High-Performance Computing (G63.2011.002/G22.2945.001), Fall 2010

Homework Set 3

Due: October 26, 2010 (at the beginning of class) **Out: October 12, 2010**

The purpose of this assignment is to let you practice GPU programming. Like in assignment 2, we are seeking an optimized program that carries out a finite-difference method[1] for the numerical solution of a partial differential equation.

We will be solving the second-order linear *wave equation* with a source term,

$$\frac{\partial^2 u}{\partial t^2} = \triangle u + f = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} + f \tag{1}$$

on the domain $\Omega = [-1,1] \times [-1,1] \times [-1,1]$ with so-called *homogeneous Dirichlet boundary conditions*

$$u(\pm 1, y, z) = 0, \quad u(x, \pm 1, z) = 0, \quad u(x, y, \pm 1) = 0 \qquad \text{for } x, y, z \in [0,1].$$

The solution is a function u that depends on time t and the three spatial variables x, y, and z. We need two initial conditions at $t = 0$ for this equation, $u(0, x, y, z)$ and $(\partial u/\partial t)(0, x, y, z)$, both of which we will choose as zero.

You will notice many second derivatives above. In Assignment 2, we used a centered-difference approximation of the second derivative, which can be written as follows:

$$u''(x_i) = \frac{u(x_{i-1}) - 2u(x_i) + u(x_{i+1})}{h^2} + O(h^2), \qquad \text{given } x_i = ih. \tag{2}$$

where h is the grid spacing.

For brevity, we will from now on write

$$u^n_{i,j,k} = u(n\Delta t, ih, jh, kh) \tag{3}$$

Using the same approximation (2) for the time derivative and solving for u^{n+1} yields

$$u^{n+1} = 2u^n - u^{n-1} + \Delta t^2 RHS$$

which means that if at timestep n we know u^n and u^{n-1} and the approximate right-hand side ($RHS = \triangle u + f$), we can compute u^{n+1}. To obtain the spatial (x, y, z) derivatives in RHS, we will use two different stencils, pictured in Figure 1. Transforming stencil A of the figure (which is just (2) in disguise) into 3D and rewriting using (3), we obtain the update formula

$$u^{n+1}_{i,j,k} = 2u^n_{i,j,k} - u^{n-1}_{i,j,k} + \frac{\Delta t^2}{h^2}(-6u^n_{i,j,k} + u^n_{i-1,j,k} + u^n_{i+1,j,k} + u^n_{i,j-1,k} + u^n_{i,j+1,k} + u^n_{i,j,k-1} + u^n_{i,j,k+1}).$$

When choosing the time step Δt, one needs to obey the so-called Courant-Friedrichs-Lewy (CFL) condition[2]

$$\Delta t \leq Ch.$$

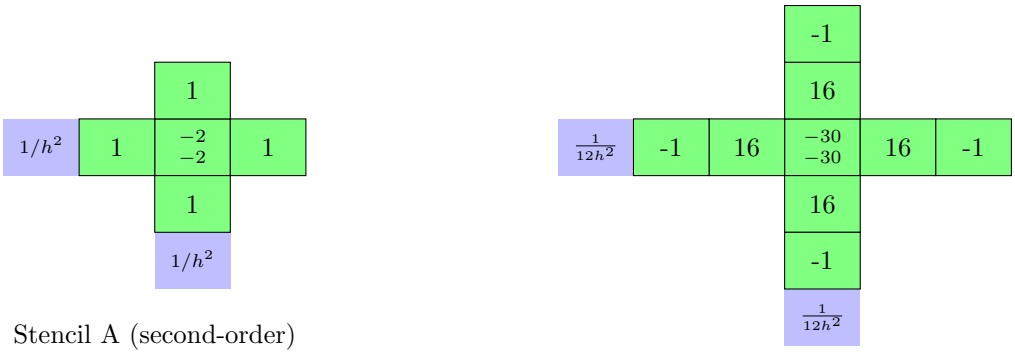
The maximal value C can be found theoretically and is called the CFL number.

We will also solve (1) using another finite-difference scheme whose stencil is shown in Figure 1 as stencil B. Stencil A results in a method which is second-order accurate in space, while Stencil B results in a fourth-order accurate method. ("Second-order" means that the error decreases as h^2.)

Once again, there is a starter kit to get you on your way, which you can get as usual (but also see the instructions on machine access further down). Once you have cloned the starter kit, type `make` and run

[1] `http://en.wikipedia.org/wiki/Finite_difference_method`

[2] `http://en.wikipedia.org/wiki/Courant-Friedrichs-Lewy_condition`



Stencil A (second-order)

Stencil B (fourth-order)

Figure 1: Finite-difference stencils for the second derivative in two dimensions. (Note that we are using three dimensions in this assignment.)

the resulting code `gpu-wave`. This will write out a number of files ending in `.bov` and `.dat`, which you may download and view on your machine using the free visualization program VisIt[3]. VisIt is available for Linux, Windows, and OS X. You should install it on your own machine to view the data. Installing on the cluster and visualizing remotely will be very slow because of the large amount of data involved in drawing the visualization. (See below for a short intro to VisIt.)

Then, you will need to do the following things, which we would like you to thoroughly document in a writeup named `writeup.pdf`. This writeup should be part of your submitted repository.

1. Change the source term f from what it is in the starter code to

$$f(t,x,y,z) = \exp(-1600((x-0.5)^2 + (y-0.2)^2 + (z-0.3)^2))(\sin(0.5t) + \cos(0.133t)).$$

2. Make the code more efficient. (The starter kit implementation is very poor.) As a starting point, consider the use of local memory. Beyond that, consider the improvements suggested in [Micikevicius, 2009, Michéa and Komatitsch, 2010, Cohen and Molemaker, 2009].

3. Estimate the following figures of merit for the original implementation and your improved implementation:

   - GFlops/s
   - MCells/s – how many million finite-difference cells are updated each second
   - Memory Bandwidth [GB/s]

   Which of those should you maximize? Do you estimate that your program will be memory-bound or compute-bound? Can you tell which of the two is actually the case from your measurements?

   The starter kit already provides a memory bandwidth estimate for its simple implementation. (You need to uncomment `#define DO_TIMING` in `gpu-wave.c` to enable timing.) Measure performance on a grid of size 32, 64, 128, and 256 in each direction. Do you see differences? If so, why?

4. Change the stencil to stencil B of Figure 1, repeat steps 2 and 3. (If necessary, adjust the CFL number. Just do this experimentally.)

5. Your code should work in single precision. Once you have a working (and tuned) single-precision code, you may also try to generalize this to double precision.

[3] `https://wci.llnl.gov/codes/visit/`

Note: Each time you change the code, you should convince yourself by ways of result visualization that your code still does what you intend–i.e. produce a plausible-looking solution for the wave equation.

Here's a cute hobby project (not part of the assignment in any way, not even for extra credit): Once you have all this machinery going, it's easy to build a solver for the heat equation

$$\frac{\partial u}{\partial t} = \triangle u + f. \tag{4}$$

(We've just removed one time derivative.) You need to figure out how to deal with the changed time evolution, and you will also need to decrease the time step quite drastically. (to $0.25h^2$) See if you can get an intuition about how the heat equation and the wave equation differ in behavior. Together with the Poisson equation we solved in assignment 2, the three equations we have seen are (arguably) the most important examples of linear partial differential equations.

References

J. Cohen and M. Molemaker. A fast double precision CFD code using CUDA. *Parallel Computational Fluid Dynamics: Recent Advances and Future Directions*, page 414, 2009. URL `http://jcohen.name/papers/Cohen_Fast_2009_final.pdf`.

D. Michéa and D. Komatitsch. Accelerating a three-dimensional finite-difference wave propagation code using GPU graphics cards. *Geophysical Journal International*, 182(1):389–402, 2010. doi: 10.1111/j.1365-246X.2010.04616.x.

P. Micikevicius. 3D finite difference computation on GPUs using CUDA. In *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*, pages 79–84. ACM, 2009. doi: 10.1145/1513895.1513905.

High-Performance Computing (G63.2011.002/G22.2945.001), Fall 2010

Technical Details about this Assignment

Machine Access

You will be using the `cuda` cluster at the NYU HPC facility for this assignment. In general, the same instructions as for `usq` apply, but every time you would have typed `usq`, you will now type `cuda`. You will need to repeat the following set-up steps from the first assignment:

- Create SSH key pair. (You will be prompted as on `usq`.)
- Add your public SSH key on `http://forge.tiker.net`. (Leave the one from `usq` in place!)
- `module load git`
- Tell `git` about yourself.

Once you complete the set-up steps, you may get the starter kit by typing

```
git clone ssh://git@forge.tiker.net:2234/hw3.git
```

Possible Issues

File System Stability

The file system on the `cuda` appears to be not quite as stable as on `usq`. You may get prompted for your password multiple times or see errors such as this one from `module load` commands in your `.bashrc`:

```
ModuleCmd_Load.c(200):ERROR:105: Unable to locate a modulefile for 'git/gnu/1.7.2.3'
```

In general, if something doesn't work, just try again and it should be fine. The HPC folks have advised me that this might be 'something to do with the automounter', but haven't come up with a fix yet (this is unrelated to the GPUs).

Sharing GPUs

Each compute node on `cuda` has only one GPU, and each GPU only accepts one running program at a time. At times, several of you may be logged into the same compute node. If someone else is using the GPU, you will see the error message

```
*** 'clCreateContext' failed with error 'invalid device'.
```

In this case, just wait a little while and try again. (Alternatively, specify `ppn=8` on your interactive job submission to make sure you're alone on the node. Note that in times of high demand, this is not a very nice thing to do, as there are only four GPUs for the whole group. If you can't get a GPU and have been trying for a while, please speak up on the list.)

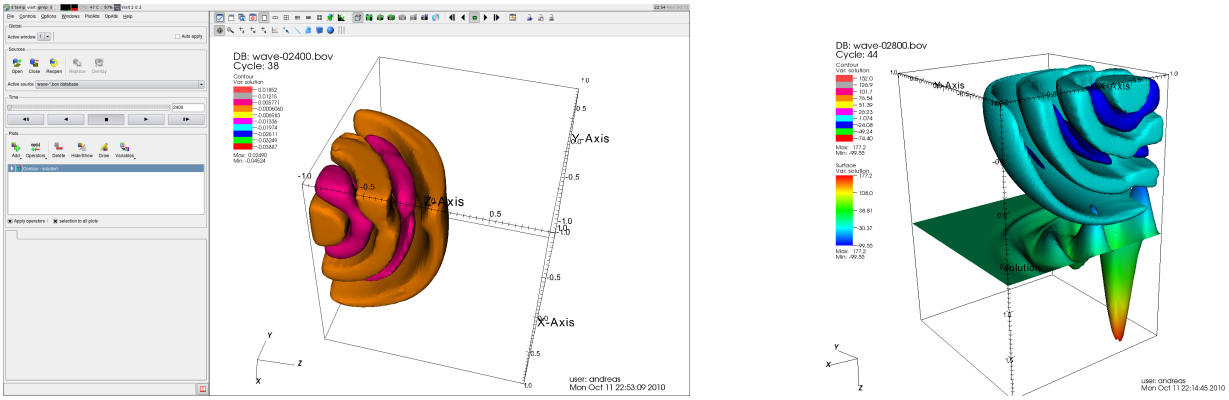


Figure 2: Left: (Roughly) what you should be seeing after you complete the intro procedure in VisIt. Right: With a bit of practice, VisIt can make rather fun visualizations.

Beware of Data Output

The GPU (even the slow, unoptimized kernel) can crank out data a few orders of magnitude faster than any hard drive can handle. For example, the code as it is in the starter kit dumps out about half a gigabyte of data in just a few seconds. Unfortunately, almost no file system will accept data this quickly. This means that the program will wait until the file system is ready to accept more data and then continue, which means your program will run more slowly. (Note that the timing code is written in such a way that it won't be affected by these slowdowns.)

Worse, your home directory is on a network file system, which is yet slower by another order of magnitude than regular hard drives. If your code runs slowly, try being more selective about which data to dump out. Also, consider storing data on the `/scratch` file system on the compute nodes.

A brief Intro to VisIt

VisIt is a free interactive parallel visualization and graphical analysis tool for viewing scientific data on Unix and PC platforms. Users can quickly generate visualizations from their data, animate them through time, manipulate them, and save the resulting images for presentations. VisIt contains a rich set of visualization features so that you can view your data in a variety of ways. It can be used to visualize scalar and vector fields defined on two- and three-dimensional (2D and 3D) structured and unstructured meshes. VisIt was designed to handle very large data set sizes in the terascale range and yet can also handle small data sets in the kilobyte range.

VisIt was developed by the Department of Energy (DOE) Advanced Simulation and Computing Initiative (ASCI) to visualize and analyze the results of terascale simulations. It was developed as a framework for adding custom capabilities and rapidly deploying new visualization technologies. After an initial prototype effort, work on VisIt began in the summer of 2000, and the initial version of VisIt was released in the fall of 2002. Although the primary driving force behind the development of VisIt was for visualizing terascale data, it is also well suited for visualizing data from typical simulations on desktop systems.

This intro describes VisIt version 2.1.

The code in the starter kit will output many files with names `wave-NNNNN.dat` and `wave-NNNNN.bov`, proportional to the number of time steps. Copy some or all of these (both `.bov` and `.dat`!) to your local machine to view them. Launch VisIt. On Linux, simply type

```
directory-where-you-untarred-visit/visit2_1_0.linux-intel/bin/visit
```

On Windows, check the 'Programs' menu, on OS X, see the 'Applications' folder after installation. Then, click 'Open', select the expanded sequence of `.bov` files, and click Ok. Then click "Add", "Contour", "solution",

and finally click “Draw”. A few contours should show up in your view window, which you can rotate by clicking and dragging your mouse. Now drag the time slider around to examine other time states. Note that VisIt will choose new values for the contours at every time level, which will make the contours ‘jump’ a lot. By double-clicking on the ‘Contour’ plot in the main window and fixing the contour limits, you may prevent this from happening.

By adding things from the “Operators” menu, you may slice and further process your data.