Parallel Programming Project Notes

Oregon State University
Mike Bailey
mjb@cs.oregonstate.edu

Why Are These Notes Here?

These notes are here to:

1. Help you setup and run your projects
2. Help you get the data you collect in the right format for submission
3. Help you get a **better grade** by doing all of this correctly!

better grade!
better grade!
better grade!
better grade!
better grade!

Project Notes, I

• Feel free to run your projects on whatever systems you have access to.

• If you don't have access to your own systems, then you can use what we have at OSU.
On-campus users will have access to Windows and Linux systems here.
Ecampus users will have remote access to our Linux systems, such as **flip** and **rabbit**.

• For the GPU and multiprocessor projects, you will have access to special systems here.

• Most of the projects will require timing to determine performance. Use the OpenMP timing functions. They give decent answers, and this will make the timing consistent across projects and across people. The OpenMP call:

double prec = omp_get_wtick( );

tells you the precision of the clock in seconds. I get 10^{-9} seconds on the systems I've been using. (I really doubt that this is true.) The OpenMP call:

double time0 = omp_get_wtime( );

samples the clock right now. It gives you wall-clock time in seconds. In parallel computing, memory latency and thread-idle time are part of the equation, so wall clock time is what you want.

How We Will Be Doing Timing

In this class, we don't want to just ***implement*** – we want to ***characterize*** performance. What speed-ups do we see, and why do we see them? How do we generalize that to other types of problems? *What insights does this give us?*

So, as part of your project assignments, you will be doing a lot of timing to determine program speed-ups.

```
#include <omp.h>

double time0 = omp_get_wtime( );                    // seconds
            . . .
double time1 = omp_get_wtime( );                    // seconds
fprintf( stderr, "Elapsed time = %10.2lf microseconds\n", 1000000. * ( time1 – time0 ) );
```

%10.2**lf** is the way to print doubles ("long float")

How Reliable is the Timing?

This way of timing measures ***wall-clock time***, which is really what we want to know in a parallel environment, *not CPU time*.

However, this puts you at the mercy of the other users on the system. If you are on one of our public systems (e.g., flip), I advise you to check the system load to see how much off your wall-clock time measurement will be due to the competition from other users. Use the Linux **uptime** command::

flip01 34% uptime
11:13:37 up 96 days, 11:52, 23 users, load average: 3.56, 3.08, 2.82

These three numbers represent total CPU load averages for the last 1, 5, and 15 minutes respectively. If the CPU load average is greater than the number of CPUs, then each CPU is over-burdened.

Clearly you want these numbers, especially the 1-minute one, to be as small as possible when you run your test. If they are "big", you might want to ssh to other systems (flip01, flip02, flip03, …) to see if you can find a better place to run or try again later.

How Reliable is the Timing? A Useful Trick!

I like to check the consistency of the timing by computing both peak speed and average speed and seeing how close they are:

```
double maxmflops = 0.;
double summflops = 0.;

for( int t = 0; t < NUMTRIES; t++ )
{
        double time0 = omp_get_wtime( );

        #pragma omp parallel for
        for( int i = 0; i < ARRAYSIZE; i++ )
        {
            C[ i ] = A[ i ] * B[ i ];
        }

        double time1 = omp_get_wtime( );
        double mflops = (double)ARRAYSIZE/(time1-time0)/1000000.;
        summflops += mflops;
        if( mflops > maxmflops )
            maxmflops = mflops;
}
printf( "    Peak Performance = %8.2lf MFLOPS\n", maxmflops );
printf( "Average Performance = %8.2lf MFLOPS\n", summflops/(double)NUMTRIES );
```

You should record the peak performance value. This gives you as close to the best answer that you will get. But, compare that with the average performance value. That will tell you how reliable that peak value is.

This is a *reliable* result:
Peak Performance = 1183.31 MFLOPS
Average Performance = 1141.41 MFLOPS

This is an *unreliable* result:
Peak Performance = 627.39 MFLOPS
Average Performance = 294.86 MFLOPS

Project Notes, II

On Linux, **Use *g++*.** The typical compile sequence for files that use OpenMP is:

g++ -o proj proj.cpp -lm -fopenmp

Note that OpenMP should always be included because we are using OpenMP calls for timing.

Note that the second character in the 3-character sequence "-lm" is an ell, i.e., a lower-case L. This is how you link in the **m**ath **l**ibrary.

Project Notes, III

• Most of these projects will require you to submit graphs. You can prepare the graphs any way you want, except for drawing them by hand. (The Excel Scatter-with-Smooth-Lines-and-Markers works well.) So that we can easily look at each other's graphs, please follow the convention that ***up is faster***. That is, do not plot seconds on the Y axis because then "up" would mean "slower". Instead, plot something like Speedup or MFLOPS or frames-per-second.

• I expect the graphs to show off your **scientific literacy** -- that is, I expect axes with numbers, labels, and units. If there are multiple curves on the same set of axes, I expect to be able to easily tell which curve goes with which quantity. After all, there is a reason this major is called Computer ***Science***. Not doing this makes your project unacceptable for grading.

You lose points if you don't do it this way.

Common Linux Commands

ls	List the files in this folder
ls -l	Make a detailed list of files in this folder
mv A B	Rename a file named *A* to a file named *B*
cp A B	Make a copy of file *A* and call it *B*
mkdir newdir	Make a new sub-folder called *newdir*
cd dir	Change the current folder to one underneath here called *dir*
cd ..	Change the current folder to the one just above here
pwd	Print the name of the folder you are in
rm A	Remove the file called *A*
g++ -o proj01 proj01.cpp -lm -fopenmp	Compile the program *proj01.cpp* into an executable called *proj01*
./proj01	Run the *proj01* executable

Diverting to a File

We all have a tendency to want to write our performance results out using *printf* (or *cout*) so that we can see them on the screen. That's fine. But, then we want to get those results into a file. You could mess with file I/O, or you could use a *divert* on the command line.

If you are currently running your program like this:

./proj01

and it prints to the **standard output** screen via *printf* or *cout*, then running it like this:

./proj01 > output.csv

will write your results into the file *output.csv*

(If you do it a second time, you will probably have to remove the previous *output.csv* first.)

You can also divert the *entire* output (**standard out *and* standard error**) of a looping script:

bash script.bash >& output

Standard Output and Standard Error

Printing from C/C++:

```
// %d is decimal integer, f = float, lf = long-float=double, \n skips to the next line:
printf( "i = %d, x = %f\n", i, x );                 // prints to standard output
fprintf( stderr, "i = %d, x = %f\n", i, x );        // prints to standard error
```

- What is the difference between printing to standard output versus standard error?
- There is no difference in where the messages appear. They both go to the console.
- But Standard Output is buffered, meaning that the system operates efficiently by writing the characters into an internal array and then eventually flushing that array to the console all at once.
- Standard Error is unbuffered, meaning that the system operates inefficiently by writing the characters one-at-a-time as they come out to the console.
- Why do we care about the difference? Because, if your program crashes, the standard output buffer crashes with it and you don't see the messages stuck in the buffer. That makes it hard to figure out what went wrong. However, you *do* see the standard error messages. For this reason, most of the example code in this class uses standard error.

Computing Performance

When computing performance, be sure that the **numerator is amount of work done** and the **denominator is the amount of time it took to do that work**. For example, if you are computing on a 2D grid and computing one value is the work done at each node, and you have NUMS*NUMS total nodes, then (NUMS*NUMS)/Δt is one good way to measure performance.

NUMS, NUMS*NUMS, 1./Δt, and NUMS/Δt are *not* good ways to measure performance as they don't reflect the true amount of **work done per time**.

If you are using ridiculously high values for NUMS, the quantity NUMS*NUMS might overflow a normal 32-bit int. You can use a long int, or just float each one separately. Instead of (float)(NUMS*NUMS)/dt, you could say (float)NUMS*(float)NUMS/Δt

If you are squaring a size number, and are using signed ints, the largest NUMS you can use is:

$$\sqrt{2,147,483,647} = 46,340$$

If you are squaring a size number, and are using unsigned ints, the largest NUMS you can use is:

$$\sqrt{4,294,967,295} = 65,535$$

Project Turn-in Procedures

13

Your project turnins will be done via *Canvas* and will consist of:

1. All source files (.cpp, .cu, .cl)
2. A report in PDF format.
3. No other files!

Submit these files separately. Don't zip or tar (etc…) anything!

Electronic submissions are due at 23:59:59 Pacific Time on the listed due date.

Your PDF report needs to include:

1. A title area on the first page: your name, email, project number, and project name
2. Tables and graphs as requested in the handout
3. An explanation of what you did and *why it worked the way it did*. Your submission will not be considered valid unless you at least attempt to explain why it works the way it does.

Your project will be graded and the score posted to Canvas.

If you did not get full credit, your grade will have an attached Canvas note telling you why.

Don't forget that title area! Because I concatenate all the class PDFs together for grading, lack of a name on your project PDF will make it difficult to know who to give this score to. It will be **5 points off** if you forget your name.

Computer Graphics

mjb – March 23, 2025

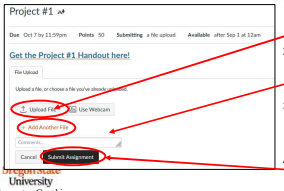
How to Turn In a Project on Canvas

14

On Canvas, the assignments will look like this.

Click this link to see the handout. I do it this way so that, if the handout needs to be clarified or corrected, I can do it quickly and instantaneously for all sections.

When you are ready to submit your files, click the **Start Assignment** button. That will bring up a dialog box that looks like this:



1. Click the **Upload File** button, browse to your first file (the order doesn't matter), and select the file.
2. Click **Add Another File**, click the new **Upload File** button, browse to the second file, and select the file.
3. Do this as many times as you need to (only submit your PDF and your code source files (.cpp, .cu, .cl).
4. When the files are all uploaded, click on **Submit Assignment.**

Oregon State University
Computer Graphics

mjb – March 23, 2025

Bonus Days

15

Projects are due at 23:59:59 on the listed due date, with the following exception:

Each of you has been granted **5** Bonus Days, which are no-questions-asked 24-hour project extensions which may be applied to any project, subject to the following rules:

1. If the project handout says "No Bonus Days", then no Bonus Days can be used on it. It is due exactly at 23:59:59 on the given due date, no later.
2. No more than **2** Bonus Days may be applied to *any one project*
3. Bonus Days cannot be applied to quizzes or tests

Bonus Days are not here to give you a 2-day vacation or to enable your procrastination! They are here to help you ***succeed*** even though you are sick, another class has a project due that same week, etc.

Bonus Days are way, way, way more valuable at the end of the term. You would be smart to hoard them until then.

Oregon State University
Computer Graphics

mjb – March 23, 2025

Bonus Days

16

To use one or more Bonus Days on a given project:

- You don't need to let me know.
- Turn-in promptness is measured ***by date***. So, after 23:59:59 on the posted due date, **it's late!**
- I will run a script to identify the projects that will have Bonus Days deducted.
- Keep track of your Bonus Days. But, if you lose track, email me and I will look it up in my master spreadsheet for you.
- If you have used Bonus Days on a given project, I will send you an email (well, a Python script will, actually) telling you how many you used and how many you have left.

Oregon State University
Computer Graphics

mjb – March 23, 2025

Silly Ways to Lose Points on Your Project

17

- You didn't put your name on the title page of the PDF report (-5)
- You submitted some other file type for your report other than a PDF (-5)
- You put files in a .zip or .tar (etc) file (-5)
- Anything else that causes me to have to go back and grade your project individually, unless it is because of something that was my fault (-5)

Oregon State University
Computer Graphics

mjb – March 23, 2025

A Warning About Virtual Machines

18

Virtual machines are, apparently, not automatically setup to do multithreading.

If you are running on your own virtual machine and are getting performance numbers that make absolutely no sense, try using one of the OSU machines.

Oregon State University
Computer Graphics

mjb – March 23, 2025

A Warning about Editing on Windows and Running on Linux

Some of you will end up having strange, unexplainable problems with your csh scripts or .cpp programs. This could be because you are typing your code in on Windows (using Notepad or Wordpad or Word) and then running it on Linux. Windows likes to insert an extra carriage return ('\r') at the end of each line, which Linux interprets as a garbage character.

You can confirm this by typing the Linux command:

od -c loop.csh

which will show you all the characters, even the '\r' (carriage returns, which you don't want) and the '\n' (newlines, which you do want).

To get rid of the carriage returns, enter the Linux command:

tr -d '\r' < loop.csh > loop1.csh

Then run loop1.csh

This works too:

sed -i -e 's/\r$//' loop.csh

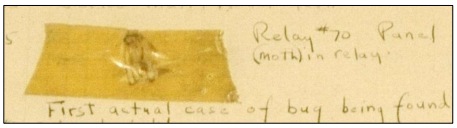
Or, on some systems, there is a utility called *dos2unix* which does this for you:

dos2unix < loop.csh > loop1.csh

Sorry about this. Unfortunately, this is a fact of life when you mix Windows and Linux.

Debugging in American History ☺

This is the bug that Dr. and Rear Admiral Grace Hopper found (and fixed) in the late 1940s in a relay panel in the Harvard Mark II computer. As you can see here, she dutifully recorded it in her log. This gave rise to the phrase "computer program bug" that we still use today. This is now part of the Smithsonian Institution American History collection.



https://americanhistory.si.edu/collections/nmah_334663