Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

NUMA with **Open MPI**

Carolin Fiedler

14 January 2015

Agenda

1. MPI and Open MPI
2. MPI Concepts
3. NUMA with (Open)MPI
4. Examples

What is MPI?

A message-passing library

interface specification

[**M**essage-**P**assing **I**nterface]

Goals of MPI

develop a widely used standard for writing message-passing programs

portability and ease of use

efficient communication | can be used in a **heterogeneous environment**

a reliable communication interface | **Thread safety**

C and Fortran bindings for the interface

Evolution of MPI

MPI-1.0 (1994)

- **MPI-1.1** (1995)
- **MPI-1.2** (1997)
- **MPI-1.3** (2008)

MPI-2.0 (1997)

- **MPI-2.1** (2008)
- **MPI-2.2** (2009)

MPI-3.0 (2012)

- ***MPI-3.1*** *(2015)*

MPI-4.0 *(20??)*

Open MPI

Hasso Plattner Institut

… is an open source* MPI implementation

FT-MPI [University of Tennessee]

LA-MPI [Los Alamos National Laboratory]

LAM/MPI [Indiana University]

with contributions from the PACX-MPI team [University of Stuttgart]

[* Open source license based on the BSD license]

Open MPI

... supports / implements:

Full MPI-3 standard conformance

Thread safety and concurrency | Dynamic process spawning

Support network heterogeneity | **Network and process fault tolerance**

High performance on all platforms | Portable and maintainable

Component-based design, documented APIs

Agenda

1. MPI and Open MPI
2. MPI Concepts
3. NUMA with (Open)MPI
4. Examples

Concepts

autonomous processes

executing their own code (MIMD style)

processes communicate via

calls to **MPI communication primitives**

each process executed in its **own address space**

(shared-memory implementations of MPI are possible)

Concepts: group, context, communicator

group

- ordered set of process identifiers
- process associated with **rank [0, n-1]**

context

- communicationg space

communicator

- provides **independent process addressing**
- MPI_COMM_WORLD and MPI_COMM_SELF

Concepts: point-to-point communication

sending and receiving of messages by processes

blocking and non-blocking versions

communiation modes: **S**tandard, **B**uffered, **S**ynchronous and **R**eady

```
int MPI_Send(
  void *buffer, int count, MPI_Datatype type,
  int destRank,
  int messageTag,
  MPI_Comm communicator)
```

Concepts: point-to-point communication

```
int MPI_Send(
void *buf, int count, MPI_Datatype type, int dest,   int tag, MPI_Comm comm)

int MPI_Recv(
void *buf, int count, MPI_Datatype type, int source, int tag, MPI_Comm comm, MPI_Status *st)
```

non-blocking versions:

```
int MPI_Isend( … , MPI_Request *req)
int MPI_Irecv( … , MPI_Request *req)
```

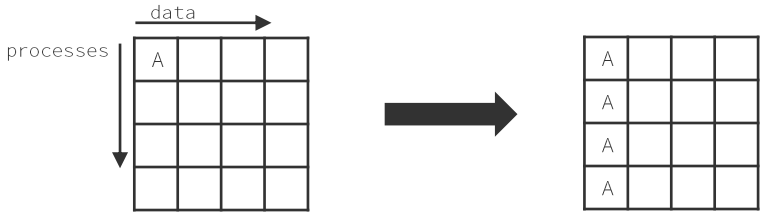
→ MPI_Wait and MPI_Test

Concepts: collective operations

communication that involves a **group of processes**

data movement and **collective computation**

```
int MPI_Bcast(void *buffer, int count, MPI_Datatype type,
              int root, MPI_Comm comm)
```

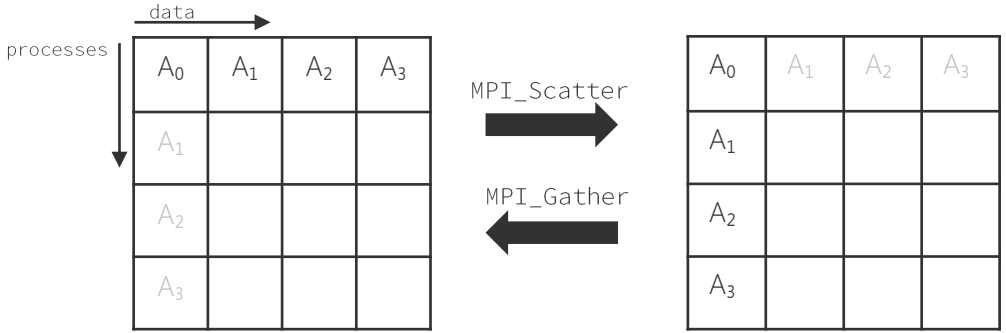


Concepts: collective operations

MPI_Gather / MPI_Scatter

gather data from all members of a group to one member /

scatter data from one member to all members of a group



MPI_Allgather

a variation on Gather where all members of a group receive the result

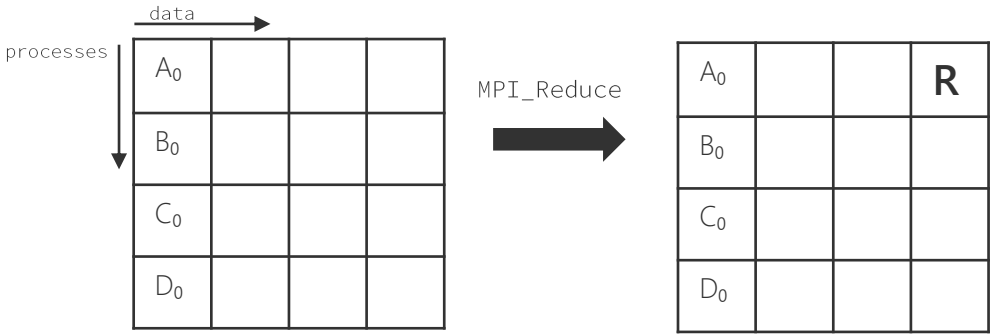
MPI_Alltoall

scatter / gather data from all members to all members of a group (complete exchange)

Concepts: collective operations

MPI_Reduce

Reduces values on all processes within a group



MPI_Op:

MAX & MIN | SUM & PROD

LAND & BAND

LOR & BOR | LXOR & BXOR

MAXLOC & MINLOC

MPI_Allreduce

result is returned to all members of a group

MPI_Reduce_scatter

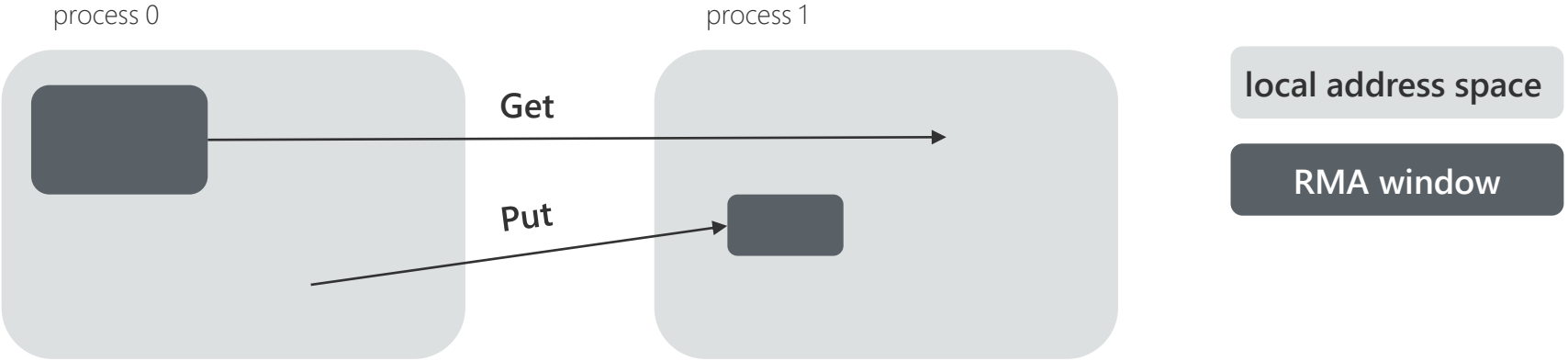
a combined reduction and scatter operation

Concepts: collective operations

```
int MPI_Gather/MPI_Scatter(
const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf,
int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
```

```
int Reduce(
const void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype,
MPI_Op op, int root, MPI_Comm comm)
```

Concepts: one-sided communication




```
MPI_Win_create

→ MPI_Get/MPI_Put & MPI_Win_fence
```

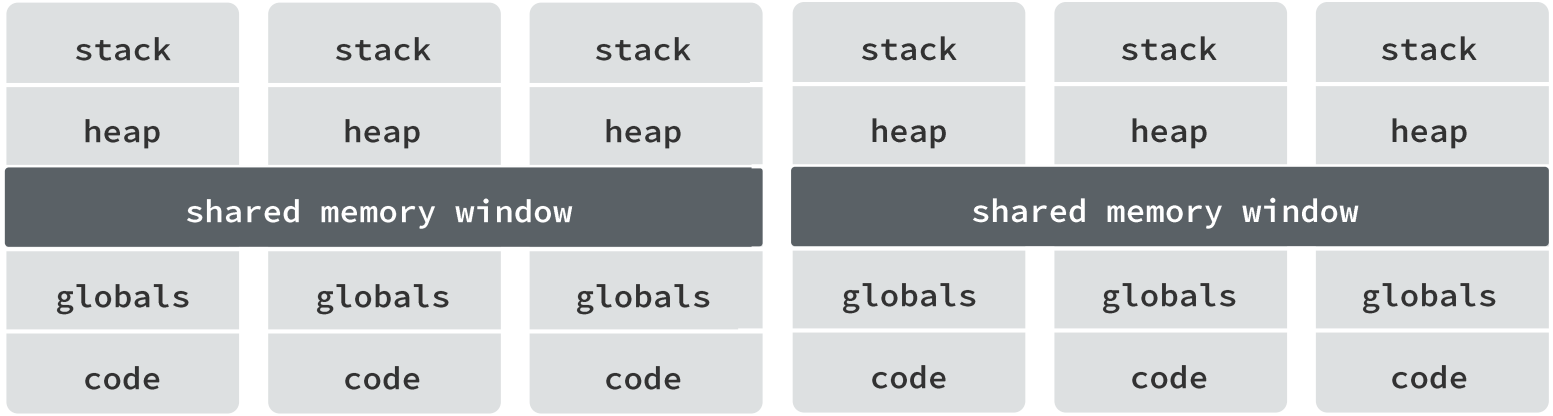
Concepts: one-sided communication

```
MPI_Win_create(
void *base, MPI_Aint size, int disp_unit, MPI_Info info, MPI_Comm comm,
MPI_Win *win)

MPI_Get/MPI_Put(
void *orig_addr, int orig_count, MPI_Datatype orig_type, int targ_rank,
MPI_Aint targ_disp, int targ_count, MPI_Datatype targ_type, MPI_Win win)

int MPI_Win_fence(
int assert, MPI_Win win)
```

Concepts: shared memory



disciplined sharing

```
MPI_Win_allocate_shared

MPI_Win_shared_query

MPI_Comm_split_type
```

Concepts: shared memory

```
int MPI_Win_allocate_shared(
MPI_Aint size, int disp_unit, MPI_Info info, MPI_Comm comm, void *baseptr,
MPI_Win *win)

int MPI_Win_shared_query(
MPI_Win win, int rank, MPI_Aint *size, int *disp_unit, void *baseptr)

int MPI_Comm_split_type(
MPI_Comm comm, int split_type, int key, MPI_Info info, MPI_Comm *newcomm)
```

Agenda

1. MPI and Open MPI
2. MPI Concepts
3. NUMA with (Open)MPI
4. Examples

Run an Open MPI program

compile

mpicc mpiProgram.c

run program

mpirun <options> program

-c, -n, -np <#>	run <#> copies of program on given nodes
-H, -host <h1,…,hN>	list of hosts on which to invoke processes
-hostfile <hostfile>	

Mapping, ranking and binding with Open MPI

mapping

assigns a default location to each process

ranking

assigns an MPI_COMM_WORLD rank value to each process

binding

constrains each process to run on specific processors

Mapping, ranking and binding with Open MPI

Hasso Plattner Institut

mapping

-map-by <foo> — map to the specified object (combine with modifiers)

[slot, hwthread, core, l1cache, l2cache, l3cache, socket, numa, board, none, ...]

-rf <rankfile> — define mapping per rank [MPI_COMM_WORLD]

--map-by ppr:n:node

on each node, launch n processes

--map-by ppr:n:socket

on each node, launch n processes * #sockets

ranking

--rank-by <foo> — rank in round-robin fashion

[slot, hwthread, core, l1cache, l2cache, l3cache, socket, numa, board, none]

binding

--bind-to <foo> — bind processes to the specified object

[slot, hwthread, core, l1cache, l2cache, l3cache, socket, numa, board, none]

-report-bindings — report any bindings for launched processes

Shared memory

MPI implementations use shared memory for

- process communication
- collective operations
- …

Open MPI

- sm btl (shared-memory Byte Transfer Layer)

Shared memory

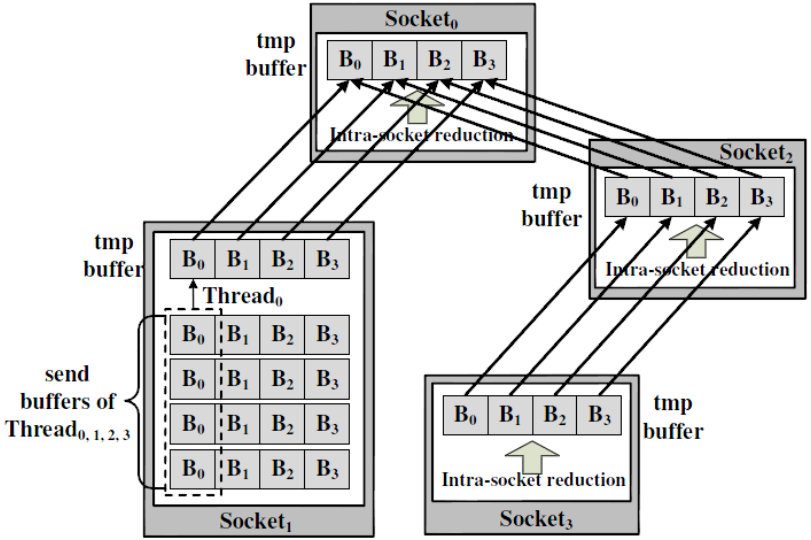


Figure 7: Hierarchical parallel-reduce, for 1 node with 4 sockets; each socket includes 4 cores.

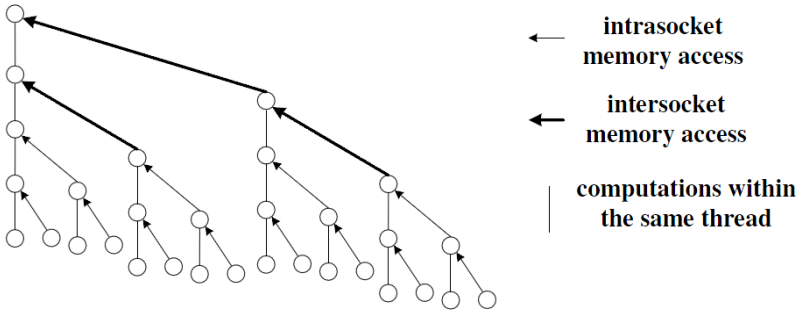


Figure 3: Binary reduction tree, for 1 node with 4 sockets; each socket includes 4 cores.

NUMA with Open MPI

Carolin Fiedler, 14 January 2015

[NUMA-aware shared-memory collective communication for MPI]

Hybrid Programming

combining MPI with different programming models

MPI was designed to encourage hybrid programming!

most common: **MPI + threads**

e.g. PGAS / Pthreads / **OpenMP**

→ **MPI + OpenMP**

Hybrid Programming

```
int MPI_Init_thread(
int *argc, char ***argv,    int required, int *provided)
```

```
MPI_THREAD_SINGLE
   Only one thread will execute.

MPI_THREAD_FUNNELED
   If the process is multithreaded, only the thread that called
   MPI_Init_thread will make MPI calls.

MPI_THREAD_SERIALIZED
   If the process is multithreaded, only one thread will make MPI library
   calls at one time.

MPI_THREAD_MULTIPLE
   If the process is multithreaded, multiple threads may call MPI at once
   with no restrictions.
```

MPI and Hardware

at least 25 % of TOP100 use MPI

→ hardware MPI implementations of MPI /

hardware acceleration for collective operations

… developed **M**PI **O**ffload **E**ngine

SGI UV 2000

[top500.org/lists/2014/11/]

Agenda

1. MPI and Open MPI
2. MPI Concepts
3. NUMA with (Open)MPI
4. Examples

Hello World

Hybrid MPI with OpenMP

… first listen to Matthias' presentation

NUMA with Open MPI

Hasso Plattner Institut

MPI is a widely used standard

hardware implementations of MPI / hardware accelaration for MPI operations

Open MPI uses autodetected topology for optimizations

Sources

William Gropp, Ewing Lusk, and Anthony Skjellum. 2014. Using Mpi: Portable Parallel Programming with the Message-Passing Interface. MIT Press, Cambridge, MA, USA.

William Gropp, Torsten Hoefler, and Rajeev Thakur. 2014. Using Advanced Mpi: Modern Features of the Message-Passing Interface. MIT Press, Cambridge, MA, USA.

MPI-3.0 standard: Message Passing Interface Forum. 2012. MPI: A Message-Passing Interface Standard Version 3.0.

LI, Shigang; HOEFLER, Torsten; SNIR, Marc. NUMA-aware shared-memory collective communication for MPI. In: Proceedings of the 22nd international symposium on High-performance parallel and distributed computing. ACM, 2013. S. 85-96

Open MPI website: https://www.open-mpi.org