HPC 563: Classical Simulation

Dominik Dahlem

March 27, 2008

Chapter 1

Setup

1.1 General Approach

- GNU autotools is used to maintain this project:
  autoconf version 2.61 automake version 1.10
- Getopt is used to parse command-line parameters.
  A help message can be displayed by specifying "-?" or "-h" to the command-line of the executable.
- The GSL library was chosen to provide some mathematical functions (i.e., `GSL_MAX` and `GSL_MIN`) and to control the floating point environment of the application.
- The cunit library was used to implement the unit tests.

1.2 Configuration

Each feature of the application is configurable. That means that the libraries required for certain features are checked when the configure script is executed and an appropriate message is presented to the user. Some features, such as MPI, GSL, and unit tests have inter-dependencies which are enforced.

- The assignment ships with a configure script generated by autotools. The following options are supported:
  - –enable-test : enables testing using cunit. The unit tests require GSL to be configured as well. An error message is produced, if the GSL library is not configured or not available.

- –enable-gcov : enables the coverage analysis using gcov and lcov.
- –enable-gsl : enables the GSL library.
- –enable-mpi : enables the parallel execution of the application using MPI. If MPI is enabled the unit tests are disabled, because the tests cannot be executed in a parallel fashion.
- –enable-debug : enables debug messages and asserts in some of the functions.
- –enable-report : Enable the report generation which is written in latex.
- –enable-plot : Enable the plot generation

1.3 Make

The project is set up in such a way that all sources can be built with a single make command in the root project directory. Additionally, it is also possible to cd into a c module or the report directory to build those individually.

- Installing the application was not considered.
- To build the project with MPI support do:

  1. No gcov, no tests, no MPI

     ```
     ./configure
     make
     ```

     Compile the sources

  2. No gcov, no tests, MPI enabled

     ```
     ./configure --enable-mpi
     make
     ```

     Compile the sources for parallel execution

  3. No gcov

     ```
     ./configure --enable-test
     make check
     ```

     This command will compile everything and run the cunit tests.

  4. Enable gcov and testing

```
./configure --enable-test --enable-gcov
make lcov
```

This command will compile everything, run the cunit tests, and generate a snazzy HTML coverage report using lcov [1].

5. Enable the report generation

```
./configure --enable-report --enable-plot
make
```

This command will compile the c, the gnuplot graphs, and the latex sources.

1.4 Assignment Layout

The assignment is structured in the following way:

```
- src       (the sources)
- test      (unit tests)
- doc       (doxygen generated code-documentation)
   - eval   (gnuplot script and surface plots)
   - report (the report)
```

A Doxygen configuration file is provided to generate the code documentation in HTML. doxygen support is integrated into the makefiles. Run: make doxygen-doc

```
- doc
   - doxygen
      - html
```

The generated doxygen report details the inter-relationships between the implemented modules and the source files.

The lcov coverage report is provided in the coverage folder. Though, only the infrastructural modules are tested, such as vector and matrix operations.

1.5 Execution

The conjugate gradient solver for Poisson's equation accepts a number of command-line arguments which are parsed with getopt. The executable resides in the src/main folder and can be called with

[1] http://ltp.sourceforge.net/coverage/lcov.php

`pdepoiss_solv`

- -s : Space dimension (Note: specify grid dimensions or the delta value)
- -t : Time dimension (Note: specify grid dimensions or the delta value)
- -d : Delta value (Note: specify grid dimensions or the delta value)
- -r : Error threshold for the conjugate gradient method
- -f : File name for the result surface
- -1 : Lower bound of the domain in the x dimension
- -2 : Upper bound of the domain in the x dimension
- -3 : Lower bound of the domain in the y dimension
- -4 : Upper bound of the domain in the y dimension
- -?/-h : The help message of the application

The space and time dimensions are required to be equal. This is ensured by the application itself. The default value for both is 26. If only one dimension is specified on the command-line then both are set to the highest value, i.e., 26 if the specified value is smaller than 26 or otherwise the specified value. The delta value will be calculated based on this setting.

Alternatively, the delta value can be specified upon which the space and time dimensions are derived. Fixing the delta value to some sensible values make sense to ensure that floating point operations are performed more accurately. It has been observed that fixing the dimensions to some value (i.e., 53) yields NaNs as a result. I assume this has something to do with floating point operations of very small numbers (such as 4.7308317893592655e-157). The following gdb debugger session identifies the location and the cause of the SIGFPE floating point exception:

```
> gdb
(gdb) set env GSL_IEEE_MODE=double-precision
(gdb) handle SIGFPE stop nopass
(gdb) file ./src/main/pdepoiss_solv
(gdb) r -s 53

Program received signal SIGFPE, Arithmetic exception.
0x08049ef2 in dcdssbmv (mat=0xbf98862c, u=0xbf9885b0, v=0xbf9885a8)
```

```
    at mult.c:47
47              v->data[i] = mat->diags[0].data[i] * u->data[i];
(gdb) p v->data[i]
$1 = 4.7308317893592655e-157
(gdb) p u->data[i]
$2 = 4.7308317893592655e-157
(gdb) p mat->diags[0].data[i]
$3 = 4
```

The error occured in the serial code of the matrix-vector multiply function `dcdssbmv()` .

These obvious errors go undetected, unless GSL is configured. GSL allows to control the floating point environment of C/C++ applications. Executing the application with

```
GSL_IEEE_MODE="double-precision," \
  mask-denormalized,mask-underflow" \
    ./src/main/pdepoiss_solv -s 53
```

ignores any errors related to denormalised and under-flowing small numbers, but it traps overflows, division by zero, and invalid operations [2].

The domain of the function space can be specified as well. The default values are set to the square region provided in the assignment description.

The file name of the gnuplot data for the surface plot can be specified which is set to result.dat as default name.

[2] http://www.gnu.org/software/gsl/manual/

Chapter 2

Results

This chapter presents the solution to the parallel conjugate gradient method to solve Poisson's equation

$$\bigtriangledown^2 u = -f \tag{2.1}$$

with $u \equiv u(x, y)$ on the square region

$$ABCD, A = (-0.5, -2), B = (2, -2), C = (2, 0.5), D = (-0.5, 0.5), \tag{2.2}$$

where the source density s given by

$$f(x, y) = 4\cos(x + y)\sin(x - y) \tag{2.3}$$

The exact solution and boundary conditions are given by

$$u = \cos(x + y)\sin(x - y) \tag{2.4}$$

which results in the following graph

Two simulation runs were performed – one in serial mode and the other one in parallel mode using the Message Passing Interface (MPI) with 16 configured nodes. Both runs are configured with $\delta = 0.01$ which yields 251 grid points. Figure 2.2 presents both graphs with the gradient on the surface illustrating the error Equation 2.5 in the conjugate gradient method compared to the analytical solution Equation 2.4.

$$\epsilon = |u(x, y) - v_{i,j}| \tag{2.5}$$

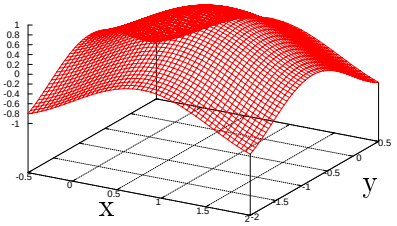

Figure 2.1: Exact Solution of Poisson's Equation

Serial Parallel

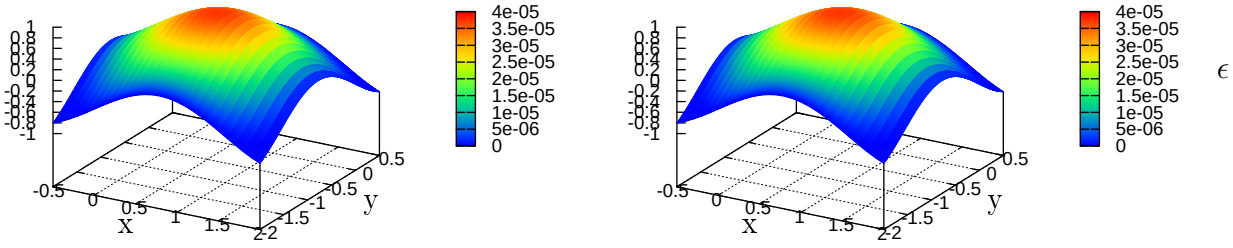

Figure 2.2: Approximate Solution of Poisson's Equation $\bigtriangledown^2 u = -f$