Assignment 5

Using Paraguin to Create Parallel Programs

C. Ferner andB. Wilkinson
October 15, 2014

Overview

The goal of this assignment is to use the Paraguin compiler to create parallel solutions using MPI to run on a distributed-memory system.

Hello World (22%)

Create a "hello world" program using the code in Figure 1. Make sure you include the empty lines (those with only a semicolon).

```
#include <paraguin.h>
#include <stdio.h>

int __guin_rank = 0;

int main(int argc, char *argv[])
{
      char hostname[256];

      printf("Master process %d starting.\n", __guin_rank);

      #pragma paraguin begin_parallel

      gethostname(hostname, 255);
      printf("Hello world from process %3d on machine %s.\n",
                                        __guin_rank, hostname);

      #pragma paraguin end_parallel

      printf("Goodbye world from process %d.\n",
                                                  __guin_rank);

      return 0;
}
```

Figure 1: Hello World Program

Compiling

Open a browser and go to the website: http://babbage.cis.uncw.edu/~cferner/compoptions.html. Click the "browse" button, navigate to your hello world program, and submit it for compilation (leave the options alone). Download the zip file. You should also click the "Clean up my files now" button because each time you upload a file to compile, the zip file will contain all of the files that are in your space.

The zip file will contain the resulting source file with MPI commands and a text file with the compiler warnings and errors. Unzip the file.

First inspect the text file for any compilation errors. If there are any errors, you will need to fix the program, and try again.

Assuming that the program compiled successfully, look in the ".out.c" file to see how the compiler implemented your program. You will compile and run your program on the Ubuntu virtual machine. You will first need to compile that using **mpicc**:

```
mpicc hello.out.c –o hello.out
```

Assuming this also compiles, you can run it with:

```
mpirun –np # hello.out
```

or:

```
mpiexec –n # hello.out
```

Take a screenshot showing that you compiled the program with **mpicc**, ran the program, and the output to include in your submission document.

What to submit

Your submission document should include the following:

1) Your name and school;
2) Whether you are a graduate or undergraduate student;
3) A copy of the hello world source program (hello.c ***not*** hello.out.c);
4) Screenshot from compiling your hello world program with the Paraguin submission page;
5) Screenshot of running the program.

Matrix Multiplication (UG – 39% ; G – 35%)

Task 1 – Create the program

In this part of the assignment, you will be implementing the matrix multiplication algorithm using the Scatter/Gather pattern. You can use the matrix addition program from the lecture slides ("Examples" from October 2nd) as a template. What you will need to do differently is:

1. You will need to broadcast the *b* matrix. Because each processor needs to multiply the rows of *a* by the columns of *b*, it will need the entire b matrix.
2. Change the main computation loop to calculate matrix multiplication instead of matrix addition.

Test this program on your Ubuntu computer. You should probably start with inputs that are only 5x5. After you can verify that it works correctly on 5x5, increase it to 512x512 using the input file **input2x512x512Doubles** and redirecting the output to a file, such as:

```
mpirun -np # hello.out > output
```

Run the **diff** command on this file and the file **output512x512mult** to make sure your program produces the correct answers. Take a screenshot of compiling the program, running it on the 512x512 inputs, and the results of the **diff** command.

Task 2 – Run the parallel version

Now that your program works correctly on your own computer, upload the ".out.c" file to one of the clusters (UNCW or UNCC). Remember that the UNCW cluster requires a job description file, or else it will not truly run in parallel. You will also need to upload the files:

- **input2x512x512Doubles**
- **output512x512mult**

Compile and run your matrix multiplication program on the cluster. Run the program using the number of processors in the range: 1, 4, 8, 12, 16, 20, 24, 28, and 32. Rename the output files to **`output.mult.1, output.mult.4, output.mult.8`**, etc. After each run, execute the command:

```
diff output512x512mult output.mult.<P>
```

where **`<P>`** is the number of processors used.

If you are unable to get it to run on many processors, do the best you can. In other words, run it on the most number of processors you can.

Create graph of the execution times compared with sequential execution and the speedup curve with linear speedup. Make sure that you provide axes labels, a legend (of there is more than one line) and a title to the graphs. Include copies of the graphs in your submission document.

What to submit

Your submission document should include the following:

1) A copy of your matrix multiplication source program;
2) A copy of the job submission file (if you used the UNCW cluster)
3) Screenshot of compiling your program with **mpicc**;
4) Screenshot of running the parallel version of the program (**`mpirun`** or **`qsub`** and **`qstat`**);
5) Screenshot of running the **`diff`** command on the output files;
6) And copies of your graphs.

Heat Distribution (UG – 39% ; G – 35%)

In this part of the assignment you will be using a stencil pattern to model the heat distribution of a room with a fireplace. Although a room is 3 dimensional, we will be simulating the room with 2 dimensions. The room is 10 feet wide and 10 feet long with a fireplace along one wall as depicted in Figure 2. The fireplace is 4 feet wide and is centered along one wall (it takes up 40% of the wall, with 30% of the walls on either side). The fireplace emits 100º C of heat (although in reality a fire is much hotter). The walls are considered to be 20º C. The boundary values (the fireplace and the walls) are considered to be fixed temperatures.

Using a Jacobi Iteration, the heat distributed is calculated to look something like that of Figure 3. Each value is computed as an average of its neighbors. There needs to be two copies of the matrix. The newly computed values need to be stored into the second matrix; otherwise the values being computed would not be based on the same values in the previous iteration. Once all the new values are computed, the newly computed values can replace the values in the last iteration.

Create a program to use the Paraguin stencil pattern to compute the heat distribution of the room described above. The data should be a 3 dimensional array, where the size of the 1st dimension is 2, and the sizes of the 2nd and 3rd dimensions is how ever many points you decide to use. The number of points for the room should be large (more than 100x100). The more points there are, the smoother the simulation of the spread of heat.

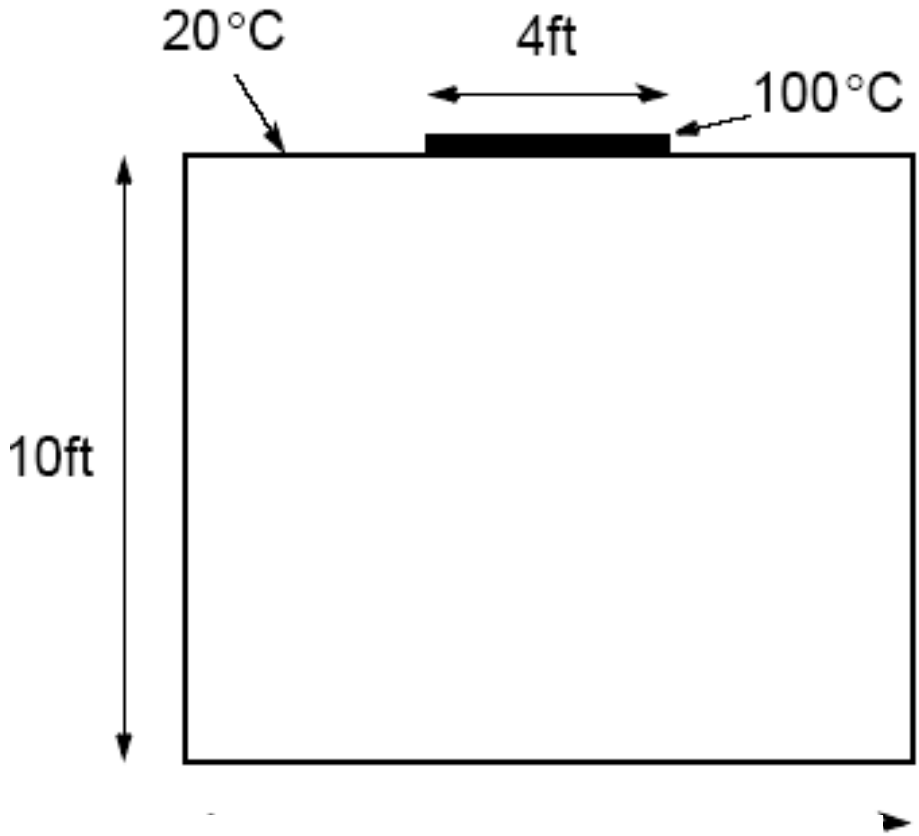

Figure 2: 10x10 Room with a Fireplace

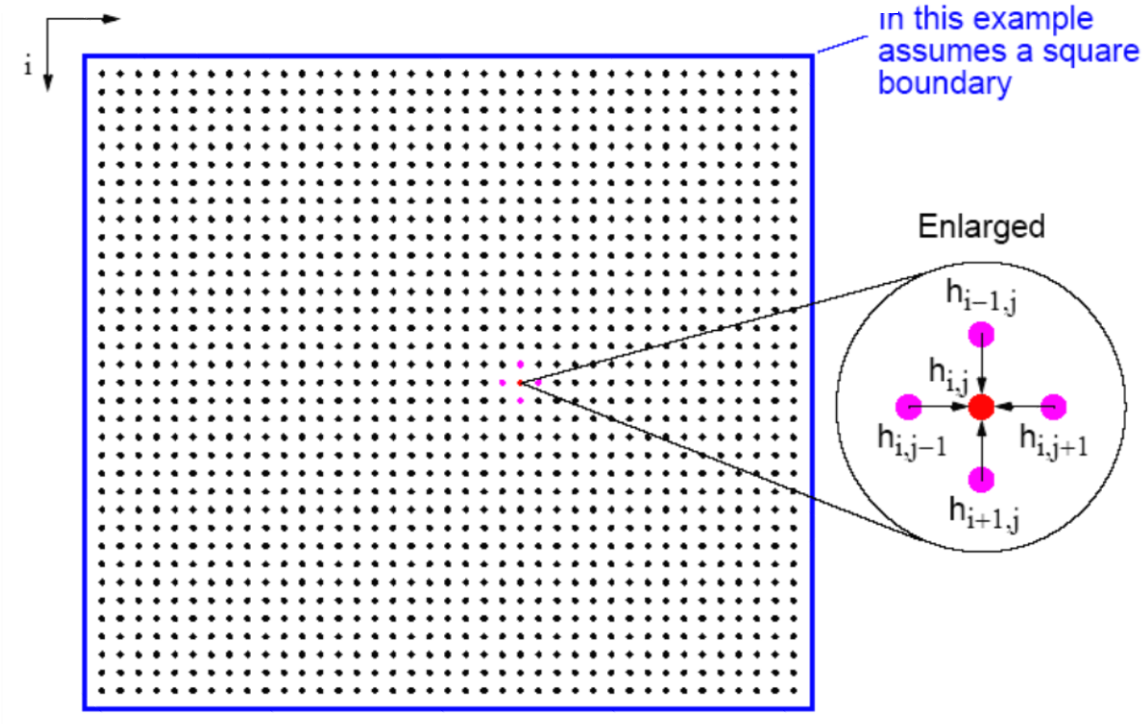

Figure 3: Jacobi Iteration

1) Initialize the values in both copies of the room all zeros (degrees Celsius) except for 40% of one wall centered where the values will be 100.
2) Using your matrix program as an example, set up a barrier and take a time stamp.
3) Use the Paraguin stencil **pragma** to indicate a stencil pattern
    a. The number of iterations should be at least 5000 in order for the data to converge. You can try larger and smaller values for the iterations to see if the data changes or not. You would like the number of iterations to be just large enough that the computed values do not change by significant sizes.
    b. The compute function should be a function to calculate the average of the value at location *i* and *j* and its nearest neighbors.
    c. Take another time stamp and calculate and report the elapsed time.
4) Print the final values of the matrix.

As with the matrix program, compile your program with the Paraguin submission page. Download the zip file and compile and run on your virtual machine. After you have successfully run the program, upload the ".out.c" file to one of the clusters and run it there. Try to run it on as many processors as you can.

Also create a graph of the final data. Figure 4 and Figure 5 show two examples of graphing the final data. The first was created using X11 (which you can only do using the UNCC cluster due to the job scheduler at UNCW), and the second is a surface graph using Microsoft Excel. The Excel graph is a *surface* graph (which was rotated so that you are looking at the wrong side of the fireplace).

What to submit

Your submission document should include the following:

1) A copy of the fireplace source program;
2) Screenshot from compiling your fireplace program with **mpicc**;
3) A copy of your job submission file (for UNCW only);
4) Screenshot of running the program;
5) Screenshot of the results from running your program;
6) And copies of your graphs.

Figure 4: Graph of Heat Distribution from X11

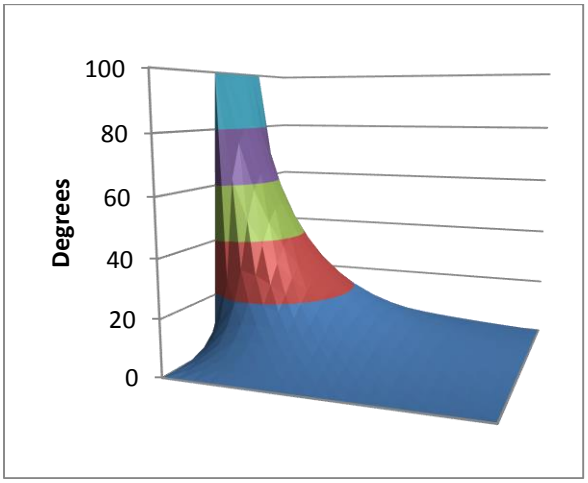

Figure 5: Graph of Heat Distribution from Excel

(Required for Graduate Students; Extra Credit (8 points) for Undergraduates

Monte Carlo Estimation of π (UG – 8% extra credit; G – 8%)

Using the sequential program to estimate π using the Monte Carlo algorithm, create a program that can be parallelized using the Paraguin compiler. To do this, you will need to:

1) Execute a barrier and take a time stamp;
2) Broadcast the number of iterations processors should use;
3) Run the creation of points within the square and count the number of points within the semicircle;
4) Reduce (as a summation) the count back to the master;
5) Print the result, error, and elapsed time from the master. (The error can be computed as the absolute difference between the value you computed and the true value of π. You can use the value 3.1415926535 as the true value of π.

Compile and run on your virtual machine to test first, then upload the ".out.c" file to one of the clusters. Compile it there and run the parallel program on 1, 4, 8, 12, 16, 20, 24, 28, and 32 processors, or as many as you are able to. Record the results and create

graphs of the results. You should have a graph showing the error as a function of the number of processors, and you should have a graph showing the error as a function of the elapsed time.

What to submit

Your submission document should include the following:

7) A copy of the Monte Carlo source program;
8) Screenshot from compiling your Monte Carlo program with **mpicc**;
9) A copy of your job submission file (for the UNCW cluster only);
10) Screenshot of running the program;
11) Screenshot of the results from running your program;
12) And copies of your graphs.