Билет 35. Информационный граф и ресурс параллелизма алгоритмов.

Граф алгоритма - ориентированный граф, состоящий из вершин, соответствующих операциям, и направленных дуг, соответствующих передаче данных (результаты одних операций передаются в качестве аргументов другим операциям) между ними.

Не следует путать его с графом управления программы и тем более с её блок-схемой.

Активно используется при исследованиях скрытого параллелизма в алгоритмах, записанных на традиционных языках программирования последовательного типа.

Особенностями графа алгоритма являются:

- его ацикличность;
- невозможность, в общем случае, его описания простым перечислением, в силу того, что его составляющие могут зависеть от внешних параметров решаемой им задачи (например, алгоритм, реализующий метод Гаусса — от размера матрицы).

Algo-Wiki:

Интересных вариантов для отражения информационной структуры алгоритмов много. Для каких-то алгоритмов нужно показать максимально подробную структуру, а иногда важнее макроструктура. Много информации несут разного рода проекции информационного графа, выделяя его регулярные составляющие и одновременно скрывая несущественные детали. Иногда оказывается полезным показать последовательность в изменении графа при изменении значений внешних переменных (например, размеров матриц): мы часто ожидаем "подобное" изменение информационного графа, но это изменение не всегда очевидно на практике.

В целом, задача изображения графа алгоритма весьма нетривиальна. Начнем с того, что это потенциально бесконечный граф, число вершин и дуг которого определяется значениями внешних переменных, а они могут быть весьма и весьма велики. В такой ситуации, как правило, спасают упомянутые выше соображения подобия, делающие графы для разных значений внешних переменных "похожими": почти всегда достаточно привести лишь один граф небольшого размера, добавив, что графы для остальных значений будут устроены "точно также". На практике, увы, не всегда все так просто, и здесь нужно быть аккуратным.

Далее, граф алгоритма - это потенциально многомерный объект. Наиболее естественная система координат для размещения вершин и дуг информационного графа опирается на структуру вложенности циклов в реализации алгоритма. Если глубина вложенности циклов не превышает трех, то и граф размещается в привычном трехмерном пространстве, однако для более сложных циклических конструкций с глубиной вложенности 4 и больше необходимы специальные методы представления и изображения графов.

Но в любом случае нужно не забывать главную задачу данного раздела - показать информационную структуру алгоритма так, чтобы стали понятны все его ключевые особенности, особенности параллельной структуры, особенности множеств дуг, участки регулярности и, напротив, участки с недерминированной структурой, зависящей от входных данных.

1.8 Ресурс параллелизма алгоритма

Здесь приводится оценка *параллельной сложности* алгоритма: числа шагов, за которое можно выполнить данный алгоритм в предположении доступности неограниченного числа

необходимых процессоров (функциональных устройств, вычислительных узлов, ядер и т.п.). Параллельная сложность алгоритма понимается как высота канонической ярусно-параллельной формы [1]. Необходимо указать, в терминах каких операций дается оценка. Необходимо описать сбалансированность параллельных шагов по числу и типу операций, что определяется шириной ярусов канонической ярусно-параллельной формы и составом операций на ярусах.

Параллелизм в алгоритме часто имеет естественную иерархическую структуру. Этот факт очень полезен на практике, и его необходимо отразить в описании. Как правило, подобная иерархическая структура параллелизма хорошо отражается в последовательной реализации алгоритма через циклический профиль результирующей программы (конечно же, с учетом графа вызовов), поэтому циклический профиль (п.1.5) вполне может быть использован и для отражения ресурса параллелизма.

Для описания ресурса параллелизма алгоритма (ресурса параллелизма информационного графа) необходимо указать ключевые параллельные ветви в терминах *конечного* и *массового* параллелизма. Далеко не всегда ресурс параллелизма выражается просто, например, через *координатный параллелизм* или, что то же самое, через независимость итераций некоторых циклов (да-да-да, циклы - это понятие, возникающее лишь на этапе реализации, но здесь все так связано… В данном случае, координатный параллелизм означает, что информационно независимые вершины лежат на гиперплоскостях, перпендикулярных одной из координатных осей). С этой точки зрения, не менее важен и ресурс *скошенного параллелизма*. В отличие от координатного параллелизма, скошенный параллелизм намного сложнее использовать на практике, но знать о нем необходимо, поскольку иногда других вариантов и не остается: нужно оценить потенциал алгоритма, и лишь после этого, взвесив все альтернативы, принимать решение о конкретной параллельной реализации.

Ярусно-параллельная форма (ЯПФ) (parallel form) - это представление графа алгоритма, в котором:

- все вершины разбиты на перенумерованные подмножества ярусов;
- начальная вершина каждой дуги расположена на ярусе с номером меньшим, чем номер яруса конечной вершины;
- между вершинами, расположенными на одном ярусе, не может быть дуг.

Высота ЯПФ - это число ярусов. Ширина яруса - число вершин, расположенных на ярусе. *Ширина* ЯПФ - это максимальная ширина ярусов в ЯПФ.

Канонической ярусно-параллельной формой называется ЯПФ, высота которой на единицу больше длины критического пути, а все входные вершины расположены на первом ярусе. Для заданного графа его каноническая ЯПФ единственна. Высота канонической ЯПФ соответствует параллельной сложности алгоритма.

Ярусно-параллельная форма графа (ЯПФ) — деление вершин ориентированного ациклического графа на перенумерованные подмножества V_i такие, что, если дуга e идет от вершины $v_1 \in V_j$ к вершине $v_2 \in V_k$, то обязательно $j < k$.

Каждое из множеств V_i называется **ярусом** ЯПФ, i — его **номером**, количество вершин $|V_i|$ в ярусе — его **шириной**. Количество ярусов в ЯПФ называется её **высотой**, а максимальная ширина её ярусов — **шириной ЯПФ**.

Для ЯПФ графа алгоритма важным является тот факт, что операции, которым соответствуют вершины одного яруса, не зависят друг от друга (не находятся в отношении связи), и поэтому заведомо существует параллельная реализация алгоритма, в которой они

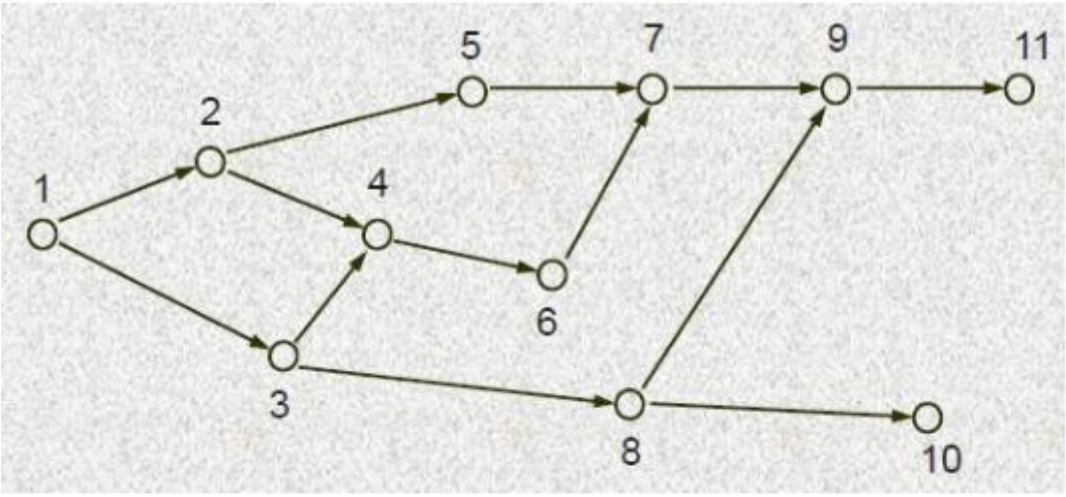
могут быть выполнены параллельно на разных устройствах вычислительной системы. Поэтому ЯПФ графа алгоритма может быть использована для подготовки такой параллельной реализации алгоритма.

Минимальной высотой всех возможных ЯПФ графа является его критический путь. Построение ЯПФ с высотой, меньшей критического пути, невозможно.

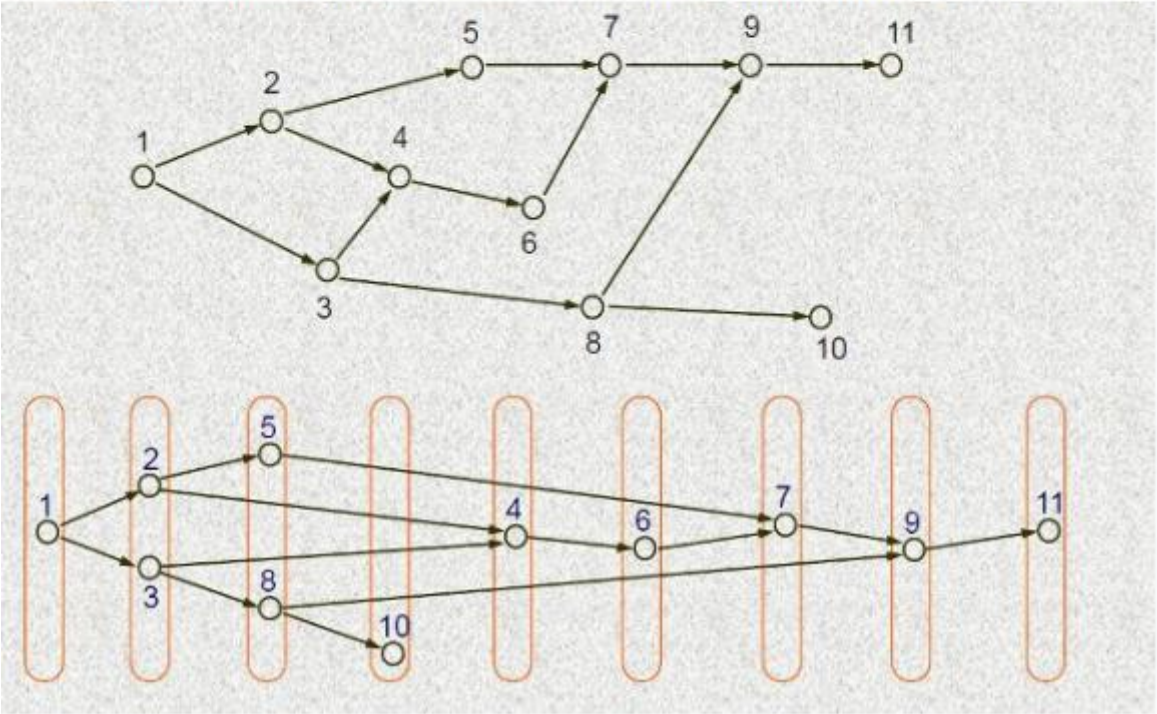
Критический путь графа — путь максимальной длины в ориентированном ациклическом графе.

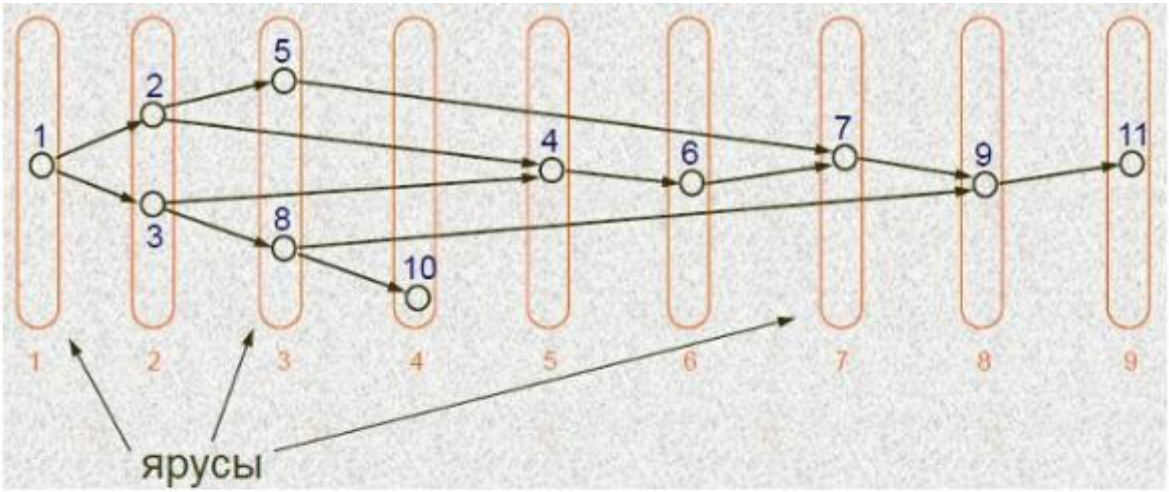
Его длина является минимальной из всех возможных высот у ярусно-параллельной формы данного ациклического графа.

Параллельная сложность (parallel complexity) алгоритма - число шагов, за которое можно выполнить данный алгоритм в предположении доступности неограниченного числа необходимых процессоров (функциональных устройств, вычислительных узлов, ядер и т.п.). Параллельная сложность алгоритма понимается как высота канонической ярусно-параллельной формы.

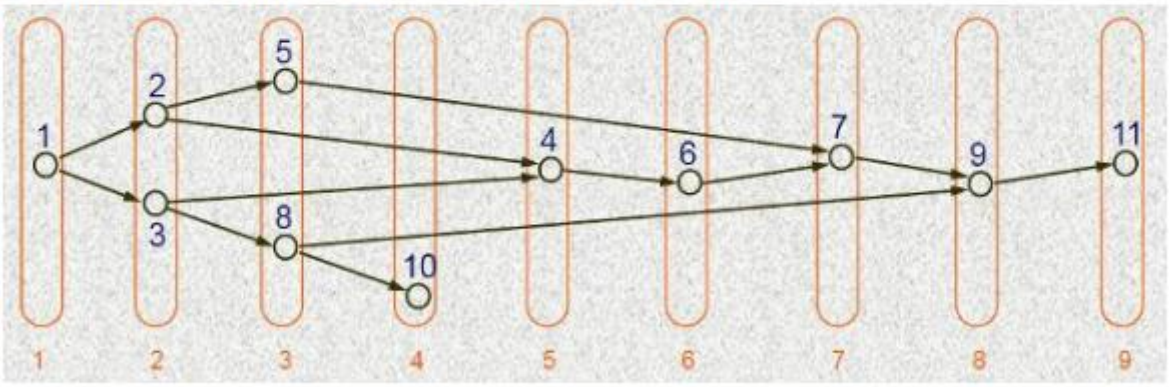


Как определить и сделать понятным ресурс параллелизма в графе алгоритма (в программе, в алгоритме) ?





- начальная вершина каждой дуги расположена на ярусе с номером меньшим, чем номер яруса конечной вершины,
- между вершинами, расположенными на одном ярусе, не может быть дуг.

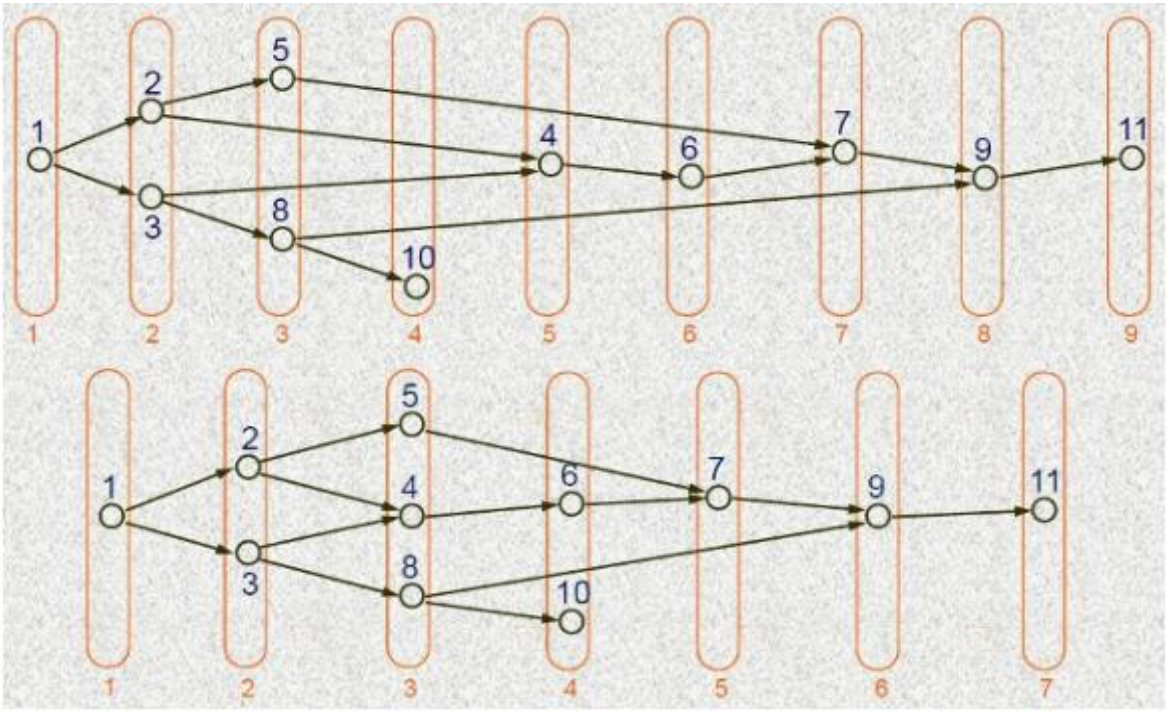


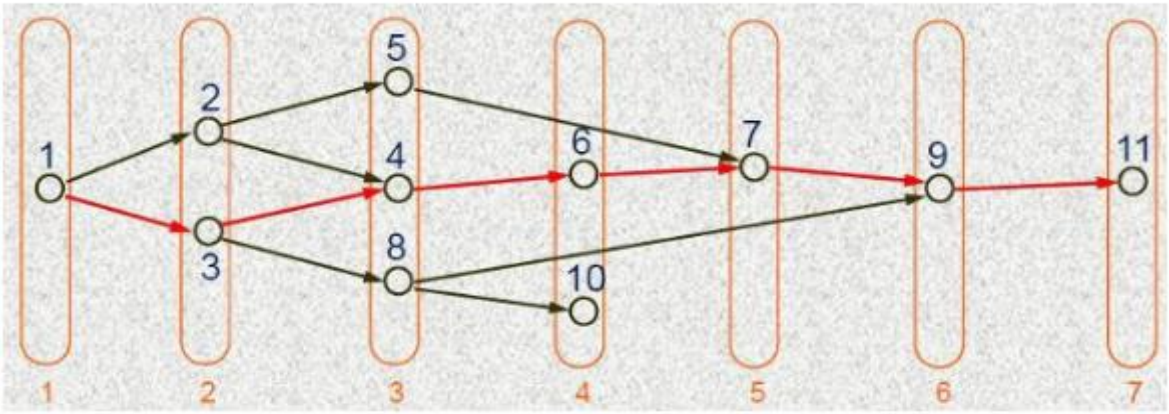
Высота ЯПФ – это число ярусов,

Ширина яруса – число вершин, расположенных на ярусе,

Ширина ЯПФ – это максимальная ширина ярусов в ЯПФ.

Высота ЯПФ = сложность параллельной реализации алгоритма/программы.





Высота канонической ЯПФ = длине критического пути + 1.

```
for( i = 0; i < n; ++i)
   for( j = 0; j < m; ++j)
      A[i][j] = A[i][j-1] + C[i][j]*x;
```

На примере Разложения Холецкого (метод квадратного корня)

1.7 Информационный граф

Опишем граф алгоритма[9][10][11] как аналитически, так и в виде рисунка.

Граф алгоритма состоит из трёх групп вершин, расположенных в целочисленных узлах трёх областей разной размерности.

Первая группа вершин расположена в одномерной области, соответствующая ей операция вычисляет функцию SQRT. Единственная координата каждой из вершин i меняется в диапазоне от 1 до n, принимая все целочисленные значения.

Аргумент этой функции

- при $i = 1$ — элемент *входных данных*, а именно a_{11};
- при $i > 1$ — результат срабатывания операции, соответствующей вершине из третьей группы, с координатами $i-1, i, i-1$.

Результат срабатывания операции является *выходным данным* l_{ii}.

Вторая группа вершин расположена в двумерной области, соответствующая ей операция a/b. Естественно введённые координаты области таковы:

- i — меняется в диапазоне от 1 до $n-1$, принимая все целочисленные значения;
- j — меняется в диапазоне от $i+1$ до n, принимая все целочисленные значения.

Аргументы операции следующие:

- a:
  - при $i = 1$ — элементы *входных данных*, а именно a_{j1};
  - при $i > 1$ — результат срабатывания операции, соответствующей вершине из третьей группы, с координатами $i-1, j, i-1$;
- b — результат срабатывания операции, соответствующей вершине из первой группы, с координатой i.

Результат срабатывания операции является *выходным данным* l_{ji}.

Третья группа вершин расположена в трёхмерной области, соответствующая ей операция $a - b * c$. Естественно введённые координаты области таковы:

- i — меняется в диапазоне от 2 до n, принимая все целочисленные значения;
- j — меняется в диапазоне от i до n, принимая все целочисленные значения;
- p — меняется в диапазоне от 1 до $i-1$, принимая все целочисленные значения.

Аргументы операции следующие:

- a:
  - при $p = 1$ элемент входных данных a_{ji};
  - при $p > 1$ — результат срабатывания операции, соответствующей вершине из третьей группы, с координатами $i, j, p-1$;
- b — результат срабатывания операции, соответствующей вершине из второй группы, с координатами p, i;
- c — результат срабатывания операции, соответствующей вершине из второй группы, с координатами p, j;

Результат срабатывания операции является *промежуточным данным* алгоритма.

Описанный граф можно посмотреть на рис.1 и рис.2, выполненных для случая $n = 4$. Здесь вершины первой группы обозначены жёлтым цветом и буквосочетанием sq, вершины второй — зелёным цветом и знаком деления, третьей — красным цветом и буквой f. Вершины, соответствующие операциям, производящим выходные данные алгоритма, выполнены более крупно. Дублирующие друг друга дуги даны как одна. На рис.1 показан граф алгоритма согласно классическому определению , на рис.2 к графу алгоритма добавлены вершины , соответствующие входным (обозначены синим цветом) и выходным (обозначены розовым цветом) данным.

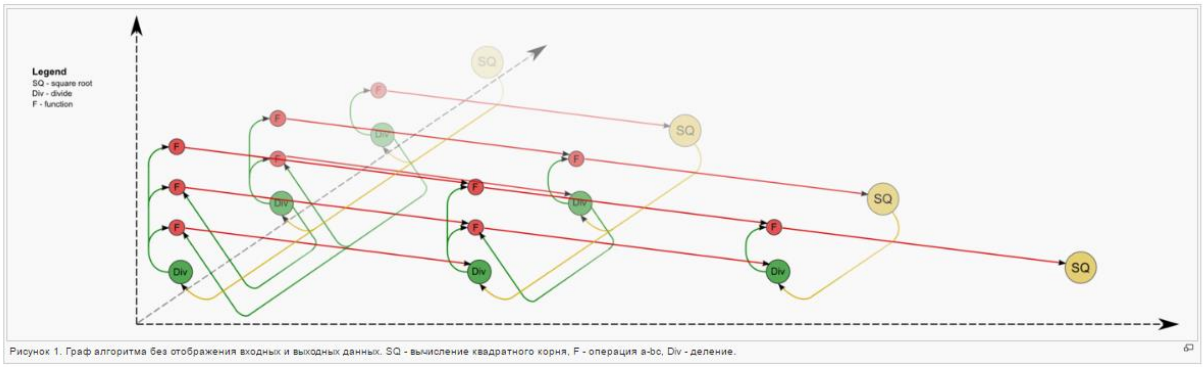


Рисунок 1. Граф алгоритма без отображения входных и выходных данных. SQ - вычисление квадратного корня, F - операция a-bc, Div - деление.

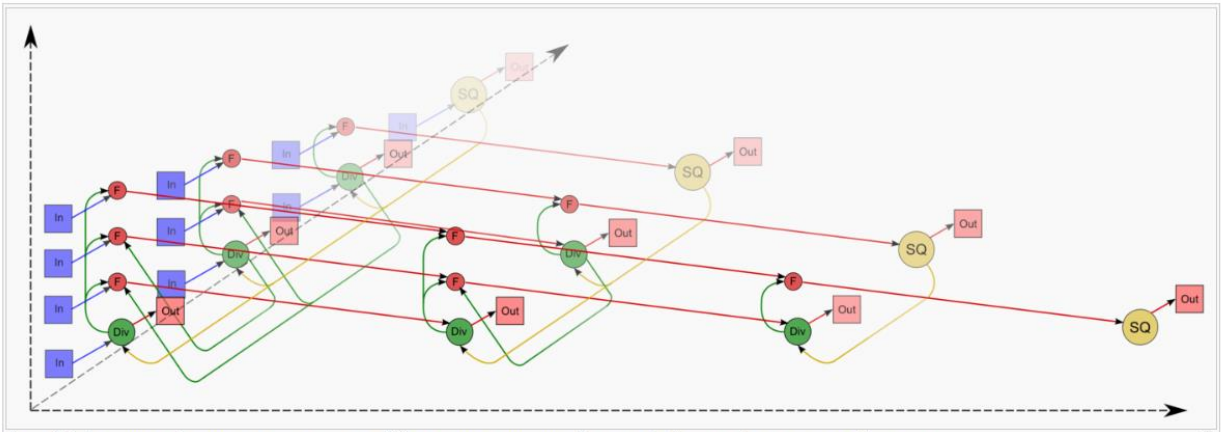


Рисунок 2. Граф алгоритма с отображением входных и выходных данных. SQ - вычисление квадратного корня, F - операция a-bc, Div - деление, In - входные данные, Out - результаты.

1.8 Ресурс параллелизма алгоритма

Для разложения матрицы порядка n методом Холецкого в параллельном варианте требуется последовательно выполнить следующие ярусы:

- n ярусов с вычислением квадратного корня (единичные вычисления в каждом из ярусов),
- $n-1$ ярус делений (в каждом из ярусов линейное количество делений, в зависимости от яруса — от 1 до $n-1$),
- по $n-1$ ярусов умножений и сложений/вычитаний (в каждом из ярусов — квадратичное количество операций, от 1 до $\frac{n^2-n}{2}$.

Таким образом, в параллельном варианте, в отличие от последовательного, вычисления квадратных корней и делений будут определять довольно значительную долю требуемого времени. При реализации на конкретных архитектурах наличие в отдельных ярусах ЯПФ отдельных вычислений квадратных корней может породить и другие проблемы. Например, при реализации на ПЛИСах остальные вычисления (деления и тем более умножения и сложения/вычитания) могут быть конвейеризованы, что даёт экономию и по ресурсам на программируемых платах; вычисления же квадратных корней из-за их изолированности приведут к занятию ресурсов на платах, которые будут простаивать большую часть времени. Таким образом, общая экономия в 2 раза, из-за которой метод Холецкого предпочитают в случае симметричных задач тому же методу Гаусса, в параллельном случае уже имеет место вовсе не по всем ресурсам, и главное - не по требуемому времени.

При этом использование режима накопления требует совершения умножений и вычитаний в режиме двойной точности, а в параллельном варианте это означает, что практически все промежуточные вычисления для выполнения метода Холецкого в режиме накопления должны быть

двойной точности. В отличие от последовательного варианта это означает увеличение требуемой памяти почти в 2 раза.

При классификации по высоте ЯПФ, таким образом, метод Холецкого относится к алгоритмам со сложностью $O(n)$. При классификации по ширине ЯПФ его сложность будет $O(n^2)$.