October 31, 2013

Tools and Techniques of Computational Science - Fall 2013

Project # 1

The task for this project is to design and implement a well-written, organized, and verified scientific application which numerically approximates the solution to the steady-state heat equation with a constant coefficient. In two dimensions, this is given by:

$$-k\nabla^2 T(x,y) = q(x,y) \tag{1}$$

You are tasked with approximating the heat equation assuming Dirichlet boundary conditions using 2nd- and 4th-order finite-difference approximations for the second derivative. Your application must support solutions in both one and two dimensions. For this project, your application will use simple iterative methods to solve the resulting linear system that is formed with the numerical approximation to Eqn. 1. In particular, your application must support solution mechanisms using either Jacobi or Gauss-Seidel.

You are free to implement your application in either C, C++, or F90. Some items for consideration and requirements when coding are summarized below:

- adopt a modular coding style - we do not want to see one big monolithic main function. Instead, organize your code into manageable chunks with functions/subroutines tasked with focused units of work. Your main driver routine should be a fairly short function.
- include comments in your code
- use dynamic memory allocations for variables that derive sizing information from runtime input options
- use SVN frequently: begin by creating a directory named "proj01" in your repository and commit often. Do not wait till you are done to commit everything.
- provide a mechanism to have multiple levels of output. At a minimum, we want to see two modes: standard and debug. The debug mode will include more verbose (debug) information when chosen.
- include performance timing measurements for key kernels within the code (eg. the solution of the linear system, initialization, etc).

The following sections highlight elements that are required for your implementation.

Input Options

Your application must derive necessary runtime options from an input file. You can specify the name of the input file as a command-line argument, or you can assume that your application will always read from the same filename. If you choose the latter, assume an input file named "input.dat". The exact format of the input file is up to you, and you may use the GRVY library installed on Stampede if you want to simplify the process (recall that API documentation is available online).

Regardless of the format, you need to support a minimum number of runtime options including the following:

- choice between 1D or 2D analysis
- choice between Jacobi or Gauss solvers
- choice between 2nd or 4th-order finite differences
- mesh sizing options
- option to run in a verification mode
- option to control standard output mode (e.g. standard vs debug)

Build System

Your application must include an autotools-based build system which works on Stampede. This build system must have configuration options to link in all 3rd party libraries that your application requires. It must also have an option to run a suite of regression tests via the "make check" target.

Since we will be using the MASA library to help aid the verification process, your application will require linkage to at least one 3rd party library. Note that we will build MASA for class use and install it on Stampede.

To help aid in this effort, we have included example autoconf macros in the shared/autotools-macros/ directory within the class SVN repository. To enable in autoconf, you can add lines similar to the following to your configure.ac file. These macros request a minimum version for the libraries and indicate whether the library linkage is optional or not.

```
AX_PATH_MASA([0.30],[yes])
AX_PATH_GRVY([0.32],[yes])
```

If successful, these macros will set relevant automake variables you can use within your build system (e.g. $(MASA_LIBS), $(MASA_CXXFLAGS), $(GRVY_LIBS), $(GRVY_CFLAGS), etc).

Code Verification

To help verify your application, we will use a manufactured solution for the steady-state heat-equation available within the MASA library. To implement, you will call routines within the MASA library to derive the right-hand side source term for Eqn. 1 at discrete mesh points. Note that the relevant solution in MASA for 2D is referenced via the name "heateq_2d_steady_const". You will also use the library to evaluate the analytic temperature solution (necessary to provide closure for your boundary conditions) and to compare your resulting numerical results to the true solution.

Provide a "verification" mode option in your input file which will enable your code to call MASA and compute a discrete l_2 norm of the difference between your numerical solution and the analytic solution. Your code should output this error norm within the standard output when running in verification mode.

Model Document

You will use the module document you completed for HW #3 as a starting point for this effort. In addition to all of the requirements from HW #3, include additional sections highlighting the following (along with any other details you find pertinent to thoroughly document your application and usage).

- Build Procedures: describe how to build your code (on Stampede)

- Input Options: describe the various input options relevant for your code
- Verification Procedures: describe how to run your code in verification mode. Provide an example of the stdout when running your code in verification mode.
- Verification Exercise: using your fancy verification mode, include results from performing a uniform mesh refinement study with your code using 2nd-order and 4th-order finite differences (in both 1D and 2D). Include two well-labeled, visually-striking plots in your model document (one for 1D, one for 2D) comparing the resulting error norms from the 2nd and 4th order uniform refinement studies. Use these results to estimate the asymptotic convergence rate for your implementation. Compare the derived rates to expected values.

  Note: You will likely encounter issues solving the resulting linear system using *Jacobi* when using a 4th-order stencil (due to a lack of diagonal dominance). For the 4th-order convergence analysis, you only need to run with Gauss-Seidel. For the 2nd-order convergence analysis, we would like to see results using both.

- Runtime Performance: for the mesh sizes used in your refine study, provide a figure or table presenting runtime performance measurements of your application. Delineate the total runtime into at least 3 sections which account for at least 90% of the total application runtime.

What to Turn In

Commit your functioning final code and updated model document (along with a pre-built pdf of the document) into your personal repository. It should compile and run *out-of-the-box* on Stampede using instructions provided in your model document.

The project is due by the end of day: Tuesday, November 26.

Grading

Your project will be graded based on the following structure:

1. Model Document (30%) - points awarded based on the completeness and quality of your document.
2. Code Quality and Build System (35%) - points awarded based on well-organized code, defensive programming constructs, and working build system on Stampede.
3. Functionality and Verification (35%)- points awarded based on completing all necessary runs, demonstrating required components (e.g. input parsing, performance timing, debug mode) and verifying the correct convergence rates of your numerical method.