Jul 1st, 12:00 AM

Using NVIDIA GPU for Modelling the Lagrangian Particle Dispersion in the Atmosphere

Jiye Zeng

Tsuneo Matsunaga

Hitoshi Mukai

International Environmental Modelling and Software Society (iEMSs)
2010 International Congress on Environmental Modelling and Software
Modelling for Environment's Sake, Fifth Biennial Meeting, Ottawa, Canada
David A. Swayne, Wanhong Yang, A. A. Voinov, A. Rizzoli, T. Filatova (Eds.)
http://www.iemss.org/iemss2010/index.php?n=Main.Proceedings

Using NVIDIA GPU for Modelling the Lagrangian Particle Dispersion in the Atmosphere

Jiye ZENG[1], Tsuneo MATSUNAGA[1], Hitoshi MUKAI[1]
[1]*National Institute for Environmental Studies*
Onogawa 16-2, Tsukuba, Ibaraki 305-8506, Japan
zeng@nies.go.jp

Abstract: A Lagrangian particle dispersion model computes trajectories for a large number of particles and is often used for describing the transport and diffusion of tracers in the atmosphere. In a typical application, computing the position changes of particles is the bottleneck that controls the total execution time. Such a model presents a good case for investigating the use of NVIDIA's Graphic Processor Unit or GPU. We developed a Lagrangian particle dispersion model based on FLEXPART to take the advantage of the GPU's parallel computing power. Preliminary tests show that using a GPU can improve the performance by as much as 10 times.

Keywords*:* Lagrangian particle dispersion model, NVIDIA GPU, Parallel computing.

1. Introduction

In environmental studies, it is common to use a Lagrangian particle model for describing the transport and diffusion of tracers in the atmosphere (e.g., Fiedler et al., 2009; Niemi et al., 2009; Cooper et al., 2010). The number of particles in one release by a model is often in the order of 10,000 to 100,000 and the time step for the integration of differential equations ranges from a few to tens of seconds (Folini et al., 2008). Furthermore, a model may release particles constantly in every few hours (Ubl, 2005). Under these conditions, computing the position changes of Lagrangian particles with time becomes the bottleneck that controls the total execution time of a model. The circumstance presents a good case for investigating the use of NVIDIA's Graphic Processor Unit or GPU, which has evolved into a highly parallel, multithreaded, many-core processor with tremendous computational horsepower and very high memory bandwidth (NVIDIA, 2009).

In this paper, we introduce a practical use of the GPU for modelling the Lagrangian particle dispersion in the atmosphere. The model is based on FLEXPART (Stohl et al., 2005). We discuss our approach to implement the kernel functions and compare the execution times of of serveral modelling processes with and without using the GPU.

2. Modelling with GPU

The FLEXPART model is based on the assumption that turbulent diffusion can be modelled as a Markov chain (Obukhov, 1959) and the trajectories of Lagrangian particles can be estimated by

$$\begin{aligned} x_i(t+\Delta t) &= x_i(t) + u_i(\mathrm{t})\Delta t \\ y_i(t+\Delta t) &= y_i(t) + v_i(\mathrm{t})\Delta t \\ z_i(t+\Delta t) &= z_i(t) + w_i(\mathrm{t})\Delta t \end{aligned} \tag{1}$$

Where x, y, and z are the three space coordinates; u, v, and w the corresponding wind velocities; and i denotes a particle. The wind velocities can be decomposed into the mean components $\overline{u_i}$, $\overline{v_i}$ and $\overline{w_i}$ and the turbulent components u_i', v_i', and w_i', i.e.,

$$\begin{aligned} u_i &= \overline{u_i} + u_i' \\ v_i &= \overline{v_i} + v_i' \\ w_i &= \overline{w_i} + w_i' \end{aligned} \tag{2}$$

The mean components are taken directly from the output of a meteorological model and the turbulent components are estimated from a number of meteorological parameters. The details are available in Stohl et al. (2005).

As it is illustrated in Figure 1, the main processes of modelling Lagrangian particles include reading and processing meteorological data, interpolating the data to the current time of particles, and calculating the changes of particle positions; therefore our task to use the GPU in modelling is to implement code for those processes.

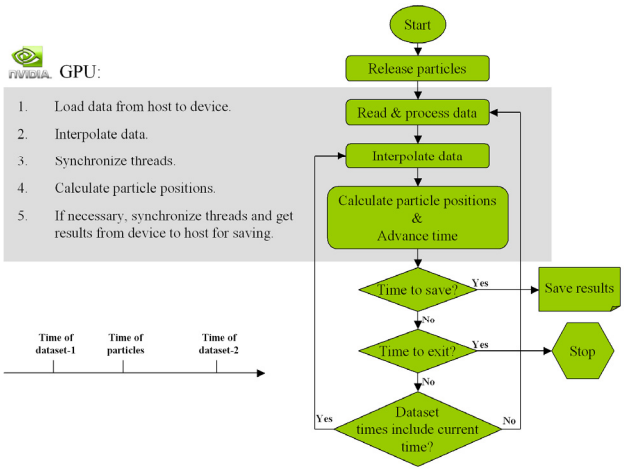

Figure 1. Illustration of modelling Larangian particle dispersion.

The implementation was done in consideration of the GPU's special features. The GPU is especially well suited to address problems that can be expressed as data-parallel computations. More specifically, the GPU is efficient to handle independent objects in parallel by multiple threads. In our implementation, each thread is assigned to a grid to interpolate data for that grid and to a particle to calculate its new position at runtime.

We used C++ with NVIDIA's CUDA™ runtime library to implement the FLEXPART model for the GPU. Our priority is to make the program executable by computers with and without the GPU. In order to minimize the debugging time, we programmed the kernel functions for the GPU in such a way that they can be used by the CPU with minimal code changes. Considering this requirement and the similarity of memory allocation and access between the GPU device and its host, the kernel functions mainly use the global memory of the GPU, although the global memory is much less efficient than the shared memory.

There are many parameters to be passed to the entry kernel function to calculate particle positions. For example, at least 11 pointers to meteorological data must be passed to the function: u-wind, v-wind, w-wind, geopotential height, air density, density gradient, planetary boundary layer height, tropopause layer height, friction velocity, convective velocity scale, and Monin-Obukhov length. Passing a long list of parameters to a function

make it difficult to maintain the code; and in CUDA™, function parameters are limited to 256 bytes. To overcome the difficulty, we encapsulate relevant variables in structures and passed them to kernel functions. For example,

```
struct sField {
    float *U;
    float *V;
    float *W;
    // more variables
    ……
};

void kernel_f(sField *F,……);
```

The pointers in sField can be set to the host memories to execute the model using the CPU or to the device memories to execute it using the GPU.

3. Results

Figure 2 shows the particle distribution from a model run for 3-day back trajectories. We compared the performances between CPU and GPU for this run. The execution conditions are listed as follows:

- OS: Windows XP 64bit edition.
- Host computer: HP Z800 Workstation, Intel Xeon CPU, 2.00GHz, 4GB RAM.
- GPU device: Tesla C1060, CUDA™ driver 2.3, clock rate 1.3GHz, 4GB global memory.
- Execution: Meteorological data input of every 3 hours, meteorological data interpolation of every minute, integration time step of 10 seconds, back trajectory length of 3 days, result output of every hour.
- Thread layout: <<<n,256>>>. One block has 256 threads. The number of blocks is n=1+(np-1)/256, where np is the number of particles.

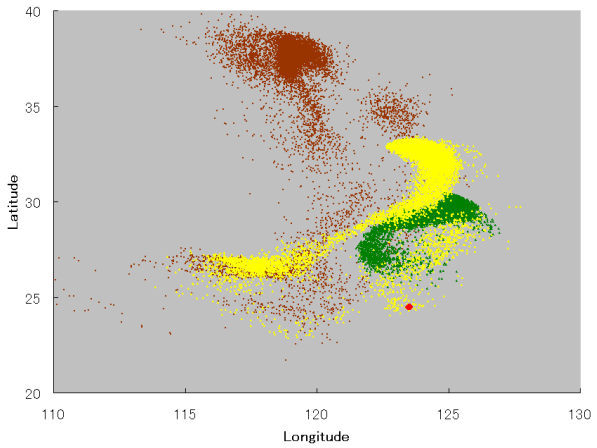

Figure 2. Distribution of 10,000 particles in a test run for 3 day back trajectories started at 1:00 on 10 January 2006. Particles are released randomly at (24.5^0N, 123.5^0E) and between 0 to 100 m above the surface. Red: initial position; green: 1 day back; yellow: 2 day back; and maroon: 3 day back.

Table 1 shows that the integration of equation (1), which involves the estimation of turbulence parameters, takes most of the total execution time, and that the integration time increases proportionally with the number of particles. Using the GPU reduces the integration time by about 8 folds for 10,000 particles to 12 folds for 100,000 particles. For

data interpolation, which is a simple operation, the improvement is about 24 folds. Reading data from hard disk to memory and saving data from memory to hard disk take relatively small amount of time. When the GPU is used, reading and saving take longer times because they involve copying data between the host memory and the device memory.

From the total execution time ratio of host to device, we can see that the performance improves rapidly with the increase of particle number when the number is smaller than 25,000 (Figure 3), but the improvement slows down after that. One explanation is that one Tesla C1060 has 30 multiprocessors and the maximum number of active threads per multiprocessor is 768; therefore the model only can activate up to 23,040 threads in the GPU. When the number of particles is smaller than the maximum of available active threads, the GPU can handle the integration of all particles at once and therefore, the more particles there are, the less the integration time per particle relative to other processing times.

Table 1. Performance comparison between the host CPU and the device GPU.

	Integration (ms)			Interpolation (ms)			Reading (ms)*			Saving (ms)**		
NP	Host	Device	Ratio	Host	Device	Ratio	Host	Device	Ratio	Host	Device	Ratio
10000	287,006	36,827	7.8	173,490	6,987	24.8	9,800	10,172	0.96	15	108	0.14
25000	712,860	72,915	9.8	172,390	6,978	24.7	9,779	10,152	0.96	126	268	0.47
50000	1,427,982	131,487	10.9	173,942	6,964	25.0	9,793	10,186	0.96	205	237	0.86
100000	2,853,360	251,253	11.4	173,218	7,156	24.2	9,828	10,185	0.96	796	811	0.98
1000000	28,358,492	2,379,439	11.9	174,886	7,541	23.2	9,912	10,179	0.97	7,543	7,658	0.98

* Each reading reads about 84 MB meteorological data to the host memory and copy them to the device memory when the GPU is used.

**Each saving writes about 200 KB to 2 MB data to hard disk and copy them from host to device when the GPU is used.

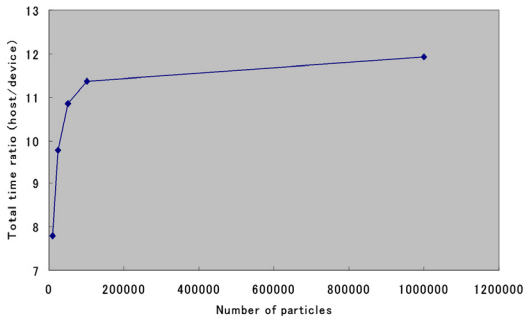

Figure 3. The total execution time ratio of host to device vs. the number of particle release.

4. Discussions

The model uses only the global memory of the GPU, which is much slower than the shared memory. The performance could be improved by using the shared memory in some parts of the kernel functions, but that would greatly increase the complexity of coding because it requires completely different implementations of kernel functions for the CPU and the GPU.

Our experiments show that using a Tesla C1060 processor can improve the execution speed for about 10 times, but there is a limit for the performance gain when the number of particles exceeds the miximum of available active threads of a GPU. It indicates that to gain more computing speed for a large application, the model should be designed to use multiple devices with each taking a part of the modelling. For example, in a one-host-multi-devices system, we can assign one device to one release for multiple releases of particles, or partition a release to smaller pieces and assign them to different devices.

Coding for the GPU is both fun and challenging, especially in transforming code written for the CPU. We would like to share two of our experiences here. First, the CUDA™ SDK toolkit for C/C++ does not like the "goto" statement. Any code section with "goto" must be rewritten. Second, any expression similar to `V[k]=V[k+1]` is problematic for optimization, because in a multiple thread environment, while one thread is reading `V[k+1]`, another thread may be writing to `V[k+1]` at the same time. In a less obvious case, the CUDA™ compiler may accept the code, but the result is uncertain, for example:

```
j = func(V[k]);
C[j] = C[j] + 1;
```

That is that a thread calls `func` with `V[k]` to produce `j`, which in turn is used as an index for writing to `C[j]`. When `j` and `k` are not the same, different threads may operate on `c[j]` and `v[k]` in undefined order. A solution to such a case is to use the atomic functions of CUDA™ SDK. But they reduce the performance significantly. We observed about 25% performance loss by using them in serverl tests.

REFERENCES

Cooper, O. R., D. D. Parrish, A. Stohl, M. Trainer, P. Nédélec, V. Thouret, J. P. Cammas, S. J. Oltmans, B. J. Johnson, D. Tarasick, T. Leblanc, I. S. McDermid, D. Jaffe, R. Gao, J. Stith, T. Ryerson, K. Aikin, T. Campos, A. Weinheimer & M. A. Avery, *Increasing springtime ozone mixing ratios in the free troposphere over western North America, Nature 463*, 344-348, 2010.

Fiedler, V., R. Nau, S. Ludmann, F. Arnold, H. Schlager, and A. Stohl, *East Asian SO_2 pollution plume over Europe – Part 1: Airborne trace gas measurements and source identification by particle dispersion model simulations, Atmospheric Chemistry and Physics 9*, 4717-4728, 2009

Folini, D., S. Ubl, and P. Kaufmann, *Lagrangian particle dispersion modeling for the high Alpine site Jungfraujoch, Journal of Geophysical Research, 113*(D18111), 2008

Niemi, J.V., , S. Saarikoski, M. Aurela, H. Tervahattu, R. Hillamo, D.L. Westphal, P. Aarnio, T. Koskentalo, U. Makkonen, H. Vehkamaki, and M. Kulmala, *Long-range transport episodes of fine particles in southern Finland during 1999–2007, Atmospheric Environment 43*, 1255–1264, 2009

NVIDIA, *CUDA™ Programming Guide, version 2.3*, 2009

Obukhov, A. M., Description of turbulence in terms of Lagrangian variables, Advances in Geophysics 6, 113–116, 1959.

Stohl, A., C. Forster, A. Frank, P. Seibert, and G. Wotawa, *Technical Note: The Lagrangian particle dispersion model FLEXPART version 6.2, Atmospheric Chemistry and Physics, 5*, 2461-2474, 2005.

Ubl, S., *Backward Lagrangian Particle Dispersion Modeling: Applications for a High Alpine Measurement Site, Doctoral Thesis ETH No. 16282, 108 p, Swiss Federal Institute of Technology Zurich*, 2005.