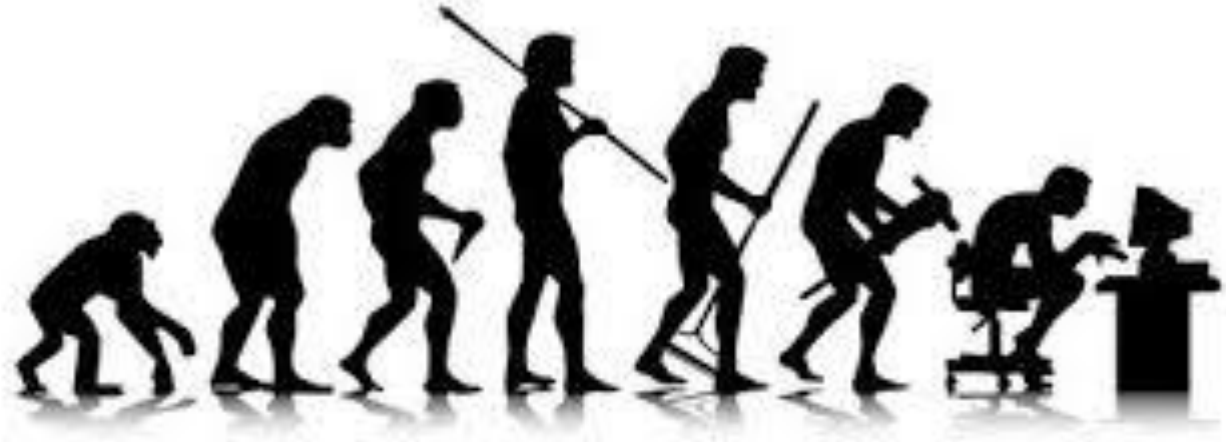
Comp 428 – Fall 2018

Tutorial 1

Introduction to MPI

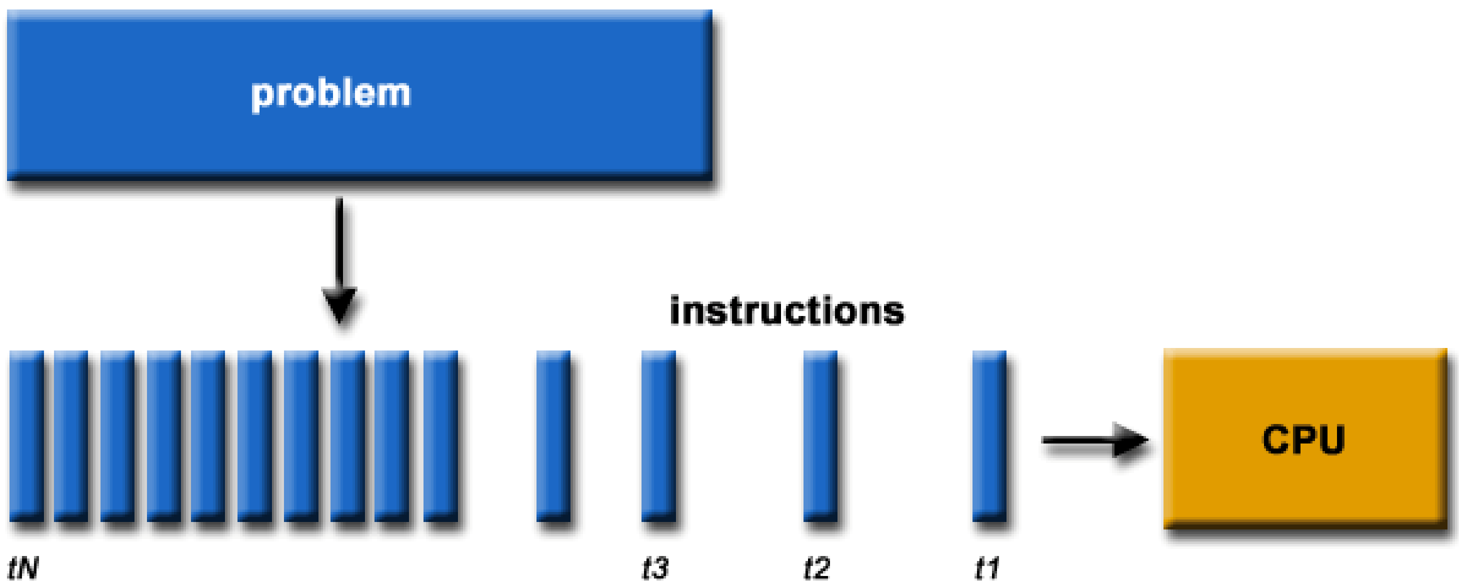


Journey of Computing Tech

Serial Computing

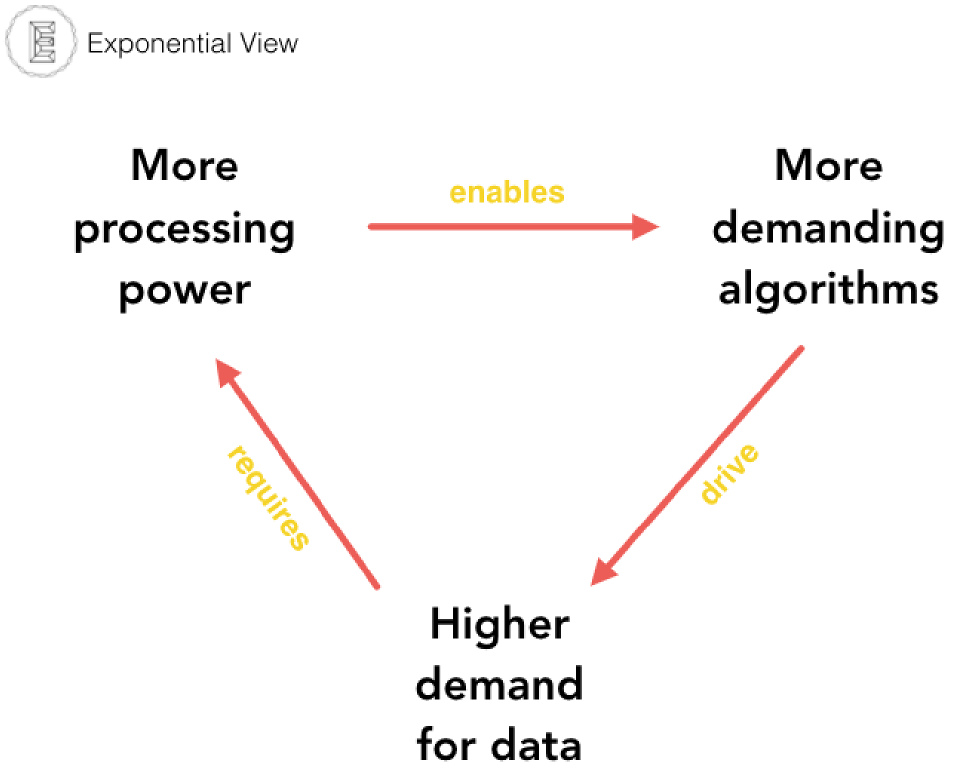
- Traditionally, software has been written for **serial** computation:
  - To be run on a single computer having a single Central Processing Unit (CPU)
  - A problem is broken into a discrete series of instructions
  - Instructions are executed one after another
  - Only one instruction may be executed at any moment in time

Serial Computing



Challenges

- Demand for edge computing



- https://medium.com/s/ai-and-the-future-of-computing/when-moores-law-met-ai-f572585da1b7

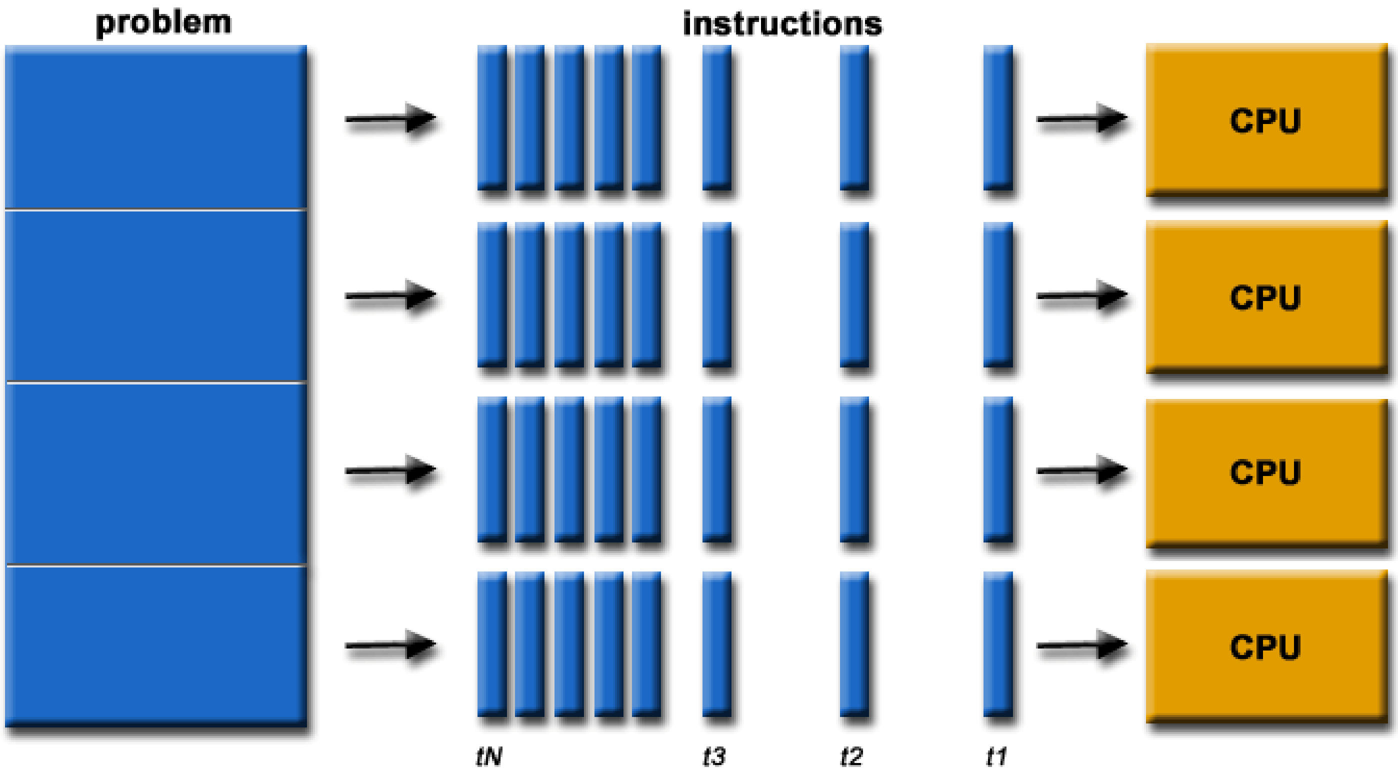
Motivation for Parallel Computing

- Technology push
  - Latest development happening on Hardware side
- Application pull
  - Complex problems require computation on large-scale data
  - Sufficient performance available only through massive parallelism

Parallel Computing

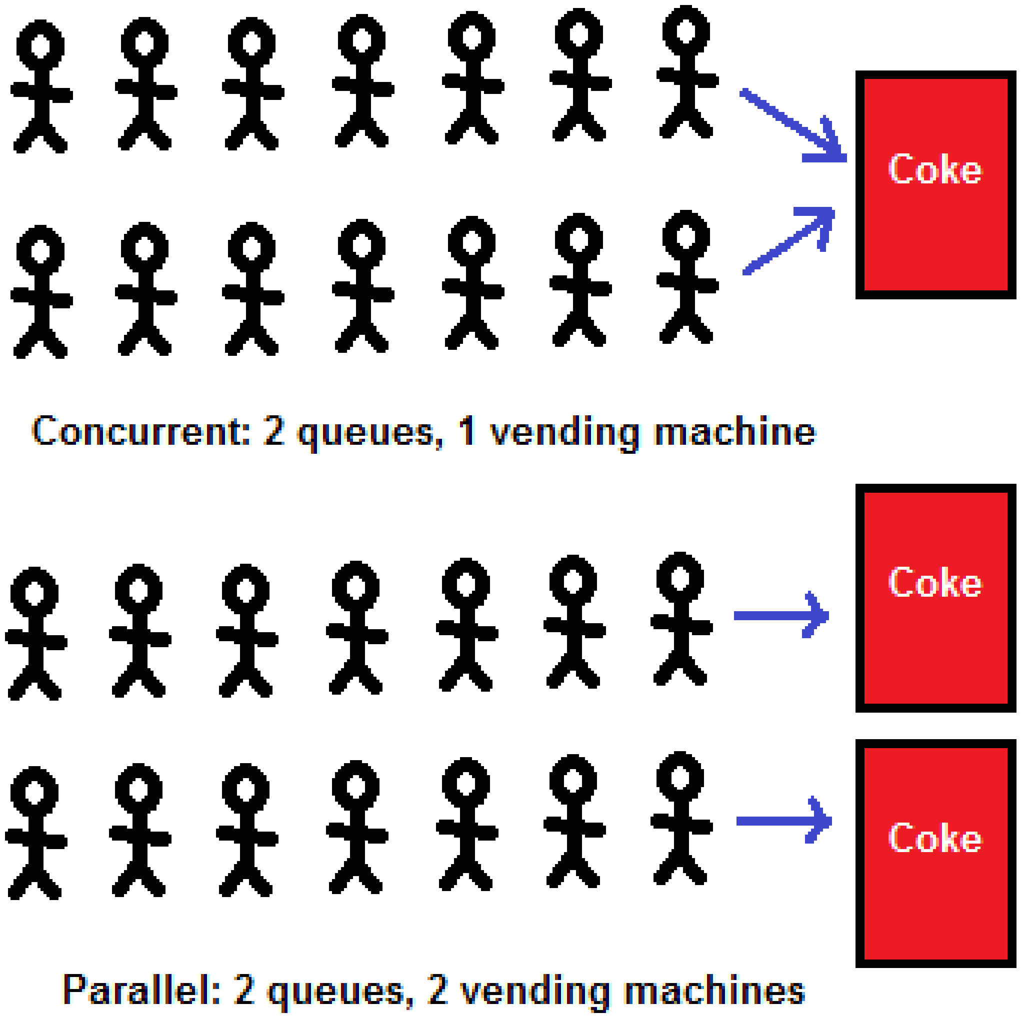
- In the simplest sense, **parallel computing** is the simultaneous use of multiple computing resources to solve a computational problem:
  - To be run using multiple CPUs
  - A problem is broken down to a series of instructions
  - Instructions from each part execute simultaneously on **different CPUs**

Parallel Computing



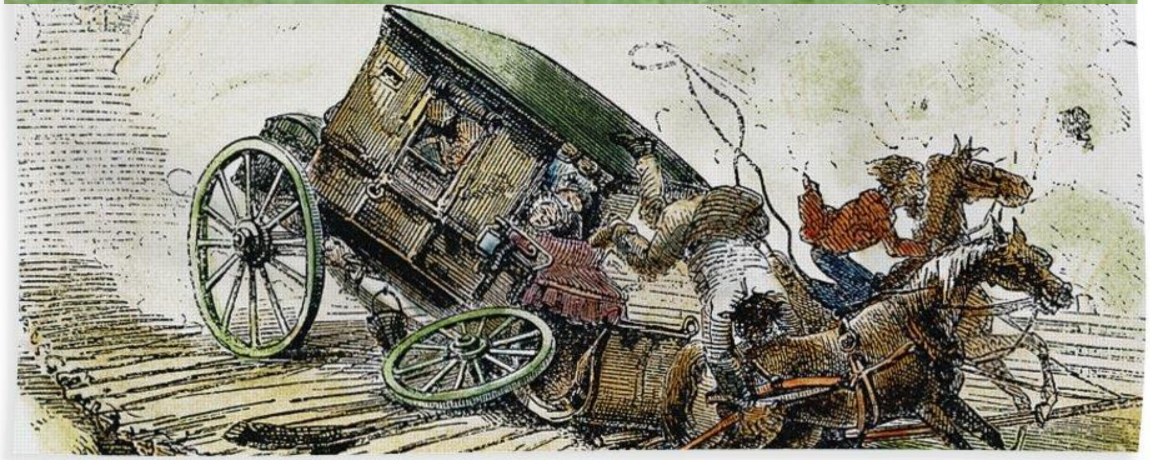
Question?

Concurrency vs Parallelism



https://alvinalexander.com/scala/fp-book/benefits-of-functional-programming

How to drive these horses?



What needs to be done

- Initialization
- Work allocation
- Computation & Communication
- Termination

Need of standards

http://www.lordofthejars.com/2017/10/adding-chaos-on-openshift-cluster.html

Message passing Interface (MPI)

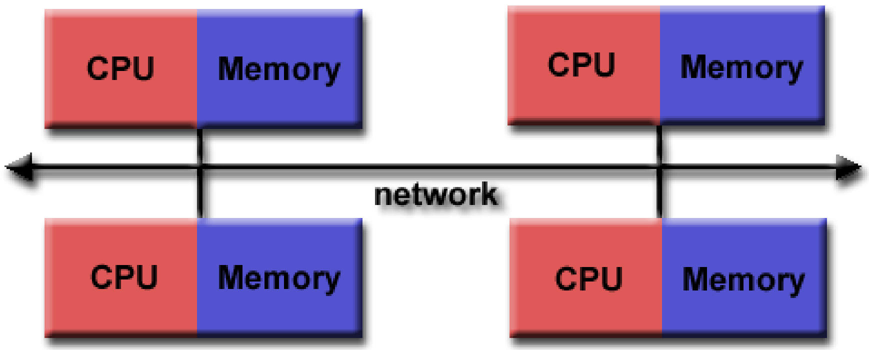
- MPI is a specification for the developers and users of message passing libraries.
- It is NOT a library - but rather the specification of what such a library should be.
- The goal of the Message Passing Interface is to provide a widely used standard for writing message passing programs. The interface attempts to be
  - Practical
  - Portable
  - Efficient
  - Flexible
- Interface specifications have been defined for C/C++ and Fortran programs.
- Complete MPI specification consists of 129 calls.

Goals of the MPI standard

- MPI's prime goals are:
  - To provide source-code portability
  - To allow efficient implementations
- MPI also offers:
  - A great deal of functionality
  - Support for heterogeneous parallel architectures
  - Allow efficient and reliable Communication

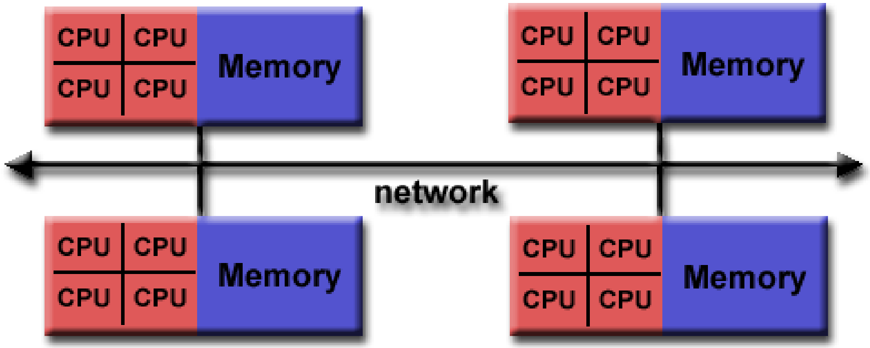
MPI Programming Model

- Originally, MPI was designed for distributed memory architectures, which were becoming increasingly popular at that time.



MPI Programming Model (Cont)

- As architecture trends changed, shared memory Symmetric Multiprocessors (SMPs) were combined over networks creating hybrid distributed memory / shared memory systems. MPI implementers adapted their libraries to handle both types of underlying memory architectures seamlessly.



MPI Programming Model (Cont)

- Today, MPI runs on virtually any hardware platform:
  - Distributed Memory
  - Shared Memory
  - Hybrid
- All parallelism is explicit: the programmer is responsible for correctly identifying parallelism and implementing parallel algorithms using MPI constructs.

General MPI Program Structure

MPI include file

Declarations, prototypes, etc.

Program Begins

.

. *Serial code*

.

Initialize MPI environment *Parallel code begins*

.

.

.

Do work and make message passing calls

.

.

.

Terminate MPI Environment *Parallel code ends*

.

. *Serial code*

.

Program Ends

MPI is Simple

- Many parallel programs can be written using just these six functions
  - MPI_Init
  - MPI_Comm_size
  - MPI_Comm_rank
  - MPI_Send
  - MPI_Recv
  - MPI_Finalize

MPI basic functions (subroutines)

- MPI_INIT: initialize MPI(MPI run time environment)
- MPI_COMM_SIZE: returns the number of MPI processes (important for decomposition)
- MPI_COMM_RANK: returns unique id for each processor (identify the PE) within the specified communicator
- MPI_SEND and MPI_RECV: MPI data is not shared but can be communicated. Each process has its own data. To communicate one process may send some data to another.
- MPI_FINALIZE: shuts down run time environment
- Note: This is crucial, failure to properly shutdown code could (will) cause other nodes in program to hang

Serial Program

```
#include <stdio.h>
int main(){
    printf ("Welcome to COMP428
        Tutorial!\n");
    return 0;
}
```

Serial to Parallel

```
#include <stdio.h>
#include”mpi.h”
int main(){
    MPI_Init (NULL, NULL);   /*start MPI*/
    printf ("Welcome to COMP428
        Tutorial!\n");
    MPI_Finalize();      /*shut down MPI*/
    return 0;
}
```

MPI Build Scripts:

- Available scripts are listed below:

Language	Script Name	Underlying Compiler
C	mpicc	gcc
	mpigcc	gcc
	mpiicc	Icc
C++	mpipgcc	pgcc
	mpiCC	g++
	mpig++	g++
	mpiicpc	icpc
	mpipgCC	pgCC

Format of MPI Calls:

- Programs must not declare variables or functions with names beginning with the prefix MPI_

C binding	
Format	rc = MPI_Xxxxxx(parameter,...)
Example	rc = MPI_Bsend(&buf, count)
Error Code	Returned as "rc". MPI_SUCCESS if successful

Communicators and Groups

- MPI uses objects called communicators and groups to define which collection of processes may communicate with each other.
- Most MPI routines require you to specify a communicator as an argument.
- Communicators and groups will be covered in more detail later. For now, simply use MPI_COMM_WORLD whenever a communicator is required - it is the predefined communicator that includes all of your MPI processes.

Rank

- Within a communicator, every process has its own unique, integer identifier assigned by the system when the process initializes. A rank is sometimes also called a "task ID".
- Ranks are contiguous and begin at zero.
- Used by the programmer to specify the source and destination of messages. Often used conditionally by the application to control program execution (if rank=0 do this / if rank=1 do that).

Error Handling:

- Most MPI routines include a return/error code parameter, as described in previous slide "Format of MPI Calls".
- However, according to the MPI standard, the default behavior of an MPI call is to abort if there is an error. This means you will probably not be able to capture a return/error code other than MPI_SUCCESS (zero).
- The standard does provide a means to override this default error handle. You can consult the error handling section of the MPI Standard located at http://www.mpi-forum.org/docs/mpi-11-html/node148.html.
- The types of errors displayed to the user are implementation dependent

Basic Features of MPI Programs

- Calls may be roughly divided into four classes:
  - Calls used to initialize, manage, and terminate communications
  - Calls used to communicate between pairs of processors. (Pair communication)
  - Calls used to communicate among groups of processors. (Collective communication)
  - Calls to create data types.

How to build a simple MPI program

- Have to include “mpi.h” .
- All MPI program must call MPI_INT as the first MPI call to initialize themselves.
- Call MPI_COMM_SIZE to get the number of processes that are running.
- Call MPI_COMM_RANK to determine there number between 0 and (size-1).
- Conditional process and general message passing can take place.
- Must call MPI_FINALIZE as the last call to an MPI library routine.

Thank You

Questions ?