Parallel Scientific Computation

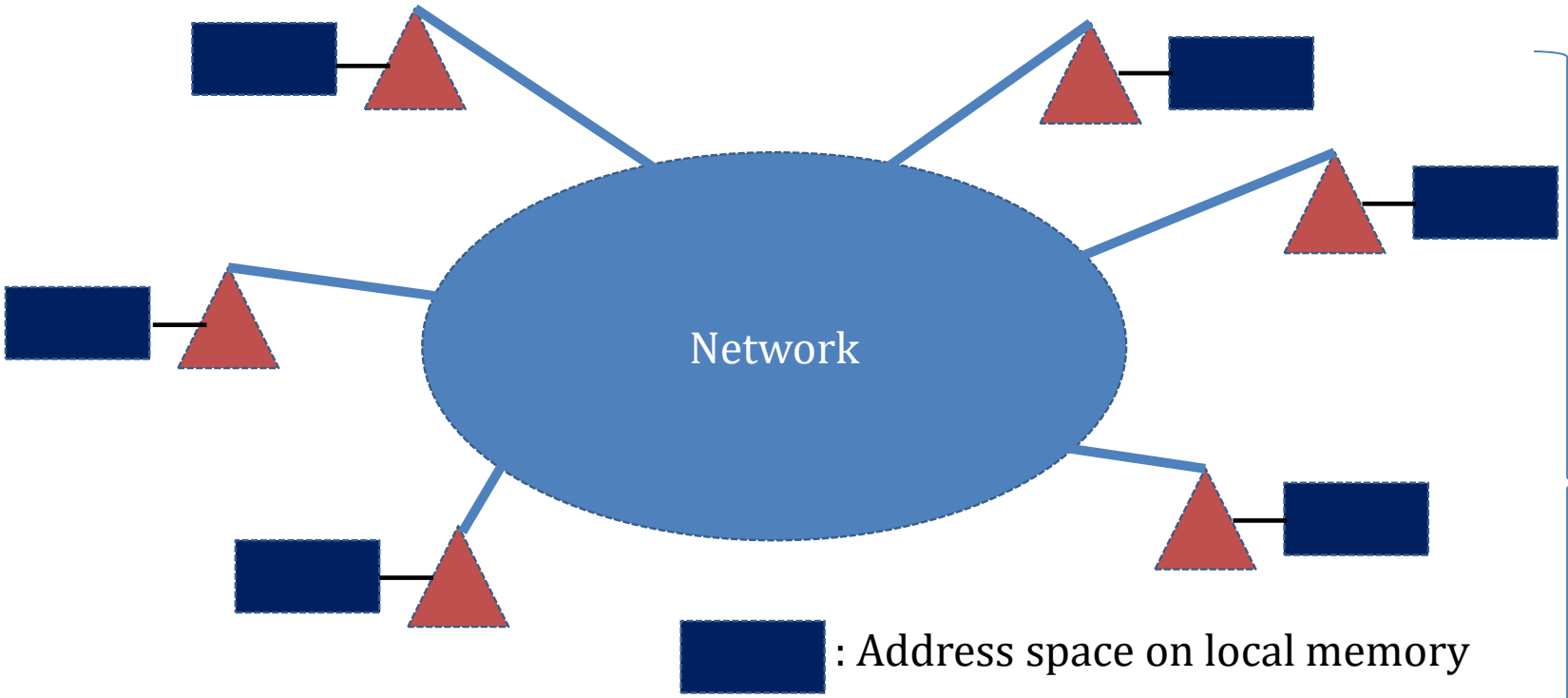
MPI 1

J.-H. Parq
IPCST
Seoul National University

What is MPI?

- Message Passing Interface
  - Communication standard protocol for parallel computing architectures
  - Important in NUMA or distributed memory systems
  - Supported languages: Fortran, C, C++
    - But there are libraries for Java, Python, R, MATLAB, Julia, OCaml and so on.
  - Famous MPI packages: MPICH, Open MPI

Message Passing Model



- *Independent* **processes** with *local* memory
- **Communication** by sending and receiving messages

Basic MPI Concepts

- Communication
  - Send & Receive
    - Transferring a message (= data + additional info.)
    - Copying a portion of data on a process's address space into another process's address space
  - Matching (a message to a receiver)
    - Discriminating messages by a **tag**
  - Point-to-point communication: one-to-one
    - Blocking or Non-blocking

Basic MPI Concepts

- Communication
  - Collective communication
    - Collective operation
    - One-to-all or All-to-one or All-to-all
    - Data movement with or without collective computation
  - Communication modes
    - Standard or Synchronous or Ready or Buffered

Basic MPI Concepts

- More on Messages
  - Buffers
    - described by **(address, count, datatype)** or **(address, maxcount, datatype)**
    - because messages can be *not contiguous* and transferred in a *heterogeneous computing* system
  - Contexts
    - *Invisible* "secondary" or "hyper" **tags**
    - Allocated at run time by the system in control of users or libraries
    - Wild-card matching is done under a context.

Basic MPI Concepts

- Naming processes
  - Group (of processes)
    - Initial group: all processes given by initial setting
  - Rank: process number in a group
- Virtual topologies
  - Graphs and grids of processes
- Communicator
  - An object binding a group of processes and context information implying a topology

Other MPI Features

- Debugging and profiling
  - MPI provides commands for error handling.
- Support for libraries
  - MPI provides tools to create parallel libraries.
- Derived datatypes
  - All datatypes in MPI are defined by libraries and users. (A datatype in MPI can have different structures in different systems.)
  - This allows datatype conversion for *heterogeneous networks*.

Comparison to OpenMP

MPI

- Distributed memory
- (MPI) Processes
- Explicit communication
- Less dependent on architecture
- Datatypes are redefined for MPI.

OpenMP

- Shared memory
- Threads
  - Threads can be constituents of a process.
- Implicit communication
- Limited by architecture
- Datatypes are as they are in the language.

Pros & Cons

Advantage

- Universality
  - Applicable to any architecture
- Perfect for embarrassingly parallel tasks
- Negligible overhead for task start and termination
- Easy debugging
- Relatively good scalability

Disadvantage

- MPI requires more work of programmers.
- Relatively large overhead on communication
- Memory cost increases as process number increases.

Musts in Your MPI Program

- **USE_MPI / include “mpif.h” / include “mpi.h”**
- **MPI_INIT(…) / MPI_Init(…)**
- **MPI_COMM_WORLD**
- **MPI_COMM_SIZE(…) / MPI_Comm_size(…)**
- **MPI_COMM_RANK(…) / MPI_Comm_rank(…)**
- **MPI_FINALIZE(…) / MPI_Finalize()**

Simple Fortran Example

```
PROGRAM
USE MPI
INTEGER ierror, rank, size
CALL MPI_INIT(ierror)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierror)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierror)
IF (rank .EQ. 0) THEN
   PRINT *,  'Hello, world!  :from' , size, 'processes'
ELSE
   PRINT *, 'I am process', rank
END IF
CALL MPI_FINALIZE(ierror)
END
```

INCLUDE "mpif.h" if mpi.mod isn't available

Simple C Example

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

main(int argc, char **argv)
{
  int rank, size;
  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  if (rank == 0) printf("Hello, world!  :from %d processes\n",size);
  else printf("I am process %d\n", rank);
  MPI_Finalize();
}
```

Simple C++ Example 1

```
#include <iostream>
#include <mpi.h>
using namespace std;

main(int argc, char **argv) {
  int rank, size;
  MPI::Init(argc,argv);
  size = MPI::COMM_WORLD.Get_size();
  rank = MPI::COMM_WORLD.Get_rank();
  if (rank == 0) {
    cout << "Hello, world!  :from " << size << " processes" << endl; }
  else cout << "I am process " << rank << endl;
  MPI::Finalize();
}
```

Simple C++ Example 2

```
#include <iostream>
#include <mpi.h>
using namespace std;

main(int argc, char **argv) {
  int rank, size;
  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  if (rank == 0) {
    cout << "Hello, world!  :from " << size << " processes" << endl; }
  else cout << "I am process " << rank << endl;
  MPI_Finalize();
}
```

About C++

- Some versions of MPI offer C++ bindings, but you can use the **plain C MPI** interface in your *C++ program*.
- Further, most of C++ bindings are deprecated in MPI 2.

Compiling

- C/C++

  mpicc hello.c –o hello

  mpicc hello.cpp –o hello

  mpic++ hello.cpp –o hello

- Fortran

  mpif77 hello.f –o hello

  mpif90 hello.f90 –o hello

  mpifort hello.f –o hello

  mpifort hello.f90 –o hello

Execution

- MPICH

  mpiexec –n 4 ./hello

  – For old MPICH

    1. ‘mpd’ or ‘mpdboot’
    2. ‘mpirun’ or ‘mpiexec’
    3. ‘mpdallexit’

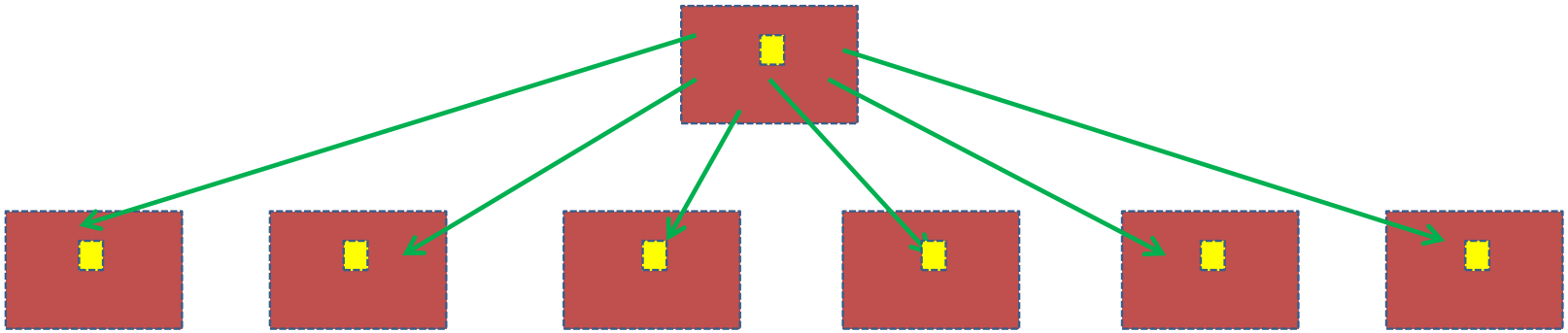
- OpenMPI

  mpiexec –np 4 ./hello

  mpirun –np 4 ./hello

Broadcast

- MPI_BCAST(buffer, count, datatype, root, comm, ierror)
- int MPI_Bcast(void *buf, int count, MPI_Datatype datatype, int root, MPI_Comm comm)

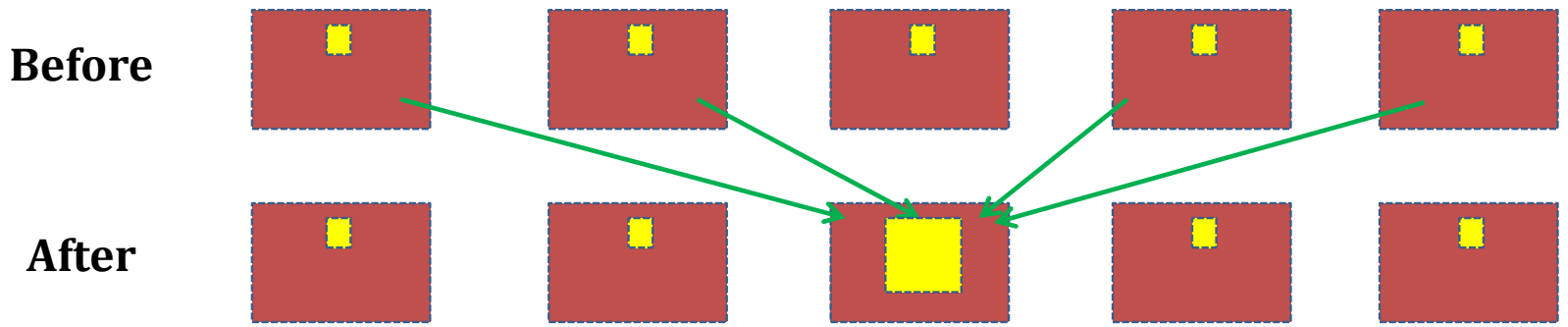


MPI_Datatype

- Fortran
  - MPI_INTERGER
  - MPI_REAL, MPI_DOUBLE_PRECISON
  - MPI_CHARACTER, MPI_LOGICAL, MPI_COMPLEX
  - MPI_BYTE, MPI_PACKED
- C
  - MPI_INT, MPI_LONG, MPI_SHORT, MPI_UNSIGNED_INT, MPI_UNSIGNED_LONG, ……
  - MPI_FLOAT, MPI_DOUBLE, MPI_LONG_DOUBLE
  - MPI_CHAR, MPI_UNSIGNED_CHAR
  - MPI_BYTE, MPI_PACKED

Reduce

- MPI_REDUCE(sendbuf, recvbuf, count, datatype, op, root, comm, ierror)
- int MPI_Reduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op Op, int root, MPI_Comm comm)



Reduce

- MPI_Op
  - MPI_SUM, MPI_PROD
  - MPI_MAX, MPI_MIN
  - MPI_MAXLOC, MPI_MINLOC
    - Max/min with its rank
    - A special datatype is necessary.
  - MPI_LAND, MPI_LOR, MPI_LXOR → logical
  - MPI_BAND, MPI_BOR, MPI_BXOR → bitwise

Defining New Op

- Creation
  - MPI_OP_CREATE(function, commute, op, ierror)
  - int MPI_Op_create(MPI_User_function *function, int commute, MPI_Op *op)
  - function: your function name
  - commute: .TRUE. (1) or .FALSE. (0)
- Annihilation
  - MPI_OP_FREE(op, ierror)
  - int MPI_Op_free(MPI_Op *op)

Defining Op Example (Fortran)

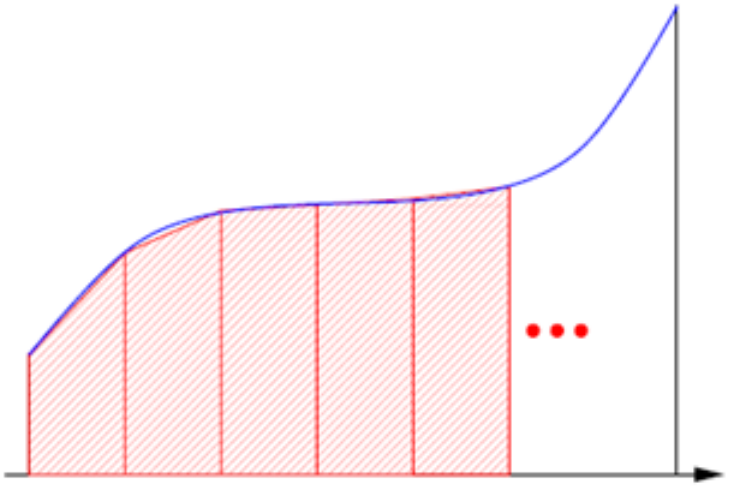
```
……
  External func
  Logical commute
  Integer newop
……
  commute = .true.
  Call MPI_OP_CREATE(func, commute, newop, ier)
……
END PROGRAM
Integer Function func(in, inout, len, type)
……
End Function
```

Defining Op Example (C)

```
void func(void *a, void *b, int *len, MPI_Datatype *dt) {
  int i;
  if (*dt == MPI_INT)
    for (i = 0; i < *len; i++) ((int*)b)[i] = 2*((int*)b)[i] + 2*((int*)a)[i];
……
}
int main() {
……
  MPI_Op newop;
……
  MPI_Op_create(func, 1, &newop)
……
}
```

Numerical Integration Revisited

- Trapezoid rule



$$\frac{\Delta x}{2}[f_0 + 2f_1 + 2f_2 + \cdots + 2f_{N-1} + f_N]$$

– Ex.)

$$\int_0^1 \frac{4}{1+x^2}\,dx = \pi$$

Numerical Integration (Fortran)

```
CALL MPI_INIT(ierror)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, numprocs, ierror)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myid, ierror)
IF (myid .EQ. 0) THEN
   PRINT *,  'Enter the number of interval:'
   READ(*,*) N
END IF
CALL MPI_BCAST(N, 1, MPI_INTERGER, 0, MPI_COMM_WORLD, ierror)
dx = 1d0/dble(N)
S = 0d0
DO I = 1+myid, N-1, numprocs
    x = dble(I)*dx
```

Numerical Integration (Fortran)

```
   S = S + 4d0/(1d0 + x*x)
END DO
CALL MPI_REDUCE(S, total, 1, MPI_DOUBLE_PRECISION, MPI_SUM, 0, &
     MPI_COMM_WORLD, ierror)
IF (myid .EQ. 0) THEN
   total = total + (4d0 + 2d0)/2d0
   ANSWER = total*dx
   PRINT *,  'Answer = ' , ANSWER
END IF
CALL MPI_FINALIZE(ierror)
```

Numerical Integration (C)

```
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
MPI_Comm_rank(MPI_COMM_WORLD, &myid);
if (myid == 0) {
   printf("Enter the number of interval: ");
   scanf("%d", &n);
}
MPI_Bcast(&n, 1, MPI_INT,  0, MPI_COMM_WORLD);
dx = 1.0/(double)n;
s = 0.0;
for (i=myid+1; i<n; i+=numprocs) {
    x = (double)i*dx;
```

Numerical Integration (C)

```
    s += 4.0/(1.0 + x*x);
}
MPI_Reduce(&s, &total, 1, MPI_DOUBLE, MPI_SUM, 0,
    MPI_COMM_WORLD);
if (myid == 0) {
    total += (4.0 + 2.0)/2.0;
    answer = total*dx;
    printf("Answer = %lf\n", answer);
}
MPI_Finalize();
```

Error Handling

- All Fortran subroutines have *ierror* variables.
  - The value of ierror may match with an error class.
- In case of C, for example,

```
int ier;
ier = MPI_Bcast(&n, 1, MPI_INT,  0, MPI_COMM_WORLD);
if (ier == MPI_ERR_INTERN) {
   ……
   MPI_Abort(MPI_COMM_WORLD, ier);    }
```

- You can quit your program without correcting errors by MPI_Abort(comm, errorcode, ierror)

Error Handling

- Error class examples
  - MPI_SUCCESS: no error
  - MPI_ERR_RANK
  - MPI_ERR_BUFFER, MPI_ERR_COUNT, MPI_ERR_TYPE, MPI_ERR_COMM
  - MPI_ERR_ROOT, MPI_ERR_OP
  - MPI_ERR_TAG, MPI_ERR_REQUEST
  - MPI_ERR_ARG
  - MPI_ERR_IN_STATUS: error in a status array
  - MPI_ERR_UNKNOWN, MPI_ERR_OTHER
  - MPI_ERR_PENDING: incomplete operation (not error)

References

- W. Gropp, E. Lusk, and A. Skjellum,
  Using MPI

- C. Evangelinos,
  Parallel Programming for Multicore Machines Using OpenMP and MPI

- Wikipedia