ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

P. Koumoutsakos, M. Troyer
ETH Zentrum, CAB H 69.2
CH-8092 Zürich

Set 12 - Multipole Method and GPUs

Issued: December 7th, 2012
Hand in: December 13th, 2012

Question 1: Vortical wake behind a cylinder with the multipole method

In the previous exercise we have demonstrated that the following vortex sheet on the surface of a cylinder:

$$\gamma(\theta) = -2U_\infty \sin\theta,$$

will cancel the normal component of an incoming free-stream velocity with strength U_∞ on the surface of the cylinder. In this exercise we will simulate the shedding of vorticity by diffusing the vortex sheet into the wake of the cylinder. Since this problem will involve large amounts of particles, we will use the Multipole Method to accelerate the velocity solver.

Assume we have discretized the cylinder surface with N_s vortices, as in the previous exercise. In this exercise, we will 'diffuse' the vortex sheet by adding at every timestep, N_s vortices into the flow. Each of these new vortices will be placed at a specified small offset normal to the existing vortices on the surface of the cylinder. The N_s new vortices will be initialized with a circulation equal to their corresponding surface vortex at the instant they are shed. We will set the offset equal to $\Delta = \sqrt{\nu \delta t}$, where ν is the viscosity of the fluid. The new vortices will represent the flow, rather than the cylinder surface. This means we will advect them and we will not update their strength after we initialized them, like the vortex sheet roll-up problem in exercise 11.

The strength of the vortex sheet on the cylinder will need to be adjusted to take into account the presence of all the free vortices in the flow, so that we still satisfy $\mathbf{u} \cdot \mathbf{n} = 0$ on the body surface. For a cylinder, we have access to the solution for the required strength of the vortex sheet in a flow with N free-stream particles. Evaluating this gives us, for the circulation of a vortex at angle θ and with discretization angle $\delta\theta = 2\pi/N_s$:

$$\Gamma(\theta) = \left(-\frac{2}{\pi} \sum_{i=1}^{N} \frac{\Gamma_i (r_i \cos(\theta - \theta_i) - R)}{R^2 + r_i^2 - 2 R r_i \cos(\theta - \theta_i)} - 2U_\infty \sin\theta \right) \delta\theta. \tag{1}$$

In this equation, the position of particle i is expressed in polar coordinates (r_i, θ_i) and R is the cylinder radius.

As in the previous question, given $Z = x + iy$ and $V = u - iv$, we have

$$V(Z) = -\frac{i}{2\pi} \sum_{i=1}^{N+N_s} \frac{\Gamma_n}{Z - Z_n}. \tag{2}$$

Using the multipole expansion, we get

$$V(Z) = -\frac{i}{2\pi}\frac{1}{Z - Z_M}\sum_{k=0}^{p}\frac{\alpha_k}{(Z - Z_M)^k} \tag{3}$$

$$\alpha_k = \sum_{n=1}^{N+N_s}\Gamma_n(Z_n - Z_M)^k \tag{4}$$

Here Z_M is the center of the cluster. If $\left(\frac{\max_n|Z_n - Z_M|}{|Z - Z_M|}\right) < 1$, then equation (3) will converge to equation (2) for $p \to \infty$.

a) Implement the Barnes-Hut Multipole method for accelerating the velocity solver of your vortex method. The algorithm consists of the following steps:

- Divide your domain into a quad-tree structure, so that each node of the tree contains less than $N_{\max}$ particles
- For each node of the tree, compute the multipole expansions of the particles in that node for $k = 1 \ldots p$, according to equation (4).
- For each point where you want to compute the velocity, traverse the tree top-down. Starting with the root box (covering the entire domain), evaluate if the box is close or far. If it is close, and if it has children, evaluate again for each of its children. If it is close but it does not have children, compute the velocity according to equation (2). If it is far, compute the velocity according to equation (3). Refer to the notes for further details.

Note that the last step is parallel in the number of particles. However, the work load for each target particle might be different, which complicates efficient parallellization of the code.

Check the accuracy of your multipole method by comparing the results with those from your N^2 solver.

b) Apply your Barnes-Hut multipole method to the evaluation of the above cylinder-wake problem. Each timestep consists of the following stages:

1. Compute the strength of the surface particles according to equation (1)
2. Compute the velocity on all free particles as induced by both the other free particles and the cylinder surface particles, using the multipole method evaluation of equation (2).
3. Advect the free particles according to their vorticity-induced velocity and the added free-stream velocity
4. Add N_s new particles to the domain, with an offset $\Delta = \sqrt{\nu\delta t}$ normal to the cylinder surface and a circulation equal to their corresponding surface particle

Choose $\delta t = 0.1$, $\nu = 0.01$, $R = 1$, $p = 4$ and $U_\infty = 0.1$. Put $N_s = 200$ particles on the surface and simulate at least up to $T = 50$. Show some figures of the particle configurations at later times.

There are three comments that need to be made here in order to keep the simulation going:

- Despite enforcing the boundary condition $\mathbf{u} \cdot \mathbf{n} = 0$, some free particles will still enter the cylinder. This is related to our point-discretization of the cylinder vortex sheet. To fix this, we will just delete all the free particles that are inside the cylinder at the end of each timestep as a zero-order solution.

- When adding particles to the flow, it is possible that they will be placed on top, or very close, to existing particles. This will cause a singularity in the velocity computation. To fix this, we will smooth the velocity field: in the denumerators of equations (1) and (2) we add a small number ϵ^2, where $\epsilon = 0.01$. With this fix we can handle the case where the source and target particles overlap without additional difficulties.
- The vortex sheet should satisfy $\bar{\Gamma} = \sum_{i=1}^{N_s} \Gamma(\theta_i) = 0$. In the evaluation of equation (1), however, over time some asymmetries will start to appear. To fix this, after evaluating equation (1), subtract from each surface particle the average mean circulation:

$$\Gamma_{\text{new}}(\theta) = \Gamma(\theta) - \frac{1}{N_s}\bar{\Gamma}$$

Question 2: Introduction to GPU computing

a) The NVIDIA CUDA toolkit includes two utility samples that demonstrate how to query device capabilities and measure GPU/CPU bandwidth. Find and run the Device Query and Bandwidth Test sample programs in the CUDA SDK (http://docs.nvidia.com/cuda/cuda-samples/index.html) and then use your findings to answer the following:

1. What type of GPU was found?
2. How many processor cores does the GPU contain? How many streaming multiprocessors?
3. What is the clock rate?
4. How much global memory is available?
5. What are the maximum block and grid sizes?
6. What are the host-to-device, device-to-host, and device-to-device memory bandwidths?

b) Vector Add is a very basic sample that implements element by element vector addition. Find, study and run the specific problem. Answer the following:

1. What are the block and grid sizes?
2. What is the name of the kernel in this program? What CUDA keyword identifies it?
3. Explain the function of the following line: `int i = blockDim.x * blockIdx.x + threadIdx.x;`
4. Briefly summarize the operation of the program, focusing on memory allocation/transfers.
5. Extend the code in order to measure the execution time of the kernel. Report the execution time of the kernel as the problem size (vector dimension) increases. Do not modify the default number of threads.
6. What is the maximum supported vector dimension for the specific configuration? Justify your answer.

Question 3: Matrix-Vector Multiplication

In this question, you are asked to develop matrix-vector multiplication using CUDA. This function is known in the BLAS standard library as sgemv (single precision) and dgemv (double precision). Your code will compute Y=A*X, where A is a $N \times N$ Lehmer matrix (as in Exercise 10), while X,Y are N-element vectors. Vector X is initialized with N equally space points (numbers) between 0 and 1.

A simple implementation approach is to run your kernel with N threads, and each thread will compute one coefficient of Y. A CPU implementation of the matrix-vector multiplication will be used to generate a correct solution which will be compared with your programs output. Both data initialization and comparison of results will be performed on the host.

Perform the following tasks:

1. Measure the performance of your GPU code for $N = 4096$ and double precision.
2. Repeat for single precision and compare the performance results.
3. Perform the previous experiments for matrix size $N = 8192$.

In your measurements you must include the time for transferring the input data (A,X) to the GPU and copying the results (Y) back to the CPU memory. Report the kernel configuration for each case.

Summary

Summarize your answers, results and plots into a PDF document. Furthermore, elucidate the main structure of the code and report possible code details that are relevant in terms of accuracy or performance. Send the PDF document, source code and related movies to your assigned teaching assistant.