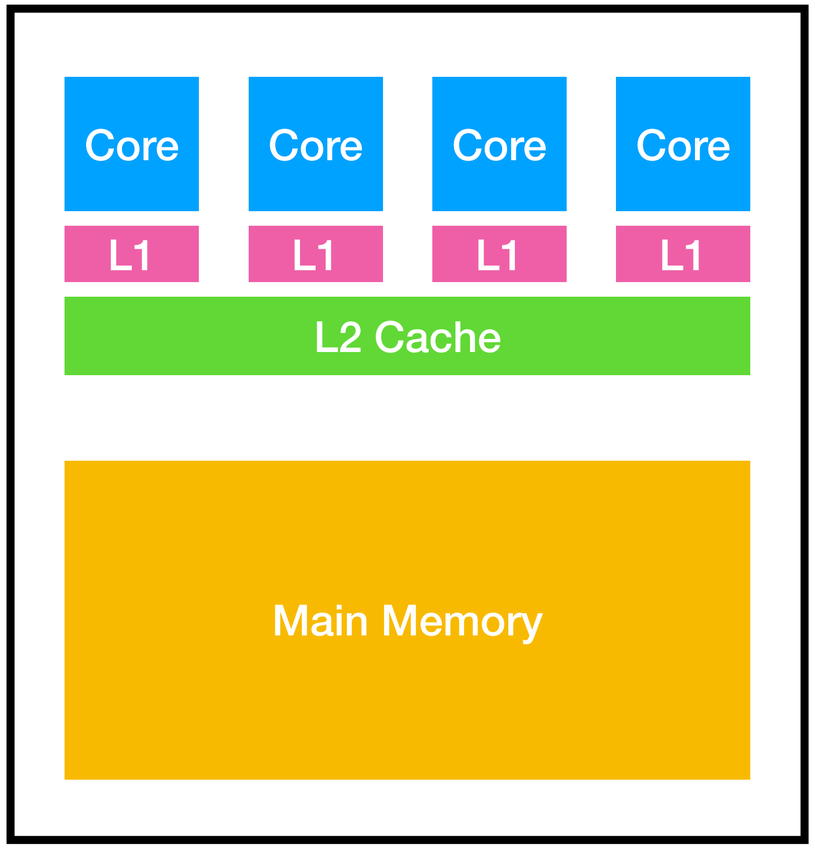
Introduction to Parallel Processing

Lecture 2 : Introduction to Distributed Systems

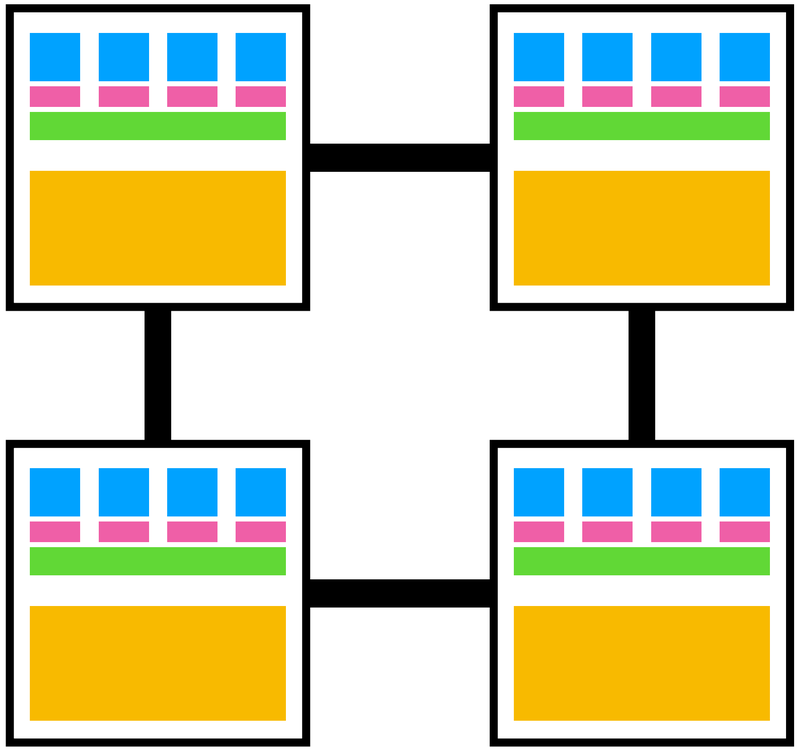
Professor Amanda Bienz

A Peak at Shared Memory



- All cores share main memory
- You have to worry about how the cores access variables in shared memory (false sharing, race conditions)
- **All cores have access to all variables allocated in shared memory**

Distributed Systems

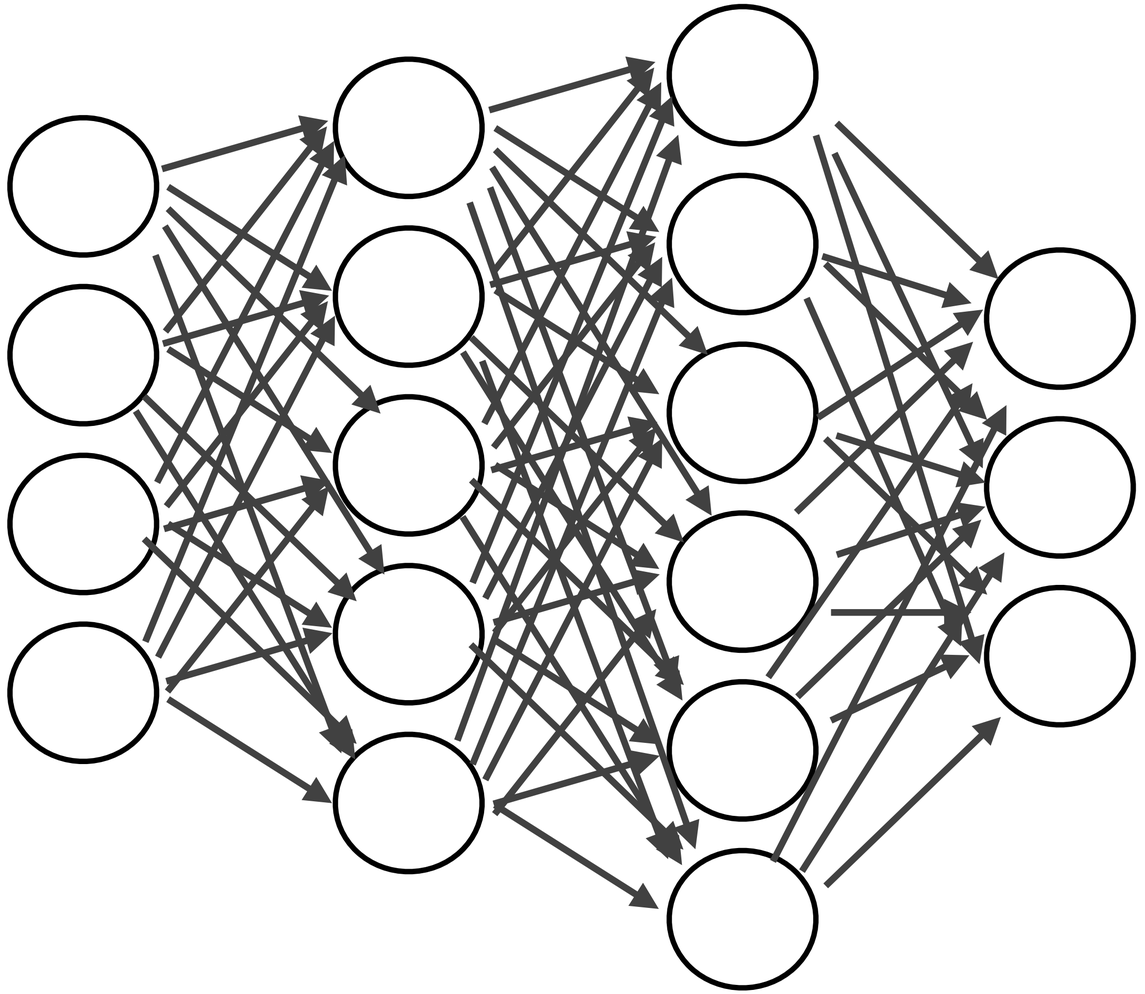


- Main memory on each node
- Nodes connected by network links
- Can work on incredibly large programs that wouldn’t fit in memory of a single node
- Data is split across the nodes, and each node cannot access data in another node’s main memory

Embarrassing Parallel Examples

- Data parallelism :
  - Say I want to read 100,000 image files, compress the image, and write the compressed image to a file
  - Can have one process read, compress and write 100,000 images
  - Can have two processes read, compress, and write 50,000 images
  - Could have 100,000 processes each read, compress, and write a single image

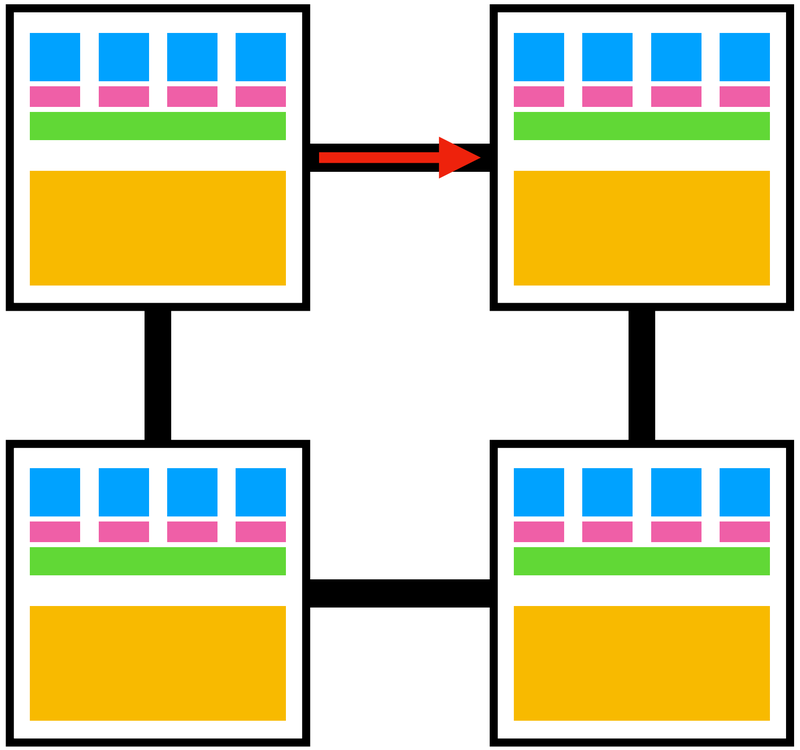
Embarrassing Parallel Examples



Baby

Dog

Message Passing



- Data is passed between nodes in messages:
  - Split into packets
  - Packets sent over links of network to destination node, where they are unpacked

MPI : Message Passing Interface

- Luckily, all of the message passing details are done under the hood for you
- MPI allows you to:
  - Send a message
  - Receive a message
  - Do some operation collectively among all processes (i.e. reduction)
  - Barrier : wait for all processes to reach
  - Many many other things

MPI Program Format

- Typically, entire program is parallelized
- Each process executes the entire program
- Typically, don’t want every process executing the same thing
- For example, if all processes send, and none receive, your program will get stuck and hang indefinitely.
- Need to write program so that instructions are split among the processes

How to compile and run

- **Compiling :** mpicc, mpicxx, mpifort
- **Running:** mpirun -n <num_procs> ./<programname>

Hello MPI World

```c
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[])
{
    MPI_Init(&argc, &argv);

    printf("Hello MPI World!\n");

    MPI_Finalize();
}
~
```

```
abienz@Amandas-MacBook-Pro cs-442-542-f20 % mpirun -n 4 ./hello_world
_mpi
Hello MPI World!
Hello MPI World!
Hello MPI World!
Hello MPI World!
```

Two Helpful Methods

- **MPI_Comm_rank(MPI_COMM_WORLD, &rank)** : sets ‘rank’ variable to the process’s ID (unique, between 0 and number of processes)

- **MPI_Comm_size(MPI_COMM_WORLD, &num_procs)** : sets ‘num_procs’ variable to hold the number of processes active in the program

- What is **MPI_COMM_WORLD**? This is called a communicator. It allows you to communicate with any other process that is also a part of the communicator. This specific communicator contains all processes in program, and is all you need to use for now.

Hello MPI World Rank

```c
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[])
{
    MPI_Init(&argc, &argv);

    int rank, num_procs;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &num_procs);

    printf("Hello MPI World from rank %d of %d!\n", rank, num_procs);

    MPI_Finalize();
}
~
```

```
abienz@Amandas-MacBook-Pro cs-442-542-f20 % mpicc hello_world_mpi.c -
o hello_world_mpi
abienz@Amandas-MacBook-Pro cs-442-542-f20 % mpirun -n 4 ./hello_world
_mpi
Hello MPI World from rank 0 of 4!
Hello MPI World from rank 1 of 4!
Hello MPI World from rank 2 of 4!
Hello MPI World from rank 3 of 4!
```