УДК 001.89+004.912

ИССЛЕДОВАНИЕ ПАРАЛЛЕЛЬНЫХ ПРИЛОЖЕНИЙ ДЛЯ ГЕТЕРОГЕННЫХ АРХИТЕКТУР С ИСПОЛЬЗОВАНИЕМ СТАНДАРТОВ SYCL И OPENMP

О. Ю. Шамаева[1] [0009-0006-3041-6682], А. М. Чернецов[2] [0000-0001-7655-2395], Л. А. Зинченко[3] [0000-0002-2298-8721]

[1, 2]*Национальный исследовательский университет «МЭИ», г. Москва, Россия*

[3]*Московский государственный технический университет им. Н. Э. Баумана, г. Москва, Россия*

[1]shamayevaoy@mpei.ru, [2]chernetsovam@mpei.ru, [3]lyudmillaa@mail.ru

Аннотация

Представлены результаты исследований параллельных приложений для гетерогенных архитектур с использованием стандарта SYCL и его реализации DPC++, а также технологии OpenMP, разработанные студентами кафедры прикладной математики и искусственного интеллекта (ПМИИ) ФГБОУ ВО «НИУ «МЭИ» в ходе выполнения научно-исследовательских работ. В качестве прикладной задачи рассмотрен поиск кратчайших путей в графах различной размерности с использованием алгоритмов Беллмана-Форда и дельта-шага.

Алгоритм Беллмана – Форда легко поддается распараллеливанию и продемонстрировал отличную масштабируемость. Для алгоритма дельта-шага, напротив, использование GPU не дало заметного ускорения из-за сложности согласования памяти и необходимости многократного копирования данных между устройствами.

Тестирование проводилось с использованием следующих аппаратных средств: центральный процессор Intel Core i5-12400F (6 ядер, 12 потоков), графический процессор NVIDIA GeForce GTX1660 (1408 CUDA-ядер). Результаты тестирования показывают, что даже в рамках одной вычислительной задачи различные реализации могут демонстрировать разное поведение при изменении параметров задачи. Это подтверждает важность предварительного анализа и тестирования при выборе аппаратно-программной архитектуры для задач, требующих высокой производительности.

Разработанный комплекс программ визуализирует зависимость ускорения параллельных реализаций алгоритма от числа вершин в графе при различных параметрах метода и успешно используется в учебно-научном процессе кафедры ПМИИ ФГБОУ ВО «НИУ «МЭИ».

Ключевые слова: *гетерогенная архитектура, задача поиска кратчайших путей в графе, стандарт SYCL, язык DPC++, технология OpenMP.*

ВВЕДЕНИЕ

Задача поиска кратчайшего пути в графе является одной из фундаментальных проблем теории графов и алгоритмической оптимизации. Ее актуальность обусловлена широким спектром приложений в современных технологических и научных областях. Перечислим основные из них.

• *Транспортные системы и навигация.* Алгоритмы поиска кратчайшего пути лежат в основе картографических сервисов, логистических систем и управления трафиком, а также в основе навигации беспилотных транспортных средств, которые должны адаптироваться к изменяющимся дорожным условиям, пробкам и авариям.

• *Телекоммуникационные сети*. *Маршрутизация данных в интернете, оптимизация пропускной способности сетей.* Маршрутизация данных должна учитывать не только кратчайший путь, но и параметры задержки, нагрузки и безопасности. Алгоритмы на основе графов помогают балансировать трафик между узлами.

• *Биоинформатика. Анализ молекулярных взаимодействий, моделирование метаболических путей*. Моделирование белковых взаимодействий или метаболических путей требует анализа графов с миллиардами узлов. Например, поиск кратчайшего пути между генами помогает идентифицировать мишени для лекарств.

• *Социальные сети и рекомендательные системы. Поиск связей между пользователями, предсказание интересов*. Кратчайший путь помогает определять «степень влияния» между людьми или находить целевые аудитории для рекламы.

• *Финансы и безопасность.* Алгоритмы поиска пути помогают находить оптимальные цепочки сделок на финансовых рынках. В блокчейн-сетях анализ

транзакционных графов помогает выявлять мошеннические схемы. Например, алгоритмы поиска пути используются для отслеживания движения криптовалют между кошельками.

С развитием технологий больших данных (Big Data) и Интернета вещей (IoT) объемы обрабатываемых графов растут экспоненциально. Например, социальные сети содержат миллиарды узлов, а транспортные системы мегаполисов генерируют графы с миллионами ребер, что требует разработки высокопроизводительных решений, способных эффективно работать в условиях ограниченных ресурсов и жестких временных рамок. При этом современные графы часто обладают свойствами, которые усложняют их обработку: динамичность, гетерогенность, масштабность.

Кроме того, переход к гетерогенным вычислительным архитектурам (CPU + GPU, FPGA, AI-ускорители) делает актуальным исследование методов распараллеливания алгоритмов для таких систем. Современные стандарты, такие как SYCL [1, 2] и его реализация DPC++ [3], предоставляют инструменты для кроссплатформенной разработки, что открывает новые возможности для оптимизации трудоемких задач.

В учебном процессе кафедры прикладной математики и искусственного интеллекта (ПМИИ) ФГБОУ ВО НИУ «МЭИ» по дисциплинам «Архитектура вычислительных систем» и «Параллельное программирование и параллельные системы» для решения задач в модели с общей памятью изучают стандарт OpenMP [4]. Однако этот стандарт оказался неэффективным для современных гетерогенных вычислительных архитектур, что было установлено в ходе проведенных исследований. В настоящей работе на примере решения задачи поиска кратчайшего пути в графе исследованы перечисленные выше стандарты для использования в гетерогенных архитектурах.

ПОСТАНОВКА ЗАДАЧИ ПОИСКА КРАТЧАЙШЕГО ПУТИ В ГРАФЕ

Рассмотрим задачу поиска кратчайших путей из некоторой вершины во все остальные.

Как известно, граф – это математическая структура, которая используется для представления множества объектов и отношений между ними и состоит из двух ключевых компонентов: вершин (или узлов) и ребер (или дуг).

Вершины – это отдельные объекты в графе. В контексте алгоритмов поиска пути вершины могут представлять физические места, такие как города на карте, или абстрактные концепции, такие как состояния в задаче планирования.

Ребра – это связи между вершинами. Ребра могут быть взвешенными или невзвешенными, а также направленными или ненаправленными. Во взвешенном графе каждому ребру присваивается значение (вес), представляющее стоимость или расстояние между двумя вершинами. В направленном графе ребра указывают направление перехода от одной вершины к другой.

Путь в графе – это последовательность вершин, в которой каждая вершина связана с следующей вершиной в последовательности ребром. Длина пути обычно определяется как сумма весов всех ребер в пути в взвешенном графе или просто кака количество ребер в пути в невзвешенном графе.

Сформулируем теперь саму задачу. Пусть дан неориентированный граф $G = (V, E)$, $|V| = n$ – число вершин, $|E| = m$ – число ребер графа с неотрицательными весами. Необходимо найти кратчайшие пути из вершины s во все остальные.

Из всего многообразия существующих алгоритмов поиска кратчайшего пути в графе [5, 6] нами исследованы два базовых алгоритма, принципиально различающихся с точки зрения возможности разработки их параллельных модификаций.

АЛГОРИТМЫ РЕШЕНИЯ ПРИКЛАДНОЙ ЗАДАЧИ

Алгоритм Беллмана – Форда

Несмотря на то что рассматриваются графы, у которых веса ребер положительные, алгоритм Беллмана – Форда позволяет найти кратчайшие пути из одной вершины графа до всех остальных, даже для графов, в которых веса ребер могут быть отрицательными. Однако, в графе не должно быть циклов от-

рицательного веса, достижимых из начальной вершины, в противном случае вопрос о кратчайших путях является бессмысленным.

На первом шаге инициализируем массив dist расстояний от исходной вершины до всех остальных вершин. Все значения установим как бесконечные, а расстояние до исходной вершины примем равным 0. На вход алгоритм принимает массив ребер графа, и он используется следующим образом: запускается цикл по количеству вершин в графе, на каждом шаге этого цикла для всех ребер *E(u,v)* производится попытка проведения релаксации. Релаксация в данном случае выглядит следующим образом: производится сравнение величины расстояния от исходной вершины до вершины *v* (dist[*v*]) и расстояния от исходной вершины до вершины *u* в сумме с ребром *E*(*u, v*), лежащим между вершинами *u* и *v*, т. е. dist[*u*]+*w*(*u, v*). Выбирается меньшая из этих величин и записывается в качестве расстояния от исходной вершины до вершины *v*. При этом работу алгоритма можно завершить на шаге *k*, если dist на этом шаге равен dist на шаге *k* + 1, т. е. за очередной шаг цикла не было успешных релаксаций [6].

Продемонстрируем работу алгоритма на примере графа, изображенного на рис. 1.

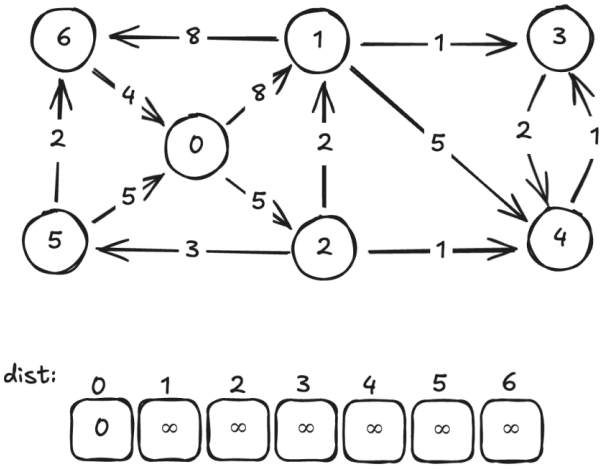

Рис. 1. Пример применения алгоритма Беллмана – Форда. Первый шаг.

Пусть необходимо узнать кратчайшие расстояния от вершины 0 до всех остальных вершин. На первом шаге значение dist[*i*] известно только для *i = 0*. На рис. 2 отмечены ребра, для которых была произведена релаксация.

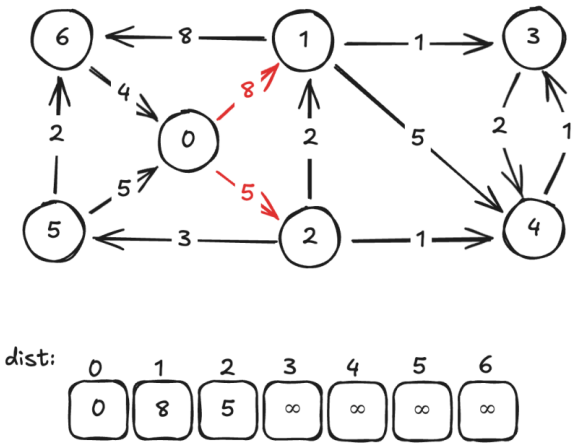

Рис. 2. Пример применения алгоритма Беллмана – Форда. Продолжение.

Для остальных ребер релаксация была неуспешной, так как расстояния *dist* до остальных вершин, кроме 0, были неизвестны. После первого шага алгоритма удалось получить расстояния до вершин 1 и 2. Красным отмечены те ребра, которые были использованы на этом шаге для нахождения расстояний. На рис. 3 отмечены ребра, для которых была произведена релаксация на втором шаге.

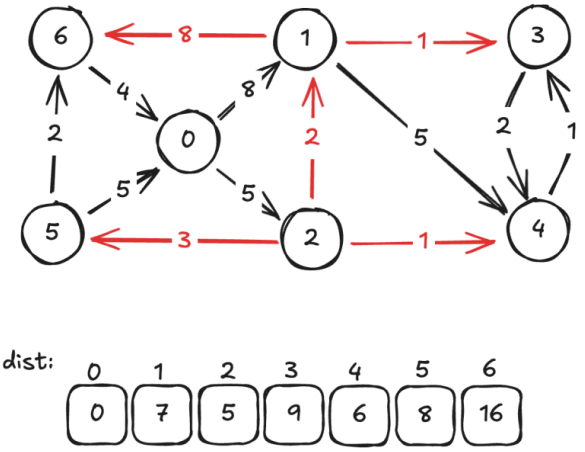

Рис. 3. Пример применения алгоритма Беллмана – Форда. Второй шаг.

После второго шага удалось получить расстояния до всех вершин, однако при последующих шагах алгоритма эти расстояния могут уменьшится. На рис. 4 отмечены ребра, для которых была произведена релаксация на третьем шаге, и показаны величины получившихся расстояний от вершины 0.

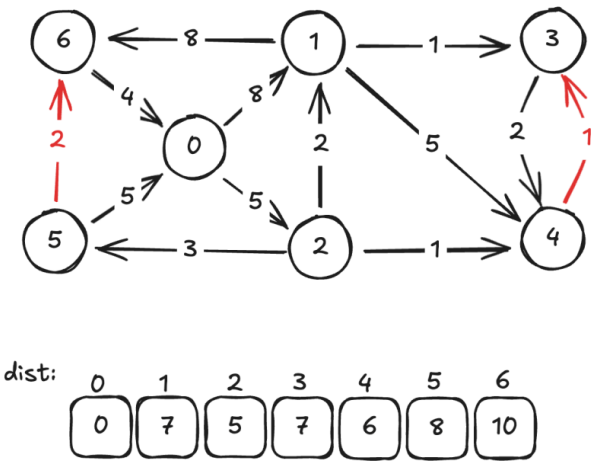

Рис. 4. Пример применения алгоритма Беллмана – Форда. Третий шаг.

После третьего шага удалось получить меньшие расстояния для вершин 3 и 6. При следующем шаге ни для одной вершины улучшить расстояние не удалось, поэтому останавливаем работу алгоритма.

Сложность такого алгоритма равна $O(mn)$. Видно, что для плотных графов (где $m \sim n^2$) сложность близка к $O(n^3)$, а для разреженных графов (где $m \sim n$) – $O(n^2)$.

Алгоритм Дейкстры

Алгоритм Дейкстры – это алгоритм поиска пути, который был разработан голландским ученым Эдсгером Дейкстрой в 1959 г. Алгоритм используют для поиска кратчайшего пути во взвешенном графе от одной вершины (начальной) до всех остальных вершин. Основная идея состоит в том, что выполняется обход графа в ширину с пересчетом оценки расстояния от источника до других вершин. Жадный принцип – это продвижение по ребрам с меньшим весом.

Создается множество *S*, содержащее уже обработанные вершины (изначально пустое). Создается также множество *Q,* содержащее все остальные вершины графа (изначально содержит все вершины графа). При этом каждой вершине графа присваивается вес, который представляет минимальную известную длину пути от начальной вершины до данной. Для начальной вершины этот вес равен 0, для всех остальных вершин – бесконечность. На каждом шаге алгоритм выбирает вершину из непосещенного множества с наименьшим весом, перемещает эту вершину в множество посещенных вершин и производит

релаксацию (обновление весов всех соседей выбранной вершины). Вес соседа обновляется, если через выбранную вершину можно добраться до этого соседа с меньшей длиной пути. Процесс продолжается, пока не будут посещены все вершины [5].

Продемонстрируем работу алгоритма Дейкстры на примере графа, изображенного на рис. 5.

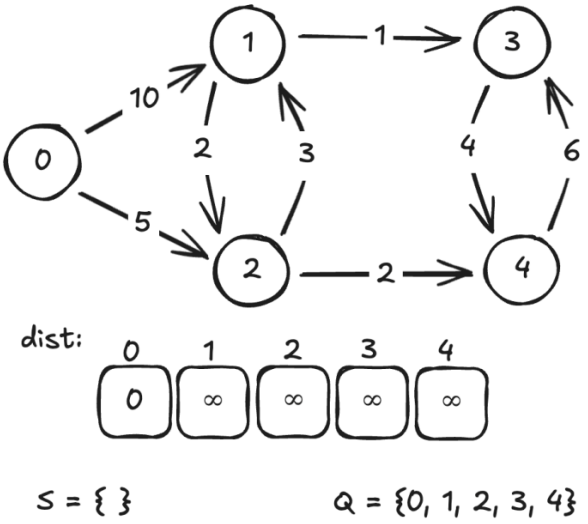

Рис. 5. Пример применения алгоритма Дейкстры.

Пусть необходимо узнать кратчайшие расстояния от вершины 0 до всех остальных вершин. На первом шаге величина *dist*[*i*] известна только для только для *i* = 0. На рис. 6. отмечены ребра, для которых будет произведена релаксация на первом шаге.

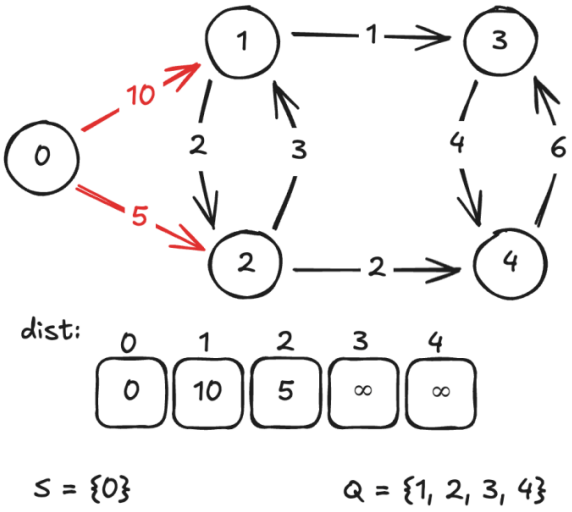

Рис. 6. Пример применения алгоритма Дейкстры. Первый шаг.

Основной была выбрана вершина 0, так как у нее было наименьшее (и единственно известное) расстояние dist. Вершина 0 была помечена как посещенная и удалена из списка *Q* непосещенных вершин. На рис. 7 отмечены ребра, для которых будет произведена релаксация на втором шаге.

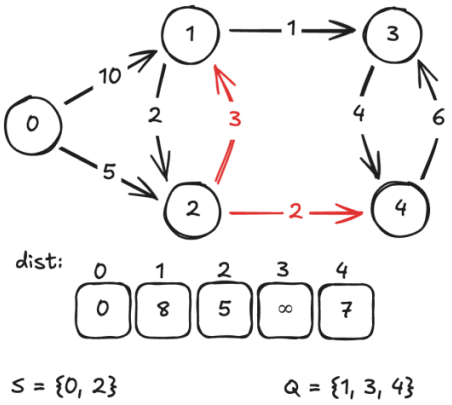

Рис. 7. Пример применения алгоритма Дейкстры. Второй шаг.

Основной была выбрана вершина 2, так как у нее было наименьшее значение dist из непосещенных. Таким образом было найдено расстояние до вершины 4 и улучшено расстояние до вершины 1. На рис. 8 показаны последующие шаги алгоритма Дейкстры.

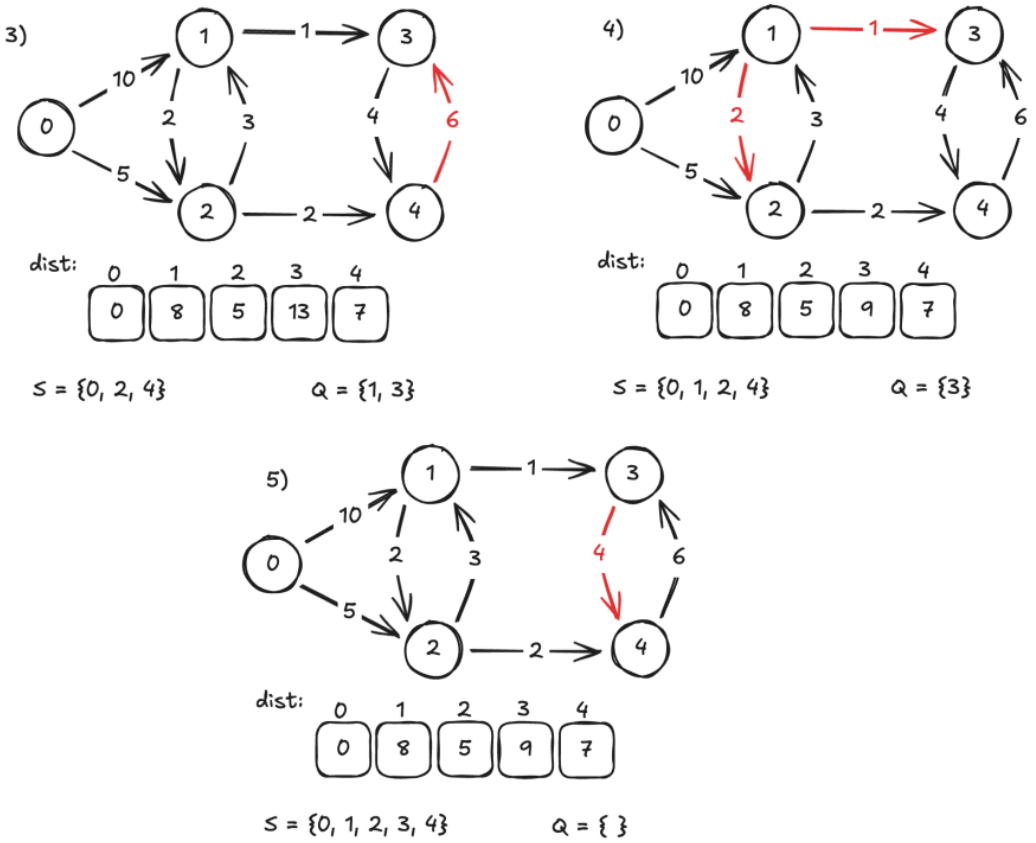

Рис. 8. Пример применения алгоритма Дейкстры. Третий шаг.

Шаги алгоритма повторяются, пока множество непосещенных вершин не станет пустым. Таким образом находятся все кратчайшие пути из вершины 0.

Асимптотика такого алгоритма составит $O(n^2)$: на каждой итерации находим минимум за $O(n)$ и проводим $O(n)$ релаксаций. Однако минимум можно искать быстрее при использовании специализированной структуры, которая поддерживает добавление элементов и нахождение минимума – кучи (приоритетной очереди).

Анализируя приведенные алгоритмы с точки зрения возможности распараллеливания и эффективной дальнейшей реализации на гетерогенных системах, можно заключить, что алгоритм Беллмана – Форда ориентируется на релаксацию всех ребер и хорошо масштабируется при распараллеливании, тогда как второй алгоритм построен на жадной стратегии и плохо поддается массовому параллелизму.

Такое сравнение особенно важно в контексте дальнейшей реализации на гетерогенных системах.

ПАРАЛЛЕЛЬНЫЕ МОДИФИКАЦИИ АЛГОРИТМОВ

Приведем параллельные модификации рассмотренных выше алгоритмов.

Параллельная модификация алгоритма Беллмана – Форда

Параллельная модификация алгоритма Беллмана – Форда принципиально не отличается от последовательного алгоритма. Поскольку на каждом шаге алгоритма можно независимо обрабатывать все ребра графа, эту часть обработки можно выполнять параллельно. Приведем псевдокод параллельного алгоритма:

для каждого i от 1 до n:
параллельно для каждого ребра $E(u, v)$ в графе:
    если dist[u] + $w(u, v)$ < dist[v]:
        dist[v] = dist[u] + $w(u, v)$;
дождаться выполнения параллельных вызовов;
Если не было изменений в dist на этом шаге:
    остановить работу алгоритма.

Параллельная модификация алгоритма Дейкстры

Классический алгоритм Дейкстры по своей природе слабо поддается распараллеливанию, так как предполагает последовательную обработку вершин, что сильно ограничивает масштабируемость на многоядерных системах. В связи с этим будем рассматривать не сам алгоритм Дейкстры, а его адаптированную версию – алгоритм дельта-шага (*delta-stepping*), который был разработан именно с учетом возможности параллельной обработки и считается одной из наиболее удачных модификаций для многопоточной среды.

Суть алгоритма заключается в том, чтобы группировать вершины по диапазонам расстояний – в отличие от классического подхода, где вершины обрабатываются строго по одной, начиная с ближайшей. Такой подход позволяет обрабатывать целые группы вершин одновременно, что значительно повышает эффективность при выполнении на устройствах с поддержкой параллельных вычислений [7].

Ключевым параметром здесь выступает величина дельта (Δ). Она определяет, где проходит граница между так называемыми «легкими» и «тяжелыми» ребрами графа. По сути, Δ управляет тем, насколько мелко или крупно будут выделены диапазоны расстояний. При правильно подобранном значении Δ можно добиться хорошего баланса между уровнем параллелизма и издержками на синхронизацию между потоками.

Алгоритм начинается с инициализации расстояний до всех вершин как бесконечно больших, за исключением начальной вершины, расстояние до которой устанавливается равным нулю. Затем выполняется серия итераций, каждая из которых состоит из двух этапов.

На первом этапе алгоритм обрабатывает так называемые «легкие» ребра, те, вес которых меньше выбранного параметра Δ. Вершины группируются в «корзины» (buckets) в соответствии с текущими оценками расстояний до них. В *i*-й корзине содержатся вершины, текущая оценка расстояния до которых удовлетворяет неравенству $\Delta * i \le \mathrm{dist} \le \Delta * (i+1)$. Все вершины, находящиеся в одной корзине, могут обрабатываться параллельно, так как их расстояния лежат в одном диапазоне [8].

Второй этап посвящен обработке «тяжелых» ребер, вес которых превышает Δ. Эти ребра требуют более аккуратной обработки и дополнительной синхронизации между параллельными потоками.

Таким образом, алгоритм можно описать следующей последовательностью шагов.

1. Выбираем корзину с наименьшим индексом, содержащую вершины.
2. Повторяем, пока текущая корзина не станет пустой:
   • извлекаем все вершины из текущей корзины;
   • обрабатываем «легкие» ребра этих вершин:
   – производим релаксацию для каждого ребра, т. е. пытаемся уменьшить расстояние до соседних вершин;
   – после релаксации расстояние от вершины-источника до обработанных вершин могло измениться, поэтому перемещаем вершины в соответствующие новые корзины;
   • обрабатываем «тяжелые» ребра для всех обработанных ранее вершин:

– аналогично обновляем расстояния и перемещаем вершины в новые корзины.

3. Алгоритм завершает работу, когда все корзины становятся пустыми. На этом этапе оценки расстояний содержат длины кратчайших путей от начальной вершины до всех остальных вершин графа.

Рассмотрим работу алгоритма дельта-шага на примере поиска всех кратчайших расстояний от вершины 0 до всех остальных вершин

Выберем Δ = 4. У вершины 0 нет «легких» ребер, поэтому сразу обрабатываются «тяжелые» ребра. На рис. 9 отмечены ребра, которые будут обработаны на первом шаге.

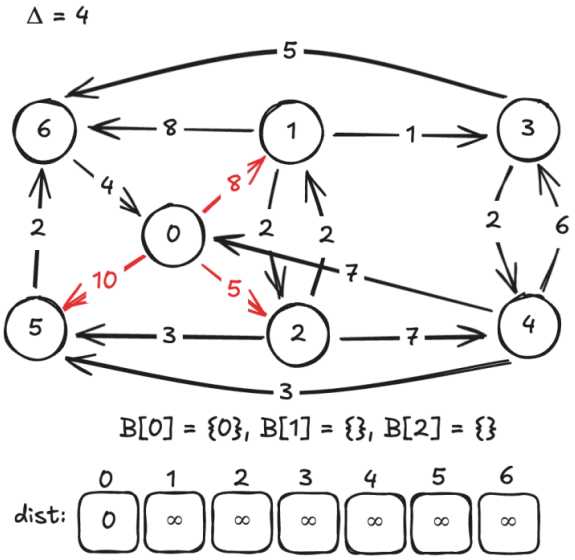

Рис. 9. Пример применения алгоритма дельта-шага.

Все вершины корзины *B*[0] были обработаны, поэтому переходим к обработке корзины *B*[1]. Сначала обрабатываются «легкие» ребра. На рис. 10 отмечены все «легкие» ребра, которые будут обработаны на втором шаге.

4.

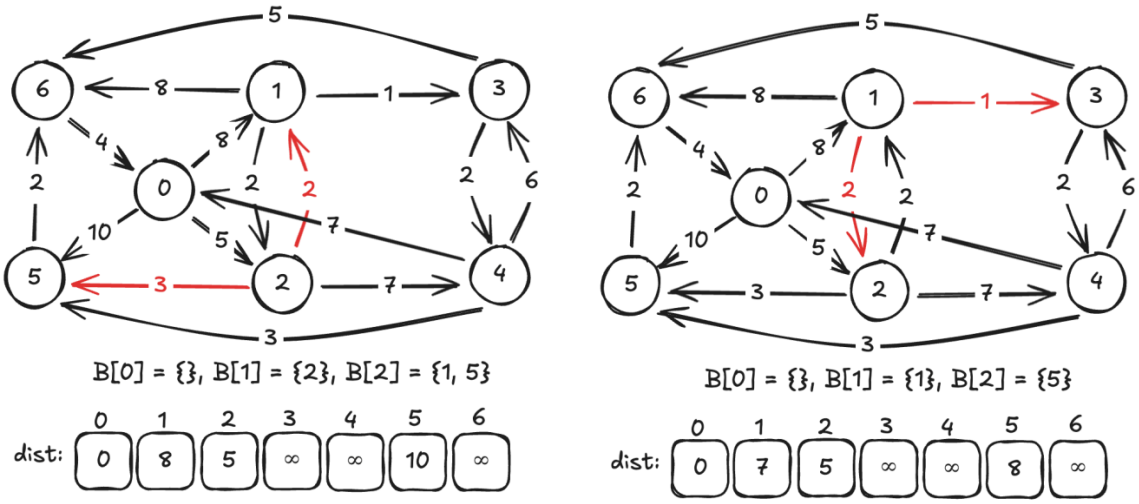

Рис. 10. Пример применения алгоритма дельта-шага. Первый шаг.

Затем обрабатываются все «тяжелые» ребра вершин, обработанных в корзине *B*[1]. На рис. 11 отмечены все «тяжелые» ребра, которые будут обработаны на втором шаге.

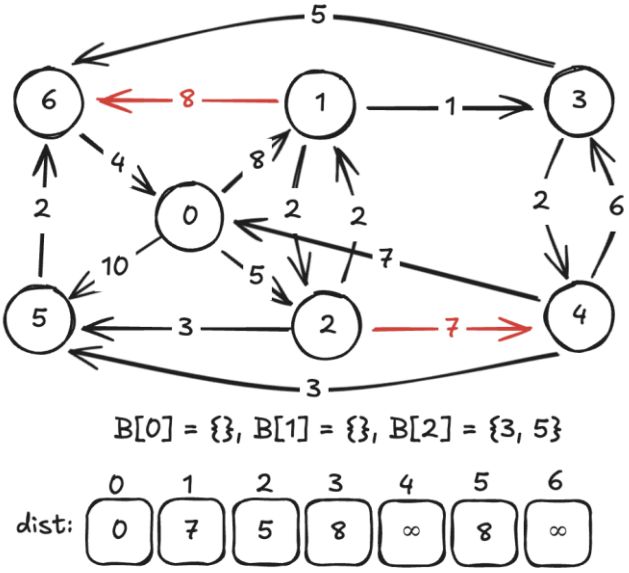

Рис. 11. Пример применения алгоритма дельта-шага. Второй шаг.

Аналогично обрабатываются сначала «легкие», а затем «тяжелые» ребра для вершин из корзины *B*[2]. На рис. 12 отмечены все ребра, которые будут обработаны на третьем шаге. При этом параллельно можно выполнять каждый шаг релаксации ребер – и «легких», и «тяжелых».

Подбор правильного параметра Δ очень важен: при Δ = 0 алгоритма дельта-шага превращается в алгоритм Дейкстры, а при Δ = ∞ – в алгоритм Беллмана – Форда.

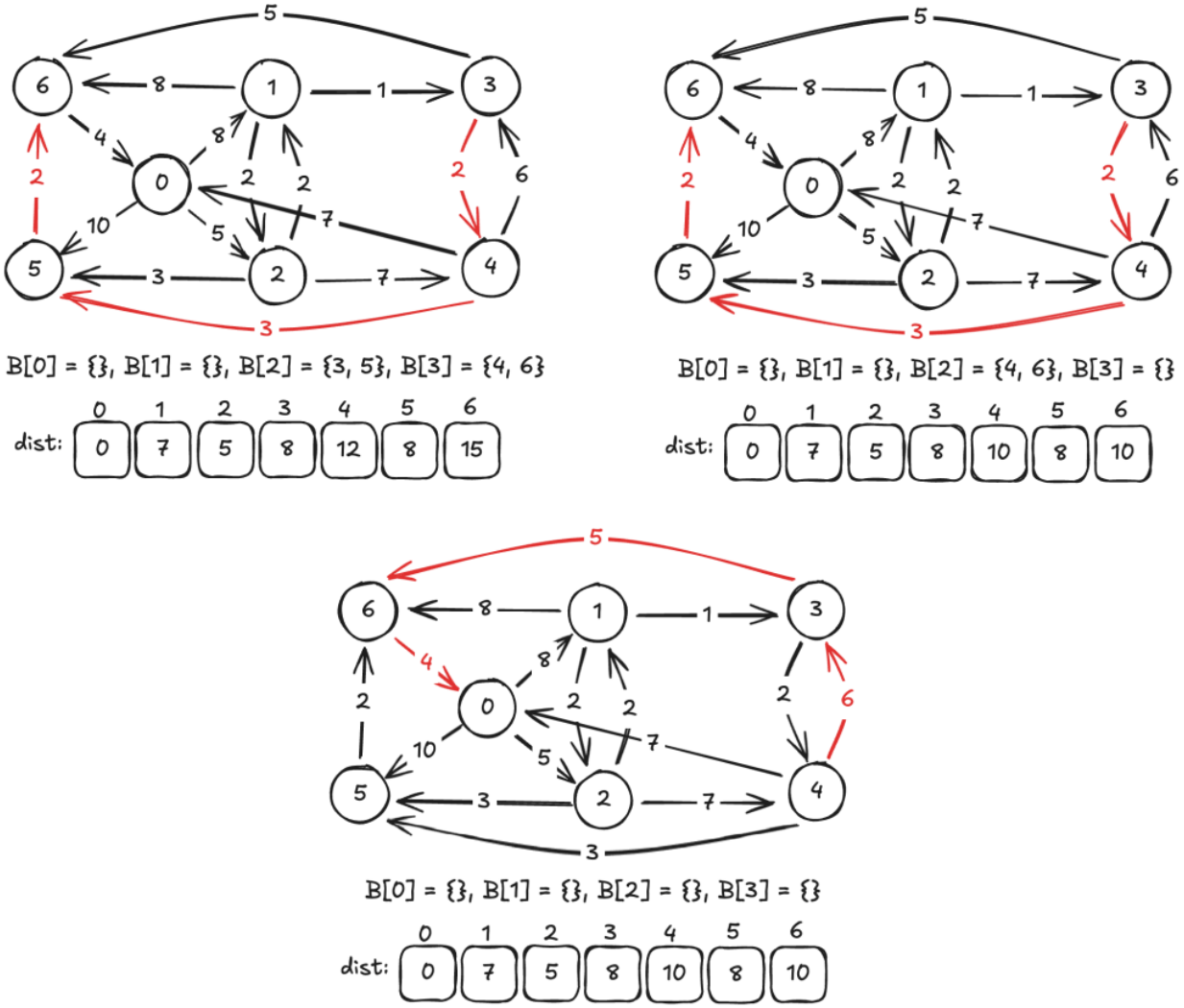

Рис. 12. Пример применения алгоритма дельта-шага. Третий шаг.

Разработанные параллельные модификации классических алгоритмов адаптированы к вычислительным средам с возможностью масштабирования и последующего исполнения на гетерогенных устройствах.

РЕАЛИЗАЦИЯ И ИССЛЕДОВАНИЕ ПРИЛОЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ СТАНДАРТОВ SYCL и OPENMP

Приведем результаты исследования разработанных параллельных модификаций с использованием стандартов SYCL и OpenMP для решения задачи поиска кратчайшего пути в графе.

Стандарт SYCL как универсальная модель программирования

Современный этап технологического развития IT-индустрии предъявляет повышенные требования к вычислительным ресурсам. Возрастающая сложность вычислений требует новых подходов и решений, способных эффективно использовать потенциал смешанных вычислительных систем. Все чаще для решения ресурсоемких задач применяют гетерогенные системы, объединяющие в рамках одной архитектуры разные процессоры. Как правило, это CPU, GPU, а также специализированные ускорители, такие как FPGA или ASIC. Основная идея здесь проста: использовать каждое устройство строго по его профилю, отдавая, например, параллельные задачи на GPU, а ветвистую логику – на CPU. Все это делается ради увеличения общей производительности и снижения времени вычислений. Такой подход открывает широкие горизонты, но и ставит перед разработчиками немало задач: от грамотного распределения нагрузки до согласования взаимодействия между компонентами.

В этой связи возникает потребность в инструментах, способных абстрагировать сложность работы с такими системами. Одним из наиболее удачных решений стал стандарт SYCL [1, 2] – он предлагает удобную и в то же время гибкую модель программирования для гетерогенных платформ. С его помощью можно написать код, который одинаково успешно выполняется на CPU, GPU и других ускорителях, не вдаваясь каждый раз в тонкости конкретного компонента. SYCL предлагает разработчику набор понятий и инструментов, позволяющих сфокусироваться на логике приложения, не теряя контроля над производительностью.

Надстройкой над SYCL служит язык DPC++ (Data Parallel C++) [3], созданный компанией Intel. Он расширяет стандарт, добавляя инструменты более тонкой настройки параллельных вычислений. Это особенно ценно в задачах,

связанных с научными расчетами, машинным обучением и высокопроизводительной аналитикой данных.

Следует подчеркнуть, что стандарт SYCL совместно с DPC++ можно рассматривать как универсальную модель программирования высокого уровня, представляющую современный и мощный инструмент для построения переносимых, эффективных и масштабируемых приложений на архитектурах с несколькими типами вычислительных устройств.

Результаты тестирования приложений

Разработанные параллельные приложения были протестированы на следующих аппаратных средствах: центральный процессор Intel Core i5-12400F (6 ядер, 12 потоков), графический процессор NVIDIA GeForce GTX1660 (1408 CUDA-ядер). Используемое программное обеспечение: SYCL 2020 (revision 10), DPC++ 2025.1, OpenMP 5.1, C++20.

Были проведены исследование и сравнение пяти реализаций:

– последовательная с использованием C++;

– параллельная с использованием DPC++ исполняемая на CPU;

– параллельная с использованием DPC++ исполняемая на CPU+GPU;

– параллельная с использованием OpenMP исполняемая на CPU;

– параллельная с использованием OpenMP исполняемая на CPU+GPU.

Основное отличие последовательной программы от параллельной состоит в том, что внутренний цикл запускается параллельно по всем ребрам. В стандарте SYCL это делается с помощью метода parallel_for, а в стандарте OpenMP – с помощью директивы #pragma omp parallel for. При этом если нашлась вершина, расстояние до которой может быть улучшено, то нужно обеспечить атомарность этого сравнения и присвоения нового значения.

Для тестирования был использован случайно сгенерированный граф, сформированный таким образом, что каждая его вершина соединена с другой вершиной с некоторой заданной вероятностью. Случайные графы имеют множество применений, например позволяют моделировать случайные социальные связи, изучать взаимодействия между людьми в Интернете.

Программа для тестирования генерирует случайный граф с заданными количеством вершин и вероятностью ребер между вершинами и случайной

длиной ребер в промежутке от 1 до 100, запускает выполнение реализованных алгоритмов нахождения кратчайших путей в графе, сравнивает найденные величины путей и время выполнения алгоритма.

Компиляция кода производилась с соответствующими параметрами, указывающими целевую архитектуру исполняемого приложения. В итоге получили пять исполняемых приложений для каждого из алгоритмов, указанных выше.

Для визуализации сравнения времени выполнения различных вариантов реализаций алгоритма Беллмана – Форда была разработана программа на языке Python 3.12, которая запускает программные реализации и строит графики зависимости времени их выполнения от количества вершин, а также от ускорения относительно последовательной реализации алгоритма.

На рис. 13 представлены графики зависимости ускорения алгоритма Беллмана – Форда от количества вершин в случайном графе при вероятности ребер 0.8 между вершинами.

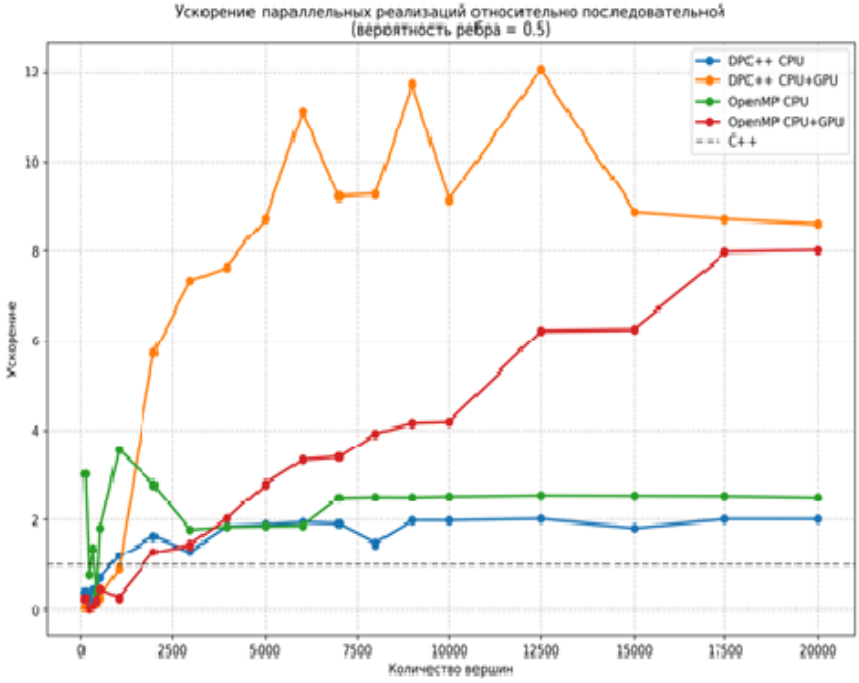

Рис. 13. Зависимость ускорения реализаций алгоритма Беллмана – Форда от количества вершин в графе с вероятностью ребра 0.8.

На рис. 14 приведены графики зависимости ускорения каждой разработанной реализации относительно последовательной версии для алгоритма

Беллмана – Форда от количества вершин в случайном графе с вероятностью 0.8 ребер между вершинами.

Анализ полученных графиков позволил сделать вывод, что реализации алгоритма с использованием гетерогенной архитектуры с CPU и GPU процессорами выполняются примерно в 10 раз быстрее последовательного алгоритма. При этом с увеличением размерности графа ускорение также увеличивается.

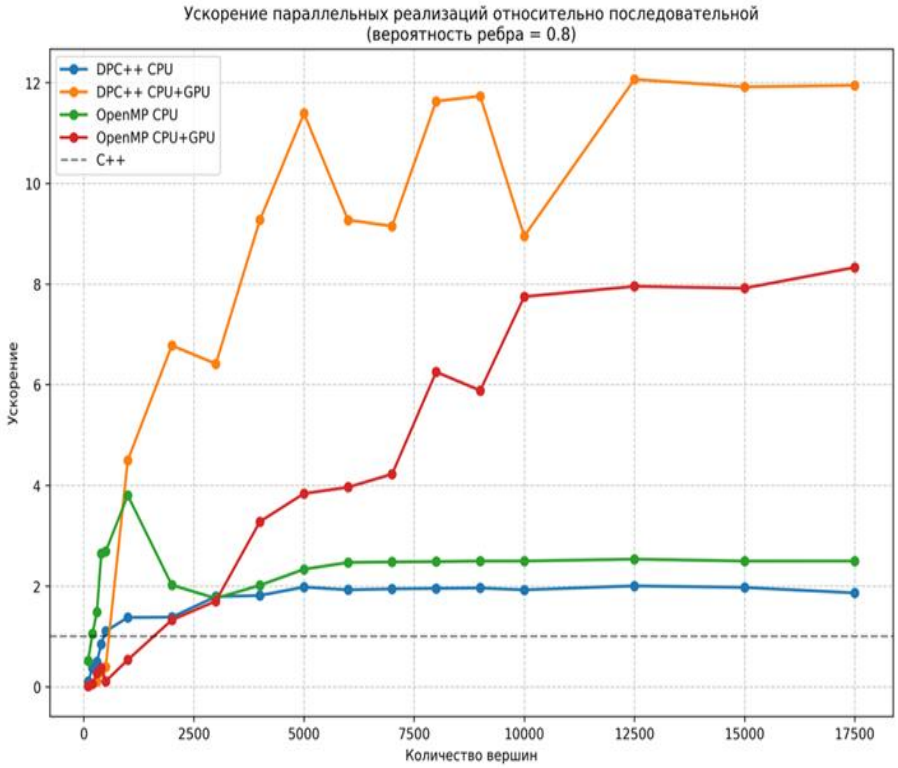

Рис. 14. Зависимость ускорения выполнения реализаций алгоритма Беллмана – Форда от количества вершин в графе с вероятностью ребра 0.8.

На графах небольшой размерности использование гетерогенных многоядерных архитектур не дает положительного эффекта, а, наоборот, ухудшает время выполнения алгоритма. Параллельные реализации с использованием только CPU оказались быстрее последовательного алгоритма примерно в 2 раза. Во всех случаях чем выше размерность графа, тем эффективней использование гетерогенной архитектуры и языка DPC++.

Такое ускорение на GPU получается за счет того, что алгоритм Беллмана – Форда очень хорошо подходит для массивного параллелизма и не требует интенсивной синхронизации данных между процессорами. Это позволяет исполь-

зовать преимущества такой гетерогенной архитектуры, состоящей из CPU и GPU.

Проанализируем время выполнения и ускорение реализаций метода дельта-шага от количества вершин. Как уже отмечалось выше, выбор параметра Δ очень важен для получения лучшего результата. На рис. 15–18 представлены графики зависимости времени выполнения и ускорения для алгоритма дельта-шага от количества вершин с различными значениями параметра Δ = 1 и Δ = 100 для каждой разработанной реализации. Отметим, что в зависимости от выбора параметра Δ время выполнения алгоритма может варьироваться до 20%. Самой быстрой оказалась параллельная реализация только с использованием CPU и стандарта OpenMP.

В реализации алгоритма дельта-шага некоторые переменные были использованы и внутри параллельно исполняющейся части. Это делает неоптимальным использование графического процессора для выполнения параллельной части кода, поскольку эти переменные необходимо часто переносить между устройствами, чтобы поддерживать их значения одинаковыми на обоих процессорах (CPU и GPU).

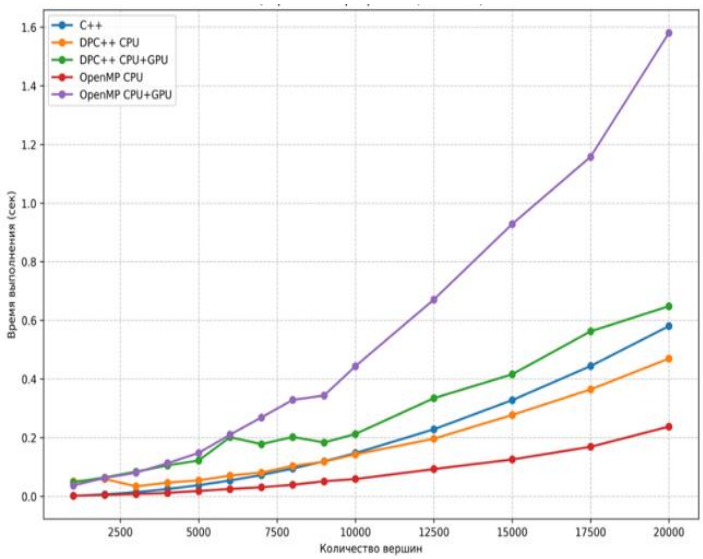

Рис. 15. Зависимость времени выполнения реализаций алгоритма дельта-шага от количества вершин в графе, вероятность ребра 0.9, Δ = 1.

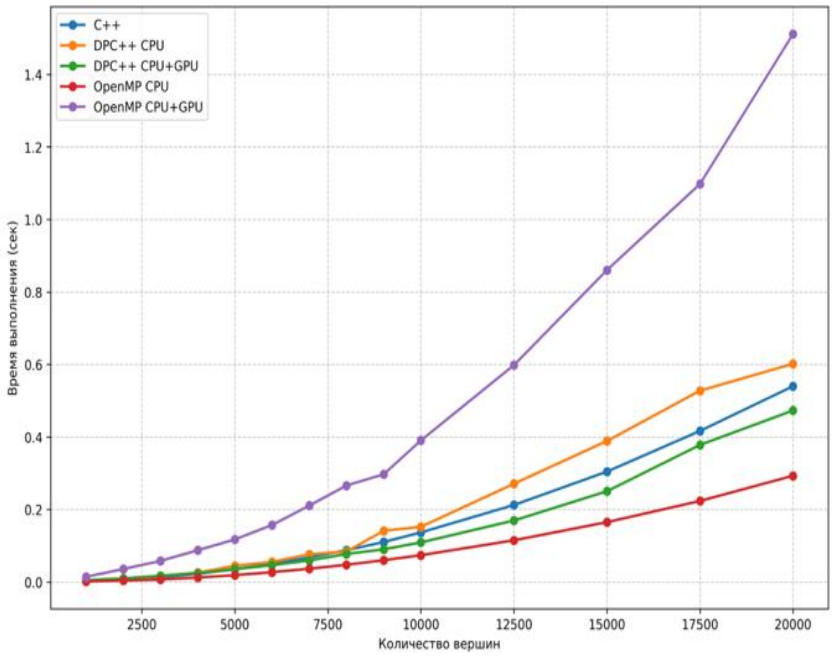

Рис. 16. Зависимость времени выполнения реализаций алгоритма дельта-шага от количества вершин в графе, вероятность ребра 0.9, Δ = 100.

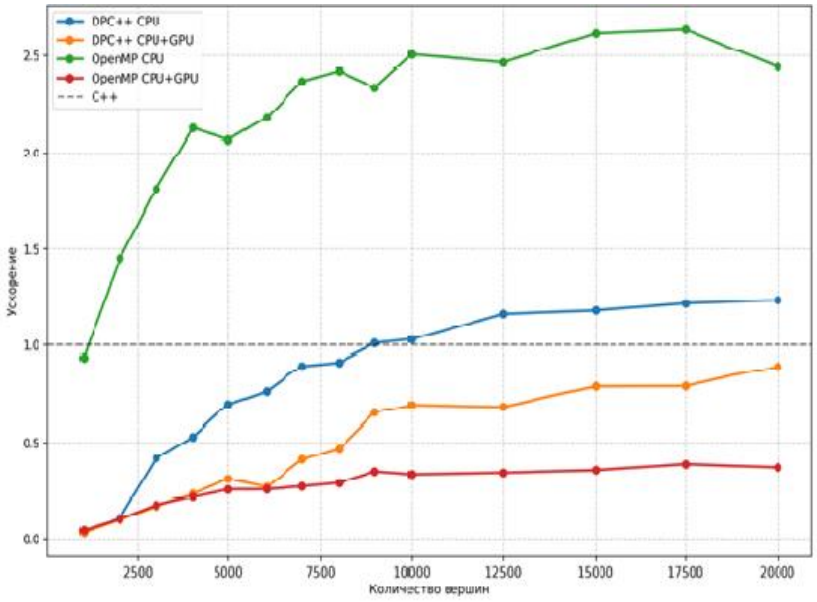

Рис. 17. Зависимость ускорения алгоритма дельта-шага от количества вершин в графе с вероятность ребра 0.9, Δ = 1.

Постоянное переключение для работы с памятью разных процессоров добавляет накладные расходы, на которые уходит большая часть времени выполнения алгоритма.

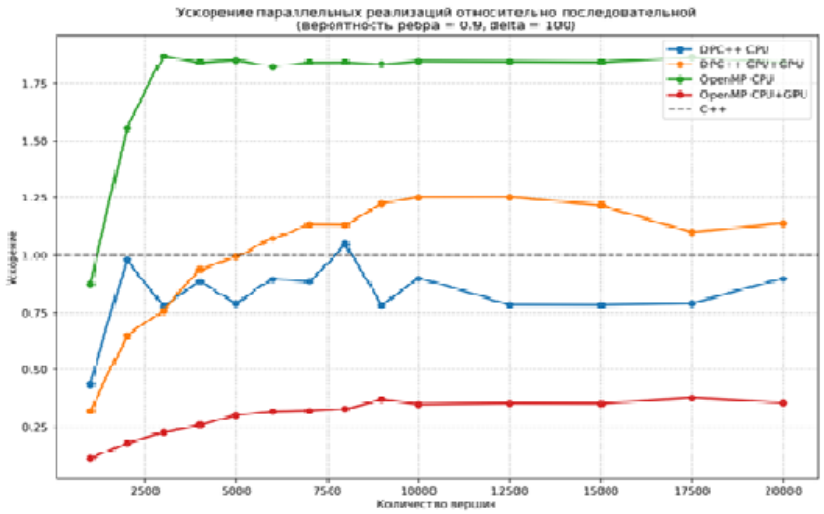

Рис. 18. Зависимость ускорения выполнения реализаций алгоритма дельта-шага от количества вершин в графе, вероятность ребра 0.9, Δ = 100.

В процессе реализации и тестирования приложений в рамках выполненных научно-исследовательских работ студентов было установлено, что алгоритм Беллмана – Форда гораздо более эффективен для использования в гетерогенных средах. Реализация алгоритма демонстрирует значительное ускорение при выполнении на центральном и графическом процессорах, особенно при большой размерности графа. Реализация с использованием стандарта OpenMP позволила добиться ускорения до 8 раз, а с использованием стандарта SYCL – до 12 раз по сравнению с последовательной версией.

Для алгоритма дельта-шага распределение задач между CPU и GPU дало менее однозначный результат: несмотря на параллелизм, заложенный в структуре алгоритма, необходимость частого переноса данных между устройствами и синхронизации состояний значительно уменьшала ускорение. Тестирование показало, что лучшую производительность обеспечивала реализация на OpenMP в многопоточном режиме на центральном процессоре, без привлечения ускорителей.

Полученные в ходы тестирования результаты подтвердили, что эффективность подхода значительно варьируется в зависимости от специфики алгоритма и используемой архитектуры. Гетерогенные решения показали преимущество в задаче с высокой степенью независимого параллелизма при минимальной необходимости синхронизации данных. Для более сложных схем, как

в алгоритме дельта-шага, эффективнее оказалось использование только CPU в многопоточном режиме.

ЗАКЛЮЧЕНИЕ

Представлены исследования возможностей гетерогенных архитектур и стандартов SYCL и DPC++, а также OpenMP для эффективного решения задачи поиска кратчайших путей в графах различных размерностей. Исследованы реализации двух разных алгоритмов решения задачи.

Алгоритм Беллмана – Форда продемонстрировал отличную масштабируемость и легко поддается распараллеливанию: реализация на DPC++ с GPU показала наиболее значительное ускорение по сравнению с последовательной версией. Для алгоритма дельта-шага, напротив, использование GPU не дало заметного ускорения из-за сложности согласования памяти и необходимости многократного копирования данных между устройствами. Лучшие результаты были достигнуты при использовании многопоточной OpenMP-реализации.

Было проведено тестирование с использованием следующих аппаратных средств: центральный процессор Intel Core i5-12400F (6 ядер, 12 потоков), графический процессор NVIDIA GeForce GTX1660 (1408 CUDA-ядер). Тестирование позволило выявить зависимости производительности от особенностей алгоритма и архитектуры аппаратной платформы. Полученные данные показывают, что даже в рамках одной вычислительной задачи различные реализации могут демонстрировать разное поведение при изменении плотности графа, числа вершин или параметров исполнения. Это подтверждает важность предварительного анализа и тестирования при выборе программной архитектуры для задач, требующих высокой производительности.

Разработанный комплекс программ визуализирует зависимость ускорения параллельных реализаций алгоритма от числа вершин в графе при различных параметрах метода и успешно используется в учебно-научном процессе кафедры ПМИИ ФГБОУ ВО “НИУ «МЭИ».

Таким образом, основной вывод заключается в необходимости учитывать баланс между универсальностью и спецификой во всех уровнях проектирования вычислительных решений: от выбора алгоритма до конкретных реализаций под выбранную платформу. Современные инструменты, такие как SYCL

и DPC++, безусловно, расширяют горизонты параллельного программирования и делают разработку под гетерогенные архитектуры более доступной. Однако, как было показано, их применение требует внимательного соотнесения теоретической модели задач и практической архитектуры исполнения, только в этом случае можно рассчитывать на достижение действительно ощутимых ускорений.

Результаты, полученные в ходе выполнения работы, не только демонстрируют работоспособность выбранных технических подходов, но и открывают возможности дальнейших исследований в направлении гибридных стратегий вычислений, автоматической настройки параметров алгоритмов под конкретную архитектуру, а также разработки обобщенных фреймворков, которые могли бы адаптивно выбирать тип исполнения в зависимости от характера входных данных.

Благодарности

Авторы благодарны выпускнику кафедры ПМИИ НИУ «МЭИ» М. Д. Ильину за реализацию и тестирование алгоритмов поиска критического пути в графах с использованием SYCL и DPC++, выполненных в рамках НИР. Исходный код программ доступен по ссылке https://github.com/maximxD/graphs.

СПИСОК ЛИТЕРАТУРЫ

1. SYCL 2020 Specification (revision 11). URL: https://registry.khronos.org/SYCL/specs/sycl-2020/html/sycl-2020.html (дата обращения: 12.05.2026)

2. SYCL Reference. URL: https://github.khronos.org/SYCL_Reference/index.html (дата обращения: 12.05.2026)

3. *Reinders James, Ashbaugh Ben, Brodman James, Kinsner Michael, Pennycook John, Tian Xinmin.* Data Parallel C++: Programming Accelerated Systems Using C++ and SYCL. Springer Nature, 2023. 630 p. https://doi.org/10.1007/978-1-4842-9691-2

4. OpenMP standard. URL: https://www.openmp.org (дата обращения: 12.05.2026)

5. *Роберт Седжвик.* Фундаментальные алгоритмы на C++. Алгоритмы на графах: Пер. с англ. СПб: ООО «ДиаСофт», 2002. С. 287–304.

6. Алгоритмы: построение и анализ. Пер. с англ. М.: Издательский дом «Вильямс», 2011. С. 672–676.

7. *Meyer U., Sanders P.* Δ-stepping: a parallelizable shortest path algorithm // J. of Algorithms. 2003. Vol. 49, No. 1. P. 114–152.

8. *Crauser, A., Mehlhorn, K., Meyer, U., Sanders, P.* (1998). A parallelization of Dijkstra's shortest path algorithm. In: Brim, L., Gruska, J., Zlatuška, J. (eds) Mathematical Foundations of Computer Science 1998. MFCS 1998. Lecture Notes in Computer Science, vol 1450. Springer, Berlin, Heidelberg. https://doi.org/10.1007/BFb0055823

INVESTIGATION OF PARALLEL APPLICATIONS FOR HETEROGENEOUS ARCHITECTURES USING THE SYCL AND OPENMP STANDARDS

O. Yu. Shamayeva[1] [0009-0006-3041-6682], A. M. Chernetsov[2] [0000-0001-7655-2395], L. A. Zinchenko[3] [0000-0002-2298-8721]

[1, 2]*National Research University «Moscow Power Engineering Institute», Moscow, Russia*

[3]*Bauman Moscow State Technical University, Moscow, Russia*

[1] Shamayevaoy@mpei.ru, [2]chernetsovam@mpei.ru, [3]lyudmilaaa@mail.ru

Abstract

The results of research on parallel applications for heterogeneous architectures using the SYCL standards and the DPC++ language, as well as OpenMP, developed by students of the Department of Applied Mathematics and Artificial Intelligence (AM&AI) at MPEI during research work are presented.

As an applied problem, the search for shortest paths in graphs of various dimensions using the Bellman–Ford and delta‑step algorithms is considered. The impact of algorithm choice on the solution efficiency, as well as the heterogeneous multi-core architecture, is examined.

The Bellman-Ford algorithm is easily parallelized and demonstrated excellent scalability. In contrast, using a GPU for the delta-step algorithm did not provide significant speedup due to the complexity of memory coordination and the need to repeatedly copy data between devices. Testing was conducted using the following hardware: an Intel Core i5-12400F CPU (6 cores, 12 threads) and an NVIDIA GeForce GTX1660 GPU (1408 CUDA cores). The test results show that even within a single computational task, different implementations can exhibit different behavior when the task parameters are changed. This confirms the importance of preliminary analysis and testing when choosing a hardware&software architecture for tasks requiring high performance.

The developed software suite visualizes the dependence of parallel algorithm implementation speedup on the number of vertices in the graph for various method parameters and is successfully used in the educational and research processes of the AM&AI Department at MPEI.

Keywords: *heterogeneous architecture, the task of finding shortest paths in a graph, SYCL standard, DPC++ language, OpenMP technology.*

REFERENCES

1. SYCL 2020 Specification (revision 11). URL: https://registry.khronos.org/SYCL/specs/sycl-2020/html/sycl-2020.html (date accessed: 12.05.2026)

2. SYCL Reference. URL: https://github.khronos.org/SYCL_Reference/index.html (date accessed: 12.05.2026)

3. *Reinders James, Ashbaugh Ben, Brodman James, Kinsner Michael, Pennycook John, Tian Xinmin.* Data Parallel C++: Programming Accelerated Systems Using C++ and SYCL. Springer Nature, 2023. 630 p. https://doi.org/10.1007/978-1-4842-9691-2

4. OpenMP standard. URL: https://www.openmp.org (date accessed: 12.05.2026)

5. *Robert Sedgewick.* Algorithms in C++: Graph algorithms Addison-Wesley, 2002, 496 p.

6. Algorithms: construction and analysis, 2nd edition. Translated from

English. Moscow: Williams Publishing House, 2011. 676 p.: ill. Parallel title English.

7. *Meyer U., Sanders P.* Δ-stepping: a parallelizable shortest path algorithm // J. of Algorithms. 2003. Vol. 49, No. 1. P. 114–152.

8. *Crauser, A., Mehlhorn, K., Meyer, U., Sanders, P.* (1998). A parallelization of Dijkstra's shortest path algorithm. In: Brim, L., Gruska, J., Zlatuška, J. (eds) Mathematical Foundations of Computer Science 1998. MFCS 1998. Lecture Notes in Computer Science, vol 1450. Springer, Berlin, Heidelberg. https://doi.org/10.1007/BFb0055823

СВЕДЕНИЯ ОБ АВТОРАХ

ШАМАЕВА Ольга Юрьевна – кандидат технических наук, доцент кафедры Прикладной математики и искусственного интеллекта Национального исследовательского университета «МЭИ», г. Москва, Россия; *о*бласть научных интересов: параллельные алгоритмы и параллельные вычисления.

Olga Yurevna SHAMAYEVA – Candidate of Sciences (Engineering), Associate professor, associate professor at the Department of Applied mathematics and Artificial Intelligence, National Research University «Moscow Power Engineering Institute», Moscow, Russia; research interests: parallel algorithms and parallel computing.

email: Shamayevaoy@mpei.ru

ORCID: 0009-0006-3041-6682

ЧЕРНЕЦОВ Андрей Михайлович – кандидат технических наук, доцент; доцент кафедры Прикладной математики и искусственного интеллекта Национального исследовательского университета «МЭИ».

Andrey Mikhailovich CHERNETSOV – candidate of Technical Sciences, Associate Professor; associate professor at the Department of Applied Mathematics and Artificial Intelligence, National Research University «Moscow Power Engineering Institute».

email: chernetsovam@mpei.ru

ORCID: 0000-0001-7655-2395

ЗИНЧЕНКО Людмила Анатольевна – доктор техн. наук, профессор, профессор МГТУ им. Н.Э. Баумана.

Lyudmila Anatolyevna ZINCHENKO – Doctor of Technical Sciences, professor, Professor at the Department IU4, BMSTU.

email: lyudmillaa@mail.ru

ORCID: 0000-0002-2298-8721

Материал поступил в редакцию 25 мая 2026 года