Московский государственный университет имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра системного анализа

Егоров Кирилл Юлианович

Построение параллельных алгоритмов для решения задачи быстродействия с фазовыми ограничениями

Курсовая работа

Научный руководитель
к.ф.-м.н., доцент И. В. Востриков

Москва, 2022

Содержание

1 Введение

Работа посвящена разработке параллельных алгоритмов для поиска оптимального по быстродействию пути между двумя точками на некоторой ограниченной территории с препядствиями, которые мешают движению. В работе эти препядствия представляют собой произвольные области, в которых невозможно дальнейшее движение.

Для представления в виде классической *задачи кратчайшего пути* исходная задача дискретизуются. Численное решение уравнения Гамильтона–Якоби–Беллмана, к которому сводится непрерывный вариант возможно (по крайней мере, в [4] предложен способ решения после нескольких приближений и дискретизации уравнения), но дискретизация задачи в целом — более популярное решение.

Необходимость решения задачи кратчайшего пути в двумерном пространстве возникает в различных сферах, но в первую очередь в транспортной и логистической. Сложно представить будущее, где проблема быстрой доставки товаров или безопасного перемещения людей перестала быть актуальной. Данная задача является классической и традиционно решается методом динамического программирования[1].

Последовательные алгоритмы, решающие данную задачу хорошо изучены и представлены в статьях [2] и [3]. Эти алгоритмы были адаптированы под поставленную задачу, и на их основе были разработаны параллельные варианты.

Было уделено вниманию различным способам организации параллелелизма: на многоядерном процессоре, на графическом процессоре, на многонодной компьютерной установке. Для каждого способа описан соответствующий алгоритм, для двух видов параллелелизма разработаны программы и замерены времена работы.

2 Постановка задачи

Рассмотрим управляемый объект, положение которого задается динамической системой дифференциальных уравнений

$$\frac{dx}{dt} = f(t, x, u) \tag{1}$$

на промежутке $t_0 \leqslant t \leqslant t_1$ с заданным начальным состоянием $x(t_0) = x^0$.

Такое описание естественно для движений, подчиняющихся второму закону Ньютона. В нашем случае $x = [x_1, x_2, v_1, v_2] \in \mathbb{R}^4,\ v_i = \dot{x}_i,\ i = \{1, 2\}$. Наложим на фазовые координаты следующие ограничения:

$$[x_1, x_2] \in \Omega, \quad [v_1, v_2] \in \Omega_v.$$

Для задачи Коши (1) поставим задачу поиска управления $u \in U$, минимизирующего следующий интегральный функционал:

$$J(u) = \int_{t_0}^{t_1} g(x(t), u(t))\, dt \to \min_{u \in U}. \tag{2}$$

Будем пользоваться методом динамического программирования. Введем функцию цены

$$V(t, x) = \min_{u \in U} \int_{t}^{t_1} g(x(t), u(t))\, dt,$$

$$V(t_1, x^1) = 0.$$

Согласно [5], если предполагать непрерывную дифференцируемость функции цены, решение задачи равносильно решению уравнения Гамильтона–Якоби–Беллмана:

$$\min_{u(t) \in U} \left\{ g(x, u) + \sum_{i=1}^{n} \frac{\partial V(x)}{\partial x_i} f_i(x, u) \right\} = 0. \tag{3}$$

Замечание. Мы рассматриваем частный случай этой задачи — задачу быстродейсвия, то есть задачу с функционалом качества $g(x, u) \equiv 1$. В этих условиях уравнение Гамильтона–Якоби–Беллмана (3) будет записано в следующем виде:

$$\min_{u \in U} \langle \nabla t(x), f(x, u) \rangle = -1.$$

Вместо дискретизации и последующего решения получившегося уравнения мы дискретизируем исходную задачу для приведения ее к виду задачи кратчайшего пути.

Пусть Π — минимальный прямоугольник, вмещающий в себя целиком множество Ω. Введем на нем равномерную сетку с шагом ε:

$$(i, j) : \ 1 \leqslant i \leqslant N,\ 1 \leqslant i \leqslant M \tag{4}$$

$$\Xi = \left\{ (x_i, y_j) \in \Pi, x_i = \varepsilon \frac{i}{N}, y_j = \varepsilon \frac{j}{M} \right\}. \tag{5}$$

Мы будем считать, что $x^0, x^1 \in \Xi$. Перемещение возможно только в "соседние" узлы сетки, содержащиеся в множестве фазового ограничения Ω. Назовем такое множество *возможным*:

$$\text{possible}(i, j) = \{(i, j) \mid i \in \{i, i \pm 1\}, j \in \{j, j \pm 1\}, (x_i, y_j) \in \Omega\}$$

Предположение. На каждом этапе пути можно ехать лишь с некоторой одной скоростью и скорость на предыдущем участке не влияет на время прохода следующего участка.

Данное предположение кажется резонным, если мы рассматриваем, например, автомобиль или человека, а шаг сетки — несколько десятков метров: такого расстояния должно хватить для изменения скорости на оптимальную для данного типа поверхности. Для объектов, не удовлетворяющих этому свойству, такое предположение не подходит. Предположение позволяет сократить размернось рассматриваемого фазового пространства с 4 до 2, но превращает управление в импульсное.

Тогда, мы можем считать известным время перехода между точкой (i, j) и любой точкой из ее возможного множества:

$$d_{i,j}(\hat{i}, \hat{j}), \text{ где } (\hat{i}, \hat{j}) \in \text{possible}(i, j).$$

Таким образом, получили матричное уравнение в неявном виде, которое предстоит решить:

$$\begin{aligned} V_{i,j} &= \min_{(\hat{i},\hat{j}) \in \text{possible}(i,j)} \{d_{i,j}(\hat{i}, \hat{j}) + V_{\hat{i},\hat{j}}\}, \\ V_{i_1,j_1} &= 0. \end{aligned} \tag{6}$$

3 Алгоритм Беллмана–Форда

Данный раздел содержит описание алгоритма Беллмана–Форда в применении к нашей задачи, а также в процессе описаны общие принципы работы построенных алгоритмов, которые будут использованы и в дальнейшем.

Традиционно для алгоритмов данного типа (алгоритм поиска кратчайшего пути) маркирование узлов сетки предполагаемым оптимальным значением функционала. Маркировка одного и того же узла может происходить один или несколько раз, но в результате работы программы маркировки каждой вершины должны соответствовать оптимальному значению функционала качества для данной вершины. Поэтому далее мы будем обозначать маркировку как $V(i,j)$, а перемаркирование как $V(i,j) \leftarrowtail v$. Изначально предполагается, что все вершины, кроме целевой, имеют маркировку $+\infty$. Целевая же вершина (i_1, j_1) имеет маркировку 0.

Ниже приведено описание алгоритма:

1. Необходимо провести $(NM-1)$ итерации алгоритма.

2. На каждой итерации происходит полный проход по сетке. Далее (i,j) — позиция при проходе.

3. Для каждой вершины из возможного множества $(\hat{i},\hat{j}) \in \text{possible}(i,j)$ происходит перемаркировка: в случае, если $V(i,j) > d_{i,j}(\hat{i},\hat{j}) + V(\hat{i},\hat{j})$, перемаркируем
$$V(i,j) \leftarrowtail d_{i,j}(\hat{i},\hat{j}) + V(\hat{i},\hat{j}).$$

Заметим, что оптимальная траектория не может иметь более $(NM-1)$ перемещения. Таким образом алгоритм действительно ищет кратчайший путь.

3.1 Распараллеливание на CPU

Ядра центрального процессора — это высокопроизводительные процессоры, которые есть в каждом компьютере. В стандартном персональном компьютере от 4 до 16 ядер. Пусть на компьютере имеется C ядер.

Предлагается распараллелить проход по сетке по переменной i. Разобьем сетку на $(C-1)$ штуку размером $\left\lfloor \frac{N}{C} \right\rfloor \times M$ и одну размера $N-(C-1)\left\lfloor \frac{N}{C} \right\rfloor \times M$. Каждое ядро должно обрабатывать собственную матрицу.

Возникающую проблему синхронизации процессов при обработке соседних подматриц предлагается решать массивом мьютексов —примитивов синхронизации, обеспечивающих взаимное исключение исполнения критических участков кода.

Перед началом основной программы каждый мьютекс из массива A размера C переводится в закрытое состояние. По окончании обработки $c \in \{0, \ldots, C-1\}$ процессом первой строки процесс переводит мьютекс A_c в открытое состояние. Перед началом обработки последней строки процесс c дожидается открытия мьютекса A_{c+1}, если $c+1 < C$.

3.2 Распараллелевание на GPU

Параллельные вычисления на графическом процессоре в последние годы очень активно развиваются. Это связано с типичной архитектурой таких процессоров: это большое количество маломощных процессоров с общей памятью. Почти всегда такие вычисления используются для майнинга криптовалюты.

На сегодняшний стандартной архитектурой для написания программ на GPU является CUDA SDK, поддерживаемая исключительно графическими картами компании Nvidia. Ввиду отсутствия данного графического процессора, мы ограничимся лишь описанием алгоритма, в сравнении времени работы программ данный алгоритм принимать участие не будет.

Предлагается вместо одного прохода по сетке на каждой итерации, проходить 8 раз (по числу максимальной мощности возможного множества). При этом каждый узел при таком проходе будет обработан параллельно. На каждом из восьми проходов будет обработано только одно возможное направление движения, причем у всех узлов оно должно совпадать. Таким образом мы обеспечиваем синхронизацию.

3.3 Распараллеливание на многонодной установке

Предлагается использовать модель *master-follower*. Пусть имеется одна *master* нода, с которой осуществляется оркестрация установки, и L *follower* нод с $C_l, l = \overline{1, L}$ доступных ядер процессора, используемых непосредственно для вычислений. Предполагается, что между нодами настроена сеть и известны адреса всех сервисов.

Программа поставляется в виде http-сервиса вычислителя, и основной программы, которая общается с вычислителями посредством http запросов. На *follower* нодах запускаются сервисы вычислителя в количестве, соответствующим количеству доступных ядер процессора.

Параллелелизм осуществляется как и в случае распараллелевания на CPU. Но из-за отсутствия общей памяти возникает проблема синхронизации данных между всеми вычислителями. Для этого по результатам обработки основная программа должна передать вычислителям минимальное инкрементальное изменение состояния для данного вычислителя. Для этого последняя строка каждого батча считается основной программой еще один раз, после чего эта строка отправлется вычислителю для синхронизации.

4 Алгоритм Дейкстры

Алгоритму Дейкстры — это самый популярный алгоритм поиска кратчайшего пути. Из-за меньшей, чем у алгоритма Беллмана–Форда алгоритмической сложности, данный алгоритм позволяет производить вычисления на сетках на порядок большего размера.

Алгоритм использует две дополнительные структуры: множество обработанных узлов и можество граничных узлов. Данные структуры имплементированы как хэш-таблицы.

Ниже приведено описание алгоритма:

1. На начало алгоритма в множестве граничных узлов находится только целевой узел.

2. На каждой итерации алгоритма из множества граничных узлов выбирается узел с минимальной маркировкой.

3. Этот узел добавляется в множество обработанных узлов и удаляется из граничного множества.

4. Затем из возможного множества выбираются узлы, которые не находятся в множестве обработанных узлов. Эти узлы добавляются в множество граничных вершин, маркировка таких вершин обновляется.

Алгоритм останавливается в случае, если на некоторой итерации алгоритма был выбран начальный узел (i_0, j_0).

4.1 Распараллелевание на CPU

Основной затратной операцией в приведенном алгоритме является поиск минимума на каждой итерации. Поэтому параллелелизм будет встроен именно в этот этап.

Для этого предлагается хранить граничное множество в C хэш-таблицах. Добавлять узел в наименее наполненную таблицу. Тогда каждый процесс ищет минимум в своей хэш-таблице, а затем основной процесс ищет минимум в массиве из C элементов.

4.2 Распараллелевание на многонодной установке

Для обеспечения синхронизации снова пользуемся инкрементальным обновлением. Заметим, что за одну итерацию алгоритма происходит одно добавление в множество обработанных вершин, и не более 7 добавлений в граничное множество. Этой информации достаточно для поддержания всех вычислителей в актуальном состоянии, поэтому она должна сообщаться сервисам-вычислителям после каждой проведенной итерации.

5 Описание программного решения

Программное решение представлено на языке *Go*. Причины для использования этого языка при написании вычислительных параллельных задач следующие:

1. Go — компилируемый язык со встроенной сборкой мусора. Это необычное сочетание качеств колоссально повышает скорость разработки, при этом язык проигрывает в производительности языку C всего в 1,5–2 раза.

2. Go представляет современные средства параллелелизма *go routines*. В отличие от классических потоков (threads) рутины управляются не операционной системой, а *go runtime*, за счет чего значительно сокращаются накладные расходы на создание таких рутин и на коммуникацию между ними.

Помимо основных алгоритмов были написаны сопутствующие программы для генрации начальных данных и визуализации результатов. Генерация карты высот происходит посредством последовательного сложения нескольких шумов Перлина с подобранными различными параметрами для получения похожего на земной ландшафта [6]. Карту с различными типами почв и фазовыми ограничениями предлагается самостоятельно нарисовать пользователю в любом графическом редакторе, например, Microsoft Paint, перед запуском программы.

В случае распараллелевания на многонодной установке, программа поставляется в виде двух docker-образов и Kubernetes манифестов для последовательного применения для деплоя на кластер, оркестрируемый Kubernetes. На других облачных оркестраторах решение не проверялось. Написание Kubernetes оператора для данного решения видится нецелесообразным, ввиду отсутствия у программы сложного жизненного цикла.

Визуализация сделана с помощью платформо-независимой спецификации OpenGL, что позволяет собирать все компоненты под различные операционный системы. Ниже представлены несколько примеров работы программы. Все алгоритмы выдают одинаковый результат, однако всвязи с разной асимптотической сложностью, алгоритм Беллмана–Форда имеют размер сетки на порядок меньше, чем для алгоритма Дейстры.

В примерах ниже, время перехода принято равным

$$d_{i,j}(\hat{i},\hat{j}) = 100 \cdot (\text{height}(\hat{i},\hat{j}) - \text{height}(i,j))_+ + 5.$$

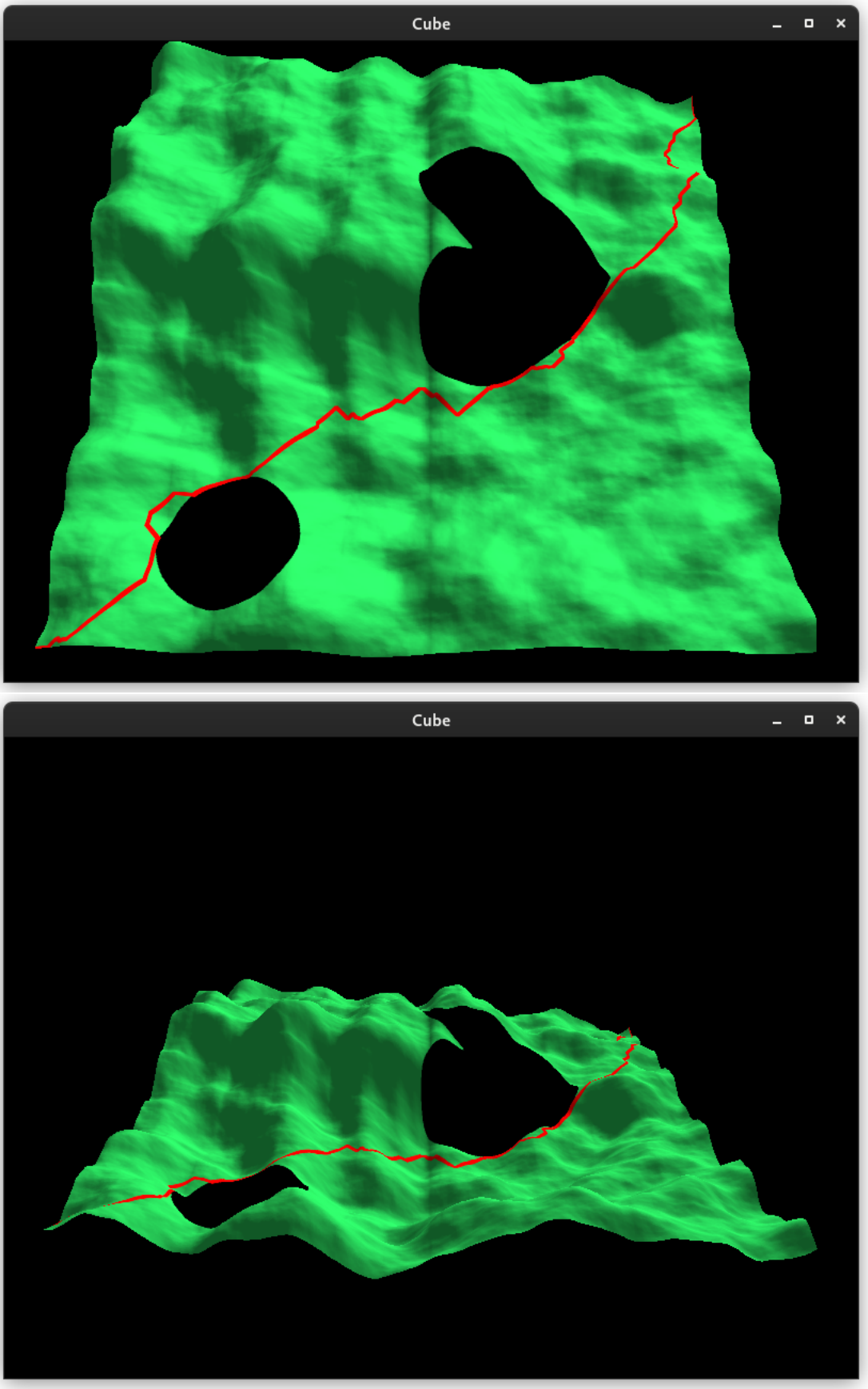

Рис. 1: Результат работы алгоритма Дейкстра на сетке 1000×1000 с начальной позицией $(1, 1)$, конечной $(1000, 1000)$.

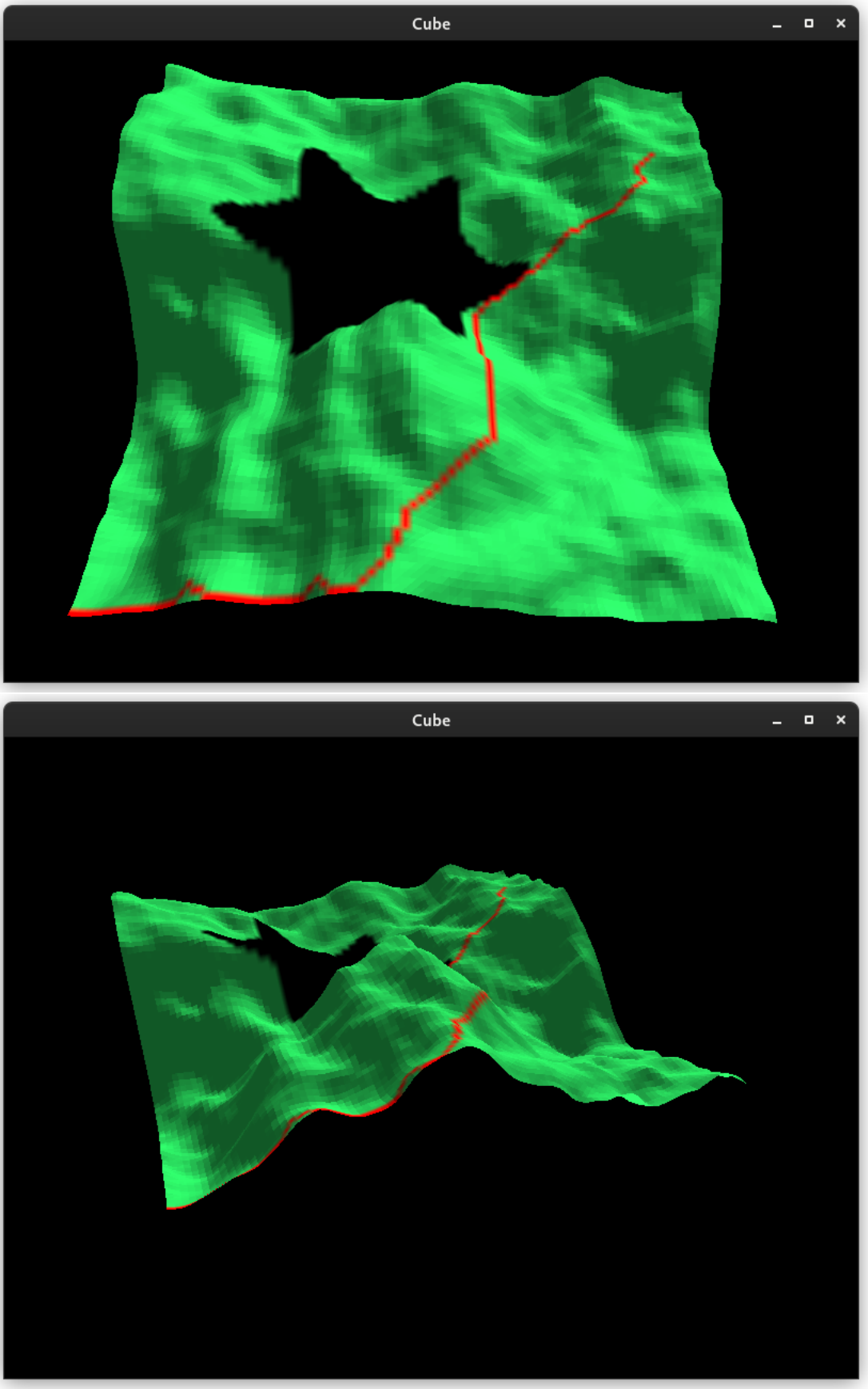

Рис. 2: Результат работы алгоритма Беллмана–Форда на сетке 100×100 с начальной позицией $(1, 1)$, конечной $(90, 90)$.

6 Сравнение алгоритмов

Ниже приведены времена работы различных версий программы в виде таблиц. Интересно прослеживается закономерность нелинейного ускорения работы программы для алгоритма Дейкстры в связи с небольшим граничным множеством относительно размера сетки. Так же примерно виден процент времени передачи данных на многонодной установке, относительно времени вычислений.

Приложения запускались на ноутбуке автора с 8 ядерным процессором Intel® Core™ i7-6700HQ и 8 ГиБ ОЗУ. Многонодная установка состояла из 3 виртуальных машин с 4 ядрами CPU и 8 ГиБ ОЗУ, две из которых использовались для вычислителей. Ноды были взяты у сервиса VK Cloud, который не предоставляет информацию об их физическом расположении и сетевой отдаленности.

Размер сетки	Классический	Парал. однонодный	Парал. многонодный
50×50	2s	700ms	2s
100×100	28s	11s	24s
250×250	14m 17s	4m 56s	10m 24s
500×500	≈4h 50m	≈1h 20m	≈2h 40m

Рис. 3: Время работы различных реализаций алгоритма Беллмана–Форда.

Размер сетки	Классический	Парал. однонодный	Парал. многонодный
500×500	2.9	4.5s	19.4s
1000×1000	36s	40s	3m 20s
2500×2500	10m 40s	6m 48s	34m 42s
5000×5000	≈1h 15m	≈39m	≈2h 40m

Рис. 4: Время работы различных реализаций алгоритма Дейкстры.

Из приведенных выше данных следует, что при небольшом объеме вычислений, параллельные алгоритмы показывают худшие результаты, чем их последовательные аналоги. Это связано с тем, что на таких объемах значительную роль играет время на синхронизацию потоков, а в случае многонодной установки еще и время, затрачиваемое на передачу информации по

сети.

Затем, при увеличении размеров сетки, параллельные алгоритмы становятся более эффективны. Алгоритм, большая часть которого распараллелена, показывает большее отставание от однопоточного. При этом видно, что увеличением вычислительных мощностей нельзя догнать произвоительность алгоритма, с априори меньшей алгоритмической сложностью. Хотя многонодные алгоритмы не оправдали, возложенных ни них надежд.

7 Заключение

В рамках задачи на основе известных алгоритмов поиска кратчайшего пути в ориентированных графах были построены параллельные алгоритмы для решения задачи быстродействия при наличии фазовых ограничений. Предложеные алгоритмы охватывают все степени параллелелизма, возможные на обычном компьюере: на ядрах центрального процессора, на графическом процессоре, на многоноднодной установке.

Также была выбрана архитектура и написана реализация параллельных алгоритмов Беллмана–Форда и Дейкстры на ядрах центрального процессора, и для многонодной установки. Были проведены замеры времени работы различных алгоритмов на одинаковых начальных данных и схожем оборудовании для сравнения. Были описаны причины, приводящие к получившимся результатам.

Список литературы

[1] Беллман Р., Дрейфус С. *Прикладные задачи динамического программирования.* М.: Наука, 1965.

[2] Shu-Xi, Wang. *The Improved Dijkstra's Shortest Path Algorithm and Its Application.* Procedia Engineering. 29. 1186-1190. 2012.

[3] Glabowski, M., Musznicki, B., Nowak, P., Zwierzykowski, P. *Review and Performance Analysis of Shortest Path Problem Solving Algorithms.* International Journal On Advances in Software. 7. 20-30. 2014.

[4] Корниенко В. С. *Численные методы решения задач "среднего поля"*:дис. к.ф.-м.н.: 05.13.18 — Сибирский федеральный университет, Красноярск, 2021.

[5] Ведякова А. О., Милованович Е. В., Слита О. В., Тертычный-Даури В. Ю. *Методы теории оптимальноого управления.* М.: Редакционно-издательский отдел Университета ИТМО. Санкт-Петербург, 2021.

[6] John Brosz, Faramarz F. Samavati and Mario Costa Sousa. *Terrain Synthesis By-Example, Advances in Computer Graphics and Computer Vision.* Communications in Computer and Information Science 4, 2007, 58–77