HPSC Lab2

Luna McBride, Mehmood Ali

September 13 2023

1 Parallel Design

This assignment utilizes multiple cores to build out a grid. The objective is to get us thinking about how parallel pieces should connect to take advantage of the strengths present in parallel computing. As a result, this task functions more as a "change your way of thinking" assignment rather than a difficult coding assignment like those present in other classes. Nonetheless, parallelizing a grid like this turned out a lot harder than it initially seemed.

In this lab, we need to implement a finite difference solver for Laplace's equation on a grid that is decomposed among multiple processors using MPI for parallelism. The linear system is solved iteratively, and the code handles boundary conditions and communication between processors effectively.

- **fd.cpp**: This is the main program file that contains the main function. It initializes MPI, performs grid decomposition, forms a linear system, solves it using either Gauss-Seidel or Jacobi iteration, and plots the results. It also includes a class called LaplacianOnGrid.
- **fd.h**: This header file contains various include statements, macros, and typedefs used throughout the code. It's included in both **fd.cpp** and **gauss_seidel.h**.
- **gauss_seidel.h**: This header file defines a function **GS_or_Jacobi** that performs Gauss-Seidel or Jacobi iterations to solve a linear system. It's used in **fd.cpp** for solving the linear system.
- **mpiinfo.h**: This header file defines the **mpiInfo** class, which contains information about the MPI environment, such as the processor's rank, number of processors, grid decomposition, and functions for exchanging boundary information between processors.

In this lab we use a parallel approach for tackling the Laplace equation through a finite difference method using MPI (Message Passing Interface). It begins by dividing the problem domain into a grid, with each grid point representing a variable to be solved for. This 2D grid is then distributed across multiple MPI processes, creating a structured processor grid, where each process is assigned a unique identifier (myPE) and a position in the processor grid (iPE, jPE). Communication plays a pivotal role, as each processor needs to exchange boundary values with its neighboring processors to facilitate the solution of the Laplace equation. These neighbors are determined by the processor's position within the grid. The communication itself is handled through MPI point-to-point communication routines. The formulation of the linear system is central to the code's functionality.

The **LaplacianOnGrid** class defines the grid and its associated variables, ultimately constructing the matrix A and the right-hand side vector b that constitute the linear system Ax = b, representing the Laplace equation. The code accommodates various boundary conditions, including Dirichlet boundary conditions, depending on the specific problem requirements. For the iterative solution phase, the code implements two iterative methods, namely **Gauss-Seidel** and **Jacobi**. These methods continue to iterate until a convergence criterion is met. Each iteration involves not only updating the interior grid points but also communicating boundary values with neighboring processors and verifying convergence against a specified tolerance or a maximum iteration count. Parallel communication, managed by the **ExchangeBoundaryInfo** function, is vital for synchronizing boundary values between neighboring processors. This function sends values from one processor to another and receives values in return. The received boundary values are then employed to update boundary nodes and apply relevant boundary conditions for the subsequent iteration.

The main parallelization strategy centers on domain decomposition. By dividing the computational domain into smaller subdomains, each assigned to a different MPI process. Processors exclusively focus on their designated portions of the grid and interact solely with neighboring processors to solve the problem.

2 Self Evaluation

2.1 Luna

I really had a problem with the loading and compiling more than anything else. I have included the ex01.bat slurm file I had to modify, but it was throwing errors with required library dependencies until I made the change to update and use gcc instead of intel like originally written. On top of that, I made an error in changing around nPEx and nPEy that I could not even see until libstdc++ was fixed and various loading attempts occurred. I really spent a lot more time fighting my computer than actually coding the mpiInfo. I am glad Mehmood was able to get that part done so quickly and efficiently in that time.

2.2 Mehmood

At first, I had trouble running my program on a single processor. It took me a while to figure out that I needed to make changes to the slurm file by setting nPEx and nPEy to 1. I also had to modify the task nodes. After some trial and error, I finally managed to run it successfully.

I also had some issues when trying to plot graphs with Gnuplot. I made a simple mistake where every time I ran the command "load pc," it asked for a file name. It took me a while to realize that I should use the command "load 'pc'" with single quotes around 'pc'.

Implementing boundary conditions and debugging the program was a bit challenging. Debugging took up a significant amount of time. I used multiple "cout" statements to check the program's outputs. When working on the boundary conditions, I learned that it was crucial to make sure that neighboring processors communicated accurately and on time. In the beginning, I got some strange plots due to communication errors between processors. I spent most of my time debugging for this lab. Due to the complex nature of the problem, we were solving, it took some time, but eventually, after watching the lecture multiple times, I was able to make it work.

3 Appendix A: Parallel Code

```
#include "fd.h"

void Exit()
{
  MPI_Barrier(MPI_COMM_WORLD);
  MPI_Finalize();
  exit(0);
}


//  ==
//  ||
//  ||   C L A S S:    m p i I n f o
//  ||
//  ==

class mpiInfo
{
 public:

  int myPE;
  int numPE;
  int nRealx,  nRealy;
  int nPEx, nPEy;
  int iPE , jPE;
  int iMin, iMax, jMin, jMax ; // The global i-j numbers on this processor
  int nei_n, nei_s, nei_e, nei_w;
  int countx, county;

  double *phiL, *phiR;
  double *phiT, *phiB;

  double *phiSend_n,  *phiSend_s;
  double *phiSend_e,  *phiSend_w;
  double *phiRecv_n,  *phiRecv_s, *phiRecv_e,  *phiRecv_w;

  MPI_Status  status;
  int         err;
  int         tag;
  MPI_Request request;

  //  -
  //  |
  //  |   GridDecomposition: Set up PE numbering system in figure below and
  //  |                      establish communication arrays.
  //  |
  //  |                      nPEx -- number of PEs in the x-direction
  //  |                      nPEy -- number of PEs in the y-direction
  //  |                      numPE = total number of PEs
  //  |
  //  |                       +-------+-------+ . . .   +-------+
  //  |                       |       |       |         |       |
  //  |                       |       |       |         | numPE |
  //  |                       |       |       |         |       |
  //  |                       +-------+-------+ . . .   +-------+
  //  |                       .       .       .         .       .
  //  |                       .       .       .         .       .
  //  |                       .       .       .         .       .
  //  |                       +-------+-------+ . . .   +-------+
  //  |                       |       |       |         |       |
  //  |                       | nPEx  | nPEx+1|         |       |
  //  |                       |       |       |         |       |
  //  |                       +-------+-------+ . . .   +-------+
  //  |                       |       |       |         |       |
  //  |                       |   0   |   1   |         | nPEx-1|
  //  |                       |       |       |         |       |
  //  |                       +-------+-------+ . . .   +-------+
  //  |
  //  |
  //  -


  void GridDecomposition(int _nPEx, int _nPEy, int nCellx , int nCelly)
  {

    nRealx = nCellx + 1;
    nRealy = nCelly + 1;

    // Store and check incoming processor counts

    nPEx = _nPEx;
    nPEy = _nPEy;

    if (nPEx*nPEy != numPE)
      {
      if ( myPE == 0 ) cout << "Fatal Error:  Number of PEs in x-y directions do
   not add up to numPE" << endl;
      MPI_Barrier(MPI_COMM_WORLD);
      MPI_Finalize();
      exit(0);
      }

    // Get the i-j location of this processor, given its number.  See figure above
    :

    jPE = int(myPE/nPEx);
    iPE = myPE - jPE*nPEx;

    // Set neighbor values

    nei_n = nei_s = nei_e = nei_w = -1;

     if ( iPE > 0      )
      {
   nei_w = myPE - 1;
      }
    if ( jPE > 0      )
      {
```

```
  nei_s = myPE-2;
     }
   if ( iPE < nPEx-1 )
     {
  nei_e = myPE+1;
     }
   if ( jPE < nPEy-1 )
     {
  nei_n=myPE+2;
     }


   countx = nRealx + 2;
   county = nRealy + 2;

   phiL = new double [ county ];
   phiR = new double [ county ];
   phiT = new double [ countx ];
   phiB = new double [ countx ];

   phiSend_n = new double [ countx ];
   phiSend_s = new double [ countx ];
   phiSend_e = new double [ county ];
   phiSend_w = new double [ county ];
   phiRecv_n = new double [ countx ];
   phiRecv_s = new double [ countx ];
   phiRecv_e = new double [ county ];
   phiRecv_w = new double [ county ];

   tag = 0;
 }

 void ExchangeBoundaryInfo(VD &Solution, VD &b)
 {
  sLOOP phiSend_n[s] = 0.;
  sLOOP phiSend_s[s] = 0.;
  tLOOP phiSend_e[t] = 0.;
  tLOOP phiSend_w[t] = 0.;

  // ----------------------------------------------
  // (1) Parallel communication on PE Boundaries:   ** See fd.h for tLOOP and
   sLOOP macros **
  // ----------------------------------------------

  // (1.1) Put them into communication arrays

  sLOOP phiSend_n[s] = Solution[pid(s, nRealy - 1)];
  sLOOP phiSend_s[s] = Solution[pid(s, 2)];
  tLOOP phiSend_e[t] = Solution[pid(nRealx - 1, t)];
  tLOOP phiSend_w[t] = Solution[pid(2, t)];

  // (1.2) Send them to neighboring PEs
  if ( nei_n >= 0 )  err = MPI_Isend(phiSend_n, nRealx + 2, MPI_DOUBLE, nei_n, 0,
    MPI_COMM_WORLD, &request);
  if ( nei_s >= 0 )  err = MPI_Isend(phiSend_s, nRealx + 2, MPI_DOUBLE, nei_s, 0,
    MPI_COMM_WORLD , &request);
  if ( nei_e >= 0 )  err = MPI_Isend(phiSend_e , nRealy + 2, MPI_DOUBLE , nei_e , 0,
    MPI_COMM_WORLD , &request);
  if ( nei_w >= 0 )  err = MPI_Isend(phiSend_w , nRealy + 2, MPI_DOUBLE , nei_w , 0,
    MPI_COMM_WORLD , &request);

  // (1.3) Receive values from neighobring PEs' physical boundaries.

  if ( nei_n >= 0 ) { err = MPI_Irecv(phiRecv_n , nRealx + 2, MPI_DOUBLE , nei_n ,
   0, MPI_COMM_WORLD , &request);   MPI_Wait(&request ,&status); }
  if ( nei_s >= 0 ) { err = MPI_Irecv(phiRecv_s , nRealx + 2, MPI_DOUBLE , nei_s ,
   0, MPI_COMM_WORLD , &request);   MPI_Wait(&request ,&status); }
  if ( nei_e >= 0 ) { err = MPI_Irecv(phiRecv_e , nRealy + 2, MPI_DOUBLE , nei_e ,
   0, MPI_COMM_WORLD , &request);   MPI_Wait(&request ,&status); }
  if ( nei_w >= 0 ) { err = MPI_Irecv(phiRecv_w , nRealy + 2, MPI_DOUBLE , nei_w ,
   0, MPI_COMM_WORLD , &request);   MPI_Wait(&request ,&status); }

  // (1.4) If new information was received, store it in the candy-coating values

  if ( nei_n >= 0 ) sLOOP Solution[pid(s, nRealy + 1)] = phiRecv_n[s] ;
  if ( nei_s >= 0 ) sLOOP Solution[pid(s, 0)] = phiRecv_s[s] ;
  if ( nei_e >= 0 ) tLOOP Solution[pid(nRealx + 1, t)] = phiRecv_e[t] ;
  if ( nei_w >= 0 ) tLOOP Solution[pid(0, t)] = phiRecv_w[t] ;

  // (1.5) Apply exchanged information as BCs

  if ( nei_n >= 0 ) sLOOP b[pid(s, nRealy + 1)] = phiRecv_n[s] ;
  if ( nei_s >= 0 ) sLOOP b[pid(s, 0)] = phiRecv_s[s] ;
  if ( nei_e >= 0 ) tLOOP b[pid(nRealx + 1, t)] = phiRecv_e[t] ;
  if ( nei_w >= 0 ) tLOOP b[pid(0, t)] = phiRecv_w[t] ;
 };


 int pid(int i,int j) { return (i+1) + (j)*(nRealx+2); }

};
```

Listing 1: mpiInfo.h

I also had to update my slurm file to get it to go. Unless I had it going with GCC and have it update, I would run into errors with libstdc++, and thus the whole thing would not load at all. Between this is and the error with my nPEx and nPEy, the files really fought back against me.

```
#!/bin/bash

# -
# |
# | This is a batch script for running a MPI parallel job on Summit
# |
# | (o) To submit this job, enter:  sbatch --export=CODE='/home/trmc7708/Lab2/
    fd_mpi/src' ex01.bat
# |
# | (o) To check the status of this job, enter: squeue -u <username>
# |
# -

# -
# |
# | Part 1: Directives
# |
# -

#SBATCH --nodes=1
#SBATCH --ntasks=4
#SBATCH --time=00:01:00
#SBATCH --partition=atesting
#SBATCH --output=ex01-%j.out

# -
# |
# | Part 2: Loading software
# |
# -

module purge
module update gcc
module load gcc
module load impi

# -
# |
# | Part 3: User scripting
# |
# -

echo "=="
echo "||"
echo "|| Begin Execution of fd in slurm batch script."
echo "||"
echo "=="

export OMPI_MCA_opal_common_ucx_opal_mem_hooks=1

srun -n 4 ./fd -nPEx 2 -nPEy 2 -nCellx 5 -nCelly 5  > tty.out

echo "=="
echo "||"
echo "|| Execution of fd in slurm batch script complete."
echo "||"
echo "=="
```

Listing 2: The Slurm File

4 Appendix B: Output

4.1 Output of the Slurm

```
myPE: 1 x0 = 1
myPE: 1 x1 = 2
myPE: 1 y0 = 0
myPE: 1 y1 = 1
```

```
myPE: 2 x0 = 0
myPE: 2 x1 = 1
myPE: 2 y0 = 1
myPE: 2 y1 = 2
myPE: 3 x0 = 1
myPE: 3 x1 = 2
myPE: 3 y0 = 1
myPE: 3 y1 = 2

---------------------------------------------

 F I N I T E   D I F F E R E N C E
 D E M O   C O D E

 Running on 4 processors

---------------------------------------------


Input Summary:
---------------------------
No. PE   in  x-direction: 2
             y-direction: 2
No. Cells in x-direction: 5
             y-direction: 5

myPE: 0 x0 = 0
myPE: 0 x1 = 1
myPE: 0 y0 = 0
myPE: 0 y1 = 1

Form Linear System:
------------------------------------------


Solve Linear System:
------------------------------------------

Jacobi/GS converged in 57 iterations

Plot Results:
------------------------------------------
```

Listing 3: tty.out

4.2 Outputs from Alpine

Before switching modules to update gcc and openmpi rather than intel

```
==
||
|| Begin Execution of fd in slurm batch script.
||
==
/home/trmc7708/Lab2/fd_mpi/src/./fd: /usr/lib64/libstdc++.so.6: version ‘GLIBCXX_3
    .4.29’ not found (required by /home/trmc7708/Lab2/fd_mpi/src/./fd)
```

```
/home/trmc7708/Lab2/fd_mpi/src/./fd: /usr/lib64/libstdc++.so.6: version ‘GLIBCXX_3
    .4.26’ not found (required by /home/trmc7708/Lab2/fd_mpi/src/./fd)
srun: error: c3cpu-a5-u34-3: task 0: Exited with exit code 1
==
||
|| Execution of fd in slurm batch script complete.
||
==
```

Listing 4: Output showing un-updated GCC warnings

After switching:

```
Lmod has detected the following error: These module(s) or extension(s) exist
but cannot be loaded as requested: "impi"
   Try: "module spider impi" to see how to load the module(s).



==
||
|| Begin Execution of fd in slurm batch script.
||
==
--------------------------------------------------------------------------
WARNING: There was an error initializing an OpenFabrics device.

  Local host:   c3cpu-a5-u34-3
  Local device: mlx5_0
--------------------------------------------------------------------------
--------------------------------------------------------------------------
WARNING: There was an error initializing an OpenFabrics device.

  Local host:   c3cpu-a5-u34-3
  Local device: mlx5_0
--------------------------------------------------------------------------
--------------------------------------------------------------------------
WARNING: There was an error initializing an OpenFabrics device.

  Local host:   c3cpu-a5-u34-3
  Local device: mlx5_0
--------------------------------------------------------------------------
--------------------------------------------------------------------------
WARNING: There was an error initializing an OpenFabrics device.

  Local host:   c3cpu-a5-u34-3
  Local device: mlx5_0
--------------------------------------------------------------------------
==
||
|| Execution of fd in slurm batch script complete.
||
==
```

Listing 5: Output showing Warnings

4.3 Plot Images, showing interconnectivity

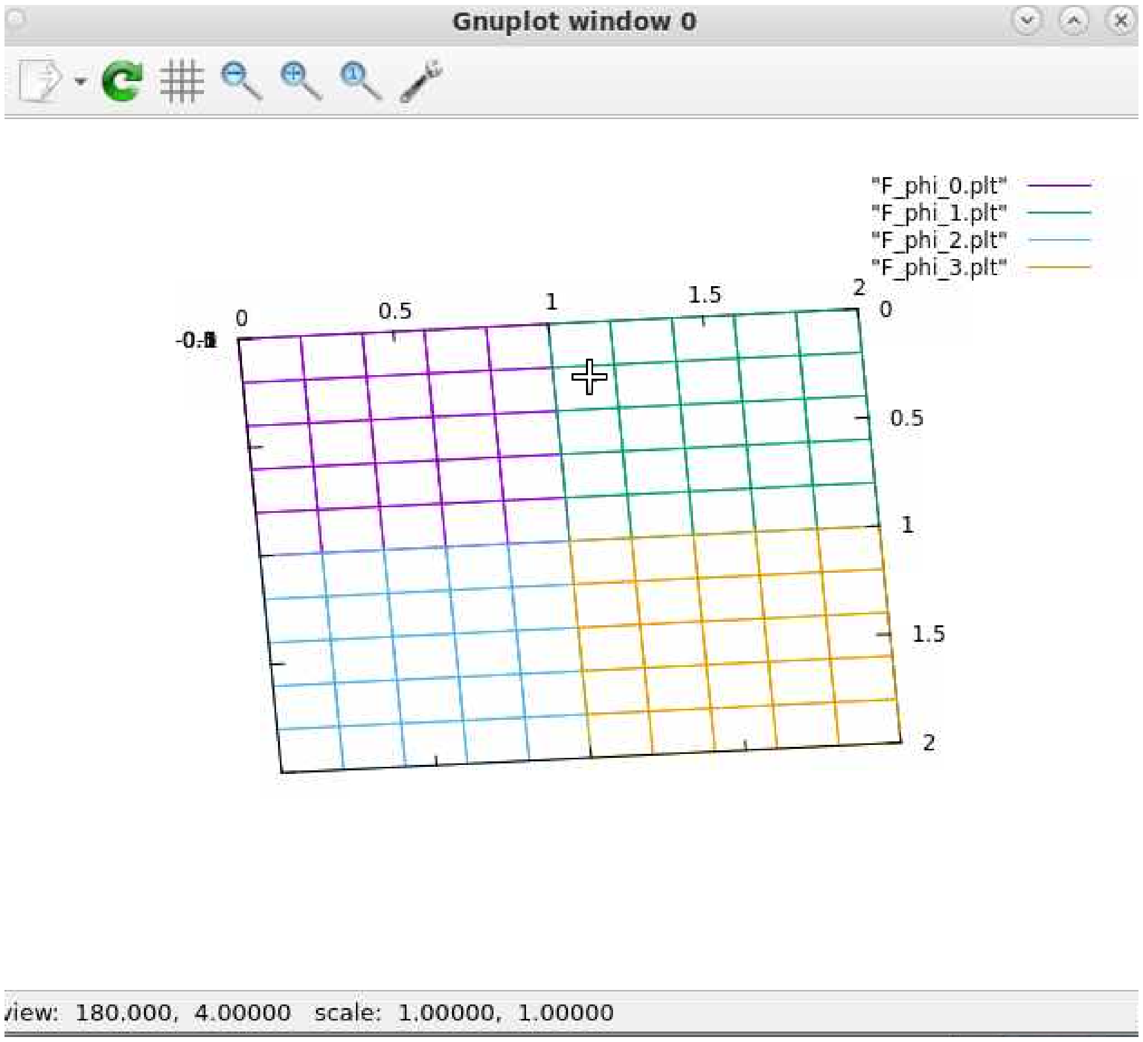


Figure 1: Top Down Image

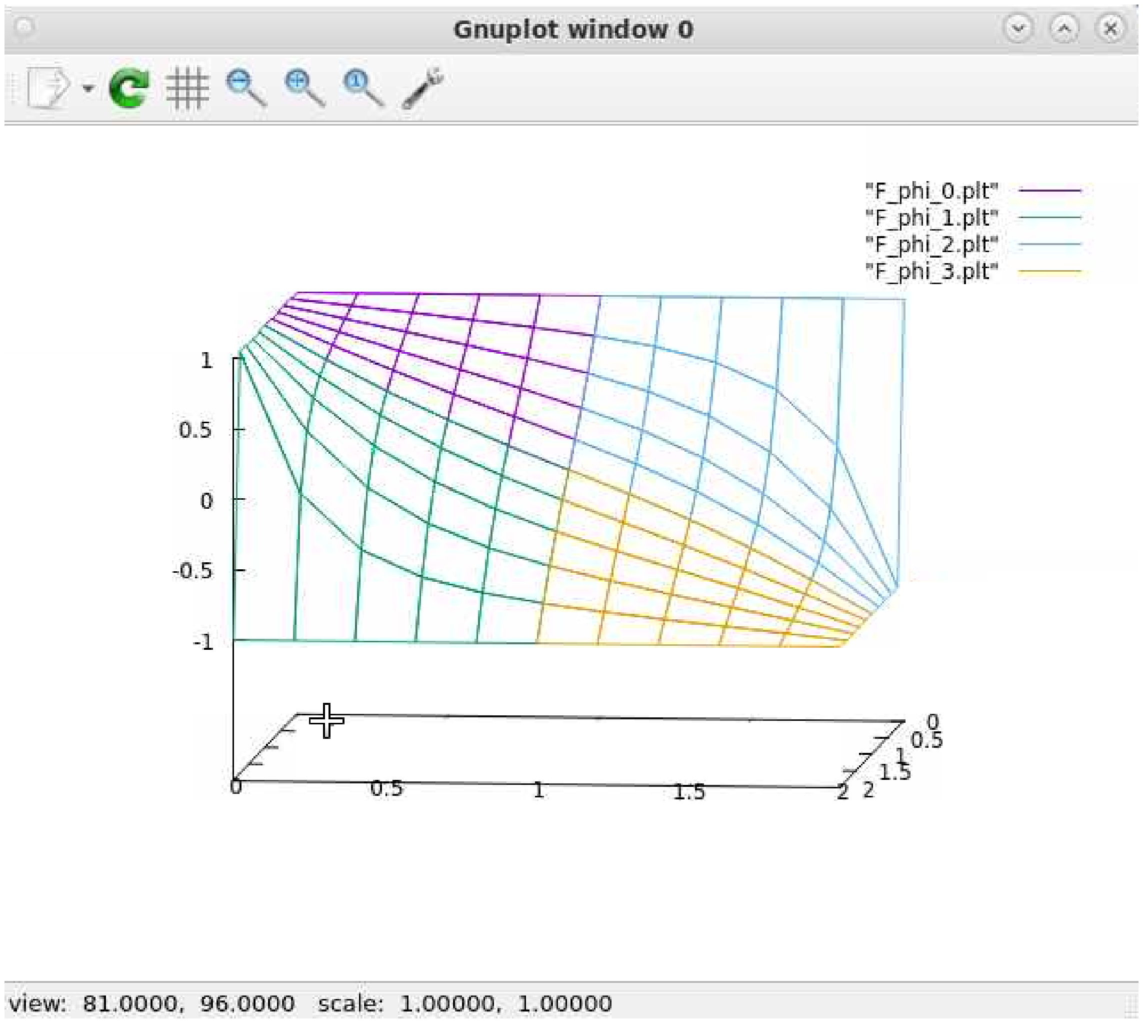


Figure 2: Side Image

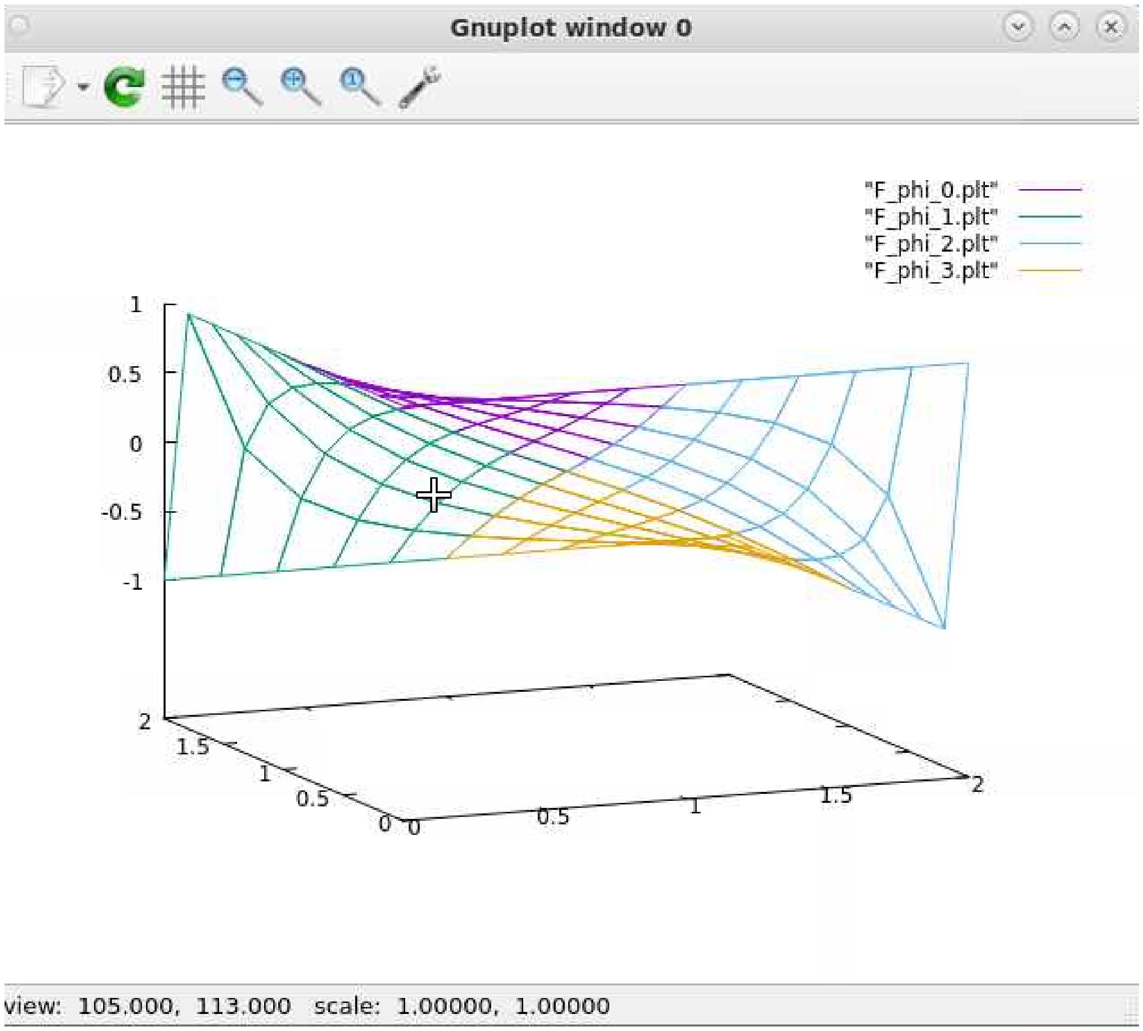
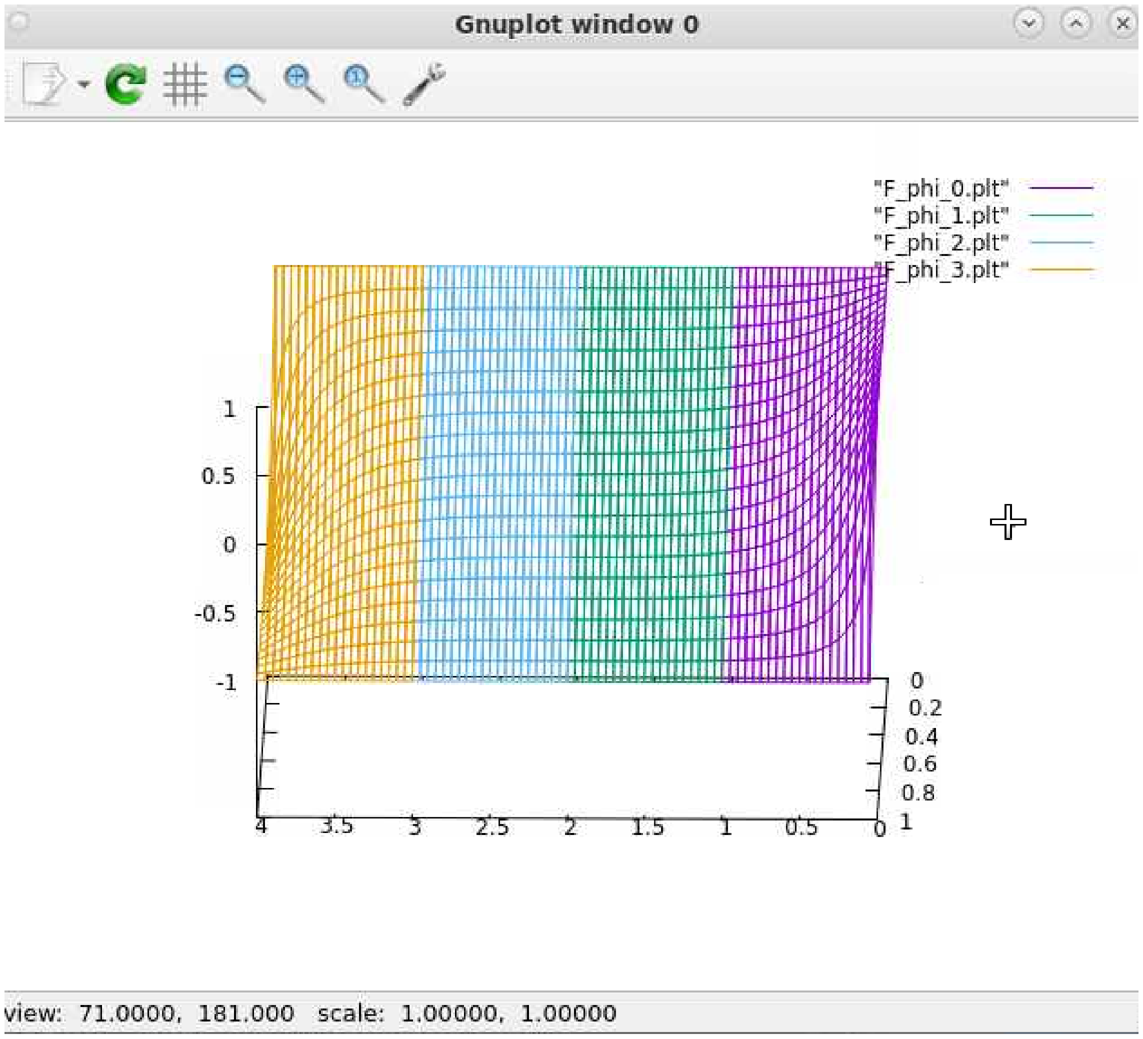


Figure 3: This one just looked cool



view: 71.0000, 181.000 scale: 1.00000, 1.00000

Figure 4: Challenge: nPEx 4 nPEy 1 with 20 cells each