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!

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.

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 or flop), 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 test on other systems (flip01, flip02, flip03, …) to see if you can find a better place to run, or try again later.
**Project Notes, II**

If you are on Linux and have access to the Intel compiler, icpc, don’t use it unless we tell you to! (icpc is so good that it often does optimizations that undermine the very things you are testing.)

**Use g++.** The compilation sequences are:

On Linux, the typical compile sequence for files that use OpenMP is:

```
g++   -o   proj proj.cpp   -O3   -lm   -fopenmp
```

```
icpc -o   proj proj.cpp   -O3   -lm   -openmp -align  -qopt-report =3  -qopt-report-phase=vec
```

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

**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.

**Making Graphs**

In Excel, I have had the most success with creating tables that look like this:

<table>
<thead>
<tr>
<th># of subdivisions</th>
<th>MegaFunctionEvaluations Per Second</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

where the 1,2,4,8 rows are holding the number of threads constant, and the 512,1024,2048,4096 columns are holding the number of subdivisions constant. The cells are holding performance numbers, with higher numbers representing faster performance.

Sweep over the entire table, select Copy, and then insert it into one of the scatterplot options.

To transpose the sense of the graph (which you also need to do), right-click on the border of the graph and then click on “Switch Row/Column”.

You lose points if you don’t do it this way.
When we plot, we will all put execution speed on the Y axis (as opposed to putting elapsed time on the Y axis). Thus, as far as performance goes, up will mean “good.”

So, for example:

### Making Graphs

As you can tell, these performance measurements will be far more intelligible when examined as a graph than as raw numbers. Thus, you are expected to have access to a good automated graphing package. If you don’t have one, or can’t get access to one – go get one!

Hand-drawn graphs, whether analog or digital, will not be accepted for your assignments.

You will also need a word processor, with a way to import your tables and graphs, and with a way to turn that document into PDF.

### How to Run Benchmarks from Scripts

In our project assignments, you will run benchmarks, that is, you will try your application using several different combinations of parameters. Setting these combinations by hand inside your program one-by-one is a pain. It is much easier to do it from a script.

In most C and C++ compilers, there is some mechanism to set a `#define` from outside the program. Many of them use the `–D` construct on the command line:

```bash
#!/bin/bash
# number of threads:
for t in 1 2 4 6 8
    do
        echo NUMT = $t
# number of subdivisions:
    for s in 2 4 8 16 32 64 128 256 512 1024 2048 3072 4096
        do
            echo NUMS = $s
g++ -DNUMS=$s -DNUMT=$t prog.cpp -o prog -lm -fopenmp
        done
    done
```

Or, in `bash` (Bourne-again Shell) ...

```bash
#!/bin/bash
# number of threads:
for t in 1 2 4 6 8
do
    echo NUMT = $t
# number of subdivisions:
    for s in 2 4 8 16 32 64 128 256 512 1024 2048 3072 4096
        do
            echo NUMS = $s
g++ -DNUMS=$s -DNUMT=$t prog.cpp -o prog -lm -fopenmp
        done
done
```
import os
for t in [1, 2, 4, 6, 8]:
    print "NUMT = %d" % t
for s in [2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 3072, 4096]:
    print "NUMS = %d" % s
    cmd = "g++   -DNUMS=%d   -DNUMT=%d   prog.cpp  -o prog -lm  -fopenmp" % (s, t)
    os.system(cmd)
    cmd = "./prog"
    os.system(cmd)

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, in the Bezier surface example, computing one height is the work done at each node and you have NUMS*NUMS total nodes, so (NUMS*NUMS)*dt is one good way to measure performance.

NUMS, NUMS*NUMS, 1./dt, and NUMS/dt 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 one separately. Instead of

\[
\frac{\text{float}(\text{NUMS*NUMS})/\text{dt}}{
\text{float}(\text{NUMS}) \times \text{float}(\text{NUMS})/\text{dt}}
\]

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

- For signed int: 2,147,483,647
- For unsigned int: 4,294,967,295

Computing Performance

Project Turn-in Procedures

Your project turnins will all be electronic.

Your project turnins will be done at [http://engr.oregonstate.edu/teach](http://engr.oregonstate.edu/teach) and will consist of:
1. Source files of everything (.cpp, .cl, .cuda)
2. A Linux or Windows executable file, if needed.
3. A report in PDF format. You can zip everything else if you want, but please leave the PDF as a separate file.

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

Your PDF report will include:
1. A title area, including your name, email, project number, and project name.
2. Any tables or graphs to show your results.
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 the class web page.

If you want to know why you did not receive full credit, send me an email asking about it, or see me during Office Hours.

Bonus Days

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

Each of you has been granted five Bonus Days, which are no-questions-asked one-day project extensions which may be applied to any project, subject to the following rules:

1. No more than 2 Bonus Days may be applied to any one project
2. Bonus Days cannot be applied to tests
3. Bonus Days cannot be applied such that they extend a project due date past the start of Test #2.

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

- You don't need to let me know ahead of time.
- Turn-in promptness is measured by date. Don’t worry if teach tells you it’s late because it is between 23:30:01 and 23:59:59. But, after 23:59:59 on the posted due date, it’s late!
- Teach has been instructed to accept your turn-in, no matter when you do it.
- If you are on-campus, print out the Bonus Day PDF, fill it out and give it to me in class or in my office.
- If you are online, fill out the Bonus Day .docx form, turn it into a PDF, and email me the PDF.
- I will check your turn-in date and be sure it matches the number of Bonus Days you are redeeming.
- You can see how many Bonus Days you have left by looking in the BDL column of the grading table on the class web site.

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.
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 test this by typing the Linux command:
```
od -c loop.csh
```
which will show you all the characters, even the \r (which you don't want) and the \n (newline, 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

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.