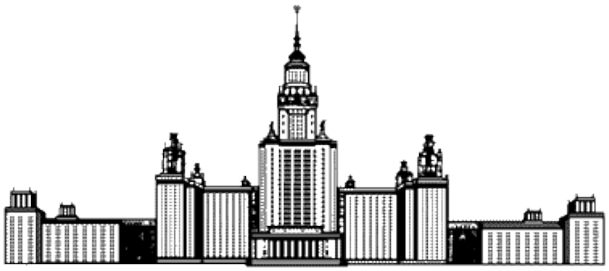


Московский государственный университет имени М.В.Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра суперкомпьютеров и квантовой информатики

Волков Никита Игоревич

Методы визуализации информационных графов прикладных алгоритмов

Научный руководитель:
вед.н.с., к.ф.-м.н.
Антонов Александр Сергеевич

Москва, 2015

Аннотация

Основная цель данной работы - создание методов визуализации графа алгоритма, имеющего описание в произвольной форме, в некоторой разработанной в рамках этой же работы стандартной форме. Решение указанной задачи подразумевает мультиэтапный процесс, который включает в себя :

1) Определение критериев эффективности визуализации информационного графа алгоритма.

2) Разработку стандартной формы представления графа алгоритма, соотвествующей определенным критериям эффективности.

3) Разработку методов приведения произвольным образом описанного графа алгоритма к полученной стандартной форме.

4) Применение разработанных методов к различным графам алгоритмов в качестве элемента полного описания этих алгоритмов.

Введение

Под информационным графом алгоритма понимается [1] связный ациклический граф, вершины которого соответствуют операциям алгоритма, а дуги - связям по данным между этими операциями. При этом в классическом определении графа алгоритма вершины соответствуют базовым вычислительным операциям. Граф алгоритма может быть представлен как в произвольном виде, так и в некоторой строго определенной форме. При этом наиболее часто используется ярусно-параллельная форма [1] , в которой на одном уровне (ярусе) располагаются все вершины-операции, которые могут быть выполнены параллельно на одном этапе работы алгоритма и требующие одинакового количества квантов времени для выполнения всех предшествующих им операций (операций, выполнение которых необходимо для формирования входных данных). Структура и сложность графа алгоритма полностью определяются алгоритмом, по которому строится граф - поэтому в общем случае граф алгоритма может иметь сколь угодно много вершин и сколь угодно много измерений, а отношение числа дуг к числу вершин ограничено таковым у полного графа с соответствующим числом вершин. Как известно, любой граф можно расположить в трёхмерном пространстве таким образом, что не будет пересечений между ребрами графа, поэтому следует пояснить следующее : под числом измерений графа алгоритма подразумевается число его логических измерений, то есть максимальная глубина вложенных циклов / рекурсии в некоторой референтной реализации этого алгоритма. Под референтной реализацией понимается реализация без каких-либо изменений алгоритма с целью его оптимизации либо использования ресурса параллелизма алгоритма (последовательная реализация).

Задача изображения графа алгоритма как элемента описания алгоритма - во-первых, обеспечить наглядное представление внутренней структуры алгоритма, в том числе и для лиц, не являющихся экспертами либо вообще не имеющими отношения к области, в которой применяется описываемый алгоритм, во-вторых, облегчить процесс подробного анализа алгоритма с целью составления его полного описания с упоминанием всех характерных особенностей алгоритма как на уровне собственно алгоритма, так и на уровне его конкретных реализаций. Для достижения первой цели визуализация графа алгоритма должна содержать вспомогательную информацию и быть структурирована таким образом, чтобы обеспечить максимальную ясность как принадлежности отдельных дуг отдельным вершинам, так и общей логической структуры алгоритма, а также компенсировать возможную сложность внутренней структуры алгоритма собственной простотой восприятия. Для реализации второй цели визуализация информационного графа алгоритма должна иметь четкую взаимосвязь со схемой работы реализации этого алгоритма и с исходным кодом реализации.

Постановка задачи

Как уже отмечалось ранее, граф алгоритма может быть визуально представлен в какой угодно форме, при этом наиболее часто используемой формой графа алгоритма является ярусно-параллельная форма, а иные формы представления графа алгоритма собственных названий не имеют. Одной из целей данной работы является создание системы рекомендаций по построению визуального представления графа алгоритма, которое может использоваться как самостоятельно, так и в качестве элемента более полного описания алгоритма. Такое представление должно удовлетворять следующим требованиям :

1) Иметь общий узнаваемый стиль выполнения, не зависящий от структуры конкретных алгоритмов. Это необходимо для удобного восприятия визуализаций графов различных алгоритмов после однократного ознакомления с принципами их построения.

2) Иметь собственную характеристику в виде чётко предъявляемых к описанию требований и методов описания, сгруппированных и вынесенных в специально сформированный стандарт.

3) Отображать логическую структуру алгоритма, на основе которого строится визуализация. В это понятие входит размещение вершин по уровням, как в ярусно-параллельной форме представления графа алгоритма, размещение вершин по плоскостям, соответствующим уровням вложенности циклов в референтной реализации алгоритма, взаимодействие по данным между операциями алгоритма.

4) Быть достаточно простым для восприятия лицами, не являющимися экспертами в области анализа структуры алгоритмов. Содержать вспомогательную информацию, упрощающую понимание графов алгоритмов со сложной внутренней структурой. В число усложняющих граф свойств входят : логическая многомерность графа алгоритма, большое число дуг в графе алгоритма по сравнению с числом вершин, высокая сложность связей по данным в графе алгоритма.

5) Иметь взаимосвязь с иными формами описания алгоритма - например, исходным кодом референтной реализации или описанием на каком-то специально предназначенном для этого языке.

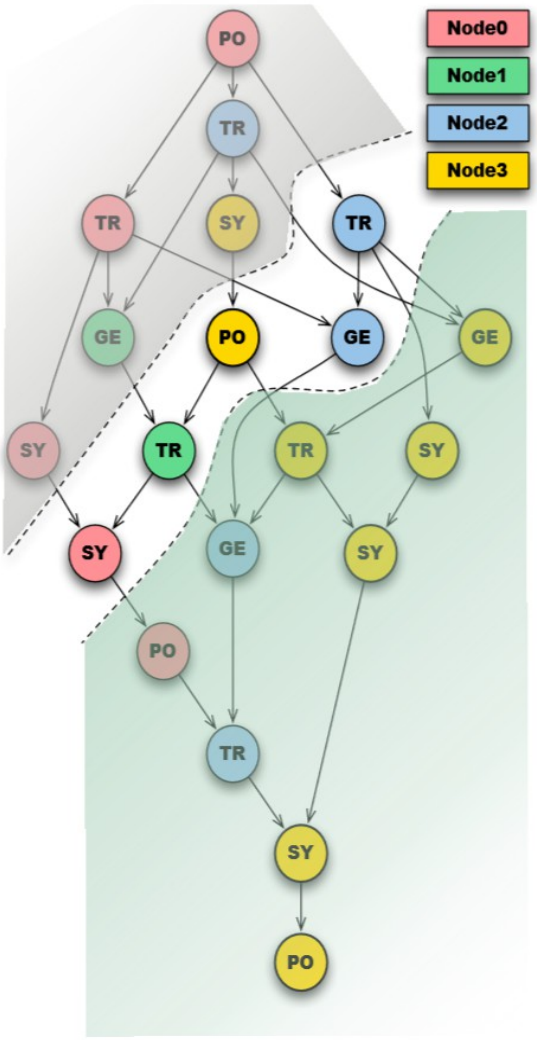
После разработки такого стандартизованного представления ключевой задачей становится разработка методов, позволяющих представить произвольный информационный граф в этой универсальной форме. Очевидно, что любой имеющийся информационный граф уже имеет описание в некоторой форме, так что эта задача сводится к преобразованию произвольного описания графа алгоритма в стандартизованное описание. Например, граф алгоритма может уже иметь визуализацию в некоторой форме, может быть описан на некотором специально

предназначенном для этого языке либо неявно представлен в виде программного кода, может быть неявно представлен в математическом описании алгоритма, а может и существовать только в виде мысленного образа некоего эксперта в соответствующей области. При этом в процессе перевода графа алгоритма в стандартизованную форму возможно использование некоторых промежуточных представлений.

Обзор методов.

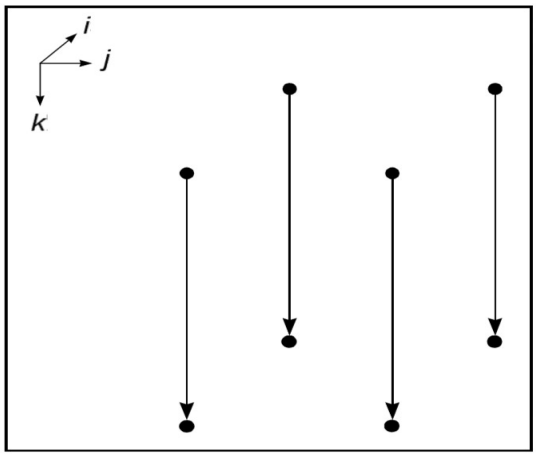
В настоящее время графы алгоритмов не находят широкого применения в качестве элемента описания прикладных алгоритмов. Однако графы алгоритмов и схожие с ними понятия используются в смежных областях, в частности, в качестве описания поведения реализации алгоритмов при их исполнении, построения графов потока управления и потока данных для некоторых программных продуктов, а также в учебных целях. Приведем несколько примеров вышеуказанного использования графов алгоритмов и схожих понятий.

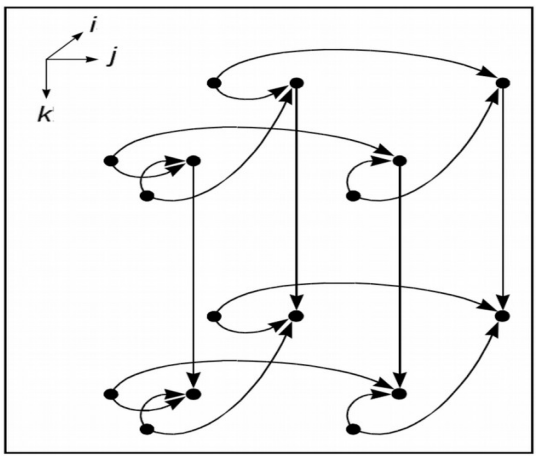
Понятие прямого ациклического графа (DAG) [6] используется для исследования взаимодействия отдельных вычислительных блоков программы. При этом DAG программной реализации алгоритма строится отдельно для каждого запуска программы, причём DAG одной и той же программы может отличаться для разных запусков.
Изображенный справа DAG показывает распределение вычислительных операций по узлам системы, на которой была запущена соответствующая реализация некоторого алгоритма. При этом на нём выделена группа элементов, находящая на одном ярусе в ярусно-параллельной форме этого DAG. Главное отличие этой структуры от графа алгоритма — её машинная зависимость.



Кроме того, понятие графа алгоритма часто используется в учебных целях. В качестве примера можно привести [1]. Отличие такого использования графа алгоритма от нашей задачи состоит в том, что учебные графы алгоритмов строятся прежде всего визуализировать само определение графа алгоритма на каком-то простом примере, а не обеспечить эффективную визуализацию произвольного графа алгоритма.

Ниже вы можете увидеть пример таких визуализаций графа алгоритма.





Исследование и построение решения.

Любой алгоритм обладает некоторым набором свойств, часть из которых находит своё отражение в графе этого алгоритма. Некоторые свойства, такие как : отношение числа зависимостей по данным к числу вычислительных блоков в алгоритме и максимальная размерность гнезд циклов в алгоритме приводят к существенному усложнению структуры алгоритма и, соответственно, визуального восприятия его графа. Поставленные задачи будем выполнять последовательно. Предположим, что у нас уже имеется полная информация о графе алгоритма, а для отработки принципов отображения графов алгоритмов и приведения стандарта их визуализации будем вначале использовать алгоритмы с максимально простой структурой, логические особенности которых видны при любом сколько-нибудь разумном способе визуализации и которые не содержат усложняющих восприятие элементов. После отработки методов визуализации на этих простейших алгоритмах перейдём к более широкому классу описываемых алгоритмов, в который попадает большинство алгоритмов, реализующих применяемые на практике вычислительные методы. Отметим, что для этой процедуры надо сначала определить такой класс алгоритмов. Затем приведем полную версию стандарта визуализации, позволяющую организовать представление графа алгоритма в виде, удовлетворяющем четырём перечисленным требованиям. Следующим этапом является установление взаимосвязи визуального представления графа алгоритма с иными формами описания алгоритма, а также инструментарий анализа полученной визуализации и ее интерактивного представления. Наконец, конечной задачей является определение собственно графа алгоритма, который, как отмечено, необходим для выполнения всех вышеперечисленных задач, по некоторому произвольному описанию алгоритма.
Опишем подробнее сложности, возникающие при построении визуализации графа алгоритма, и будем ссылаться на эти особенности структуры алгоритма при их дальнейшем упоминании.

- **Большое количество информационных зависимостей (зависимостей по данным) в алгоритме , а значит , большое количество дуг в графе алгоритма.** В качестве примеров рассмотрим алгоритм суммирования массива длины n сдваиванием и алгоритм блочного

перемножения матриц (не рекурсивный вариант с 6-ю циклами). Для обоих алгоритмов подсчитаем коэффициент S = E/V , то есть число ребер графа алгоритма , деленных на число его вершин , причём сделаем это для произвольного n .В алгоритме сдваивания для массива длины n число вершин есть $V=\sum_{i=1}^{i=\log 2n}\frac{n}{2^i}\simeq 2*n$: Число ребер равно $E=\sum_{i=1}^{i=\log 2n}\frac{n}{2^i}\simeq 2*n$, то есть $S=E/V\simeq 1, n\to\infty$, что является очень хорошим результатом. Если же мы рассмотрим алгоритм блочного умножения матриц , то увидим , что это отношение намного хуже даже на уровне пересылок между различными блоками , а именно : $V=n^3 \quad E=n^2(n-1)(2n-1)$, то есть получаем $S=E/V\simeq 2n, n\to\infty$, то есть граф алгоритма при изображении будет содержать большое число дуг , затрудняющих понимание его структуры. Для лучшего понимания проблемы приведем граф блочного умножения матриц , в котором вершины — операции вычисления всего блока , а не отдельного элемента матрицы , то есть в классическом графе алгоритма ребер было бы еще больше.

- **Большая логическая размерность графа алгоритма** . Как известно , любой граф можно изобразить в трёхмерном пространстве таким образом , чтобы его рёбра не пересекались [3]. Однако , во-первых , изображения графов двумерный , а во-вторых , и трёхмерная модель графа алгоритма не способна в полной мере отразить структуру графов с размерностью > 3. К примеру , уже рассмотренное нами блочное умножение матриц имеет размерность 6. Наконец , крупные алгоритмы , состоящие из большого числа отдельных компонент , такие как Lin pack Benchmark и HPCG , имеют слишком сложную структуру для того , чтобы среднестатистический пользователь мог изучать их как единое целое. В качестве примера достаточно вспомнить , что в вышеизложенном графе каждая вершина в действительности является блоком , структуру которого можно увидеть на изображении справа.

Еще одну сложность, связанную не с логической структурой отображаемого алгоритма, а с понятием непосредственно графа алгоритма представляет **оторванность классического графа алгоритма от входных и выходных данных.** Входные и выходные данные в графе алгоритма не обозначаются отдельными вершинами , так как не являются вычислительными операциями. В то же время , во многих алгоритмах взаимосвязь входных данных и непосредственно использующих эти данные вычислительных операций нетривиальна . Обратимся к методу Холецкого для LU-разложения матрицы. Какие элементы графа принимают элементы исходной матрицы в качестве входных данных , результатом каких операций являются элементы двух матриц , полученных после декомпозиции ? Граф алгоритма в классическом виде не даёт ответов на эти вопросы.

Построение визуализации ГА для простейших алгоритмов.

Как было отмечено выше, в рамках этого пункта не нужно искать методы борьбы с перечисленными особенностями логической структуры графа алгоритма в предположении ее простоты. Поэтому сфокусируемся на основных принципах визуализации графа алгоритма, не запрещая возможное впоследствии добавление в неё вспомогательной информации. Иначе

говоря, определим, что должна содержать визуализация алгоритма независимо от того, какой конкретно граф визуализируется.

Базовая визуализация графа алгоритма.

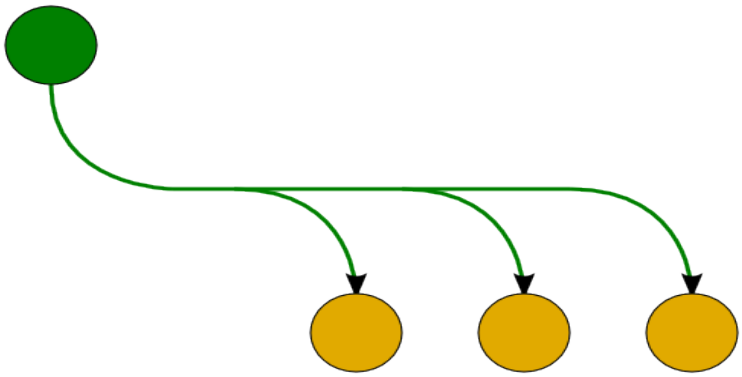
- Визуализация графа алгоритма должна состоять как минимум из одного изображения , содержащего **надграф** графа алгоритма , представляющий собой классический граф алгоритма , дополненный вершинами , соответствующими входным данным. *В этом стандарте графом алгоритма будем именовать именно этот надграф.*
- Граф алгоритма на этом изображении должен быть представлен для частного случая задачи с конечным объёмом входных данных , минимальным для дачи полного представления о внутренней структуре алгоритма и его характерных особенностях , а также ресурсе параллелизма.
- Граф алгоритма должен быть дополнен вершинами и дугами , отображающими схему поступления данных на вход алгоритма и схему вывода результатов.
- Изображение не должно содержать никаких других графов и расширений исходного графа алгоритма , за исключением описанных в предыдущем пункте.
- Изображение может содержать пояснения , дающие дополнительную информацию о структуре графа , как то : оси декартовой системы координат , нумерацию входных данных , индексацию вершин по ярусам , пунктирные линии для обозначения границ схемы алгоритма и.т.п.

После того, как определено содержание визуализации для любого графа алгоритма, необходимо предложить и описать стандартную форму представления основных элементов графа алгоритма. При этом будем иметь в виду, что те же методы будут применяться для отображения более сложных алгоритмов, поэтому уже сейчас необходимо каким-то образом разделять операции и пересылки данных разного рода. Поэтому определим

Структурная схема визуализации графа алгоритма.

- Вершины графа алгоритма , соответствующие вычислительным операциям , обозначаются кругами , размер которых совпадает для всех операций , за исключением вершин , соответствующих макроблокам с большей вложенностью по базовым операциям в сравнении с остальными операциями , встречающихся в этом изображении. Такие макроблоки обозначаются более крупно.
- Вершины графа алгоритма , соответствующие входным данным , необходимо обозначать квадратами , размер которых совпадает для всех элементов входных данных.
- Каждая вершина графа должна быть проиндексирована некоторым кодовым словом , обозначающим конкретную операцию либо элемент входных данных. В случае наличия в визуализации графа алгоритма дополнительных изображений , разъясняющих структуру операций , расположенных в вершинах графа , кодовое слово должно присутствовать на этих изображениях.

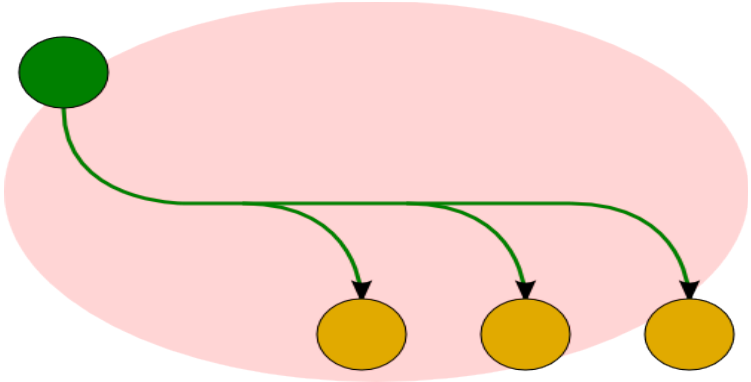
- Дуги графа обозначаются линиями со стрелками на концах , соответствующих «адресату» данных. Возможно использование одной линии для изображения рассылки данных от одной вершины нескольким. Использование той же техники для изображения пересылки результатов нескольких операций одной также допускается , но не допускается сочетание этих методов в визуализации одной пересылки данных.



Кроме того, в качестве дополнительного средства передачи информации о внутренней структуре графа алгоритма определим

Цветовую схему визуализации графа алгоритма.

- Любые вершины , соответствующие операциям одного вида , необходимо обозначать одним цветом , вне зависимости от входных данных для этих вершин. Любые вершины , соответствующие операциям различного вида , необходимо обозначать разным цветом опять же вне зависимости от их входных данных.
- Цвета должны быть полупрозрачными и обеспечивать возможность изображения проекции графа алгоритма в несколько слоёв.
- Цвет любой дуги графа должен быть идентичен цвету вершины , из которой выходит эта дуга.
- На изображениях , поясняющих структуру конкретных операций в графе алгоритма должен присутствовать фон в виде круга , цвет которого соответствует цвету этой операции в базовом изображении графа алгоритма.



Наконец, необходимо отработать эти базовые схемы для визуализации алгоритмов на простейших графах алгоритмов. Определим логическую размерность графа алгоритма через максимальную размерность гнезд циклов в референтной реализации этого алгоритма. При этом мы будем говорить о базовых вычислительных операциях, то есть тех, выполнение которых не подразумевает выполнения некоторого цикла или имеет сложность O(1). Размерность гнезд циклов также определяется по базовым вычислительным операциям. Именно поэтому в примере выше алгоритм блочного перемножения матриц имеет размерность 6. В качестве простейших графов алгоритмов будем использовать алгоритмы, имеющие логическую размерность не более 2 и отношение числа дуг к числу вершин не более O(1). Под эти условия подходят, к примеру, алгоритмы из класса операций вектор-вектор и матрица-вектор. Примеры визуализации графов таких алгоритмов приведены в практической части.

Методы визуализации трёхмерных графов алгоритмов.

В этой части мы покажем, что ограничение на максимальную логическую размерность алгоритма, равную трём, допускает очень широкий класс алгоритмов, и практически все применяемые на практике алгоритмы либо принадлежат этому классу, либо могут быть разбиты на макроблоки - подалгоритмы, принадлежащие этому классу, при этом логическая размерность самих алгоритмов не превосходит 3 при учёте выполнения макроблока как базовой операции. Наконец, процедура разбития алгоритма на макроблоки может выполняться рекурсивно таким образом, чтобы на каждом этапе логическая размерность алгоритма не превосходила 3 при учете макроблока следующего этапа разбиения алгоритма на подзадачи как базовой вычислительной операции.

Для начала предложим несколько примеров вычислительных алгоритмов, обладающих именно этой сложностью [4] :

•Метод Холецкого для LU-разложения матрицы.

Для разложения матрицы порядка n методом Холецкого в последовательном (наиболее быстром) варианте требуется:

n вычислений квадратного корня,
n(n – 1)/2 делений
(n^3 – n)/6 сложений (вычитаний)
(n^3 – n)/6 умножений

[4] Умножения и сложения (вычитания) составляют основную часть алгоритма. При этом использование режима накопления требует совершения умножений и

вычитаний в режиме двойной точности , что ещё больше увеличивает долю умножений и сложений/вычитаний во времени, требуемом для выполнения метода Холецкого. Таким образом, метод Холецкого в последовательной реализации относится к алгоритмам с кубической сложностью.

•Метод Гивенса (вращений) для QR-разложения матрицы.

[4] В последовательной версии основная сложность алгоритма определяется прежде всего массовыми операциями вращения. Они, если не учитывать возможную разрежённость, составляют (в главной составляющей) n^3/3 операций комплексного умножения или, если не прибегать к ухищрениям, 4n^3/3 вещественных умножений и 2n^3/3 вещественных сложений/вычитаний.
При классификации по последовательной сложности, таким образом, метод Гивенса относится к алгоритмам с кубической сложностью.

•Перемножение плотных матриц.

[4] Для умножения двух квадратных матриц порядка n (т.е. при m=n=l) в последовательном (наиболее быстром) варианте требуется по n^3 умножений и сложений.
Для умножения матрицы размером m строк на n столбцов на матрицу размером m строк на n столбцов в последовательном (наиболее быстром) варианте требуется по m*n*l умножений и сложений.

При этом использование режима накопления требует совершения умножений и сложений в режиме двойной точности, что ещё больше увеличивает затраты во времени, требуемом для выполнения умножения матрицы на вектор.

При классификации по последовательной сложности, таким образом, алгоритм умножения матриц относится к алгоритмам с кубической сложностью (в случае не квадратных матриц - с трехлинейной).

•Метод Гаусса решения СЛАУ (прямой и обратный ход).

[4] Для прямой подстановки порядка n в последовательном (наиболее быстром) варианте требуется:

по (n^2 – n)/2 умножений и сложений (вычитаний).
При этом использование режима накопления требует совершения умножений и вычитаний в режиме двойной точности (или использования функции вроде DPROD в

Фортране), что ещё больше увеличивает затраты во времени, требуемом для выполнения прямой подстановки.

При классификации по последовательной сложности, таким образом, метод обратной подстановки относится к алгоритмам с квадратичной сложностью.

- **Сдвиг аргументов многочлена по схеме Горнера.**

[4] Для вычисления сдвига аргументов многочлена по схеме Горнера для многочлена степени n, количество операций умножения равно количеству операций сложения и равно n(n+1)/2. Поэтому алгоритм должен быть отнесён к алгоритмам квадратичной сложности по количеству последовательных операций.

Таким образом, существуют вычислительные алгоритмы сложности не выше указанной для решения некоторого ряда классических вычислительных задач. Существование таких алгоритмов делает нецелесообразным использование для решения соответствующих задач алгоритмов с большей сложностью либо большим порядком сложности. Заметим также, что для решения вычислительных задач, в случае наличия такой возможности, используются алгоритмы с полиномиальной сложностью. Поэтому наиболее широкое ограничение для алгоритмов с не более чем трёхмерной логической структуры установим как полиномиальную вычислительную сложность. Именно кубическая сложность является верхним порогом для ряда несложных алгоритмов, включающих в себя операции над матрицами, и такие алгоритмы сами по себе имеют трёхмерную логическую структуру. Если же сложность алгоритма превосходит кубическую, то всегда можно провести описанную выше процедуру разбиения алгоритма на макроблоки, то есть свести алгоритм к вложенным подалгоритмом, каждый из которых имеет не более чем кубическую сложность.

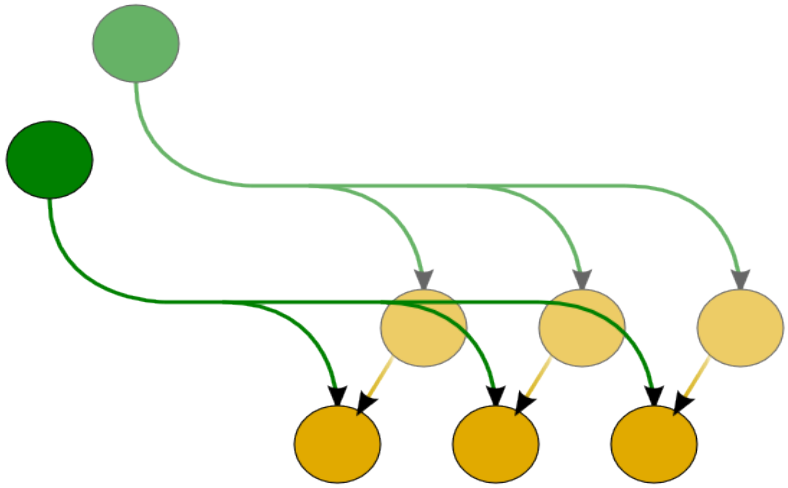
Визуализация графа алгоритма с трёхмерной логической структурой, очевидно, предполагает размещение вершин и дуг алгоритма в трёхмерном пространстве (в качестве визуализации предлагается либо трёхмерная модель, либо проекция). Для максимально упорядоченного представления вершин графа алгоритма воспользуемся следующим : так как любой граф укладывается в трёхмерном пространстве , то все вершины графа возможно разбить на конечное число групп и расположить вершины таким образом , что каждая группа будет полностью расположена в одной из нескольких параллельных плоскостей. Далее каждую плоскость можно рассматривать как граф простейшего, согласно нашему определению, алгоритма, и изобразить её в соответствии с базовым стандартом, после чего отдельно отобразить дуги , связанные с разными плоскостями. Кроме того , необходимо ввести дополнительные пояснения в изображение графа алгоритма , т. к. трёхмерная структура сложнее для восприятия. Дополним структурную и цветовую схемы графа алгоритма для случая алгоритмов с трёхмерной логической структурой.

Дополнительная структурная схема для трёхмерных алгоритмов.

- На любом изображении , относящимся к визуализации графа размерности 3 и более , должны быть изображены оси декартовой системы координат, при этом параллельные плоскости, на которых размещаются вершины графа алгоритма, параллельны также и одной и плоскостей декартовой системы координат. Выбор этой параллельной плоскости может зависеть от конкретного алгоритма, но при отсутствии явных преимуществ отображения алгоритма, даваемых таким выбором, в качестве плоскости "по умолчанию" используется oXY.
- На изображении графов с трёхмерной структурой должны присутствовать пунктирные линии , обозначающие границы визуализации графа алгоритма, при этом каждая такая линия должна обозначать границу области для размещения вершин графа алгоритма в той параллельной плоскости, где располагается указанная линия. Примеры такой вспомогательной информации будут приведены в практической части.

Дополнительная цветовая схема для трёхмерных алгоритмов.

- Вся трёхмерная модель графа алгоритма укладывается в набор параллельных плоскостей , параллельных плоскости Oxz в введенной декартовой системе координат , причём расстояние между ближайшей к Oxz плоскости и Oxz равно расстоянию между любыми двумя соседними плоскостями. Каждая плоскость отображается в соответствии с базовым стандартом.
- Все вершины и дуги , относящиеся к одной плоскости , имеют одинаковую степень прозрачности цвета. Вершины и дуги , находящиеся в разных плоскостях , имеют разную степень прозрачности , при этом наименее прозрачными являются вершины с наименьшим значением ординаты.
- Все дуги , соединяющие различные параллельные плоскости , имеют градиентную окраску и прозрачность цвета на их концах соответствует уровню прозрачности плоскостей с вершинами , которым инцидентна данная дуга. Координатная ось Oy при этом изображается на длину до последней из параллельных плоскостей и также попадает под это правило.



Поскольку рассматривается максимально широкий класс алгоритмов, которые могут быть представлены в виде с трёхмерной логической структурой, то не подразумевается никаких особенностей графа алгоритма, упрощающих его устройство, что означает потенциальное наличие у графа алгоритма всех перечисленных качеств, усложняющих его визуальное восприятие. Это означает, что необходимо модифицировать визуализацию графа алгоритма таким образом, чтобы нивелировать воздействие этих присущих алгоритму свойств. Последовательно решим эти проблемы, введя соответствующие правила для расширения визуализации ГА.

Большое количество дуг в графе алгоритма.

Основная возникающая при этом визуальная проблема — дуги перекрывают друг друга и определение того , каким вершинам инцидентна конкретная дуга , становится затруднительным. В связи с этим предлагается следующее решение :

- Изображается проекция графа на плоскость Oxz в предположении , что вершины на разных параллельных плоскостях , соответствующие характерным блокам операций , имеют одинаковые значения ординат и аппликат.
- Вершины графа алгоритма в трёхмерной модели располагаются так , чтобы сохранилось расположение относительно Oxz и при этом образовался аналогичный набор параллельных плоскостей относительно плоскости Oyz. После чего изображается аналогичная проекция на плоскость Oyz.
- В графе алгоритма выделяются характерные по своей структуре блоки , которые изображаются отдельно с индексацией вершин , позволяющей определить , как именно этот блок расположен в графе алгоритма.
- Все полученные дополнительные изображения включаются в визуализацию алгоритма.

Многомерность графа алгоритма.

Более чем трёхмерные структуры сложно воспринимать именно как структуры в n-мерном пространстве. Поэтому многомерная структура конкретного алгоритма воспринимается как трёхмерная , что ведёт к непониманию того , как именно работает приведенный алгоритм. Эту проблему будем решать следующим образом :

- Многомерные структуры удобно воспринимать в качестве некой иерархии , где любая n – k -мерная гиперплоскость в исходном n-мерном пространстве представляется как единый макроблок , то есть n-мерная структура сводится к k-мерной структуре из макроблоков , где k <= 3. Далее каждый макроблок , представляющий собой n-k мерное пространство , аналогично разбивается на гиперплоскости. Процесс продолжается , пока n-k > 3. k на каждом шаге выбирается равным 1-3 в зависимости от структуры собственно алгоритма.
- В визуализации алгоритма не используется изображение графа алгоритма в полном виде. Вместо этого исходный граф алгоритма представляется только в виде типовых макроблоков в качестве вершин графа алгоритма. Каждый макроблок опять же изображается в виде графа алгоритма в соответствии с построенной на предыдущем этапе иерархической структурой , пока не будет достигнут уровень базовых вычислительных операций. Таким образом , получается набор изображений. Кроме того , на каждом этапе решается проблема большого числа дуг в графе алгоритма по методу , описанному пунктом выше.

Оторванность графа алгоритма от входных и выходных данных.

Эта проблема решается проще остальных. Как уже было показано , вместо графа алгоритма строится его надграф , полученный добавлением к исходному графу алгоритма вершин , соответствующих входных и выходным данным. Эти вершины не используются на этапе макроблоков , а участвуют только в тех изображениях , где инцидентные этим вершинам дуги входят в вершины , соответствующие базовым операциям. На более высоких иерархических уровнях такие вершины включаются в макроблоки. В случае размерности 3 и более базовое изображение вершин , соответствующих входам и выходам , не содержит. Вместо этого используется дополнительное изображение , содержащее искомый надграф.

Стандартизация схемы визуализации ГА.

Граф алгоритма, отображенный в виде, соответствующем сформулированным в предыдущих пунктах правилам, имеет вид, упрощающий восприятие его логической структуры. Однако граф алгоритма должен ещё и "иметь общий архетип и стиль выполнения, не зависящий от структуры конкретных алгоритмов". При этом стандарт визуализации не должен быть слишком строгим, а именно не должен включать в себя требования к использованию для визуализации графа алгоритма строго определенного инструментария и строго определенных мелких элементов визуального стиля. Помимо этого, полезно предоставить некоторые образцы визуализации

графов алгоритмов, на которые можно ориентироваться, а также некоторый базовый набор шаблонов, используемых при визуализации алгоритма. Поэтому определим базовый стандарт визуализации алгоритма, содержащий в себе выдержки из описанных нами правил и дополненный некоторыми рекомендациями по формату создаваемых визуализаций графов алгоритмов.

1 Необходимый минимум изображений и описаний

Визуализация алгоритма должна состоять как минимум из одного изображения , содержащего граф алгоритма. Граф алгоритма на этом изображении должен быть представлен в полном виде для частного случая задачи (с конечным , небольшим числом входных данных) , дающем , тем не менее , полное представление о структуре алгоритма и его характерных особенностях. Кроме того , граф алгоритма должен быть дополнен вершинами и дугами , отображающими схему поступления данных на вход алгоритма и схему вывода результатов.
Изображение не должно содержать никаких других графов и расширений исходного графа алгоритма , противоречащих его определению , за исключением описанных в пункте "в". Однако изображение может содержать краткие пояснения , дающие дополнительную информацию о структуре графа. Например , информацию о объеме входных данных.

2 Особенности схематичного изображения ГА

2.1 Структурная схема визуализации ГА

В качестве инструмента визуализации используется любой редактор векторной графики (формат .svg) либо средства, позволяющие получить схожее изображение. Финальные версии построенных схем визуализации конвертируются в изображение любого удобного формата.
Вершины графа обозначаются кругами , размер которых должен совпадать для всех однотипных операций. Требования к относительному размеру вершин для разных операций жестко не декларируются , но рекомендуется обозначать крупнее операции внешних циклов / процедур.
Каждая вершина графа должна быть проиндексирована некоторым кодовым текстом , обозначающим конкретную операцию. В случае наличия дополнительных изображений с разъяснением структуры операций , кодовое слово должно присутствовать на этих изображениях.
Дуги графа обозначаются линиями со стрелками на концах , соответствующих "адресату" данных. Возможно использование одной линии с ответвлениями для изображения рассылки данных от одной вершины нескольким. В этом случае такая "магистраль" данных должна иметь на изображении большую толщину , нежели одиночные пересылки. Аналогично , допускается та же техника для изображения пересылки результатов нескольких операций для одной операции.

2.2 Цветовая схема визуализации ГА

Любые операции с одинаковой структурой необходимо обозначать одним цветом вне зависимости от входных данных. Операции с разной структурой необходимо обозначать разными цветами , опять же независимо от входных данных.

Цвета обозначения операций (вершин графа) выбираются пользователем , но рекомендуется использовать светлые тона. Кроме того , цвета должны быть в некоторой степени прозрачными и не полностью перекрывать обзор расположенных сзади слоёв в случае многослойного изображения.

В случае наличия дополнительных изображений , поясняющих структуру конкретных операций в графе алгоритма , задний фон этих изображений должен представлять собой круг того же цвета , каким обозначается операция в исходном графе алгоритма.

Цвет любой дуги графа должен быть идентичен цвету вершины , из которой выходит эта дуга.

В случае изображения графов с размерностью >= 3 набор плоскостей графа алгоритма может быть изображён использованием цветов разной насыщенности для разных плоскостей графа , при использовании одной и той же схемы насыщенности для всех вершин и дуг одного слоя.

Ярусно-параллельная форма алгоритма может изображаться по аналогичной методике. Одной схемой цветовой насыщенности изображаются все вершины и дуги , относящиеся к одному ярусу. Кроме того , возможна

визуализация ярусно-параллельной формы в виде набора изображений , на каждом из которых присутствует две схемы цветовой насыщенности - одна для выделенного на конкретном изображении яруса и вторая для всех остальных вершин и дуг графа алгоритма.

3 Методы визуализации сложных алгоритмов

3.1 Многомерные алгоритмы

Размерностью алгоритма в контексте этого пункта будем называть максимальную вложенность операций , встречающуюся в алгоритме.

Более чем трёхмерные структуры трудны для восприятия именно как структуры в n-мерном пространстве. Поэтому многомерная структура конкретного алгоритма воспринимается как трёхмерная , что ведёт к непониманию того , как именно работает приведенный алгоритм. Для решения проблемы предлагается следующий набор правил визуализации.

Многомерные структуры удобно воспринимать в качестве некой иерархии , где любая n – k -мерная гиперплоскость в исходном n-мерном пространстве представляется как единый макроблок , то есть n-мерная структура сводится к k-мерной структуре из макроблоков , где k <= 3. Далее каждый макроблок , представляющий собой n-k мерное пространство , аналогично разбивается на гиперплоскости. Процесс продолжается , пока n-k > 3. k на каждом шаге выбирается равным 1-3 в зависимости от структуры собственно алгоритма.
В визуализации алгоритма не используется изображение графа алгоритма в полном виде. Вместо этого исходный граф алгоритма представляется в виде типовых макроблоков в качестве вершин графа алгоритма. Каждый макроблок опять же изображается в виде графа алгоритма в соответствии с построенной на предыдущем этапе иерархической структурой , пока не будет достигнут уровень базовых вычислительных операций. Таким образом , получается набор изображений. Кроме того , на каждом этапе решается проблема большого числа дуг в графе алгоритма по методу , описанному пунктом выше.

3.2 Алгоритмы с нетривиальными входными данными

Алгоритмы, имеющие до такой степени сложную структуру входных данных, что ее отображение само по себе представляет собой граф алгоритма достаточно высокой сложности, отображаются в виде нескольких графов алгоритмов, на каждом из которых отображена только одна часть входных данных (по принципу однотипности этих данных). Таким образом, вместо одного надграфа ГА со сложной структурой входных данных получаем несколько надграфов с более простой.

3.3 Алгоритмы с множественными связями

Иначе выражаясь , большое количество дуг в графе алгоритма. Основная возникающая при этом визуальная проблема — дуги перекрывают друг друга и определение того , каким вершинам инцидентна конкретная дуга , становится затруднительным. Для решения этой проблемы используется следующая методика.

Изображается проекция графа на плоскость Oxz в предположении , что вершины на разных параллельных плоскостях , соответствующие характерным блокам операций , имеют одинаковые значения ординат и аппликат. Вершины графа алгоритма в трёхмерной модели располагаются так , чтобы сохранилось расположение относительно Oxz и при этом образовался аналогичный набор параллельных плоскостей относительно плоскости Oyz.
Изображается аналогичная проекция на плоскость Oyz.
В графе алгоритма выделяются характерные по своей структуре блоки , которые изображаются отдельно с индексацией вершин , позволяющей определить , как именно этот блок расположен в графе алгоритма.
Все полученные дополнительные изображения включаются в визуализацию алгоритма.

А также определим некоторый базовый набор элементов, которые можно шаблонизировать и которые так или иначе используются в рамках визуализации графа алгоритма. В число этих элементов входят :

- Базовая вершина графа алгоритма с произвольным кодовым словом, обозначающем некоторую выполняемую в ней операцию, определенная в нескольких шаблонных цветах.

- Базовая дуга графа алгоритма (обозначающая пересылку данных от одной вершины к одной вершине), также определенная в нескольких шаблонных цветах (далее это свойство подразумевается везде).
- Операция рассылки данных от одной вершины к нескольким.
- Операция сбора данных из нескольких вершин в одной.
- Операция, аналогичная MPI-функции all-to-all.
- Оси декартовой системы координат, используемой в визуализации трёхмерных алгоритмов (заметим, что и простейший алгоритм мы можем визуализировать как частный случай трёхмерного с всего одной параллельной плоскостью.)
- Некоторые наиболее распространенные кодовые слова, как то базовые арифметические операции и их комбинации, а также универсальное кодовое слово, обозначающее некоторую операцию (значение этого КС для конкретного ГА должно поясняться в легенде, приложенной к визуализации).

И, возможно, некоторые другие элементы. Этого набора элементов достаточно для визуализации несложных графов алгоритмов исключительно с его использованием, а для алгоритмов, где предъявленного шаблона недостаточно, возможно построение некоторого прообраза визуализации ГА на основе этого шаблона с дальнейшим приведением к конечному виду, что позволяет сохранить стиль визуализации.

Методы связи ГА с иными элементами описания алгоритма.

Визуализация графа алгоритма может использоваться как самостоятельно, так и в качестве части более полного описания алгоритма, начиная от его математической формулировки и заканчивая особенности параллельной реализации для конкретных архитектур. Поэтому важным элементом визуализации графа алгоритма является ее взаимосвязь с другими элементами описания алгоритма. Так как граф алгоритма в его классическом понимании строится независимо от его реализации, по его словесному / математическому описанию, то с этой формой описания он уже имеет чёткую взаимосвязь. Нас прежде всего будет интересовать элемент описания алгоритма, имеющий наиболее важное практическое значение, а именно исходный код некоторой референтной реализации этого алгоритма. Под словом "референсная" понимается реализация алгоритма на последовательной архитектуре, без каких-либо оптимизаций. Именно такие реализации включаются в описание алгоритма, потому что их цель - отображение структуры алгоритма в понятной форме, а не максимально быстрое решение соответствующей вычислительной задачи. В качестве примера такой референтной реализации можно привести последовательную реализацию метода Гивенса (вращений) для QR-разложения матрицы [2].

Основной цикл :

```
DO  I = 1, N-1
    DO J = I+1, N
        CALL PARAMS (A(I,I), A(J,I), C, S)
        DO K = I+1, N
            CALL ROT2D (C, S, A(I,K), A(J,K))
        END DO
    END DO
END DO
```

Собственно вращение :

```
SUBROUTINE ROT2D (C, S, X, Y)
ZZ = C*X - S*Y
Y = S*X + C*Y
X = ZZ
RETURN
END
```

Вычисление параметров вращения :

```
SUBROUTINE PARAMS (X, Y, C, S)
Z = MAX (ABS(X), ABS(Y))
IF (Z.EQ.0.) THEN
C OR (Z.LE.OMEGA) WHERE OMEGA - COMPUTER ZERO
    C = 1.
    S = 0.
ELSE IF (Z.EQ.ABS(X))
    R = Y/X
    RR = R*R
    RR2 = SQRT(1+RR)
    X = X*RR2
    Y = (1-RR2)/R
    C = 1./RR2
    S = -C*R
ELSE
    R = X/Y
    RR = R*R
    RR2 = SQRT (1+RR)
    X = SIGN(Y, X)*RR
    Y = R - SIGN(RR,R)
    S = SIGN(1./RR2, R)
    C = S*R
END IF
RETURN
END
```

Решетчатым структурам в графе алгоритма соответствуют вложенные параметрические циклы (один меняющийся параметр - одно измерение решетчатой структуры). Эта взаимосвязь ещё раз демонстрирует отношение трёхмерной визуализации ГА к вычислительным алгоритмам полиномиальной сложности, не превосходящей O(n^3) в идеальном случае. Соответственно,

вычислительным операциям, расположенным в теле цикла или гнезда циклов, соответствует вершина графа алгоритма либо несколько его вершин, образующих макроблок. В качестве примера покажем взаимосвязь между исходным кодом референтной реализации и ГА для метода Гивенса QR-разложения матрицы.

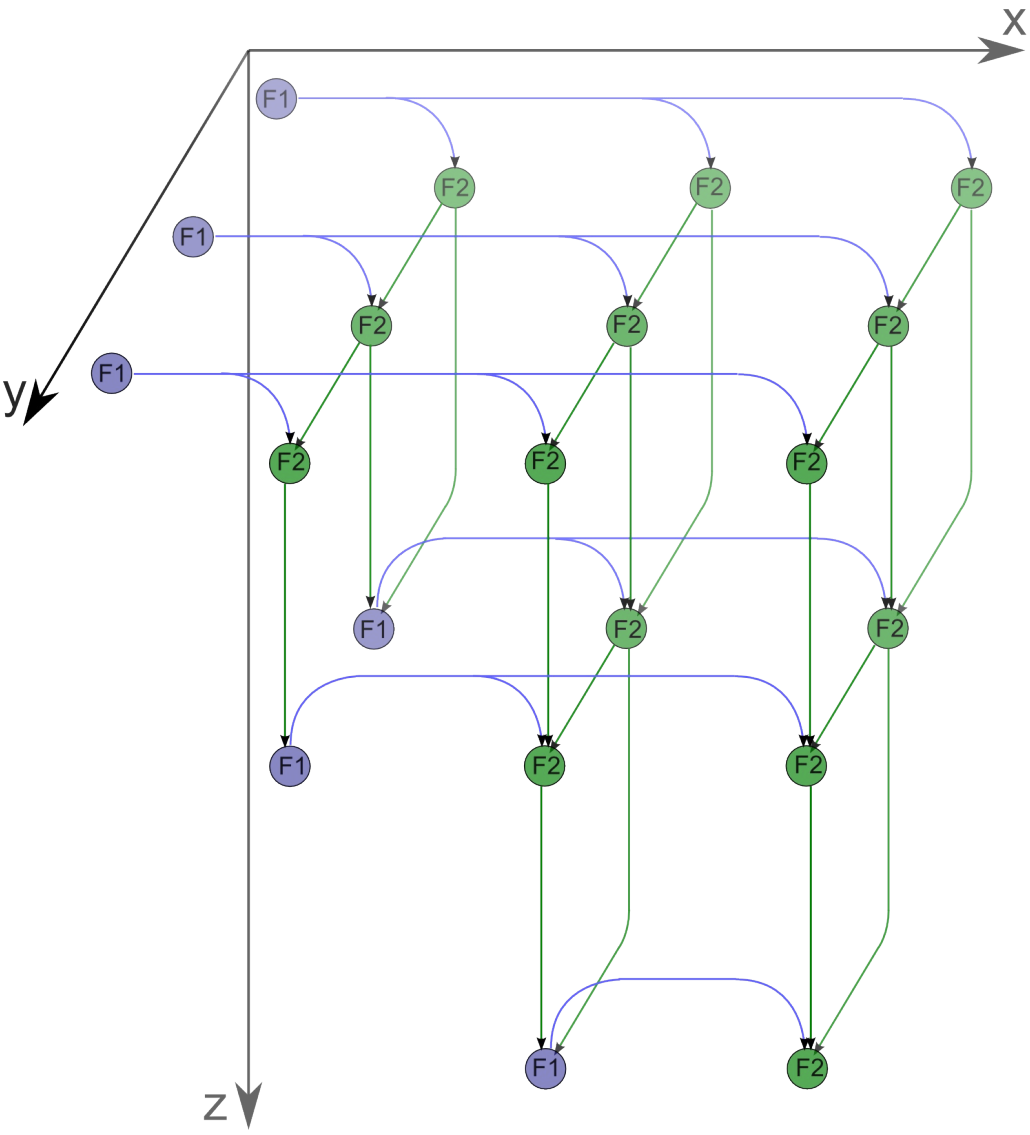
Внешнему циклу соответствует смещение по оси oZ

Среднему циклу соответствует смещение по оси oY

Внутреннему циклу соответствует смещение по оси oX

Вершинам, помеченным F1, соответствует операция вычисления параметров вращения.

Вершинам, помеченным F2, соответствует операция выполнения вращения.



Более сильной связью с исходным кодом референтной реализации обладает ярусно-

параллельная форма графа алгоритма. Поэтому перед дальнейшим анализом приведем методику отдельного отображения ярусно-параллельной формы графа алгоритма по имеющейся визуализации графа алгоритма в введенной стандартной форме.

Отображение ярусно-параллельной формы алгоритма. Определение ярусно-параллельной формы дано в начале работы. Возможны различные способы отображения ярусно-параллельной формы графа алгоритма, однако наиболее логичны следующие два метода :

- Граф алгоритма изображается в своём обычном виде , после чего некоторым образом выделяется первый ярус графа алгоритма. На следующем изображении аналогичные действия совершаются со вторым ярусом , процесс продолжается , пока не остались отображенные ярусы. Результатом является набор изображений , каждое из которых содержит граф алгоритма с выделением одного из ярусов. Существует ПО , способное генерировать подобные изображения в классическом виде.
- Обратим внимание , что ярусно-параллельная форма двумерна по своей структуре , т. к. обладает двумя главными параметрами — высотой и максимальной шириной яруса. Воспользуемся этим свойством следующим образом. В двумерной декартовой системе координат построим несколько равноудалённых друг от друга прямых , параллельных оси Oy , число которых равно числу ярусов таких , что все точки прямых имеют положительное значение абсциссы. Вне зависимости от размерности исходного графа алгоритма разместим все вершины графа алгоритма на этих прямых так , чтобы все вершины , находящиеся на одном и том же ярусе , принадлежали одной прямой , а вершины на различных ярусах находились на разных прямых. При этом номера ярусов , вершины которых лежат на каждой из прямых расположены в порядке удаления прямых от оси Oy. Полученное изображение назовём **разложением по ярусам** графа алгоритма.

В чистом виде оба этих метода, однако, не слишком удобны для связи с исходным кодом, потому что в первом случае нет строго разграничения различных этапов выполнения алгоритма, а значит, нельзя соотнести произвольный подграф ГА с произвольным блоком исходного кода, а во втором случае теряется структурная схема алгоритма, в особенности вложенность одних операций в другие. Логичным решением видится разбиение не графа алгоритма на ярусы, но исходного кода референтной реализации на блоки, причём основной критерий для выделения части кода в отдельный блок - наличие в ней ни во что не вложенного цикла или гнезда циклов. Каждому такому блоку будет соответствовать 1 ярус, который может, в свою очередь, как размещаться на одной из параллельных плоскостей для трёхмерного отображения ГА, так и являться трёхмерной решетчатой структурой.

Построение ГА по алгоритму - методы

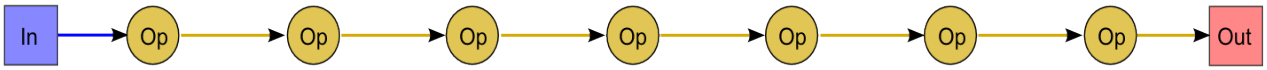
До сих пор мы говорили о случае, когда алгоритм уже имеет заданный граф в какой-либо форме, пусть даже не в визуальной, а, к примеру, в форме таблицы смежности. Однако на

практике граф алгоритма может и не быть определен, то есть алгоритм может иметь описание в какой-то принципиально иной форме - начиная от исходного кода некоторой референтной реализации и заканчивая математическим описанием.

Для построения графа алгоритма необходимо выделить в описании информацию о вершинах графа алгоритма и информацию о дугах графа алгоритма. Поскольку стандартная форма визуализации ГА строится для частных случаев алгоритма на конечных данных малых объёмов, мы можем выделить вершины и дуги, основываясь на референтной реализации.

Практическая часть

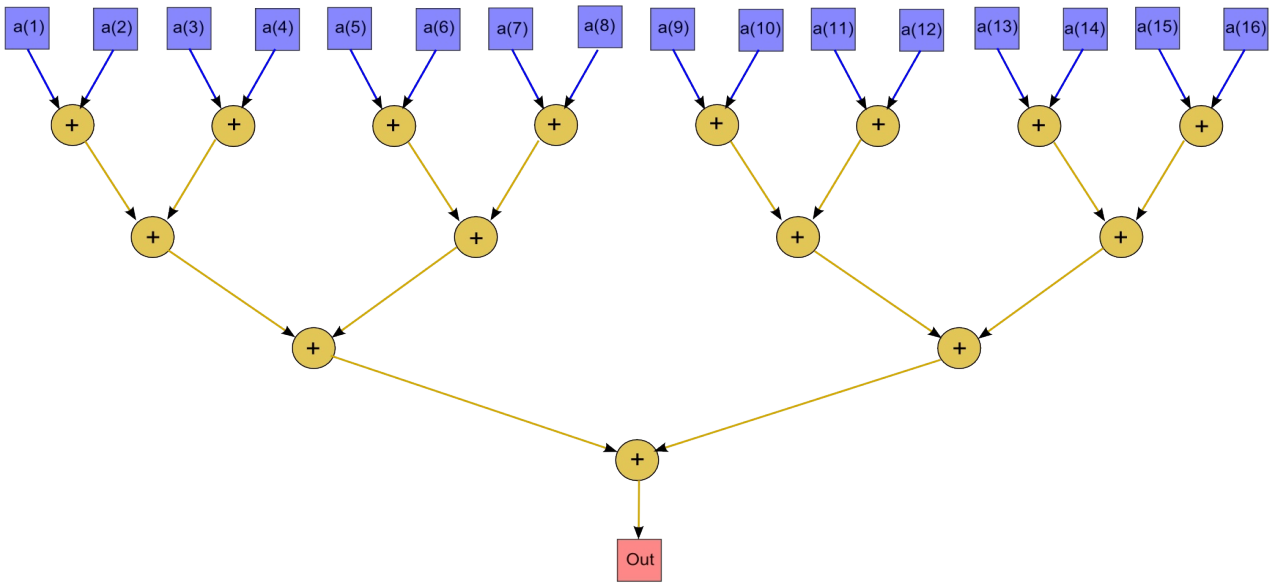
Итак, организована схема представления графа алгоритма в виде, удовлетворяющем ряду поставленных задач. Однако эта схема, за исключением нескольких приведенных примеров, является сугубо теоретическими выкладками. Для отработки принципов визуализации ГА на практике результаты для ряда алгоритмов включаются в их полное описание в качестве одного из пунктов. Граф алгоритма, несмотря на свою взаимосвязь с референтной реализацией является, вообще говоря, машинно-независимой характеристикой алгоритма, поэтому и включается в машинно-независимую часть описания. Подобно тому, как стандарт отображения графа алгоритма разрабатывался для различных графов алгоритмов от наиболее простых и потому узких классов алгоритмов к более широким, визуализации ГА в рамках описания алгоритмов также будут расположены в порядке возрастания сложности их логической структуры. Все приведенные графы алгоритмов снабжаются кратким словесным описанием алгоритма и сноской, поясняющей логическую структуру каждого ГА.



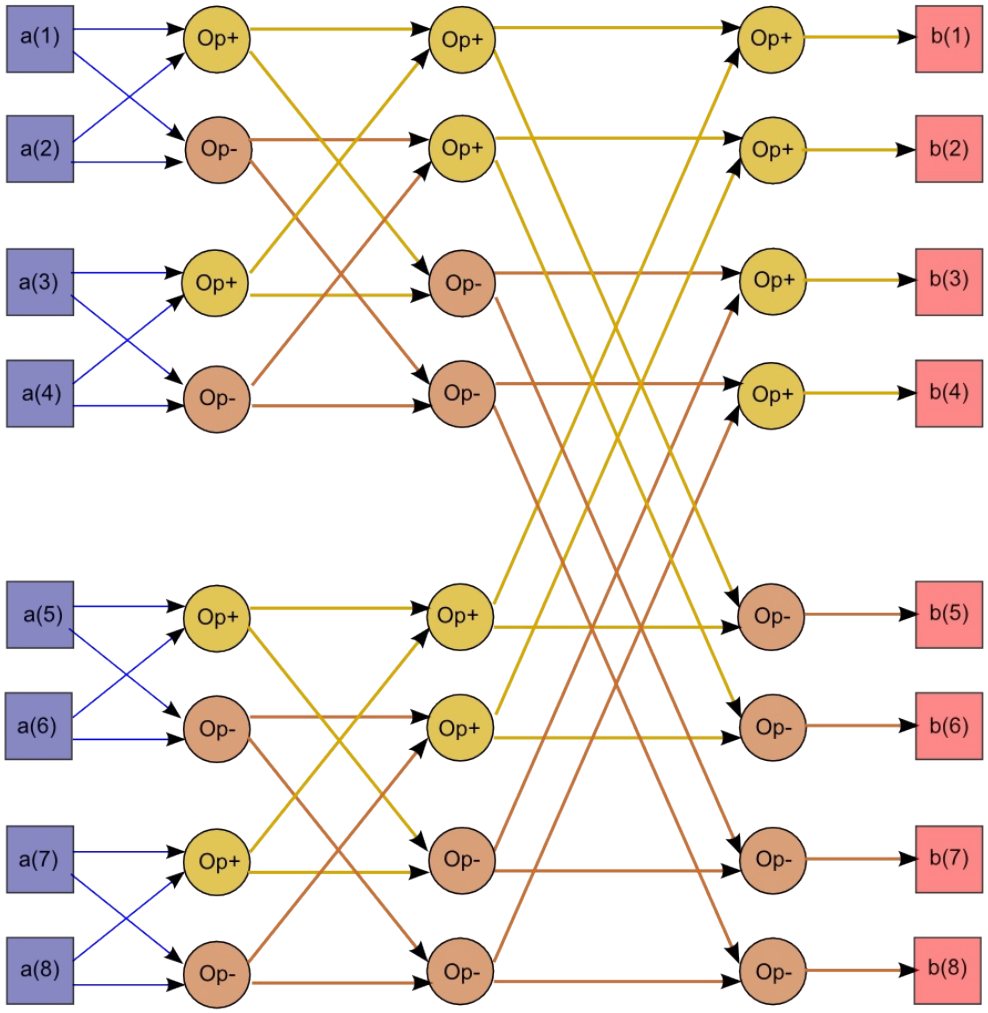
Для начала опишем самый простой возможный алгоритм, описание которого с помощью ГА имеет смысл - это некоторый полностью последовательный алгоритм без какого-либо ресурса параллелизма. Вершина IN отвечает входным данным, вершина Out отвечает выходным данным, вершина OP соответствует некоторой операции.

Теперь приведём описание иных алгоритмов, попадающих в категорию "простейших" по нашей классификации через размерность логической структуры алгоритма.

Суммирование массива методом сдваивания

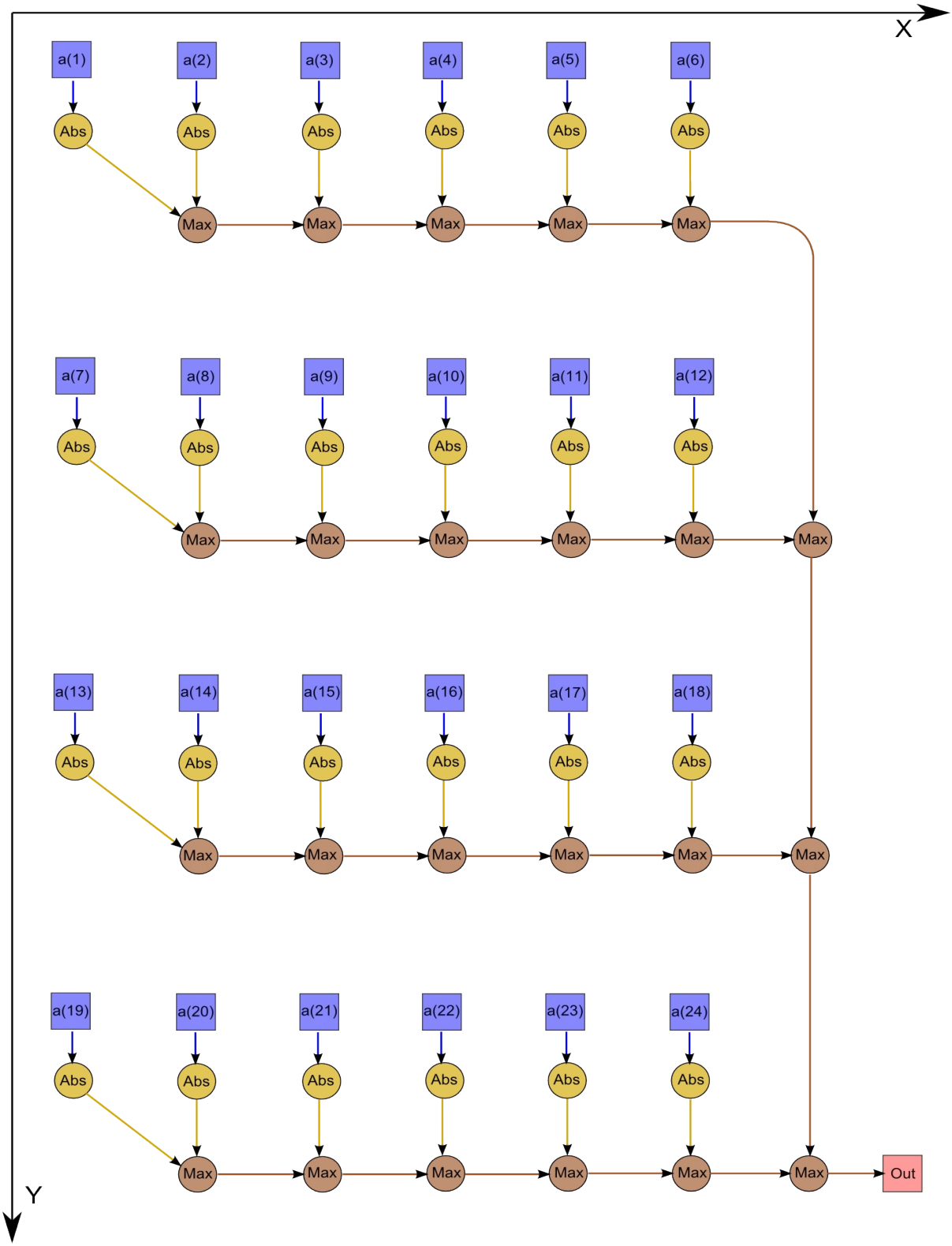


Простой алгоритм Кули-Тьюки для быстрого преобразования Фурье. Этот простейший алгоритм работает исключительно для степеней двойки. Op+ - операция сложения двух комплексных чисел. Op- - операция вычитания двух комплексных чисел и умножения результата вычитания на комплексное число (поворотный множитель). В последнем столбце операций умножение не производится вообще. Привязка вершин выполнена по оси абсцисс — к параметру внешнего цикла, по оси ординат — к обрабатываемым элементам массива.



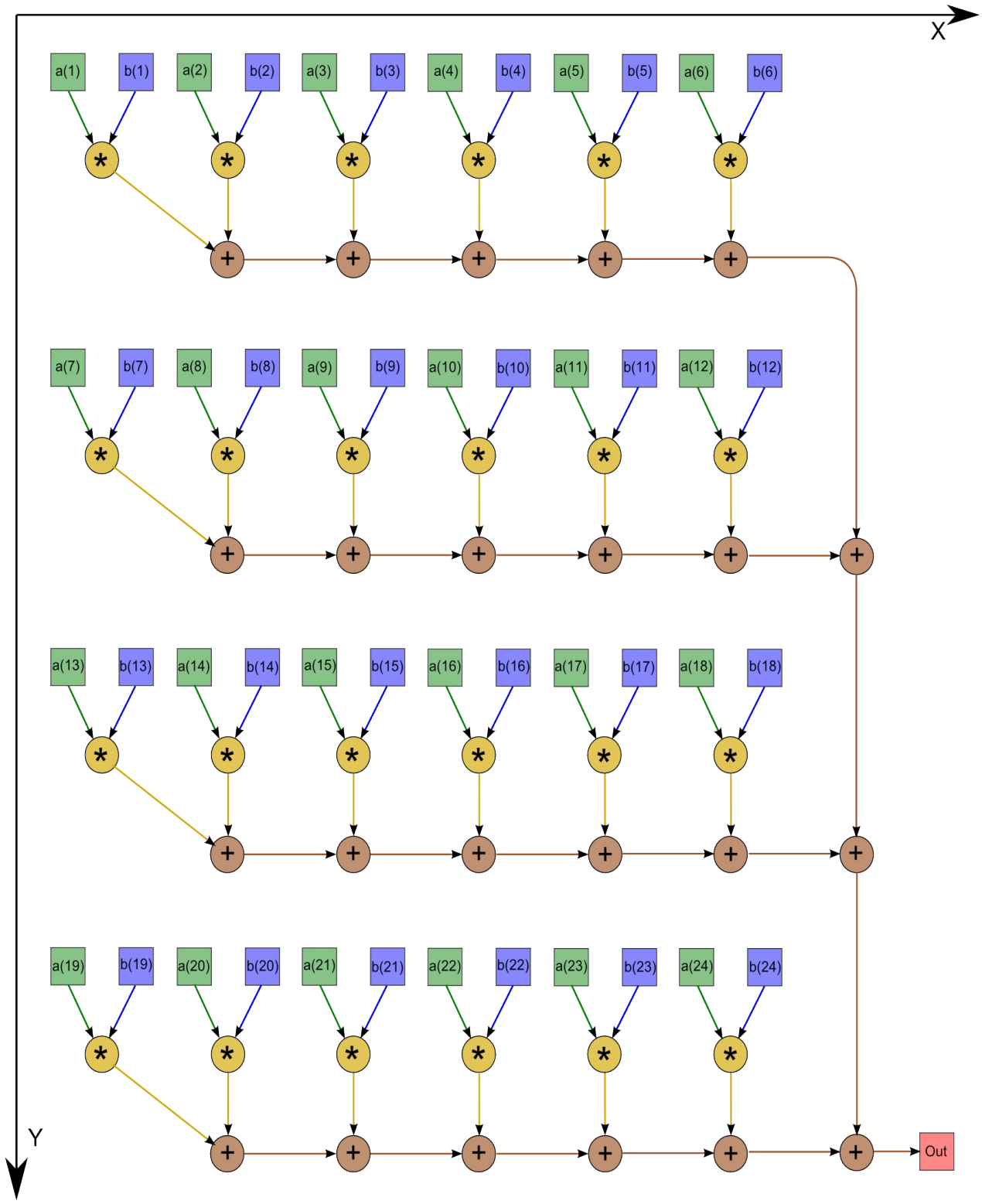
Предложим еще несколько алгоритмов, которые формально входят в класс «простейших» и отображаются в частном случае трёхмерной формы (т.е. только с 1 используемой параллельной плоскостью.

Равномерная норма вектора, последовательно-параллельный вариант.

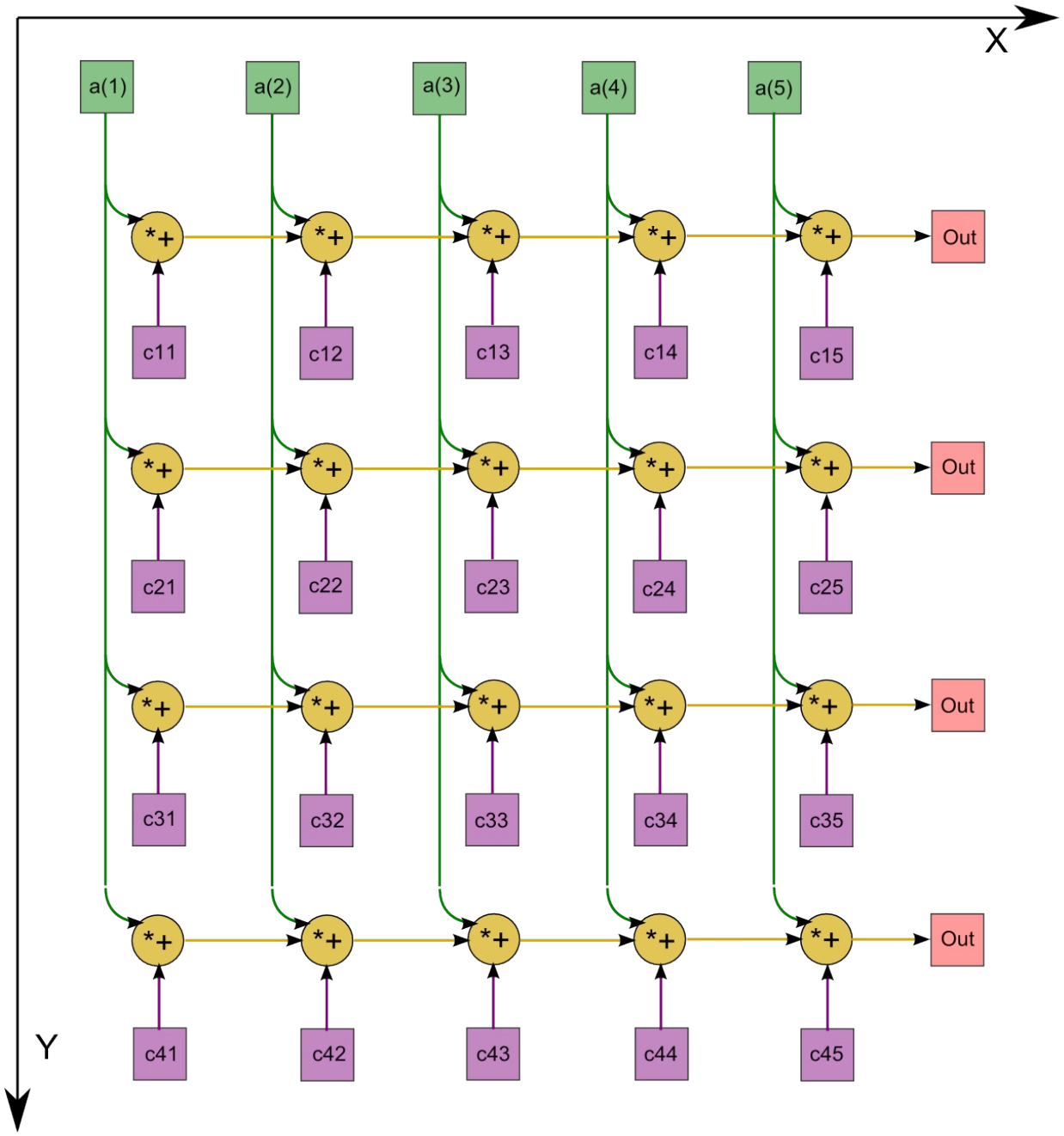


Приведен вариант алгоритма с экономией вызовов функции максимума.

Скалярное произведение векторов, последовательно-параллельный вариант.



Вариант алгоритма с экономией операций сложения.
Умножение плотной матрицы на вектор.



Как и в предыдущих приведенных графах алгоритма, здесь отображаются входные данные. Этот граф алгоритма соответствует базовому варианту алгоритма без каких-либо оптимизаций. Приведем еще несколько визуализаций графов алгоритмов, не относящихся к матрично-векторным операциям, но попадающих в класс «простейших» и отображенных в виде частного случая алгоритмов с трёхмерной структурой.

Общая схема квадратурных (кубатурных) методов численного интегрирования по отрезку

(многомерному кубу).

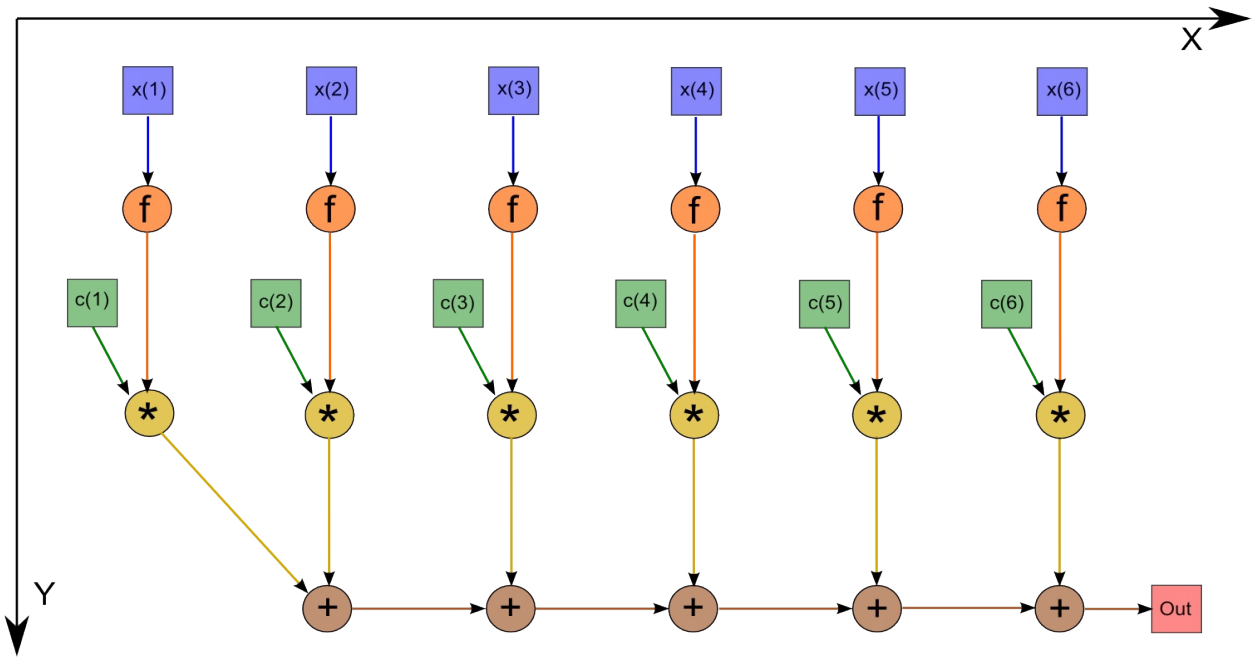
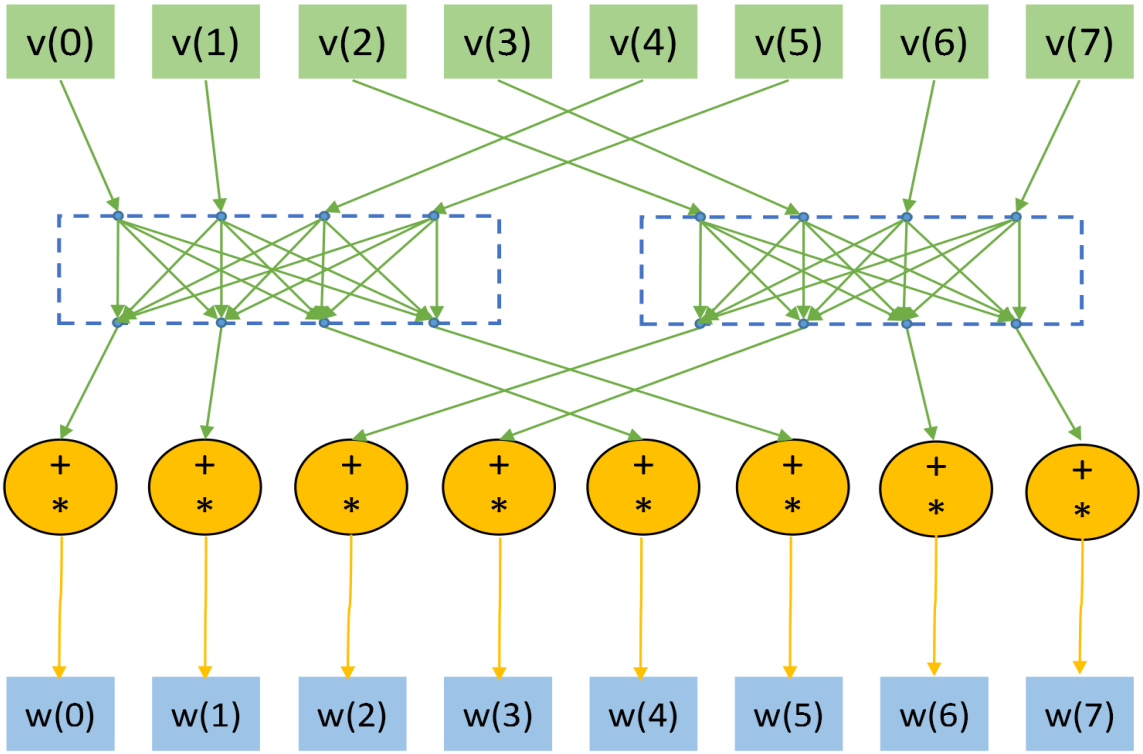
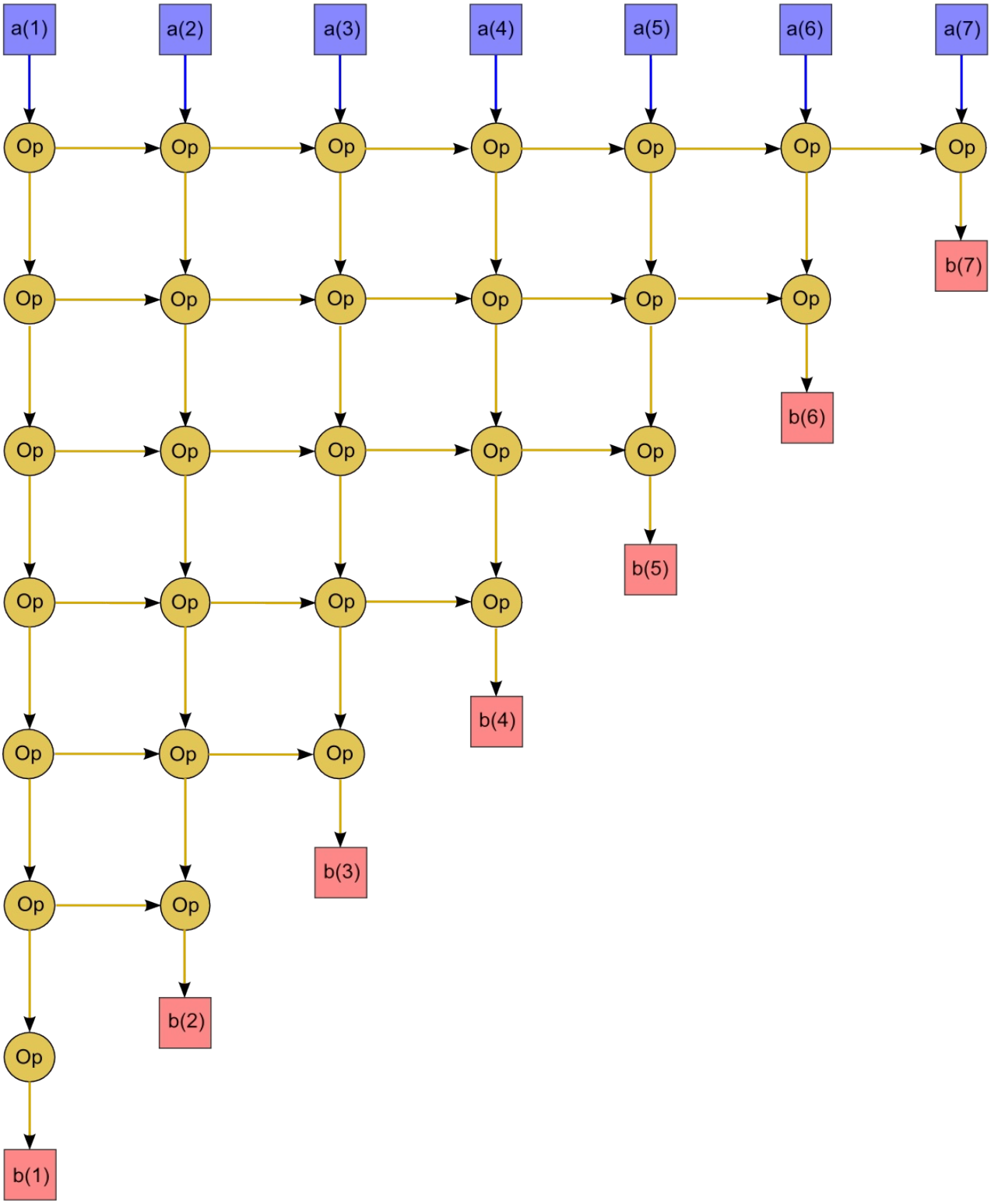


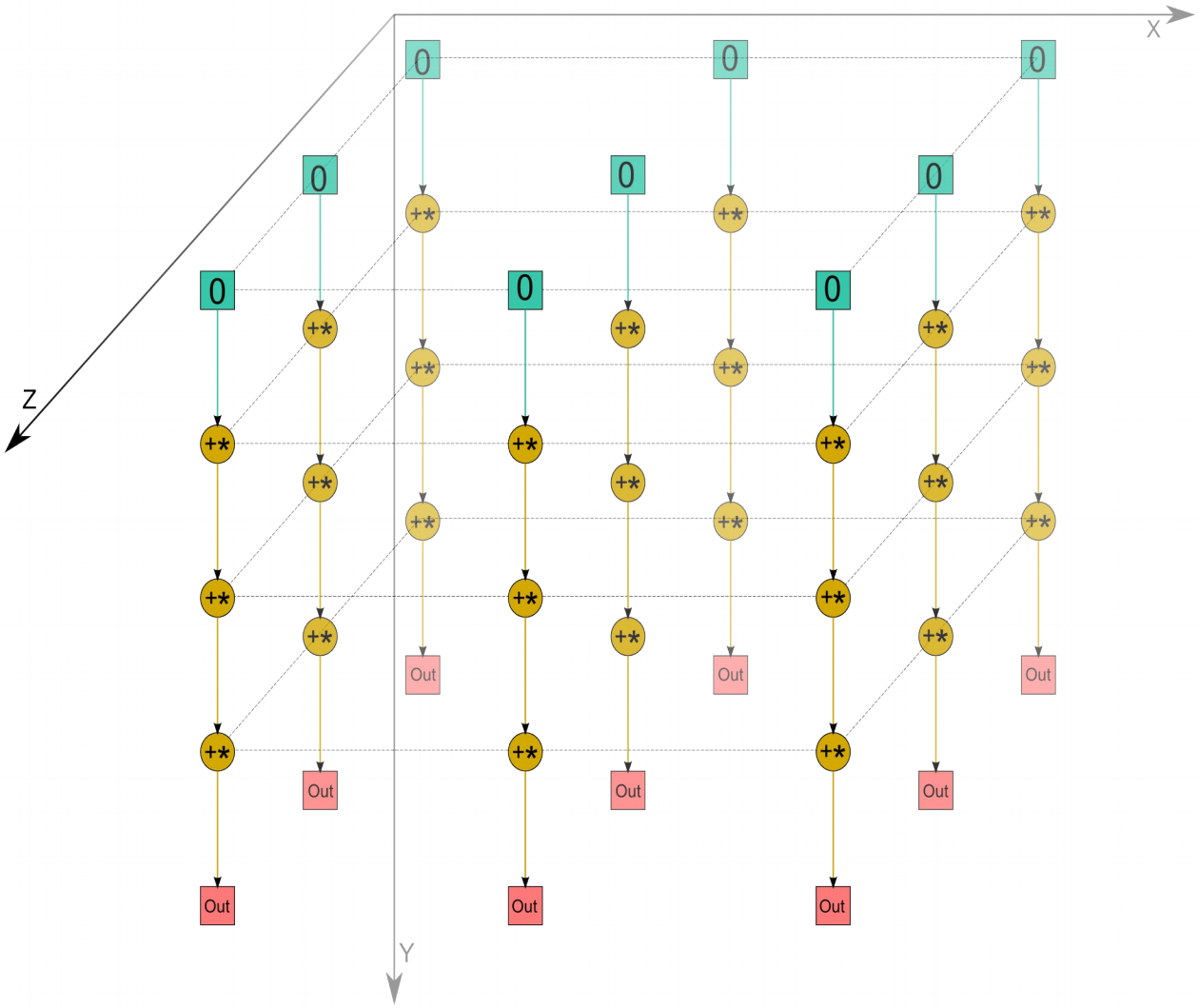
Схема двухкубитного преобразования квантового вектора-состояния.



Сдвиг аргументов многочлена по схеме Горнера, последовательный вариант. Каждая вершина, за исключением входных и выходных, соответствует операции cd + e.

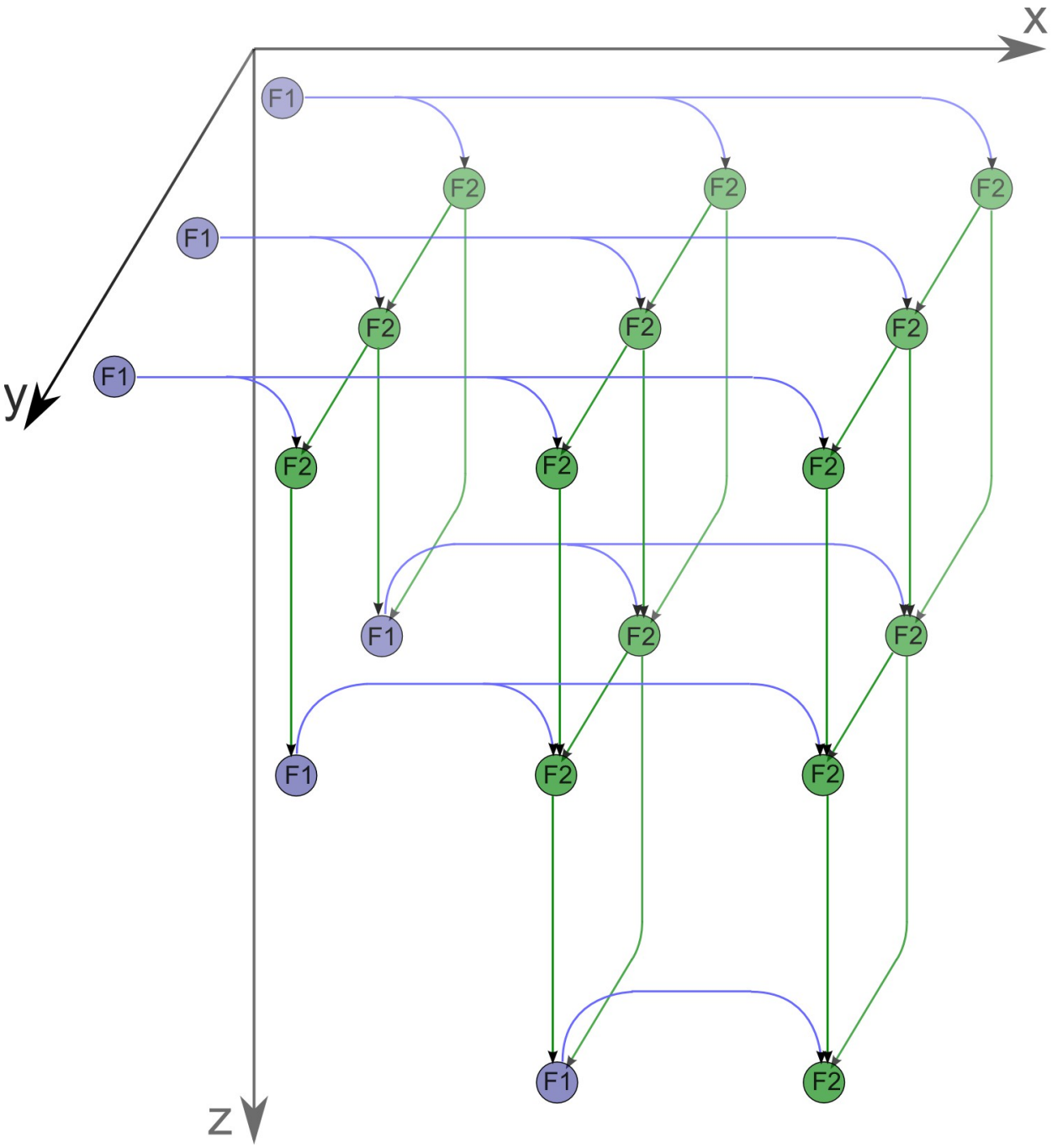


Теперь перейдём к описанию более сложных алгоритмов, а именно к наиболее широкому, как мы выяснили, классу трёхмерных алгоритмов или же алгоритмов, которые могут быть сведены к трёхмерным посредством разбиения на макроблоки. Начнём с наиболее простого и интуитивно понятного алгоритма этого класса — перемножения плотных матриц.



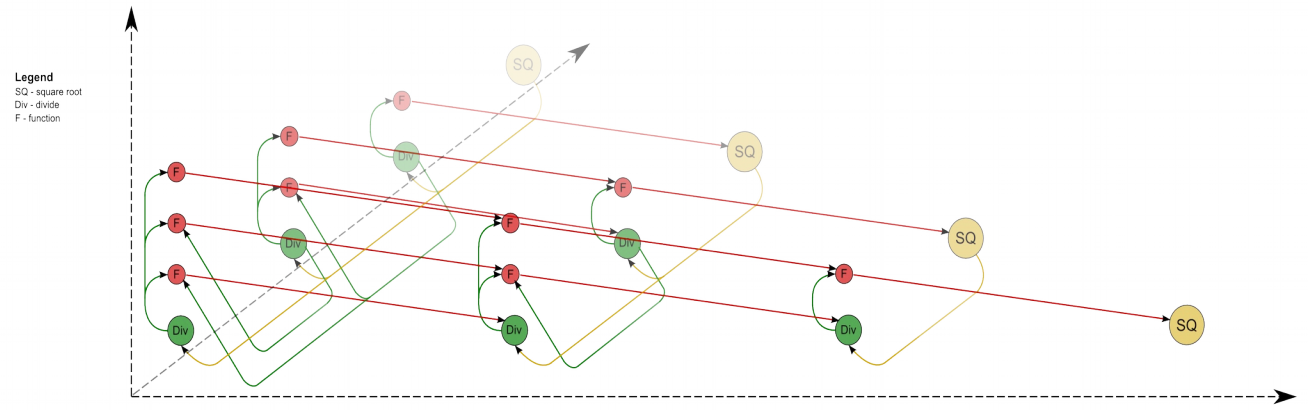
Здесь смещению по оси oY соответствует внутренний цикл, вычисляющий некоторый элемент результата, смещениям по оси oX и oZ соответствуют внешние циклы, их порядок с точки зрения структуры графа алгоритма значения не имеет, а для квадратных матриц, как на приведенном примере, и вовсе незаметен. Обратим внимание, что здесь алгоритм представлен в ярусно-параллельной форме по второй из описанных нами методик, но в трёхмерном пространстве, то есть с сохранением решетчатой структуры по циклам. Вспомогательные пунктирные линии отмечают параллельные плоскости, на которых размещаются вершины графа алгоритма. В каждой вершине проводится операция сложения произведения двух элементов входных матриц с результатом, полученном на предыдущем этапе.

Уже знакомый нам метод Гивенса для QR-разложения матрицы.

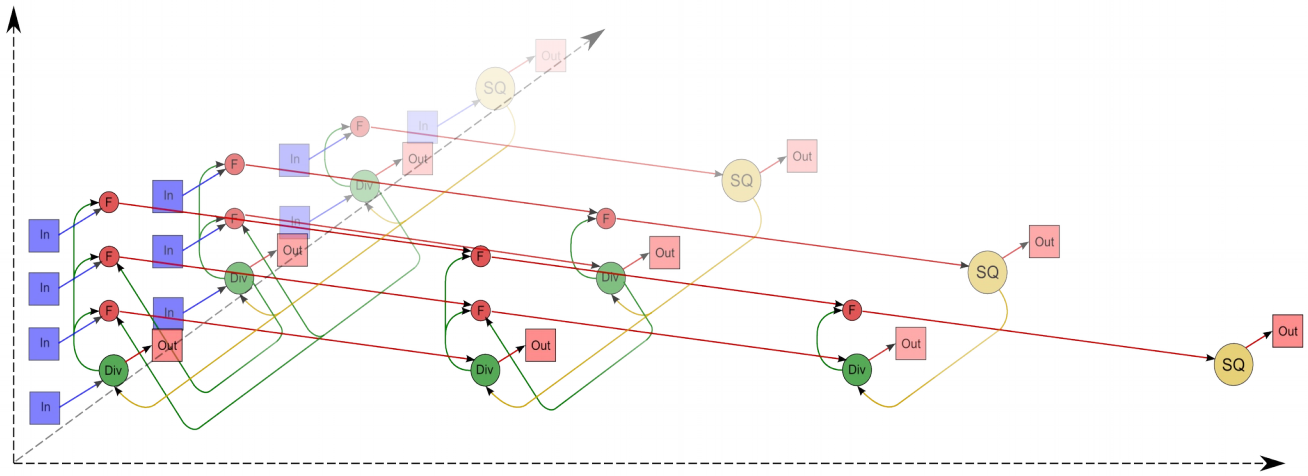


Описывать этот алгоритм подробнее нужды нет, так как его структура была объяснена в разделе «связь графа алгоритма с другими формами описания алгоритма». Отметим лишь, что выполнение собственно поворота эквивалентно операции умножения двух комплексных чисел.

Метод Холецкого для LU-разложения матрицы. Приведены варианты с отображением входных и выходных данных и без них, так как структура данных, особенно выходных, достаточно нетривиальна.



Это граф алгоритма без отображения входных и выходных данных. SQ – вычисление квадратного корня, F – операция a – bc, Div – деление.



Тот же граф алгоритма, но уже с отображением входных и выходных данных. Оба графа приведены для случая n = 4, что соответствует правилу о использовании для визуализации частного случая алгоритма для конечного и малого объёма входных данных, однако, достаточного для отображения логической структуры алгоритма.

Заключение.

В рамках данной работы был решен ряд задач, связанных визуализацией графов алгоритмов, а именно :

1) Разработан набор рекомендаций по отрисовке графов алгоритмов, по которым имеется полная информация о числе дуг и числе вершин, обеспечивающий единый стиль визуализации графа алгоритма независимо от конкретного алгоритма.
2) Разработан набор методов, позволяющих изобразить в понятной форме графы алгоритмов, имеющие сложную внутреннюю структуру, в том числе определены конкретные методы для отображения ряда характерных особенностей, присущих графам алгоритмов.
3) Набор имеющихся методик использован для визуализации ряда графов алгоритмов различной сложности, которые используются в качестве элемента полного описания алгоритма в рамках проекта Algowiki.

Список литературы.

1) Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. – СПб.: БХВ - Петербург, 2002.
2) В.В.Воеводин, Ю.А.Кузнецов. Матрицы и вычисления. М.: Наука, 1984.
3) Алексеев В. Б., Ложкин С. А. *Элементы теории графов и схем.* - М.: Изд-во МГУ, 1991.
4) http://algowiki-project.org/
5) **Emerging Heterogeneous Technologies for High Performance Computing,** 22nd International Heterogeneity in Computing Workshop, IPDPS, May 20, 2013, Boston, MA.