Introduction to parallel computing using MPI and OpenMP on Ada and Terra

June 5, 2017

Outline

- Intro to OpenMP
  - OpenMP basics
  - Compiling
  - Running interactively
  - Running batch
- Intro to MPI
  - MPI basics
  - Compiling
  - Running interctively
  - Running batch
- Hybrid Batch jobs
- Distributing work using OpenMP and MPI

References

HPRC Offers OpenMP and MPI shortcourses every Fall and Spring

- hprc.tamu.edu/wiki/index.php/HPRC:SC:OpenMP
- hprc.tamu.edu/shortcourses/SC-MPI/

What is OpenMP?

API for ***shared memory*** parallel C/C++ , Fortran programs

- Compiler pragmas/directives

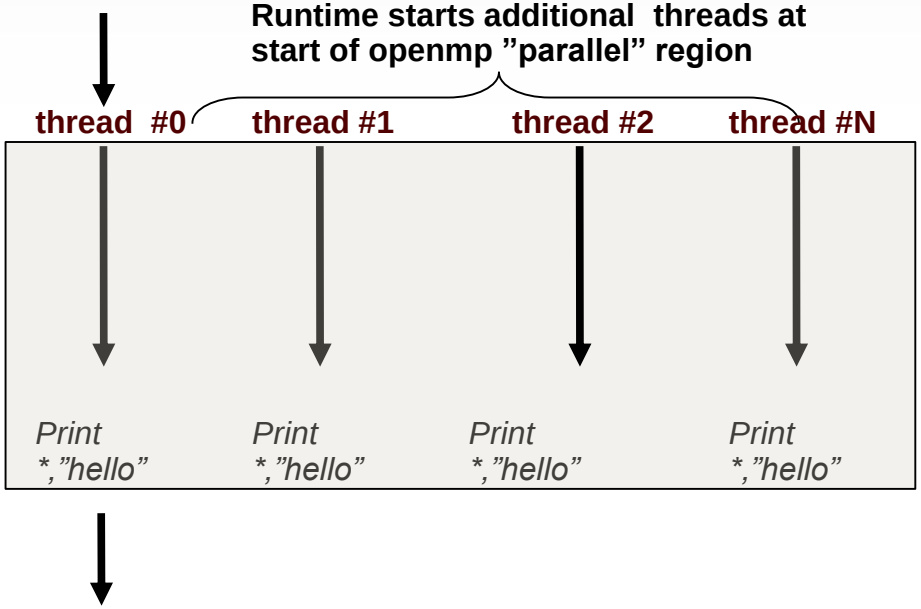
```
#pragma omp directive [clauses]
{
    // block of code
}
```

```
!$OMP DIRECTIVE   [clauses]
    // block of code
!$OMP END DIRECTIVE
```

- Runtime subroutines/functions
- Environment variables

parallel pragma/directive

```
// code before parallel region
            :
#pragma omp parallel
{
   printf( ”Hello\n”);
}
            :
// code after parallel region
```



This is called the FORK/JOIN model

- OpenMP programs start with a single thread; the master thread (Thread #0)
- At start of **parallel** region master starts team of parallel threads (FORK)
- Statements in parallel region are executed concurrently by every thread
- At end of parallel region, all threads synchronize, and join master thread (JOIN)

Implicit barrier.

Useful OpenMP Functions

- Runtime function **omp_get_num_threads()**
  - ✓ Returns number of threads in parallel region
  - ✓ Returns 1 if called outside parallel region
- Runtime function **omp_get_thread_num()**
  - ✓ Returns id of thread in team
  - ✓ Value between [0,n-1] // where n = #threads
  - ✓ Master thread always has id 0
- Runtime function **omp_get_max_threads()**
  - ✓ Returns upper bound #threads in parallel region

OpenMP HelloWorld

Don't forget to Include OpenMP library!

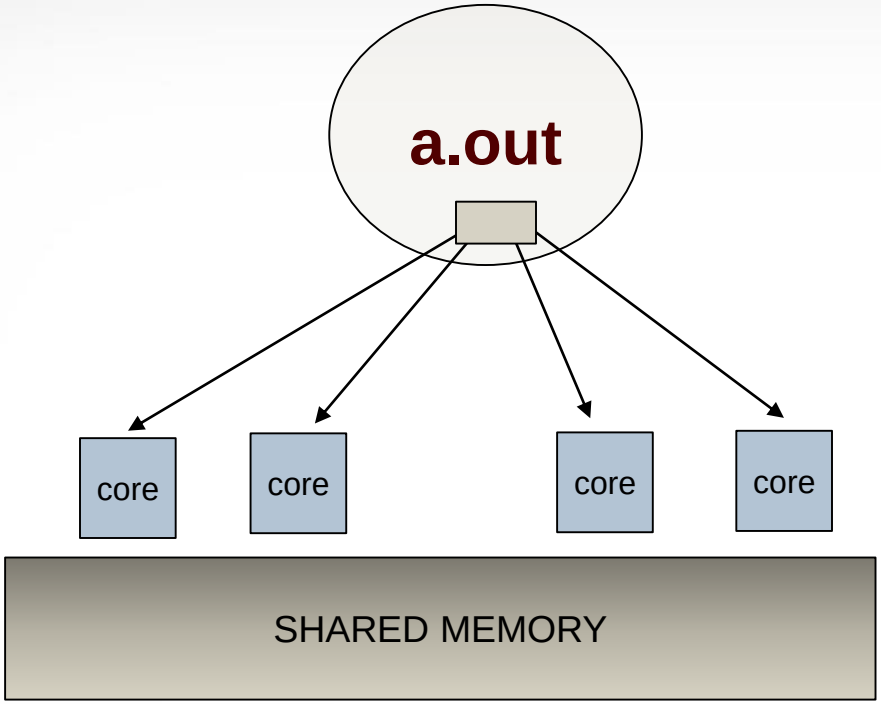
```
#include <iostream>
#include  <omp.h>

int main() {
#pragma omp parallel
  {
     int id = omp_get_thread_num();
     printf("Hello from  thread  %d \n",id);
  }
  return 0;
}
```

OpenMP library function

OpenMP pragma

How is Process mapped to cores?



- Single process
- Utilize multiple threads
- Shared memory

Compiling OpenMP program

Compiling an OpenMP program only requires one additional flag to let the compiler know to process the OpenMP pragmas

Intel (icc/icpc/ifort)	**-qopenmp** (used to be –openmp, deprecated)
pgi (pgcc/pgcpp/pgfortran)	**-mp**
GNU (gcc/g++,gfortran)	**-fopenp**

Example:

```
module load intel/2017A
icpc –qopenmp –o myomp.x myomp.cpp
```

Running OpenMP program (on login node)

When running OpenMP programs you need to specify #threads you want to use

Example:

```
[netid@cluster ~] export OMP_NUM_THREADS=4
[netid@cluster ~] ./myomp.x
```

max 8 CORES and 1 HOUR CPU TIME per login session. Anyone found violating the processing limits will have their processes killed without warning. Repeated violation of these limits will result in account suspension.

OpenMP batch (ada)

```
#BSUB -J OMP  -o OMP.%J -L /bin/bash
#BSUB -W 3:00 -R "rusage[mem=2500]"  -M 2560

#BSUB -n 4  -R "span[ptile=4]"

#don't forget to load the compiler module
module load intel/2017A

#set number of threads
export OMP_NUM_THREADS=4

#run the program
./myomp.x
```

- #BSUB values for **–n** and **ptile** must match
  - must be 20 or less
- OMP_NUM_THREADS much match **#BSUB -n**

OpenMP batch (terra)

```
##!/bin/bash
#SBATCH --export=NONE  --get-user-env=L
#SBATCH --job-name=OMP --output=Example3Out.%j
#SBATCH --time=1-12:00:00 --mem=4096M
#SBATCH --ntasks=1   --cpus-per-task=4

#don't forget to load the compiler module
module load intel/2017A

#set number of threads
export OMP_NUM_THREADS=4

#run the program
./myomp.x
```

- **–ntasks** =1
- **--cpus-per-task** must match **OMP_NUM_THREADS**
  - Must be 28 or less (fit on single node)

OpenMP batch (terra alternative)

```
##!/bin/bash
#SBATCH --export=NONE  --get-user-env=L
#SBATCH --job-name=OMP --output=Example3Out.%j
#SBATCH --time=1-12:00:00 --mem=4096M
#SBATCH --ntasks=4   --ntasks-per-node=4

#don't forget to load the compiler module
module load intel/2017A

#run the program
./myomp.x
```

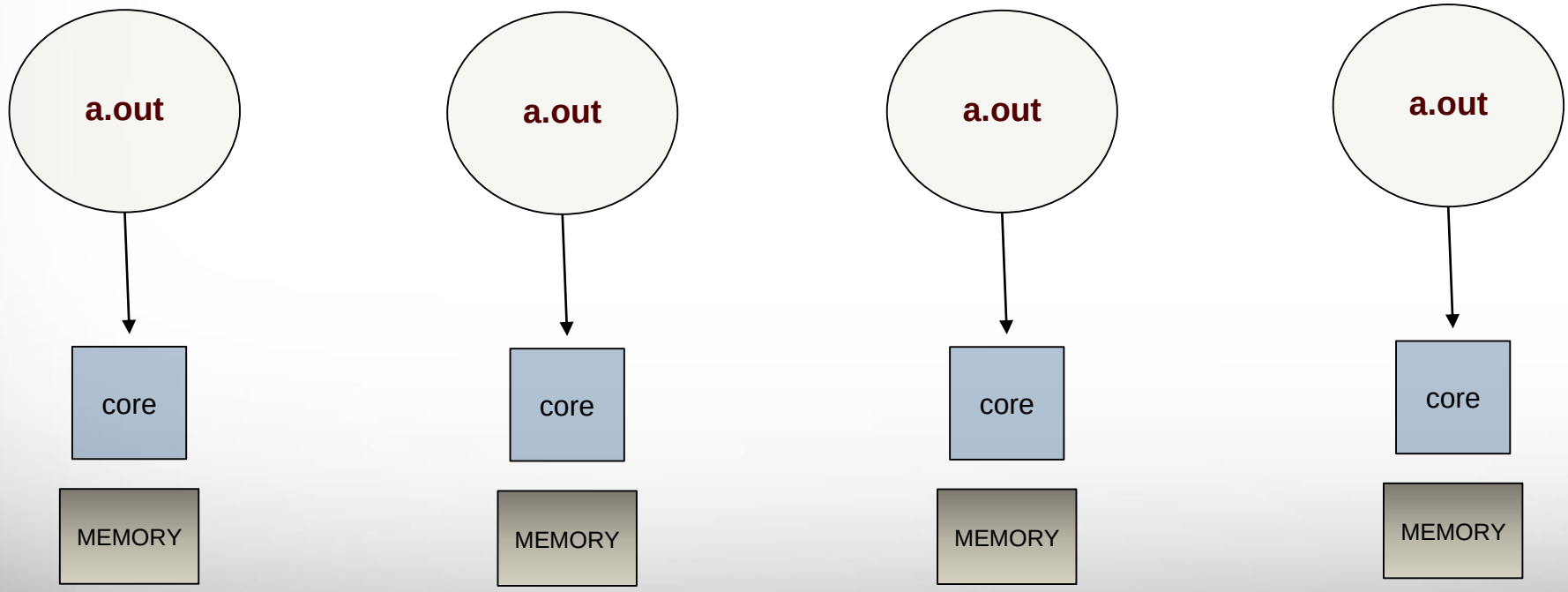
- **–ntasks** and **-ntask-per-node** must match
  - must be 28 or less
- OMP_NUM_THREADS much match **-ntasks**

What is MPI?

- MPI → Message Passing Interface
- Specification to implement message passing
- Multiple Implementations (Intel MPI, OpenMPI,MPICH)

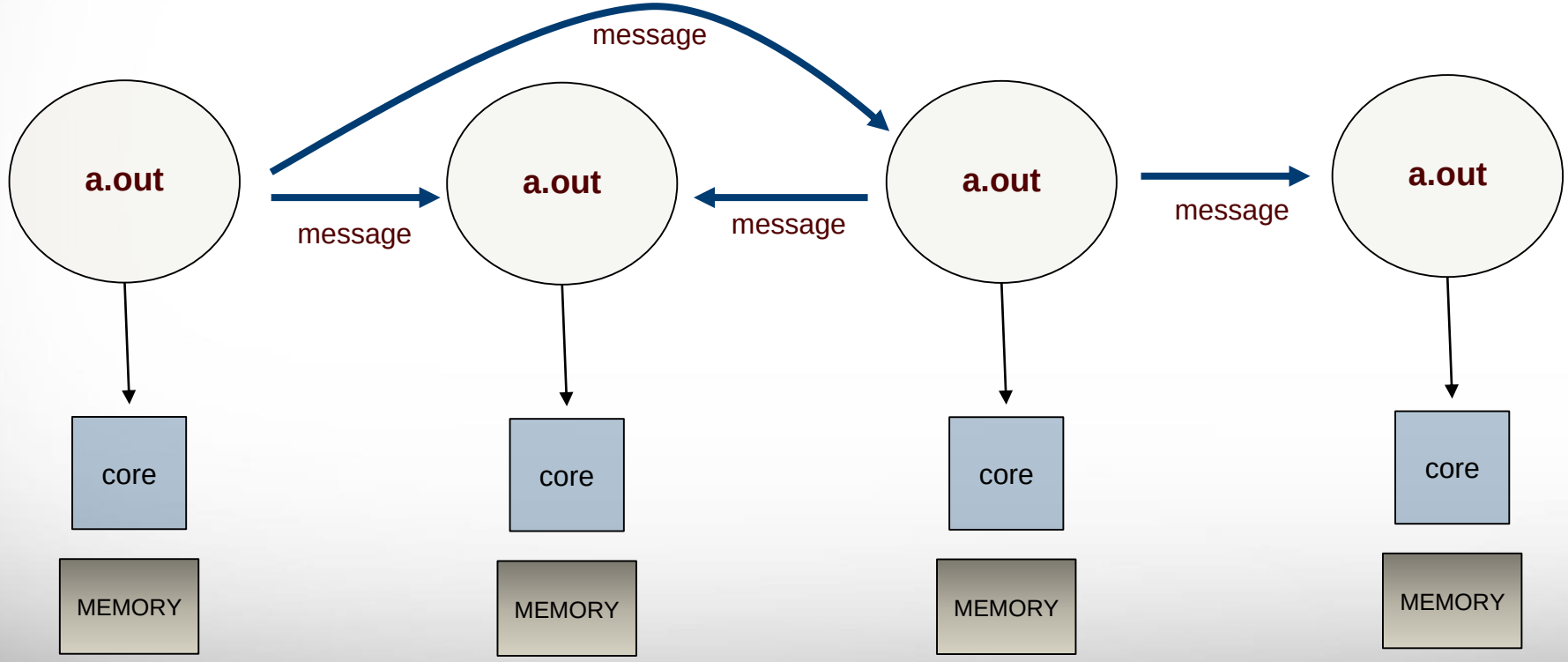
What is MPI?

- MPI → Message Passing Interface
- Specification to implement message passing
- Multiple Implementations (Intel MPI, OpenMPI,MPICH)



What is MPI?

- MPI → Message Passing Interface
- Specification to implement message passing
- Multiple Implementations (Intel MPI, OpenMPI,MPICH)



MPI HelloWorld

Don't forget to Include MPI library!

function to get the rank of the process

```
#include <iostream>
#include  <mpi.h>

int main(int arc, char** argv) {
   MPI_Init(&argc,&argv);
   int rank;
   MPI_Comm_rank(MPI_COMM_WORLD,&rank);
   printf("Hello from  task  %d \n",rank);
   MPI_Finalize();
   return 0;
}
```

Initialize the MPI environment

Stop the MPI environment

Compiling MPI program

We will use Intel MPI wrappers here. Wrappers for other implementations might have different names

wrapper	language
mpiicc (mpicc)	**C**
mpiicpc (mpicxx)	**C++**
mpiifort ((mpifc)	**Fortran**

USE **"<wrapper> -show"** to see full compiler command

Example:

```
module load intel/2017a
mpiicpc –o mympi.x mympi.cpp
mpiicpc -show
```

Running MPI program (on login node)

To run mpi programs use the mpirun launcher

mpirun [mpi options] <prog> [prog args]

(where <prog> is the name of the executable, has to be on the $PATH)

max 8 CORES and 1 HOUR CPU TIME per login session. Anyone found violating the processing limits will have their processes killed without warning. Repeated violation of these limits will result in account suspension.

Running MPI program (on login node)

option	description
-np (-n)	**Defines number of tasks**
-perhost (-ppn)	**Defines how many tasks to start per node (round robin)**
-hosts	**Defines on what hosts to start the tasks**
-h	**Shows all the mpirun options**

max 8 CORES and 1 HOUR CPU TIME per login session. Anyone found violating the processing limits will have their processes killed without warning. Repeated violation of these limits will result in account suspension.

MPI batch (ada)

```
#BSUB -J MPI  -o MPI.%J -L /bin/bash
#BSUB -W 3:00 -R "rusage[mem=2500]"  -M 2560

#BSUB -n 16  -R "span[ptile=4]"

#don't forget to load the compiler module
module load intel/2017A

#run the program
mpirun ./mympi.x
```

- No need to set mpi options (such as **–np)**
  - mpirun will use requirements from batch file
  - some mpirun options will be overridden
- If **mpirun –np** set, must match **#BSUB –np** value

MPI batch (terra)

```
##!/bin/bash
#SBATCH --export=NONE  --get-user-env=L
#SBATCH --job-name=OMP --output=Example3Out.%j
#SBATCH --time=1-12:00:00 --mem=4096M
#SBATCH --ntasks=8   --ntasks-per-node=4

#don't forget to load the compiler module
module load intel/2017A

#run the program
mpirun ./mympi.x
```

- No need to set mpi options (such as **–np)**
  - mpirun will use requirements from batch file
  - some mpirun options will be overridden
- If **mpirun –np** set, must match **#SBATCH –ntasks**

MPI/OpenMP hybrid batch (ada)

```
#BSUB -J MPI  -o MPI.%J -L /bin/bash
#BSUB -W 3:00 -R "rusage[mem=2500]"  -M 2560

#BSUB –n 64  -R "span[ptile=8]"

#don’t forget to load the compiler module
module load intel/2017A

#run the program
export OMP_NUM_THREADS=4
export I_MPI_JOB_RESPECT_PROCESS_PLACEMENT=0
mpirun –np 16 –perhost 2 ./mympi.x
```

- mpirun will override batch scheduler requirements
- No need to use whole nodes
- **-perhost * #threads = ptile**

MPI/OpenMP hybrid batch (ada)

```
#BSUB -J MPI  -o MPI.%J -L /bin/bash
#BSUB -W 3:00 -R "rusage[mem=2500]"  -M 2560

#BSUB -n 16  -R "span[ptile=2]"
#BSUB -x

#don't forget to load the compiler module
module load intel/2017A

#run the program
export OMP_NUM_THREADS=10
mpirun ./mympi.x
```

- Will start total 16 tasks, 2 per node
- Use **#BSUB –x** to reserve whole node to schedule OpenMP threads
- Ideally OMP_NUM_THREADS*ptile=20

MPI/OpenMP batch (terra)

```
##!/bin/bash
#SBATCH --export=NONE  --get-user-env=L
#SBATCH --job-name=OMP --output=Example3Out.%j
#SBATCH --time=1-12:00:00 --mem=4096M
#SBATCH --ntasks=8   --ntasks-per-node=4
#SBATCH –cpus-per-task=4

#don’t forget to load the compiler module
module load intel/2017A

#run the program
Mpirun ./mympi.x
```

- Can use –cpus-per-task to set OMP threads
  - No special tricks needed

Distributing Work (case study)

Compute SUM of all elements in an array

Array A

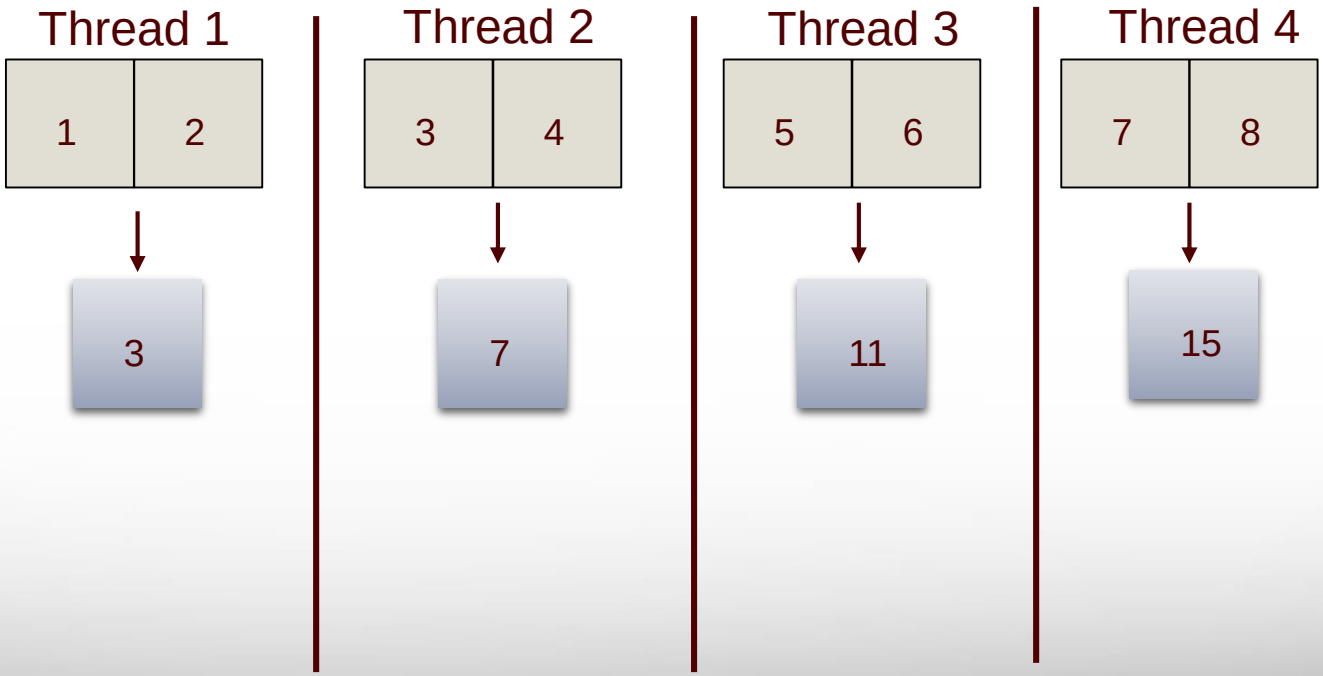
Distributing Work (case study)

Compute SUM of all elements in an array

Array A

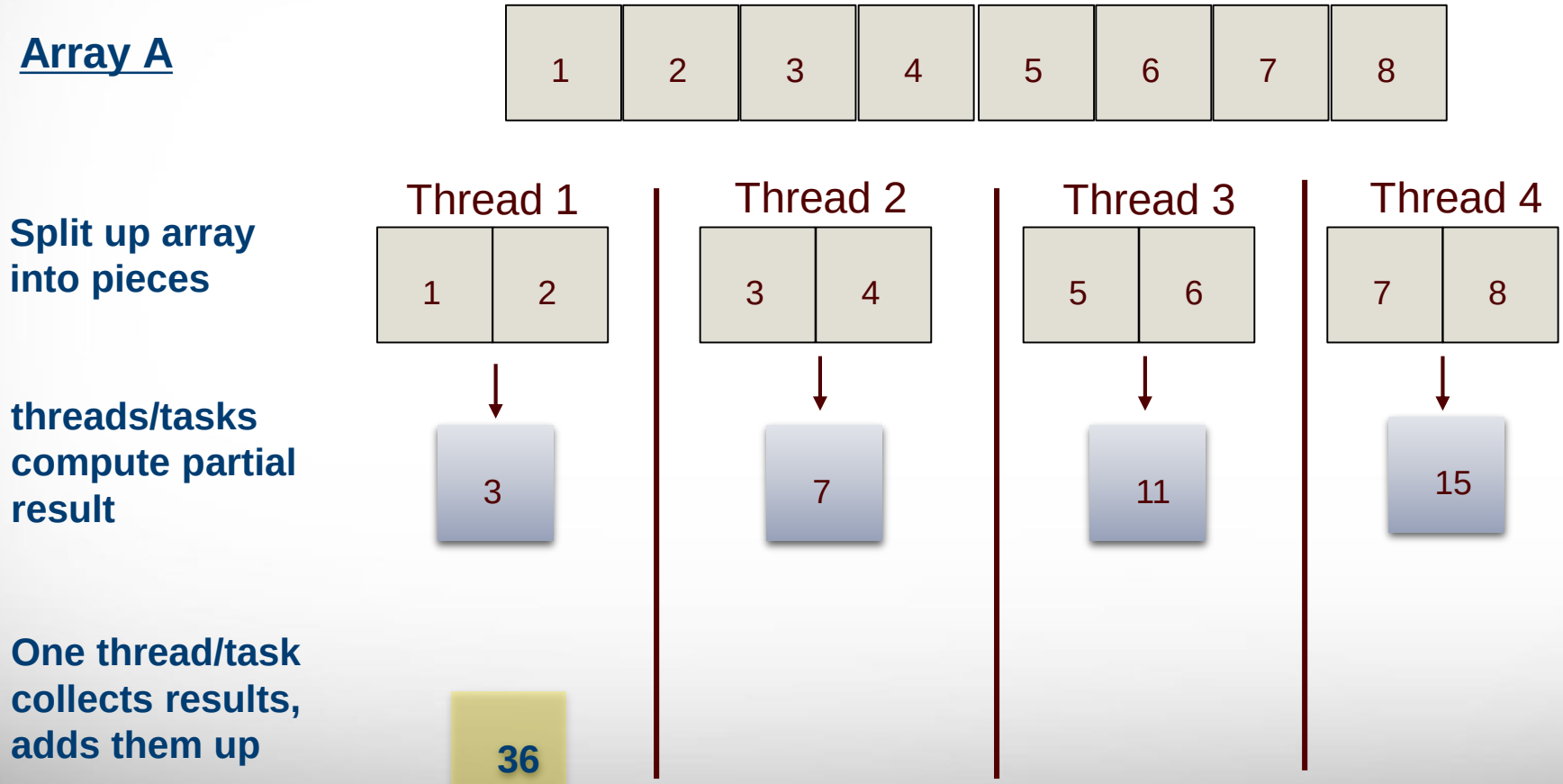
Split up array into pieces

threads/tasks compute partial result



Distributing Work (case study)

Compute SUM of all elements in an array



OpenMP **for** pragma

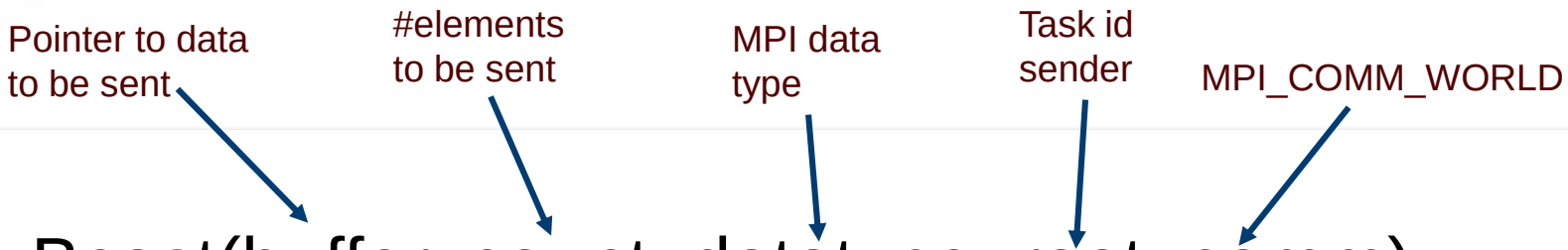
OpenMP **"for"** pragma distributes the iterations of a loop over all the threads, each thread iterates over a subset of loop.

```
#include  <omp.h>

int sum(int* elems, int size) {
   int tot, sums[omp_get_max_threads()];
#pragma omp parallel
{
    tot=omp_get_num_threads();
    int id=omp_get_thread_num();
    sums[id]=0;
#pragma omp for
    for (int i=0; i<size; ++i)
      sums[id]=sums[id]+elems[id];
 }
for (int i=0;i<tot;++i) res = res + sums[i];
}
```

MPI communication

NO SHARED memory, data has to be sent/received



- MPI_Bcast(buffer, count, datatype, root, comm)
  - One process ('root') sends data, all others receive the data
- MPI_SEND(buf, count, datatype, dest, tag, comm)
  - Point to point communication
  - Bloking send (expects a matching blocking receive)
- MPI_Recv(buf, count, datatype, src, tag, comm,stat)
  - Point to point communication
  - Bloking receive (expects a matching blocking send)

MPI communication

1. Task 0 broadcasts array
2. tasks compute partial sums
3. Task receives psum from right neighbor, updates sends to left neighbor
4. Task 0 will compute final sum

```
int sum(int* elems, int total, int task_id, int num_tasks) {
   // task 0 broadcasts #elems and elems
  MPI_Bcast(&total,1,MPI_INT,1,...);
  MPI_Bcast(elems,total,MPI_INT,1, ...);
  psum=0;
  for (int i=0;i<counter;++i) psum=psum+elems[i];
  if (task_id == num_tasks-1) {
     MPI_Bsend(&psum1,MPI_INT,(task_id-1),...);
  } else if (task_id > 0) {
     MPI_Recv(&pt,1,MPI_INT,(task_id+1), ...);
     psum=psum+pt;
     MPI_Bsend(&psum1,MPI_INT,(task_id-1),...);
   } else {
      MPI_Recv(&pt,1,MPI_INT,(task_id+1),...);
   }
   return psum+pt;
}
```

QUESTIONS?