Tools and Techniques of Computational Science - Fall 2015

Assignment 2

This homework is due by 11:59 PM on Friday 11/06/2015. Answers to this assignment must be submitted to Canvas. In this assignment, you need to submit both written responses and code. The written portion must be done in LaTeX but please submit the PDF only. Do not submit the TeX sources.

Exercises

Exercise 2.0 – Compiling a 3rd Party Library

This assignment is designed to get you familiar with building/installing an autotools-based 3rd party library following the example we did in class. You should carry out this exercise on *Stampede* and submit interactive jobs to support your compilation and testing efforts (recall that you can use the *idev* utility to aid in submitting an interactive job to the development queue).

Note: Do not perform these tests on the login nodes.

To begin, grab a copy of the GSL source tarball (version 1.16). For convenience, you can copy a pre-downloaded version from:

```
~csim/public/gsl-1.16.tar.gz
```

Untar this distribution and prepare for compiling 4 different configurations of GSL.

1. GNU gcc with no optimization
2. GNU gcc with -O3 optimization
3. Intel icc with no optimization
4. Intel icc with -O3 optimization

I recommend taking advantage of VPATH builds to support this effort. Consequently, consider making 4 directories corresponding to the configurations above. To complete this exercise, you need to fill in the Table below documenting the time it takes to compile each case serially ("make") versus the time it takes to compile in parallel ("make -j 16"). Note that since you are compiling on a compute node, using all the processors available is reasonable.

Configuration	Compilation Times		Regression Tests	
	Serial (secs)	Parallel (secs)	Passes	Failures
gcc/noOpt				
gcc/O3				
icc/noOpt				
icc/O3				

In addition to tracking the compile times for each case, run the regression test suite provided by GSL using the "make check" target". You will need to run these tests serially. For each case, document how many regression tests pass or fail in your table. Note that make will normally stop if it encounters an error. However, you can ask it to continue (so that you can discover all the regression failures, if you encounter one) using "make -k". In your write-up, be sure to include the exact configure line you used to setup each build.

Based on the results you obtained, answer the following additional questions?

- Which is faster to build - optimized Intel or GNU? How much faster is it?
- How much faster was a parallel build of non-optimized GNU versus its serial counterpart?
- If you encountered any regression failures, document specifically which directories the failures occurred in (and the associated compiler configuration).

Bonus question: If you did encounter a regression failure, can you identify the *addition* of a compiler flag that allows all regression tests to pass?

Exercise 2.1 – Simple Makefile

Using the example C src code files in:

```
~csim/public/buildsystems-example/standalone $
```

Create a simple Makefile to build an executable named "tardis". Your Makefile should include the following features:

- A "clean" target which removes all associated object files and executable name
- Convenience variables to provide system flexibility:
  - Variable that defines the desired executable name
  - Variable that defines the desired C compiler
  - Variable that defines the location of the desired GSL top-level path to link against based on an external environment variable (e.g. $TACC\_GSL\_DIR$)
  - Variable that defines the compiler flags to use

Exercise 2.2 – Revisiting numerical integration

Write a simple C, C++ or Fortran 90+ code that uses numerical integration to evaluate an analytic function. For the numerical integration, use a simple composite mid-point rule to perform the integration. For a fixed interval, the midpoint approximation is as follows:

$$\int_{x_{min}}^{x_{max}} f(x)\,dx \approx h\,f(x_{min} + \frac{h}{2}) \tag{1}$$

where $h = x_{max} - x_{min}$. Note that h will be a constant in this assignment. The analytic function to integrate is:

$$f(x) = cos(x) \tag{2}$$

Your code should accept at least 3 command-line arguments that are used to control the limits of integration and the discretization interval. It should also have a "-h" option to show how to run the code. If no command-line arguments are provided, your executable should show the help message and exit.

In addition to computing the numerical integration, your code should compute the error between your numerical solution and the exact analytic solution to the definite integral which is computed as:

$$\int_{x_{min}}^{x_{max}} cos(x)\,dx = sin(x_{max}) - sin(x_{min}) \tag{3}$$

Your code should approximate the definite integral and output the absolute error for a given NP, where NP is the number of discretization points between $[x_{min}, x_{max}]$ as:

$$Error = |Numerical - Exact| \tag{4}$$

Once your code is complete, run the following cases on $[x_{min}, x_{max}] = [0.0, 5.0]$ and time the runtime performance for each case:

- NP=10
- NP=50
- NP=100
- NP=200

Create two plots based on the results. The first plot should compare the wallclock runtime for the 4 cases in this problem as well as the 4 cases from the bash script implementation. The second plot should show the error as a function of NP (on a logscale please).

Exercise 2.3 – Sterling's approximation

Write a computer program to compute the factorial of every integer between 1 and 100, and compare the results to that given by a simple form of Sterling's approximation.

$$ln(N!) = Nln(N) - N \tag{5}$$

Plot the difference and estimate the point at which Sterling's approximation will be within 0.1% of the exact result. For this exercise you must use a compiled language (either C, C++ or F $\geq$ 90)

Exercise 2.4 – More one-liners

At this point in the class, we have covered many different unix commands, all of which can be used to construct both unix "one-liners" and be used in shell scripting. For this exercise, construct at least 5 different problems for you to solve via a "one-liner" and use the below (or other) unix commands to construct the answer. Below are 45 commands that can be used to construct the "one-liners". Bonus points will be awarded for using multiple commands in one "one-liner" as well as for creative use of the below commands. However, you are not limited to using the commands below.

- at
- awk
- cal
- cat
- comm
- convert
- curl
- cut
- date
- diff
- dos2unix
- echo
- expand
- fc
- fmt
- fold
- grep
- head
- less
- lpr
- locate
- lsof
- mail
- mc
- mkdir
- more
- nice
- paste
- ps
- pstree
- renice
- rsync
- sed
- sort
- strings
- tail
- tailf
- tac
- tar
- top
- uniq
- watch
- wc
- wget
- xargs