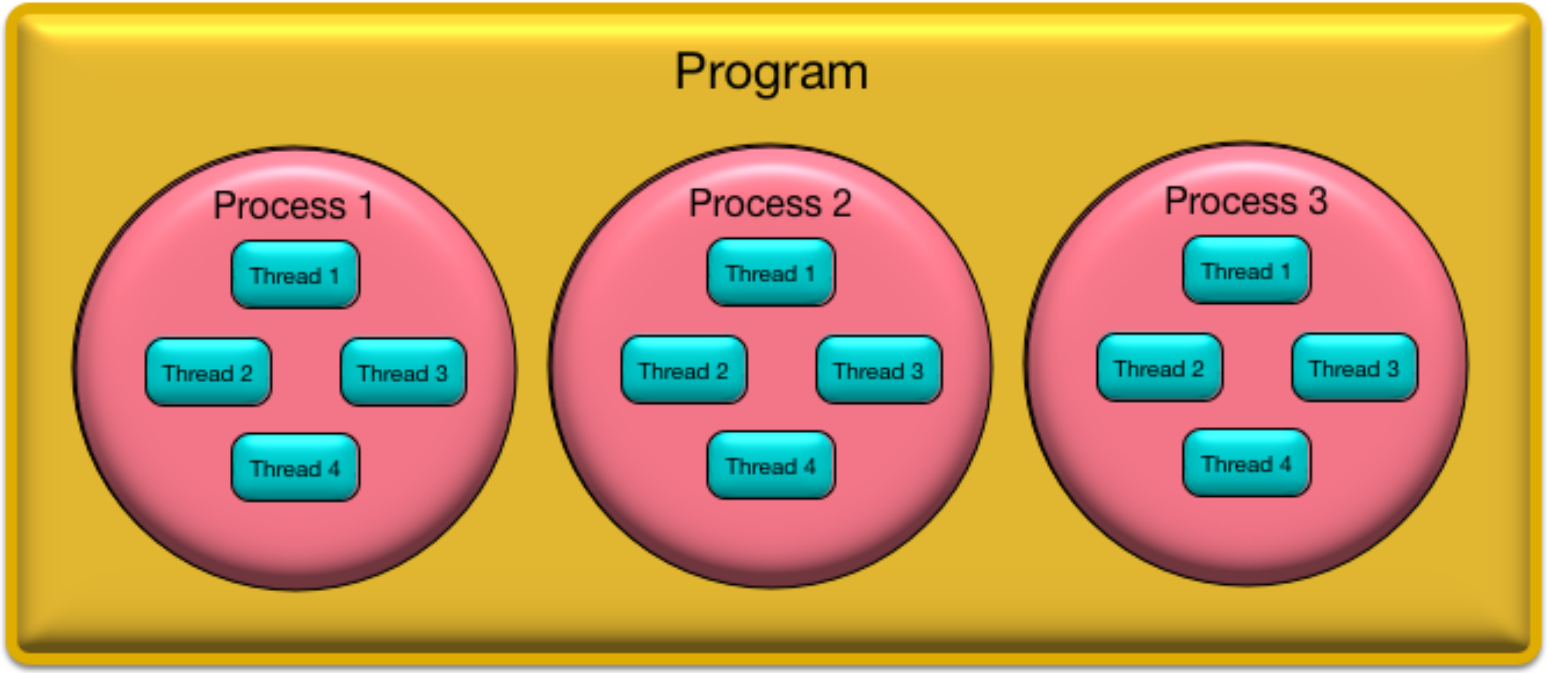
Parallel Computing

Introduction

Ackwonledgement: https://nyu-cds.github.io/python-mpi

Program, Process, Thread

Program: an executable file residing on disk.
Process: one or more executing instances of a program. Processes have separate address spaces.
Task: In MPI, a process is sometimes called a *task*, but these are used interchangably. We will always use “process” rather than “task”.
Thread (or lightweight process): one or more threads of control within a process. Threads share the same address space.



Program, Process, Thread

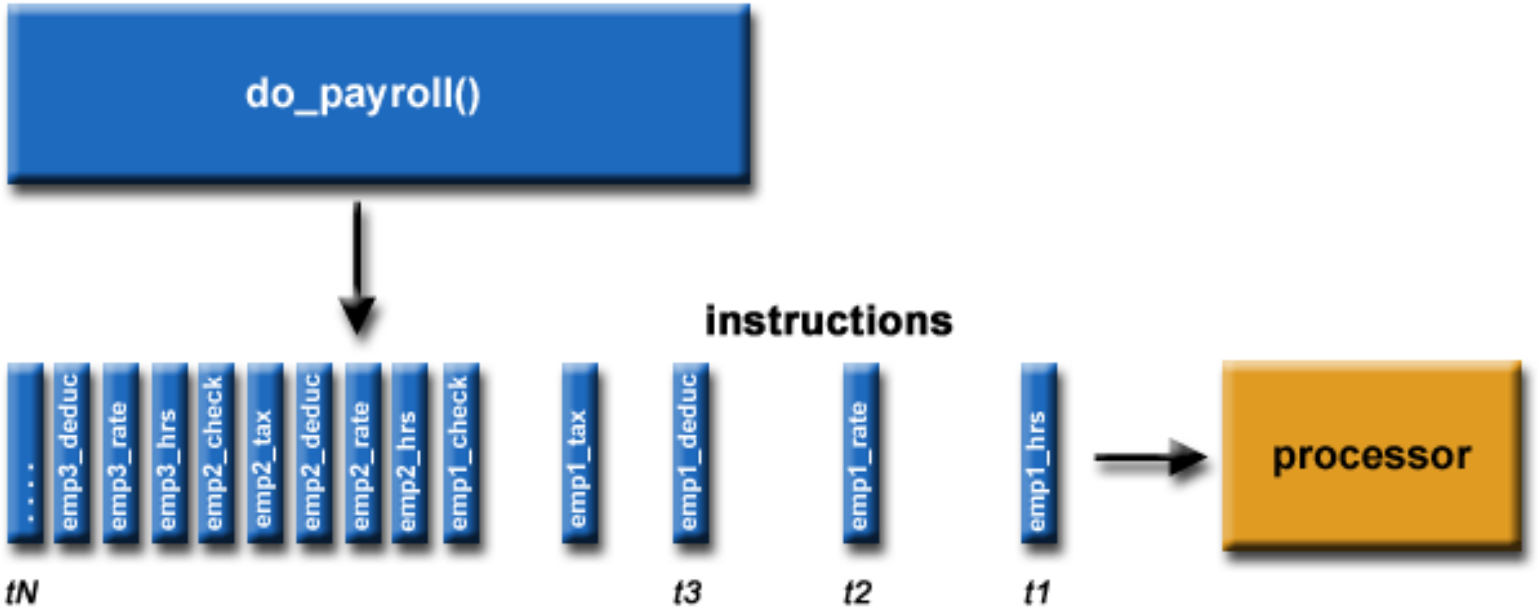
So, to review:

1. The program starts out as a text file of programming code,
2. The program is compiled or interpreted into binary form,
3. The program is loaded into memory,
4. The program becomes one or more running processes.
5. Processes are typically independent of each other,
6. While threads exist as the subset of a process.
7. Threads can communicate with each other more easily than processes can,
8. But threads are more vulnerable to problems caused by other threads in the same process.

What is parallel computing?

Traditionally, software has been written for serial computation:

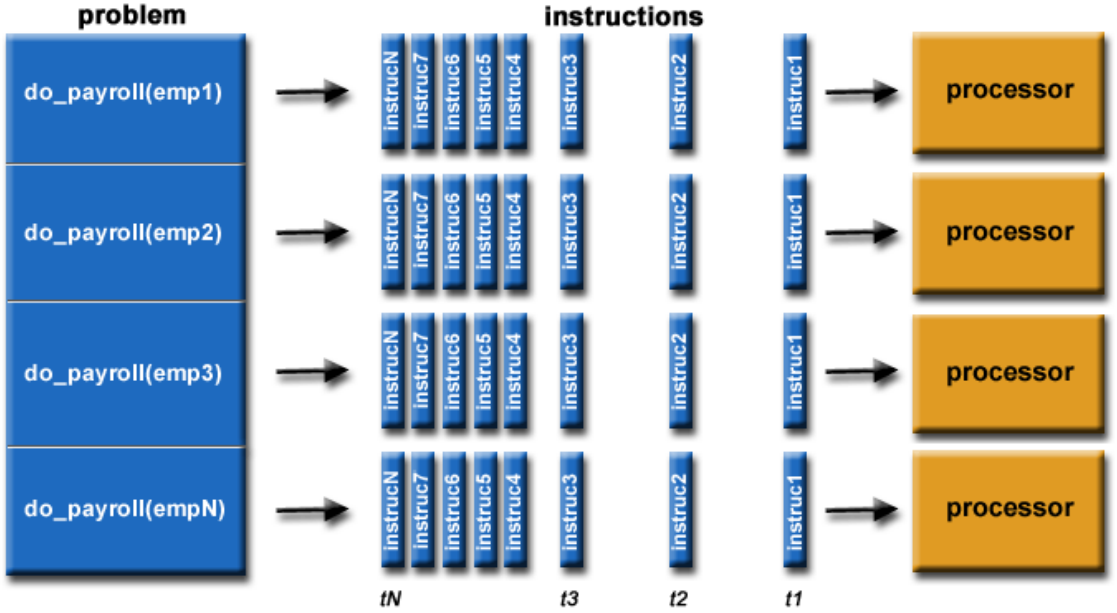
- A problem is broken into a discrete series of instructions
- Instructions are executed sequentially one after another
- Executed on a single processor
- Only one instruction may execute at any moment in time



What is parallel computing?

Parallel computing is the simultaneous use of multiple compute resources to solve a computational problem:

- A problem is broken into discrete parts that can be solved concurrently
- Each part is further broken down to a series of instructions
- Instructions from each part execute simultaneously on different processors
- An overall control/coordination mechanism is employed



Why do we need parallel programming?

Need to solve larger problems

- Require more memory
- Computation takes much longer
- Huge amounts of data may be required

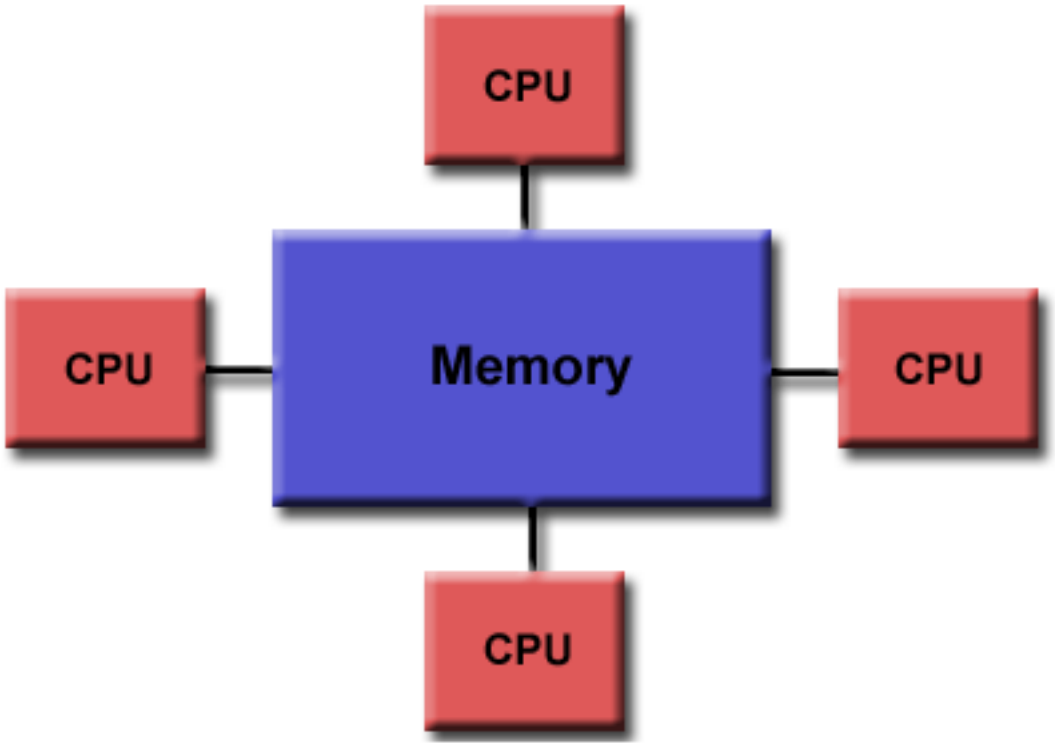
Parallel programming provides

- More CPU resources
- More memory resources
- The ability to solve problems that were not possible with serial program
- The ability to solve problems more quickly

Hardware: two basic approaches

Shared Memory Computer

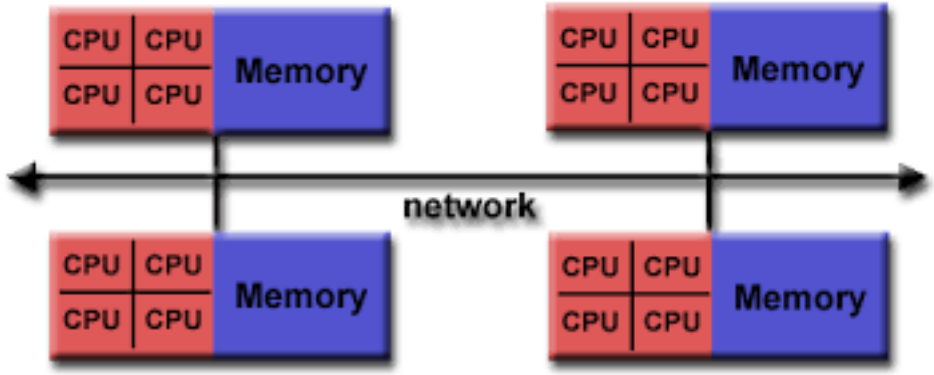
- Used by most laptops/PCs
- Multiple cores (CPUs)
- Share a global memory space
- Cores can efficiently exchange/share data

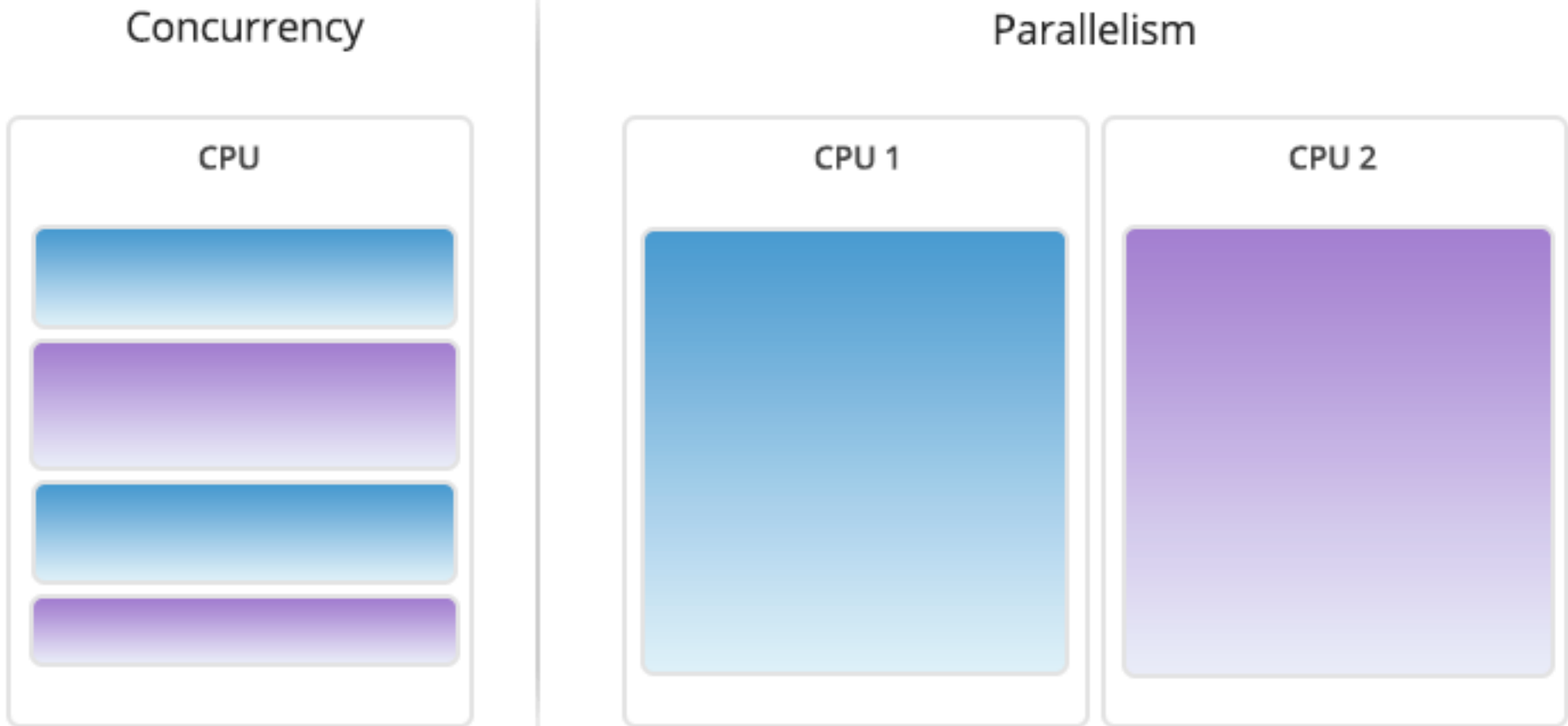


Hardware: two basic approaches

Distributed Memory (ex. Compute cluster)

- Collection of nodes which have multiple cores
- Each node uses its own local memory
- Work together to solve a problem
- Communicate between nodes and cores via messages
- Nodes are networked together




Concurrency
CPU
Parallelism
CPU 1
CPU 2

Software: Parallel programming models

Directive-based parallel programming language

- OpenMP (most widely used)
- Directives tell processor how to distribute data and work across the processors
- Directives appear as comments in the serial code
- Implemented on shared memory architectures

Directives are commands and information to the compiler for altering or interpreting the code

Message Passing

- MPI (most widely used)
- Pass messages to send/receive data between processes
- Each process has its own local variables
- Can be used on either shared or distributed memory architectures

Software: Parallel programming models

```c
#include <stdio.h>
#include <omp.h>

int main(void)
{
    #pragma omp parallel
    printf("Hello, world.\n");
    return 0;
}
```

```c
int main(int argc, char **argv)
{
    int a[100000];

    #pragma omp parallel for
    for (int i = 0; i < 100000; i++) {
        a[i] = 2 * i;
    }

    return 0;
}
```

This example is embarrassingly parallel, and depends only on the value of `i`. The OpenMP `parallel for` flag tells the OpenMP system to split this task among its working threads. The threads will each receive a unique and private version of the variable. For instance, with two worker threads, one thread might be handed a version of `i` that runs from 0 to 49999 while the second gets a version running from 50000 to 99999.

Software: Parallel programming models

```
#include "mpi.h"

int main( int argc, char *argv[] )
{
    int numprocs, myrank;

    MPI_Init( &argc, &argv );
    MPI_Comm_size( MPI_COMM_WORLD, &numprocs );
    MPI_Comm_rank( MPI_COMM_WORLD, &myrank );

    if ( myrank == 0 )          /* manager process */
        manager_code ( numprocs );
    else                        /* worker process */
        worker_code ( );
    MPI_Finalize( );
    return 0;
}
```

Software: Parallel programming models

Pros of OpenMP

- Easier to program and debug than MPI
- Can still run the program as a serial code
- Serial code statements usually don’t need modification
- Code is easier to understand and maybe more easily maintained

Cons of OpenMP

- Can only be run in shared memory computers
- Requires a compiler that supports OpenMP
- Mostly used for loop parallelization

Software: Parallel programming models

Pros of MPI

- Runs on either shared or distributed memory architectures
- Can be used on a wider range of problems than OpenMP
- Each process has its own local variables
- Distributed memory computers are less expensive than large shared memory computers

Cons of MPI

- Requires more programming changes to go from serial to parallel version
- Can be harder to debug
- Performance is limited by the communication network between the nodes

Parallel programming issues

Goal is to reduce execution time

- Computation time
- Idle time - waiting for data from other processors
- Communication time - time the processors take to send and receive messages

Load Balancing

- Divide the work equally among the available processors

Minimizing Communication

- Reduce the number of messages passed
- Reduce amount of data passed in messages

Many problems scale well to only a limited number of processors

Amdahl's law

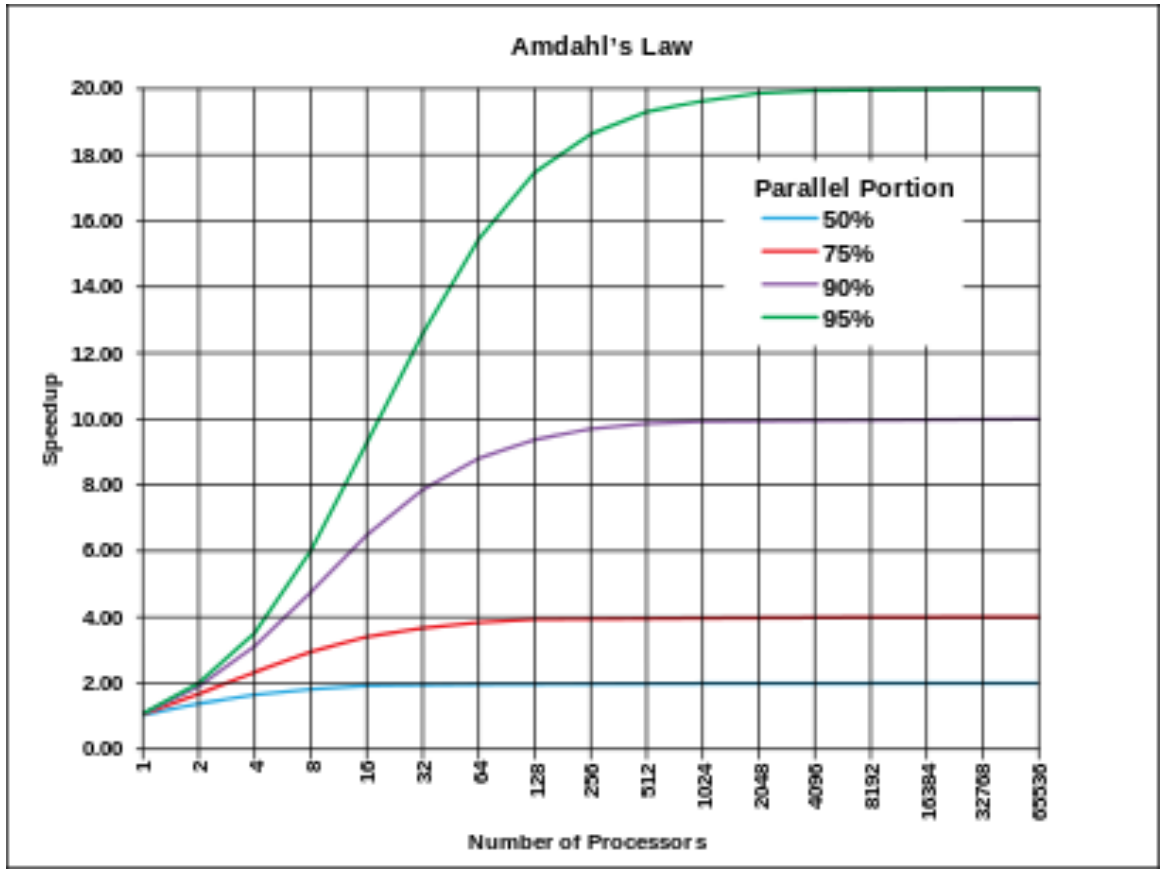
$$speedup = \frac{1}{(1 - P) + \frac{P}{n}}$$

- *speedup* is the theoretical speedup of the execution of the whole program
- *n* is the number of parallel threads/processes
- *P* is the fraction of the algorithm that can be made parallel

Amdahl's law

$$speedup = \frac{1}{(1 - P) + \frac{P}{n}}$$

Basically, this is saying that the amount of speedup a program will see by using *n* cores is based on how much of the program is serial (can only be run on a single CPU core) and how much of it is parallel (can be split up among multiple CPU cores).



Programming approaches

SPMD - Single Program, Multiple Data Streams

- Each processor is executing the same program on different data
- A parallel execution model that assumes multiple cooperating processes executing a program
- The most common style of parallel programming and the one used by MPI

MPMD - Multiple Programs, Multiple Data Streams

- Multiple processors executing at least two independent programs
- Manager/worker strategies fit into this category
- Web browser and web server is another example

What is MPI?

Message Passing Interface (**MPI**) is a standardized and portable message-passing standard designed by a group of researchers from academia and industry to function on a wide variety of parallel computing architectures. The standard defines the syntax and semantics of a core of library routines useful to a wide range of users writing portable message-passing programs in C, C++, and Fortran. There are several well-tested and efficient implementations of MPI, many of which are open-source or in the public domain. These fostered the development of a parallel software industry, and encouraged development of portable and scalable large-scale parallel applications.

https://en.wikipedia.org/wiki/Message_Passing_Interface

MPI Installation

On your laptop

These instructions are based on installing compilers and MPI via the **conda package manager**, as it provides a convenient way to install binary packages in an isolated software environment.

- The instructions focus on installation on **MacOS** and **Linux** computers, as well as **Windows computers using the Windows Subsystem for Linux (WSL)**.
- Instructions for installing WSL on Windows can be found here
- Installing compilers and MPI natively on Windows is also possible through Cygwin and the Microsoft Distribution of MPICH, but we recommend that you instead use WSL which is available for Windows 10 and later versions.

https://pdc-support.github.io/introduction-to-mpi/setup.html

MPI Installation

Installing conda

Begin by installing Miniconda:

1. Download the 64-bit installer from here for your operating system
   - for MacOS and Linux, choose the bash installer
   - on Windows, open a Linux-WSL terminal and type: `wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh`. If wget is not a recognised command, type `sudo apt-get install wget` and provide the password you chose when installing WSL.
2. In a terminal, run the installer with `bash Miniconda3-latest-<operating-system>-x86_64.sh` (replace with correct name of installer)
3. Agree to the terms of conditions, specify the installation directory (the default is usually fine), and answer "yes" to the questions "Do you wish the installer to initialize Miniconda3 by running conda init?"

You now have miniconda and conda installed. Make sure that it works by typing `which conda` and see that it points to where you installed miniconda (you may have to open a new terminal first).

We recommend that you create an isolated conda environment (this is good practice in software development):

Bash

```
conda create --name mpi-intro python=3.7
conda activate mpi-intro
```

MPI Installation

If you want to use Python for the exercises, you will need to install mpi4py. mpi4py can be installed either using pip or conda, but with pip you will need to install MPI yourself first (e.g. OpenMPI or MPICH), while conda will install its own MPI libraries. If you don't already have MPI installed on your laptop, it will be easiest to use conda:

Bash

```
(mpi-intro) $ conda install -c conda-forge mpi4py
```

Please also verify the installation. The following command should not give an error message:

Bash

```
(mpi-intro) $ python -c "from mpi4py import MPI"
```

and the following command should give a version number:

Bash

```
(mpi-intro) $ mpirun --version
```