EPL372 Lab Exercise 5: (MPI Lecture) Introduction to MPI

References: <u>https://computing.llnl.gov/tutorials/mpi/</u> <u>https://computing.llnl.gov/tutorials/mpi/exercise.html</u> <u>http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf</u>

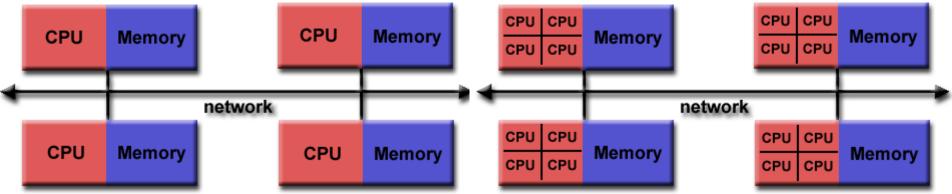
release_x86_irqs(dead_task))/* This gets called before we allocate a new thread and copy " the current usek into it "void prepare_to_copy(struct task_struct "tsk){unlazy_fpu(tsk);}int copy_thread(int nr, unsigned long clone_ftags, unsigned long each unsigned long unused, struct task_struct "p.struct pt_regs " regs} struct pt_regs " childregs; struct task_struct "tsk)(unlazy_fpu(tsk);}int copy_thread(int nr, unsigned long clone_ftags, unsigned long each unsigned long unused, struct task_struct "p.struct pt_regs " regs} struct pt_regs " childregs; struct task_struct "tsk)(unlazy_fpu(tsk);}int copy_thread(int nr, unsigned long clone_ftags, unsigned long each unsigned long unused, struct task_struct "p.struct pt_regs " regs} struct pt_regs " childregs; struct task_struct "tsk)(integs >eax = 0; childregs->esp = esp; p->set_child_tid = p->clear_child_tid = NULL; p->thread.esp = (unsigned long) childregs; p->thread.esp0 = (unsigned long) (childregs+1); p->thread.eip = (unsigned long) ret_from_fork_savesegment(ts p->thread.ts_io_bitmep = kmalloc(IO_BITMAP_BYTES, GFP_KERNEL); if (Ip->thread.ts_io_bitmap) return -ENOMEM; memocpy(p->thread.ts_io_bitmap, tsk->thread.ts_io_bitmap, IO_BITMAP_BYTES);)* Set a new TLS for the child thread? */if (clone_ftags & CLONE_SETTLS) { struct desc_struct "desc; struct user_desc info; int idx; err = -EFAULT; if (copy_from_user(&info, (void __user *)childregs->esi, sizeof(info))) goto out; err = +EINVAL; if (LDT_empty(&info)) goto out; idx = info.entry_number; if (idx < GDT_ENTRY_TLS_MIN || idx > GDT_ENTRY_TLS_MAX) goto out; desc = p->thread.tls_array + idx - GDT_ENTRY_TLS_MIN; desc->a = LDT_entry_a(&info); desc->b = LDT_entry_b(&info); err = 0,1

Why MPI?

It is NOT a library - but rather the specification of what such a library should be.

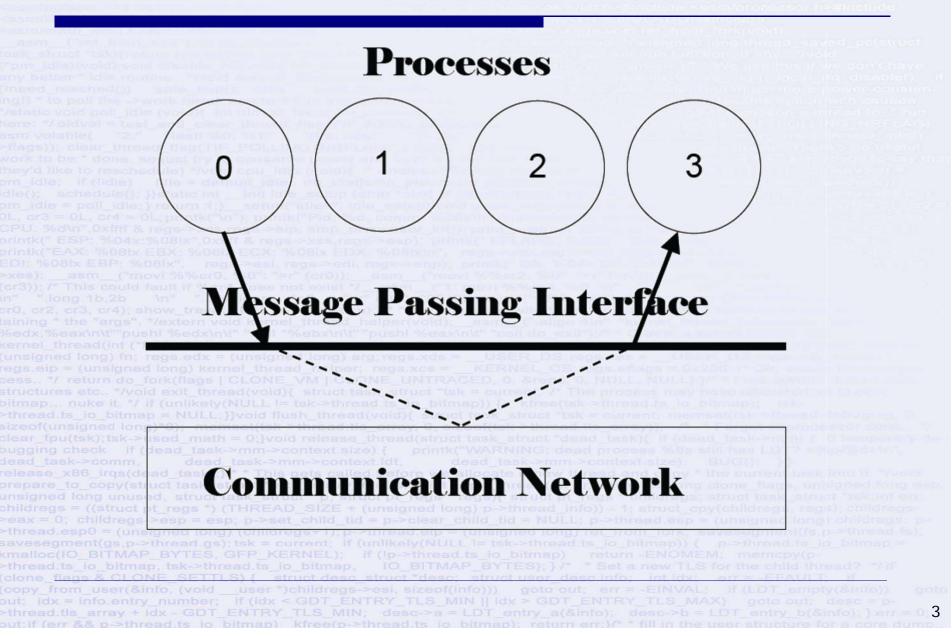
An MPI library is the most important piece of software in parallel programming.

All the world's largest supercomputers are programmed using MPI



childregs = ((struct pt_regs *) (THREAD_SIZE + (unsigned long) p->thread_info)) - 1; struct_cpy(childregs, regs); childregs >eax = 0; childregs->esp = esp; p->set_child_tid = p->clear_child_tid = NULL; p->thread.esp = (unsigned long) childregs; p->thread.esp0 = (unsigned long) (childregs+1); p->thread.eip = (unsigned long) ret_from_fork; sevesegment(fs p->thread.fs); savesegment(gs,p->thread.gs); tsk = current; if (unlikely(NULL != tsk->thread.ts_io_bitmap)) { p->thread.ts_io_bitmap = kmalloc(IO_BITMAP_BYTES, GFP_KERNEL); if (lp->thread.ts_io_bitmap) return -ENOMEM; memcpy(p->thread.ts_io_bitmap, tsk->thread.ts_io_bitmap, IO_BITMAP_BYTES); } / * Set a new TLS for the child thread? */if (clone_flags & CLONE_SETTLS) { struct desc_struct*desc; struct user_descinfo; int idx; err = EFAULT if (copy_from_user(&info, (void _user *)childregs->esi, sizeof(info))) goto out; err = -EINVAL; if (LDT_empt(&info)) goto out; idx = info.entry_number; if (idx < GDT_ENTRY_TLS_MIN || idx > GDT_ENTRY_TLS_MAX) goto out; desc = p->thread.tls_array + idx - GDT_ENTRY_TLS_MIN; desc->a = LDT_entry_a(&info); desc->b = LDT_entry_b(&info);) err = 0.2 out; if (err && p->thread.ts_io_bitmap) kfree(p->thread.ts_io_bitmap); return err;)/* fill in the user structure for a core dump.

Message Passing Programming Paradigm



Message Passing Programming Paradigm

□ All variables are private

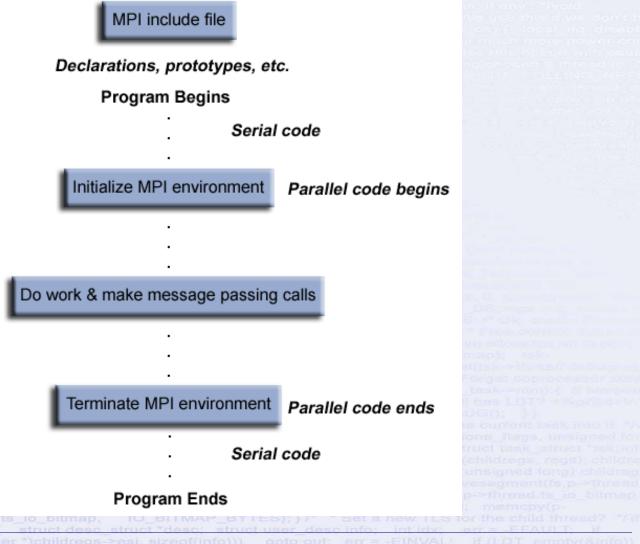
- □ Processes communicate with messages using:
 - Special subroutine calls
- □ Typically:
 - A single program is running on each processor

[ar3); // This Equild fault if %cr4 does not exist */___sem__Itsmoor %kers, so with solution and the set of the set

General MPI Program Structure

mp.h>#include

5



Communicators and Groups

MPI uses objects called **communicators** and **groups** to define which collection of processes may communicate with each other.

Most MPI routines require you to specify a communicator as an argument.

MPI_COMM_WORLD is the predefined communicator that includes all of your MPI processes.

exes): __asm__("movi %%%cr (cr3)); /* This could fault if %c (rd") " long 1b.2b M" " p cr0, cr2, cr3, cr4); show_trac taining " the "args". '/extern w %edx;%eaxintt""push1 %edx) kernel_thread(int (*fn)(void *) (unsigned long) fn; regs.edx regs.eip = (unsigned long) ke cess. */ return do_fork(flags structures etc. "/void exit th bitmap... nuke it. */ if (unlikel >thread.ts_io_bitmap = NULL sizeof(unsigned long)*8); ms clear_fpu(tsk);tsk->used_mat bugging check if (dead_tas dead_task->comm, dea release_x86_irqs(dead_task) orepare_to_copy(struct task_ unsigned long unused, struct childregs = ((struct pt_regs *) >eax = 0; childregs->esp = e >thread.esp0 = (unsigned lor savesegment(gs,p->thread.gs kmalloc(IO_BITMAP_BYTES



MPI_COMM_WORLD

ik->nm) { // isophony de is LOT? = %op%d>is / (); }); ument isek into it %void s_flags, unsigned long dep it lask_atruct 'tskiint en idregs, regs); childregs igned long) childregs; pegment(is,p->thread is); thread.ts_io_bilmep = nemcply(phe child thread? */if = = FFAULT if

(copy_from_user(&info, (void __user *)childregs->esi, sizeof(info))) goto out; err = -EINVAL; if (LDT_empty(&info)) goto out; idx = info.entry_number; if (idx < GDT_ENTRY_TLS_MIN |] idx > GDT_ENTRY_TLS_MAX) goto out; desc = p->thread.tls_array + idx - GDT_ENTRY_TLS_MIN; desc->a = LDT_entry_a(&info); desc->b = LDT_entry_b(&info); } err = 0.6 out; if (err && p->thread.ts_io_bitmap) kfree(p->thread.ts_io_bitmap); return err.)/* fill in the user structure for a core dump

MPI Syntax

□ Header file:

#include <mpi.h>

□ Function Format:

error = MPI_Xxxxx(parameter, ...);

MPI_Xxxxx(parameter, ...);

EDI: %081x EBP: %081x", regs->eal, regs->eal

Rank

Within a communicator, every process has its own unique, integer identifier assigned by the system when the process initializes. A rank is sometimes also called a "task ID". Ranks are contiguous and begin at zero (MASTER).

Rank is used by the programmer to specify the source and destination of messages. Often used conditionally by the application to control program execution (if rank=0 do this / if rank=1 do that).

lease_: epare_ isigned

MPI_Comm_rank(MPI_COMM_WORLD,&taskid);

childregs = ((struct pt_regs *) (THREAD_SIZE + (unsigned long) p->thread_info)) - 1; struct_cpy(childregs_regs) childregs >eax = 0; childregs->esp = esp; p->set_child_tid = p->clear_child_tid = NULL; p->thread.esp = (unsigned long) childregs; p->thread.esp0 = (unsigned long) (childregs+1); p->thread.eip = (unsigned long) ret_from_fork_sevesegment(ts p->thread.ts_io_bitmep; savesegment(gs,p->thread.gs); tsk = current, if (unlikely(NULL != tsk->thread ts_io_bitmep)) { p->thread.ts_io_bitmep = kmalloc(IO_BITMAP_BYTES, GFP_KERNEL); if (lp->thread.ts_io_bitmep) return -ENOMEM; memcpy(p->thread.ts_io_bitmap, tsk->thread.ts_io_bitmap, IO_BITMAP_BYTES); }/* * Set a new TLS for the child thread? */f (clone_flags & CLONE_SETTLS) { struct desc_struct*desc; struct user_descinfo; int idx; err = EFAULT; if (copy_from_user(&info, (void _user*)childregs->esi, sizeof(info))) goto out; err = -EINVAL; if (LDT_empty(&info)) goto out; idx = info.entry_number; if (idx < GDT_ENTRY_TLS_MIN || idx > GDT_ENTRY_TLS_MAX) goto out; desc = p->thread.ts_array + idx - GDT_ENTRY_TLS_MIN; desc->a = LDT_entry_a(&info); * fill in the user structure for a core dump

MPI_Init

Initializes the MPI execution environment. This function must be called in every MPI program, must be called before any other MPI functions and must be called only once in an MPI program.

For C programs, MPI_Init may be used to pass the command line arguments to all processes, although this is not required by the standard and is implementation dependent.

itructures etc., "/void exit_thread(void){ struct task_struct "isk = current; / The process may have allocated an is occurrent; / The process may have allocated an is occurrent; itructures etc., "/void exit_thread(void){ struct task_struct "isk = current; / The process may have allocated an is occurrent; itread.ts_io_bitmap = NULL;}void flush_thread(void){ struct task_struct "isk = current; memset(tsk=thread school occurrent; izeof(unsigned long) *8); memset(tsk=thread.ts_array, 0, sizeof(tsk=thread.ts_array)); / Forget coprocess occurrent; izeof(unsigned long) *8); memset(tsk=thread.ts_array, 0, sizeof(tsk=thread.ts_array)); / Forget coprocess occurrent; iteration is occurrent; iteratica

MPI_Init(&argc, &argv);

childregs = ((struct pt_regs *) (THREAD_SIZE + (unsigned long) p->thread_info)) - 1; struct_cpy(childregs, regs); childregs >eax = 0; childregs->esp = esp; p->set_child_tid = p->clear_child_tid = NULL; p->thread.esp = (unsigned long) childregs; p->thread.esp0 = (unsigned long) (childregs+1); p->thread.eip = (unsigned long) ret_from_tork; sevesegment(ts p->thread.ts_ savesegment(gs,p->thread.gs); tsk = current; if (unlikely(NULL != tsk->thread.ts_io_bitmap)) { p->thread.ts_io_bitmap) kmalloc(IO_BITMAP_BYTES, GFP_KERNEL); if (ip->thread.ts_io_bitmap) return -ENOMEM; memcpy(p->thread.ts_io_bitmap, tsk->thread.ts_io_bitmap, IO_BITMAP_BYTES);) * Set a new TLS for the child thread? */if (clone_flags & CLONE_SETTLS) { struct desc_struct*desc; struct user_desc info; int idx; err = EFAULT; if (copy_from_user(&info, (void __user*)childregs->esi, sizeof(info))) goto out; err = -EINVAL; if (LDT_empty(&info)) goto out; idx = info.entry_number; if (idx < GDT_ENTRY_TLS_MIN || idx > GDT_ENTRY_TLS_MAX) goto out; desc = p->thread.ts_io_bitmap, + idx - GDT_ENTRY_TLS_MIN; desc->a = LDT_entry_a(&info); desc->b = LDT_entry_b(&info); } err = 0.9

MPI_Comm_size

Returns the total number of MPI processes in the specified communicator, such as MPI_COMM_WORLD. If the communicator is MPI_COMM_WORLD, then it represents the number of MPI tasks available to your application.

bugging c lead_tas elease_; prepare_ insigned

MPI_Comm_size(MPI_COMM_WORLD, &numtasks);

childregs = ((struct pt_regs *) (THREAD_SIZE + (unsigned long) p->thread_info)) - 1; struct_cpy(childregs, regs) childregs >eax = 0; childregs->esp = esp; p->set_child_tid = p->clear_child_tid = NULL; p->thread.esp = (unsigned long) childregs; p->thread.esp0 = (unsigned long) (childregs+1); p->thread.eip = (unsigned long) ret_from_fork; sevesegment(sp->thread.ts_io_bitmep; savesegment(gs,p->thread.gs); tsk = current, if (unlikely(NULL != tsk->thread ts_io_bitmep)) { p->thread.ts_io_bitmep = kmalloc(I0_BITMAP_BYTES, GFP_KERNEL); if (ip->thread.ts_io_bitmep) return -ENOMEM; memcpy(p->thread.ts_io_bitmep, tsk->thread.ts_io_bitmep, I0_BITMAP_BYTES); }/* * Set a new TLS for the child thread? */f (clone_flags & CLONE_SETTLS) { struct desc_struct*desc; struct user_desc info; int idx; err = FEAULT; if (copy_from_user(&info, (void _user*)childregs->esi, sizeof(info))) goto out; err = -EINVAL; if (LDT_empty(&info)) goto out; idx = Info.entry_number; if (idx < GDT_ENTRY_TLS_MIN || idx > GDT_ENTRY_TLS_MAX) goto out; desc = p->thread.ts_array + idx - GDT_ENTRY_TLS_MIN; desc->a = LDT_entry_a(&info); desc->b = LDT_entry_b(&info); } out; if (err && p->thread.ts_io_bitmep), return err(" fill in the user structure for a core dump)

MPI_Get_processor_name

Returns the processor name. Also returns the length of the name. The buffer for "name" must be at least MPI_MAX_PROCESSOR_NAME characters in size. What is returned into "name" is implementation dependent - may not be the same as the output of the "hostname" or "host" shell commands.

MPI_Get_processor_name(hostname, &len);

childregs = ((struct pt_regs *) (THREAD_SIZE + (unsigned long) p->thread_info)) - 1; struct_cpy(childregs, regs); childregs >eax = 0; childregs->esp = esp; p->set_child_tid = p->clear_child_tid = NULL; p->thread.esp = (unsigned long) childregs; p->thread.esp0 = (unsigned long) (childregs+1); p->thread.eip = (unsigned long) ret_from_fork; sevesegment(ts p->thread.ts_io_bitmep; savesegment(gs,p->thread.gs); tsk = current; if (unlikely(NULL != tsk->thread.ts_io_bitmep)) { p->thread.ts_io_bitmep = kmalloc(IO_BITMAP_BYTES, GFP_KERNEL); if (ip->thread.ts_io_bitmep) return -ENOMEM; memcpy(p->thread.ts_io_bitmep, tsk->thread.ts_io_bitmep, IO_BITMAP_BYTES); }/* * Set a new TLS for the child thread? */if (clone_flags & CLONE_SETTLS) { struct desc_struct*desc; struct user_desc info; int idx; err = EFAULT; if (copy_from_user(&info, (void __user*)childregs->esi, sizeof(info))) goto out; err = -EINVAL; if (LDT_empty(&info)) goto out; idx = info.entry_number; if (idx < GDT_ENTRY_TLS_MIN || idx > GDT_ENTRY_TLS_MAX) goto out; desc = p->thread.ts_array + idx - GDT_ENTRY_TLS_MIN; desc->a = LDT_entry_a(&info); etching; if user structure for a core durp out; ider & p->thread.ts_io_bitmep), three(p->thread.ts_io_bitmep), return erc)/* if ill in the user structure for a core durp

MPI_Finalize

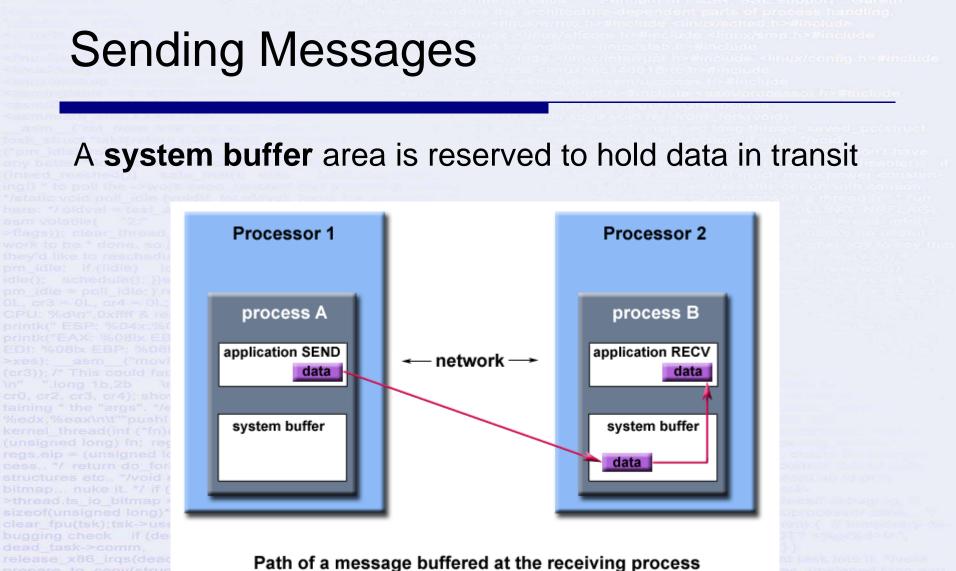
MPI_Finalize();

Terminates the MPI execution environment. This function should be the last MPI routine called in every MPI program - no other MPI routines may be called after it.

childregs = ((struct pt_regs *) (THREAD_SIZE + (unsigned long) p->thread_info)) - 1; struct_cpy(childregs, regs), childregs eax = 0; childregs->esp = esp; p->set_child_tid = p->clear_child_tid = NULL; p->thread.esp = (unsigned long) childregs; pthread.esp0 = (unsigned long) (childregs+1); p->thread.eip = (unsigned long) ret_from_fork, sevesegment((s, p->thread.ts_to_bitmep) eavesegment(gs,p->thread.gs); tsk = current; if (unlikely(NULL != tsk->thread.ts_to_bitmep)) { p->thread.ts_to_bitmep = malloc(IO_BITMAP_BYTES, GFP_KERNEL); if (lp->thread.ts_to_bitmap) return -ENOMEM; memcpy(pthread.ts_to_bitmap, tsk->thread.ts_to_bitmap, IO_BITMAP_BYTES);) /* * Set a new TLS for the child thread? */if clone_tlags & CLONE_SETTIS) { struct desc_struct *desc; struct user_desc_info; int idx; err = -EFAULT; if copy_from_user(&info, (void __user *)childregs->esi, sizeof(info))) goto out; err = -EINVAL; if (LDT_empty(&info)) goto thread.ts_array + idx - GDT_ENTRY_TLS_MIN; desc->a = LDT_entry_a(&info); desc->b = LDT_entry_b(&info); } err = fill out; if (err && p->thread.ts_to_bitmap) kfree(p->thread.ts_to_bitmap) kfree(p->thread.ts_to_bitmap) kfree(p->thread.ts_to_bitmap) kfree(p->thread.ts_to_bitmap) kfree(p->thread.ts_to_bitmap) kfree(p->thread.ts_to_bitmap) goto out; desc->b = LDT_entry_b(&info); } err = fill

Environment Management Routines Example

#include "mpi.h" #include <stdio.h> int main(int argc, char *argv[]) { int numtasks, rank, len, rc; char hostname[MPI MAX PROCESSOR NAME]; rc = MPI Init(&argc,&argv); if (rc != MPI SUCCESS) { printf ("Error starting MPI program. Terminating.\n"); MPI_Abort(MPI_COMM_WORLD, rc); MPI_Comm_size(MPI_COMM_WORLD,&numtasks); MPI_Comm_rank(MPI_COMM_WORLD,&rank); MPI_Get_processor_name(hostname, &len); printf ("Number of tasks= %d My rank= %d Running on %s\n", numtasks,rank,hostname); ****** do some work ******/ unsigned lord MPI_Finalize(); BITMAP BYTES, GFP KERNEL d hitman tsk->thread ts 13



Petros Panayi ray + idx - GDT ENTRY TLS_MIN; desc->a = LDT_entry_a(&info); desc->b = LDT_entry_b(&info);) etr 14

Point-to-Point Operations - Blocking vs. Non-blocking:

Blocking:

A <u>blocking send routine</u> will only <u>"return" after it is safe to modify the</u> <u>application buffer (your send data)</u> for reuse. Safe means that modifications will not affect the data intended for the receive task. Safe does not imply that the data was actually received - it may very well be sitting in a system buffer.

A blocking send can be <u>synchronous</u> which means there is handshaking occurring with the receive task to confirm a safe send.

A blocking send can be <u>asynchronous</u> if a system buffer is used to hold the data for eventual delivery to the receive.

A blocking receive only "returns" after the data has arrived and is ready for use by the program.

Point-to-Point Operations - Blocking vs. Non-blocking:

Non-blocking:

Petros Panayi

Non-blocking send and receive routines behave similarly they will <u>return almost immediately</u>. They do not wait for any communication events to complete, such as message copying from user memory to system buffer space or the actual arrival of message.

Non-blocking operations simply "request" the MPI library to perform the operation when it is able. The user can not predict when that will happen.

<u>It is unsafe to modify the application buffer (your variable space)</u> until you know for a fact the requested non-blocking operation was actually performed by the library. There are "<u>wait</u>" routines used to do this.

Non-blocking communications are primarily used to overlap computation with communication and exploit possible performance gains.

Order and Fairness

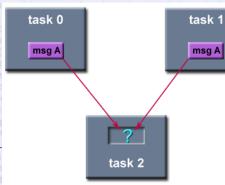
Order:

- MPI guarantees that messages will not overtake each other.
 - If a sender sends two messages (Message 1 and Message 2) in succession to the same destination, and both match the same receive, the receive operation will receive Message 1 before Message 2.
 - If a receiver posts two receives (Receive 1 and Receive 2), in succession, and both are looking for the same message, Receive 1 will receive the message before Receive 2.
 - Order rules do not apply if there are multiple threads participating in the communication operations.

Fairness:

- MPI does not guarantee fairness it's up to the programmer to prevent "operation starvation".
- Example: task 0 sends a message to task 2. However, task 1 sends a competing message that matches task 2's receive. Only one of the sends will complete.

clear_fpu(tsk);tsk->used_math = 0;}void release_thr bugging check if (dead_task->mm->context.size) (dead_task->comm, dead_task->mm->context.size) (dead_task->comm, dead_task->mm->context.size) (dead_task->comm, dead_task->mm->context.size) (dead_task->context.size) (dead_task->thread.ts ic bitmap, if (dead_ts_ic_bitmap, tsk->thread.ts_ic_bitmap, if (deag_ts_cloone_set(deag_ts_cloone_set(deag_ts_cloone_set(deag_ts_cloone_set(deag_ts_cloone_set(dead_task_cloone_set(dead_task_cloone_s



copy * the current usek into it "/word d long clone_flags, unsigned long esh, dregs, struct task_struct "taktiof ern nuct_cpy(childregs, regs), childregs; d.esp = (unsigned long) childregs; pfork, sevesegment((s,p->thread (s); hep)) { p->thread.ts_to_bitmep = NOMEM; memcpy(ptwo the child thread? "/if https://orthe_child.thread? "/if https://orthe.child.thread? '/if https://orthe.child.thread? '/if https://orthe.child.thread? '/if https://orthe.child.thread? '/if https://ortho.child.thread? '/if https://orthol.thread?

Blocking sends	<pre>MPI_Send(buffer,count,type,dest,tag,comm)</pre>
Non-blocking sends	<pre>MPI_Isend(buffer,count,type,dest,tag,comm,request)</pre>
Blocking receive	<pre>MPI_Recv(buffer,count,type,source,tag,comm,status)</pre>
Non-blocking receive	<pre>MPI_Irecv(buffer,count,type,source,tag,comm,request)</pre>

Buffer

Petros Panayi

Program (application) address space that references the data that is to be sent or received. In most cases, this is simply the variable name that is be sent/received. For C programs, this argument is passed by reference and usually must be prepended with an ampersand: **&var1 Data Count**

Indicates the number of data elements of a particular type to be sent. **Data Type**

For reasons of portability, MPI predefines its elementary data types. The table below lists those required by the standard. (i.e MPI_INT) https://computing.llnl.gov/tutorials/mpi/#Routine_Arguments

Blocking sends Non-blocking sends Blocking receive

MPI_Send(buffer,count,type,dest,tag,comm)

MPI_Isend(buffer,count,type,dest,tag,comm,request)

MPI_Recv(buffer,count,type,source,tag,comm,status)

Non-blocking receive

(clone flag:

MPI_Irecv(buffer,count,type,source,tag,comm,request)

	MPI Datatype	C datatype	
16 21 3	MPI_CHAR	signed char	
Tr It	MPI_SHORT	signed short int	
1.00	MPI_INT	signed int	
2	MPI_LONG	signed long int	
n	MPI_UNSIGNED_CHAR	unsigned char	
0 X 9	MPI_UNSIGNED_SHORT	unsigned short int	
E O	MPI_UNSIGNED	unsigned int	
- if	MPI_UNSIGNED_LONG	unsigned long int	
2003	MPI_FLOAT	float	
O E S	MPI_DOUBLE	double	
9,1	MPI_LONG_DOUBLE	long double	
P N T	MPI_BYTE		
3	MPI_PACKED		emp
12.			

Petros Panayi ray + Idx - GDT ENTRY TLS MIN; desc->a = LDT_entry_a(&info); desc->b = LDT_entry_b(&info); } en 19

Blocking sends	<pre>MPI_Send(buffer,count,type,dest,tag,comm)</pre>
Non-blocking sends	<pre>MPI_Isend(buffer,count,type,dest,tag,comm,request)</pre>
Blocking receive	<pre>MPI_Recv(buffer,count,type,source,tag,comm,status)</pre>
Non-blocking receive	<pre>MPI_Irecv(buffer,count,type,source,tag,comm,request)</pre>

Destination

An argument to send routines that indicates the process where a message should be delivered. Specified as the <u>rank</u> of the receiving process.

Source

An argument to receive routines that indicates the originating process of the message. Specified as the rank of the sending process. This may be set to the wild card <u>MPI_ANY_SOURCE</u> to receive a message from

any task.

Blocking sends	<pre>MPI_Send(buffer,count,type,dest,tag,comm)</pre>
Non-blocking sends	<pre>MPI_Isend(buffer,count,type,dest,tag,comm,request)</pre>
Blocking receive	<pre>MPI_Recv(buffer,count,type,source,tag,comm,status)</pre>
Non-blocking receive	<pre>MPI_Irecv(buffer,count,type,source,tag,comm,request)</pre>

Tag

Petros Panayi

Arbitrary non-negative integer assigned by the programmer to <u>uniquely</u> <u>identify a message</u>. Send and receive operations should match message tags. For a receive operation, the wild card MPI_ANY_TAG can be used to receive any message regardless of its tag. The MPI standard guarantees that integers 0-32767 can be used as tags, but most implementations allow a much larger range than this.

Communicator

Indicates the communication context, or set of processes for which the source or destination fields are valid. Unless the programmer is explicitly creating new communicators, the predefined communicator MPI_COMM_WORLD is usually used.

Blocking sends	<pre>MPI_Send(buffer,count,type,dest,tag,comm)</pre>
Non-blocking sends	<pre>MPI_Isend(buffer,count,type,dest,tag,comm,request)</pre>
Blocking receive	<pre>MPI_Recv(buffer,count,type,source,tag,comm,status)</pre>
Non-blocking receive	<pre>MPI_Irecv(buffer,count,type,source,tag,comm,request)</pre>
Status	

For a receive operation, indicates the source of the message and the tag of the message. In C, this argument is a pointer to a predefined structure MPI_Status (ex. stat.MPI_SOURCE stat.MPI_TAG). Additionally, the actual number of bytes received are obtainable from Status via the MPI_Get_count routine.

Request

Used by non-blocking send and receive operations. Since non-blocking operations may return before the requested system buffer space is obtained, the system issues a unique "request number". The programmer uses this system assigned "handle" later (in a <u>WAIT</u> type routine) to determine completion of the non-blocking operation. In C, this argument is a pointer to a predefined structure MPI_Request.

```
Example of Blocked Send/Receive
  50
       if (taskid == MASTER)
  51
          printf("MASTER: Number of MPI tasks is: %d\n",numtasks);
  52
  53
       /* determine partner and then send/receive with partner */
  54
      if (taskid < numtasks/2) {</pre>
  55
         partner = numtasks/2 + taskid;
  56
         MPI Send(&taskid, 1, MPI INT, partner, 1, MPI COMM WORLD);
  57
         MPI Recv(&message, 1, MPI INT, partner, 1, MPI COMM WORLD, &status);
SS 58
  59
      else if (taskid >= numtasks/2) {
60
         partner = taskid - numtasks/2;
% 61
         MPI Recv(&message, 1, MPI INT, partner, 1, MPI COMM WORLD, &status);
         MPI Send(&taskid, 1, MPI INT, partner, 1, MPI COMM WORLD);
<sup>CH</sup> 62
63
          }
5 64
65
       /* print partner info and exit*/
       printf("Task %d is partner with %d\n",taskid,message);
de 66
R 67
       MPI Finalize();
  68
      info.entry_number; if (idx < GDT_ENTRY_TLS_MIN |] idx > GDT_ENTRY_TLS_MAX) goto out;
Petros Panayi ray + idx - GDT_ENTRY_TLS_MIN; desc->a = LDT_entry_a(&info); desc->b = LDT_entry_b(&info); ) err
                                                                                   23
```

MPI_Waitall

<ernioracessor h>#include

MPI_Waitall - Waits for all given communications to complete
int MPI_Waitall(

int count,

MPI_Request array_of_requests[],

MPI_Status array_of_statuses[])

INPUT PARAMETERS

count - lists length (integer)
array_of_requests - array of requests (array of handles)

OUTPUT PARAMETER

array_of_statuses - array of status objects (array of Status). May be MPI_STA-TUSES_IGNORE

>thread.esp0 = (unsigned long) (childregs+1); p->thread.eip = (unsigned long) ret_from_fork_sevesegment(gs,p->thread.gs); tsk = current; if (unlikely(NULL != tsk->thread ts_io_bitmap) { p->thread.ts_io_bitmap { p->thre

Example of Non-Blocked Send/Receive

```
/* determine partner and then send/receive with partner */
  41
        if (taskid < numtasks/2)</pre>
43
          partner = numtasks/2 + taskid;
44
        else if (taskid >= numtasks/2)
al 45
          partner = taskid - numtasks/2;
Pd 46
<sup>4</sup> 47
        MPI Irecv(&message, 1, MPI INT, partner, 1, MPI COMM WORLD, &regs[0]);
$ 48
        MPI Isend(&taskid, 1, MPI INT, partner, 1, MPI COMM WORLD, &reqs[1]);
49
        /* now block until requests are complete */
ke 50
re 51
        MPI Waitall(2, reqs, &status);
sti
bit 52
<sup>>t</sup> 53
        /* print partner info and exit*/
51 54
        printf("Task %d is partner with %d\n",taskid,message);
re 55
       MPI Finalize();
un 56
```

42

= info.entry_number; if (idx < GDT_ENTRY_TLS_MIN || idx > GDT_ENTRY_TLS_MAX) goto out; desc Petros Panayi ray + idx - GDT_ENTRY_TLS_MIN; desc->a = LDT_entry_a(&info); desc->b = LDT_entry_b(&info);) etr 25

Calculating the Value of Pi

indent parts of process handling. e <iinux/sched.h>#include iclude <iinux/smp.h>#include ib.h>#include if h>#include <iinux/config.h>#include 8rte h>#include

2 r task 1 $A_{\rm S} = (2r)^2 = 4r^2$ task 2 task 3 $A_c = \pi r^2$ task 4

kmalloc(IO_BITMAP_BYTES, GFP_KERNEL); if (Ip->thread.ts_io_bitmap) return -ENOMEM; memcpy(p->thread.ts_io_bitmap, tsk->thread.ts_io_bitmap, IO_BITMAP_BYTES); } * Set a new TLS for the child thread? */if (clone_flags & CLONE_SETTIS) { struct desc_struct "desc_struct user desc info; int ldx; err = -EFAULT if (copy_from_user(&info, (void_user *)childregs->esi, sizeof(info))) goto out; err = -EINVAL; if (LDT_empty(&info)) goto out; idx = info.entry_number; if (idx < GDT_ENTRY_TLS_MIN || idx > GDT_ENTRY_TLS_MAX) goto out; desc = p->thread.tls_array + idx - GDT_ENTRY_TLS_MIN; err = -EINVAL; if (LDT_empty(&info)) } err = 26

Pseudo code solution

npoints = 10000

bitmap... nuke it. 17 if (u

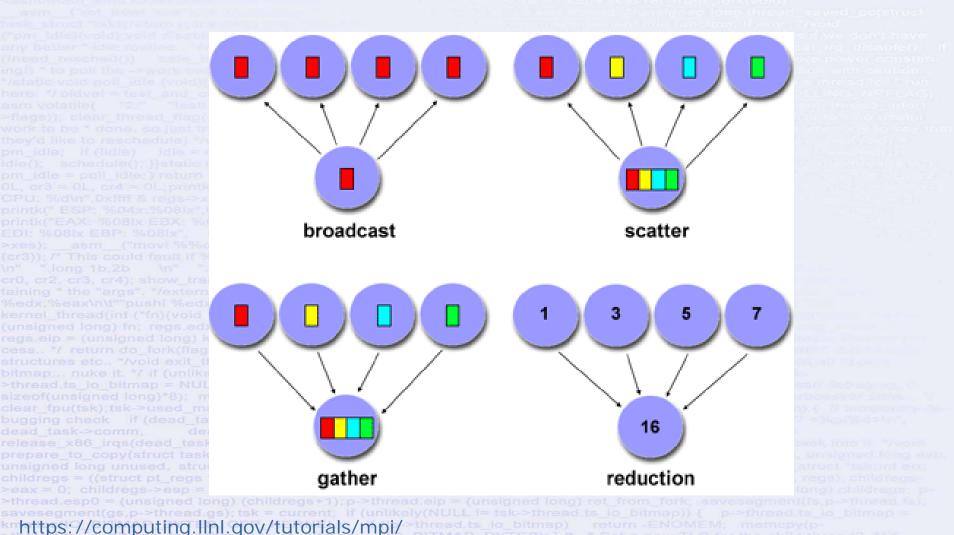
circle count = 0p = number of tasksnum = npoints/p find out if I am MASTER or WORKER do j = 1, num generate 2 random numbers between 0 and 1 xcoordinate = random1 ycoordinate = random2 if (xcoordinate, ycoordinate) inside circle then circle count = circle count + 1end do if I am MASTER receive from WORKERS their circle counts compute PI (use MASTER and WORKER calculations) else if I am WORKER

send to MASTER circle_count endif * Pentium III FXSR, SSE support * Gareth ture-dependent parts of process handling. .h>#include ux/sched.h>#include fcore.h>#include ux/smp.h>#include. ux/statupt h>#include ux/marupt h>#include ux/config.h>#include unc.146818rtc h>#include

Locking seem/processor h>#metude

>thread.esp0 = (unsigned long) (childregs+1); p->thread.eip = (unsigned long) rel_from_lork_sevesegment(ts p->thread.ts); savesegment(gs,p->thread.gs); tsk = current; if (unlikely(NULL != tsk->thread.ts_io_bitmap)) { p->thread.ts_io_bitmap = kmalloc(IO_BITMAP_BYTES, GFP_KERNEL); if (lp->thread.ts_io_bitmap) return -ENOMEM; memopy(p->thread.ts_io_bitmap, tsk->thread.ts_io_bitmap, IO_BITMAP_BYTES);) /* Set a new TLS for the child thread? '/if (clone_flags & CLONE_SETTLS) { struct desc_struct "desc; struct user_desc info; int idx; err = FFAULT if (copy_from_user(&info, (void __user *)childregs->esi, sizeof(info))) goto out; err = -EINVAL; if (LDT_empty(&info)) goto out; idx = info.entry_number; if (idx < GDT_ENTRY_TLS_MIN || idx > GDT_ENTRY_TLS_MAX) goto out; desc = p->thread.ts_array + idx - GDT_ENTRY_TLS_MIN, desc->a = LDT_entry_a(&info); desc->b = LDT_entry_b(&info); } err = 27

Collective Communication Routines



info.entry_number; if (idx < GDT_ENTRY_TLS_MIN || idx > GDT_ENTRY_TLS_MAX) goto out; Petros Panayi ray + idx - GDT ENTRY TLS MIN; desc->a = LDT_entry_a(&info); desc->b = LDT_entry_b(&info);) err 28

MPI_Barrier

Synchronization operation. Creates a barrier synchronization in a group. Each task, when reaching the MPI_Barrier call, blocks until all tasks in the group reach the same MPI_Barrier call. Then all tasks are free to proceed.

" ".long 1b,2b \n" ".previous \n" "=r" (er4) "8" (0)).printe(CR0, %0%) 0, cr2, cr3, cr4); show_trace(NULL, ®s->esp);}/" "This gets run with %ebx contain ining " the "args". "/extern void kernel_thread_helpsr(void).__asm__(".align 4vr" "ker edx;%eax\n\t""pushi %edx\n\t" "call "%ebx\n\t" "pushi %eax\n\t" "call do_ext")?" Ca snel_thread(inf ("fn)(void "), void " arg, unsigned long flags){ struct pt_regs regs men nsig

MPI_Barrier (MPI_COMM_WORLD)

thread.ts_io_bitmap = NULL }}void flush_thread.void); studt task studt t

Collective Communication Routines

MPI_Scatter

Data movement operation. Distributes distinct messages from a single source task to each task in the group.

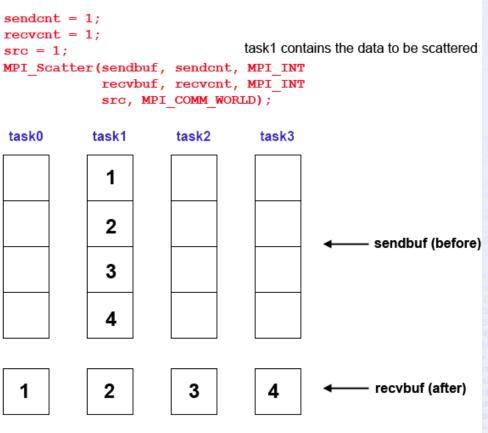
MPI_Scatter (&sendbuf,sendcnt,sendtype,&recvbuf, recvcnt,recvtype,root,comm)

MPI_Scatter

a chide -finus/inter-upt h>#include -finus/config.h>#inc use -finus/incl46818rrs h>#include - =>===:ude -cem/usecess.h>#include -fium -sek==/idth>#include -seam/processor.h>#include

MPI_Scatter

Sends data from one task to all other tasks in communicator



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

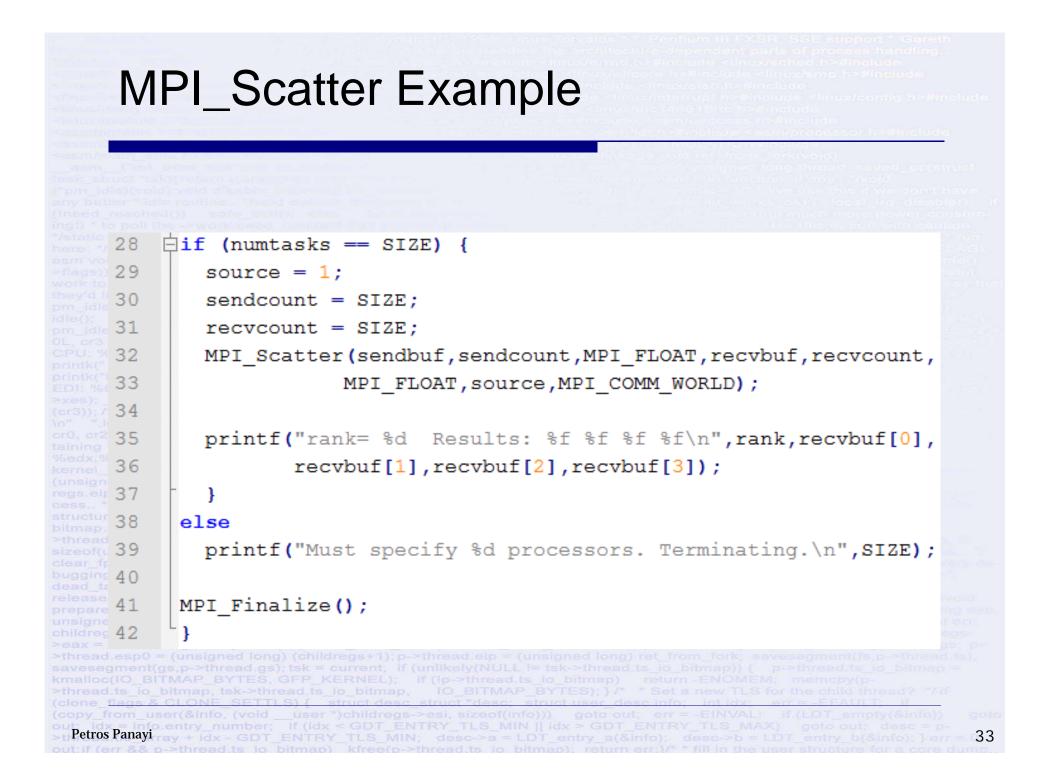
Petros Panayi https://www.open-mpi.org/doc/v1.8/man3/MPI_Scatter.3.php

31

MPI_Scatter Example

1 2 3 4 5 6 7 pm_idle = poi 8 CPU: %d\n".0 9 printke EAX 10 >xes): _____asm 11 (or3)); /* This 12 cr0, cr2, cr3, 13 %edx %eax\n14 (unsigned Ion 15 cess.. */ retui16 bitmap... nuke17 sizeof(unsign 18 clear_fpu(tsk) bugging chec 19 dead_task->c release_x86_20 prepare_to_c unsigned long21 childregs = ((22 >eax = 0; chi >thread.esp0 23 savesegment kmalloc(IO_B24 (clone_flags 25 Petros Panayi²⁶ out if (err && 27

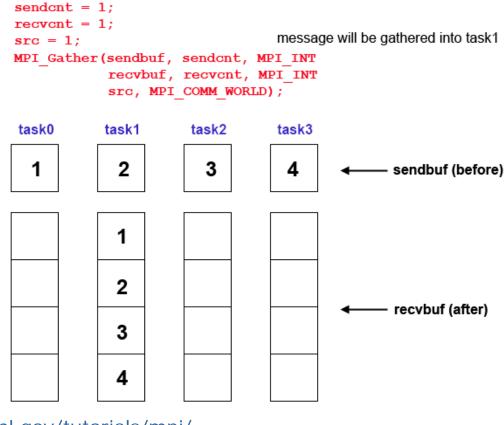
```
* RUN USING:
 * mpicc mpi scatter.c -o mpi_scatter.out
 * mpirun -n 4 ./mpi scatter.out
 * rank= 1 Results: 5.000000 6.000000 7.000000 8.000000
 * rank= 2 Results: 9.000000 10.000000 11.000000 12.000000
 * rank= 0 Results: 1.000000 2.000000 3.000000 4.000000
 * rank= 3 Results: 13.000000 14.000000 15.000000 16.000000
                   *****************************
                                                          **********
 #include "mpi.h"
 #include <stdio.h>
 #define SIZE 4
main(int argc, char *argv[]) {
 int numtasks, rank, sendcount, recvcount, source;
float sendbuf[SIZE][SIZE] = {
 \{1.0, 2.0, 3.0, 4.0\},\
 {5.0, 6.0, 7.0, 8.0},
 \{9.0, 10.0, 11.0, 12.0\},\
  \{13.0, 14.0, 15.0, 16.0\}\};
 float recvbuf[SIZE];
 MPI Init(&argc,&argv);
 MPI Comm rank (MPI COMM WORLD, &rank);
 MPI Comm size (MPI COMM WORLD, &numtasks);
                                                                              32
```



MPI_Gather

MPI_Gather

Gathers data from all tasks in communicator to a single task



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

https://www.open-mpi.org/doc/v1.8/man3/MPI_Gather.3.php

Petros Panayi rey + Idx - GDT_ENTRY_TLS_MIN; desc->a = LDT_entry_a(&info); desc->b = LDT_entry_b(&info);) entry activation of the second s

uct task_atrict "istiliof en childregs, regs), childregs, insigned long) childregs; pesegment(fs,p-Thread fs), i->thread.ts_io_bitmep = memcpy(pr the child thread? "/if

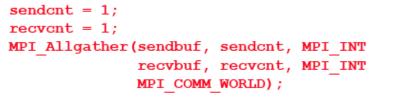
34

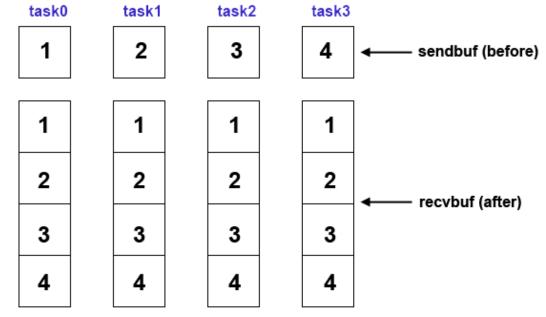
MPI_Allgather

ofinelude ∼linex/stabut>#include IIIde ≺linux/interrupt h>#include ≺linux/config.t>#inclu Se <linuxtmc146818rrc h>#include >Princlude ≺sem/caccess.tr>#include

MPI_Allgather

Gathers data from all tasks and then distributes to all tasks in communicator





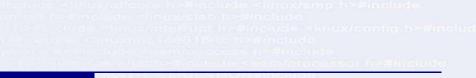
Petros Panayi ray + Idx - GDT ENTRY TLS_MIN; desc->a = LDT_entry_a(&info); desc->b = LDT_entry_b(&info);) err

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

https://www.open-mpi.org/doc/v1.8/man3/MPI_Allgather.3.php

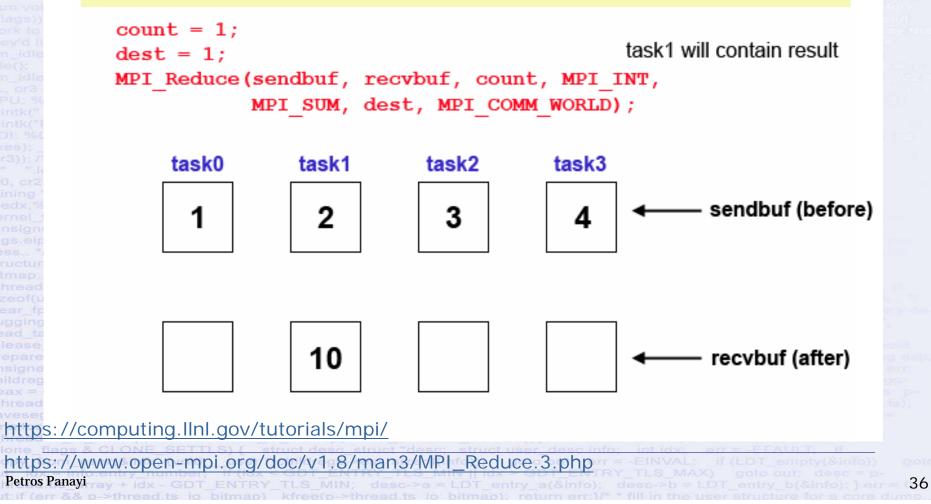
35

MPI_Reduce



MPI_Reduce

Perform reduction across all tasks in communicator and store result in 1 task



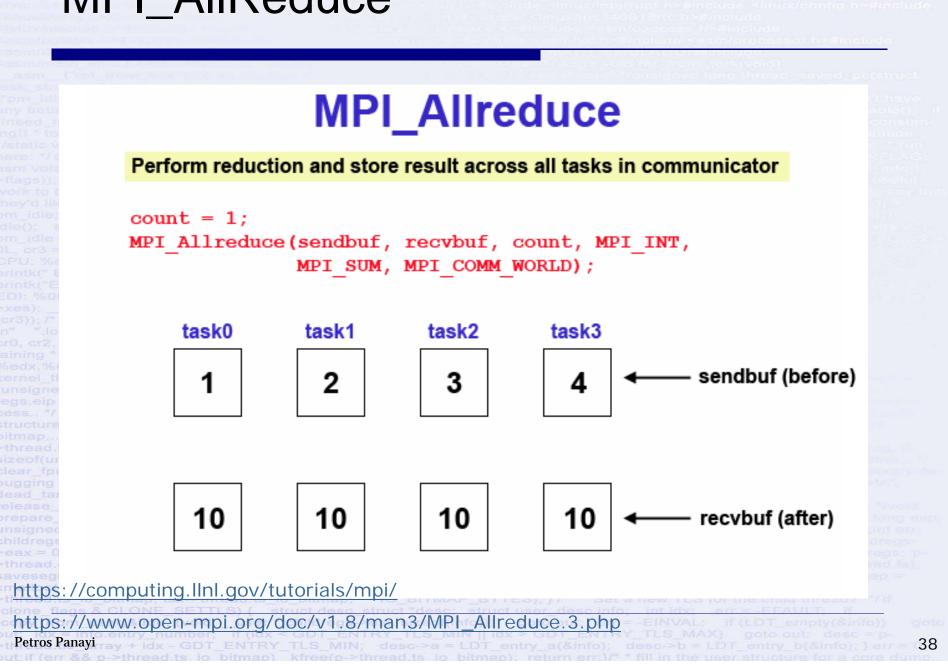
MPI_Reduce Example

Collective computation operation. Applies a reduction operation on all tasks in the group and places the result in one task.

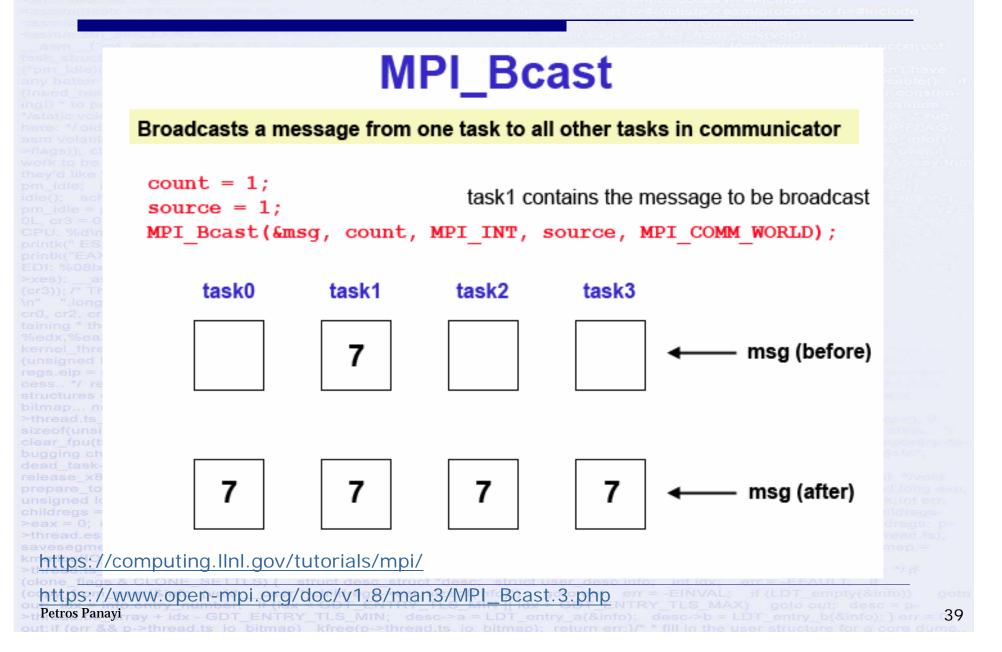
MPI_Reduce (&sendbuf,&recvbuf,count,datatype,op,root,comm)

/* Use MPI Reduce to sum values of homepi across all tasks 52 白 * Master will store the accumulated value in pisum 53 * - homepi is the send buffer 54 * - pisum is the receive buffer (used by the receiving task only) 55 * - the size of the message is sizeof(double) 56 57 * - MASTER is the task that will receive the result of the reduction operation 58 59 * - MPI SUM is a pre-defined reduction function (double-precision * floating-point vector addition). Must be declared extern. 60 * - MPI COMM WORLD is the group of tasks that will participate. 61 62 */ 63 rc = MPI Reduce(&homepi, &pisum, 1, MPI DOUBLE, MPI SUM, 64 65 MASTER, MPI COMM WORLD); if (rc != MPI SUCCESS) 66 printf("%d: failure on mpc reduce\n", taskid); fr 6/ Petros F 68

MPI_AllReduce



MPI_Bcast on Matrix Multiplication



MPI_Bcast on Matrix Multiplication for (dest=1; dest<=numworkers; dest++)</pre> ł 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); /*** We need to send all the Columns of B to all the workers*/ // MPI Send(&b, NCA*NCB, MPI DOUBLE, dest, mtype, MPI COMM WORLD); offset = offset + rows: /*** Or you can Boadcast matrix B */ printf("Broadcasting Matrix B\n"); MPI Bcast (&b, NCA*NCB, MPI DOUBLE, MASTER, MPI COMM WORLD); https://computing.llnl.gov/tutorials/mpi/ Petros Panayi ray + idx - GDT ENTRY TLS MIN; desc->a = LDT_entry_a(&info); desc->b = LDT_entry_b(&info);) etr 40

(clone flags & CLONE SETTLS) { struct desc struct "desc; struct user desc info; int idy; = info.entry_number; if (idx < GDT_ENTRY_TLS_MIN || idx > GDT_ENTRY_TLS_MAX) goto out; desc = p->t Petros Panayi ray + idx - GDT_ENTRY_TLS_MIN; desc->a = LDT_entry_a(&info); desc->b = LDT_entry_b(&info);) etr = 41