Lab 4: Introduction to MPI

It’s important to understand that MPI is HUGE, it’s solidly the most important API in HPC

There are hundreds of constructs available in MPI, we’re going to look at maybe… 4. The level of depth you get to in MPI is up to you if you want to do this.

MPI is up to version 4 (experimental) and 3.2/3.3 (stable), and has been around since the 1990s, but really the whole thing builds on itself, so we are mostly interested in the early features that let you do message passing for parallel algorithms, the more advanced features are more for optimising real code.

1. First Step: You will need to install MPI

sudo apt install mpich

sudo apt-get install openmpi-bin openmpi-common libopenmpi-dev libgtk2.0-dev

You’ll need a program:

|  |
| --- |
| /\* Compile command line with:  mpicxx Lab4MPI.cpp -o lab4 run with:  mpirun -np 4 ./lab4  \*/  // NOTE THIS IS A C Program not a C++ one, not that it matters much.  #include <mpi.h>  #include <stdio.h>  int main**(**int argc**,** char**\*\*** argv**)** **{**  // Initialize the MPI environment  MPI\_Init**(NULL,** **NULL);**  // Get the number of processes  int world\_size**;**  MPI\_Comm\_size**(**MPI\_COMM\_WORLD**,** **&**world\_size**);**  // Get the rank of the process  int world\_rank**;**  MPI\_Comm\_rank**(**MPI\_COMM\_WORLD**,** **&**world\_rank**);**  // Get the name of the processor  char processor\_name**[**MPI\_MAX\_PROCESSOR\_NAME**];**  int name\_len**;**  MPI\_Get\_processor\_name**(**processor\_name**,** **&**name\_len**);**  // Print off a hello world message  printf**(**"Hello world from processor %s, rank %d out of %d processors\n"**,**  processor\_name**,** world\_rank**,** world\_size**);**  // Finalize the MPI environment.  MPI\_Finalize**();**  **return** 0**;**  **}** |

Compile and run:

mpicxx Lab4MPI.cpp -o lab4  
mpirun -np 4 ./lab4

Notice that to take advantage of MPI requires using the specific MPI compiler and to invoke the program with an MPI runtime

So what was wrong with that program? First, every thread does the same thing, second it does in random order, third, there’s not really much of a message being passed.

1. Let’s start with the last one – messages. MPI relies on MPI\_Send and MPI\_Recv to well, send and receive messages.

|  |
| --- |
| /\*  If needed you might need to export the version specific path to the mpi and openmpi binaries, I think the updated installation on ubuntu fixes this but hard to know. That’s why we do labs!  export PATH=$PATH:/usr/mpich-3.4/bin  export PATH=$PATH:/usr/lib64/openmpi/bin  run with  [sri@localhost HPC]$ mpicxx Lab4MPI0.cpp -o lab4p2  [sri@localhost HPC]$ mpirun -np 4 ./lab4p2  \*/  // NOTE THIS IS A C Program not a C++ one, not that it matters much.  //the only difference is in how the printf works vs a cout  #include <mpi.h>  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  int main**(**int argc**,** char**\*\*** argv**)** **{**  // Initialize the MPI environment  MPI\_Init**(NULL,** **NULL);**  /\*  int rank, size;  MPI\_Init(&argc, &argv);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &size);  \*/  int world\_size**;**  MPI\_Comm\_size**(**MPI\_COMM\_WORLD**,** **&**world\_size**);**  // Get the rank of the process  int world\_rank**;**  MPI\_Comm\_rank**(**MPI\_COMM\_WORLD**,** **&**world\_rank**);**  /\* sri fixing some variable messes  rank = world\_rank;  size = world\_size;  \*/  int message**[**2**];**  int dest**,** src**;**  int tag**=**0**;**  MPI\_Status status**;**  **if** **(**world\_size **==**1**){**  printf**(**"requires more than one process"**);**  MPI\_Finalize**();**  **}**  **if** **(**world\_rank **!=**0**)**  **{**  message**[**0**]** **=** world\_rank**;**  message**[**1**]** **=** world\_size**;**  dest **=**0**;**  MPI\_Send**(**message**,** 2**,** MPI\_INT**,** dest**,** tag**,** MPI\_COMM\_WORLD**);**  **}**  **else**  **{**  **for** **(**src**=**1**;** src**<** world\_size**;** src**++)**  **{**  MPI\_Recv**(**message**,** 2**,** MPI\_INT**,** src**,** MPI\_ANY\_TAG**,** MPI\_COMM\_WORLD**,** **&**status**);**  printf**(**"Hello from Process %d of %d\n"**,** message**[**0**],** message**[**1**]);**  **}**  **}**  MPI\_Finalize**();**  **return** 0**;**  **}** |

That loops in order, only because it traverses the messages in order. But nothing actually guarantees the ordering (also note that it’s process 0 that’s printing off the other processes)

4. MPI reduction

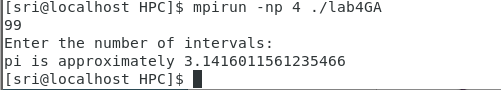
Note that there are many built in reduction operations (see the textbook table 8.2)

So, we want to gather all of the results and traverse them in order.   
  
Note that below is an application found both in the text and numerous locations on the web, it’s an example of a monte carlo method for approximating pie, and is based on:

<https://computing.llnl.gov/tutorials/mpi/#Exercise1>

|  |
| --- |
| #include <mpi.h>  int main **(** int argc**,** char **\*\***argv **)**  **{**  int n**,** i**,** pool\_size**,** my\_rank**;**  double mypi**,** pi**,** h**,** sum**,** x**,** a**;**  MPI\_Init**(NULL,** **NULL);**  /\* MPI\_Init(&argc, &argv);  MPI\_Comm\_size(MPI\_COMM\_WORLD, &pool\_size);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &my\_rank);  \*/  // int pool\_size;  MPI\_Comm\_size**(**MPI\_COMM\_WORLD**,** **&**pool\_size**);**  // Get the rank of the process  // int my\_rank;  MPI\_Comm\_rank**(**MPI\_COMM\_WORLD**,** **&**my\_rank**);**  static const int ROOT **=** 0**;** //MPI::ROOT;  **if** **(**my\_rank **==** ROOT**)** **{**  printf**(**"Enter the number of intervals: "**);**  scanf**(**"%d"**,&**n**);**  **if** **(**n**==**0**)** n**=**100**;**  **}**  MPI\_Bcast**(&**n**,** 1**,** MPI\_INT**,** ROOT**,** MPI\_COMM\_WORLD**);**  h **=** 1.0 **/** **(**double**)** n**;**  sum **=** 0.0**;**  **for** **(**i **=** my\_rank **+** 1**;** i **<=** n**;** i **+=** pool\_size**)** **{**  x **=** h **\*** **((**double**)**i **-** 0.5**);**  sum **+=** 4.0 **/** **(**1.0 **+** x**\***x**);**  **}**  mypi **=** h **\*** sum**;**  MPI\_Reduce**(&**mypi**,** **&**pi**,** 1**,** MPI\_DOUBLE**,** MPI\_SUM**,** ROOT**,**  MPI\_COMM\_WORLD**);**  **if** **(**my\_rank **==** ROOT**)** printf**(**"\npi is approximately %.16f\n"**,** pi**);**  MPI\_Finalize**();**  **return** 0**;**  **}** |

mpicxx Lab4GatherAll.cpp -o lab4GA



Forcing ordering:

If you need (for some reason) to force the ordering on MPI it’s actually non trivial.

<https://stackoverflow.com/questions/17570996/mpi-printing-in-an-order>

Basically, what you do is send everything to a specific node, which then needs to buffer it and print in order.

Somewhat outside of scope for a lab.

Lab for today:

Given a 2D array:

int sample\_array[4][5] = {

{50, 55, 62, 70, 85},

{35, 42, 45, 47, 49},

{32, 33, 36, 37, 38},

{25, 30, 30, 35, 30},

}

Broadcast that to each node (obviously the more correct way to do this is to only send the part you want) every node.

Have each thread (4 of them) sum up the results and reduce the result back.

Here is some code to get you started (it generates a random array)

|  |
| --- |
| #include "mpi.h"  #include <stdio.h>  #include <stdlib.h>  // taken from https://stackoverflow.com/questions/18945129/mpi-broadcasting-2d-array  // as an interesting starting point for our problem.  int main**(**int argc**,** char**\*\*** argv**)**  **{**  int procNum**,** procRank**;**  int m**,**n**;**  int sumProc **=** 0**,** sumAll **=** 0**;**  int**\*\*** arr**;**  MPI\_Status status**;**  /\*  MPI\_Init ( &argc, &argv );  MPI\_Comm\_size ( MPI\_COMM\_WORLD, &procNum );  MPI\_Comm\_rank ( MPI\_COMM\_WORLD, &procRank );  \*/    MPI\_Init**(NULL,** **NULL);**  MPI\_Comm\_size**(**MPI\_COMM\_WORLD**,** **&**procNum**);**  MPI\_Comm\_rank**(**MPI\_COMM\_WORLD**,** **&**procRank**);**      **if** **(**procRank **==** 0**)**  **{**  printf**(**"Type the array size \n"**);**  scanf**(**"%i %i"**,** **&**m**,** **&**n**);**  **}**  MPI\_Bcast**(&**m**,** 1**,** MPI\_INT**,** 0**,** MPI\_COMM\_WORLD**);**  MPI\_Bcast**(&**n**,** 1**,** MPI\_INT**,** 0**,** MPI\_COMM\_WORLD**);**  // This bit creates the array    /\*    // this works but creates the array on the stack.  int sample\_array[4][5] = {  {50, 55, 62, 70, 85},  {35, 42, 45, 47, 49},  {32, 33, 36, 37, 38},  {25, 30, 30, 35, 30},    in CPP there are a couple of 'right' ish ways to do this:  the syntactically correct but slow system is:    int\*\* a = new int\*[rowCount];  for(int i = 0; i < rowCount; ++i)  a[i] = new int[colCount];  is workable syntax    But then you need to delete everything at the end in a good program    for(int i = 0; i < rowCount; ++i) {  delete [] a[i];  }  delete [] a;  Beyond the scope of today, a much faster way is to make one big memory blob -  int \*ary = new int[sizeX\*sizeY];  // ary[i][j] is then rewritten as  ary[i\*sizeY+j]    https://stackoverflow.com/questions/936687/how-do-i-declare-a-2d-array-in-c-using-new      }    \*/  arr **=** **new** int**\*[**m**];**  **for** **(**int i **=** 0**;** i **<** m**;** i**++)**  arr**[**i**]** **=** **new** int**[**n**];**  **if** **(**procRank **==** 0**)**  **{**  //HPC Lab 4 This generates a random array, you want to replace this with your own static array.  //  **for** **(**int i **=** 0**;** i **<** m**;** i**++)**  **{**  **for** **(**int j **=** 0**;** j **<** n**;** j**++)**  **{**  arr**[**i**][**j**]** **=** rand**()** **%** 30**;**  printf**(**"%i "**,** arr**[**i**][**j**]);**  **}**  printf**(**"\n"**);**  **}**  **}**  //MPI\_Bcast(&arr[0][0], m\*n, MPI\_INT, 0, MPI\_COMM\_WORLD);  // this bit broadcasts the array to each node  **for** **(**int i **=** 0**;** i **<** m**;** i**++)**  MPI\_Bcast**(**arr**[**i**],** n**,** MPI\_INT**,** 0**,** MPI\_COMM\_WORLD**);**    // you need two lines of code here to sum the array (rows and columns)    sumProc **+=** 1**;**  MPI\_Reduce**(&**sumProc**,&**sumAll**,**1**,**MPI\_INT**,**MPI\_SUM**,**0**,**MPI\_COMM\_WORLD**);**      **if** **(**procRank **==** 0**)**  **{**  printf**(**"sumAll = %i \n"**,** sumAll**);**  **}**  **delete** **\***arr**;**  MPI\_Finalize**();**  **return** 0**;**  **}** |