ARL-TR-8110 ● AUG 2017

US Army Research Laboratory

# The Performance Improvement of the Lagrangian Particle Dispersion Model (LPDM) Using Graphics Processing Unit (GPU) Computing

by Leelinda P Dawson

## 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-TR-8110 ● AUG 2017

US Army Research Laboratory

# The Performance Improvement of the Lagrangian Particle Dispersion Model (LPDM) Using Graphics Processing Unit (GPU) Computing

**by Leelinda P Dawson**
*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) |
|---|---|---|
| August 2017 | Technical Report | December 2016–July 2017 |

**4. TITLE AND SUBTITLE**
The Performance Improvement 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

**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-TR-8110

**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**
The Lagrangian Particle Dispersion Model (LPDM) simulates the ensemble average transport of aerosols and gases in turbulent wind conditions. Many atmospheric transport and dispersion models, such as LPDM, can take multiple hours to complete execution, which makes them not useful for rapid release and planning purposes. This report documents the implementation process of integrating the LPDM code with graphics processing unit (GPU) computing technology to improve its performance. As a result, the execution time of the GPU-accelerated LPDM application was much faster than the original LPDM application.

**15. SUBJECT TERMS**
Lagrangian Particle Dispersion Model, LPDM, 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 | 18 | 19b. TELEPHONE NUMBER (Include area code) 301-394-5636 |

## Contents


List of Figures iv

1. Introduction 1

2. GPU and Development Environment 1

3. Performance Profiling 3

4. CUDA Fortran Interface 4

5. Results 6

6. Conclusion and Future Work 7

7. References 9

List of Symbols, Abbreviations, and Acronyms 10

Distribution List 11

## List of Figures

Fig. 1 Central processing unit (CPU) vs. graphics processing unit (GPU) ..... 2
Fig. 2 Performance profile of LPDM application using PGPROF profiler .... 3
Fig. 3 Original LPDM code vs. GPU-accelerated LPDM code with compiler output ..... 5
Fig. 4 LPDM execution times without and with GPC computing ..... 6
Fig. 5 Original LPDM vs. GPU-accelerated LPDM data results ..... 7

## 1. Introduction

Atmospheric transport and dispersion models are used to predict downwind hazards associated with the release of hazardous materials.[1] However, many of these models may take many hours to run a single simulation on the computer's central processing unit (CPU), which makes them not useful in emergency response situations. In order for these models to be useful for rapid release and planning purposes in the battlefield, their applications and simulations must be performed in a short time frame.

The Lagrangian Particle Dispersion Model (LPDM) is an atmospheric transport model that simulates the ensemble average transport of aerosols and gases in turbulent wind conditions.[2] Both the mean wind advection and the turbulent diffusion processes are considered in this model. The diffusion by the turbulent wind is simulated with a Gaussian random process, which is the most intensive portion of the model computation. Consequently, this can cause the model to take a long time to produce a single simulation. Therefore, graphics processing unit (GPU) technology was explored and utilized to increase the performance of the LPDM's code so simulations could be performed in a shorter time period, thus becoming more useful on the battlefield in the future.

GPU computing is the usage of a GPU together with a CPU to accelerate compute-intensive software applications. The purpose of this document is to describe the implementation process of integrating GPU computing technology with the LPDM code.[2] During the experiment, 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 was much faster than the original LPDM application.

## 2. GPU and Development Environment

In GPU computing, the compute-intensive portions of the application are offloaded to the GPU, while the remainder of the application code runs on the CPU.[3] As shown in Fig. 1, the CPU consists of a few cores, whereas the GPU has a massive parallel architecture consisting of thousands of cores designed for processing multiple processes or tasks simultaneously and more efficiently. As a result, this allows a GPU-accelerated application to run much faster than a CPU application.

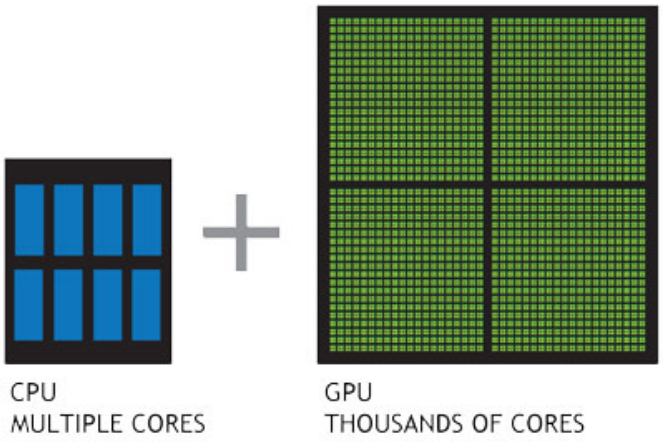


**Fig. 1 Central processing unit (CPU) vs. graphics processing unit (GPU)[3]**

The code of the original LPDM application was developed in Fortran 90.[4] Therefore, the Portland Group Inc. (PGI) Fortran compiler version 16.10.0 was used to compile the LPDM code during the experiment. One of the main goals was to incorporate GPU computing technology into code with few modifications to the original code. The US Army Research Laboratory's (ARL) high-performance computer, Excalibur, located at the ARL Department of Defense Supercomputing Resource Center (DSRC), was used for its GPU computing capability during the experiment. It has Nvidia Tesla K40 GPU accelerators containing 32 GPU nodes consisting of 1024 cores.

CUDA is a parallel computing platform and application programming interface (API) model that was created and designed by Nvidia to give direct access to the GPU for general purpose processing.[5] CUDA is designed to work easily with multiple programming languages, including Fortran. CUDA is a heterogeneous computing platform in which both the CPU and GPU are used, where the host refers to the CPU and its memory and the device refers to the GPU and its memory. The host manages memory for both itself and the device. It also launches kernels, which are subroutines executed on the device, and these kernels are executed by many GPU threads in parallel. During this experiment, the GPU-accelerated LPDM code used CUDA 7.5 for Compute Capability 3.0 with 16 GPU blocks and 128 threads per block. Thus, there were 2048 active threads executing in parallel on Excalibur, which assisted in improving the code's performance.

CUDA also supports simple programming frameworks like OpenACC (for open accelerators). OpenACC is a programming standard designed to simplify parallel programming for CPU/GPU systems with little programming effort.[6] OpenACC provided the necessary framework to parallel the LPDM code with very few

modifications to the original code by simply using compiler directives. CUDA and OpenACC were successfully utilized to integrate GPU computing into LPDM code, which will be discussed in the next sections of this document.

## 3. Performance Profiling

The first and most important step in trying to parallelize the LPDM code was to assess and identify the locations or functions in the code responsible for the bulk of the execution time. In other words, a performance profile of the application was needed to determine the hotspots and bottlenecks to maximize success. Otherwise, too much time would have been spent on parallelizing the wrong parts of the code with no improvements in its performance.

PGI's PGPROF profiler was used to determine which parts of the LPDM code would benefit the most from GPU acceleration. This profiler is a component of the PGI accelerator compiler located on Excalibur. The baseline LPDM application used during the experiment took approximately 8 minutes to complete execution on the CPU. As shown in Fig. 2, the Gaussian process function, *gaussdev*(), located in the LPDM code is taking most of the execution time according to PGPROF profiler. There are multiple recursive calls to this function in the code, and therefore it was determined that this function would benefit the most from GPU acceleration.

```
excalibur10> pgf90 -fast -Minfo=all -Mprof=ccff lagmod.f90
excalibur10> pgprof./a.out
======== CPU profiling result (bottom up):
Time(%)      Time  Name
 36.81%  187.716s  MAIN_
 36.81%  187.716s  | main
 19.34%  98.6435s  __fss_log_fma3
 19.34%  98.6435s  | __gss_log
 19.34%  98.6435s  |   gaussdev_
 17.44%  88.9122s  pghpf_rnum
 16.54%  84.3316s  | gaussdev_
  0.90%  4.58063s  | main
 17.33%  88.3921s  gaussdev_
  2.27%  11.5616s  _mp_p
  2.27%  11.5616s  | gaussdev_
  1.35%  6.89095s  __c_mzero4
  1.35%  6.88094s  | MAIN_
  1.35%  6.88094s  | | main
  0.00%  10.001ms  | main
  1.24%  6.30086s  pgf90_compiled
  1.11%  5.68078s  | gaussdev_
  0.12%  620.09ms  | MAIN_
  0.12%  620.09ms  |   main
```

**Fig. 2 Performance profile of LPDM application using PGPROF profiler**

## 4. CUDA Fortran Interface

As mentioned in Section 3, the function *gaussdev* was determined to be the hotspot for the LPDM code via the PGPROF profiler. Normal (or Gaussian) distribution is commonly used in modeling natural phenomena accurately. It was discovered in further investigation of this function that it used Fortran's built-in random number generator (RNG) to generate millions of pseudorandom numbers from the uniform distribution and utilized an algorithm to convert the numbers to the normal distribution. However, Fortran's built-in RNG is incompatible with the GPU computing framework, CUDA. Nevertheless, CUDA provides a built-in RNG library called CUDA Random Number Generation library (cuRAND) that produces high-performance GPU-accelerated random number generation.[7] cuRAND provides an interface to generate random numbers all at once from within a CUDA function or kernel running on the GPU from multiple RNG distributions, such as normal and uniform distributions.

It was determined through further investigation that the best solution would be to utilize cuRAND library via OpenACC to parallelize the code and generate millions of pseudorandom numbers from the normal distribution, since the *gaussdev* function is a hotspot in the original LPDM application and its RNG is incompatible with CUDA framework. In addition, the goal of minimizing the programming effort was met by using OpenACC. During the experiment, a CUDA Fortran interface was developed that connects the CUDA code to the GPU-accelerated LPDM application via OpenACC. The code contains a CUDA Fortran device function or kernel called *random_cuda*, which implements cuRAND's built-in normal distribution RNG to generate pseudorandom numbers with mean of 0.0 and standard deviation of 1.0 on the GPU and stores them into a parameter.

The CUDA Fortran interface consists of the following 3 source code files: *lagmod.f90*, *called_function.cu*, and *called_function.f90*. The first file, *lagmod.f90*, is the main LPDM Fortran code, which was slightly modified to make OpenACC calls to the CUDA device function, *random_cuda*, as shown in Fig. 3. The second file, *called_function.cu*, contains the actual implementation of the CUDA device function, *random_cuda*, which contains cuRAND initialization and normal distribution function calls (i.e., *curand_init* and *curand_normal* respectively). The last file, *called_function.f90*, contains the CUDA Fortran interface module, which interfaces the main LPDM Fortran application with the CUDA device function code. All of these files were used and linked to generate a GPU-accelerated LPDM executable capable of running on Excalibur and utilizing its GPU nodes.

Original LPDM code:

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

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

GPU-accelerated LPDM code:

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

!$acc routine (random_cuda) seq
integer:: seed = 1234
real, dimension(:), allocatable :: r
integer :: n = 1000000
allocate(r(n))

!$acc kernels copyout (r)
!$acc loop
do i = 1, n
  call random_cuda(seed, n, r)
enddo
!$acc end kernels

aa=r
```

Compiler output of GPU-Accelerated LPDM code:

```
pgfortran -acc -Mcuda -Minfo=accel -ta=nvidia:cuda7.5,cc30  -c
lagmod.f90 -o lagmod.o /opt/pgi/16.10.0/linux86-64/16.10/lib
lagmod.f90:
lagmod:
    157, Generating copyout(r(:))
    159, Loop is parallelizable
         Accelerator kernel generated
         Generating Tesla code
        159, !$acc loop gang, vector(128) ! blockidx%x threadidx%x
```

**Fig. 3 Original LPDM code vs. GPU-accelerated LPDM code with compiler output**

As depicted in Fig. 3, the original Fortran LPDM code was slightly modified to include GPU acceleration by using the CUDA Fortran interface described above. Note that this figure only shows parts of the original code that were modified to include GPU acceleration where there was a call to the *gaussdev* function in an iterative *for* loop. Since there were multiple calls to the *gaussdev* function in the code, a similar technique was used for each call of this function to further accelerate the application. However, the remainder of code was unchanged, which means that the majority of the code was still being executed on the CPU (host) and only these compute-intensive sections of the code were being executed on the GPU (device).

The typical sequence of operations for accelerating the LPDM application through the GPU is as follows:

1. Initialize host variable, *aa*.
2. Declare and allocate GPU memory for the 3 device variables: *seed, n, r*.
3. Call the CUDA device function, *random_cuda*, recursively to perform the transfer of data and execute the kernel.
4. Transfer the data results from the device to the GPU variable, *r*, using OpenACC's *copyout* feature. Then this variable is copied to the host variable, *aa*.

## 5. Results

As mentioned in Section 4, the CUDA Fortran interface was developed and used to generate a GPU-accelerated LPDM executable capable of running on Excalibur using its GPU nodes. In order to use the GPU nodes on Excalibur, a job script or Portable Batch System (PBS) batch script, *lagmod_interface.pbs*, was developed, so a job could be submitted to its queue to request the necessary computing resources to successfully run the executable.

The original LPDM application ran on the CPU with the execution time of 7 min and 57 s, while the GPU-accelerated LPDM application ran with the execution time of 4 min and 35 s, as depicted in Fig. 4. The GPU-accelerated LPDM code is running approximately 1.73 times faster than the original LPDM code. This proves that GPU computing technology can improve the performance of the LPDM code.

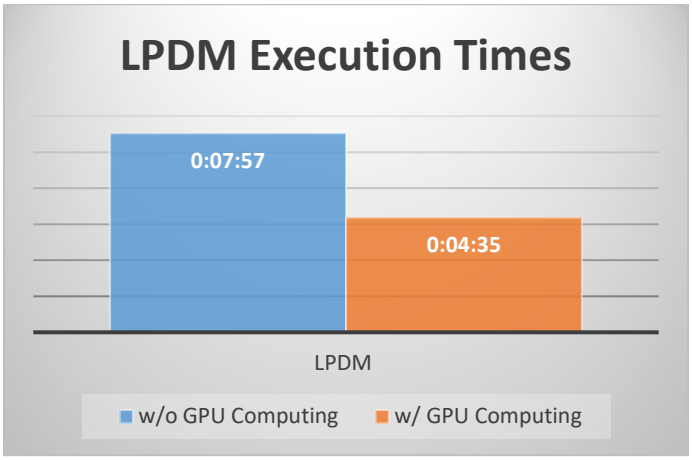


**Fig. 4 LPDM execution times without and with GPC computing**

Furthermore, the data results from running both the original LPDM and GPU-accelerated LPDM applications are fairly different when the RNG uses the same random seed but they are similar when the RNG uses different random seeds, as shown in Fig. 5. The results simulate comparable concentration levels of the average transport of aerosols and gases when they are released under turbulent wind conditions. However, there were some differences in the first data result due to the fact that the random seeds in the RNG for the GPU-accelerated LPDM application were the same, whereas they were different in the RNG for the original LPDM application. Therefore, different random seeds for the RNG were utilized by the GPU-accelerated LPDM application to produce the second data result shown in Fig. 5. This can possibly cause an increase in the execution time of the application. Consequently, more investigation will be needed to further optimize the GPU-accelerated LPDM application in the future since the second data results are almost identical to the results of the original LPDM application. In addition, there might be some fundamental differences in how both Fortran's RNG and CUDA's RNG generate their pseudorandom numbers using the normal and uniform distributions. As a result, this can also be the cause for the slight differences in the data results.

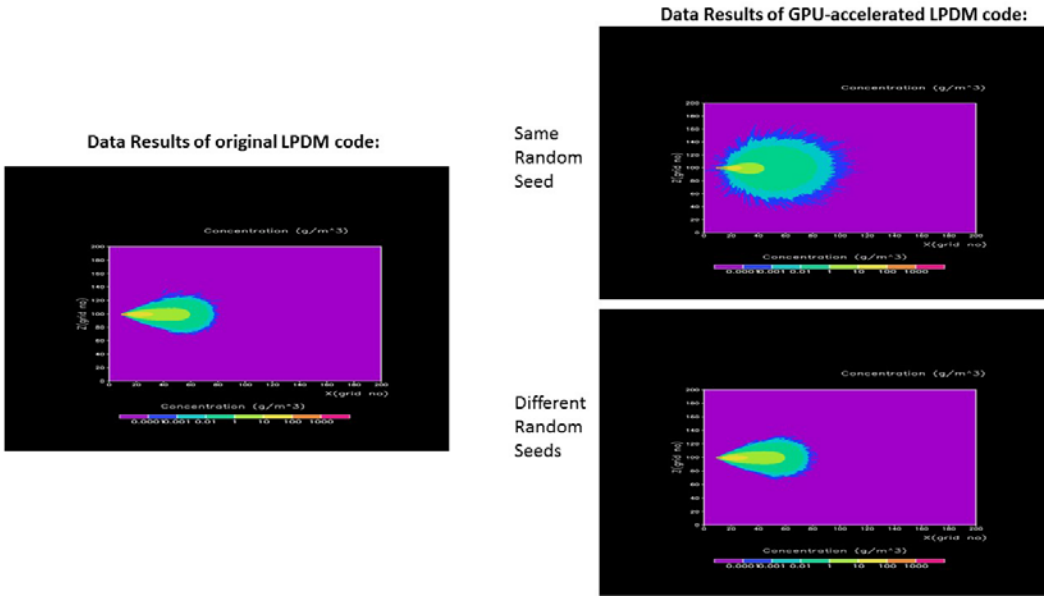


**Fig. 5 Original LPDM vs. GPU-accelerated LPDM data results**

## 6. Conclusion and Future Work

The process of integrating GPU computing technology with the LPDM application was largely successful. The main goals of improving the code performance with little programming effort was achieved by utilizing a CUDA Fortran interface via

OpenACC. The execution time of the GPU-accelerated LPDM application is approximately twice as fast as the original LPDM application.

The experiment described in this document was the first effort to integrate GPU computing technology with the LPDM application. Even though some performance improvements were achieved, other techniques for performance tuning will need to be investigated to further speed up the GPU-accelerated LPDM application in the future, especially once different random seeds in the RNG are utilized by the application (as mentioned in Section 5). These techniques include exploring other CUDA and OpenACC methods, reducing the data transfer latency between the CPU and GPU, and possibly increasing the number of threads per block on the GPU. Also, since Excalibur only has CUDA 7.5 available to use at this time, it would be good to try a newer version of CUDA in the future to determine if there is further performance improvement. In addition, the concept of using GPU computing technology to improve the execution time of atmospheric models could possibly be applied to many other models, such as the Atmospheric Boundary Layer Environment (ABLE) model, in the future. More importantly, the experiment described in this document demonstrates that GPU computing technology could possibly provide a method for significantly decreasing the execution times for various atmospheric and weather models, which could be used for rapid release and planning purposes in the battlefield.

## 7. 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 M. A Lagrangian stochastic model for aerial spray transport above an oak forest. Agricultural and Forest Meteorology. 1995:76:277–291, ISSN 0168-1923.

3. GPU vs. CPU? What is GPU computing? Santa Clara (CA): Nvidia Corporation; 2017 [accessed July 2017]. http://www.nvidia.com/object/what-is-gpu-computing.html.

4. Fortran. Wikipedia, the free encyclopedia; 2017 [accessed 2017 July]. https://en.wikipedia.org/wiki/Fortran.

5. What is CUDA? Santa Clara (CA): Nvidia Corporation; 2017 [accessed 2017 July]. http://www.nvidia.com/object/cuda_home_new.html.

6. What is OpenACC? OpenACC-standard.org; 2017 [accessed 2017 July]. https://www.openacc.org/.

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

## List of Symbols, Abbreviations, and Acronyms

| | |
|---|---|
| ABLE | Atmospheric Boundary Layer Environment |
| API | Application Programming Interface |
| ARL | US Army Research Laboratory |
| CPU | central processing unit |
| cuRAND | CUDA Random Number Generation library |
| DSRC | Defense Supercomputing Resource Center |
| GPU | graphics processing unit |
| LPDM | Lagrangian Particle Dispersion Model |
| PBS | Portable Batch System |
| PGI | Portland Group Inc. |
| RNG | random number generator |

| | |
|---|---|
| 1 (PDF) | DEFENSE TECHNICAL INFORMATION CTR DTIC OCA |
| 2 (PDF) | DIRECTOR US ARMY RESEARCH LAB RDRL CIO L IMAL HRA MAIL & RECORDS MGMT |
| 1 (PDF) | GOVT PRINTG OFC A MALHOTRA |
| 3 (PDF) | DIR USARL RDRL CIE M L DAWSON Y WANG B MACCALL |

INTENTIONALLY LEFT BLANK.