Параллельное и распределенное программирование, грид-технологии, программирование на графических процессорах

https://doi.org/10.25559/SITITO.021.202502.176-190

УДК 519.6

Исследование структуры гетерогенного реконфигурируемого вычислителя для эффективной работы матричного метода муравьиных колоний

Ю. П. Титов

Оригинальная статья

ФГБОУ ВО «Московский авиационный институт (национальный исследовательский университет)», г. Москва, Российская Федерация

Адрес: 125993, Российская Федерация, г. Москва, Волоколамское шоссе, д. 4

kalengul@mail.ru

Аннотация

Развитие современных вычислительных систем направлено на увеличение количества вычислителей и ускорителей, в том числе и разнородных: матричных ускорителей, MIMD, SIMD и др., что требует модернизации вычислительных алгоритмов с учетом структуры компьютера. В работе рассматривается матричная модификация метода муравьиных колоний (ACO) для решения параметрической задачи. Для решения параметрической задачи, поиска оптимальных значений параметров, в методе ACO применяется графовая структура данных, в которой для каждого значения каждого параметра выделяется одна вершина, а муравей-агент должен выбрать по одному значению для каждого параметра. Данная структура позволяет рассматривать работу ACO как операции над матрицами. Предложенная матричная модификация, работающая с оптимизированной графовой структурой, позволяет достичь ускорения от 13 до 22 раз по сравнению с оригинальным методом при выполнении на CPU. Во многом такое ускорение обусловлено работой современного оптимизирующего компилятора C++. Предложен алгоритм представления матричного ACO для SIMD ускорителя и гетерогенного вычислителя. Проведены исследования на ускорителе с применением технологии NVIDIA CUDA, ускорение составило от 7 до 18 раз. Для применения ACO на CUDA требуется пересмотр алгоритма, разделение данных по типам памяти, правильное разделение на потоки и блоки, решения проблем синхронизации и передачи данных между CPU и GPU. Предложен алгоритм для гетерогенного вычислителя, выполняющего матричные преобразования на CPU и вычисление пути муравья-агента на GPU, который показал ускорение от 24 до 38 раз. Проведено теоретическое исследование эффективности применения гетерогенного матричного ACO на гетерогенном вычислителе, состоящем из наборов MIMD ядер и SIMD ускорителей. Предложен подход к определению предела теоретического ускорения алгоритма и оптимальная структура гетерогенного реконфигурируемого вычислителя.

Ключевые слова: метод муравьиных колоний, параллельные вычисления, параметрическая задача, SIMD, MIMD, реконфигурируемый гетерогенный вычислитель, оптимальная структура, хэш-таблица

Конфликт интересов: автор заявляет об отсутствии конфликта интересов.

Для цитирования: Титов Ю. П. Исследование структуры гетерогенного реконфигурируемого вычислителя для эффективной работы матричного метода муравьиных колоний // Современные информационные технологии и ИТ-образование. 2025. Т. 21, № 2. С. 176-190. https://doi.org/10.25559/SITITO.021.202502.176-190

Parallel and Distributed Programming, Grid Technologies, GPU Programming

Study of the Structure of a Heterogeneous Reconfigurable Computer for Efficient Operation of the Matrix Ant Colony Method

Original article

Yu. P. Titov

Moscow Aviation Institute (National Research University), Moscow, Russian Federation

Address: 4 Volokolamskoe shosse, Moscow 125993, Russian Federation

kalengul@mail.ru

Abstract

The advancement of modern computing systems focuses on increasing the number of computational units and accelerators, including heterogeneous types such as matrix accelerators, MIMD, SIMD, and others. This trend necessitates the modernization of computational algorithms to align with the underlying architecture of the computer. This paper presents a matrix modification of the Ant Colony Optimization (ACO) method for addressing parametric problems. In this context, the ACO method employs a graph data structure where each vertex corresponds to a specific value of a parameter, and the ant agent is tasked with selecting one value for each parameter. This framework allows us to interpret the workings of ACO as matrix operations. The proposed matrix modification, which utilizes an optimized graph structure, achieves speedups ranging from 13 to 22 times compared to the original method when executed on a CPU. Much of this acceleration can be attributed to the capabilities of modern optimizing C++ compilers. Additionally, we introduce an algorithm for implementing matrix ACO on SIMD accelerators and heterogeneous computing systems. Research conducted on an accelerator using NVIDIA CUDA technology demonstrated speedups between 7 and 18 times. To effectively apply ACO in a CUDA environment, it is essential to revise the algorithm, categorize data by memory types, properly organize streams and blocks, and address synchronization and data transfer challenges between the CPU and GPU. Furthermore, we propose an algorithm for heterogeneous computers that performs matrix transformations on the CPU while calculating the path of the ant agent on the GPU, achieving acceleration factors ranging from 24 to 38 times. A theoretical analysis of the efficiency of utilizing heterogeneous matrix ACO on a system comprising MIMD cores and SIMD accelerators has also been conducted. This study includes a novel approach for determining the theoretical limits of algorithmic acceleration and outlines an optimal architecture for a reconfigurable heterogeneous computer.

Keywords: ant colony optimization, parallel computing, parametric problem, SIMD, MIMD, reconfigurable heterogeneous computer, optimal structure, hash table

Conflict of interests: The author declares no conflict of interests.

For citation: Titov Yu.P. Study of the Structure of a Heterogeneous Reconfigurable Computer for Efficient Operation of the Matrix Ant Colony Method. *Modern Information Technologies and IT-Education.* 2025;21(2):176-190. https://doi.org/10.25559/SITITO.021.202502.176-190

Введение

Современное развитие и распространение параллельных и суперкомпьютерных систем позволяет использовать высокоскоростные вычисления не только для сложных научных и государственных задач, но и для частных, коммерческих вычислений отдельных фирм. Для таких вычислений все более актуальным является разработка универсальных оптимизаторов, позволяющих производить параметрическую оптимизацию (поиск оптимальных значений параметров аналитических или имитационных моделей), оптимизацию гиперпараметров и переупорядочивание наборов значений параметров, отправляемых на вычислитель. При этом оптимизационный алгоритм тоже выполняет свою работу на гетерогенном вычислителе, что требует разработки эффективных алгоритмов, разделенных на SIMD и MIMD этапы. Максимальный прирост эффективности работы обычно обеспечивается SIMD ускорителем, который выполняет одну операцию с различными наборами данных. Для таких ускорителей необходимо разрабатывать матричные алгоритмы, включающие циклы фиксированного количества итераций, с минимизацией неявных условий и переходов. Среди метаэвристических алгоритмов исследуется матричная модификация метода муравьиных колоний (ACO) [1, 2].

Эффективные модификации метода муравьиных колоний

Помимо классической задачи коммивояжера, современное применение метода муравьиных колоний исследуется для решения задач множества коммивояжеров (mTSP) [3] и задач расположения объектов (QAP) [4, 5]. Хотя оригинальный алгоритм муравьиной колонии продемонстрировал многообещающие результаты при решении задачи коммивояжера, несбалансированные механизмы исследования и эксплуатации могут приводить к стагнации, когда все муравьи следуют одной и той же траектории. В процессе работы алгоритма происходит либо исследование новых областей пространства поиска, либо эксплуатация соседних высококачественных решений [6]. Если исследование преобладает, алгоритм может углубляться в бесполезные области, а чрезмерная эксплуатация может привести к преждевременному сходимости и плохим результатам. Различия между алгоритмами ACO заключаются в том, как они управляют балансом между исследованием и эксплуатацией.

Модификации метода муравьиных колоний для достижения этого баланса обычно делятся на три группы: управление феромонами, параметризация и гибридизация [7]. К первой группе относятся различные механизмы, используемые в алгоритмах оптимизации муравьиных колоний, такие как элитизм в элитной системе муравьев (EAS), обучение следам в системе колонии муравьев (ACS), основанное на муравьях Q-обучение (AntQ), ранжирование в системе муравьев на основе рангов (RAS), ограничение следов в MMAS и вычитание феромонов в системе муравьев «лучший-худший» (BWAS) и другие [7]. Ко второй группе относятся методы, где исследование и эксплуатация поддерживается параметризацией, в которых значения параметров алгоритма ACO изменяются онлайн или офлайн. Офлайн-методы представляют собой либо схему проб и ошибок, либо схему машинного обучения [8]. Обе схемы применяются до запуска алгоритма и близки к методологии мультистарта. Онлайн-методы представляют собой предложения по изменению значений параметров во время запуска. Это может увеличить сложность и нагрузку на разработку алгоритма. К третьей группе относятся успешные комбинациях базовых техник управления феромонами с локальными алгоритмами поиска, которые значительно улучшают качество решений ACO. Применение локального поиска, такого как 2-Опт, в сочетании с ACO показало свою эффективность, а также были предложены различные гибридные подходы с использованием других методов оптимизации. Наибольшее распространение получили комбинации генетического метода и метода муравьиных колоний, а также метода муравьиных колоний и метода роя частиц [9, 10].

В исследовании рассматривается применение метода муравьиных колоний для решения задач параметрической оптимизации. В подобных задачах необходимо определить значения параметров, обеспечивающие оптимальное значения целевой функции. Если параметры устанавливаются для внешней модели, то такая задача носит название оптимизацией гиперпараметров, часто требуется не поиск точного оптимального значения, а быстрое нахождение рациональных значений параметров, обеспечивающих близкие к оптимальным значения целевой функции.

Параллельные модификации метода муравьиных колоний, выполняющиеся на CPU, GPU и с применением OpenMP

В методе муравьиных колоний муравьи-агенты перемещаются группами по K муравьев-агентов. При этом перемещении состояние графа и других общих данных не изменяется, что позволяет осуществлять перемещение параллельно. Первоначально работы по исследованию параллельных модификаций метода муравьиных колоний начались еще в конце 90-х годов и разделяли параллельные модификации ACO на мелкозернистую парализацию, когда каждый муравей-агент выполняется в своем потоке и крупнозернистую, когда в одном потоке работают отдельные колонии муравьев, взаимодействующие между собой через различные механизмы. Чаще всего мелкозернистые алгоритмы уступали крупнозернистым из-за простоты алгоритма движения одного муравья-агента и накладных расходов, связанных с запуском параллельных потоков и процессов. Что бы сократить

коммуникации между процессами, был предложен метод частично асинхронной параллельной реализации (PAPI), в котором происходит обмен информацией между колониями после фиксированного числа итераций [11]. Для крупнозернистых модификаций ACO эффективным активно исследовался подход «главный/подчиненный» для решения задачи TSP использует концепцию главного потока. Главный поток собирает решения из параллельных колоний муравьев и транслирует информацию об обновлении каждой колонии для их матриц феромонов [12]. Для ускорения вычислений на CPU применялись мелкозернистые модификации ACO с применением технологии OpenMP [13]. Технология позволяет обеспечить параллельное выполнение циклов без сложной переработки кода.

Дальнейшие исследования метода муравьиных колоний обусловились развитием вычислительной техники и появлением на GPU ядер производства NVIDIA и технологии параллельных вычислений CUDA. Этот тип параллельных ACO основан на модели параллельного обмена данными (DPAM). Основная концепция DPAM заключается в том, чтобы связать каждого муравья с группой потоков с общей памятью, например, блоком потоков CUDA. Кроме того, назначая каждый поток одному или нескольким городам, потоки в блоке могут вычислять правило перехода состояний совместно, тем самым улучшая параллелизм на уровне данных. В такой постановке исследователи рассматривают GPU вычислитель как SIMD *(Simple Instruction, Multiple Data)* ускоритель, предполагая, что каждое действие алгоритма выполняется над различными данными. Полученные результаты в модификации метода муравьиных колоний для решения задачи TSP на CUDA для современных вычислительных мощностей показывают отличные результаты, например для модификации Min-Max [14], обычно совместно с применением taboo search [15] ускорение может быть более 20 раз. Близкой к параметрической задаче является задача QAP в которой транспортный граф представляется графом назначений. При параллельной реализации на GPU последовательно выполняются этапы: обновления феромона, построения решений, вычисление стоимости решения [16]. Отдельно стоит отметить исследования автоматической оптимизации кода, реализуемого на SIMD GPU с применением дополнительных библиотек. Данные исследования показывают, что методы автоматической оптимизации близки по эффективности с ручной оптимизацией метода [17]. Возможность распараллеливания на GPU для решения проблем TSP и QAP обычно упирается в необходимость взаимодействия с общими данными [18, 19]. Другой важной проблемой при реализации модификаций на GPU является трудность реализации SIMD логики на GPU ввиду возможности выполнения на GPU MIMD вычислений с задержкой в warp отдельных потоков [19]. Отдельно стоит отметить исследование методов муравьиных колоний для решения задачи TSP на Xeon Phi [20].

Матричная модификация метода муравьиных колоний

Рассмотренные параллельные модификации ACO решают задачи TSP и QAP, в которых каждый муравей-агент выполняет определенную последовательность действий, зависящую от правил поиска решения. В работе рассматривается задача дискретной параметрической оптимизации, в которой путь муравья-агента определяет значения параметров, на основе которых с применением имитационных или аналитических моделей вычисляется значение целевой функции. Для определения пути строится параметрический граф, в котором для каждого параметра определяется множество вершин. Каждая вершина определяет дискретное значение параметра. Если параметр может принимать значения в интервале (-10,10) с шагом (точностью) 10^{-1}, то для параметра будет создана 201 вершина. Муравей-агент выбирает по одной вершине для каждого параметра. После определения значений для всех параметров вычисляется значение целевой функции и, после нахождения решений группы агентов, происходит занесение и испарение весов-феромона.

Для реализации матричной формализации метода муравьиных колоний важно обеспечить фиксированную размерность данных, включая количество вершин для каждого параметра в параметрическом графе. Это позволит использовать предопределённую последовательность операций и установить определённое количество итераций в циклах. Чтобы минимизировать максимальное количество вершин, графовый слой делится на подслои, содержащие коэффициенты аддитивной и мультипликативной свертки [21]. Рациональное разделение на подслои предполагает, что слой значений параметров разделяется на десятичные деления с указанной точностью, а десятичное деление со значениями в диапазоне от 0 до 9 разделяется на два слоя: 0; 2; 4; 6; 8 и 0; 1. В результате для параметра со значениями в интервале (-10,10) и шагом 10^{-1} будет создано 5 слоев (знаковый слой, два слоя для целых и два слоя для десятых значений). Увеличение точности до 10^{-2} потребует добавления еще 2 слоев на 2 и 5 вершины соответственно. Таким образом, максимальное количество вершин при таком подходе составит пять.

Когда метод муравьиных колоний взаимодействует с аналитическими или имитационными моделями, время, необходимое для поиска значения целевой функции, становится критически важным по сравнению с временем работы самого метода. Для сокращения времени вычисления значения целевой функции используется промежуточное хранилище данных, чаще всего в виде хэш-таблицы. В процессе вычисления маршрута муравьём-агентом происходит поиск ранее найденного маршрута в хэш-таблице, и значение целевой функции вычисляется только в том случае, если данный путь отсутствует в таблице. Были разработаны модификации метода муравьиных

колоний, такие как ACOCCyN и ACOCCyI, которые позволяют продолжить поиск решения даже если муравей-агент уже встретил это решение ранее [22, 23]. Это обеспечивает то, что каждое решение рассматривается только один раз, что значительно повышает эффективность работы алгоритма и позволяет не только находить оптимальные решения, но и продолжать поиск рациональных вариантов. Большинство модификаций метода муравьиных колоний используют эвристическую информацию. Например, для задачи коммивояжера (TSP) это может быть информация о длине дуги, а для задачи о назначениях (QAP) – стоимость назначения. Наличие такой информации существенно влияет на эффективность работы метода; без неё алгоритм может застрять на более или менее хороших решениях. Среди модификаций ACO только непрерывные модификации ACOR, основанные на гаусовских смесях обходятся без явно указания эвристической информации. В работе [24] предложена и исследована новая вероятностная формула выбора агентом значения параметра. Предложенная формула учитывает, как вес-феромон, так и количество ранее сделанных выборов значений параметров. Благодаря данной формуле алгоритм на ранних итерациях увеличивает вероятность выбора менее рассмотренных значений, что способствует более эффективному поиску решений.

В основе алгоритма ACO лежит вероятностный выбор вершин (в оригинальном алгоритме дуг для решения задач TSP) в графе. При решении задачи TSP вероятность выбора конкретной дуги определяется на основе информации о длине дуги η_{ij} (эвристической информации) и количестве весов-феромона $\tau_{ij}(t)$

$$P_{ij,k}(t) = \frac{\tau_{ij}(t)\ {}^{1}\!/\!_{\eta_{ij}}}{\sum_{l \in J_{i,k}} \tau_{il}(t)\ {}^{1}\!/\!_{\eta_{il}}}.$$

Первоначально на всех дугах вес-феромон одинаков, но после того, как группа из *K* муравьев-агентов определят путь коммивояжера *Y* происходит изменение весов-феромна в соответствии с формулой: $T(t+1) = \upsilon T(t) + Q / \sum Y$. В результате на следующих итерациях дуги, которые участвовали в коротких маршрутах коммивояжера имеют больше весов-феромона и соответственно вероятность выбора данной дуги на следующей итерации.

Матричная формализация ACO предлагает рассмотрение алгоритма в виде матричных и векторных преобразований [25]. Параметрический граф представлен вектором параметров $P = \{p_1, p_2 \ldots p_i, .. p_n\}$, матрицей значений $V = \{v_{1,i}, v_{2,i}, \ldots v_{j,i}, \ldots v_{m_i,i}\}, i = \overline{1,n}$. Все строки *i* матрицы $V = (n \times m)$ одной размерности $m = \max_i |V_i|$, если количество значений для определенного параметра (слоя или подслоя) меньше *m*, то добавляются фиктивные значения с начальным весом-феромоном равным 0. Установка начального значения гарантирует что в модификациях ACO муравьи-агенты не выберут данную вершину. Аналогично создается матрица весов-феромона $T_{norm} = (n \times m)$, матрица посещения вершин $\Theta = (n \times m)$. При такой постановке формула для определения вероятности выбора значения примет вид ($\lambda_1, \lambda_2, \lambda_3$ - коэффициенты аддитивной свертки):

$$Z = \lambda_1 T_{norm} + \lambda_2\ {}^{1}\!/\!_{\Theta} + \lambda_3\ {}^{\Theta}\!/\!_{\Theta'}, Z = (n \times m)$$
$$Z' = \sum_{j=1}^{m} Z_j, \|Z'\| = n \quad (1)$$
$$P = {}^{Z}\!/\!_{Z'}, P = (n \times m)$$

Представленные матричные преобразования состоят из умножения матрицы на число, сложения элементов матрицы и вычисления отношений матрицы и вектора, что может быть легко реализовано на матричных вычислителях. Путь муравья агента определяется путем определения значений функции распределения по вычисленным вероятностям. $F = (n \times m), F_{i,j} = \sum_{k=1}^{j} P_{i,k}$. Выбор конкретной вершины определяется методом обратной функции, для которого требуется значение равномерно распределенной случайной величины $(r_{i,k})$, которое может быть представлено вектором $\|R\| = n$, а учитывая множество агентов на одной итерации, матрицы $R = (K \times n)$, где *K* – параметр метода муравьиных колоний, определяющий количество муравьев-агентов (поколение) на итерации. Применяя метод обратной функции необходимо определить значение *s*, для удовлетворения неравенства $F_{i,s-1} \le r_{i,k} \le F_{i,s}$, для $\forall i$. Пути муравьев-агентов хранятся в матрице $X = (K \times n)$, и вычисляется вектор значений целевой функции $\|Y\| = K$.

После получения всех значений целевой функции происходит обновление феромона-весов: $T(t+1) = \upsilon T(t) + Q / \sum Y$. Несоответствие размерностей $T = (n \times m)$ и $\|Y\| = K$ разрешается путем добавления значения элемента вектора *Y* только для вершин, значений параметров, которые выбрал муравей-агент, определенных матрицей $X = (K \times n)$. Алгоритмическая запись данной процедуры может быть представлена в виде $T[X[k, j], j](t+1) = T[X[k, j], j](t+1) + Q / Y[k]$, с циклами по переменным $j = \overline{1,n}$ и $k = \overline{1,K}$.

Алгоритм матричного метода муравьиных колоний состоит из 3-х основных этапов:

1. Подготовка параметрического графа, которая заканчивается вычислением матрицы значений функций распределения *F*. На данной итерации отдельные параметры, слои и подслои вершин, не зависят друг от друга и могут быть вычислены параллельно. Цикл выполняется только по всем вершинам одного слоя параметров.
2. Поиск пути муравьями-агентами путей. Данный алгоритм может выполняться параллельно для каждого агента, так как агенты на одной итерации не взаимодействуют друг с другом и параметрический граф не изменяют. По результатам работы этапа вычисляется вектор значений целевой функции *Y* и пути муравьев-агентов *X*.
3. «Испарение» весов-феромона и добавление весов-феромона от путей агентов. Так как для каждого параметра, для каждого слоя и подслоя, муравей-агент выбрал ровно одно значение, одну вершину, то выполнение занесения и испарения феромона может выполняться параллельно для каждого значения параметра.

Цель исследования

Реализацию параллельных вычислений с применением технологий CUDA GPU или OpenMP на CPU часто называют *SIMD (Simple Instruction, Multiple Data)* вычислениями. В определённой степени — это верно, поскольку каждый поток CUDA выполняет операции кода одновременно, что исключает ситуации гонок даже при использовании общей памяти. Однако возможность выбора номера потока и выполнения конструкций вида: if (threadIdx.x==0) {выполнить код только для первого потока}, а также применение условий и циклов с неопределённым числом итераций делает невозможным одновременное выполнение инструкций. Такие фрагменты кода обрабатываются на *MIMD (Multiple Instruction, Multiple Data)* ядрах, в то время как участки кода с явным указанием номера исполняющего потока представляют собой *SISD (Single Instruction, Single Data)* фрагменты. Алгоритм матричного метода ACO, разделённый на SISD, MIMD и SIMD фрагменты, представлен на рисунке 1.

Использование гетерогенного вычислителя позволяет эффективно распределять выполнение каждого фрагмента на соответствующем вычислительном блоке, что обеспечивает максимальное ускорение при реализации алгоритма. SISD-фрагмент присутствует в любом алгоритме и, как правило, отвечает за загрузку данных и сохранение результатов, что происходит в едином экземпляре во время выполнения алгоритма. В результате SISD-фрагмент требует фиксированного времени, обычно зависящего от размерности данных. В расчётах время его выполнения остаётся постоянным, и его можно не учитывать. Ускорение SISD-фрагмента достигается с помощью FPGA-ускорителей, оптимизированных под конкретную задачу. Для каждой задачи коэффициент ускорения уникален и определяется архитектурой FPGA-модуля. Для оценки эффективности и ускорения работы матричного ACO могут быть вычислены следующие времена:

Р и с. 1. Алгоритм матричного метода ACO с разделением на SISD, MIMD и SIMD фрагменты

F i g. 1. Matrix ACO algorithm with decomposition into SISD, MIMD, and SIMD fragments

Источник: здесь и далее в статье все таблицы и рисунки составлены автором.

Source: Hereinafter in this article all tables and figures were made by the author.

- Старт алгоритма и создания необходимых структур данных *(Stage Start)*. Так как данный этап выполняется в единственном экземпляре, то его выполнение возможно только на SISD системах. При анализе эффективности работы системы время выполнение данного этапа является постоянным и обязательным, им можно пренебречь при построении гетерогенного вычислителя.
- Накладные расходы на итерации *(Stage Delt)*. Эти расходы связанны с увеличением счетчика, переключением контекста, вычислением времени вложенных этапов и т.д.
- Первый этап *(Stage 1)*, связанный с матричными преобразованиями с целью получения матрицы *F*, может полностью выполняться на SIMD ускорителях.
- Второй этап *(Stage 2)*, связанный с поиском путей муравьями-агентами может выполняться на SIMD вычислителях в случае отсутствия взаимодействия с хэш-таблицей. Данный этап лучше реализовывать с помощью MIMD компоненты или совместно SIMD ускорителем и MIMD компонентой.

- Третий этап *(Stage 3)*, связанный с обновлением матриц Θ и T состоит только из матричных преобразований и может выполняться на SIMD ускорителе полностью.
- Для матричных алгоритмов ACO, реализованных на CPU возможно отдельное вычисление частных времен: время поиска значений целевой функции (OF), время сохранения записи в хэш-таблицу *(Hash save)*, время проверки наличия записи в хэш-таблице *(Hash search)* и времени поиска позиции методом обратной функции (Search *s*). На GPU вычисление данных времен, синхронизированно с общим временем выполнения затруднено.

На SIMD ускорителе могут выполняться участки кода, которые позволяют выполнять одинаковые операции с различными данными и при этом одинаковым алгоритмом получения результата. Например, на этапах 1 и 3, в которых необходимо заполнить матрицы фиксированного размера, алгоритм можно представить последовательностью одинаковых операций для каждого потока. На втором этапе идеология SIMD нарушается при поиске позиции *s*. Даже если цикл проводить по всей размерности слоя, не завершая цикл при нахождении позиции, то результаты проверки условия $F_{i,s-1} \le r_{i,k} \le F_{i,s}$ будут различны для различных потоков. Данный цикл, как и работу модификаций ACOCCyN и ACOCCyI необходимо вычислять на MIMD ядрах. После выполнения данных операций и перед переходом на SIMD вычисления необходимо синхронизировать процессы.

Отдельно следует отметить особенности SIMD вычислений по взаимодействию с общими, для потоков, данными. Например, для выполнения вычислений на GPU с технологией CUDA может быть задано количество потоков в одном блоке и количества блоков. На этапе 2 количество блоков может быть задано равным количеству муравьев-агентов, а количество потоков – равным числу параметров. При таком подходе вычисление значений целевой функции *Y* и взаимодействие модификаций ACOCN, ACOCCyN и ACOCCyI с хэш-таблицей проводится параллельно всеми потоками одного блока с одним и тем же номером муравья агента. Однако инструкции выполняются строго параллельно, что исключает гонки данных, и результат выполнения будет полностью совпадать с результатами выполнения операции одним потоком. Но применение атомарных операций, например атомарного увеличения счетчика количества повторных найденных решений в хэш-таблицу, будет проводиться каждым потоком отдельно, подобно вычислениям на MIMD ядрах.

Целью данной работы является конфигурирование гетерогенного вычислителя, который включает в себя как SIMD, так и MIMD компоненты, для достижения максимального коэффициента ускорения при выполнении матричных модификаций ACO с учетом взаимодействия с хэш-таблицей и алгоритмами ACOCN и ACOCCyN. Это позволит оптимизировать использование ресурсов вычислительной системы и повысить общую эффективность алгоритма. Кроме того, важно провести анализ производительности на каждом этапе, чтобы выявить потенциальные узкие места и возможности для дальнейшей оптимизации.

Описание эксперимента

Для эффективной работы параллельного метода муравьиных колоний реализуется вероятностная формула (1) без учета третьего слагаемого $\lambda_3 = 0$ для задач большой размерности и с учетом третьего слагаемого для задач малой размерности [24]. Отсутствие третьего слагаемого уменьшит количество операций для вычисления формулы и позволит отказаться от вычисления и хранения матрицы Θ'. Исследуются алгоритмы с применением хэш-таблицы: ACOCNI и ACOCCyN с ограничением в 100 дополнительных итераций. Параметрическая задача обязательно декомпозируется на слои с максимальным количеством вершин в одном слое равным 5. Данная декомпозиция требуется для увеличения количества слоев, каждый из которых может быть обработан в отдельном потоке на SIMD вычислителе и уменьшении количества дополнительных вершин, требующихся для представления в виде матрицы *(n×m)*.

Для исследования эффективности работы параллельного метода муравьиных колоний разработано программное обеспечение[1] на языке C/C++ для работы на графических ядрах CUDA для GPU, произведенных фирмой NVIDIA, а также на CPU, C++v20 CUDA v12.5. На современных процессорах имеется наборы расширений для выполнения матричных инструкций для параллельной обработки данных, например: *SSE (Streaming SIMD Extensions), AVX (Advanced Vector Extensions)* и другие. Сравнение проводилось с классической реализацией метода муравьиных колоний (реализованной в виде отдельных функций в том же программном обеспечении), построенной на языке C++ с применением объектов и оптимизированной для эффективного решения тех же задач, что и матричные модификации. Также исследовалось ускорение матричного метода муравьиных колоний при применении технологии OpenMP. Так как на время выполнения CPU Bound процесса в ОС влияет не только аппаратное обеспечение, но и его загруженность процессами, то все алгоритмы запускаются из одной программы последовательно в пакетном режиме. Такой подход позволяет проводить последовательные исследования при более-менее фиксированном состоянии ОС. Между разными запусками программы нагрузка на вычислительные ресурсы может изменяться. В связи с этим основную ценность имеют не абсолютные значения времени, а относительные значения ускорения, которые устойчивы к перезапускам программы.

Отдельно стоит отметить особенности в задании генератора случайных чисел при выполнении

[1] ACO SIMD [Электронный ресурс] // GitHub, 2025. URL: https://github.com/kalengul/ACO_SIMD (дата обращения: 28.04.2025).

вычислений на CUDA. Технология CUDA предполагает использование независимых блоков и потоков с минимизацией связи по общей памяти. В отличие от матричных процессоров CPU для которых достаточно одного генератора случайных чисел со значением seed, для CUDA необходимо задание уникального начального значения seed для каждого потока при каждой передаче управления GPU. В случае совпадения последовательности псевдослучайных чисел муравьи-агенты будут осуществлять одинаковый вероятностный выбор и получать одинаковые пути. В результате резко возрастает количество совпадений решений в хэш-таблице и резко увеличивается время работы модификации ACOCCyN. Для решения данной проблемы при вычислении seed использовалась информация о номере блока (blockIdx.x), номере потока (threadIdx.x) и времени (clock64()). При этом рассматривалась модификация, в которой все три этапа матричной модификации ACO выполняются в одной процедуре CUDA, т.е. между этапами 1,2 и 3 нет возвращения управления CPU. В подобной модификации все итерации ACO выполняются на GPU и требует единовременного задания генератора случайных чисел.

Среди параметрических бенчмарков рассматривалась задача поиска оптимума функций Растригина, Розенброка, Шаффера, Акли, «Сферической» функции, Гринванка, Захарова, Шейфеля, Леви, Михалевича-Викинского, и других из работы [26]. Полученные результаты, оценки времени, ускорения и отношение отдельных этапов близки для различных функций бенчмарков, поэтому в работе результаты представлены для «функции Шаффера»

$$f(x') = \frac{1}{2} - \frac{\sin^2(\sqrt{x'}) - 0{,}5}{1 + 0{,}001x'}, x' = \sum_i x_i^2, x_i \in (-10{,}10), \forall i$$

с точностью до 10^{-9} при различных количествах параметров и решении задачи минимизации. $\lambda_1, \lambda_2, \lambda_3 = 1, Q = 1, \upsilon = 0{,}999, K = 500, N = 500$.

При декомпозиции одного параметра реализован 21 слой. При этом рассматривались задачи с 2-мя (классический бенчмарк), 4-мя, 8-ю и 16-ю параметрами. На практике 2-х критериальные задачи малой размерности встречаются нечасто и для их решения применяют более простые алгоритмы. В результате развития вычислительных систем, матричных вычислителей и технологий параллельной обработки инструкций все больше интереса вызывают задачи большой размерности, например, для задачи с 16-ю параметрами количество слоев достигает 336. Оптимизация задач сверхбольшой размерности является актуальной и выполнимой, но требует дальнейшего исследования [27, 28]. Предложенная матричная модификация зависит от вычислительных мощностей: количества ядер, максимально возможного количества SIMD потоков или количества блоков и потоков в одном блоке при вычислении с применением технологии CUDA.

Замеры временных интервалов проводились на видеокарте NVIDIA GeForce RTX 3060 Laptop GPU с архитектурой Ampere (максимальное количество потоков в одном блоке составляет 1024, 6GB оперативной памяти GDDR6) и процессоре Intel Core i5-12450H, 16Gb 2-х канальной оперативной памяти DDR4. Вычислялась оценка математического ожидания времени выполнения алгоритма и его отдельных этапов при сравнении классического метода муравьиных колоний, матричной реализации на CPU (однопоточной и с применением OpenMP), реализация на GPU

Таблица 1. **Оценка математического ожидания времени выполнения (в мс) 500 итераций 500 муравьями-агентами при поиске оптимума функции Шеффера с использованием разных модификаций ACO**
Table 1. **Estimated expected execution time (ms) of 500 iterations by 500 ant agents when searching for the optimum of the Schaffer function using different ACO modifications**

Итоговое время		Количество параметров / потоков n			
		42	84	168	336
Матричный ACO на GPU CUDA модель 1+2+3	$\tilde{M}$	704,33	961,78	2359,28	7225,25
	ДИ	±3,33	±3,22	±4,76	±12,25
Матричный ACO на GPU CUDA модель 1+2	$\tilde{M}$	692,52	965,18	2374,20	7279,21
	ДИ	±2,08	±1,60	±3,11	±8,48
Матричный ACO на GPU CUDA модель 1	$\tilde{M}$	1771,42	2585,02	10783,53	17218,85
	ДИ	±2,79	±1,04	±131,60	±127,58
Матричный ACO на CPU	$\tilde{M}$	571,80	1028,98	1937,10	3808,67
	ДИ	±8,02	±8,50	±6,73	±15,93
Матричный ACO с OpenMP	$\tilde{M}$	591,84	1061,81	1972,09	3835,72
	ДИ	±2,61	±3,63	±5,04	±11,39
Матричный ACO на гетерогенном вычислит.	$\tilde{M}$	331,72	600,03	1129,42	2153,27
	ДИ	±3,66	±3,05	±9,62	±6,02
Классический ACO	$\tilde{M}$	12820,68	17638,37	28472,06	51677,42
	ДИ	±23,00	±76,71	±37,33	±415,70

Ускорение по отношению классическому ACO		Количество параметров / потоков n			
		42	84	168	336
Матричный ACO на GPU CUDA модель 1+2+3	$\tilde{M}$	18,20	18,34	12,07	7,15
	ДИ	±0,24	±0,26	±0,05	±0,06
Матричный ACO на GPU CUDA модель 1+2	$\tilde{M}$	18,51	18,27	11,99	7,10
	ДИ	±0,17	±0,22	±0,04	±0,06
Матричный ACO на GPU CUDA модель 1	$\tilde{M}$	7,24	6,82	2,64	3,00
	ДИ	±0,02	±0,03	±0,01	±0,03
Матричный ACO на CPU	$\tilde{M}$	22,42	17,14	14,70	13,57
	ДИ	±1,01	±0,39	±0,11	±0,24
Матричный ACO с OpenMP	$\tilde{M}$	21,66	16,61	14,44	13,47
	ДИ	±0,32	±0,22	±0,08	±0,22
Матричный ACO на гетерогенном вычислит.	$\tilde{M}$	38,65	29,40	25,21	24,00
	ДИ	±2,36	±0,82	±0,77	±0,72

с технологией CUDA в трех вариантах: отдельно этапы 1 (получения матрицы *F*), 2 (поиск решений муравьев-агентов с применением хэш-таблицы) и 3 (изменения состояний матриц Θ и *T*) – модель 1+2+3; с объединением этапа 1 и 3 – модель 1+2; и при выполнении всех этапов в одном алгоритме – модель 1. Результаты вычисления времени для алгоритма ACOCCyN и доверительные интервалы для доверительной вероятности равной 0.95, полученные по 100 прогонам приведены в таблице 1.

По результатам исследований матричное представление метода муравьиных колоний обеспечивает ускорение в 7-18 раз при использовании технологии CUDA и в 13-21 раз при выполнении алгоритма на CPU. При этом наилучшее ускорение показывает матричный однопоточный ACO на CPU, обгоняя в том числе и матричный ACO на CPU с применением ускорения технологией OpenMP. Это связано с подключением оптимизирующего компилятора C++, позволяющего выполнять матричные преобразования с задействованием все мощностей компьютера в том числе и инструкций SSE/AVX.

Весомый вклад в работу матричного ACO вносит этап поиска пути муравьями-агентами, так как он менее предсказуемый. На рисунке 2 представлена доля времени, затрачиваемого на выполнение 2-го этапа (вычисление матриц *R*, *X*, вектора *Y*, работа модификации ACOCCyN [22]) по отношению к общему времени работы модификаций с применением хэш-таблицы (a) и без нее (b). Для алгоритмов, не прибегающих к хэш-таблицам, второй этап сводится лишь к определению пути муравья-агента и вычислению значения целевой функции. Эти процессы, как правило, выполняются с высокой скоростью благодаря параллельному вычислению. Однако с увеличением размерности задачи наблюдается заметный рост доли времени, затрачиваемого на вычисление путей агентов, что указывает на усложнение задачи. В то же время внедрение хэш-таблицы, даже если рассматривать лишь операции проверки отсутствия записи и добавления новых данных, значительно увеличивает общее время выполнения 2-го этапа. На основании проведенных исследований можно сделать вывод о том, что для алгоритмов, работающих без использования хранилищ данных, необходимо оптимизировать взаимодействие с матрицами *T* и Θ. В случае применения хэш-таблицы акцент следует сделать на оптимизации этапа взаимодействия с этой структурой данных и 2-го этапа.

Так как для матричной модификации ACO существенные задержки вызваны поиском пути агентами, то было предложена модификация, выполняющаяся на гетерогенном вычислителе. В данной модификации 1-ый и 3-ий этапы из-за матричной природы вычислений выполняются на CPU с применением оптимизирующего компилятора для C++, транслирующего код в инструкции SSE/AVX для работы с матрицами, а поиск пути муравьем-агентом (этап №2) выполняется на GPU с применением технологии CUDA. Для 2-го этапа в качестве входных данных требуются матрицы *V*, *T* и *F*, а в результате выполнения получается матрица *X* и вектор *Y*. Генерация матрицы *R* (отдельных ее элементов) производится с применением функций GPU. Такой подход показал наибольшую эффективность, коэффициент ускорения от 24 до 38 раз. При подробном исследовании времени второго этапа определено, что большую часть, около 80-90% занимает копирование данных и передача управления.

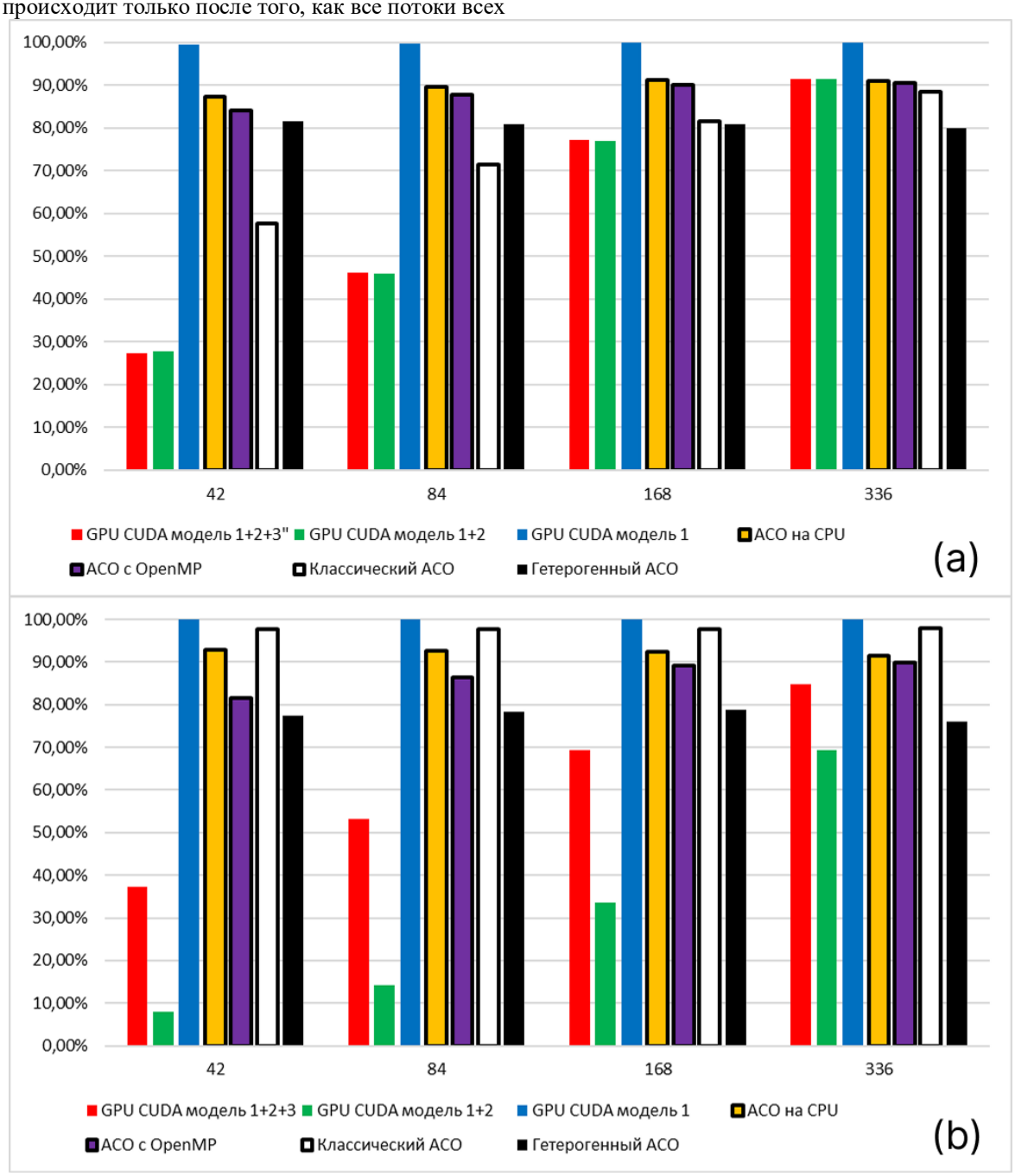
Исследование времен работы модификаций ACO с применением хэш-таблицы и без нее показала, что для классического ACO и матричного ACO хэш-таблица не дает существенных задержек так как муравьи-агенты в данных алгоритмах, определяют уникальный набор значений на каждой итерации и взаимодействие с хэш-таблицей сводится к проверке существования решения и добавления нового значения (Таблица 2).

Тем не менее переключение с SIMD вычисления на MIMD с необходимостью проводить проверку наличия решения в хэш-таблицах приводит к накладным расходам на GPU (более 2-х раз для модификаций на GPU CUDA). В то время как на CPU, с наличием

множества различных преобразователей на одном кристалле ядра, данное переключение не приводит к существенной потере времени.

Исследование оптимальной структуры гетерогенного вычислителя

При выполнении вычислений на платформе CUDA главный поток CPU находится в состоянии ожидания (блокировки), пока не завершатся все операции, выполняемые на GPU. Завершение работы функций на GPU происходит только после того, как все потоки всех блоков успешно выполнили заданные задачи, что создает эффект глобальной барьерной блокировки. К тому же переключение контекста и передача управления, как и последующая передача данных является трудоемким вычислительным процессам и приводит к задержкам. В отличие от традиционных архитектур, где CPU и GPU функционируют поочередно, передавая управление, гетерогенные вычислители представляют собой более сложные системы, состоящие из множества компонентов SIMD и MIMD, а также SISD-компоненты.



Р и с. 2. Гистограммы отношения времени выполнения 2-го этапа матричного алгоритма ACO к общему времени выполнения матричного алгоритма ACO для модификаций с использованием хеш-таблицы (а) и без нее (б)

F i g. 2. Histograms of the ratio of the execution time of stage 2 of the matrix ACO algorithm to the total execution time of the matrix ACO algorithm for modifications with a hash table (a) and without a hash table (b)

Т а б л и ц а 2. **Ускорение алгоритма без использования хеш-таблицы из 500 итераций 500 муравьями-агентами при поиске оптимума функции Шеффера на графах разной размерности с использованием матричных модификаций ACO**

T a b l e 2. **Speedup of the algorithm without using a hash table for 500 iterations by 500 ant agents when searching for the optimum of the Schaffer function on graphs of different sizes using matrix ACO modifications**

Ускорение ACO без применения хэш-таблицы по сравнению с модификацией ACOCCyN		Количество параметров / потоков n			
		42	84	168	336
Матричный ACO на GPU CUDA модель 1+2+3	$\tilde{M}$ ДИ	0,89 ±5,5x10^{-3}	0,88 ±3,3x10^{-3}	1,35 ±6,0x10^{-3}	1,94 ±1,3x10^{-2}
Матричный ACO на GPU CUDA модель 1+2	$\tilde{M}$ ДИ	1,29 ±7,0x10^{-3}	1,59 ±1,1x10^{-2}	2,98 ±1,8x10^{-2}	3,94 ±2,6x10^{-2}
Матричный ACO на GPU CUDA модель 1	$\tilde{M}$ ДИ	1,08 ±1,9x10^{-3}	1,15 ±7,5x10^{-4}	2,30 ±6,6x10^{-2}	1,79 ±2,4x10^{-2}
Матричный ACO на CPU	$\tilde{M}$ ДИ	1,26 ±2,5x10^{-2}	1,15 ±1,4x10^{-2}	1,10 ±5,8x10^{-3}	1,07 ±9,5x10^{-3}
Матричный ACO с OpenMP	$\tilde{M}$ ДИ	1,15 ±6,9x10^{-3}	1,11 ±5,2x10^{-3}	1,09 ±3,9x10^{-3}	1,08 ±4,4x10^{-3}
Матричный ACO на гетерогенном вычислит.	$\tilde{M}$ ДИ	1,18 ±2,4x10^{-2}	1,17 ±1,2x10^{-2}	1,20 ±1,5x10^{-2}	1,20 ±5,4x10^{-3}
Классический ACO	$\tilde{M}$ ДИ	1,76 ±7,3x10^{-3}	1,41 ±1,3x10^{-2}	1,22 ±3,2x10^{-3}	1,13 ±1,4x10^{-2}

Это разнообразие архитектурных элементов позволяет разбить алгоритм на отдельные этапы и выполнять вычисления параллельно, что существенно снижает задержки при доступе к общей памяти и обеспечивает эффективную синхронизацию между MIMD-ядрами и SIMD-ускорителями. Создание оптимизированной структуры гетерогенного вычислителя является ключом к достижению максимального коэффициента ускорения алгоритма, особенно в контексте минимизации времени выполнения модификаций ACO. Такой подход не только повышает общую производительность вычислительных процессов, но и позволяет более эффективно использовать ресурсы системы, что является критически важным в задачах с высокой вычислительной нагрузкой.

Для примера проектирования, оптимальная структура гетерогенного вычислителя будет опираться на временные характеристики и коэффициенты ускорения матричной реализации на CPU в сравнении с классической реализацией для 500 итераций, 500 муравьев-агентов при решении задачи вычисления оптимума функции Шаффера с 336 слоями в структуре данных с применением алгоритма ACOCCyN. Исследование гетерогенной реализации ACO затруднено неоднородностью систем вычисления времени, часть функций вычисляет время на CPU, а часть на GPU. Наиболее подходящими являются алгоритмы, реализуемые на CPU.

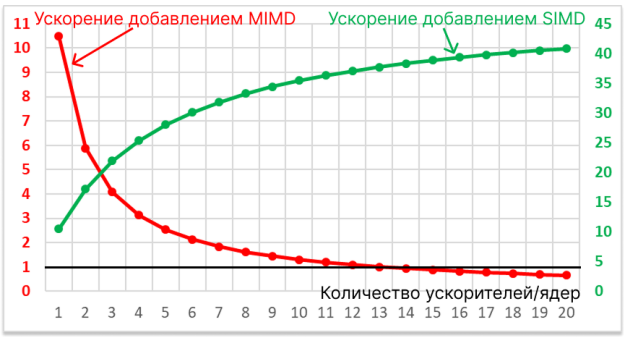
Оценка математического ожидания времени выполнения классической модификации $T_0 = T_{start} + T_{delt} + T_{stage1} + T_{stage2} + T_{stage3} = 51.677$ секунд, время без учета SISD компоненты $T_1 = T_0 - (T_{start} + T_{delt}) = 48.169$, доля распараллеливаемого фрагмента $\beta = 0.902$ что указывает на неэффективность глубокой оптимизации SISD компоненты. Высокая задержка связана с необходимостью инициализации хэш-таблицы, которая может быть ускорена многопоточным вычислителем, так для классического алгоритма без применения хэш-таблицы $\beta = 0.998$. Применение гибридного, состоящего из SIMD и MIMD компонент, вычислителя позволяет уменьшить время матричного алгоритма $T_{1,1} = 3.835$, которое разделяется по этапам на $T_{1,1} = T_{S1} + T_{SM} + T_{S3} = 0.054 + 3.480 + 0.301$, где T_{S1}, T_{S3} – время работы SIMD компоненты для этапа 1 и 3 соответственно, T_{SM} – время работы SIMD и MIMD компонент на втором этапе. Для дальнейшего разделения времени работы SIMD и MIMD компонент во втором этапе используется время работы модификации метода не использующей хэш-таблицу, так как данная модификация может быть целиком выполнена только на SIMD ускорителе. В результате общее время выполнения блока SIMD $T_S = T_{S1} + T_{S2,NONCUDA} + T_{S3} = 0.054 + 3.240 + 0.301 = 3.595$, а время MIMD $T_M = T_{SM} - T_{S2,NONCUDA} = 0.240$. Без учета SISD компоненты $T_{1,1} = T_S + T_M = 3.595 + 0.240 = 3.835$.

Для оценки эффективности и определения коэффициента ускорения необходимо вычислить ускорение, полученное за счет применения SIMD

вычислителя. Так как центральный процессор Intel Core i5-12450H подключает матричный ускоритель для всех вычислений, связанных с матрицами, то точная оценка коэффициента ускорения, связанного с намеренным использованием SIMD ускорителя на CPU затруднена, используется оценка $\rho = T_{S,ClassicACO} / T_S = 47.169 / 3.595 = 13.12$. На основе измерения времени классического алгоритма можно определить соотношение времени выполнения MIMD фрагмента к общему времени выполнения распараллеливаемого блока $\varphi = T_{M,ClassicACO} / T_1 = 1 / 48.169 = 0.02$. Доля SIMD фрагмента составит $(1-\varphi)T_1 = 0.98 * 48.169 = 47.205$. При наличии q процессоров (ядер) вычислителя возможен как параллельный запуск нескольких матричных модификаций метода муравьиных колоний с различными исходными данными (вариант 1), так и последовательный их запуск с возможностью проведения параллельной работы с хэш-таблицей в MIMD фрагменте (вариант 2). Параллельная работа с хэш-таблицей будет выполняться в режиме деления, когда количество вычислений, выполняемых одним процессором обратно пропорционально количеству процессоров. Ускорение вычислений в режиме деления может быть описано законом Амдала. В первом варианте общее время выполнения одновременно q и менее задач будет равно $T_q^1 = T_1$. Во втором варианте $T_q^2 = qT_{1,1}$, где $T_{q,1}$ – время работы матричной модификацией с учетом ускорения работы MIMD фрагмента на q потоках. $T_{q,1} = T_S + T_M / k$, где k – коэффициент ускорения на q ядрах. $T_S = \frac{(1-\varphi)T_1}{\rho}; T_M = \varphi T_1$, следовательно $T_{q,1} = T_1(\frac{1-\varphi}{\rho} + \frac{\varphi}{k})$, а коэффициент ускорения на q процессорах, связанный с применением матричного параллельного алгоритма относительного параллельного классического алгоритма $K_q = \frac{T_q^1}{T_q^2} = \frac{T_1}{qT_1(\frac{1-\varphi}{\rho}+\frac{\varphi}{k})} = \frac{k\rho}{q((1-\varphi)k+\varphi\rho)}$. В идеальном случае, когда $k=q$ получаем $K_q = \frac{\rho}{(1-\varphi)q+\varphi\rho}$ (рисунок 3). При $K_q > 1$ наращивание ядер MIMD фрагмента остается эффективным, в рассмотренном примере более 13 ядер MIMD не будут приносить должного ускорения.

Также возможно ускорение работы матричного метода путем увеличения количества SIMD ускорителей. Так как ускорители не могут параллельно выполнять вычисления, то будем задействовать их для запуска нескольких расчетов для различных путей. При r ускорителях коэффициент ускорения может быть вычислен как отношение времени работы матричной модификации алгоритма к работе матричной модификации при условии ускорения процесса выполнения на r SIMD ускорителях. $K_r = \frac{T_r^1}{T_r^2} = \frac{T_1}{\frac{T_S}{k}+T_M} = \frac{T_1}{T_1(\frac{1-\varphi}{k\rho}+\varphi)} = \frac{k\rho}{k\rho\varphi+1-\varphi}$.

На рисунке 3 на дополнительной, правой оси показано изменение коэффициента ускорения в зависимости от количества ускорителей SIMD при оптимальном случае, в котором $k=r$. $K_q = K_r = 10.481$ при $q=1$ и $r=1$, отличие от полученного значения 13.57 (таблица 2) связано с ускорением SISD компоненты, выполняющей инициализацию хэш-таблицы и загрузку данных. Предельное значение коэффициента может быть вычислено при условии $k=r; r \to \infty$ получаем $K_r = 1/\varphi = 48.176$.



Р и с. 3. Ускорение за счет увеличения количества SIMD-ускорителей и MIMD-ядер
F i g. 3. Speedup due to an increased number of SIMD accelerators and MIMD cores

В общем виде коэффициент ускорения вычислений с применением q MIMD вычислителей и r SIMD ускорителей может быть вычислен по формуле: $K_{q,r} = \frac{T_1}{qT_1(\frac{1-\varphi}{\rho k_r}+\frac{\varphi}{k_q})} = \frac{\rho k_r k_q}{q((1-\varphi)k_q+\varphi\rho k_r)}$. При оптимальных MIMD и SIMD компонентах, т.е при $k_r = r, k_q = q : K_{q,r} = \frac{\rho r}{(1-\varphi)q+\varphi\rho r}$.

Исследование оптимальной структуры гетерогенного вычислителя с учетом механизма реконфигурации

Для достижения максимального ускорения требуются различные структуры вычислителей, содержащие различные количества ядер и ускорителей. На практике количество ядер и ускорителей в составе гибридного вычислителя фиксировано. Как правило, вследствие конструктивных особенностей, выполняется $q>r^2$. Начиная с некоторого значения $q'<q$ прирост коэффициента ускорения при заданном $r=const$ становится незначительным. Поэтому оказывается целесообразным образование в вычислителе двух структур:

- Однородной MIMD структуры из $(q\text{-}q')$ универсальных ядер без SIMD ускорителей
- Гибридной структуры, содержащей q' MIMD ядер и r SIMD ускорителей, в которой с одним ускорителем взаимодействуют $d = \frac{q'}{r}$ ядер.

[2] Степаненко С. А. Мультипроцессорные среды суперЭВМ. Масштабирование эффективности. М. : ФИЗМАТЛИТ, 2016. 312 с.

В такой постановке общий коэффициент ускорения складывается из коэффициента ускорения однородной MIMD структуры $(K_M = q - q')$ и гибридной структуры $K_{q',r} = rK_{1,d} = r\frac{\rho d}{(1-\varphi)+\varphi\rho d}$.

$$K^{re}_{q,d} = q - q' + r\frac{\rho d}{(1-\varphi)+\varphi\rho d} = q - q' + r\frac{\rho q'}{(1-\varphi)+\varphi\frac{\rho q'}{r}} = q - q' + r\frac{r\rho q'}{(1-\varphi)r+\varphi\rho q'} = q - qx + r\frac{r\rho qx}{(1-\varphi)r+\varphi\rho qx}$$

где $x = q'/q$. Для вычисления оптимального количества ядер в гибридной структуре, вычислим производную по x для коэффициента ускорения K используя правило дифференцирования частного для гибридного коэффициента ускорения.

$$u = r\rho qx, v = (1-\varphi)r + \varphi\rho qx, \frac{du}{dx} = r\rho q, \frac{dv}{dx} = \varphi\rho q.$$

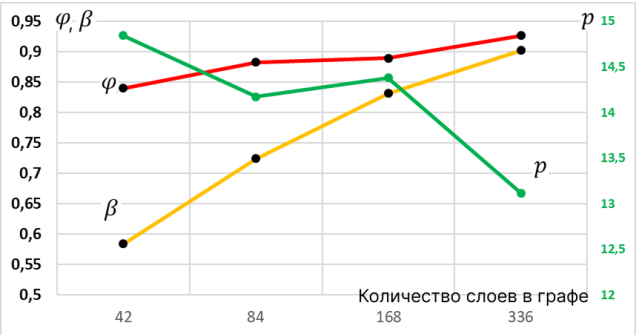
$$\frac{dK^{re}_{q,d}}{dx} = -q + \frac{r\rho q((1-\varphi)r+\varphi\rho qx) - r\rho qx(\varphi\rho q)}{((1-\kappa)r+\varphi\rho qx)^2} = -q + \frac{r^2\rho q(1-\varphi)}{((1-\varphi)r+\varphi\rho xq)^2} = 0 \Rightarrow$$

$$r^2\rho q(1-\varphi) = q((1-\varphi)r+\varphi\rho xq)^2 \Rightarrow$$

$$x = \frac{r(\sqrt{\rho(1-\varphi)} - (1-\varphi)}{q\varphi\rho}$$

Подставляя значения для исследованной в предыдущем подразделе задаче получаем $x = 43.58/q$, а $q' = \lfloor qx \rfloor$. Например, в случае наличия одного ускорителя SIMD и 16 ядер MIMD рекомендуется 5 ядер MIMD отнести к однородной MIMD структуре, а 11 ядер MIMD дополнить к ускорителю SIMD в гибридной структуре. При этом $K^{re}_{q,d} = 39.02$ Если же ускорителей SIMD в системе 5, то только одно ядро MIMD можно будет освободить из гибридной структуры $K^{re}_{q,d} = 110.55$.

Полученные итоговые формулы зависят от соотношения объемов распараллеливаемых вычислений β, соотношении объемов SIMD и MIMD компонент φ, коэффициенту ускорения SIMD ускорителем по сравнению с MIMD ядром ρ, а также планируемыми коэффициентами распараллеливания для MIMD и SIMD компонент соответственно k_q, k_r. На рисунке 12 представлены зависимости представленных характеристик для CPU матричной модификации метода муравьиных колоний при различной размерности задачи.



Р и с. 4. Изменение коэффициентов β и φ (левая ось), ρ (правая ось) в зависимости от размерности задачи для матричной модификации муравьиной колонии, работающей на CPU

F i g. 4. Variation of the coefficients β and φ (left axis) and ρ (right axis) depending on the problem dimension for the CPU-based matrix ant colony modification

На рисунке 4 видно, что влияние MIMD компоненты существенно уменьшается с увеличением размерности задачи. Коэффициенты β и φ стремятся к единице и для размерности 336 слоев больше 0.9, т.е. 90% всех вычислений сосредоточено в SIMD ускорителе.

Заключение

По результатам выполненных исследований достигнуто ускорение модификаций матричных модификаций ACO до 39 раз по сравнению классической реализацией ACO при решении параметрической задачи на оптимальном графе. Оптимальный граф формируется путем разложения всего множества значений одного параметра на слои, и определение итогового значения путем линейной комбинации значений в слоях. Данный граф не нарушает логику работы ACO так как фактически преобразует задачу малой размерности с небольшим количеством параметров, где у каждого параметра много значений, в задачу большой размерности, где у каждого параметра мало значений. Зависимость между параметрами и значением целевой функции на графе определяется количеством весов-феромона у каждого значения параметра. В таком виде максимальное количество значений в одном слое равно 5 и позволяет добиться максимальной скорости поиска и возможностей параллельной обработки.

Максимальное ускорение достигнуто матричной гетерогенной модификацией, в которой «добавление» и «испарение» весов-феромона, а также выполнение формулы вероятностного перехода для вычисления вероятности выбора значения, формирование матрицы значений функций распределения по полученным вероятностям осуществляется на CPU за счет оптимизации компилятором C++ соответствующих матричных преобразований. Поиск путей муравьев-агентов, вычисление значения целевой функции и работа модификаций ACOCCyN производится на GPU.

При этом следует отметить, что реализация матричного ACO полностью на GPU показывает результаты хуже, чем реализация матричного ACO полностью на CPU. Во многом это связано с особенностями параллелизма CUDA: необходимо грамотно взаимодействовать с различными тапами памяти (общей, локальной и константной), необходимо грамотно работать с количеством потоков и блоков, передача данных между оперативной памяти CPU и оперативной памяти GPU, блокировка потока CPU при выполнении кода GPU и синхронизация всех потоков перед выходом из функции GPU. Для улучшения алгоритма на GPU требуется тщательное изучение данных аспектов.

Другим направлением исследований является прямое применение SIMD функций на CPU: SSE и AVX. Данные функции позволяют производить одну операцию с векторами до 256 байт, в который может быть записано множество значений. Но из-за

размерности графа в 5 элементов прирост в таких алгоритмах является минимальным. Максимальный прирост может быть достигнут при кратности 4 количества значений в слое. Наиболее близким является 100 значений в одном слое.

Тем не менее в работе продемонстрирована эффективность переработки алгоритмов для вычисления их на гетерогенном вычислителе с разделением на SIMD и MIMD компоненты. Получено максимальное ускорение до 38 раз. Предложена оптимальная структура реконфигурируемого гетерогенного вычислителя.

References

1. Maniezzo V., Boschetti M.A., Stützle T. Matheuristics: Algorithms and Implementations. *EURO Advanced Tutorials on Operational Research*. Cham: Springer; 2021. 214. https://doi.org/10.1007/978-3-030-70277-9
2. Dorigo M., Stützle T. Ant Colony Optimization. Cambridge, Massachusetts: MIT Press; 2004. 321 p.
3. Uslu M.O., Erdoğdu K. Ant Colony Optimization and Beam-Ant Colony Optimization on Traveling Salesman Problem with Traffic Congestion. *DEUFMD*. 2024;26(78):519-527. https://doi.org/10.21205/deufmd.2024267820
4. Sagban R.F., Ku-Mahamud K.R., Abu Bakar M.S. Reactive max-min ant system with recursive local search and its application to TSP and QAP. *Intelligent Automation & Soft Computing*. 2017;23(1):127-134. https://doi.org/10.1080/10798587.2016.1177914
5. Ghimire B., Mahmood A., Elleithy K. Hybrid Parallel Ant Colony Optimization for Application to Quantum Computing to Solve Large-Scale Combinatorial Optimization Problems. *Applied Sciences*. 2023;13:11817. https://doi.org/10.3390/app132111817
6. Črepinšek M., Liu S.-H., Mernik M. Exploration and Exploitation in Evolutionary Algorithms: A Survey. *ACM Computing Surveys*. 2013;45:35. https://doi.org/10.1145/2480741.2480752
7. Dorigo M., Birattari M. Swarm intelligence. *Scholarpedia*. 2007;2(9):1462.
8. Pellegrini P., Stützle T., Birattari M. A critical analysis of parameter adaptation in ant colony optimization. *Swarm intelligence*. 2012;6:23-48. https://doi.org/10.1007/s11721-011-0061-0
9. Danesh M., Danesh S. Optimal design of adaptive neuro-fuzzy inference system using PSO and ant colony optimization for estimation of uncertain observed values. *Soft Computing – A Fusion of Foundations, Methodologies and Applications*. 2024;28(1):135-152. https://doi.org/10.1007/s00500-023-09194-6
10. Yin C., Fang Q., Li H., et al. An optimized resource scheduling algorithm based on GA and ACO algorithm in fog computing. *The Journal of Supercomputing*. 2024;80:4248-4285. https://doi.org/10.1007/s11227-023-05571-y
11. Bullnheimer B., Kotsis G., Strauß C. Parallelization strategies for the ant system. *Applied Optimization*. 1998;24:87-100. https://doi.org/10.1007/978-1-4613-3279-4_6
12. Randall M., Lewis A. A parallel implementation of ant colony optimization. *Journal of Parallel and Distributed Computing*. 2002;62(9):421-1432. https://doi.org/10.1006/jpdc.2002.1854
13. Mansour I.B., Alaya I.B., Tagina M. A New Parallel Hybrid MultiObjective Ant Colony Algorithm Based on OpenMP. In: Weghorn H. (Ed.) Proceedings of the 17th International Conference on Applied Computing (AC2020). IADIS Press; 2020. p. 19-26. https://doi.org/10.33965/ac2020_202013l003
14. Skinderowicz R. Implementing a GPU-based parallel MAX–MIN ant system. *Future Generation Computer Systems*. 2020;106:277-295. https://doi.org/10.1016/j.future.2020.01.011
15. Zeng Z., Cai Y., Chung K.L., Lin H., Wu J. A Fast Fully Parallel Ant Colony Optimization Algorithm Based on CUDA for Solving TSP. *IET Computers & Digital Techniques*. 2023:9915769. https://doi.org/10.1049/2023/9915769
16. Tsutsui S. ACO on Multiple GPUs with CUDA for Faster Solution of QAPs. In: Coello C.A.C., Cutello V., Deb K., Forrest S., Nicosia G., Pavone M. (eds.) Parallel Problem Solving from Nature – PPSN XII. PPSN 2012. *Lecture Notes in Computer Science*. Vol. 7492. Berlin, Heidelberg: Springer; 2012. p. 174-184. https://doi.org/10.1007/978-3-642-32964-7_18
17. de Melo Menezes B.A., Herrmann N., Kuchen H., et al. High-Level Parallel Ant Colony Optimization with Algorithmic Skeletons. *International Journal of Parallel Programming*. 2021;49:776-801. https://doi.org/10.1007/s10766-021-00714-1
18. Shan H. A novel travel route planning method based on an ant colony optimization algorithm. *Open Geosciences*. 2023;15(1):20220541. https://doi.org/10.1515/geo-2022-0541
19. Yang L., Jiang T., Cheng R. Tensorized Ant Colony Optimization for GPU Acceleration. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '24 Companion). New York, NY, USA: Association for Computing Machinery; 2024. p. 755-758. https://doi.org/10.1145/3638530.3654394
20. Dzalbs I., Kalganova T. Accelerating supply chains with Ant Colony Optimization across range of hardware solutions. *Computers & Industrial Engineering*. 2020;147:106610. https://doi.org/10.1016/j.cie.2020.106610
21. Sudakov V.A., Titov Yu.P. Investigation of the parametric graph model in the ant colony method. *Mathematical Models and Computer Simulations*. 2025;17(2):126-136. https://doi.org/10.1134/S2070048224700996
22. Sinitsyn I.N., Titov Yu.P. Control of Set of System Parameter Values by the Ant Colony Method. *Automation and Remote Control*. 2023;84(8):893-903. https://doi.org/10.1134/S0005117923080106

23. Sinitsyn I.N., Titov Yu.P. Investigation of Algorithms for Cyclic Search for Additional Solutions when Optimizing the Order of Hyperparameters by the Ant Colony Method. *Sistemy' vy'sokoj dostupnosti* = Highly Available Systems. 2023;19(1):59-73. (In Russ., abstract in Eng.) https://doi.org/10.18127/j20729472-202301-05
24. Sinitsyn I.N., Titov Yu.P. Optimization of the Order of Hyperparameters of Computational Cluster by the Ant Colony Method. *Sistemy' vy'sokoj dostupnosti* = Highly Available Systems. 2022;18(3):23-37. (In Russ., abstract in Eng.) https://doi.org/10.18127/j20729472-202203-02
25. Sudakov V., Titov Y. Matrix-Based ACO for Solving Parametric Problems Using Heterogeneous Reconfigurable Computers and SIMD Accelerators. *Mathematics.* 2025;13(8):1284. https://doi.org/10.3390/math13081284
26. Sudhanshu M.K. Some New Test Functions for Global Optimization and Performance of Repulsive Particle Swarm Method. University Library of Munich, Germany, MPRA Paper. 2006. https://doi.org/10.2139/ssrn.926132
27. Chetverushkin B.N., Sudakov V.A, Titov Y.P. Graph Condensation for Large Factor Models. *Doklady Mathematics.* 2024;109(3):246-251. https://doi.org/10.1134/S1064562424702090
28. Chetverushkin B.N., Sudakov V.A, Titov Y.P. Modeling of high-dimensional systems based on graph condensation. *Matematicheskoe modelirovanie.* 2025;37(3):59-74. (In Russ., abstract in Eng.) https://doi.org/10.20948/mm-2025-03-04

Поступила 28.04.2025; одобрена после рецензирования 04.06.2025; принята к публикации 18.06.2025.

Submitted 28.04.2025; approved after reviewing 04.06.2025; accepted for publication 18.06.2025.

Об авторе:

Титов Юрий Павлович, доцент кафедры 304 «Вычислительные машины системы и сети», ФГБОУ ВО «Московский авиационный институт (национальный исследовательский университет)» (125993, Российская Федерация, г. Москва, Волоколамское шоссе, д. 4), кандидат технических наук, **ORCID: https://orcid.org/0000-0002-9093-6755,** kalengul@mail.ru

Автор прочитал и одобрил окончательный вариант рукописи.

About the author:

Yuri P. Titov, Associate Professor of the Chair of Computing Machines, Systems and Networks, Moscow Aviation Institute (National Research University) (4 Volokolamskoe shosse, Moscow 125993, Russian Federation), Cand. Sci. (Eng.), **ORCID: https://orcid.org/0000-0002-9093-6755,** kalengul@mail.ru

The author has read and approved the final manuscript.