

ПАРАЛЛЕЛЬНАЯ МАТРИЧНАЯ МОДИФИКАЦИЯ МЕТОДА МУРАВЬИНЫХ КОЛОНИЙ ДЛЯ РЕШЕНИЯ ПАРАМЕТРИЧЕСКИХ ЗАДАЧ НА ГЕТЕРОГЕННЫХ ВЫЧИСЛИТЕЛЯХ

Ю. П. Титов^{1,*}, В. А. Судаков²

¹ Московский авиационный институт (национальный исследовательский университет), Москва, Россия

² Институт прикладной математики им. М. В. Келдыша РАН, Москва, Россия

Статья посвящена исследованию и разработке модификаций метода оптимизации муравьиных колоний (ACO) для повышения производительности решения сложных параметрических задач с использованием технологии параллельных вычислений на графических процессорах (GPU) с архитектурой CUDA. Предложен и исследован подход с разделением алгоритма на три части: подготовку матрицы, расчет путей муравьев-агентов и изменение весов феромонов. Использование CUDA позволяет ускорить модификацию матрицы ACO до 43 раз. Исследованы ограничения использования постоянной и локальной памяти, а также предложены алгоритмы автоматического переключения памяти в зависимости от размерности задачи, а также от количества агентов-муравьев.

The article is devoted to the study and development of modifications of the ant colony optimization (ACO) method to improve the performance of solving complex parametric problems using parallel computing technology on graphics processing units (GPUs) with the CUDA architecture. An approach is proposed and studied with dividing the algorithm into three parts: matrix preparation, calculation of ant agent paths, and changing pheromone weights. Using CUDA allows accelerating the modification of the ACO matrix up to 43 times. The limitations of using constant and local memory are studied, and algorithms for automatic memory switching are proposed depending on the problem dimension, as well as on the number of ant agents.

PACS: 89.20.Ff

ВВЕДЕНИЕ

Метод оптимизации, основанный на поведении муравьиной колонии, впервые был предложен как Ant Algorithms [1], позже расширенный до Ant System [2]. Обобщенный метод муравьиных колоний (Ant Colony Optimization — ACO) использует муравьев-агентов, каждый из которых вычисляет некоторый путь в графе с целью определения пути ком-

* E-mail: kalengul@mail.ru

мивояжера. Модификации алгоритма АСО успешно реализуются в параллельном исполнении благодаря двум основным видам параллелизма: крупнозернистому (итерации или целые колонии) и мелкозернистому (муравьи-агенты). Крупнозернистый подход представлен модификацией P-ACO, разработанной в 2002 г., а значительный прогресс произошел после появления архитектуры CUDA в 2007 г., позволившей реализовать вариант MMAS (Max–Min Ant System) на GPU, известный как GMMAS [3].

В работе [4] предложена методика PartialACO, которая снижает вычислительные расходы путем обновления лишь отдельных маршрутов, сохраняя важные данные предыдущих шагов. Концепция Alternating Updates в ACS-GPU-Alt [5] заключается в разделении графа на две зоны с обновлением феромонов попеременно, что уменьшает конфликты доступа к памяти и повышает эффективность GPU. Алгоритм MCACO [6], также разработанный на CUDA, запускает несколько колоний муравьев одновременно, каждая из которых работает независимо, увеличивая пространство поиска решений.

Параллельные версии MMAS показали значительное увеличение производительности, при котором достигается ускорение в 21 раз и генерируются миллионы решений в секунду для средних задач TSP [7]. А в работе [8] представлена стратегия Static–Dynamic Balanced Candidate Set Strategy (ID-CS), оптимизирующая распределение работы между потоками GPU, обеспечивающая улучшенную производительность и уменьшающая количество глобальных обращений к данным.

Исследования матричных модификаций АСО, предназначенных для решения задач параметрической оптимизации, показали их эффективность по сравнению с другими метаэвристическими методами [9]. Разработана специальная графовая структура, позволяющая определять значения параметров (параметрическая матрица), и предложен алгоритм автоматического формирования параметрической матрицы оптимальной структуры [10]. Матричный алгоритм разделяется на этапы: вычисление матрицы значений функции распределения; вычисление путей муравьев-агентов; добавление и испарение весов феромона. Каждый этап может выполняться на различных вычислителях: процессорах SIMD с применением инструкций SSE/AVX; ускорителях SIMT на ядрах CUDA или с применением технологии MPI.

МАТРИЧНЫЕ МОДИФИКАЦИИ АСО

Классическая параллельная модификация широко изучена как крупнозернистая модификация и имеет ряд недостатков: ограниченность ресурсов, ядер центрального процессора, сложности в синхронизации и необходимость использования механизмов IPC. Переход к матричной оптимизации, в которой работа модификаций алгоритма описана различными операциями над матрицами, позволит увеличить эффектив-

ность параллельного выполнения алгоритма. Оптимизирующий компилятор C++ матричные операции, обычно выполняющиеся в циклах, преобразует в оптимизированный под процессор код, применяющий векторные инструкции SSE/AVX.

Модификации ACO, использующие инструкции SSE/AVX, предполагают фиксированную размерность всех матриц, кратную 4 или 8 в зависимости от типов данных (https://github.com/kalengul/ACO_SIMD). Оптимальные матрицы, получаемые в результате разложения набора значений параметров, имеют количество строк, равное 5, а количество столбцов определяется количеством слоев/параметров. Применение специальных матриц, снижающих эффективность однопоточного алгоритма, но позволяющих применять инструкции SSE/AVX, помогает существенно сократить количество циклов. Разработаны специальные матрицы, содержащие избыточные вершины, но позволяющие на этапе нормализации матриц обойтись без вложенных циклов за счет выполнения цикла по всем столбцам матриц. Подобная модификация позволяет обеспечить 5% прироста к скорости выполнения программы без снижения качества поиска. При этом применение матричной модификации с инструкциями SSE/AVX и ускорением циклов при помощи OpenMP на процессорах 13th Gen Intel Core i5-13600K, 12th Gen Intel Core i5-12450H, 9th Gen Intel Core i5-9400F, 8th Gen Intel Core i5-8300H обеспечивает ускорение до 13 раз по сравнению с классическим алгоритмом.

Применение матричной модификации при выполнении вычислений на ускорителях SIMD с применением технологии CUDA позволяет осуществить параллельные вычисления на множестве потоков, разделенных на блоки и нити. Исследованы три варианта организации матричной модификации ACO с применением технологии CUDA: итерация полностью выполняется на CPU (MatrixACO_Layer); каждый этап матричной модификации выполняется на GPU отдельно с возвратом управления по окончании этапа на CPU (MatrixACO_Basic); объединение этапов работы с матрицами и отдельное выполнение поиска путей муравьями-агентами (MatrixACO_Optimal). Сравнение времени работы алгоритмов приводится по результатам оценки математического ожидания и доверительного интервала (95 %) по результатам 200 прогонов при проведении 500 итераций 500 муравьями-агентами при поиске оптимума

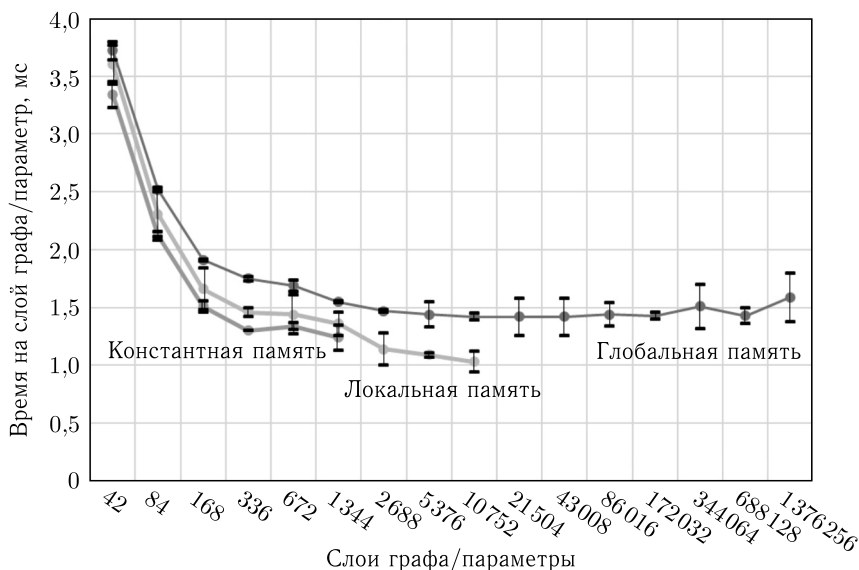
$$\text{функции Шэффера } f(x_1, x_2) = \frac{1}{2} + \frac{\sin^2\left(\sqrt{\sum_i x_i^2}\right) - 0,5}{1 + 0,001\left(\sum_i x_i^2\right)} \text{ на интервале}$$

$x_i \in [-10, \dots, 10]$ с точностью до 10^{-10} . Результаты приводятся для $i = 8$ для параметрической матрицы, где каждый параметр раскладывается на 21 подслей, образуя 168 столбцов. Вычисления производились на сервере DELL POWEREDGE C4140 Intel Xeon Gold 6130 2.1 GHz 16 Core (2Tb ОЗУ) + NVIDIA Tesla V100 16GB SXM2. Классические модификации

АСО выполняют итерации за 51,69 мс на один слой с доверительным интервалом $\pm 3,988$, а для алгоритма MatrixACO_Layer время составляет $(2,16 \pm 0,02)$ мс, что дает ускорение в 24 раза.

Для этапов, в которых выполняются матричные преобразования, возможным является алгоритм, в котором каждый блок выполняет вычисления для одного муравья-агента, а каждый поток в блоке — для отдельного параметра. В таком алгоритме нет необходимости выполнять циклы по всем параметрам. (MatrixACO_Basic = $(1,58 \pm 0,08)$ мс, MatrixACO_Optimal = $(1,58 \pm 0,08)$ мс на один слой, ускорение в 32,7 раз.) Альтернативным вариантом является алгоритм, в котором в потоках и блоках выполняются определенные действия отдельно для каждого муравья-агента, но в процессе выполняется цикл по столбцам матрицы. (MatrixACO_Basic = $(1,66 \pm 0,16)$ мс, MatrixACO_Optimal = $(1,66 \pm 0,17)$ мс на один слой.) Альтернативный вариант позволяет легко масштабироваться для решения параметрических задач большой размерности. На рисунке приведена зависимость времени поиска решений, нормализованная на один слой, от размерности параметрической матрицы и типа используемой памяти GPU.

Разработана гетерогенная модификация метода муравьиных колоний, выполняющая эффективные вычисления на персональных компьютерах и ноутбуках, содержащих GPU с поддержкой CUDA. Данная модификация выполняет этапы работы с матрицами на CPU, а вычисления путей муравьями-агентами — на GPU. Для 12th Gen Intel Core i5-12450H



Оценка времени проведения всех итераций алгоритмом MatrixACO_Optimal

2.00 GHz (16Gb ОЗУ) с NVIDIA GeForce RTX 3060 Laptop GPU классические модификации АСО выполняют итерации за $(143,52 \pm 17,26)$ мс, матричная модификация с инструкциями SSE/AVX — за $(11,5 \pm 2,30)$ мс, MatrixACO_Optimal на CUDA — за $(7,48 \pm 2,05)$ мс, а гетерогенная модификация — за $(5,78 \pm 1,64)$ мс.

ЗАКЛЮЧЕНИЕ

Проведенные исследования показали эффективность как матричной модификации АСО, реализованной с помощью инструкций SSE/AVX или на GPU с поддержкой CUDA, так и гетерогенных модификаций. Максимальное ускорение до 43 раз достигается при использовании вычислителя DELL POWEREDGE C4140 с решением 128 параметрических задач. Тем не менее гетерогенная модификация позволяет эффективно использовать матричные модификации АСО на персональных вычислителях.

СПИСОК ЛИТЕРАТУРЫ

1. *Colorni A., Dorigo M., Vittorio M.* Distributed Optimization by Ant Colonies // Proc. First Eur. Conf. on Artificial Life. 1991. V. 142. P. 134–142.
2. *Dorigo M., Gambardella L.M.* Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem // IEEE Trans. Evolution. Comput. 1997. V. 1, No. 1. P. 53–66.
3. *Bai H., OuYang D., Li X., He L., Yu H.* MAX-MIN Ant System on GPU with CUDA // 2009 Fourth Intern. Conf. on Innovative Computing, Information and Control (ICICIC). IEEE. 2009. P. 801–804.
4. *Chitty D.M.* Applying ACO to Large Scale TSP Instances // UK Workshop on Comput. Intelligence. Springer, 2017. P. 104–118.
5. *Skinderowicz R.* The GPU-Based Parallel Ant Colony System // J. Parallel Distrib. Comput. 2016. V. 98. P. 48–60; doi: 10.1016/j.jpdc.2016.04.014.
6. *Zhang D., You X., Liu S., Yang K.* Multi-Colony Ant Colony Optimization Based on Generalized Jaccard Similarity Recommendation Strategy // IEEE Access. 2019. V. 7. P. 157303–157317; doi: 10.1109/ACCESS.2019.2949860.
7. *Skinderowicz R.* Implementing a GPU-Based Parallel MAX-MIN Ant System // Future Generation Comput. Systems. 2020. V. 106. P. 277–295; doi: 10.1016/j.future.2020.01.011.
8. *Huan Z.B., Fu G.T., Fa T.H., Dong D.Y., Bai P., Xiao C.* High Performance Ant Colony System Based on GPU Warp Specialization with a Static–Dynamic Balanced Candidate Set Strategy // Future Generation Comput. Systems. 2021. V. 125. P. 136–150; doi: 10.1016/j.future.2021.06.041.
9. *Sudakov V., Titov Y.* Matrix-Based ACO for Solving Parametric Problems Using Heterogeneous Reconfigurable Computers and SIMD Accelerators // Mathematics. 2025. V. 13, No. 1284; doi: 10.3390/math13081284.
10. *Sudakov V.A., Titov Y.P.* Investigation of the Parametric Graph Model in the Ant Colony Method // Math. Models Comput. Simul. 2025. V. 17. P. 126–136; doi: 10.1134/S2070048224700996.