«Проблема повышения скорости компьютерных вычислений»

Тема: «Оптимизация архитектуры процессоров, суперкомпьютеры»

Архитектура конкретной ЭВМ определяется следующими пунктами

1) Архитектурными принципами, заложенными во внутреннюю архитектуру простого или сложного универсального процессора.
2) Архитектура процессора определенна конкретными конструктивно-технологическими решениями, использующимися в конкретном процессорном элементе.
3) Архитектура ЭВМ определяется языком программирования низкого уровня Assembler.

Архитектурные принципы

Элементарной базой современного этапа развития ЭВМ является **сверхбольшая интегральная схема (СБИС**). В отличие от обычных интегральных схем СБИС характеризуется сверхвысокой степенью концентрации активных элементов на кристалле.

Вычислительные архитектуры на СБИС

1) **Память.** Реализуется на БИС и СБИС. Представляет алгоритм (стек, очередь) доступа к оперативно запоминающему устройству (ОЗУ) и программируемому постоянно запоминающему устройству (ППЗУ). Память может быть как общей, так и локальной.
2) **Логика + память**. Реализуется на программируемой логической интегральной схеме (ПЛИС). Реализация алгоритмов обрабатывается в специально создаваемой вычислительной системе из стандартных вычислительных блоков. В отличие от обычных интегральных схем, логика работы ПЛИС не определяется при изготовлении, а задаётся посредством программирования. При программировании в память ПЛИС записывается двоичный код, с помощью которого создается требуемая конфигурация из элементов, которые имеются в ПЛИС. Кроме операций доступа к памяти могут выполняться логические операции, операции ввода вывода, обработка данных в регистрах. С помощью этой ВА можно сделать вычислительную систему, лишенную избыточности. Альтернативой ПЛИС являются заказные БИС и СБИС, которые существенно дороже, и компьютеры (микроконтроллеры), которые из-за программного способа реализации алгоритмов медленнее ПЛИС.
3) **Систолическая**. Систолическая ВА предназначена для реализации только отдельных классов алгоритмов, в специально создаваемых процессорных сетях, размещаемых на процессорных платах или в заказных СБИС. Такая процессорная плата может представлять собой ускоритель вычислений или сопроцессором. Процессорная плата, и заказная СБИС являются внешними устройствами по отношению к вычислительной системе (ВС). Реализуется на систолических процессорах, цифровых сигнальных процессорах (ЦСП) и транспьютероподобных цифровых сигнальных процессорах (ТП ЦСП). Систолическая схема вычислений представляет собой конвейер, в котором данные проходят обработку от одной границы ВС – входа к другой границе — выходу. Аппаратная реализация алгоритмов в процессорной сети позволяет добиться ускорения, на 1-3 порядка,. В конкретной процессорной сети может быть реализован только один или узкий класс алгоритмов, поэтому реализуемые алгоритмы в ней называют жесткими алгоритмами.
4) **Мультимикропроцессорная.** Реализуется на транспьютерах (Т) и ТП ЦСП. Транспьютер — интегрально (в одном чипе или кристалле) выполненное семейство системных компонент, в составе которых: процессор, таймер, память, каналы последовательного ввода-вывода, контроллер внешней оперативной памяти. Мультимикропроцессорная ВА представляет собой сеть ПЭ, в которой выполняется набор программных модулей. Для реализации используются однородные и неоднородные процессорные сети. Однородные состоят из одинаковых ПЭ. Неоднородные – из ПЭ разных типов, для объединения которых в единую вычислительную сеть, требуются специальные согласующиеся интерфейсные СБИС. Каждый из ПЭ хорошо программируется и представляет собой универсальный МП, имеющий внешние коммуникационные каналы – линки связи, с помощью которых МП соединяются в сети. Через линии связи осуществляется передача между программными модулями, выполняемыми на разных МП. Другой способ объединения процессоров в сеть предполагает использование общей памяти (многопортовой). Класс

реализуемых алгоритмов значительно шире, чем в систолических ВА. За счет возможностей программирования ПЭ (транспьютеров и ТП ЦСП), а также адаптеров и коммутаторов, позволяющих подключать ПК и изменять конфигурацию процессорной сети. Для подключения ТП ЦСП к ПК, используется реализуемое на том же кристалле устройство, называемое интерфейса хоста – процессора. Электронный коммутатор, входящий в состав транспьютерной элементной базы, всегда управляется транспьютером, на котором имеется программа, изменяющая соединения транспьютеров в сети.

5) **Универсальная.** Реализуется на процессорах фирм INTEL, AMD, VIA. Среди универсальных ВА имеются одно-процессорные и много-процессорные вычислительные системы. Они предназначены для решения широкого спектра задач, в отличие от остальных названных ВА на СБИС, которые являются узкоспециализированными. Это достигается с помощью системы команд операционных систем и систем программирования CISC и RISC процессоров (CISC – с полным набором и RISC – сокращенным набором команд). Исторически сложилось, что после CISC процессоров появились RISC процессоры. Системы команд CISC процессоров позволяют с помощью одной операции обрабатывать набор данных, представляющих собой вектор, и в качестве результата получать либо скаляр, либо векторное значение.
   Система команд RISC процессоров состоит из трёх групп команд:
   а) позволяющих считать данные из памяти и загрузить в регистр ЦП;
   б) позволяющих разгрузить регистр ЦП и сохранить данные в памяти;
   в) позволяющих выполнить операцию в регистрах ЦП.
   Универсальная ВА на СБИС может выполнять в каждый момент времени одну или несколько команд. Среди них можно выделить простые и сложные процессоры. Простой процессор содержит только один ЦП, а сложный – представляет собой процессорное устройство, состоящий из 2х и более процессоров. Каждый из этих процессоров может выполнять одновременно одну или несколько команд. Практически все современные процессоры имеют на кристалле устройство, которые называется “Конвейер команд”, на вход которого поступает поток команд, одновременно обрабатывается несколько команд, а на выходе получается поток результатов, получаемый в результате выполнения команд.
6) **Нейровычислительная.** Для реализации нейросетей разработаны специализированные ПЭ, называемые нейрочипами, которые имеют на кристалле определенное количество нейронов, входов и выходов, память весов. Среди нейрочипов встречаются аналоговые, цифровые и гибридные. Наиболее быстродействующие - аналоговые, но у них имеется существенный недостаток, связанный с точностью представления данных. Цифровые нейрочипы имеют высокую точность, но выполняют операции с большей задержкой, чем аналоговые. Производительность нейросети задается для процесса вычисления и обучения. Процесс получения результата характеризуется количеством соединений или передач данных между нейронами в единицу времени, а процесс обучения характеризуется временем, которое необходимо затратить на обучение решения конкретной задачи, что фактически соответствует вычислительной производительности системы, вычисляемой количеством операций в секунду. Нейросети по режиму работы делятся на два вида:
   а) синхронные характеризуются одновременным выполнением во всей сети, либо только операции передачи данных, либо только вычислением;
   б) асинхронная, которая таким свойством не обладает, т.е. в каждый момент времени в сети происходят различные операции - соединение нейронов и выполнение вычислений.
   Кроме нейронов для реализации этой ВА применяют процессоры фирмы INTEL, AMD, медийные процессоры, ЦСП, транспьютеры. Все перечисленные типы процессоров эффективно вычисляют операцию умножения с накоплением $c=axb+c$ которая используется для получения выходного значения нейронов.

Архитектура современных процессоров.

1) **Фон Неймановская ВА.** Конструкция ЭВМ основывается на предположениях Джона фон Неймана, выдвинутых им в 1944 г. при разработке нового компьютера EMAC. Он сформулировал основные принципы работы компьютера:

1. принцип программного управления вычислениями на процессоре;
2. управляющие программы и данные хранятся в оперативной памяти;
3. Обмен данными между памятью и процессором осуществляется через специальное устройство – общую шину (ОШ).

Также были сформулированы и другие принципы, но 3 названных являются фундаментальными и определяют способ управления вычислениями, назначение и объем оперативной памяти, и способ передачи данных между памятью и процессором через специальное коммуникационное устройство – каналы данных. Фон Неймановская ВА также называется архитектурой общей шины, которая состоит из шин адреса, данных и управления. Существует также мультиплексированная шина, в которой шина адреса и шина данных являются совмещённой, а шина управления отдельной. В мультиплексированную ОШ встроено специальное устройство – мультиплексор, выполняющий функции коммутатора для передачи данных по одному проводнику шине адреса и данных. Для фон Неймановской ВА характерна единая оперативная память с единым адресным пространство и одним ЦП.

2) **Гарвардская ВА.** Является общим случаем фон Неймановской ВА, т.к. тоже имеет общую шину. В отличии от классической фон Неймановской ВА, которая используется в таких фирмах как Intel, AMD, ГВА используется в цифровых сигнальных процессорах (ЦСП). Принципы построения ЦСП на основе ГВА:
   1. выполнение всех команд за один рабочий такт;
   2. использование RISC архитектуры процессора;
   3. применение различных ОЗУ, а также раздельных шин адреса, команд и данных для связи изолированных памятей с устройствами ЦП;
   4. введение аппаратного умножителя в устройство ЦП или в процессорное ядро (ПЯ);
   5. введение в систему команд составных команд (умножение с накоплением c=a*b+c), которое также выполняется за один рабочий такт.

   Умножение с накоплением может быть выполнено с помощью большего количества команд (умножение, сложение, загрузка данных из памяти в регистры и из регистров в память), каждая из которых выполняется за один такт. Использование команды умножения с накоплением повышает производительность и сокращает объем памяти, использующейся программой.
3) **Транспьютерная ВА.**
4) **ВА длинного командного слова.**
5) **Другие, в частности производные от перечисленных ВА.**

Рост производительности компьютеров пока подчиняется феноменологическому *закону Г.Мура*, который гласит, что производительность компьютеров увеличивается в два раза каждые 18 месяцев. В XX веке этот закон был применим и к быстродействию (тактовой частоте) микропроцессоров. Сегодня возможности увеличения тактовой частоты исчерпаны, но рост производительности продолжается за счет увеличения числа процессоров в **суперкомпьютерах** и серверах, а также построения многоядерных персональных компьютеров. При этом справедливость закона Мура сохраняется в отношении степени интеграции СБИС.

В табл. 1 приведены сведения о лидерах мирового списка Top500 наиболее производительных компьютеров мира. Этот список публикуется два раза в год, официальная страница Top500 находится по адресу http://www.top500.org/. Данные табл. 1 подтверждают рост производительности по показательному закону. В ноябрьских списках 2004-2007 годов первое место занимал суперкомпьютер BlueGene/L (рис. 1), включающий более 16 тысяч процессорных модулей (рис. 2), каждый из которых состоит из четырех двухпроцессорных СБИС.

Таблица 1

Год	Компьютер	Максимальная производительность, Tflop/s	Число процессоров
1999	Intel ASCI Red	2,4	9632
2000	IBM ASCI White	4,9	8192

2001	IBM ASCI White	7,3	8192
2002	NEC Earth Simulator	35,9	5120
2003	NEC Earth Simulator	35,9	5120
2004	IBM BlueGene/L	70,7	32760
2005	IBM BlueGene/L	136,8	65536
2006	IBM BlueGene/L	281	131000
2007	IBM BlueGene/L	478.2	212992
2008	IBM Roadrunner	1025	122400
2009	Cray XT5-HE	1756	224162
2010	Tianhe-1A	2570	186368
2011	SPARC64 VIIIfx 2.0GHz	8162	548352

Тема: «Параллельные вычисления»

В настоящее время практически нет такой сферы человеческой деятельности, где бы в той или иной форме не использовалось численное моделирование. Особенно это относится к областям науки и современных технологий. Поиски новых знаний, решений все более усложняющихся технологических задач привели к пониманию того, что чисто эмпирический путь решения сложных проблем давно себя исчерпал. Эффективное использование накопленного многими поколениями запаса знаний должно быть направлено на построение математических моделей, достаточно адекватно отражающих суть изучаемых явлений или объектов. Как правило, математическая модель представляет собой некоторую совокупность дифференциальных, интегральных, алгебраических или каких-то иных соотношений, определенных в области, так или иначе связанных с предметом исследований. Для реализации этих моделей нужно построить алгоритм решения задачи, описывающей эту модель, что в подавляющем большинстве случаев возможно лишь с помощью численных методов. Чаще всего здесь используется принцип дискретизации изучаемого объекта вместе с окружающей его средой, когда они разбиваются на отдельные элементы, на которые переносятся дискретные (численные) аналоги связей, описываемых математической моделью. В результате получаются системы уравнений с очень большим числом неизвестных. Понятно, что чем больше число таких дискретных элементов, тем более точным получается результат, а потому уровень дискретизации определяется только возможностью компьютера решить возникающую систему уравнений за разумное время. Так, при расчетах обтекания летательного аппарата сложной конфигурации общее число элементов, на которые разбивается расчетная область, составляет порядка 6^{10}.

Основные принципы распараллеливания численных алгоритмов.

Общая методология следующая. Как на последовательном, так и на параллельном компьютере решение задачи находится в результате выполнения множества простых операций, имеющих обычно аргументы двух типов – входные и выходные, причем входными являются, как правило, результаты выполнения других операций. Понятно, что любая операция – потребитель аргументов – не может начать выполняться раньше, чем закончится выполнение всех операций – поставщиков для нее аргументов. Тем самым на множестве всех операций алгоритма разработчик программы явно или неявно устанавливает частичный порядок. Для любых двух операций порядок определяет одну из возможностей: либо указывает, какая из операций должна выполняться раньше, либо констатирует, что обе операции могут выполняться независимо друг от друга. При этом оказывается, что при одном и том же частичном порядке общий временной порядок всего множества операций может быть различным, но любой из них дает один и тот же

результат. Поэтому сохранение частичного порядка, заданного программой, и есть то условие, выполнение которого гарантирует однозначность результата. При этом в рамках одного и того же частичного порядка возможен выбор любой реализации.

Пусть программа описывает какой-то алгоритм. Построим ориентированный граф, вершинами которого будут множества всех операций алгоритма. Выберем любую пару вершин (u, v). Пусть, согласно частичному порядку, операция, соответствующая вершине u, должна поставлять аргументы операции, соответствующей вершине v. Тогда проведем дугу из вершины u в вершину v. Если соответствующие операции могут выполняться независимо друг от друга, дугу проводить не будем. Если аргументом операции являются начальные данные или выходные данные, то эти вершины не будут иметь соответственно входящих и исходящих дуг. Построенный таким образом граф есть граф информационной зависимости реализации алгоритма при фиксированных входных данных или просто – граф алгоритма. Такой граф есть ориентированный ациклический мультиграф. Его ацикличность следует из того, что в любых программах реализуются только явные вычисления (даже в рекурсиях) и никакая величина не может определяться через саму себя. Граф действительно является мультиграфом, поскольку две вершины могут быть связаны несколькими дугами. Это будет иметь место тогда, когда в качестве разных аргументов одной и той же операции используется одна и та же величина.

Ообозначим граф стандартным образом G = (V, E), где V – множество вершин, E – множество дуг. Выберем в графе вершины, не имеющие предшествующих, и пометим их индексом 1. Удалим из графа помеченные вершины и инцидентные им дуги. Оставшийся граф также является ацикличным. Выберем в нем вершины, не имеющие предшествующих, и пометим их индексом 9 2 и т.д. до тех пор, пока не исчерпаем весь граф. Полученная форма графа называется **канонической параллельной формой**. Группа вершин, имеющих одинаковый индекс, называется ярусом, а число вершин в такой группе – его шириной. Число ярусов в параллельной форме называется высотой параллельной формы, а максимальная ширина ярусов – ее шириной.

Конечно, это идеальное представление алгоритма, предполагающее, что все операции одного яруса начинаются и заканчиваются одновременно. Кроме неодновременности завершения операций не учитывается время, затрачиваемое на передачу аргументов, т.е. на пересылки и многое другое. Однако, приведение алгоритма к канонической параллельной форме позволяет оценить запас параллелизма в нем, что позволяет понять, как такой алгоритм лучше всего реализовать на компьютере с конкретной параллельной архитектурой. Конечно, представить сколько-нибудь сложный алгоритм, содержащий сотни и тысячи простых операций в параллельной форме не представляется возможным, однако ничто не мешает в вершинах графа размещать не отдельные операторы, а целые блоки программы, с небольшим числом входных и выходных аргументов, имеющие свою внутреннюю структуру. Тем самым можно с помощью параллельной формы реализовывать и стратегию крупноблочного распараллеливания.

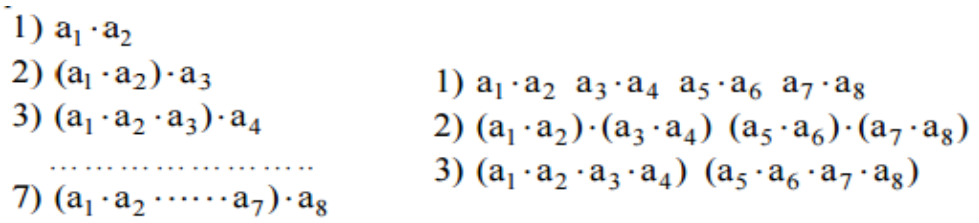

Рисунок 4: Стратегии алгоритмов.

Рассмотрим простой пример. Пусть нужно перемножить 8 чисел a_1, a_2, … a_8. Обычная схема процесса последовательного умножения представлена на рис. 4 слева. В этом случае высота алгоритма равна 7, ширина равна 1. Если вычислительная система имеет более одного процессора, то при данной схеме умножения на всех шагах все они будут простаивать, кроме одного. Изменим теперь алгоритм умножения, называемый процессом сдваивания, см. рис. 4 справа. Графы параллельной формы этих алгоритмов приведены на рис. 5.

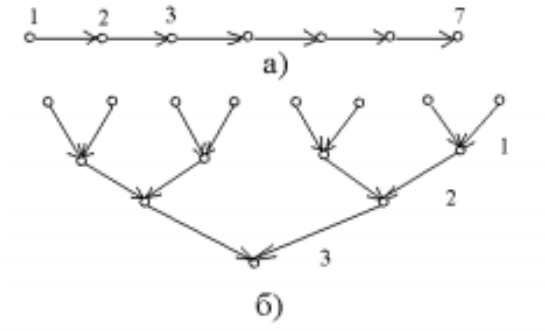

Рисунок 5: Графы алгоритмов.

Тема: «Оптимизация вычислительных алгоритмов»

Постановка вопроса об оптимизации имеет следующее основание. При выборе метода решения задачи исследователь ориентируется на некоторые ее свойства и выбирает алгоритм решения в зависимости от них, причем алгоритм будет применимым и при решении других задач, обладающих этими свойствами. Поэтому при теоретическом исследовании алгоритмов вводят в рассмотрение некоторый класс задач *P*, обладающих определенными свойствами. При выборе метода решения исследователь располагает некоторым множеством *M* методов решения. При применении метода к решению задачи решение будет получено с некоторой погрешностью. Величина

$$E\ (P,\ m) = \sup_{p \in P} |\varepsilon\ (p,\ m)\ | \tag{2}$$

называется погрешностью метода на классе задач *P*, а величина

$$E\ (P,\ M) = \inf_{m \in M} E\ (P,\ m) \tag{3}$$

- оптимальной на классе оценкой погрешности методов из множества *M*. Если существует метод такой, что

$$E\ (P,\ m_0) = E\ (P,\ M), \tag{4}$$

то этот метод называется оптимальным. Такая схема рассмотрения проблемы оптимизации вычислительных алгоритмов, восходящая к А. Н. Колмогорову, изучает множество задач вычисления интеграла

$$I\ (f) = \int_0^1 f\ (x)\ dx \tag{5}$$

при условии $|f^{(n)}| \leqslant A$, M - множество всевозможных квадратур ,

$$I\ (f) \approx \sum_{j=1}^{N} C_j f\ (x_j); \tag{6}$$

каждая квадратура задается совокупностью $2N$ чисел C_j и x_j. Задача о минимальном количестве информации, необходимой для запоминания функции из данного класса с требуемой точностью, также может быть уложена в эту схему. Рассматривается и более сложная постановка проблемы, где трудоемкость реализации алгоритма в определенном смысле увязывается с объемом используемой памяти. Оптимальные алгоритмы построены для незначительного числа типов задач. Однако для большого числа вычислительных задач построены методы, по своим асимптотическим характеристикам близкие к оптимальным.

Исследование характеристик оптимальных на классах вычислительных алгоритмов решения задач складывается из двух частей: построение конкретных методов решения с возможно лучшими характеристиками и получение оценок снизу характеристик вычислительных алгоритмов. По существу первая часть вопроса является основной проблемой теории численных методов и в большинстве случаев рассматривается независимо от проблемы оптимизации. Получение оценок снизу обычно сводится к оценке снизу е-энтропии или поперечников соответствующих пространств; иногда оно проводится независимо, но с использованием техники, аналогичной технике получения указанных оценок.

Вычислительные алгоритмы условно делят на пассивные и активные. В первом случае алгоритм решения задачи не зависит от получаемой в ходе решения задачи информации, а во втором - зависит. В частности, при вычислении интеграла используемая информация о функции есть обычно информация о ее значениях в N точках. В случае пассивного алгоритма интеграл вычисляется по формуле

$$I(f) \approx \sum_{j=1}^{N} C_j f(P_j), \tag{7}$$

где веса C_j и P_j из области W определения f задаются заранее. Активные алгоритмы вычисления интеграла укладываются в следующую схему: задается точка $P_1 \in \Omega$. и функции

$$\begin{gathered} Q_q = \Phi_q(Q_1, \ldots, Q_{q-1}; y_1, \ldots, y_{0-1}), \quad q = 2, \ldots, N, \\ S_N(Q_1, \ldots, Q_N; y_1, \ldots, y_N), \end{gathered} \tag{8}$$

где y_j, S_N –числа, $Q_j \in \Omega$. Затем последовательно вычисляются $f(P_1)$, $P_2 = \Phi_2(P_1; f(P_1))$, $f(P_2)$, $P_3 = \Phi_3(P_1, P_2; f(P_1), f(P_2))$ и так далее до $f(P_N)$, и полагается

$$I(f) \approx S_N(P_1, ..., P_N; f(P_1), ..., f(P_N). \tag{9}$$

В случае выпуклых классов подынтегральных функций, центрально симметричных относительно функции , $f \equiv 0$ оптимальная оценка на классе пассивных алгоритмов совпадает с оптимальной оценкой на классе активных алгоритмов

Пример. Алгоритм Кули-Тьюки быстрого преобразования Фурье.

1.1 Общее описание алгоритма

Простой алгоритм Кули-Тьюки - один из вариантов **быстрого преобразования Фрье** для *комплексных* векторов с размерностью, равной степени двойки, без использования специфичных приёмов, использующихся для степеней четвёрки, восьмёрки и др. Заключается в последовательном применении метода быстрого преобразования Фурье и сведении преобразования к последовательности преобразований Фурье размерности 2 и выполнения умножений на т.н. поворотные множители. Несмотря на то, что проигрывает алгоритмам Кули-Тьюки, разлагающим степени двойки на степени 4, 8 и др. и использующим их специфику, весьма распространён, что связано с самой простой из алгоритмов БПФ записью программной реализации.

1.2 Математическое описание алгоритма

1.2.1 Рекурсивное описание

Вектор записывается по строкам по 2 элемента в каждой. После этого над каждой строкой выполняется преобразование Фурье порядка 2, получившиеся элементы умножаются на поворотные множители $\exp(2\pi i(m-1)(j-1)/n)\exp(2\pi i(m-1)(j-1)/n)$, где m - номер строки, j - номер столбца. После чего выполняется БПФ порядка $n/2$ над каждым из столбцов. Поскольку для 1-го столбца поворотные множители равны 1, то реально умножение на них не выполняется, а умножения на поворотные множители элементов второго столбца соединяются с преобразованием Фурье порядка 2. Эта комбинация, называемая "бабочкой" в среде специалистов по БПФ, и является основной операцией в простом алгоритме Кули-Тьюки. "Бабочка" состоит из вычисления суммы двух комплексных чисел, а также из вычисления их разности с последующим умножением на комплексное число. Всего на каждом шаге выполняется $n/2$ "бабочек", а шагов - $l-1$. Последний, l-й шаг вычисляет только суммы и разности.

1.2.2 Тригонометрические функции

Несмотря на то, что в здесь используются множители $\exp(2\pi i(m-1)(j-1)/n)\exp(2\pi i(m-1)(j-1)/n)$, нецелесообразно вычислять их в процессе выполнения алгоритма Кули-Тьюки, поскольку вычисления косинусов и синусов (в мнимой экспоненте) тогда составили бы основную долю вычислений алгоритма. Поэтому обычно (как и в других версиях БПФ) поворотные множители вычисляются заранее и хранятся в специальном массиве.

1.3 Вычислительное ядро алгоритма

Вычислительное ядро алгоритма составляют "бабочки", состоящие из вычисления суммы двух комплексных чисел, а также из вычисления их разности с последующим умножением на комплексное число. Всего их $(1/2)n\log_2 n$ штук, при этом в n/2 из них умножение не выполняется.

1.4 Последовательная сложность алгоритма

Если считать только главные члены выражений для последовательной сложности алгоритма, то простой алгоритм Кули-Тьюки может быть выполнен за $n\log_2 n$ операций комплексного сложения и $(1/2)n\log_2 n$ операций комплексного умножения. Таким образом, алгоритм Кули-Тьюки может быть отнесён к *линейно-логарифмическому* классу по последовательной сложности.

1.5 Информационный граф

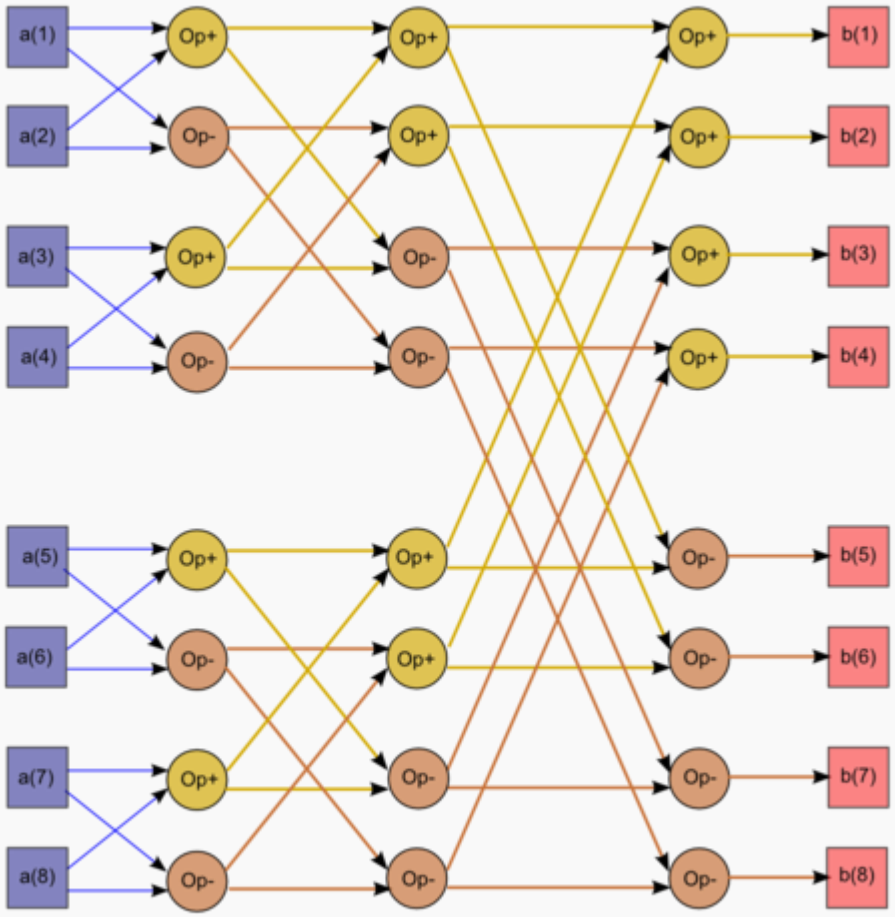

Рисунок 6: Простой алгоритм Кули-Тьюки для n=8. Op+ - операция сложения двух комплексных чисел. Op- - операция вычитания двух комплексных чисел и умножения результата вычитания на комплексное число (поворотный множитель). В последнем столбце операций умножение не производится. Привязка вершин выполнена по оси абсцисс - к параметру внешнего цикла, по оси ординат - к обрабатываемым элементам массива

Как видно из рисунка, этот граф не является линейным ни по размерам, ни по формулам для дуг графа. По размерам он линейно-логарифмический, а формулы дуг имеют экспоненциальные компоненты. В элементарной "бабочке" на i-м шаге каждый раз участвует пара элементов массива, у которых запись их номеров, уменьшенных на единицу, в двоичной системе различается только в i-1-м бите.

1.6 Ресурс параллелизма алгоритма

Если считать только главные члены выражений, то простой алгоритм Кули-Тьюки имеет критический путь, состоящий из $n\log_2 n$ операций комплексного сложения/вычитания и $n\log_2 n$ операций комплексного умножения. Таким образом, простой алгоритм Кули-Тьюки может быть отнесён к *логарифмическому* классу по параллельной сложности. По ширине сложность алгоритма *линейна*.