CUDA PARALLELIZATION OF A 2-D NON-HYDROSTATIC COMPRESSIBLE ATMOSPHERIC MODEL

MATTHEW R. NORMAN

Project Webpage: `http://climlab04.meas.ncsu.edu/csc548/`

1. Problem Description

Computational fluid dynamics in general require large computational resources. The same is true for an atmospheric model which simulates non-hydrostatic density-stratified flow with a gravity source term. There have been many applications of CUDA to CFD problems as can be seen by the many papers on [1]. In fact, a full-scale global atmospheric model has been parallelized for CUDA[2].

For my graduate research, I am developing methods for integrating the 2-D non-hydrostatic atmospheric Partial Differential Equations (PDEs) that require very low communication when parallelized via domain decomposition. I already have a code in Fortran 90 implementing this in serial demonstrating certain positive traits of the scheme numerically. Because computation and communication is highly local, I believe this model would be an ideal candidate for parallelization with CUDA.

The equations of motion being approximated numerically are four conservation laws conserving mass, momentum, and entropy with a gravity source term (given below):

$$\frac{\partial}{\partial t}\begin{bmatrix}\rho\\ \rho u\\ \rho w\\ \rho\theta\end{bmatrix}+\frac{\partial}{\partial x}\begin{bmatrix}\rho u\\ \rho u^2+p\\ \rho wu\\ \rho\theta u\end{bmatrix}+\frac{\partial}{\partial z}\begin{bmatrix}\rho w\\ \rho uw\\ \rho w^2+p\\ \rho\theta w\end{bmatrix}=\begin{bmatrix}0\\ 0\\ -\rho g\\ 0\end{bmatrix}$$

The algorithm for the numerical approximation is too complicated to describe in too much detail (at least the mathematics behind it). For purposes of this project, the following gives a summary of the operations:

- For each time step
  - For one direction
    - ∗ For each cell
      - · Reconstruct intra-cell variations from surrounding cells
    - ∗ For each cell interface
      - · Evaluate interface fluxes from surrounding intra-cell variations
    - ∗ For each cell
      - · Update based on interface fluxes
  - For the other direction
    - ∗ For each cell
      - · Reconstruct intra-cell variations from surrounding cells
    - ∗ For each cell interface
      - · Evaluate interface fluxes from surrounding intra-cell variations
    - ∗ For each cell
      - · Update based on interface fluxes
  - For each cell in 2-D

[1] `http://www.nvidia.com/object/cuda_home.html`

[2] `http://www-ad.fsl.noaa.gov/ac/GPU_Parallelization_NIM.html`

  * Compute cell gravity source term
  * Update cell based on source term

The opportunities for parallelism are in the for loops. Therefore, there will be four separate kernels: one to perform reconstructions, one to evaluate fluxes, one to perform updates, and one for source term updates. Each of these steps is dependent upon the previous, so this will provide a natural way to synchronize as well.

1.1. **Algorithm Basics.**

Here, I am attaching a couple of excerpts from a paper in preparation describing some aspects of the flux computations. I am not including anything novel because the paper has not been submitted yet. It isn't of much consequence to the CUDA parallelization itself though.

In this study, we use a two-dimensional, compressible, non-hydrostatic model which explicitly conserves mass, momentum, and potential temperature (an entropy-related variable). A Cartesian rectangular grid is used for spatial discretization. The equation set is as follows:

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}(\mathbf{U})}{\partial x} + \frac{\partial \mathbf{H}(\mathbf{U})}{\partial z} = \mathbf{S} \tag{1.1}$$

where $\mathbf{U} = [\rho, \rho u, \rho w, \rho\theta]^\top$, $\mathbf{F} = \left[\rho u, \rho u^2 + p, \rho u w, \rho u\theta\right]^\top$, $\mathbf{G} = \left[\rho w, \rho w u, \rho w^2 + p, \rho w\theta\right]^\top$, $\mathbf{S} = [0, 0, -\rho g, 0]^\top$, ρ is the density, u is the horizontal wind, w is the vertical wind, p is the pressure, and θ is the potential temperature which is related to the real temperature T by $\theta = T\left(p_0/p\right)^{R_d/c_p}$. The equation set is closed by the equation of state: $p = C_0\left(\rho\theta\right)^\gamma$ where the constant C_0 is defined by: $C_0 = R_d^\gamma p_0^{-R_d/c_p}$. The constants are $\gamma = c_p/c_v \approx 1.4$, $R_d = 287$ J kg^{-1} K^{-1}, $c_p = 1004$ J kg^{-1} K^{-1}, $c_v = 717$ J kg^{-1} K^{-1}, and $p_0 = 10^5$ Pa. This set of conservation laws, neglecting the source term due to gravitational acceleration, is hyperbolic and admits a diagonalization with real eigenvalues and real eigenvectors.

In finite volume models, only the cell averages are evolved. The equation set is integrated over a computational cell $\Omega_i \in \left[x_{i-1/2,j}, x_{i+1/2,j}\right] \times \left[z_{i,j-1/2}, z_{i,j+1/2}\right]$ where $x_{i\pm1/2,j} = x_{i,j} \pm \Delta x/2$ and $z_{i,j\pm1/2} = z_{i,j} \pm \Delta z/2$. Then, the Gauss divergence theorem is applied to the flux divergence integrals transforming them into line integrals of the normal flux over the cell boundary. On a rectangular, Cartesian grid, this is given as:

$$\frac{\partial \overline{\mathbf{U}}_{i,j}}{\partial t} + \frac{\mathbf{F}_{i+1/2,j}(\mathbf{U}) - \mathbf{F}_{i-1/2,j}(\mathbf{U})}{\Delta x_{i,j}} + \frac{\mathbf{H}_{i,j+1/2}(\mathbf{U}) - \mathbf{H}_{i,j-1/2}(\mathbf{U})}{\Delta z_{i,j}} = \overline{\mathbf{S}}_{i,j}$$

where overbars represent cell averaged quantities. The flux evaluations in the scheme presented in this study are fully discrete, meaning the equations are integrated in time directly in one step. Since any integral can be represented as the average of the integrand multiplied by the interval, the final form we use is:

$$\overline{\mathbf{U}}_{i,j}^{n+1} = \overline{\mathbf{U}}_{i,j}^{n} - \frac{\Delta t}{\Delta x_{i,j}}\left(\overline{\mathbf{F}_{i+1/2,j}(\mathbf{U})}^{\Delta t} - \overline{\mathbf{F}_{i-1/2,j}(\mathbf{U})}^{\Delta t}\right) - \frac{\Delta t}{\Delta z_{i,j}}\left(\overline{\mathbf{H}_{i,j+1/2}(\mathbf{U})}^{\Delta t} - \overline{\mathbf{H}_{i,j-1/2}(\mathbf{U})}^{\Delta t}\right) + \overline{\overline{\mathbf{S}}_{i,j}}^{\Delta t}$$

where the overbar with a Δt superscript denotes a temporal average.

Finally, a second-order accurate Strang splitting is applied to integrate the fluxes in a sequence of 1-D sweeps. Consider the following update operators on any given cell:

$$\delta_x\left(\overline{\mathbf{U}}_{i,j}\right) = \overline{\mathbf{U}}_{i,j} - \frac{\Delta t}{\Delta x}\left(\overline{\mathbf{F}_{i+1/2,j}\left(\overline{\mathbf{U}}_{i-\frac{s-1}{2},j}, \ldots, \overline{\mathbf{U}}_{i+\frac{s+1}{2},j}\right)}^{\Delta t} - \overline{\mathbf{F}_{i-1/2,j}\left(\overline{\mathbf{U}}_{i-\frac{s+1}{2},j}, \ldots, \overline{\mathbf{U}}_{i+\frac{s-1}{2},j}\right)}^{\Delta t}\right) = 0$$

$$\delta_z\left(\overline{\mathbf{U}}_{i,j}\right) = \overline{\mathbf{U}}_{i,j} - \frac{\Delta t}{\Delta z}\left(\overline{\mathbf{H}_{i,j+1/2}\left(\overline{\mathbf{U}}_{i,j-\frac{s-1}{2}}, \ldots, \overline{\mathbf{U}}_{i,j+\frac{s+1}{2}}\right)}^{\Delta t} - \overline{\mathbf{H}_{i,j-1/2}\left(\overline{\mathbf{U}}_{i,j-\frac{s+1}{2}}, \ldots, \overline{\mathbf{U}}_{i,j+\frac{s-1}{2}}\right)}^{\Delta t}\right) = 0$$

where s is the size of the stencil used for spatial reconstruction (see section **??**). The splitting procedure is implemented as: $\mathbf{U}^* = \delta_x\left(\mathbf{U}^n\right)$, $\mathbf{U}^{n+1} = \delta_z\left(\mathbf{U}^*\right)$, $\mathbf{U}^{**} = \delta_z\left(\mathbf{U}^{n+1}\right)$, $\mathbf{U}^{n+2} = \delta_x\left(\mathbf{U}^{**}\right)$. Thus, the directional update order is reversed each time step. The only approximation made thus far is the directional splitting which is formally second-order accurate in time. Therefore, with the exception of the splitting and the source term, the accuracy depends on the approximation of the time-averaged interface fluxes.

In this study, we utilize two spatial reconstructions to view the behavior of the scheme as order of accuracy is increased. They are the piecewise linear (minmod limiter) method and the weighted essentially non-oscillatory (WENO) method. The piecewise linear reconstruction requires a three-cell stencil of data which we will refer to as: $\overline{\psi}_k$ where $k = -1, 0, 1$. The reconstruction is as follows:

$$\widetilde{\psi}^p(\xi) = \overline{\psi}_0 + c_1(\xi - \xi_0)$$

$$c_1 = \begin{cases} 0 & if \quad \left(\overline{\psi}_0 - \overline{\psi}_{-1}\right)\left(\overline{\psi}_1 - \overline{\psi}_0\right) \le 0 \\ \min\left(\frac{\overline{\psi}_0 - \overline{\psi}_{-1}}{\Delta\xi}, \frac{\overline{\psi}_1 - \overline{\psi}_0}{\Delta\xi}\right) & otherwise \end{cases}$$

where ξ is x or z depending upon the direction and ξ_0 is the center of the upwind cell. The WENO reconstruction is the same as in Capdeville [4] and requires a 5-cell stencil. WENO is fifth-order accurate in the best case (smooth data) and third-order accurate in the worst case (discontinuous data).

This scheme does not share the same conservation issues as many other characteristics-based schemes. In the typical difference splitting such as the Roe scheme, a special averaging of the conserved variables (Roe averaging) must be performed at an interface in order to construct eigenmatrices which render a conservative scheme overall. When splitting the flux directly, however, any averaging will produce a conservative scheme. With the proposed solver, even if the conserved variable vector itself were decomposed into characteristics (rather than the flux directly), any averaging will still produce a conservative scheme because cells communicate via a single-valued flux regardless. This also means that even if the reconstructions themselves are not conservative, the overall scheme will still be conservative.

Many current finite volume methods semi-discretize the non-linear equations. They perform the integration in time by applying an ODE solver such as Runge-Kutta to achieve higher-order temporal accuracy. In a fully discrete method, the temporal integral is computed directly without using ODE solvers (essentially forward Euler in nature). If the spatial reconstruction is accurate to $O(\Delta x^n)$, then the accuracy in time is also $O(\Delta x^n)$. Also, because of the CFL restriction, $\Delta t \propto \Delta x$ which means that the temporal accuracy is also $O(\Delta t^n)$. However, temporal accuracy is formally restricted to second-order regardless of the 1-D truncation error in individual sweeps because of the dimensional splitting. Still, for CFL restricted problems, spatial error dominates the total truncation error. Therefore, there is good reason to use higher-order reconstructions to reduce the spatial truncation error.

Concerning modern architectures and increased number of grid points, two concerns are scalability and memory usage. Architectures are reaching the petascale not by increased processor speed but by increasing the number of processors (already on the order of 10^5) requiring massively parallel integration methods. Also, as the mesh sizes are refined, the energy required to keep data active in fast memory (RAM) is also a concern. The proposed finite volume solver offers some advantages compared to many existing schemes in these regards. The popular explicit methods to obtain higher than first-order accuracy in time for the full non-linear set of equations are 1) centered schemes like leapfrog, 2) multi-stage methods like Runge-Kutta (RK), and 3) multi-step methods like Adams-Bashforth (AB). For scalability in a local and explicit framework, it is best to have minimal communication between nodes over the course of the simulation. This means the spatial reconstructions should have as small a halo region as possible, limiting the size of each exchange. It also means that the number of exchanges should be limited to reduce the latency of communication.

Leapfrog and AB require one flux evaluation per time step. However, to remove the “checkerboard” instability in time for leapfrog, an Asselin (sometimes called Robert) filter must be employed. This filter formally reduces the accuracy to first-order in time. The leapfrog method also requires storage of three time levels for each model state variable. AB methods suffer from linear computational modes which amplify for higher time steps, and they do not hold the promise to go beyond a CFL of unity in an explicit framework. Multi-stage methods like RK can have excellent stability properties, requiring no additional filtering (in time). However, they always require multiple exchanges of the halo per time step. Because of network latency, this can be very constraining for scalability. The proposed method only requires one halo exchange of only one time level of data per time step, and it only requires one time level of storage for each model state variable. Therefore, it provides a minimal communication and minimal storage framework for finite volume solvers. Additionally, it has the potential for extension past a CFL of unity in an explicit framework.

1.2. **Problem-Scaling and Speed-up.** If we consider problem scaling, there is an added difficulty present in all explicit time-dependent fluid codes: the CFL number. Basically, if we halve the grid spacing, we also must halve the time step. Assume that the wallclock time for one time step to be proportional to the amount of work to do in one time step $(D/\Delta x)^2$ where D is the domain size in one dimension, and Δx is the grid spacing. The number of time steps is $T/\Delta t$ where T is the total simulation time and Δt is the time step. The total wallclock time, then, is $C/\left(\Delta t \Delta x^2\right)$ where C is a constant or proportionality with D and T absorbed in.

If we wish to refine by a factor of N, we must do so both to the grid spacing and the time step to maintain numerical stability. Therefore, the new wallclock time is $C/\left[\left(\Delta t/N\right)\left(\Delta x/N\right)^2\right] = N^3 C/\left(\Delta t \Delta x^2\right)$. A grid refinement of N requires a factor of N^3 more time. Thus, we can relate the required speed-up as follows: $S = N^3$. If we want to cut the grid spacing in half and have the same wallclock time, we must have a GPU speed-up of 8. Likewise, if we want to cut the grid spacing by a fourth, we need a GPU speed-up of 64 to finish in the same amount of time.

This is a very unforgiving relation which asymptotes quickly. It, therefore, becomes obvious why we need so much computational power to throw at fluid problems in general. In 3-D, the situation is even worse. A refinement of N requires a factor of N^4 more time to complete. Therefore, a speed-up of 256 is required to cut the grid spacing by a factor of 4 and finish in the same amount of time. What I would like to see in this project is the grid spacing cut by a factor of 4 in each direction while finishing in the same amount of time. This will require a speed-up of 64 as mentioned earlier. This is probably an overly ambitious goal, but it is something to shoot for at least.

1.3. **Memory Requirements & Thread Occupancy.** The main limitation of GPUs for scientific codes is per-block shared memory. The maximum thread occupancy for a block is determined by the requirements for shared and per-thread memory. Computation of the maximum thread occupancy for a block is non-trivial because there are declared per-block shared variables, and each thread contributes to the total shared memory requirements. It is generally recommended to have 128 or more threads running in a block. Some have resorted to single precision because of this. However, for the present, I am not willing to do this because I have not tested this algorithm in single precision to ensure no large cancellation errors and the like are occurring. Correctness is a higher priority than speed-up, and I don't have the time to ensure correctness with single precision for this project.

2. Milestone Timeline

October 31, 2009: Have the existing code ported from Fortran 90 to C with primitive CUDA kernels in place for the 1) reconstruction, 2) flux evaluation, 3) updates, and 4) source terms. At this point, I do not expect to have any caching to per-block shared memory. I will use global memory accesses, and at this point, the only limit to thread occupancy is the amount of memory taken up by a single thread. It is very likely that in the long run, there will be multiple thread occupancies per block between the different kernels because of differing memory requirements and halo dependencies. For instance, the reconstruction stage requires a halo of two cells (in terms of 1-D), the flux stage requires a halo of one cell, and the update and source stages do not requires a halo.

November 7, 2009: Implement pre-caching in the per-block shared memory for variables with more than two accesses per kernel call. This involves each thread pulling in a near equal amount of data and synchronizing before continuing the work. A large number of threads will hide some of the latency in a pipelining effect. Given the limited amount of time to get this project together, it may be wise to have some groundwork laid for cudaMPI (or homemade DMA + MPI) communication between different GPUs at this point.

November 14, 2009: Have the code running with CUDA across multiple GPUs with some form of MPI communication. At this point, the code needs to be close to ready for benchmarking against serial code.

November 21, 2009: Have all of the benchmarks and speed-ups computed. In the end, there should be 4 versions of the code: 1) serial, 2) CUDA with global memory accesses only, 3) CUDA with shared memory caching, 4) CUDA + MPI. The speed-up of each version should be computed by this point and the project presentation ready.

References. A few references to methods similar in nature to mine are: Ahmad [1], Ahmad and Linderman [2], Bale et al. [3], Carpenter et al. [5], Leveque [6], Lin [7], Lin and Rood [8, 9], Roe [11, 10], Shi and Toro [12], Shu [13], Shu and Osher [14]

For similar-natured CFD problems that have been parallelized for CUDA, a good website to check is the CUDA zone website[3]. On that site, the similar project titles are “Running Unstructured Grid CFD Solvers on Modern Graphics Hardware”, “A Fast Double Precision CFD Code using CUDA”, “Multi-GPU Incompressible Navier-Stokes Solver”, “Numerical Weather Prediction”.

References

[1] Ahmad, N., 2008. The f-wave riemann solver for meso- and micro-scale flows. AIAA Paper 2008-465.

[2] Ahmad, N., Linderman, J., 2007. Euler solutions using flux-based wave decomposition. International Journal for Numerical Methods in Fluids 54, 47–72.

[3] Bale, D., Leveque, R. J., Mitran, S., Rossmanith, J. A., 2002. A wave-propagation method for conservation laws and balance laws with spatially varying flux functions. SIAM Journal on Scientific Computing 24, 955–978.

[4] Capdeville, G., 2007. A central weno scheme for solving hyperbolic conservation laws on non-uniform meshes. Journal of Computational Physics 227, 2977–3014.

[5] Carpenter, R. L., Droegemeier, K. K., Woodward, P. R., Hane, C. E., 1990. Application of the piecewise parabolic method (ppm) to meteorological modeling. Monthly Weather Review 118, 586–612.

[6] Leveque, R. J., 2002. Finite Volume Methods for Hyperbolic Problems. Cambridge University Press, Cambridge, MA.

[7] Lin, S. J., 2004. A “vertically lagrangian” finite-volume dynamical core for global models. Monthly Weather Review 132, 2293–2307.

[8] Lin, S. J., Rood, R. B., 1996. Multidimensional flux-form semi-lagrangian transport schemes. Monthly Weather Review 124, 2046–2070.

[9] Lin, S. J., Rood, R. B., 1997. An explicit flux-form semi-lagrangian shallow-water model on the sphere. Quarterly Journal of the Royal Meteorological Society 123, 2477–2498.

[10] Roe, P. L., 1981. Approximate riemann solvers, parameter vectors, and difference schemes. Journal of Computational Physics 43, 357–372.

[11] Roe, P. L., 1986. Characteristic-based schemes for the euler equations. Annual Review of Fluid Mechanics 18, 337–365.

[12] Shi, J., Toro, E. F., 1996. Fully discrete high-resolution schemes for hyperbolic conservation laws. International Journal for Numerical Methods in Fluids 23, 309–323.

[13] Shu, C. W., 1999. High order ENO and WENO schemes for computational fluid dynamics. In: Barth, T. J., Deconinck, H. (Eds.), High-Order Methods for Computational Physics. Vol. 9 of Lecture Notes in Computational Science and Engineering. Springer, pp. 439–582.

[14] Shu, C. W., Osher, S., 1988. Efficient implementation of essentially non-oscillatory shock-capturing schemes. Journal of Computational Physics 77, 439–471.

[3] `http://www.nvidia.com/object/cuda_home.html`