УДК 004.42

Вестник СибГАУ
2014. № 3(55). С. 157–161

УЛУЧШЕНИЕ РЕАЛИЗАЦИИ МЕТОДА ДИФФЕРЕНЦИАЛЬНОЙ ЭВОЛЮЦИИ НА ГРАФИЧЕСКИХ ПРОЦЕССОРАХ

М. А. Фарков[1], А. И. Легалов[1,2]

[1]Сибирский федеральный университет
Российская Федерация, 660041, г. Красноярск, просп. Свободный, 79
E-mail: mihail.farkov@gmail.com
[2]Сибирский государственный аэрокосмический университет имени академика М. Ф. Решетнева
Российская Федерация, 660014, г. Красноярск, просп. им. газ. «Красноярский рабочий», 31
E-mail: legalov@mail.ru

Проведён анализ существующих реализаций метода дифференциальной эволюции с использованием графических процессоров. Представлена модель вычислений, уменьшающая издержки, возникающие при вызове вычислительных ядер, за счёт объединения логически связанных этапов метода дифференциальной эволюции, имеющая проработанную структуру распределения памяти, направленную на объединение запросов к глобальной памяти графического процессора, и позволяющая эффективно использовать Compute Unified Device Architecture (CUDA) потоки для решения большого количества задач оптимизации.

Ключевые слова: графические процессоры, CUDA, дифференциальная эволюция.

Vestnik SibGAU
2014, No. 3(55), P. 157–161

AN IMPROVEMENT OF THE DIFFERENTIAL EVOLUTION METHOD IMPLEMENTATED ON GPU

M. A. Farkov[1], A. I. Legalov[1, 2]

[1]Siberian Federal University
69, Svobodnyi prosp., Krasnoyarsk, 660041, Russian Federation
E-mail: mihail.farkov@gmail.com
[2]Siberian State Aerospace University named after academician M. F. Reshetnev
31, Krasnoyarsky Rabochy Av., Krasnoyarsk, 660014, Russian Federation
E-mail: legalov@mail.ru

Differential evolution is a very effective numerical optimization method which applied to diverse the set of computationally intensive tasks. Due to the features of the algorithm, it is very suitable for graphics processing unit (GPU). Most of the algorithm stages can be executed independently that corresponds to the basic programming paradigm of GPU (single instruction multiple data). Besides, an algorithm has a regular memory structure for internal data. The use of GPU allows to improve the speed of the algorithm significantly. The analysis of existing implementations of differential evolution method on GPU is performed. Moreover, existing implementations of the differential evolution algorithm on GPU use several unoptimized techniques which restrict effective application of the algorithm to tasks which use multiple optimization procedures. A new computational model that improves current implementations is described. A presented model reduces kernel calls latency due to combining logically-connected kernel into a single global kernel. The algorithm allows to use computational grid which contains single computational block. This approach satisfies requirements of the differential evolution approach to size of population (from five to ten times greater than number of optimized variables) and allows to use GPU internal barrier synchronization techniques. Besides a proposed implementation has regular data allocation in the global memory of GPU that provides coalescence of requests to the slowest global memory thus all threads in warp can read and write information from global memory per single request. Moreover it allows to use Compute Unified Device Architecture (CUDA) streams in very effective manner. In fact, a proposed model can simultaneously execute as much optimization procedure as multiprocessors available on GPU and belimited only by computer capabilities restrictions.

Keywords: GPGPU, CUDA, differential evolution.

Введение. Метод дифференциальной эволюции эвристического алгоритма для выполнения глобальной оптимизации впервые был предложен в [1]. Решение задачи оптимизации некоторой целевой функции определяется набором из *NP* векторов, размерностью *D*. Каждый элемент вектора – это параметр, участвующий в процедуре оптимизации. Набор векторов называется кластером или популяцией (*V*). Начальное состояние векторов кластера определяется случайным образом в пределах пространства поиска. Алгоритм подразумевает три основные стадии: мутацию, кроссовер, выбор. Прежде всего, на каждой последующей итерации метода ($G + 1$) для каждого вектора кластера происходит формирования мутанта согласно

$$Vm_{i(G+1)} = V_{r1(G)} + F(V_{r2(G)} - V_{r3(G)}), \qquad (1)$$

где $Vm_{i(G+1)}$ – мутантный вектор для *i*-го вектора; $V_{r1(G)}$, $V_{r2(G)}$, $V_{r3(G)}$ – некоторые векторы предыдущей итерации метода, выбранные случайно; *F* – константа метода $\in [0,2]$.

После формирования мутантного вектора, с его использованием происходит формирование тестового (trial) вектора согласно

$$Vt_{ij} = \begin{cases} Vm_{ij} & rand() \le CF \\ V_{ij} & rand() > CF \end{cases}, \qquad (2)$$

где Vt_{ij} – *j*-й параметр *i*-го тестового вектора; Vm_{ij} – *j*-й параметр *i*-го мутантного вектора; V_{ij} – *j*-й параметр *i*-го вектора текущей итерации метода; *CF* – параметр метода $\in [0,1]$.

После формирования тестового вектора происходит вычисление целевой функции. Если её значение оказывается предпочтительнее, тестовый вектор замещает оригинальный вектор. Процесс выполняется итеративно для каждого вектора на каждом шаге метода.

Метод дифференциальной эволюции обладает высокой степенью параллелизма по данным, так как в пределах шага метода обработка каждого отдельного вектора является полностью независимой. Кроме того, метод имеет небольшое количество параметров. В связи с этим метод хорошо подходит для реализации с использованием массивно-параллельных архитектур, таких как графические процессоры. Существует несколько реализаций метода с использованием графических процессоров. Однако не представлено решений, достаточно полно использующих такие ресурсы графического процессора, как многопоточное исполнение вычислительных ядер, и ориентированных на параллельное решение большого количества задач оптимизации.

Существующие реализации. Первая наивная реализация метода с использованием графических процессоров представлена в [2]. В предложенном варианте каждая стадия метода дифференциальной эволюции реализуется посредством отдельного вызова вычислительного ядра. Все данные между вызовами ядер сохраняются в глобальной памяти графического процессора. Решение о прекращении процесса вычислений происходит на центральном процессоре. При этом используется одномерная сетка с большим количеством вычислительных блоков. Реализация представлена на рис. 1. В алгоритме используются следующие вычислительные ядра:

– Ядро I: первоначальная реализация кластера векторов решений метода с использованием случайно сгенерированных чисел;

– Ядро E: вычисление значения целевой функции;

– Ядро P: подготовка к выполнению мутации; определение индексов векторов, использующихся для формирования мутантных векторов;

– Ядро M: формирование мутантных векторов;

– Ядро C: выполнение кроссовера, т. е. формирование тестовых векторов;

– Ядро R: замещение в случае необходимости векторов предыдущей итерации метода соответствующими тестовыми векторами на основании вычисленного значения целевой функции.

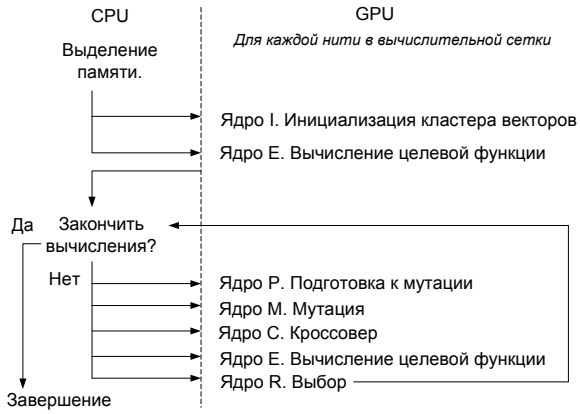

Рис. 1. Реализация метода дифференциальной эволюции на графическом процессоре

К недостаткам реализации следует отнести излишнее разделение вычислений на ядра, что не оправдано логикой метода дифференциальной эволюции, кроме того, в реализации используется предгенерация случайных чисел на центральном процессоре, что привносит дополнительные задержки, связанные с пересылкой информации между памятью центрального процессора и памятью графического процессора.

После первой реализации было представлено значительное количество работ по улучшению отдельных аспектов метода и адаптации для решения различных задач. В работах [3; 4] используется библиотека CURAND для генерации случайных чисел в процессе вычислений на графическом процессоре для исключения сложностей с генерацией случайных чисел посредством нескольких потоков центрального процессора и пересылки их в память графического процессора. Кроме того, используются возможности текстурной памяти для ускорения работы. В реализации [5] предлагается использовать предварительно сгенерированные на графическом процессоре случайные числа. Решение в целом соответствует представленным ранее. Вычислительный процесс организован

как последовательный вызов нескольких вычислительных ядер. В работе [6] графический процессор используется только для вычисления целевой функции, все остальные этапы метода дифференциальной эволюции выполняются центральным процессором. Реализация, предложенная в [7], в целом соответствует наивной версии. В вышеописанных работах делаются лишь незначительные попытки улучшить отдельные компоненты наивной реализации. Ключевыми недостатками, присутствующими во всех описанных работах, являются отсутствие динамического управления нагрузкой в зависимости от возможностей доступных графических процессоров, а также излишнее разделение логически связанных этапов метода дифференциальной эволюции на вычислительные ядра.

Более существенные изменения в оригинальный алгоритм были предложены в [8]. Авторы сократили количество вычислительных ядер путём объединения логически связанных операций метода. Таким образом, были объединены ядра I и E, а также ядра M, C, R и E (далее – ядро MCRE), что позволило более эффективно использовать разделяемую память за счёт уменьшения количества вызовов вычислительных ядер, что сказалось на производительности метода. Также авторы использовали CUDA-потоки для выполнения нескольких независимых ядер параллельно. Так, ядро, комбинированное с ядром MCRE, выполняется одновременно с ядром P. Кроме того, было добавлено автоматическое конфигурирование вычислительно сетки блоков на основании доступных ресурсов графического процессора, что является важным нововведением в свете того, что при фиксированной настройке вычислительной сетки, не согласованной с требованиями конкретной задачи, и без учёта ресурсов конкретного графического процессора нельзя правильно распределить нагрузку на мультипроцессоры, в результате чего они могут оказаться недогруженными. К существенным преимуществам метода следует отнести привнесение конфигурирования вычислений в процессе выполнения, а также тенденцию к объединению логически связанных этапов метода дифференциальной эволюции в единые вычислительные ядра. При этом следует отметить, что предложенный подход к использованию потоков не является оправданным. Генерация случайных чисел для выполнения стадии мутации метода дифференциальной эволюции не является вычислительно затратной процедурой и, как правило, не сравнима по затратам со стадией отбора, когда необходимо выполнять вычисление целевой функции для всех векторов кластера.

Таким образом, существующие реализации имеют ряд недостатков, ограничивающие вычислительный процесс. Прежде всего, в процессе функционирования алгоритма происходит большое количество вызовов вычислительных ядер, которые налагают дополнительные издержки, что является следствием другого недостатка – использования мультиблочного распределения вычислительной нагрузки. CUDA не имеет явных средств для выполнения синхронизации вычислений между блоками, поэтому для того, чтобы быть уверенным, что на очередном этапе алгоритма предыдущий этап был выполнен всеми нитями, требуется запуск вычислительных ядер и их явная синхронизация со стороны центрального процессора. Это, в свою очередь, налагает существенные ограничения на использование CUDA-потоков.

Улучшение реализации.

1. Перераспределение вычислительной нагрузки и использование нескольких CUDA-потоков. Следует отметить, что один вычислительный блок CUDA поддерживает до 1024 нитей на один вычислительный блок. Согласно рекомендации для метода дифференциальной эволюции из [1; 9; 10] следует, что оптимальный размер кластера векторов находится в пределах [5D, 10D]. Таким образом, одного вычислительного блока достаточно для покрытия задач с большим количеством оптимизируемых переменных. Кроме того, это позволяет использовать быструю, встроенную, барьерную синхронизацию в пределах одного блока [11]. Такое распределение нагрузки позволяет использовать CUDA-потоки для запуска нескольких процедур оптимизации одновременно и асинхронно, используя все возможности имеющихся мультипроцессоров на графическом процессоре [12; 13]. Это важно в случае построения программ, требующих выполнения глобальной оптимизации над большим количеством объектов, а также позволяет проще управлять вычислительной нагрузкой в процессе работы программы за счёт имеющихся в CUDA функций для асинхронной работы.

2. Объедение вычислительных ядер. Продолжая идею, предложенную в [8], также вполне возможно объединить ядро P с имеющимся основным ядром MCRE. Генерация индексов векторов для формирования мутантного вектора не является затратной процедурой, и её вынос в отдельный CUDA-поток не оправдывает вычислительные издержки, возникающие из-за вызова ядер их синхронизации, а также использования глобальной памяти для хранения информации. Таким образом, при выделении отдельного генератора случайных чисел для каждой нити можно объединить эти ядра.

3. Объединение запросов к глобальной памяти и уменьшение дивергенции варпов. Неоспоримым преимуществом метода дифференциальной эволюции в принципе, а также применительно к реализации на графическом процессоре, является регулярность структур данных, используемых методом, и небольшой объём требуемой памяти. Фактически для выполнения метода требуется только сохранять информацию о текущем значении оптимизируемого параметра для каждого вектора, его мутантного и тестового вектора, а также для значения целевой функции. Кроме того, следует отметить, что такая структура данных позволяет реализовать её размещение в глобальной памяти графического процессора таким образом, чтобы максимизировать объединения запросов к глобальной памяти, что важно для достижения существенного повышения производительности [14;

15]. Схема распределения памяти представлена на рис. 2.

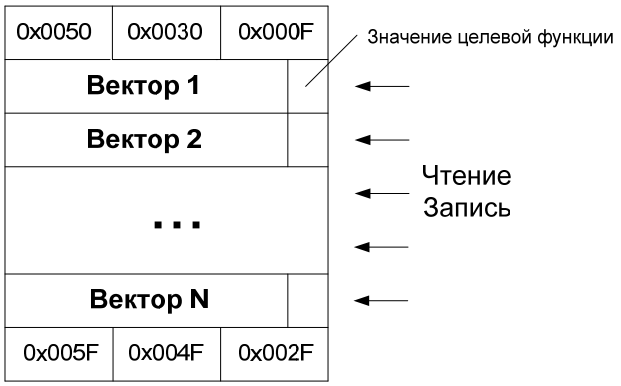

Рис. 2. Распределение памяти для кластера векторов в памяти графического процессора

Согласно такой структуре распределения данных вектора метода дифференциальной эволюции располагаются в памяти не последовательно целиком, а последовательно по каждой ячейки вектора, т. е. сначала располагаются все ячейки для сохранения значения целевой функции, затем все ячейки для хранения первого оптимизируемого параметра и т. д. При использовании интегральных или вещественных типов данных гарантируется соблюдение выравнивания данных, обращение к последовательно упорядоченным ячейкам глобальной памяти в пределах одного варпа и, как следствие, объединение запросов.

Из-за неполной линейности метода, в частности при формировании тестового вектора в зависимости от установленных параметров метода, а также на этапе выбора вектора для нового шага алгоритма, в зависимости от значения целевой функции могут возникать расхождения варпов. Это негативно влияет на производительность, так как требует сериализации вычислений, т. е. последовательного выполнения всех веток ветвления, требуемых разными нитями в варпе. Для преодоления этой проблемы следует заменять прямые операции ветвления (if…else) посредством логических вычислений. Итоговая схема предлагаемой реализации отображена на рис. 3.

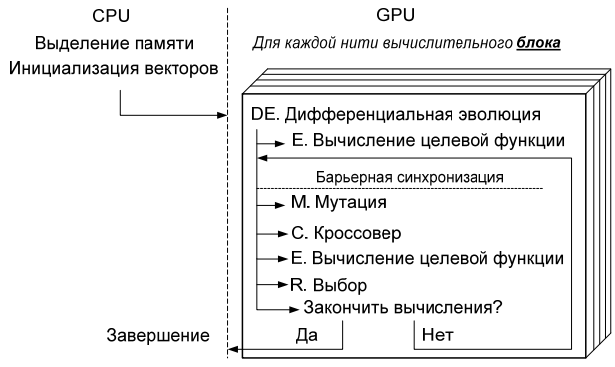

Рис. 3. Предлагаемая реализация

Заключение. В работе представлена улучшенная схема реализации метода дифференциальной оптимизации на графическом процессоре, позволяющая минимизировать издержки вызовов вычислительных ядер, использовать быструю встроенную барьерную синхронизацию и удобную для распределения нагрузки между несколькими CUDA-потоками. Всё это в комплексе делает подобную схему построения вычислительного процесса удобной для реализации программ, целью которых является выполнение большого количества операций глобального поиска.

References

1. Storn R., Price K. *Differential evolution – A simple and efficient heuristic for global optimization over continuous spaces.* Journal of Global Optimization. 1997, 11(4), p. 341–359.

2. Veronese L.P., Krohling R.A. *Differential evolution algorithm on the GPU with C-CUDA*. In Proceedings of Evolutionary Computation (CEC). 2010.

3. Gonzalez D., Barriga N. G. *Fully parallel differential evolution.* In Competition of GPUs for Genetic and Evolutionary Computation at the 2011 Genetic and Evolutionary Computation Conference. GECCO'2011.

4. Kromer P., Platos J., Snasel V., Abraham A. *A comparison of many-threaded differential evolution and genetic algorithms on CUDA*. Nature and Biologically Inspired Computing, 2011, p. 509–514.

5. Ramirez-Chavez L. E., Coello Coello C. A., Rodriguez-Tello E. A GPU-Based Implementation of Differential Evolution for Solving the Gene Regulatory Network Model Inference Problem. In Proceedings of: Proceedings of the Fourth International Workshop on Parallel Architectures and Bioinspired Algorithms, WPABA, 2011.

6. Davendra D., Gaura J., Bialic-Davendra M., Senkerik R. *GPU based enhanced differential evolution algorithm: a computational analysis.* In Proceedings of 26th European Conference on Modelling and Simulation, 2012, ISBN: 978-0-9564944-4-3.

7. Kromer P., Platos J., Snasel V., Abraham A. *Many-Threaded Differential Evolution on the GPU.* Massively Parallel Evolutionary Computation on GPGPUs, Natural Computing Series, Springer, 2013, p. 121–147.

8. Qin A. K., Raimondo F., Forbes F., Ong Y. S. *An improved CUDA-based implementation of differential evolution on GPU*. In Proceedings of the 14th annual conference on Genetic and evolutionary computation. GECCO '12, p. 991–998.

9. Price K., Storn R., Lampinen J. *Differential Evolution: A Practical Approach to Global Optimization*. Springer, 2005, ISBN: 3540209506.

10. Storn R. *On the usage of differential evolution for function optimization.* NAFIPS, 1996, p. 519–523.

11. Sanders J., Kandrot E. *CUDA by Example: An Introduction to General-Purpose GPU Programming.* Addison-Wesley. 2010, ISBN 978-0-13-138768-3.

12. Kirk D. B., Hwu W. *Programming Massively Parallel Processors. A Hands-on Approach.* Elsevier, 2010, ISBN: 978-0-12-381472-2.

13. Hwu W. *GPU Computing Gems Emerald Edition.* Elsevier, 2011, ISBN: 978-0123849885.

14. Wilt N. *CUDA Handbook: A Comprehensive Guide to GPU Programming.* Addison-Wesley. 2013, ISBN: 978-0-321-80946-9.

15. Hwu W. *GPU Computing Gems Jade Edition.* Elsevier, 2011, ISBN: 978-0123859631.