BMCS3003 Distributed Systems and Parallel Computing

L03 - Memory Management

Presented by

Assoc Prof Ts Dr Tew Yiqi
May 2023

Table of contents

01

Centralised Memory Management

An architecture in which memory allocations are grouped based on their type, lifetime, or other requirements.

As computing tasks get larger and larger, may need to enlist more computers to the job.

Tasks feeder

As computing tasks get larger and larger, may need to enlist more computers to the job.

Bigger:

more memory and storage

Faster:

each processor is faster

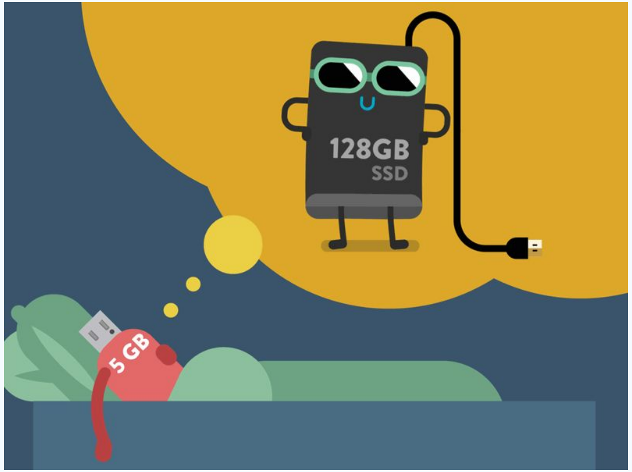
More tasks:

do many computations simultaneously

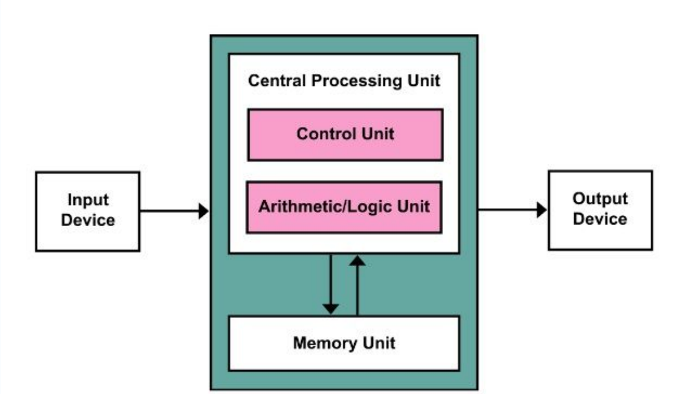
Review of Centralised Memory Management

Virtual memory

- Extending the size of available memory beyond its physical size of RAM
- Paging versus segmentation
- Internal versus external fragmentation
- Segment placement algorithms
- Page replacement algorithms
    - page faults and thrashing



Simple Memory Model

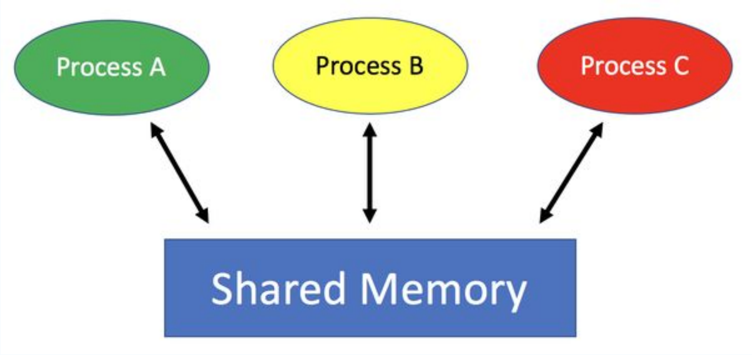


In Simple Memory Model access times for all processors are equal.

Requires strict control of degree of multi-programming.

Often does not use virtual memory or caching because of overhead.

Shared Memory Model



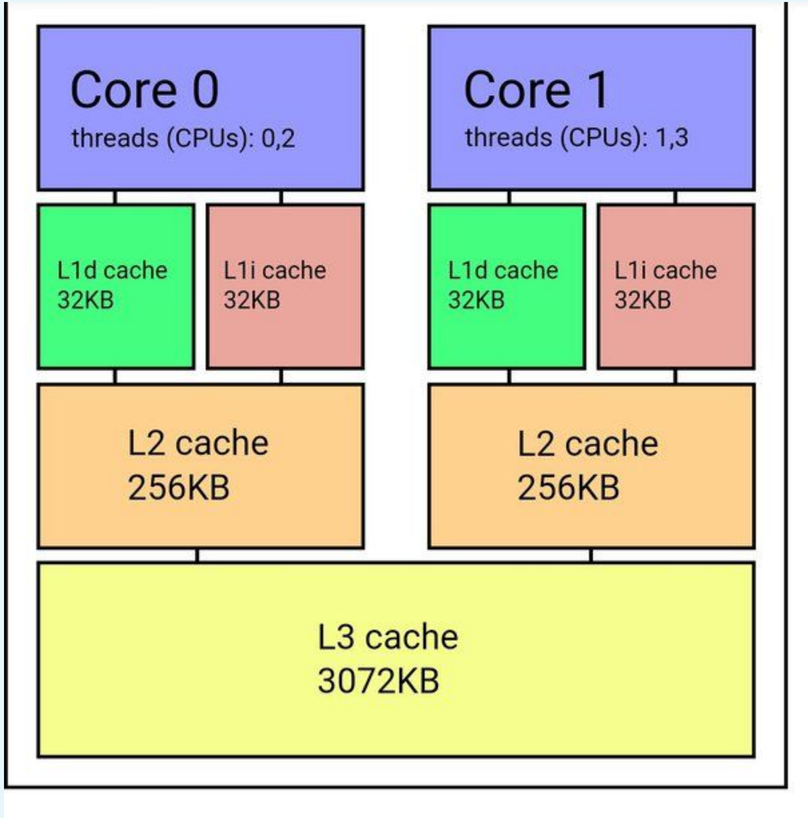
Used for inter-process communication.

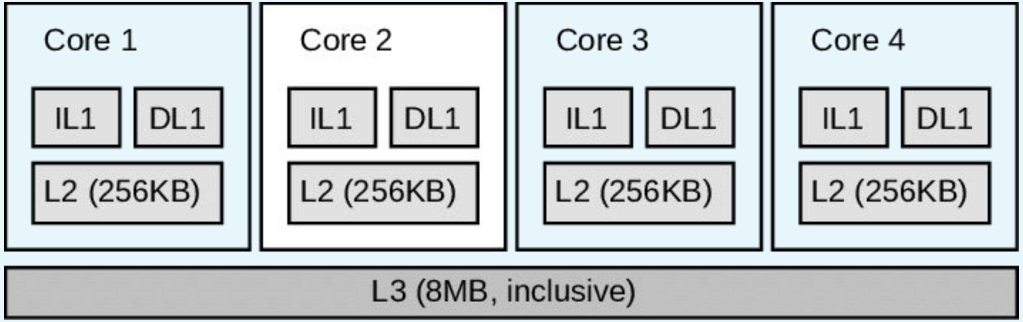
Multiple processes share memory locations.

May includes physical RAMs, local cache, and secondary storage.

Memory access takes place via common bus thus a possibility of a **bus contention**.

Example: OpenMP



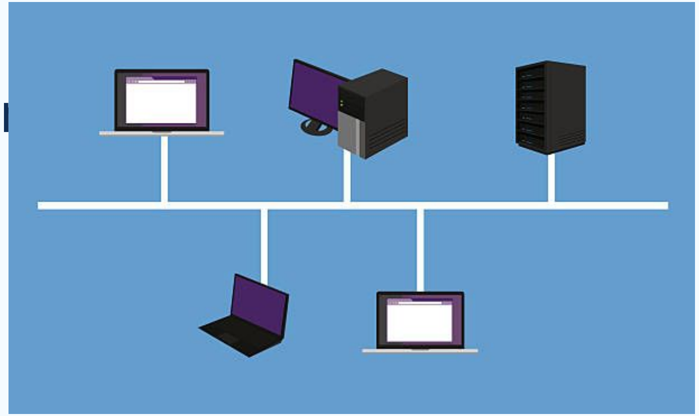


Intel Core i&-970 Processor

Bus Contention

Bus contention occurs when the demand for bus access is excessive.

Bus contention may cause a bottleneck in a shared memory system.



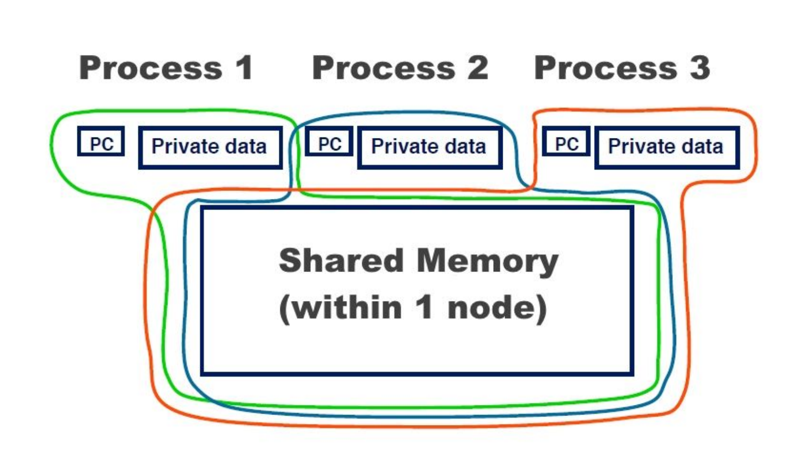
Larger multiprocessor systems (>32 CPUs) cannot use a single bus to interconnect CPUs to memory modules, because bus contention becomes unmanegeable.

Shared Memory Performance

Performance is an important issue in a *Shared Memory System*.

Important issues:

- scalability ability to accommodate growth without sacrificing performance
- real time needs
  - overlap of communication and computation
  - prefetching of data
- non local memory references are expensive (up to 10:1 ratio)

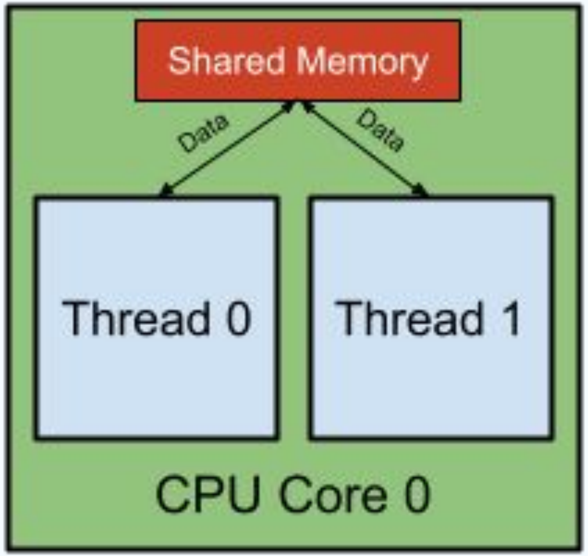


More reference: [OpenMP MIMD]

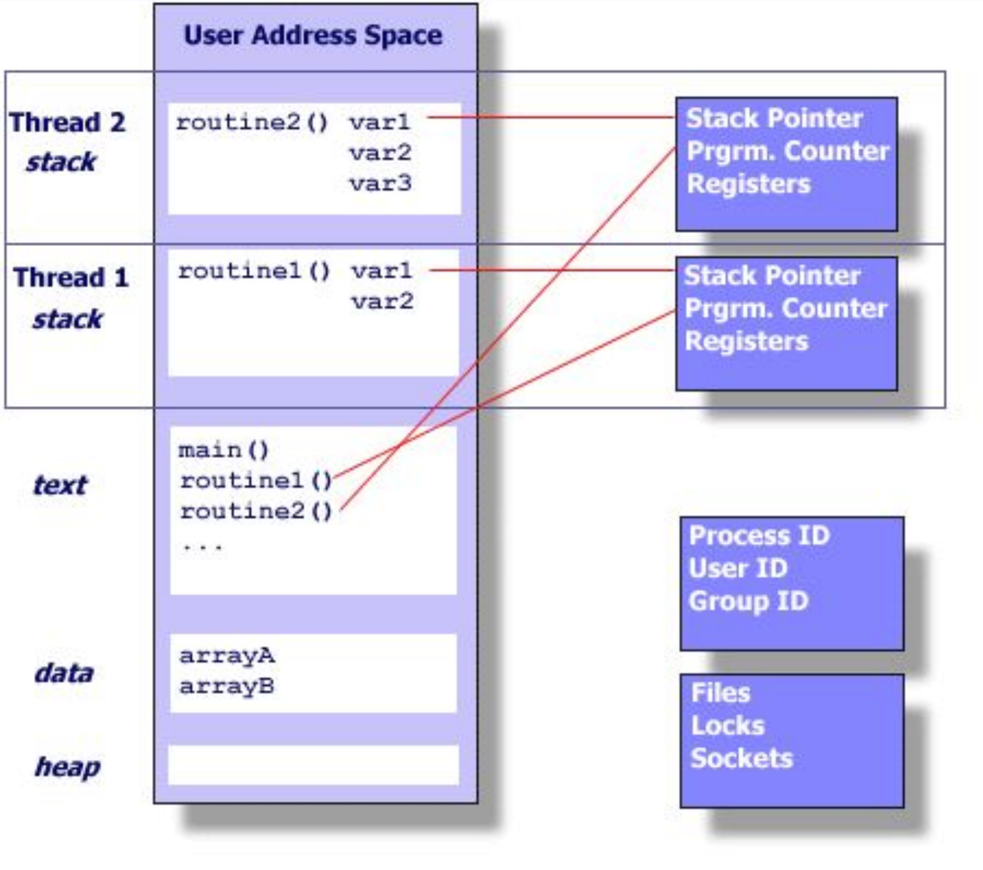
Shared Memory Performance

Large symmetric multiprocessor systems offered more compute resources to solve large computationally intense problems.

- Threading is the most popular shared memory programming technique
- Advantage:
  - makes it easy for a developer to divide up work, tasks, and data.
- Disadvantage:
  - data races
- E.g., OpenMP



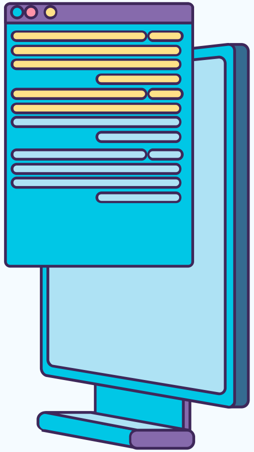
Threads are lightweight process and share process state among multiple threads. This greatly reduces the cost of switching contexts



OpenMP is an acronym for **Open M**ulti-**P**rocessing. OpenMP is a directive-based Application Programming Interface (API) for developing parallel programs on shared memory architectures. The OpenMP standard is maintained by the OpenMP ARB, a corporation whose board of directors includes representatives from many major computer hardware and software vendors.

Source: Cornell University

> "The OpenMP ARB (Architecture Review Boards) mission is to standardize directive-based multi-language high-level parallelism that is performant, productive and portable."



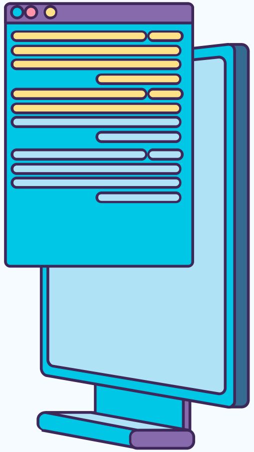
```
// A pragma defines beginning of a
parallel region and specifies iterations
of the following for loop should be
spread across openmp threads and is the
extent of the parallel region

#pragma omp parallel for
for (i =0; i < n; i++)
{
    . . . ;
    computations to be completed ;
    . . . ;
}
```

Official Documentation
OpenMP API

More Lessons:
Lawrence Livermore National Laboratory

Jaka's Corner



```c
#include <unistd.h>
#include <stdlib.h>
#include <omp.h>
#include <stdio.h>
#define THREADS 4
#define N 16
int main ( ) {
      int i;
      #pragma omp parallel for schedule(static) num_threads(THREADS)
      for (i = 0; i < N; i++) {
            /* wait for i seconds */
            sleep(i);
            printf("Thread %d has completed iteration %d.\n",
            omp_get_thread_num( ), i);
      }
      /* all threads done */
      printf("All done!\n");
      return 0;
}
```

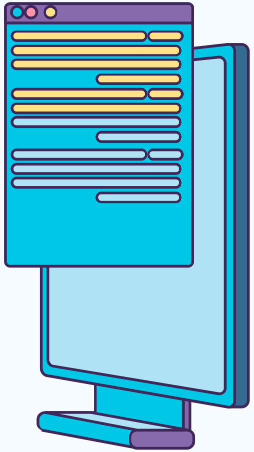
What is this code doing ?

What do the OpenMP semantics specify ?

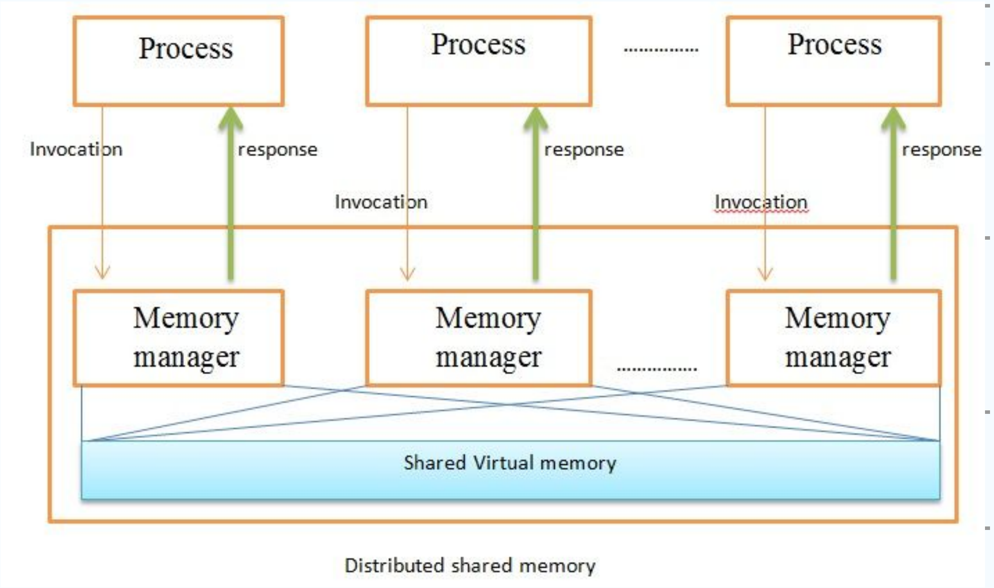
How might you accomplish this ?

Under the hood:

1. Scheduling
2. Work (in parallel)
3. Reduction
4. Barrier ...



Distributed Shared Memory (DSM)



Source: [Mehrnazzhian]

- Concept introduced in 1989.
- DSM presents a logical shared memory for multi-computer systems.
- DSM maintains communication and data consistency for applications.
- Usually portions of local memory are mapped onto DSM.
- DSM maintains a directory service for all data residing in the system.

Distributed Shared Memory Management

DSM Management involves two main decisions:

- How to distribute shared data ?
- How many readers and writers should be allowed for a shared data segment?

Distributing Shared Data

Replication

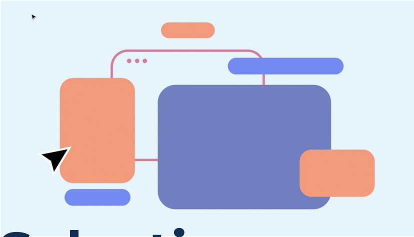
maintaining multiple copies of shared data at various locations

Migration

moving the copy of shared data to various locations (only one copy of shared data is allowed)

DSM Performance

Issue to be considered



Thrashing

How to avoid thrashing, i.e. the situation where data constantly travels between various locations?

Block Size Selection

What is the best size for sharable data block? (page size, packet size, segment size)

Data Location

Which is the best original site for shared data location?

Implementation Location

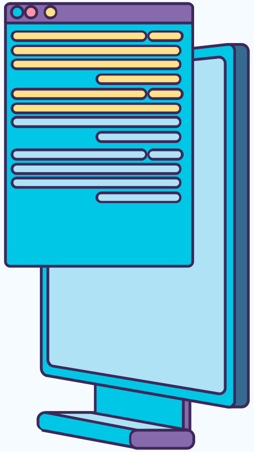
Where should be DSM implemented? (hardware or software or both)

Distributed Memory Programming

Explicitly packaged data and sent it to another system in the cluster.

Message Passing Interface (MPI)

- Explicitly handle the decomposition of the problem across the cluster as well as make sure messages were sent and received in the proper order.



What is MPI ?

- ➢ Message passing is a programming model for coordination processes on these computers
- ➢ Each process as its own data
- ➢ Data is exchanged by sending and receiving messages
- ➢ Programmers use these tools to build multi-computer computation

Official MPI community:
MPI Forum

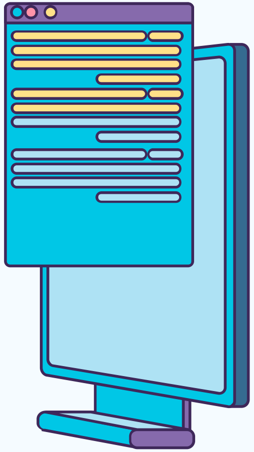
Exercises:
UPM

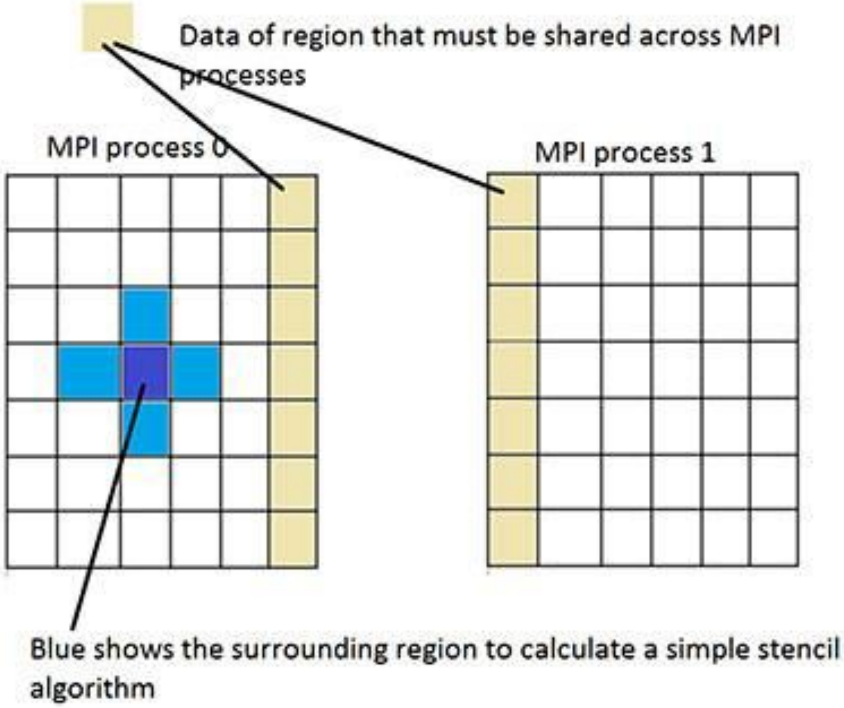
LLNL

Rabernat (Python)

MPI Tutorial (AWS)

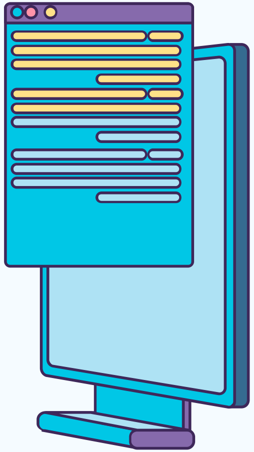
Microsoft MPI





C - calls for MPI process 0 to send and receive border data with MPI process 1

```
CALL MPI_ISEND(SBUF,  scount, MPI_REAL, 1, stag, MPI_COMM_WORLD, ireq,ierr)
CALL MPI_IRECV(RBUF,rcount,MPI_REAL,0,rtag,MPI_COMM_WORLD, rreq,irerr)
```



What is MPI ?

- ➢ MPI is a standard library for message passing.
- ➢ Ubiquitous in high performance technical computing.
- ➢ Nearly every big academic or commercial simulation or data analysis running on multiple nodes uses MPI directly or indirectly.

Official MPI community:
MPI Forum

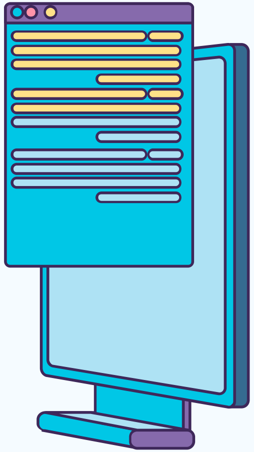
Exercises:
UPM

LLNL

Rabernat (Python)

MPI Tutorial (AWS)

Microsoft MPI



Why bother with MPI ?

Don't we have network libraries?

- ➢ Optimized for performance
- ➢ Will take advantage of fastest transport found:
  - ○ Shared memory (Within a computer)
  - ○ Fast cluster interconnects (Infiniband , Myrinet ..)between computers (nodes)
  - ○ TCP/IP if all else failes

Official MPI community:
[MPI Forum]

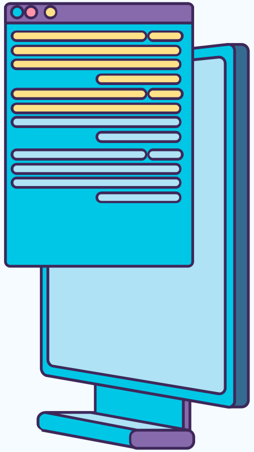
Exercises:
[UPM]

[LLNL]

[Rabernat (Python)]

[MPI Tutorial (AWS)]

[Microsoft MPI]



Why bother with MPI ?

Don't we have network libraries?

- ➢ Optimized for performance.
- ➢ Will take advantage of fastest transport found.
- ➢ Enforces certain guarantees:
    - ○ Reliable messages
    - ○ In order arrival of messages

Official MPI community:
[MPI Forum]

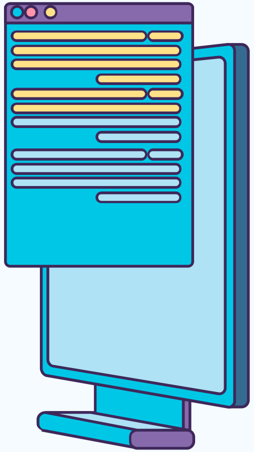
Exercises:
[UPM]

[LLNL]

[Rabernat (Python)]

[MPI Tutorial (AWS)]

[Microsoft MPI]



Why bother with MPI ?

Don't we have network libraries?

- ➢ Optimized for performance.
- ➢ Will take advantage of fastest transport found.
- ➢ Enforces certain guarantees.
- ➢ Designed for multi-node technical computing:
  - ○ Completely standard: Available everywhere
  - ○ Has specialized routines for collective operations

Official MPI community:
MPI Forum

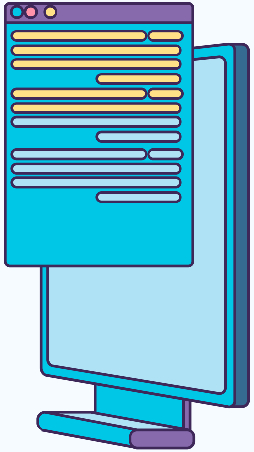
Exercises:
UPM

LLNL

Rabernat (Python)

MPI Tutorial (AWS)

Microsoft MPI



MPI basic functions

Set up / teardown

MPI_INIT
MPI_FINALIZE

Who am I?

MPI_COMM_SIZE
MPI_COMM_RANK

Message Passing

MPI_SEND
MPI_RECV

More Examples:

PDC-support

By Jarno Rantaharju et al.

Universiti of Minnesota

By David Porter & Mark Nelson

Princeton Plasma Physics Lab

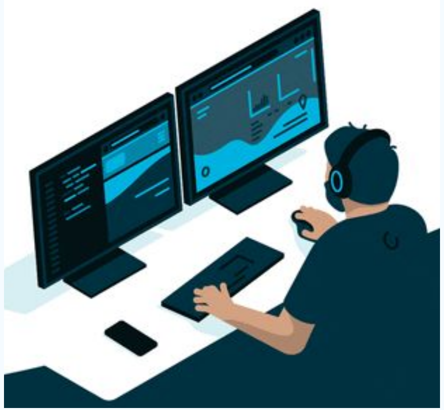
By Stephane Ethier

Question

So is MPI the way to do all distributed computing ?

A Simple MPI Program

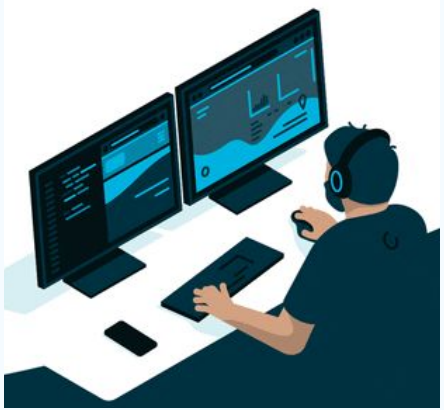
```
#include "mpi.h"
#include <stdio.h>
Int main (int argc, char *argv[]){
    Int rank, size ;
    MPI_Init(NULL NULL);
    MPI_Comm_rank(MPI_COMM_WORLD , &rank);
    MPI_Comm_size(MPI_COMM_WORLD , &size);
    printf("Hello! I am %d of %d\n", rank, size );
    MPI_Finalize();
    return 0;
}
```



- **MPI_COMM_WORLD** indicates the set of all processes
- All MPI functions return error code
- Update values by passing pointers as arguments

A Simple MPI Program

```
#include "mpi.h"
#include <stdio.h>
Int main (int argc, char *argv[]){
    Int rank, size ;
    MPI_Init(NULL NULL);
    MPI_Comm_rank(MPI_COMM_WORLD , &rank);
    MPI_Comm_size(MPI_COMM_WORLD , &size);
    printf("Hello! I am %d of %d\n", rank, size );
    MPI_Finalize();
    return 0;
}
```



- Compile and Run with cmd:

```
mpi exec np 2 ./hello
```

```
Hello! I am 0 of 2
Hello! I am 1 of 2
```

What just happened ?

- ➢ mpiexec launched 4 processes on your computer
- ➢ Each process ran the program hostname
- ➢ Each ran independently
- ➢ Can do the same thing with 8, 27, or 93 processes
- ➢ For performance reasons, usually use as many processes as there are processors

... but Painful !

OpenMP

- ➔ Add pragmas to existing program
- ➔ Compiler + runtime system arrange for parallel execution
- ➔ Rely on shared memory for communication

MPI

- ➔ Must rewrite program to describe how single process should operate on its data and communicate with other processes.
- ➔ Explicit data movement: programmer must say exactly what data goes where and when.
- ➔ Advantage: Can operate on systems that don't have shared memory.

MPI - Process Identification

When running with P processes:

- Size : P
  Total number of processes
- Rank: Number between 0 and P 1
  Identity of individual process

- Library Functions
  MPI_Comm_size
  MPI_Comm_rank

Comparing Frameworks

Multiple Processes

MPI

Fixed processes number created
All execute code starting with main
Isolated address spaces

FORK

Processes created during program execution
Replicate address space upon creation, but then isolated

Multiple Threads

pthread_create

Threads created during program execution
Shared address space

OpenMP

Set of threads created at beginning of program
Recruited to execute tasks spawned by #pragma omp parallel
Shared address space

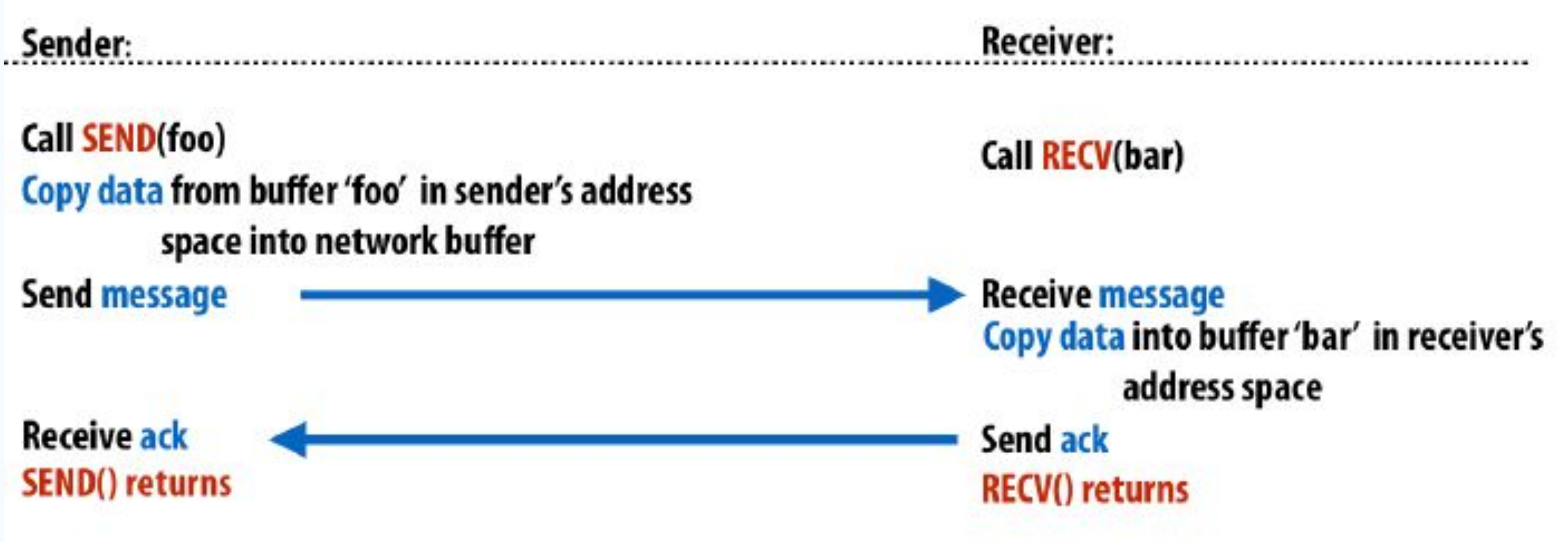
Synchronous Sending and Receiving

send(): call returns when sender receives acknowledgement that message data resides in address space of receiver.

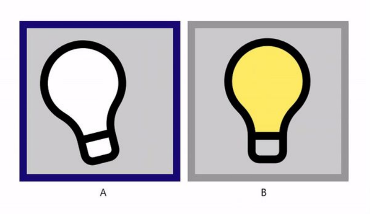
recv(): call returns when data from received message is copied into address space of receiver and acknowledgement sent back to sender

Potential for deadlock if all processes attempt to send and then receive

Synchronous Sending and Receiving



Asynchronous Sending and Receiving



send(): call returns immediately

- Buffer provided to send() cannot be modified by calling thread since message processing occurs concurrently with process execution

- Calling thread can perform other work while waiting for message to be sent.

recv(): posts intent to receive in the future, returns Immediately

- Use checksend(), checkrecv() to determine actual status of send/receipt

- Calling thread can perform other work while waiting for message to be received

Asynchronous Sending and Receiving

Sender:

Call SEND(foo)
SEND returns handle h1

Copy data from 'foo' into network buffer
Send message →

Call CHECKSEND(h1) // if message sent, now safe for thread to modify 'foo'

Receiver:

Call RECV(bar)
RECV(bar) returns handle h2

Receive message
Messaging library copies data into 'bar'
Call CHECKRECV(h2)
// if received, now safe for thread
// to access 'bar'

RED TEXT = executes concurrently with application thread

MPI Send / Receive Operation

Synchronous

MPI_Send

MPI_Recv

Asynchronous

MPI_Isend

MPI_Irecv

MPI_Wait

Memory Migration

Memory Migration requires two fundamental decisions:

- When in the migration process will we migrate memory?
- How much memory needs to be migrated?

Vector Add Example

Example

Steps to parallelize Vector Add with GPU

1. Make the variables accessible to both CPU and GPU
2. Launch configuration of the kernel function
3. Parallelize the kernel function to run on GPU

More Examples

Matrix Transpose

https://www.hpc.cineca.it/content/exercise-15

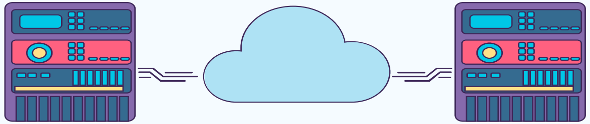
Matrix Multiplication

https://www.hpc.cineca.it/content/exercise-16

2D Laplace Equation

https://www.mcs.anl.gov/research/projects/mpi/tutorial/mpiexmpl/src/jacobi/C/main.html

Summary of Chapter



- **Shared address space**

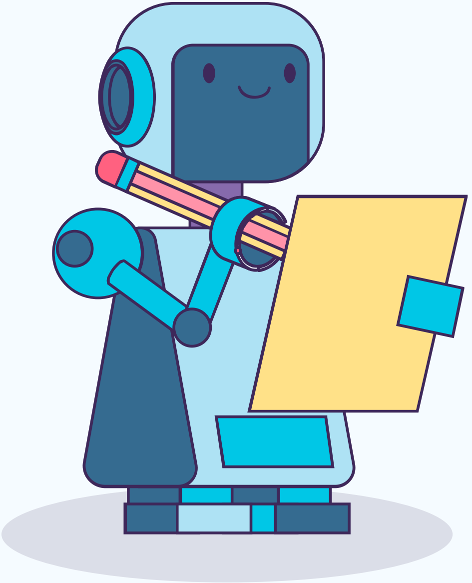
> Communication is unstructured, implicit in loads and stores.
> Natural way of programming, but can shoot yourself in the foot easily.
> Program might be correct, but not perform well.

- **Message passing**

> Structure all communication as messages.
> Often harder to get first correct program than shared address space.
> Structure often helpful in getting to first correct, scalable program.

- **Data parallel**

> Structure computation as a big "map" over a collection.
> Assumes a shared address space from which to load inputs/store results, but model severely limits communication between iterations of the map (goal: preserve independent processing of iterations).
> Modern embodiments encourage, but don't enforce, this structure.



Thank you

CREDITS: This presentation template was created by **Slidesgo**, and includes icons by **Flaticon**, and infographics & images by **Freepik**