ARL-TN-0907 • SEP 2018

US Army Research Laboratory

The Second Approach to Improving the Performance of the Lagrangian Particle Dispersion Model (LPDM) Using Graphics Processing Unit (GPU) Computing

by Leelinda P Dawson and Yansen Wang

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TN-0907 ● SEP 2018

US Army Research Laboratory

The Second Approach to Improving the Performance of the Lagrangian Particle Dispersion Model (LPDM) Using Graphics Processing Unit (GPU) Computing

by Leelinda P Dawson and Yansen Wang
Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED (From - To)**
September 2018	Technical Note	October 2017–August 2018

4. TITLE AND SUBTITLE

The Second Approach to Improving the Performance of the Lagrangian Particle Dispersion Model (LPDM) Using Graphics Processing Unit (GPU) Computing

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Leelinda P Dawson and Yansen Wang

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

US Army Research Laboratory
ATTN: RDRL-CIE-M
2800 Powder Mill Road
Adelphi, MD 20783-1138

8. PERFORMING ORGANIZATION REPORT NUMBER

ARL-TN-0907

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Certain atmospheric transport and dispersion models can take many hours to complete execution, such as the Lagrangian Particle Dispersion Model (LPDM) that simulates the ensemble average transport of aerosols and gases in turbulent wind conditions. The execution time needs to be reduced for these models, so they can be useful for rapid release and planning purposes. Performance improvements were achieved during the initial approach by using one technique of graphics processing unit (GPU) computing technology. This report documents the second approach of the implementation and integration processes using GPU computing technology to improve the LPDM code's performance. The execution time of GPU-accelerated LPDM application implementing the second approach was faster than both the original LPDM application without GPU computing and the GPU-accelerated LPDM application implementing the initial approach.

15. SUBJECT TERMS

Lagrangian Particle Dispersion Model, graphics processing unit, GPU, CUDA, OpenACC

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Leelinda P Dawson
Unclassified	Unclassified	Unclassified	UU	16	**19b. TELEPHONE NUMBER (Include area code)** 301-394-5636

Contents

List of Figures

1. Introduction

Atmospheric transport and dispersion models are used to predict downwind hazards associated with the release of hazardous materials.[1] The Lagrangian Particle Dispersion Model (LPDM) computes the trajectories of many air parcels or aerosols in a turbulent flow field and makes an ensemble average to give the concentration field.[2] The LPDM can be coupled with the atmospheric boundary flow models[3,4] inline or offline for the transport and dispersion predictions. Since LPDM with a large domain can take many hours to complete execution, there is a great need to reduce their execution times so they can be useful for rapid release and planning purposes on the battlefield.

As discussed in Dawson,[5] there were some performance improvements of the LPDM utilizing one technique of graphics processing unit (GPU) computing technology during the initial effort. However, more performance improvements were preferred, so additional GPU computing techniques were explored to further improve the LPDM code's performance. It is common practice to experiment with multiple GPU computing techniques until optimal performance is achieved. As stated in Dawson,[5] the most intensive portion of the model computation in the LPDM code is the Gaussian random process, which simulates the diffusion by small turbulent eddies. Thus, a different approach that further reduces the execution time of the Gaussian random process using other GPU computing techniques was investigated.

The purpose of this report is to describe the second approach of the implementation and integration processes of GPU computing technology to improve the LPDM code's performance. Similar to the initial approach, the LPDM code was successfully integrated with GPU computing technology with few modifications to the original code. The execution time of the GPU-accelerated LPDM application implementing the second approach was faster than both the original LPDM application without GPU computing and the GPU-accelerated LPDM application implementing the initial approach.

2. GPU and Development Environment

In GPU computing, the compute-intensive portions of the application are mainly offloaded to the GPU, while the remainder of the application code runs on the central processing unit (CPU).[6] The CPU has fewer running cores compared to the GPU where there are thousands of cores designed to run parallel applications simultaneously and more efficiently. As a result, GPU-accelerated applications are able to run much faster than a CPU application.

The code of the original LPDM[2] application was developed in Fortran 90.[7] Similar to the first approach described in Dawson,[5] the Portland Group (PGI) Fortran compiler version 16.10.0 was used to compile the LPDM code during the second approach. In addition, an updated version of the CUDA 8.0[8] application interface for Compute Capability 3.5 was used during the second approach compared to CUDA 7.5 for Compute Capability 3.0 used in the initial approach. Also, the OpenACC[9] application programming framework was used, similar to the initial approach. The LPDM code was easily modified to utilize GPU computing technology with little programming effort by implementing both CUDA and OpenACC.

3. CUDA Host Code

As discussed in Dawson,[5] the function *gaussdev* was determined to be the hotspot for the LPDM code via the PGPROF profiler. This function implements the Gaussian or normal random distribution, which is commonly used in modeling natural atmospheric turbulence accurately. Furthermore, CUDA has a built-in random number generator (RNG) library called CUDA Random Number Generation library (cuRAND) that produces high-performance GPU-accelerated random number generation.[10] This library provides an interface to generate multiple pseudorandom numbers implementing CUDA on the GPU from several RNG distributions including Gaussian or normal distribution.

The initial approach discussed in Dawson[5] implemented cuRAND using a CUDA Fortran interface via OpenACC from a device code developed by the author. The CUDA Fortran interface connected the CUDA device code to the GPU-accelerated LPDM application via OpenACC using three source code files. In contrast, as shown in Fig. 1, the second approach implements cuRAND using CUDA Fortran host code via OpenACC, where PGI provides a prebuilt interface module for the cuRAND library. PGI Fortran cuRAND library routines[8] produce an array of pseudorandom numbers at runtime when they are simply called from CUDA Fortran host code. The host code consists of one source code file, *lagmod.f90*, which is the main LPDM Fortran code that was slightly changed to include function calls to the PGI Fortran cuRAND library. This host code approach is an easier technique because it uses one source code file compared to the device code approach that uses three source code files. Therefore, it achieves greater code performance with less programming effort compared to the initial approach.

As depicted in Fig. 1, the original Fortran LPDM code was slightly modified to include GPU acceleration by using the CUDA Fortran host code via OpenACC. Note that this figure shows an excerpt of the code where it was successfully

modified to include GPU computing methods. There were multiple function calls to the PGI Fortran cuRAND library in the code, which replaced each call to the *gaussdev* function located in the original Fortran LPDM code since it was determined to be the most intensive section for model computation. The random seed of the RNG was determined by the system's clock during each call to the RNG using the *curandSetPseudoRandomGeneratorSeed* function. The value of the RNG's mean and standard deviation was initialized and set to 0.0 and 1.0, respectively, before each call to generate the pseudorandom numbers using the *curandGenerateNormal* function. The rest of the code for the GPU-accelerated LPDM application was not modified and still runs on the CPU.

Original LPDM code:

```
integer,parameter::no_pt=1000000
real::aa(no_pt)

do i=1,no_pt
  aa(i)=gaussdev()
enddo
```

GPU-accelerated LPDM Host code:

```
use cudafor
use curand

integer,parameter::no_pt=1000000
real::aa(no_pt)

real::mean=0.0
real::stddev=1.0
integer::istat
integer::clock
type(curandGenerator)::g
real, dimension(:), allocatable::r
integer::n = 1000000
allocate(r(n))
r=0.0

!$acc data copy(r)
istat=curandCreateGenerator(g,CURAND_RNG_PSEUDO_XORWOW)
!$acc host_data use_device(r)
call SYSTEM_CLOCK(clock)
istat=curandSetPseudoRandomGeneratorSeed(g,clock)
istat=curandGenerateNormal(g,r,n,mean,stddev)
istat=curandDeviceSynchronize()
!$acc end host_data
istat = curandDestroyGenerator(g)
!$acc end data

aa=r
```

Compiler output of GPU-Accelerated LPDM Host code:

```
pgfortran  -acc  -Mcuda  -Minfo=accel  -ta=nvidia:cuda8.0,cc35    -c
lagmod.f90 -o lagmod.o /opt/pgi/16.10.0/linux86-64/16.10/lib
lagmod.f90:
lagmod:
    193, Generating copy(r(:))
```

Fig. 1 Original LPDM code vs. GPU-accelerated LPDM host code with compiler output

The common sequence of operations from the GPU-accelerated LPDM Fortran host code is as follows:

1) Initialize variables: host array (*aa*), host RNG (*g*), RNG's mean (*mean*), and RNG's standard deviation (*stddev*).

2) Initialize device array (*r*) and allocate its GPU memory.

3) Call PGI's cuRAND library routines to create RNG, set seed of RNG based on the system's clock, generate normal pseudorandom numbers, synchronize GPU, and destroy RNG.

4) Transfer the data results from the GPU to the device array, *r*, using OpenACC's *copy* feature. Then, this variable is copied to the host array, *aa*.

4. Results

As stated in Section 3, the CUDA Fortran host code was successfully used to generate a GPU-accelerated LPDM application that was faster than both the original LPDM application without GPU computing and the GPU-accelerated LPDM application implementing the initial approach using the CUDA device code discussed in Dawson.[5] As depicted in Fig. 2, the original LPDM application without GPU computing ran on the CPU with the execution time of 7 min and 57 s, the GPU-accelerated LPDM application implementing the first approach with the CUDA device code ran with the execution time of 4 min and 35 s, and the GPU-accelerated LPDM application implementing the second approach with the CUDA host code ran with the execution time of 2 min and 47 s. The GPU-accelerated LPDM application implementing the second approach using CUDA host code is running approximately 2.85 times faster than the original LPDM application without GPU computing and 1.65 times faster than the GPU-accelerated LPDM application implementing the initial approach using the CUDA device code.

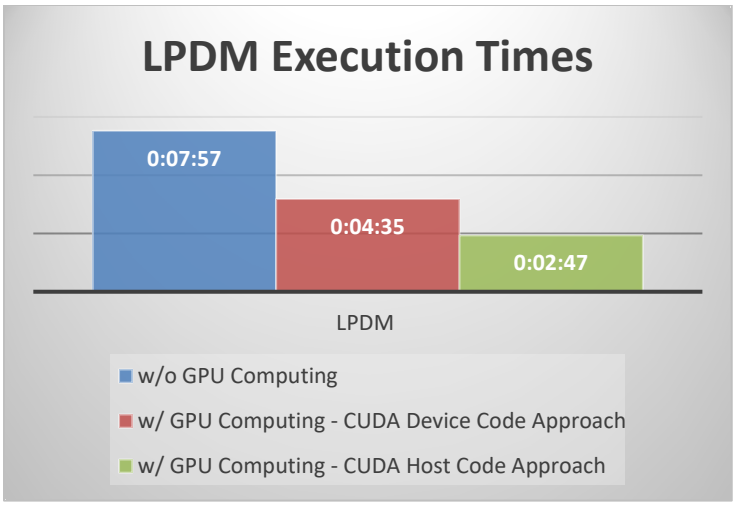
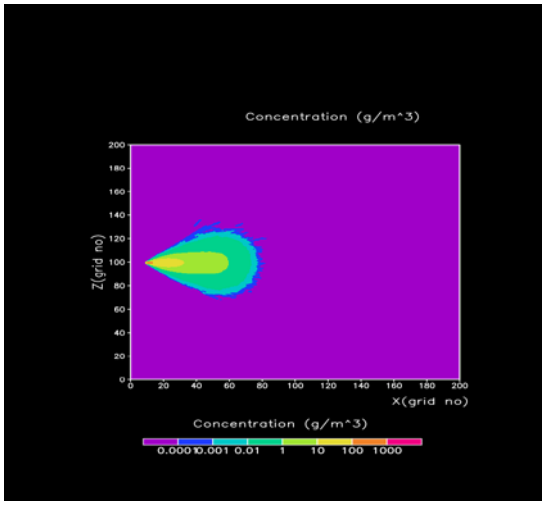


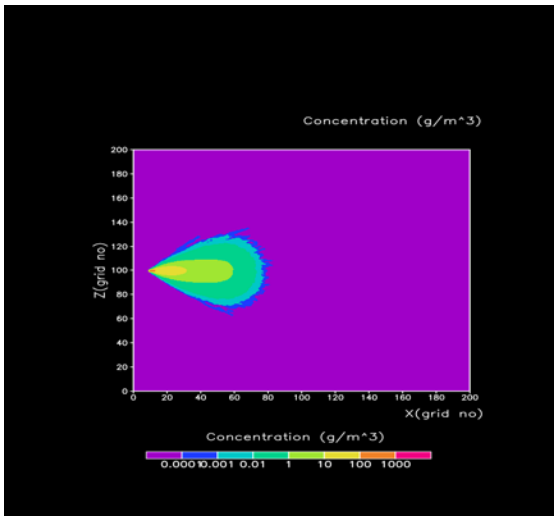
Fig. 2 Comparison of LPDM execution times without GPU computing and with the two GPU computing approaches

As shown in Fig. 3, the graphical data results simulate the average transport of aerosols and gases released under turbulent wind conditions. The data results from running the original LPDM application without GPU computing, the GPU-accelerated LPDM application using the CUDA device code approach,[5] and the GPU-accelerated LPDM application using the CUDA host code approach are almost identical, even though the execution time to produce each result varies based on the approach as shown in Fig. 2. Therefore, the second approach of the GPU-accelerated LPDM application implementing the CUDA host code was determined to be the best method in achieving optimal application performance, since it produced similar results in less execution time compared to both the original LPDM application without GPU computing and the initial approach implementing the CUDA device code.

Original LPDM w/o GPU Computing:



LPDM w/ GPU Computing – CUDA Device Code:



LPDM w/ GPU Computing – CUDA Host Code:

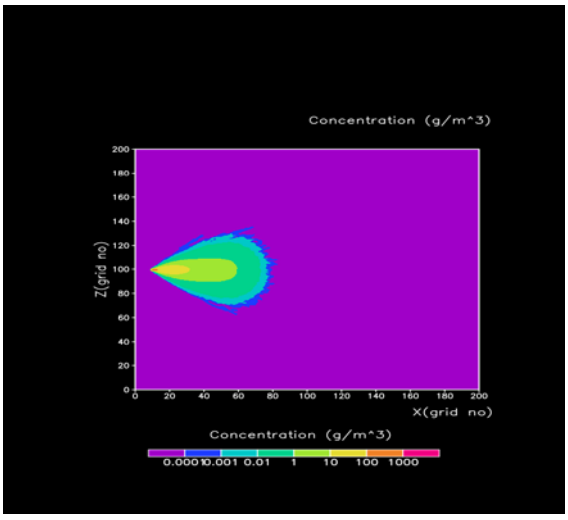


Fig. 3 Data results of original LPDM without GPU computing vs. the two GPU-accelerated LPDM approaches

5. Conclusion

The second approach to improve the performance of the GPU-accelerated LPDM application implementing the CUDA host code was proven to be a better method than the initial approach implementing the CUDA device code discussed in Dawson.[5] The goal of improving the LPDM application's performance with little

programming effort was also achieved since the second approach required less programming effort compared to the initial approach. The execution time of the GPU-accelerated LPDM application implementing the second approach using the CUDA host code was approximately three times faster than the original LPDM application without GPU computing and approximately two times faster than the GPU-accelerated LPDM application implementing the initial approach using the CUDA device code.

The experiment described in this report was the second effort in utilizing GPU computing technology with the LPDM application to increase its performance compared to the first effort discussed in Dawson.[5] Improved performance was mainly due to the implementation of a different GPU computing technique that used the CUDA host code via OpenACC along with a newer version of CUDA (i.e., CUDA 8.0 for Compute Capability 3.5). Furthermore, if the execution times of LPDM and/or other atmospheric models need to be further decreased, then additional GPU computing techniques should be explored to achieve optimal application performance in which these models could possibly be used for rapid release and planning purposes on the battlefield.

6. References

1. Hanna S, White J, Troler J, Vernot R, Brown M, Gowardhan A, Kaplan H, Alexander Y, Moussafir J, Wang Y, Williamson C, Hannan J, Hendrick E. Comparisons of JU2003 observations with four diagnostic urban wind flow and Lagrangian particle dispersion models. Atmospheric Environment. 2011;45(24):4073–4081. ISSN 1352-2310.

2. Wang Y, Miller D, Anderson D, McManus. A Lagrangian stochastic model for aerial spray transport above an oak forest. Agricultural and Forest Meteorology. 1995;76:277–291. ISSN 0168-1923.

3. Wang Y, Williamson C, Garvey D, Chang S, Cogan J. Application of a multigrid method to a mass consistent diagnostic wind model. J Appl Meteorology. 2005;44:1078–1089.

4. Wang Y, MacCall BT, Hocut CM, Zeng X, Fernando HJS. Simulation of stratified flows over ridge using a lattice Boltzmann model. Environ Fluid Mech. 2018. https://doi.org/10.1007/s10652-018-9599-3.

5. Dawson L. The performance improvement of the Lagrangian particle dispersion model (LPDM) using graphics processing unit (GPU) computing. Adelphi (MD): Army Research Laboratory (US); 2017. Report No.: ARL-TR-8110.

6. Krewell K. What is the difference between a CPU and a GPU? Santa Clara (CA): Nvidia Corporation; 2009 [accessed 2018 Aug]. https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/

7. Fortran90. Fortran90 1.0 documentation: 2018 [accessed 2018 Aug]. http://www.fortran90.org/

8. PGI compilers and tools: Fortran CUDA library interfaces guide version 2017. Santa Clara (CA): Nvidia Corporation; 2017.

9. What is OpenACC? OpenACC-standard.org; 2017 [accessed 2018 Aug]. https://www.openacc.org/.

10. cuRAND. Santa Clara (CA): Nvidia Corporation; 2017 [accessed 2018 Aug]. https://developer.nvidia.com/curand.

List of Symbols, Abbreviations, and Acronyms

CPU	central processing unit
cuRAND	CUDA Random Number Generation Library
GPU	graphics processing unit
LPDM	Lagrangian Particle Dispersion Model
OpenACC	for Open Accelerators
PGI	Portland Group Inc.
RNG	random number generator

1 (PDF) DEFENSE TECHNICAL INFORMATION CTR DTIC OCA

2 (PDF) DIR ARL IMAL HRA RECORDS MGMT RDRL DCL TECH LIB

1 (PDF) GOVT PRINTG OFC A MALHOTRA

3 (PDF) ARL RDRL CIE M L DAWSON Y WANG B MACCALL