OpenMP & MPI

CISC 879

Tristan Vanderbruggen & John Cavazos

Dept of Computer & Information Sciences

University of Delaware

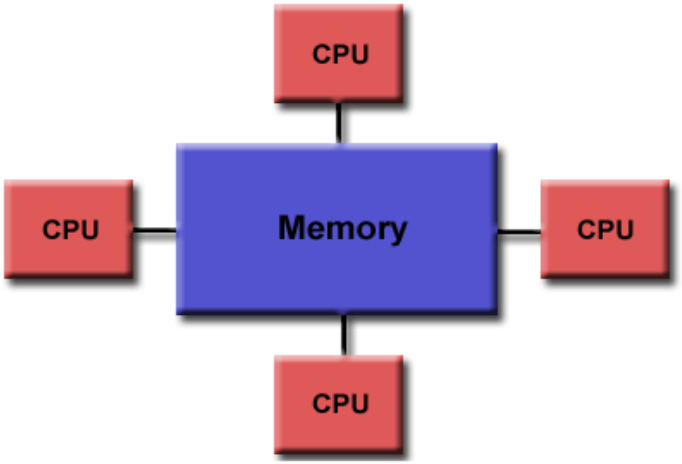
Lecture Overview

- Introduction
- OpenMP
  - Model
  - Language extension: directives-based
  - Step-by-step example
- MPI
  - Model
  - Runtime Library
  - Step-by-step example
- Hybrid of OpenMP & MPI
- Conclusion

OpenMP: Open Multi-Processing
Intranode parallelism

1.1 - OpenMP: Model

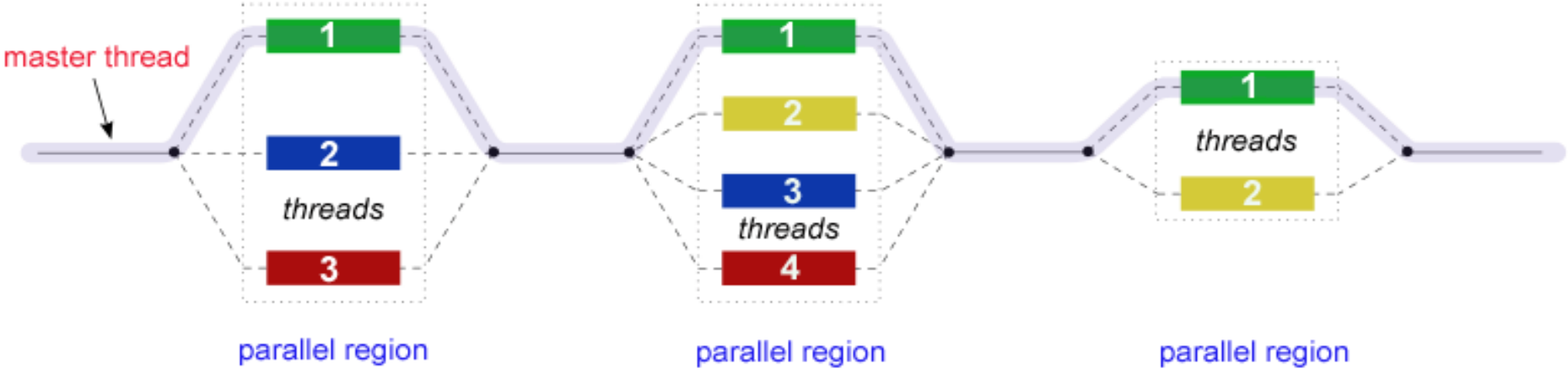
- Shared Memory Model:
  - multi-processor/core



Source: https://computing.llnl.gov/tutorials/openMP/

1.1 - OpenMP: Model

- Thread-level Parallelism:
  - parallelism through threads
  - typically: number of threads match number of cores
- Fork - Join Model:



Source: https://computing.llnl.gov/tutorials/openMP/

1.1 - OpenMP: Model

- Explicit Parallelism:
  - programmer has full control over parallelization
  - can be as simple as inserting compiler directives in a serial program
  - **or**, as complex as inserting subroutines to set multiple levels of parallelism, locks and even nested locks

Source: https://computing.llnl.gov/tutorials/openMP/

1.2 - OpenMP: Language

- OpenMP is not exactly a language.
  - It is an extension for C and Fortran.
- It is a **Directive-Based Language Extension**
- It works by annotating a sequential code

Source: https://computing.llnl.gov/tutorials/openMP/

1.2 - OpenMP: Language

- in C, it uses pragmas

  #pragma omp construct [clause, ...]

- in Fortran, it uses *sentinels* (!omp, Comp, or *$omp):

  !$OMP construct [clause, ...]

Source: https://computing.llnl.gov/tutorials/openMP/

1.2 - OpenMP: Language

- ***constructs*** are functionalities of the language
- ***clauses*** are parameters to those functionalities
- ***construct*** + ***clauses*** = ***directive***

Source: https://computing.llnl.gov/tutorials/openMP/

1.3 - OpenMP: Step-by-step Example

Two examples:

- the classic HelloWorld
- a matrix multiplication

1.3 (a) - OpenMP: Hello World

```c
#include <omp.h>

#include <stdio.h>
#include <stdlib.h>

int main() {
  int nthreads, tid;

  /* Fork a team of threads giving them their own copies of variables */
  #pragma omp parallel private(nthreads, tid)
  {

    /* Obtain thread number */
    tid = omp_get_thread_num();
    printf("Hello World from thread = %d\n", tid);

    /* Only master thread does this */
    if (tid == 0) {
      nthreads = omp_get_num_threads();
      printf("Number of threads = %d\n", nthreads);
    }

  }  /* All threads join master thread and disband */

}
```

OpenMP: Environment Variables

- OpenMP has a set of environment variables that control the runtime execution

- OMP_NUM_THREADS=num
  - default number of threads contained by a parallel region
- OMP_SCHEDULE=algorithm
  - algorithm = dynamic or static
  - the algorithm to be use for scheduling

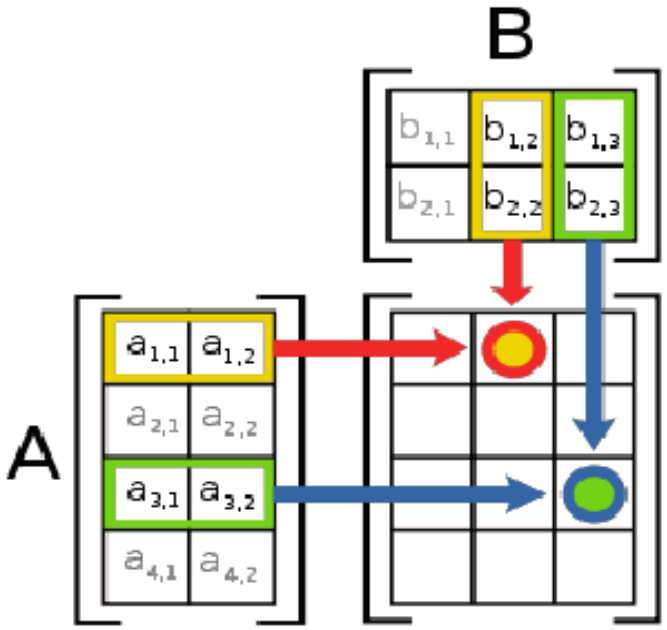
1.3 (a) - OpenMP: Hello World

- Compile:
  - $> gcc -fopenmp helloworld-omp.c -o helloworld-omp
- Run:
  - $> qlogin -pe threads 8
  - $> cd hpc-II
  - $> export OMP_NUM_THREADS=8
  - $> ./helloworld-omp

```
Hello World from thread = 2
Hello World from thread = 7
Hello World from thread = 0
Number of threads = 8
Hello World from thread = 3
Hello World from thread = 6
Hello World from thread = 5
Hello World from thread = 1
Hello World from thread = 4
```

1.3 (b) - OpenMP: Matrix Multiply

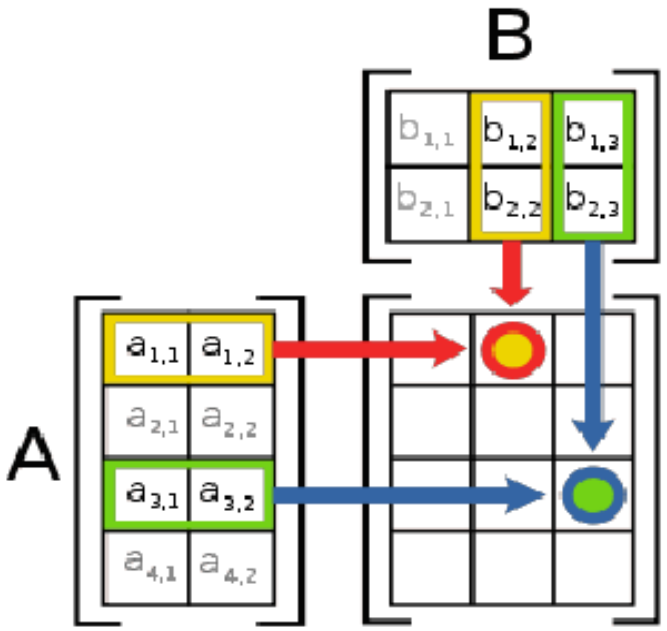
$$(AB)_{i,j} = \sum_{k=1}^{p} A_{ik} B_{kj}$$



1.3 (b) - OpenMP: Matrix Multiply

```
#include "size-def.h"

float A[N][P]; // op 1
float B[P][M]; // op 2
float C[N][M]; // res

int main() {
  unsigned long i, j, k;

  for (i = 0; i < N; i++)
    for (k = 0; k < P; k++)
      A[i][k] = rand();
  for (k = 0; k < P; k++)
    for (j = 0; j < M; j++)
      B[k][j] = rand();
  for (i = 0; i < N; i++)
    for (j = 0; j < M; j++)
      C[i][j] = rand();

  for (i = 0; i < N; i++) {
    for (j = 0; j < M; j++) {
      for (k = 0; k < P; k++) {
        C[i][j] += A[i][k] * B[k][j];
      }
    }
  }

  return 0;
}
```

$$(AB)_{i,j} = \sum_{k=1}^{p} A_{ik} B_{kj}$$



1.3 (b) - OpenMP: Matrix Multiply

```
#include <omp.h>

#include "size-def.h"

float A[N][P]; // op 1
float B[P][M]; // op 2
float C[N][M]; // res

int main() {
  unsigned long i, j, k;

  for (i = 0; i < N; i++)
    for (k = 0; k < P; k++)
      A[i][k] = rand();
  for (k = 0; k < P; k++)
    for (j = 0; j < M; j++)
      B[k][j] = rand();
  for (i = 0; i < N; i++)
    for (j = 0; j < M; j++)
      C[i][j] = rand();
```

1.3 (b) - OpenMP: Matrix Multiply

```
  #pragma omp parallel shared(A,B,C) private(i,j,k)
  {

    #pragma omp for schedule (static)
    for (i = 0; i < N; i++) {
      for (j = 0; j < M; j++) {
        for (k = 0; k < P; k++) {
          C[i][j] += A[i][k] * B[k][j];
        }
      }
    }

  }

  return 0;
}
```

1.3 (b) - OpenMP: Matrix Multiply

- #pragma omp parallel shared(A,B,C) private(i,j,k)
  - create a parallel region
    - fork a ***team*** of threads (usually as many as cores)
  - arrays A, B, C are *shared* among the threads
  - the "iterators" are *private* to each threads

1.3 (b) - OpenMP: Matrix Multiply

- #pragma omp for schedule (static)
  - declare a parallel for-loop
    - to be executed by the ***team***
  - *schedule* precise how the iterations have to be divided
    - static/dynamic
    - chunk size

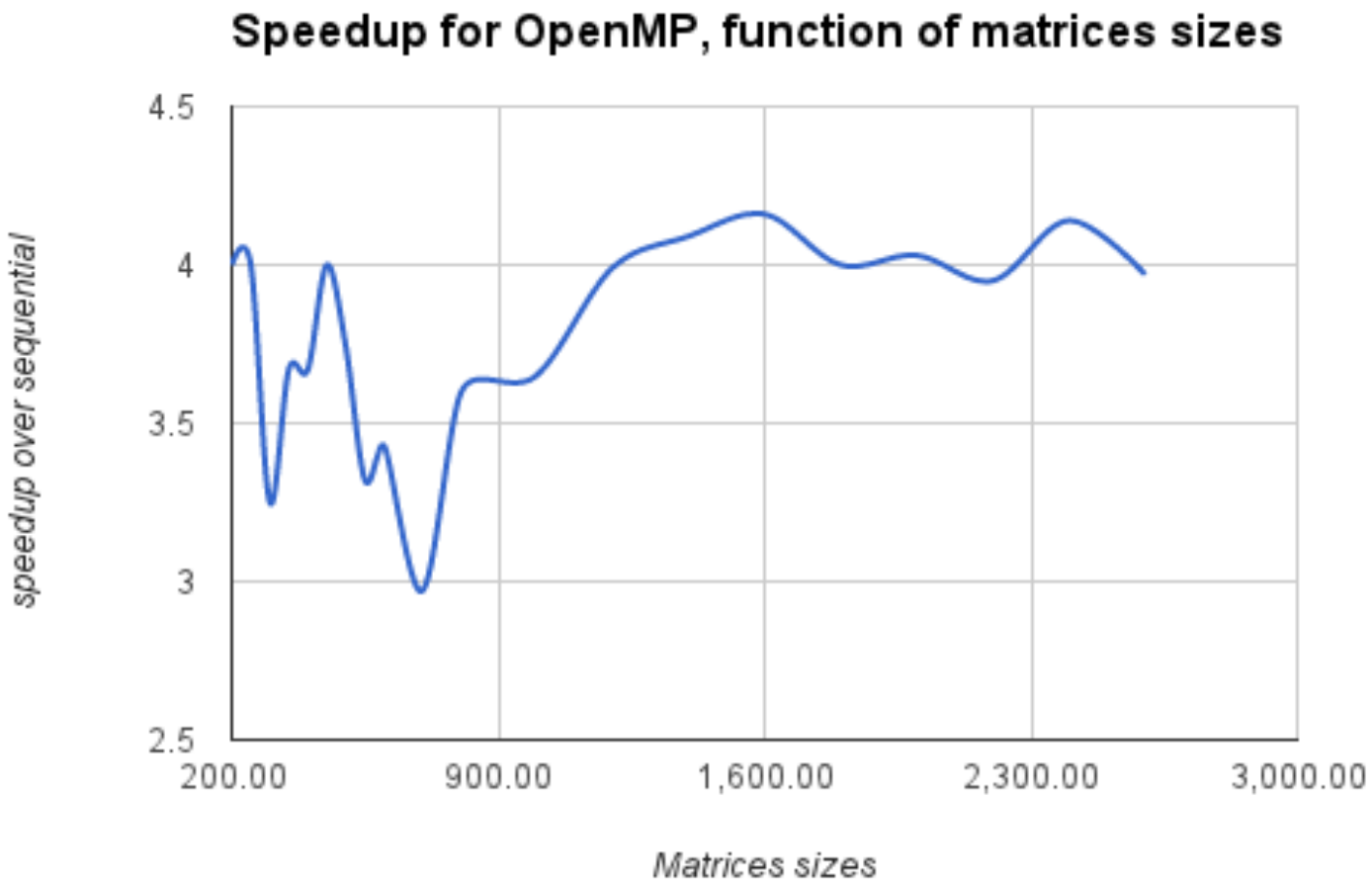
1.3 (b) - OpenMP: Matrix Multiply

- on Intel i7 4 cores
- for 512x512 float matrices
- Sequential: 0.92s
- OpenMp : 0.24s

Speedup of 3.83

1.3 (b) - OpenMP: Matrix Multiply

But the speedup depends on the input size:



1.4 - OpenMP: Construct

- Constructs:

  a. ***barrier*** : synchronisation point

  b. ***single*** : only executed by one thread of the ***team***

  c. ***master*** : only executed by the master

  d. ***critical*** : only one thread at anytime

  e. ***sections*** / ***section*** : declare task parallelism

1.4 - OpenMP: Clause

- clauses:

  a. ***shared***/***private*** apply to variables list

  b. ***default*** policy for variables sharing
     - either ***shared*** or ***none***

  c. ***firstprivate*** take a list of ***private*** variables to be initialized

  d. ***lastprivate*** take a list of ***private*** variables to be copy out

  e. ***reduction*** take an operation and a list of scalar variables

  f. ***num_thread*** either
     - from the team to be used
     - in the team

1.4 - OpenMP: Barrier example

```
int main() {

  double startTime;

  #pragma omp parallel private (startTime) num_threads(4)
  {
    startTime = omp_get_wtime();
    // Each thread sleep ID second (master thread sleep 0 s)
    while( (omp_get_wtime() - startTime) < (double)(omp_get_thread_num()));
    printf("I (%d) finish to count\n", omp_get_thread_num());
    // Each thread will wait other
    #pragma omp barrier
    printf("I (%d) pass the Barrier\n", omp_get_thread_num());

    #pragma omp single
    {
      printf("I (%d) am the only one executing this code\n", omp_get_thread_num());
    }

    #pragma omp master
    {
      printf("I (%d) am the Master\n", omp_get_thread_num());
    }
  }

  return 0;
}
```

1.4 - OpenMP: Barrier example

```
tristan@tristan-laptop:~/classes/hpc-lecture/lecture2$ ./barrier-omp
I (0) finish to count
I (1) finish to count
I (2) finish to count
I (3) finish to count
I (3) pass the Barrier
I (3) am the only one executing this code
I (2) pass the Barrier
I (0) pass the Barrier
I (1) pass the Barrier
I (0) am the Master
```

1.4 - OpenMP: Reduction

```c
#include <omp.h>

#include <stdio.h>

int main() {
  int   i, n, chunk;
  float a[100], b[100], result;

  n = 100;
  chunk = 10;
  result = 0.0;
  for (i=0; i < n; i++) {
    a[i] = i * 1.0;
    b[i] = i * 2.0;
  }

  #pragma omp parallel for default(shared) private(i) \
              schedule(static, chunk) reduction(+:result)
  for (i=0; i < n; i++)
    result = result + (a[i] * b[i]);

  printf ("result=%f\n", result);

  return 0;
}
```

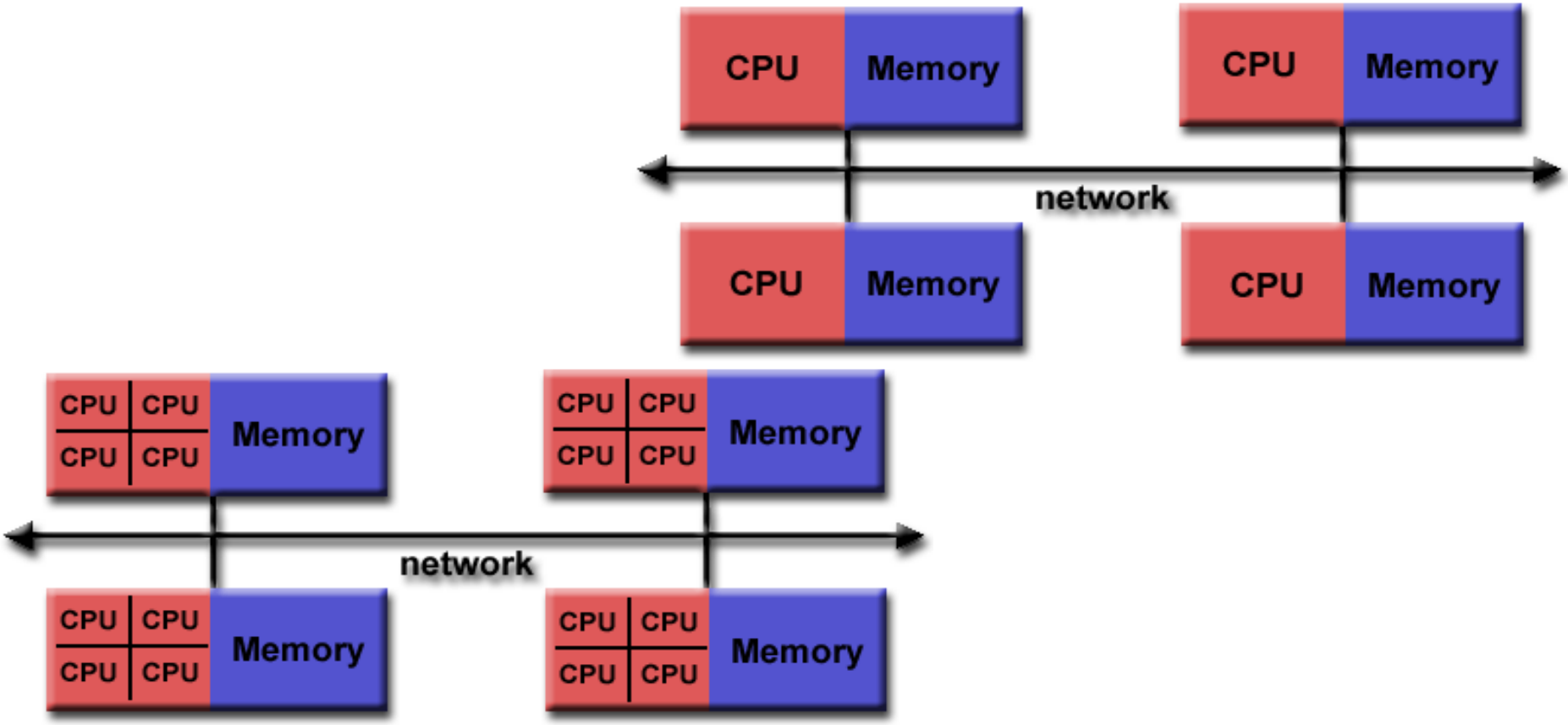
Questions ?

Any questions about OpenMP?

Message Passing Interface: internodes parallelism

2.1 - MPI: Model

- Distributed Memory, originally
- today implementation support shared memory SMP



Source: https://computing.llnl.gov/tutorials/mpi/

2.2 - MPI: Language

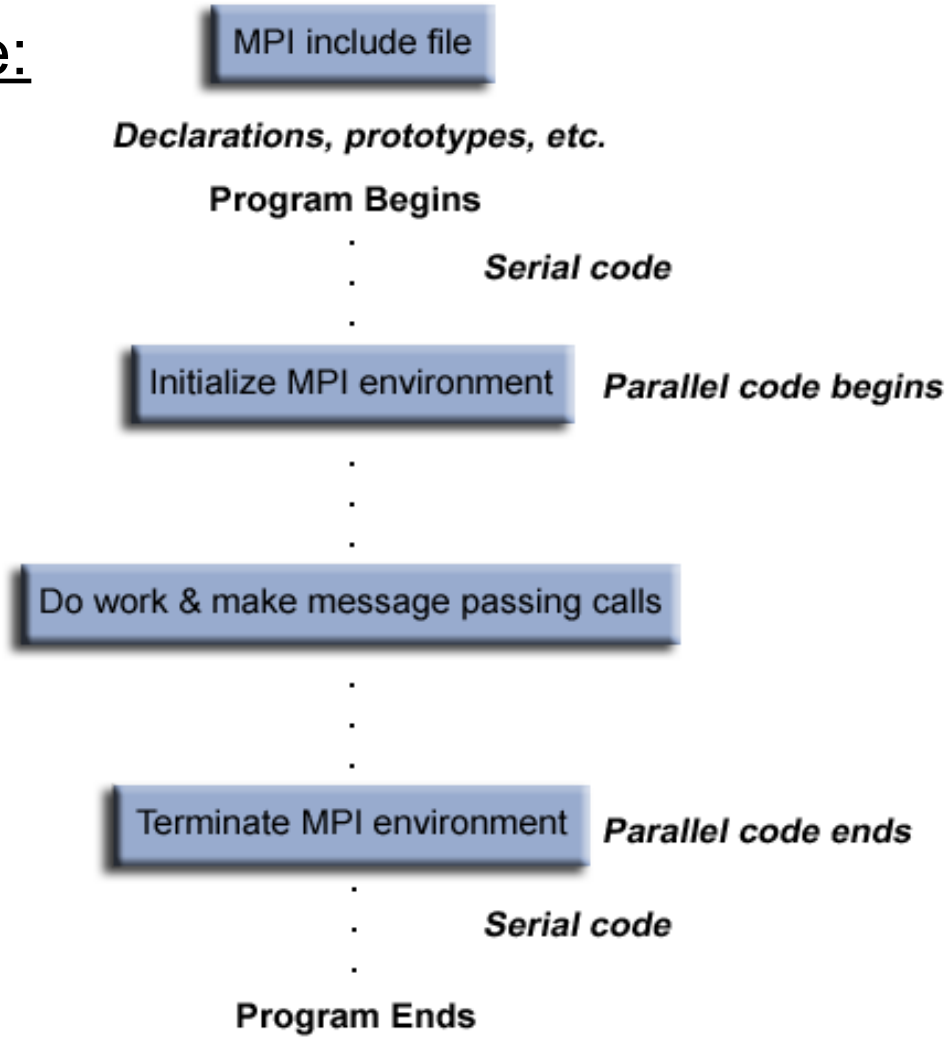
- MPI is an Interface
  - MPI = Message Passing Interface
- Different implementations are available for C / Fortran

C Binding	
Format:	`rc = MPI_Xxxxx(parameter, ... )`
Example:	`rc = MPI_Bsend(&buf,count,type,dest,tag,comm)`
Error code:	Returned as "rc". MPI_SUCCESS if successful
Fortran Binding	
Format:	`CALL MPI_XXXXX(parameter,..., ierr)` `call mpi_xxxxx(parameter,..., ierr)`
Example:	`CALL MPI_BSEND(buf,count,type,dest,tag,comm,ierr)`
Error code:	Returned as "ierr" parameter. MPI_SUCCESS if successful

Source: https://computing.llnl.gov/tutorials/mpi/

2.3 - MPI: Step-by-step Examples

MPI Program Structure:



Source: https://computing.llnl.gov/tutorials/mpi/

2.3 (a) - MPI: Hello World

```c
#include "mpi.h"

#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[]) {
  int numtasks, taskid, len;
  char hostname[MPI_MAX_PROCESSOR_NAME];

  MPI_Init(&argc, &argv);

  MPI_Comm_size(MPI_COMM_WORLD, &numtasks);

  MPI_Comm_rank(MPI_COMM_WORLD, &taskid);

  MPI_Get_processor_name(hostname, &len);

  printf ("Hello from task %d on %s!\n", taskid, hostname);

  if (taskid == 0)
    printf("MASTER: Number of MPI tasks is: %d\n",numtasks);

  MPI_Finalize();

  return 0;
}
```

2.3 (a) - MPI: Hello World

- Compile
  - $> mpicc helloworld-mpi.c -o helloworld-mpi
  - mpicc provide includes directories and libraries paths

2.3 (a) - MPI: Hello World

- Run
  - On one node:
    - mpirun -n $NB_PROCCESS ./helloworld-mpi
  - On a cluster with qsub (Sun Grid Engine)
    - qsub -pe mpich $NB_PROCESS mpi-qsub.sh
    - mpi-qsub.sh:

```
#!/bin/bash
#
#$ -cwd
#
mpirun -np $NSLOTS ./matmul-mpi
```

2.3 (b) - MPI: Matrix Multiply

```
#define NRA N                   /* number of rows in matrix A */
#define NCA P                   /* number of columns in matrix A */
#define NCB M                    /* number of columns in matrix B */
#define MASTER 0                 /* taskid of first task */
#define FROM_MASTER 1            /* setting a message type */
#define FROM_WORKER 2            /* setting a message type */


int     numtasks,               /* number of tasks in partition */
        taskid,                 /* a task identifier */
        numworkers,             /* number of worker tasks */
        source,                 /* task id of message source */
        dest,                   /* task id of message destination */
        mtype,                  /* message type */
        rows,                   /* rows of matrix A sent to each worker */
        averow, extra, offset, /* used to determine rows sent to each worker */
        i, j, k, rc;            /* misc */
double a[NRA][NCA],             /* matrix A to be multiplied */
       b[NCA][NCB],             /* matrix B to be multiplied */
       c[NRA][NCB];             /* result matrix C */
MPI_Status status;
```

2.3 (b) - MPI: Matrix Multiply

MPI initialization:

```
MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD,&taskid);
MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
if (numtasks < 2 ) {
 printf("Need at least two MPI tasks. Quitting...\n");
 MPI_Abort(MPI_COMM_WORLD, rc);
 exit(1);
 }
numworkers = numtasks-1;
```

2.3 (b) - MPI: Matrix Multiply

Master initialization:

```
if (taskid == MASTER)
{
   printf("mpi_mm has started with %d tasks.\n",numtasks);
   printf("Initializing arrays...\n");
   for (i=0; i<NRA; i++)
      for (j=0; j<NCA; j++)
         a[i][j]= i+j;
   for (i=0; i<NCA; i++)
      for (j=0; j<NCB; j++)
         b[i][j]= i*j;
```

2.3 (b) - MPI: Matrix Multiply

```
/* Send matrix data to the worker tasks */
averow = NRA/numworkers;
extra = NRA%numworkers;
offset = 0;
mtype = FROM_MASTER;
for (dest=1; dest<=numworkers; dest++)
{
   rows = (dest <= extra) ? averow+1 : averow;
   printf("Sending %d rows to task %d offset=%d\n",rows,dest,offset);
   MPI_Send(&offset, 1, MPI_INT, dest, mtype, MPI_COMM_WORLD);
   MPI_Send(&rows, 1, MPI_INT, dest, mtype, MPI_COMM_WORLD);
   MPI_Send(&a[offset][0], rows*NCA, MPI_DOUBLE, dest, mtype,
             MPI_COMM_WORLD);
   MPI_Send(&b, NCA*NCB, MPI_DOUBLE, dest, mtype, MPI_COMM_WORLD);
   offset = offset + rows;
}
```

2.3 (b) - MPI: Matrix Multiply

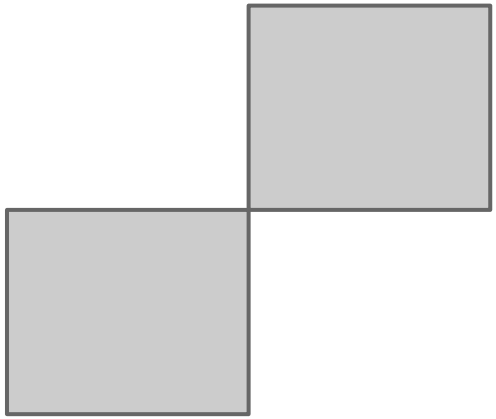
```
/* Receive results from worker tasks */
mtype = FROM_WORKER;
for (i=1; i<=numworkers; i++)
{
   source = i;
   MPI_Recv(&offset, 1, MPI_INT, source, mtype, MPI_COMM_WORLD, &status);
   MPI_Recv(&rows, 1, MPI_INT, source, mtype, MPI_COMM_WORLD, &status);
   MPI_Recv(&c[offset][0], rows*NCB, MPI_DOUBLE, source, mtype,
            MPI_COMM_WORLD, &status);
   printf("Received results from task %d\n",source);
}
```

2.3 (b) - MPI: Matrix Multiply

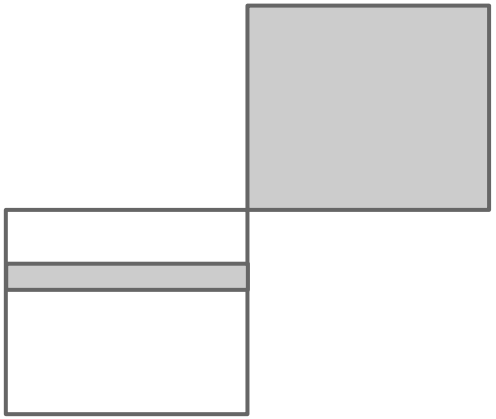
```
if (taskid > MASTER)
{
   mtype = FROM_MASTER;
   MPI_Recv(&offset, 1, MPI_INT, MASTER, mtype, MPI_COMM_WORLD, &status);
   MPI_Recv(&rows, 1, MPI_INT, MASTER, mtype, MPI_COMM_WORLD, &status);
   MPI_Recv(&a, rows*NCA, MPI_DOUBLE, MASTER, mtype, MPI_COMM_WORLD, &status);
   MPI_Recv(&b, NCA*NCB, MPI_DOUBLE, MASTER, mtype, MPI_COMM_WORLD, &status);

   for (k=0; k<NCB; k++)
      for (i=0; i<rows; i++)
      {
         c[i][k] = 0.0;
         for (j=0; j<NCA; j++)
            c[i][k] = c[i][k] + a[i][j] * b[j][k];
      }
   mtype = FROM_WORKER;
   MPI_Send(&offset, 1, MPI_INT, MASTER, mtype, MPI_COMM_WORLD);
   MPI_Send(&rows, 1, MPI_INT, MASTER, mtype, MPI_COMM_WORLD);
   MPI_Send(&c, rows*NCB, MPI_DOUBLE, MASTER, mtype, MPI_COMM_WORLD);
}
MPI_Finalize();
```

2.3 (b) - MPI: Matrix Multiply

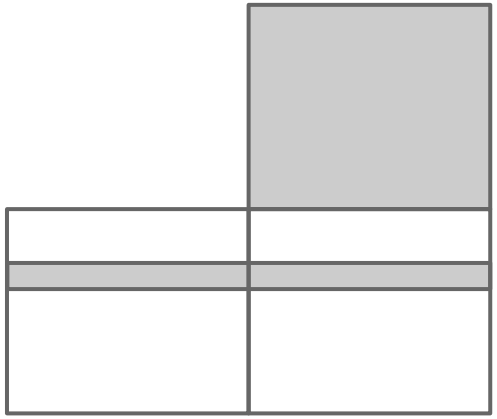
1 - On master after initialization



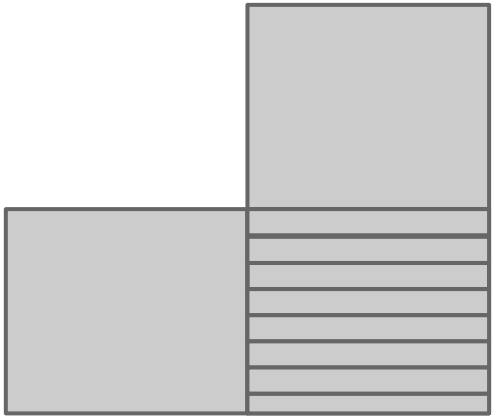
2 - On worker after comm



3 - On worker after computation



4 - On master after comm

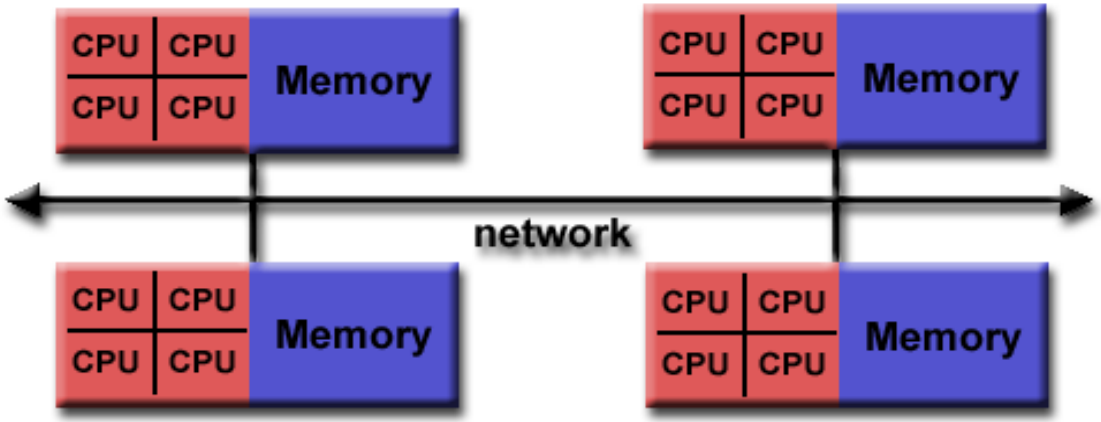


Questions?

Any questions about MPI?

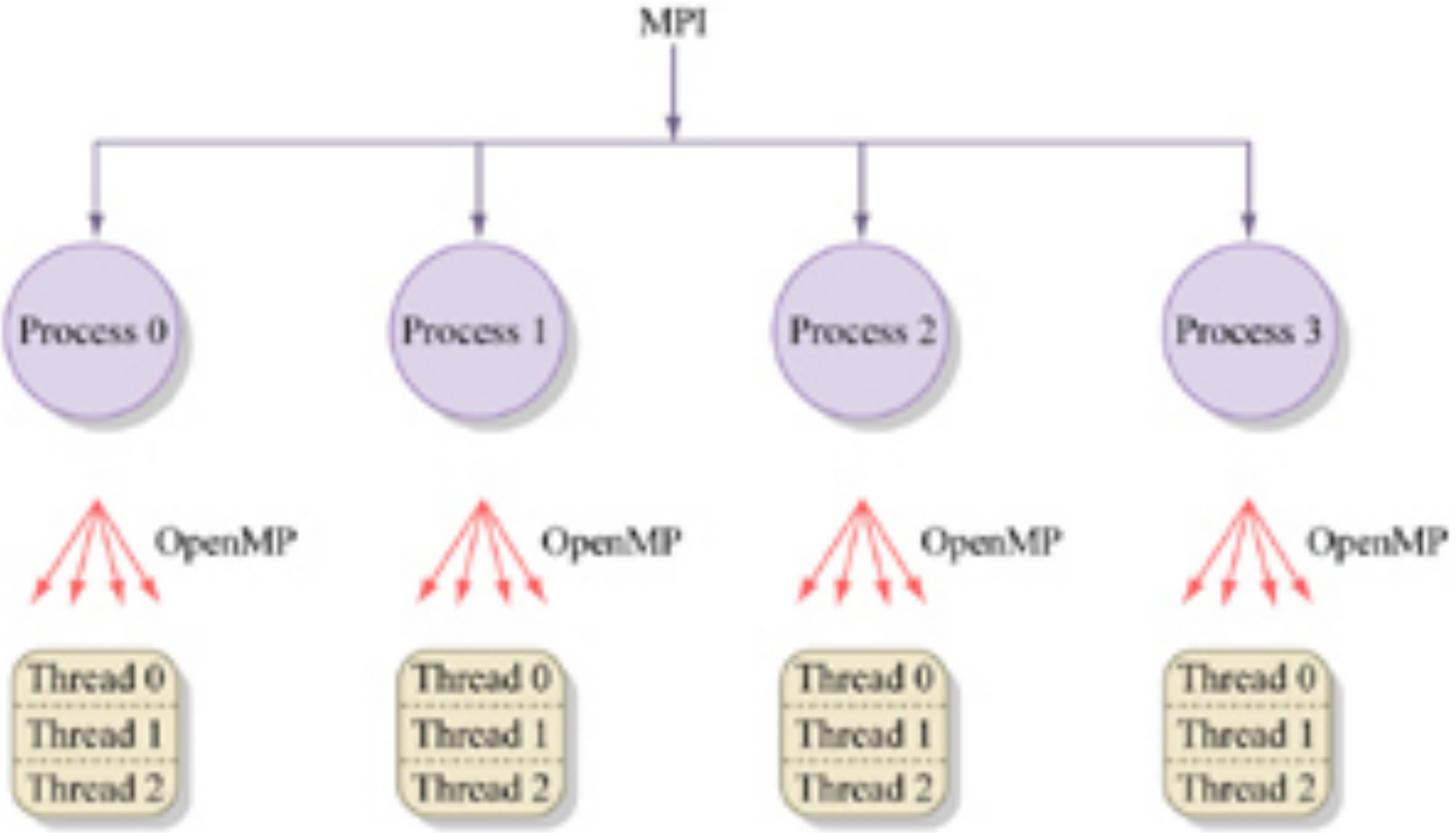
3 - OpenMP & MPI

- MPI : Internodes
- OpenMP : Intranode



- MPI work on SMT processors
  - Message Passing on top of Shared Memory
- Hybrid of OpenMP & MPI:
  - The best of two worlds?

4 - OpenMP & MPI



https://github.com/cavazos-lab/hpc-lecture/blob/master/lecture3/matmul-mpi-omp.c

Questions?

Any questions?