PPCES2010 - MPI Introduction

Lab Exercises
March 22, 2010

Part 1: MPI Basic Exercises on the Virtual Machine

To start the VM on the laptop, double-click on the **VMware** icon on the desktop. Click on the *SUS_HPC_Kurs_1* tab and then on the green arrow to power on this virtual machine. Log in with "**hpclab**" as user name (the password will be announced during the lab).

You can exchange data between the Laptop's Windows 7 host system and the VM with a shared folder, which is mounted in the VM at **/mnt/hgfs/shared/PPCES2010.**

You can also access the cluster from within the VM and exchange data with **ssh** and **scp** commands.
The examples are prepared to work with the Sun compiler and the Sun MPI library on the VM. You don't need to use any module commands there.

Click on the desktop icon labeled with *Terminal* to open a console.
Extract the MPI exercises into your VM home directory from either the shared folder

```
cd /home/hpclab
gtar xf /mnt/hgfs/SharedFolder/PPCES2010/MPI/mpi_intro.tar
```

or from the cluster

```
cd /home/hpclab
scp hpclabXZ@cluster-linux:/home/hpclab/PPCES2010/MPI/mpi_intro.tar .
gtar xf mpi_intro.tar
```

You can find information about the MPI routines in the manual pages (man MPI_Xxxx) or in the web at http://www-unix.mcs.anl.gov/mpi/ or http://www-unix.mcs.anl.gov/mpi/www/www3/ .

Exercise 1

Change into the directory simple/c or simple/f depending on your preference for C or Fortran

```
cd ~/mpi_intro/simple/X
```

Compile, link and execute program "simple" with 4 MPI processes

```
gmake clean
gmake build
gmake go
```

Repeat the execution several times. What do you observe?

Exercise 2

Modify this program "simple", such that the master (rank 0)
reads a positive integer number n from standard input
and sends it to all slaves (rank 1 ... rank np-1)

Exercise 3

Change into the directory ring/c or ring/f depending on your preference for C or Fortran

cd ~/mpi_intro/ring/X

Modify this program, such that the master (rank 0)
initializes an array of n 8-byte real numbers with values 1.0, 2.0, ... , n .
Then this array shall be passed around all processes from rank 0 to rank 1 to rank 2, ... to rank np-1, and then back to rank 0 like in a ring.

Exercise 4

Modify this program, such that the total sum of all these numbers of all process
is printed out by the master.
Check the result: np * (1.0 + 2.0 + …+ np) = np * n * (n + 1) / 2

Exercise 5

Change into the directory fix_me/c or fix_me/f depending on your preference for C or Fortran

cd ~/mpi_intro/fix_me/X

The program contains a bug. What is the symptom? What is the problem?

gmake clean
gmake build
gmake go NPROCS=2

or use the TotalView debugger (does not currently not work for the C version in the VM)

gmake tv

Exercise 6

Change into the directory pi/c or pi/f depending on your preference for C or Fortran

cd ~/mpi_intro/pi/X

Improve the program pi, which contains a very stupid parallel approach
of parallelizing a numerical integration.

Exercise 7a

Change into the directory jacobi/c or jacobi/f depending on your preference for C or Fortran

cd ~/mpi_intro/jacobi/X

The program solves a finite difference discretization of the Helmholtz equation:

(d2/dx2)u + (d2/dy2)u - alpha u = f

using the Jacobi iterative method.
The parallelization of the subroutine jacobi is very dangerous. Why?
Change the exchange of the ghost cells using
i) **MPI_Sendrecv**
ii) asynchronous receive operations **MPI_Irecv**
iii) asynchronous receive and asynchronous send operations **MPI_Irecv, MPI_Isend**

In order to continue your work on the cluster under Linux, you can pack and transfer your solutions

cd ~
tar cf my_mpi_lab.tar mpi_intro
scp my_mpi_lab.tar hpclabXZ@cluster-linux

Part 2: MPI Exercises on the Linux-Cluster

(find the MPI Exercises for Windows on the Windows HPC and Visual Studio Lab Sheet)

Login into the Linux-Cluster with your lab account **hpclabXZ** using the NX-frontend machines:

cluster-x.rz.rwth-aachen.de
cluster-x2.rz.rwth-aachen.de

Open a new connection by double-click on the NX icon on the Desktop and enter the hostname, user name (login) and password.
For your first login with NX you need to click on *Configure* to perform some advanced tuning. You should specify the expected bandwidth of your network connection (*ADSL*). We recommend to disable the direct draw option: ***Configure --> Advanced --> Disable direct draw for screen rendering***
After you started an NX session, you should open a terminal and log in to the lab machine (where *Z* is the rightmost digit on your badge):

ssh –l hpclabXZ lab*Z*

Either extract the original MPI exercises into your cluster home directory

cd ~
gtar xf /home/hpclab/PPCES2010/MPI/mpi_intro.tar .

Or continue work on the files which you transferred from the VM previously

cd ~
gtar xf /home/hpclab/PPCES2010/MPI/my_mpi_lab.tar .

The examples are prepared for the Sun Studio compilers and Sun MPI library.
Use the **module** command to check and prepare your environment.

Using the Batch System

Long-running computations and computations which must run on empty machine to minimize side effects (e.g. benchmarks) should be submitted to the batch system (Oracle Grid Engine, formerly known as SGE). A batch script must be submitted using the **qsub** command (or alternatively using the **qmon** GUI which is not covered here). A batch script is basically a shell script containing all commands which must be executed, and all (recommended) necessary options for the batch system. It is also possible to provide all options as qsub command line parameters. An example batch script is the *submit.sh* file. Note, that the batch script can (and should!) be tested interactively before submitting. The interactive test lets you directly find eventual errors omitting the batch system waiting time. To let the batch file run interactively, it must be marked as “executable” by the command:

chmod 755 submit.sh

and then execute it by

./submit.sh

After you have tested your script you can submit it to the batch system:

qsub submit.sh

The batch system tells you the job ID of the submitted job:

Your job xxxxxx ("submit.sh") has been submitted

The status of all your jobs can be controlled by the **qstat** command. Only waiting jobs and jobs in an error state are shown; finished jobs are not listed.
You can cancel a job with the **qdel** command:

qdel *xxxxxx*

where *xxxxxx* is the job ID.

Exercise 7b

Which of the solutions of exercise 7a runs best on the HPC cluster?
Read the input data from file **input.big**

Exercise 8

Change into the directory hybrid/c or hybrid/f depending on your preference for C or Fortran

cd ~/mpi_intro/hybrid/X

The program jacobi solves a finite difference discretization of the Helmholtz equation:

(d2/dx2)u + (d2/dy2)u - alpha u = f

using Jacobi iterative method. The parallelization employs a combination of MPI and OpenMP.
Compile and start the program interactively and in batch mode.
Do first tests with the small input data set in file **input** and then with the larger data set in file **input.big**.
What is the optimal number of MPI processes and OpenMP threads per MPI process when running the program on two nodes?

Exercise 9

Like Exercise 8. Use the Intel compiler and compare the performance.

PPCES2010 - MPI Introduction

Solutions of the Lab Exercises
March 22, 2010

Solution 1

The MPI processes print out their rank id and execution time in a non-deterministic sequence.

Solution 2

```
// Do work here ...
     if ( myrank == 0 ) {
       printf("Enter integer number:\n");
       scanf("%d",&number);
     }

     if ( MPI_Bcast(&number,1,MPI_INT,0,MPI_COMM_WORLD) != 0 ) {
          printf ("error!\n"); return 1;
     }
     printf("Rank %d: number %d received\n", myrank, number);
```

```
! Do work here ...
     if ( myrank == 0 ) then
       write (*,*) "Enter integer number:"
       read(*,*) number
     end if

     call MPI_Bcast (number, 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
     write (*,*) "Rank ", myrank, ": number ", number, " received"
```

Solution 5

The message tags of the send and receive calls do not match. Because the amount of data which is sent is only small, the send operation is non-blocking and returns immediately. Thus rank 0 runs into the barrier and waits for rank 1 to reach the barrier too, while rank 1 infinitely waits at the receive call.

Solutions 3 + 4

```
//! Do work here ...
     left  = (nprocs + myrank - 1) % nprocs;
     right =  (myrank + 1) % nprocs;
     tag = 23;
     printf("myrank= %d left neighbor= %d rigth neighbor= %d\n", \
      myrank,left,right);
     MPI_Barrier ( MPI_COMM_WORLD );

     while(TRUE)
     {
        // The master reads an integer number form standard input
        // and sends it to all slaves
        if ( myrank == master )
        { //----------------------- Master part-----------------
          printf("Array size ? (0 quits, max %d = 128MB\n",1024*1024*16);
          scanf("%d",&n);
        }
        MPI_Bcast ( &n, 1, MPI_INT, master, MPI_COMM_WORLD);
        if ((n <= 0) || (n > 1024*1024*128 )) break;
        array = (double *) calloc(n,sizeof(double)) ;

        if( myrank == 0 )
        { //----------------------- Master part--------------
           for(i=0;i<n;i++)
              array[i] = i+1;
           t1 = MPI_Wtime();
           MPI_Send ( array, n, MPI_DOUBLE, right, tag, MPI_COMM_WORLD);
           MPI_Recv ( array, n, MPI_DOUBLE, left, tag, MPI_COMM_WORLD, \
                MPI_STATUS_IGNORE);
           t2 = MPI_Wtime();
           printf(" message length %d bytes   transfer rate %lf  MB/s\n", \
                n,1e-6 * (double)(8 * n * nprocs) / (t2-t1));
           printf("Rank %d finalizing after %lf seconds\n", myrank, t2-t1);
        }
        else
        { //----------------------- Slave part--------------------
           for(i=0;i<n;i++)
              array[i] = 0.0;
           MPI_Recv ( array, n, MPI_DOUBLE, left, tag, MPI_COMM_WORLD, \
                MPI_STATUS_IGNORE);
           MPI_Send ( array, n, MPI_DOUBLE, right, tag, MPI_COMM_WORLD);
        }
        mysum = 0.0;
        for (i=0;i<n;i++) mysum += array[i];
        MPI_Reduce (&mysum, &totalsum, 1, MPI_DOUBLE, MPI_SUM, 0, \
           MPI_COMM_WORLD);
        if ( myrank == 0 ) // Master part
           printf("the total sum is %f    check %f\n", totalsum, \
                (double)nprocs * n * (n + 1 ) / 2);
        free( array );
     }
```

Solution 6

```
//! Do work here ...
     MPI_Barrier ( MPI_COMM_WORLD );
     if ( myrank == master )
     {
        while(TRUE)
        {
          printf("Enter number of intervals ? (0 quits, max %d )\n", \
                1024*1024);
          scanf("%d",&n);
          // measure runtime
          t1 = MPI_Wtime() ;
          if ( n <= 0 || n > 1024*1024 ) {
            // tell all slaves to terminate
            for (islave=1; islave<= nprocs-1; islave++) {
              x = 0.0;
              MPI_Send (&x,1,MPI_DOUBLE,islave,ready_tag, MPI_COMM_WORLD );
            }
            break;
          }

          h = 1.0 / n;
          sum = 0.0;

          for ( i = 1; i<= min(n,nprocs-1); i++) {
            x = h * ((double)i-0.5);
            MPI_Send ( &x, 1, MPI_DOUBLE, i, work_tag, MPI_COMM_WORLD );
          }

          for ( i = nprocs; i<=n; i++ ) {
            MPI_Recv ( &fx, 1, MPI_DOUBLE, MPI_ANY_SOURCE, MPI_ANY_TAG, \
                MPI_COMM_WORLD, &status );
            sum += fx;
            x = h * ((double)i-0.5);
            MPI_Send ( &x, 1, MPI_DOUBLE, status.MPI_SOURCE, work_tag, \
                MPI_COMM_WORLD );
          }
          for ( i = 1; i<=min(n,nprocs-1); i++) {
            MPI_Recv ( &fx, 1, MPI_DOUBLE, MPI_ANY_SOURCE, MPI_ANY_TAG, \
                MPI_COMM_WORLD, &status );
            sum += fx;
          }
          pi = h * sum;
          // print the answer.
          printf("pi is approximately: %12.9f    Error is: %12.9e\n", pi, \
                fabs(pi-pi25dt));
          // measure runtime
          t2 = MPI_Wtime();
          printf("Rank %d finalizing after %lf seconds\n", myrank, t2-t1);
        }

      } else { // slave

          while (TRUE) {
```

```
            // waiting for work
            MPI_Recv ( &x, 1, MPI_DOUBLE, master, MPI_ANY_TAG, \
                MPI_COMM_WORLD, &status );
            if ( status.MPI_TAG == ready_tag ) {
              break;
            } else if ( status.MPI_TAG == work_tag ) {
              fx = func(x);
              MPI_Send ( &fx, 1, MPI_DOUBLE, master, ready_tag, \
                MPI_COMM_WORLD );
            } else {
              printf ("received illegal tag\n");
              MPI_Abort(MPI_COMM_WORLD, 99);
            }
          }
        }
```

Solution 7a

In the original program version all processes send their data to the neighbors first and then execute the receive call. For small messages, which are usually sent non-blocking this might work well. But for longer messages, for which the MPI runtime does not provide enough buffer space, the send call may not be able to return.

Solution 7a - iii

```
      do while (k.le.maxit .and. error.gt. tol)
          error = 0.0

! Copy new solution into old
          left = me-1
          if ( me .eq. 0 ) left = MPI_PROC_NULL
          right = me+1
          if ( me .eq. np-1 ) right = MPI_PROC_NULL
          reqcnt = 0
!            receive stripe mlo from left neighbour
             reqcnt = reqcnt + 1
             call MPI_IRECV( uold(1,mlo), n, MPI_DOUBLE_PRECISION, &
     &          left, 11, MPI_COMM_WORLD,reqary(reqcnt),ierr)
!            receive stripe mhi from right neighbour
             reqcnt = reqcnt + 1
             call MPI_IRECV( uold(1,mhi), n, MPI_DOUBLE_PRECISION, &
     &          right, 12, MPI_COMM_WORLD,reqary(reqcnt),ierr)
!            send stripe mhi-1 to right neighbour
             reqcnt = reqcnt + 1
             call MPI_ISEND ( u(1,mhi-1), n, MPI_DOUBLE_PRECISION, &
     &          right, 11, MPI_COMM_WORLD,reqary(reqcnt),ierr)
!            send stripe mlo+1 to left neighbour
             reqcnt = reqcnt + 1
             call MPI_ISEND ( u(1,mlo+1), n, MPI_DOUBLE_PRECISION, &
     &          left, 12, MPI_COMM_WORLD,reqary(reqcnt),ierr)

          do j=mlo+1,mhi-1
             do i=1,n
                uold(i,j) = u(i,j)
             enddo
          enddo

          call MPI_WAITALL ( reqcnt, reqary, reqstat, ierr)
          if ( ierr .ne. 0 ) then
             do i = 1, reqcnt
                print *, me, ': ', i, (reqstat(j,i),j=1,MPI_STATUS_SIZE)
             end do
          end if

! Prevent the compiler from moving access to u and uold in front of the
! MPI_WAITALL call

          call MPI_Get_address ( u, address, ierr)
          call MPI_Get_address ( uold, address, ierr)
```

```
! Compute stencil, residual, & update
         do j = mlo+1,mhi-1
            do i = 2,n-1
!     Evaluate residual
               resid = (ax*(uold(i-1,j) + uold(i+1,j)) &
     &                + ay*(uold(i,j-1) + uold(i,j+1)) &
     &                 + b * uold(i,j) - f(i,j))/b
! Update solution
               u(i,j) = uold(i,j) - omega * resid
! Accumulate residual error
               error = error + resid*resid
            end do
         enddo
         error_local = error
         call MPI_ALLREDUCE ( error_local, error,1, &
     &      MPI_DOUBLE_PRECISION,MPI_SUM,MPI_COMM_WORLD,ierr)
! Error check
         k = k + 1
         error = sqrt(error)/dble(n*m)
!
      enddo                     ! End iteration loop
```

Solution 7 a - i

```
!    exchange stripe with right neighbor
     left = me-1
     if ( me .eq. 0 ) left = MPI_PROC_NULL
     right = me+1
     if ( me .eq. np-1 ) right = MPI_PROC_NULL

     call MPI_Sendrecv ( u(1,mhi-1), n, MPI_DOUBLE_PRECISION, right, 11,  &
          uold(1,mhi), n, MPI_DOUBLE_PRECISION, right, 11, &
          MPI_COMM_WORLD, MPI_STATUS_IGNORE, ierr)

!    exchange stripe with left neighbor
     call MPI_Sendrecv ( u(1,mlo+1), n, MPI_DOUBLE_PRECISION, left, 11,  &
     &     uold(1,mlo), n, MPI_DOUBLE_PRECISION, left, 11, &
           MPI_COMM_WORLD, MPI_STATUS_IGNORE, ierr)
```