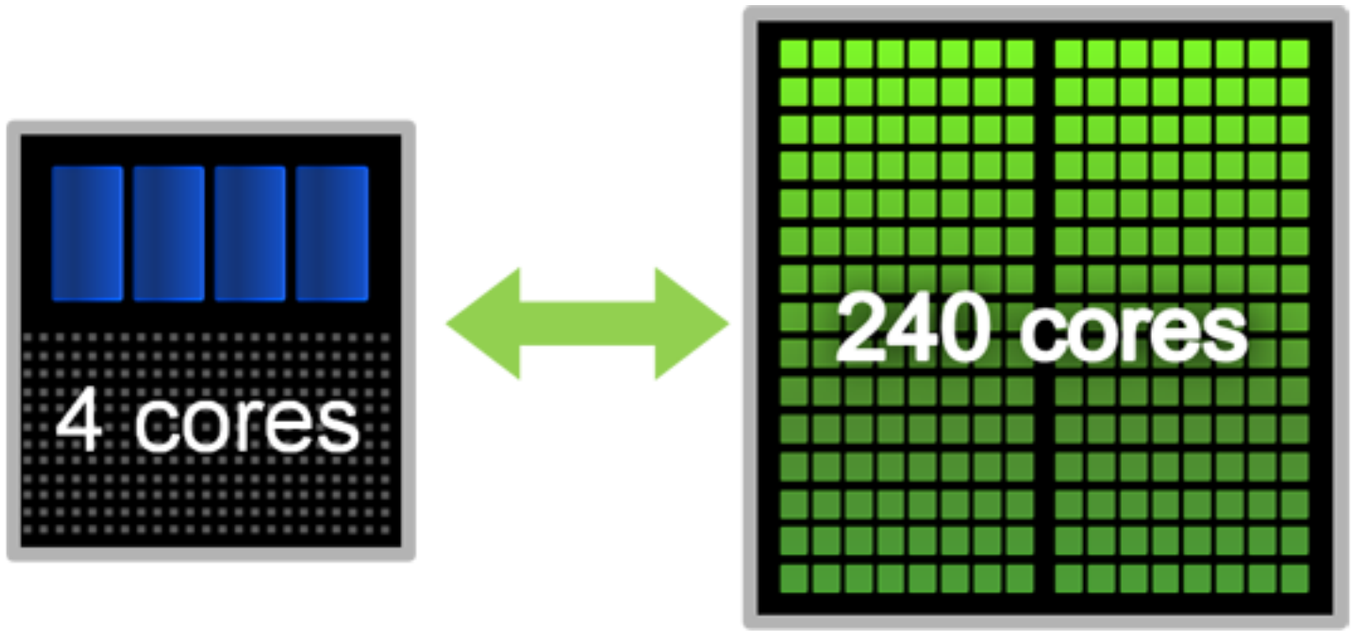


- The different executable programs may communicate to transfer data

Other Parallel Programming Models

THE UNIVERSITY OF IOWA

- GPUs (Graphics Processing Units) contain many (hundreds) of processing cores, allowing for rapid vector processing (Single Instruction, Multiple Data)



• CUDA (Compute Unified Device Architecture) programming allows one to call on this powerful computing engine from codes written in C, Fortran, Python, Java, and Matlab.

• This is an exciting new way to achieve massive computing power for little hardware cost, but memory access bandwidth limitations constrain the possible applications.

Parting Thoughts

• Part of the challenge of parallel computing is that the most efficient parallelization strategy for each problem generally requires a unique solution.

• It is generally worthwhile spending significant time considering alternative algorithms to find an optimal one, rather than just implementing the first thing that comes to mind

• But, consider the time required to code a given parallel implementation
  - You can use a less efficient method if the implementation is much easier.
  - You can always improve the parallelization scheme later. Just focus on making the code parallel first.

TIME is the ultimate factor is choosing a parallelization strategy---Your Time!

References

THE UNIVERSITY OF IOWA

Introductory Information on Parallel Computing

- Designing and Building Parallel Programs, Ian Foster
  http://www.mcs.anl.gov/~itf/dbpp/
  -Somewhat dated (1995), but an excellent online textbook with detailed discussion about many aspects of HPC. This presentation borrowed heavily from this reference
- Introduction to Parallel Computing, Blaise Barney
  https://computing.llnl.gov/tutorials/parallel_comp/
  -Up to date introduction to parallel computing with excellent links to further information
- MPICH2: Message Passage Inteface (MPI) Implementation
  http://www.mcs.anl.gov/research/projects/mpich2/
  -The most widely used Message Passage Interface (MPI) Implementation
- OpenMP
  http://openmp.org/wp/
  -Application Program Interface (API) supports multi-platform shared-memory parallel programming in C/C++ and Fortran
- Numerical Recipes
  http://www.nr.com/
  -Incredibly useful reference for a wide range of numerical methods, though not focused on parallel algorithms.
- The Top 500 Computers in the World
  http://www.top500.org/
  -Updated semi-annually list of the Top 500 Supercomputers

References

Introductory Information on Parallel Computing

- Message Passing Interface (MPI), Blaise Barney
  https://computing.llnl.gov/tutorials/mpi/
  -Excellent tutorial on the use of MPI, with both Fortran and C example code

- OpenMP, Blaise Barney
  https://computing.llnl.gov/tutorials/openMP/
  -Excellent tutorial on the use of OpenMP, with both Fortran and C example code

- High Performance Computing Training Materials, Lawrence Livermore National Lab
  https://computing.llnl.gov/?set=training&page=index
  -An excellent online set of webpages with detailed tutorials on many aspects of high performance computing.