Теоретический материал по векторно-конвейерным системам (специальность ВМ, IV курс)

1. История создания векторно-конвейерных систем

Сегодня параллелизмом в архитектуре компьютеров уже мало кого удивишь. Все современные микропроцессоры используют тот или иной вид параллельной обработки, реализованный в миниатюрных рамках одного кристалла. Вместе с тем сами эти идеи появились очень давно. Изначально они внедрялись в самых передовых, а потому единичных, компьютерах своего времени. Затем после должной отработки технологии и удешевления производства они начинали использоваться в компьютерах среднего класса, и, наконец, сегодня всё это в полном объеме воплощается в рабочих станциях и ПК.

Для того чтобы убедиться, что все основные нововведения в архитектуре современных вычислительных систем (ВС) используются еще со времен, когда ни микропроцессоров, ни понятия суперкомпьютеров еще не было, совершим маленький экскурс в историю, что позволит иметь более полное представление об интересующих нас ВКС.

Проект, начатый в 1956 году и завершившийся в 1962 году, представил миру компьютер ATLAS. Данный проект по праву считается значительной вехой в истории развития вычислительной техники. Для нас этот компьютер интересен, прежде всего, тем, что в нем впервые начали использовать конвейерный принцип обработки команд. Цикл обработки команды разбили на четыре ступени: выборка команды, вычисление адреса операнда, выборка операнда и выполнение операции. Это позволяло, в благоприятной ситуации, сократить среднее время выполнения команды с 6 до 1,6 мкс и увеличить производительность компьютера до 200 тысяч вещественных операций в секунду.

К числу ЭВМ, в которых широкое применение нашел конвейер команд, относится одна из лучших отечественных машин БЭСМ – 6. Эта ЭВМ занимает особое место в истории развития отечественной вычислительной техники. БЭСМ - 6, разработанная под руководством академика С. А. Лебедева в 1966 г., была в течение многих лет самой быстродействующей в стране благодаря целому ряду интересных решений, в том числе и конвейеру команд. Последний обеспечивался использованием восьми независимых модулей ОЗУ, работающих в режиме чередования адресов, и большого числа быстрых регистров, предназначенных также и для буферизации командной информации. Это позволило получить на БЭСМ – 6 производительность 1 млн. операций в секунду. Новые принципы, заложенные в её архитектуру, структурную организацию и математическое обеспечение, в значительной степени повлияло на создаваемые позднее вычислительные комплексы следующих поколений.

В 1969 году Control Data Corporation выпустила компьютер CDC – 7600, центральный процессор которого содержит восемь независимых конвейерных функциональных устройств (КФУ) – одновременное использование и параллелизма, и конвейерности. В CDC – 7600 реализована двухуровневая память: вычислительная секция в обычном режиме работает со “сверхоперативной” памятью (64К 60 – разрядных слов), куда при необходимости подкачиваются данные из основной памяти (512К 60 – разрядных слов). Время такта CDC – 7600 равно 27,5 нс, средняя производительность 10 – 15 млн. операций в секунду.

Идея включения векторных операций в систему команд компьютеров давно привлекала исследователей. Во-первых, использование векторных операций позволяет намного проще записать многие вычислительные фрагменты в терминах машинных команд. Если нужно сложить два вектора C = A + B, то отпадает необходимость в таких вспомогательных командах, как увеличение индекса, проверка условия выхода из цикла,, перехода на начало тела цикла и многих других – достаточно воспользоваться соответствующей векторной командой. Код становится компактней, а эффективность соответствующих фрагментов, как правило, намного выше традиционных вариантов. Во-вторых, и это не менее важно, многие алгоритмы можно выразить в терминах векторных операций. И, наконец, в-третьих, подобные операции позволяют добиться максимального эффекта от использования конвейерных устройств.

Примеров успешных проектов ВС с использованием векторных команд немало. Одним из первых примеров явилась отечественная разработка компьютера Стрела, имеющего в системе команд ”групповые операции”. Более поздние варианты, появившиеся в конце 60-х

годов прошлого века, это ВС ASC производства компании Texas Instruments и STAR 100 компании Control Data Corporation. Однако “классикой жанра”, несомненно, является линия векторно-конвейерных суперкомпьютеров Cray.

В 1972 г. С. Крэй, вице-президент и генеральный конструктор многих машин, покидает компанию Control Data Corporation и основывает новую компанию Cray Research, которая в 1976 г. выпускает свой первый векторно-конвейерный компьютер Cray-1. Отличительными особенностями архитектуры данного компьютера являются векторные команды, независимые КФУ и развитая регистровая структура. Конвейерность и независимость функциональных устройств определили высокую теоретическую производительность компьютера (две операции за такт), векторные команды упростили путь к высокой производительности на практике (эффективная загрузка конвейеров), а работа функциональных устройств с регистрами сделала высокую производительность возможной даже при небольшом числе операций (исключительно малая латентность при запуске векторных команд).

Общие характеристики Cray-1: время такта 12,5 нс, 12 КФУ, причем все функциональные устройства могут работать одновременно и независимо друг от друга, оперативная память до 1 Мслова, пиковая производительность компьютера 160 млн. операций в секунду.

С появлением компьютера Cray-1 началась история ВКС. Архитектура оказалась настолько удачной, что компьютер дал начало целому семейству машин, а его название стало нарицательным для обозначения сверхмощной вычислительной техники.

С середины 90-х годов прошлого века векторные компьютеры стали заметно сдавать позиции, уступая свои места массивно-параллельным компьютерам с распределенной памятью. Это и понятно. Соотношение цена/производительность для уникальных систем не может конкурировать с продукцией массового производства, а именно этот параметр для многих является решающим. Поговаривали даже о закате векторного направления в целом. Однако в марте 2002 года корпорация NEC объявила о завершении основных работ по созданию параллельной ВС “the Earth Simulator”, основу которой составляют векторные процессоры. Система состоит из 640 вычислительных узлов, в каждом узле по 8 векторных процессоров, что дает 5120 процессоров во всей системе в целом. Поскольку пиковая производительность одного процессора составляет 8 Гфлопс, то для всего компьютера эта величина превышает 40 Тфлопс. Производительность на тесте Linpack составила 35,6 Тфлопс, т. е. 89% от пика – прекрасный показатель! Так что говорить о не перспективности данного направления в развитии вычислительной техники явно рано.

2. Векторно-конвейерные системы: принципы, структура, основные характеристики, система команд и программирование для ВКС

Эффективность использования различных вариантов организации ВПВС связана со структурами вычислительных алгоритмов.

Алгоритм – это описание вычислительного процесса в виде последовательности более простых вычислительных конструкций. Алгоритм включает спецификации этих вычислительных конструкций и отношений следования между ними. Алгоритмические структуры отличаются друг от друга содержанием простых вычислительных конструкций и типом отношения следования.

Для одновременного выполнения команд, не являющихся зависимыми, существуют две дополнительные возможности: использование нескольких процессоров (по числу команд) и применение одного процессора конвейерного типа. Реализация этих возможностей и заключает в себе принцип параллельной обработки данных, формируя тем самым два способа параллельной обработки данных: собственно параллелизм и конвейерность. Конвейеризация обработки – это общий метод повышения пропускной способности систем, выполняющих повторяющиеся операции. Основа этого подхода состоит в том, что система может быть разделена на ступени обрабатывающих устройств, позволяющих начинать исполнение новой команды прежде, чем будет завершена предыдущая.

2.1. Принципы векторно-конвейерной обработки

Принцип векторной обработки основан на существовании значительного класса задач использующих операции над векторами. Алгоритмы этих задач в соответствии с терминологией Флинна относятся к классу ОКМД (одиночный поток команд, множественный поток данных). В самом широком смысле векторизация – это преобразование операций, выполняемых по ходу процесса решения задачи, из скалярной традиционной формы в векторную.

Наиболее общим типом векторизации является *синтаксическая векторизация*, т.е. такой способ преобразования последовательности команд, при котором не учитывается смысловое значение команд и данных, а учитывается только их форма. Проблема данного типа векторизации состоит в том, что не всегда можно найти такое формальное описание, позволяющее превратить некоторую программно – информационную структуру в векторную.

Для избежания данной ситуации используют другой подход называемый *семантической векторизацией*. Он подразумевает интерпретацию содержащихся в программах структур данных и подстановку векторных конструкций, выполняемых с учетом смыслового содержания задачи, а не её формы.

В широком смысле существуют два препятствия на пути векторизации:

- структура данных может и не быть вектором, что происходит по разным причинам, начиная от постоянного присутствия «небольших» дыр в структурах данных до сильно разреженных структур.
- рекурсивные вычисления, т.к. ключевое условие реализации векторной обработки состоит в том, чтобы элементы, участвующие в одних и тех же операциях, не взаимодействовали между собой при исполнении этих операций.

Таким образом, с точки зрения архитектуры вычислительных систем векторная обработка представляет собой совершенно особый эволюционный этап в развитии параллельных систем и вычислений и соответственно требует оборудования, обладающего большими логическими возможностями.

Основа принципа конвейерной обработки состоит в том, что система может быть разделена на ступени обрабатывающих устройств, позволяющих начинать исполнение новой команды прежде, чем будет завершена предыдущая.

Предположим, что нам нужно найти сумму двух векторов, состоящих из 100 вещественных чисел каждый. В нашем распоряжении есть устройство, которое выполняет суммирование пары чисел за пять тактов работы компьютера. Устройство сделано так, что на всё время выполнения данной операции оно блокируется, и никакой другой полезной работы выполнять не может, т.е. является последовательным. В таких условиях вся операция будет выполнена за 500 тактов.

С другой стороны известно, что сложение каждой пары чисел выполняется в виде последовательности микроопераций, таких как сравнение порядков, сложение мантисс, нормализация и т.п. И что примечательно: в процессе обработки каждой пары входных данных микрооперации задействуются только один раз и всегда в одном и том же порядке: одна за другой. Это означает, что если первая микрооперация выполнила свою работу и передала результаты второй, то для обработки текущей пары она больше не понадобится, и значит, что она вполне может начинать обработку следующей ждущей на входе устройства пары аргументов.

Сконструируем устройство следующим образом. Каждую микрооперацию выделим в отдельную часть устройства и расположим их в порядке выполнения. В первый момент времени входные данные поступают на обработку первой частью. После выполнения первой микрооперации первая часть передает результаты своей работы второй части, а сама берет новую пару. Кода входные аргументы пройдут через все этапы обработки, на выходе устройства появится результат выполнения операции.

Такой способ вычислений и носит название *конвейерной обработки*. Каждая часть устройства называется *ступенью конвейера*, а общее число ступеней – *длиной конвейера*. *Время выполнения одной операции* конвейерным устройством равно сумме времен срабатывания всех ступеней конвейера.

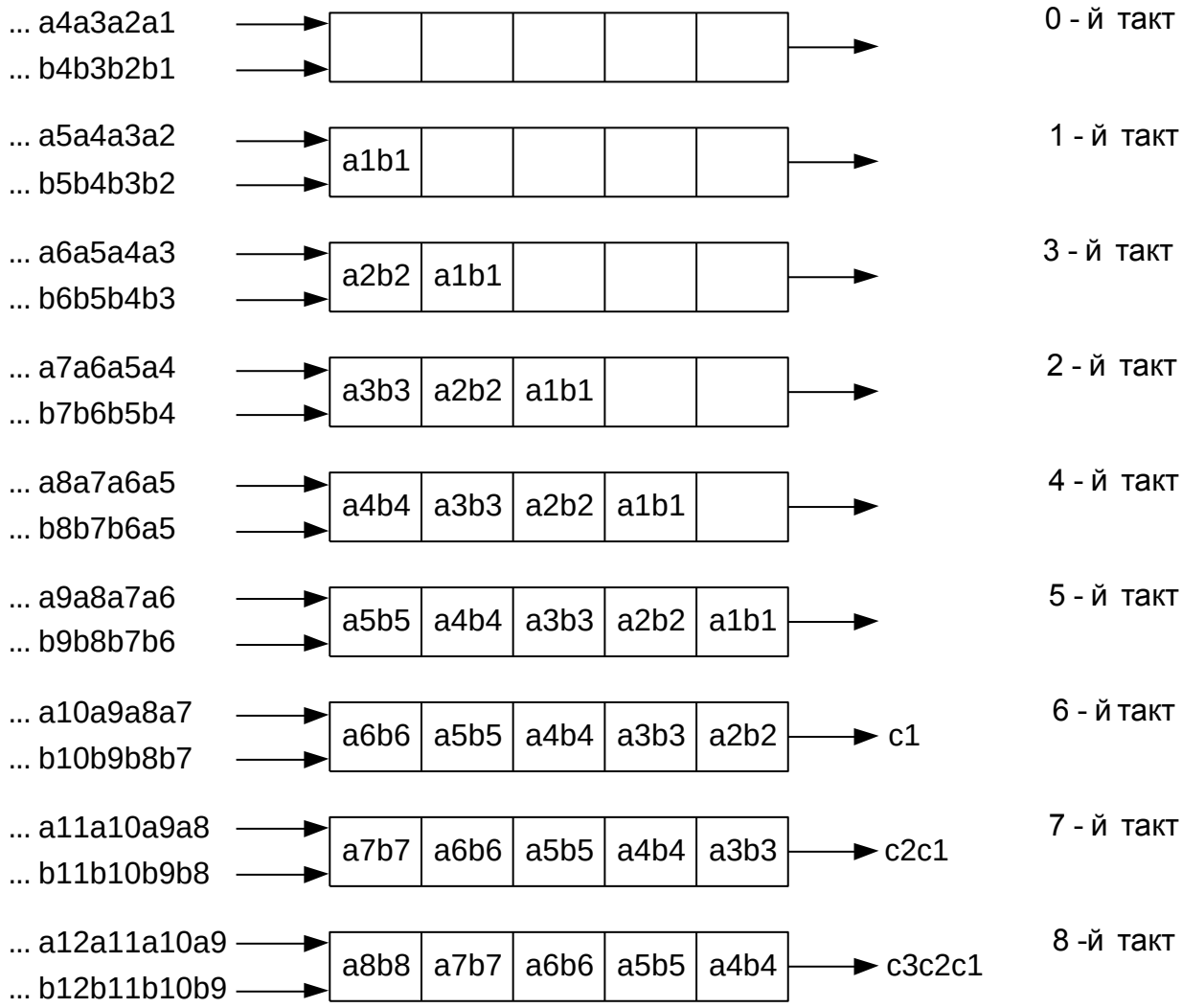

Рисунок 1 - Суммирование векторов C = A + B с помощью конвейерного устройства. Каждая из пяти ступеней конвейера срабатывает за один такт.

Предположим, что для выполнения операции сложения вещественных чисел спроектировано конвейерное функциональное устройство (КФУ), состоящее из пяти ступеней, срабатывающих за один такт каждая. Это значит, что одна операция сложения двух чисел выполнится за пять тактов, т. е. за столько же времени, за сколько такая же операция выполнялась на последовательном устройстве в предыдущем случае.

Теперь рассмотрим процесс сложения двух массивов (рисунок 1). Как и в последовательном случае через пять тактов будет получен результат сложения первой пары чисел. Но при обработке на КФУ одновременно с первой парой прошли частичную обработку и другие элементы. Это позволяет каждый последующий такт на выходе КФУ получать сумму очередных элементов. На выполнение всей операции потребуется 104 такта, вместо 500 тактов при использовании последовательного устройства, что дает выигрыш во времени почти в пять раз.

В общем случае, если конвейерное устройство содержит n ступеней, а каждая ступень срабатывает за одну единицу времени, то время обработки 1 независимых операций этим устройством составит $l + n - 1$ единиц. Если это же устройство использовать в монопольном режиме (как последовательное), то время обработки будет равно $l*n$. В результате для больших значений 1 получили ускорение почти в n раз за счет использования принципа конвейерной обработки.

Таким образом, одним из главных преимуществ векторной обработки является то, что она очень эффективно осуществляется в конвейерном режиме.

2.2. Структура ВКС

Основной тип организации ВКС изображен на рисунке 7. В такой системе имеется один (или небольшое количество) векторный процессор, в котором обработка производится по принципу конвейера, скалярный процессор, процессор обработки команд, а также память для хранения программ и данных. В ряде систем такого типа используется широкий набор регистров для хранения векторов, что позволяет уменьшить число обращений к памяти. Память делится на главную память и память второго уровня, что обеспечивает необходимую емкость и пропускную способность при приемлемой скорости. Векторные системы имеют в своем составе также скалярный процессор, исполняющий скалярные операции, причем, для того чтобы обеспечить наиболее полную загрузку конвейерного процессора, скалярные операции выполняются параллельно с векторными.

Рассмотрим компоненты векторного процессора, также изображенные на рисунке 2. Такой процессор выполняет векторные команды путем засылки элементов векторов в конвейер с интервалом, равным длительности прохождения одной стадии обработки. Результаты операций в виде потока данных пересылаются в память.

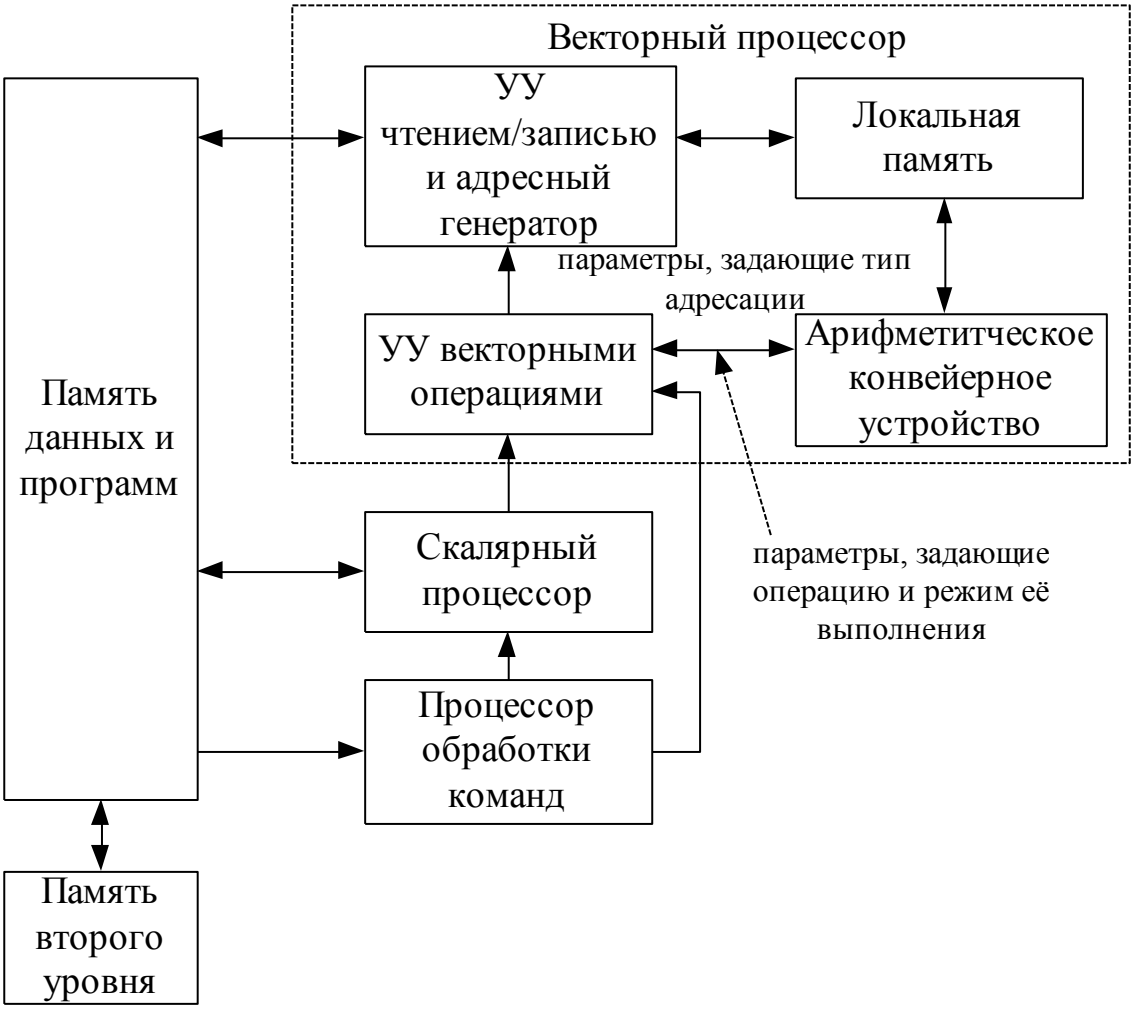

Рисунок 2 - Структура векторно - конвейерной вычислительной системы

Основным компонентом векторного процессора является **арифметическое конвейерное устройство** (АКУ). Условием достижения высокой производительности арифметических (функциональных) устройств является их организация по конвейерному принципу.

По степени функциональности выделяют следующие АКУ:

- специализированные, характеризуются тем, что имеют более простую конструкцию и могут быть оптимально приспособлены для выполнения своей единственной операции.
- многофункциональные, отличаются большей гибкостью и более выгодны в стоимостном отношении. Должны обладать способностью к перенастройке, которая может быть

а) статической, т.е. конвейер заранее настраивается на выполнение определенной операции в рамках конкретной задачи и при необходимости может быть на короткое время, но заблаговременно перенастроен на выполнение другой операции.

б) динамической, что позволит конвейеру обрабатывать поток данных, которым могут соответствовать различные операции, следовательно, такие устройства имеют гораздо более сложную конструкцию и схему управления, вследствие чего их применение нецелесообразно из-за высокой стоимости.

Конвейерное функциональное устройство (КФУ) строится из простых элементарных функциональных (арифметических) устройств. В состав КФУ входят элементарные ФУ, реализующие простые операции. Например, в случае КФУ выполняющего операцию сложения чисел с ПЗ, соответствующие элементарные ФУ последовательно выполняют такие операции, как сравнение порядков, сдвиг мантиссы, сложение мантисс и т. п. Таким образом, после прохождения всех элементарных ФУ в конвейере операция оказывается выполненной. Элементарные ФУ называются *ступенями конвейера*, число ступеней в конвейере – *длиной* конвейера. Рассмотренный пример реализуется на КФУ построенном как линейная цепочка простых элементарных ФУ. КФУ такой конструкции относят к линейному типу конвейерных устройств. В линейных конвейерах данные могут следовать строго по порядку ступеней: от ступени i к ступени i + 1, как показано на рисунке 3а).

В нелинейных КФУ допускаются как опережающие, так и обратные связи (рисунок 3б)). Линейные КФУ отличаются простотой конструкции и управления, однако, нелинейные конвейеры позволяют исполнять, например, рекурсивные операции или вычислять скалярное произведение.

Число КФУ, типы операций, которые они должны выполнять, число ступеней конвейеров и время прохождения каждой ступени – это те параметры, выбор которых составляет основную задачу конструирования вычислительных систем.

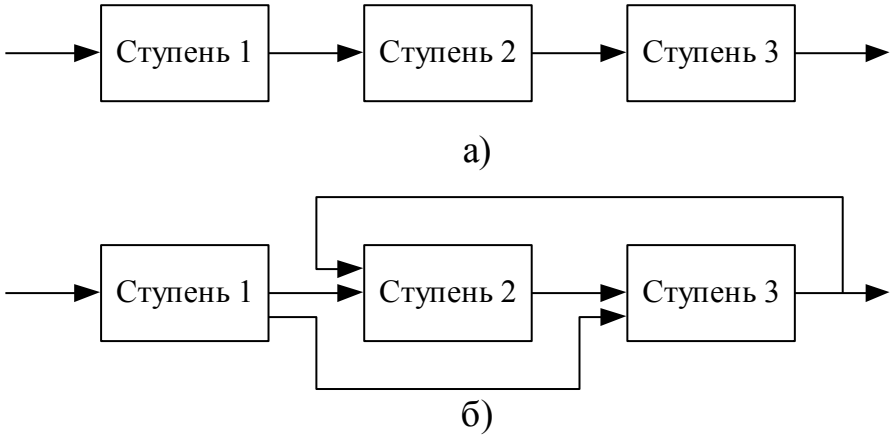

Рисунок 3 - Типы конвейерных устройств: а) линейный конвейер, б) нелинейный конвейер

В состав векторного процессора также входит **локальная память**, которая играет роль буфера между основной памятью и процессором. Для связи локальной памяти с основной памятью и процессором требуется несколько информационных шин:

- одна или две для операндов, поступающих из основной памяти;
- одна для записи результатов в память;
- две для подачи операндов в конвейерное обрабатывающее устройство (или устройства если их несколько);
- одна для приема результата из конвейера.

Для обеспечения максимальной пропускной способности некоторые из названных шин должны работать параллельно, по этой причине локальная память разбивается на ряд блоков, которые соединены с главной памятью и конвейерными устройствами через коммутационную сеть (рисунок 4). Один из возможных путей организации взаимодействия, который приводит к упрощению коммутационной сети, состоит в том, чтобы иметь для каждого конвейерного устройства отдельный буфер. Но тогда возникают ограничения при использовании результатов одного конвейера в качестве операндов другого конвейера. Кроме того, локальная память используется не оптимально, поскольку не является общим ресурсом для всех конвейеров.

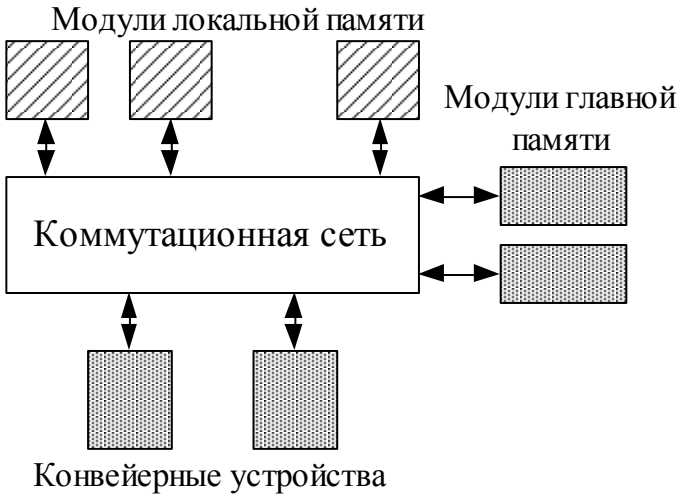

Рисунок 4 - Схема подключения локальной памяти

Размеры и структура локальной памяти существенным и весьма сложным образом влияют на производительность процессора.

Управление доступом к данным, находящимся в памяти, осуществляется **адресным генератором**, а параметры векторных операций устанавливаются **векторным УУ**.

Адресное устройство генерирует адреса главной памяти, по которым осуществляется доступ к элементам векторов. В простейшем и наиболее распространенном случае адреса элементов образуют естественную последовательность. В некоторых векторных системах организация памяти такова, что запись и считывание осуществляется целыми блоками, состоящими из последовательно расположенных слоев (такой блок иногда называют «суперсловом»). В этом случае адресное устройство должно генерировать единственный адрес. Применяются также схемы адресации, в которых элементы располагаются на некотором постоянном расстоянии друг от друга. В этих случаях генерация адресов производится с использованием операций сложения, выделения подматриц, состоящей в формировании комбинации последовательных и равноотстоящих адресов и произвольного доступа.

2.3. Уровни конвейеризации

При выполнении векторная команда проходит определенные этапы, схематично представленные на рисунке 5.

Рисунок 5 - Обработка векторной команды

Одним из путей уменьшения времени выполнения векторных команд является применение многоуровневой конвейеризации на всех этапах, которые проходит команда в процессе выполнения.

На *первом уровне* исполнение команды делится на две стадии: подготовка и исполнение (рисунок 6). При наличии такого конвейера одна команда исполняется, а другая готовится к исполнению.

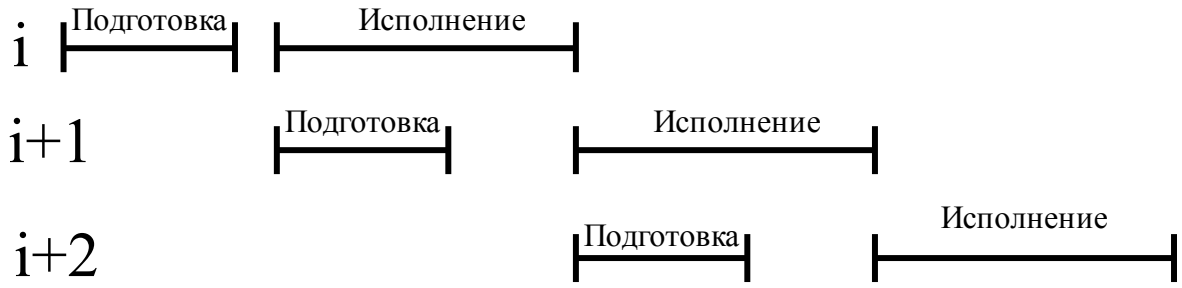

Рисунок 6 - Конвейеризация первого уровня

Поскольку обе эти стадии, в свою очередь, делятся на более мелкие операции, устройства подготовки и исполнения также могут быть конвейеризованы, образуя *второй уровень* конвейеризации (рисунок 7).

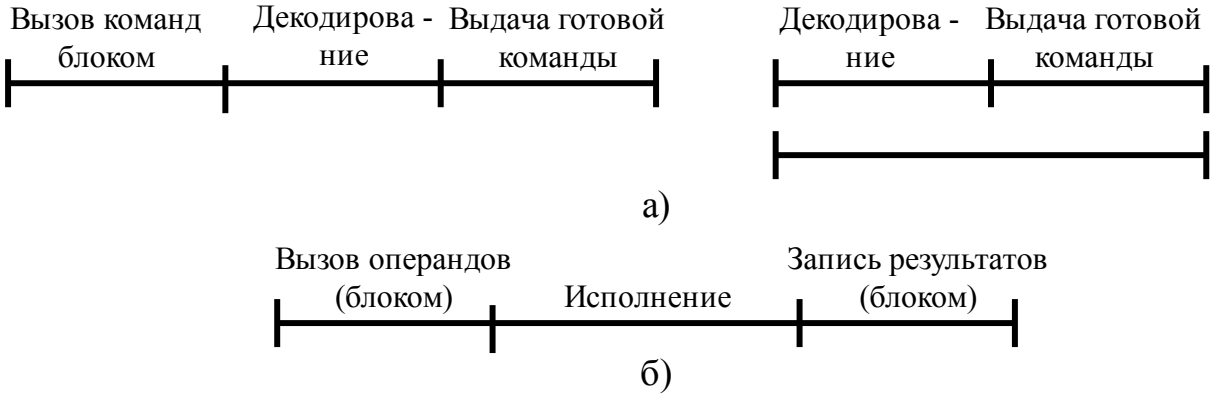

Рисунок 7 - Конвейеризация второго уровня: а) устройство подготовки команд, б) устройство исполнения команд

Обычно устройство подготовки включает следующий конвейер: вызов, декодирование и засылку команды в исполнительное устройство, которое также может быть конвейеризировано: вызов операндов, исполнение операции и засылка результатов.

Некоторые из перечисленных операций являются достаточно сложными и требуют дальнейшего разбиения. Так образуется третий уровень конвейеризации. Особенно важна конвейеризация доступа к памяти и арифметических операций (рисунок 8).

Рисунок 8 - Конвейеризация третьего уровня: арифметическое устройство

Чтобы показать, какой выигрыш в производительности дает ВКС благодаря применению конвейеризации команд обработки векторов, рассмотрим пример выполнения операции сложения двух векторов. На рисунке 9а) приведены скалярная программа и временная диаграмма её исполнения на обычной ЭВМ, в которой реализован опережающий просмотр в сочетании с конвейерным исполнением операций. На рисунке 9б) изображены соответствующая векторная программа и временная диаграмма.

Рисунок 9 - Вычисление суммы двух векторов на скалярном (а) и векторном (б) процессорах

По приведенной на рисунке 9б) диаграмме можно сделать вывод, что для ускорения процесса выполнения векторной команды сложения применена конвейеризация второго уровня. Как видно конвейеризации подвергнуто только исполнительное устройство: вызов операндов (операции чтения), исполнение операции (сложение) и засылка результатов (запись).

2.4. Основные характеристики ВКС

Пропускная способность конвейера определяется временем прохождения самой медленной стадии. Следовательно, при разработке конструкции необходимо обеспечить равенство скоростей обработки на всех стадиях. В сложных конвейерных устройствах данное условие нереализуемо. В таких случаях для повышения пропускной способности применяют два метода:

- межстадийной буферизации, предназначенный для тех машин, в которых времена задержек на каждой ступени различны. Благодаря данному методу пропускная способность конвейера определяется не максимальной, а средней задержкой (рисунок 10). Буферизация оказывается полезной во многих местах конвейера, но особенно – в интерфейсе памяти, так как, во-первых, её быстродействие невелико, а, во-вторых, запросы к памяти могут носить разный характер (вызов команды, формирование адреса, вызов операндов, запись результата).

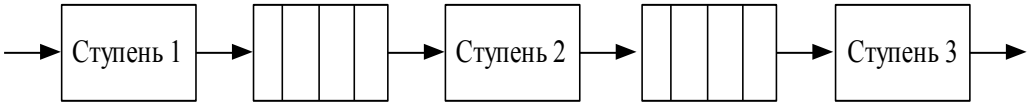

Рисунок 10 - Конвейер с буферизацией между ступенями

- параллельное дублирование медленных стадий (к которым относятся, например, стадия обращения к памяти и стадия выполнения операций в устройстве обработки). Так, память обычно выполняется в виде набора модулей, а для разных арифметических операций используются отдельные устройства (универсальные или специализированные).

Основным показателем производительности такой идеализированной векторной машины является максимальная пропускная способность (*Rmax*). Она равна пропускной способности самой медленной ступени, которая в свою очередь, определяется как результат деления числа однотипных устройств этой ступени на величину времени прохождения

$$R\max = \left(\frac{M}{Tc}\right)\min . \qquad (1)$$

Главная задача при создании процессора состоит в получении максимально высокой пропускной способности. Для этого необходимо обеспечить минимальные времена прохождения всех стадий конвейера, а там, где это возможно, применить параллельно – последовательную структуру.

Существует ряд факторов, приводящих к тому, что при выполнении реальных программ производительность векторных ЭВМ оказывается значительно ниже максимально возможной. Рассмотрим наиболее важные **факторы, снижающие пропускную способность** и выделим показатели, которые могут служить мерой снижения производительности.

1) Скалярная обработка

Невозможно построить реальную прикладную программу, состоящую только из векторных операций. Значительная часть вычислений остается скалярной, причем некоторые скалярные команды служат для подготовки векторных команд и управления их прохождением. Поскольку векторные устройства не обеспечивают эффективного исполнения этих команд, в состав оборудования входит скалярный процессор. Всякий раз, когда исполнение какой-либо скалярной операции не полностью перекрывается по времени векторными вычислениями, происходит снижение производительности.

Величина снижения может быть получена исходя из доли скалярных операций (*f*) и отношения максимальной пропускной способности в векторном режиме к пропускной способности в скалярном режиме (*r*). В итоге получим *коэффициент снижения пропускной способности*

$$d = \frac{R\max}{R} = f*r + (1-f), \qquad (2)$$

где *Rmax* - максимальная пропускная способность (полная загрузка всех конвейеров), *R* - реальная пропускная способность, *d* - отношение максимальной пропускной способности в векторном режиме к пропускной способности в скалярном режиме.

Как видно из рисунка 11, снижение производительности может быть весьма значительным.

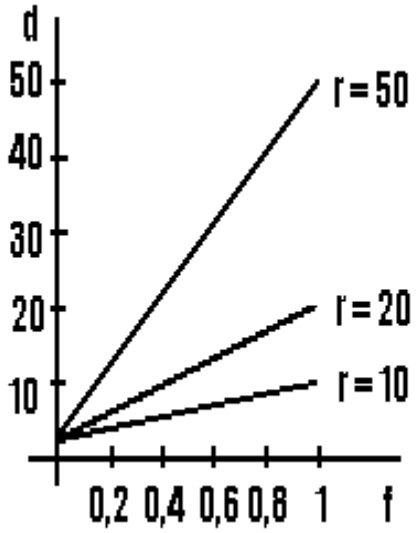

Рисунок 11 - Коэффициент снижения производительности

Так, например, при $f = 0{,}1$ и $r = 50$ реальная производительность меньше максимальной в 5,9 раза. Конкретные значения f и r варьируются в зависимости от системы и прикладной области.

2) Длина вектора

Максимальная производительность конвейера достигается только при бесконечной длине вектора.

Как и прежде, будем считать, что КФУ состоит из l ступеней, срабатывающих за один такт. *Тактом конвейера* называется временной интервал между моментом загрузки конвейера очередным элементом данных. На величину такта основное влияние оказывает элементная база, наличие и глубина промежуточных буферов, сложность операций на обрабатывающих устройствах.

Два вектора из n элементов можно сложить одной векторной командой, либо выполнить подряд n скалярных команд сложения элементов этих векторов. Если n скалярных команд одна за другой выполняются на КФУ, то они будут обработаны за $l + n - 1$ тактов. Вместе с тем, на практике выдержать ритм, определяемый этой формулой, довольно сложно: каждый такт нужно обеспечивать новые входные данные, сохранять результат, проверять необходимость повторной итерации, быть может увеличивать значения индексов и т.д. и т.п. В итоге, необходимость выполнения множества вспомогательных операций приводит к тому, что становится невозможным каждый такт подавать входные данные на вход КФУ. В конвейере появляются “пузыри”, а значит, на выходе уже не каждый такт появляются результаты, и эффективность работы устройства снижается.

При использовании векторных команд в формуле добавляется еще одно слагаемое: $ti + l + n - 1$, где ti - это время, необходимое для инициализации векторной команды (время настройки конвейера). Исполнение векторной команды не требует практически никаких вспомогательных действий, кроме момента её инициализации или поддержки сегментации. Однако присутствие $t_н$ определяет тот факт, что для небольших значений n соответствующие векторные команды выгоднее исполнять не в векторном, а в обычном скалярном режиме.

Поскольку ни ti, ни n не зависят от значения l, то *с увеличением длины входных векторов эффективность конвейерной обработки возрастает*. Если под эффективностью обработки понимать реальную производительность P КФУ, равную отношению числа выполненных операций l к времени их выполнения t, то зависимость производительности от длины входных векторов определяется следующим соотношением

$$P = \frac{l}{t} = \frac{l}{(l + n\text{-}1)\cdot tc + ti} = \frac{1}{tc + \frac{ti}{l} + (n\text{-}1)\cdot \frac{tc}{l}}, \qquad (3)$$

где tc – это время прохождения самой медленной стадии конвейера.

На рисунке 12 показан примерный вид этой зависимости. С увеличением числа входных данных реальная производительность КФУ все больше и больше приближается к его пиковой производительности. Однако у этого факта есть и исключительно важная обратная сторона –

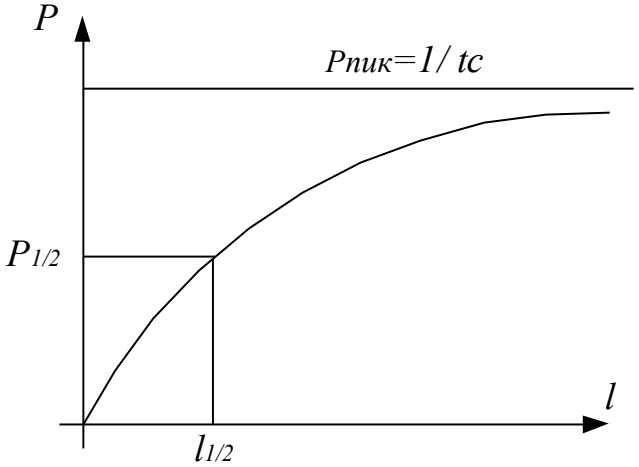

Рисунок 12 - Зависимость производительности КФУ от длины входного набора данных

пиковая производительность любого конвейера недостижима на практике.

Одним из основных показателей, характеризующих эффективность ВКС и определяющих область их использования является $l_{1/2}$ – из основных это длина вектора при которой производительность либо конвейера, либо всей ВКС в целом достигает половины максимальной.

3) Стартовое время конвейера

Максимальная производительность конвейерной машины достигается при обработке длинных векторов, поскольку только в этом случае доля времени, затрачиваемого на начальном этапе (загрузка параметров, реконфигурация, ожидание первого результата), оказывается максимальной. Рассмотрим формулу 3, введя понятие стартового времени конвейера t_{start}. Известно, что первый элемент вектора результата будет получен с задержкой определяемой формулой 4. Эта величина характеризует *время загрузки конвейера* t_z и находится как произведение числа ступеней конвейера *n* на время прохождения самой медленной стадии конвейера *tc*. В реальных системах время, затрачиваемое на запуск конвейера (*стартовое время*) можно представить как сумму времени настройки и времени загрузки конвейера (формула 5). В результате подстановок получаем формулу 6.

$$t_z = n \cdot t_c \tag{4}$$

$$t_{srart} = t_i + t_z = t_i + n \cdot t_c \tag{5}$$

$$P = \frac{l}{t_i + (n + l - 1) \cdot t_c} = \frac{l}{t_i + n \cdot t_c + l \cdot t_c - t_c} = \frac{l}{t_{srart} + (l - 1) \cdot t_c} \tag{6}$$

Наличие стартового времени приводит к тому, что векторная обработка оказывается эффективней скалярной лишь начиная с некоторого порога длины векторов. Соответствующей количественной мерой может служить *коэффициент ускорения вычислений* как функция длины вектора, а именно

$$S(l) = \frac{T_{scalar}}{T_{vector}} = \frac{l \cdot t_{scalar}}{t_{start} + (1 - l) \cdot T_c} \approx \frac{r}{\frac{t_{start}}{l \cdot T_c} + 1}, \tag{7}$$

где *r* - введенное ранее отношение максимальных пропускных способностей при векторном и скалярных режимах соответственно. Отсюда найдем приближенную пороговую длину вектора

$$l \min \approx \frac{t_{start}}{r \cdot T_c} = \frac{t_{start}}{t_{scalar}}, \tag{8}$$

где t_{scalar} - время выполнения скалярных операций.

4) Зависимости по данным и по управлению

а) Зависимости по данным

Условием работы процессора в режиме пиковой производительности является одновременное исполнение нескольких команд, обеспечивающее постоянную занятость всех функциональных устройств. При наличии зависимостей по данным между командами, т.е. когда одна команда не может исполняться до завершения другой, это условие нарушается. Такого рода зависимость может существовать между любыми двумя командами. Примером двух зависимых векторных команд может служить следующий фрагмент:

$$Y \leftarrow \sum Ai \cdot Bi$$

$$Z \leftarrow C + Y$$

В этом случае снижение производительности может быть предотвращено с помощью образования цепочек операций, когда исполнение данной векторной команды начинается

сразу, как только образуются компоненты участвующих в ней векторных операндов (рисунок 13). Данный метод носит название метода «сцепления». Он может применяться в большом числе практических случаев, однако не всегда.

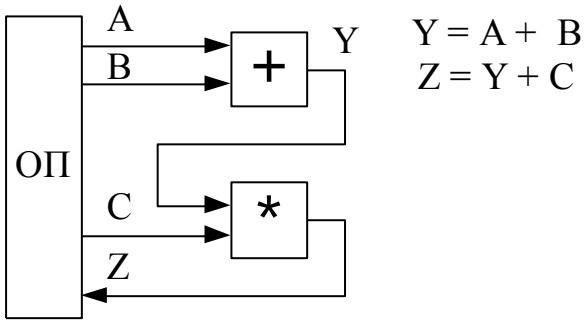

Рисунок 13 - Образование цепочки операций

Например, его нельзя использовать в следующем примере:

$$C \leftarrow A \times B$$

$$Z \leftarrow Y \times C$$

поскольку для начала исполнения второй команды необходимо вычислить скалярный результат первой.

б) Зависимости по управлению

Данный тип зависимости существует двумя командами в ситуациях до тех пор, пока не завершится исполнение предыдущей команды, и остается неизвестным, какая из двух окажется следующей. Такая зависимость обычно порождается условными переходами и является одним из наиболее существенных отрицательно влияющих на производительность факторов в скалярных процессорах с опережающим просмотром. В векторных ЭВМ, где часть операторов цикла заменена векторными командами, влияние данного фактора несколько ослаблено, однако остается еще значительным.

При наличии зависимостей данного типа, когда команда, с которой начинается та или иная ветвь, остается неизвестной до тех пор, пока не завершится исполнение предыдущей команды и последовательность не прервется, в потери времени входит полный цикл обработки команды: вызов, декодирование и др. Для уменьшения этих потерь применяют различные конструктивные средства:

- предварительный вызов начальных команд обеих ветвей, являющихся возможными продолжениями данной;
- использование нескольких командных буферов для хранения разных участков программ;
- хранение, задержанных ветвей в виде, позволяющем загрузить процессор сразу после завершения выбора ветви.

Эти методы, разработаны для высокопроизводительных скалярных процессоров.

5) Реализация условных операторов.

Векторная команда как единое целое не может быть выполнена в ситуациях, когда вид операции над элементами вектора зависит от условия, как, например, в следующем фрагменте:

if $X_i < 0$ then Zi←Xi - Yi else Zi←Xi + Yi

Один из возможных выходов состоит в том, чтобы выполнить все операции в скалярном режиме, однако, это привело бы к резкому снижению производительности, Более перспективной альтернативой является использование для управления векторной операцией так называемого вектора режима. Это битовый вектор, содержащий такое же число элементов, как и каждый из векторов в управляемой операции. Управление в данном случае означает то, что операция выполняется только над теми элементами векторов, которым соответствует значение 1 в одноименных позициях вектора режима.

Безусловно, описанный способ лучше, чем скалярный, однако и при нем все еще остаются факторы, снижающие производительность.

6) Ограничения связанные с памятью

а) Локальная память

Для поддержания высокой скорости конвейера арифметических операций операнды должны, интенсивно поступать из памяти. Это, в свою очередь, требует разделения памяти, на модули с возможностью одновременного доступа, а также включения буфером для операндов и результатов. Такой 6уфер сглаживает изменения пропускной способности тракта обмена данными между памятью и арифметическими устройствами, а также позволяет осуществлять выборку из памяти элементов векторов в порядке, отличающемся от того, который должен быть на входе обрабатывающих устройств. Это существенно расширяет диапазон прикладных задач, которые могут эффективно решаться в векторном режиме. Буферная память служит также для хранения результатов, которые в последующих векторных командах выступают в качестве операндов.

По способу организации локальной памяти векторные процессоры делятся на два класса. К первому относятся те, в которых команды исполняются по принципу “регистр-регистр”, а ко второму - по принципу “память-память”.

В первом случае локальная память представляет собой набор векторных регистров, адресуемых в командах, а во втором: локальную память, работающую как - буферное устройство и остающуюся невидимой для программиста. Основные преимущества систем первого класса состоят в следующем:

- команды, в которых указаны короткие адреса регистров, имеют меньшую длину;
- промежуточные результаты естественным образом хранятся на регистрах;
- облегчается процесс контроля зависимостей между командами.

Главный недостаток данного подхода связан с ограниченностью разрядности и числа регистров.

б) Механизм адресации и недостаточно эффективная пропускная способность памяти

Поскольку время доступа к памяти существенно превышает длительность одной стадии обрабатывающего конвейера, а в процессе исполнения программ требуются одновременные обращения к памяти за различными операндами, обеспечение адекватной пропускной способности памяти в векторном режиме, когда элементы векторов-операндов занимают последовательный ряд адресов достигается за счет организации памяти в виде нескольких модулей и применения принципа чередования адресов (интерливинг/расслоение). Это позволяет даже при максимальной пропускной способности осуществлять бесконфликтный доступ к блокам последовательно адресуемых элементов векторов.

в) Конфликты доступа к памяти

К ограничивающим факторам относятся конфликты доступа, возникающие при одновременном появлении нескольких разных - обращений к памяти. Необходимым условием работы системы с максимальной производительностью является своевременное считывание команд из памяти, вызов скалярных и векторных операндов и запись зарабатываемых результатов. Указанные запросы реализуются через равные порты и “мешают” друг другу, что приводит к неполному использованию пропускной способности памяти. Для того чтобы отрицательное влияние этих конфликтов на общую производительность было минимальным, устройство управления доступом к памяти должно соответствующим образом планировать порядок обслуживания запросов.

Например, в системе Cray X-MP память разделена на четыре секции, каждая из которых имеет четыре порта: три обслуживают запросы процессоров и один обслуживает ввод/вывод. Порты A и B предназначены для чтения, а порт C для записи. Возникающие конфликты разрешаются с помощью системы приоритетов, причем последние зависят от вида обращения и времени поступления. Из сказанного ясно, что получить адекватную оценку производительности можно только путем прогонки программ на реальном оборудовании.

г) Объем памяти

При решении многих прикладных задач массивы данных оказываются столь большими, что не могут уместиться в главной памяти. В этих случаях часть данных приходится хранить в памяти второго уровня и осуществлять подкачку и удаление информации в процессе счета. Размер главной памяти, скорость подкачки, способ сегментирования - все эти факторы оказывают значительное влияние на производительность.

7) Ограничения, связанные со специализированными арифметическими устройствами

Максимальная пропускная способность системы достигается только в том случае, когда все арифметические устройства используются полностью. Как было отмечено выше, полной загрузке этих устройств препятствует наличие зависимостей по данным и по управлению. Другим ограничивающим фактором уменьшающим эффективную производительность, является специализированность арифметических устройств, состоящих из устройства сложения, устройства умножения и т.д. В этом случае полное использование арифметического оборудования возможно только на специально подобранной смеси векторных и скалярных операций. Следовательно, реальная производительность зависит от состава операций в конкретной программе.

Ввиду сложного характера взаимодействия между факторами, снижающими производительность, и зависимостью степени влияния этих факторов от конкретных программ оценка производительности в целом должна осуществляться на специально подобранных прикладных задачах. Приближенная оценка может быть получена с помощью контрольных задач, состоящих из набора типовых тестовых программ. Проблема, связанная с такой приближенной оценкой, заключается в трудности определения термина «типовой». Искусственные контрольные задачи, содержащие в нужном соотношении скалярные операции и векторные команды различных типов, могут дать лишь самые грубые оценки. Недостаток данного метода состоит в том, что он не учитывает взаимосвязи между командами, роль которых в формировании реального показателя производительность может быть значительной.

Одними из наиболее широко используемых для оценки векторных машин контрольных задач являются так называемые *Ливерморские циклы*, разработанные в Национальной лаборатории им. Лоуренса в г. Ливерморе (США). Интересно отметить, что для всех типов машин оценки, полученные на разных циклах, сильно отличаются друг от друга. Например, согласно этим оценкам производительность Cray-1 колеблется от 3 до 90 млн. операций с плавающей запятой в секунду. Кроме того, хотя наилучшие и наихудшие с точки зрения производительности циклы практически одни и те же для всех машин, характер зависимости производительности от типа цикла на разных машинах существенно отличается один от другого. Это подтверждает тезис о специфическом влиянии на производительность тех или иных особенностей организации систем. На производительность оказывает влияние также качество используемого компилятора.

2.5. Система команд ВКС. Особенности программирования для ВКС

Главной отличительной чертой ВКС является наличие в их “репертуаре” векторных команд. С помощью этих команд задаются операции над полными векторами. Эти операции заменяют обычные циклы, широко используемые в скалярных вычислениях.

ВКС обычно имеет широкий набор разнообразных команд. Кроме обычных скалярных команд этот набор включает векторные команды и специальные команды, предназначенные для управления векторными операциями.

Типовой набор векторных команд включает:

- арифметические операции;
- операции сравнения;
- команды загрузки и записи в память;
- команды для обработки разреженных векторов.

Скалярные команды используются для формирования адресов элементов и контроля длин векторов.

В некоторых векторных ЭВМ наиболее часто встречающиеся в программах операции, такие, как скалярное произведение, вычисление среднего значения и ряд других, реализованы в виде *векторных макрокоманд*. Это позволяет использовать одну команду вместо целой подпрограммы.

Так же, как и скалярная, векторная команда должна включать:

- код операции;
- адреса и длины векторных операндов и результата;
- статус процессора при записи;
- адрес следующей исполняемой команды.

С помощью кода операции производится выборка одной из заранее составленных таблиц, на основании которой осуществляется управление КФУ и производятся необходимые для начального запуска этого устройства действия. Если операнды (результаты) находятся:

а) в регистрах, то векторная команда задает:

- номер регистра:
- число элементов;
- тип данных:
- еще некоторую информацию, необходимую для организации доступа.

б) в памяти верхнего уровня, тогда в векторной команде должны быть заданы:

- базовый адрес вектора;
- число элементов;
- тип данных в каждом элементе;
- схема хранения вектора в памяти.

Поскольку элементы векторов иногда располагаются по адресам, не образующим непрерывной последовательности, процедура формирования адресов обращений к этим элементам может оказаться весьма сложной.

Векторные операнды хранятся в памяти таким образом, при котором уменьшаются требования к объему памяти и обеспечивается эффективный доступ к различным векторным подструктурам. Схема размещения данных в памяти зависит от требований, предъявляемых к подструктурам, возможностей механизма адресации и организации памяти.

Существуют два класса схем размещения данных в памяти, соответствующих двум типам векторных структур: плотным и разреженным. В первом случае хранятся все элементы векторов, причем последовательно идущие элементы занимают, как правило, последовательно расположенные ячейки памяти. Во втором случае хранятся только ненулевые элементы и вектор соответствующих индексов или заменяющий его битовый вектор.

При работе с плотными массивами используются следующие схемы распределения данных в памяти:

- последовательная, т.е. хранение элементов векторов в последовательно расположенных ячейках (рисунок 14а);
- с регулярной структурой, т.е. столбцовые элементы матриц хранятся по строкам (рисунок 14б);
- в виде подматриц (рисунок 14в);
- в произвольном порядке (эта ситуация возникает, например, при исполнении программ, содержащих зависимости по данным, и при непрямой адресации).

Схема доступа к нерегулярным структурам реализуется с *помощью битовых векторов* и *индексных векторов*. Использование битовых векторов (т.е. таких, элементы которых принимают значения 0 или 1 в зависимости от выполнения или невыполнения какого-либо условия) обеспечивает быстрый и эффективный способ избирательного доступа. Эти векторы генерируются с помощью ряда команд, таких, как сравнение векторов и операции ИЛИ/И над векторами, или с помощью условной установки разрядов. Битовые векторы хранятся в компактной форме, занимая слово или последовательность слов, и используются в качестве управляющих векторов в векторных операциях. Вектор индексов представляет собой целочисленный вектор, элементы которого могут использоваться в качестве полных адресов памяти или смещений, которые добавляются к базовому адресу.

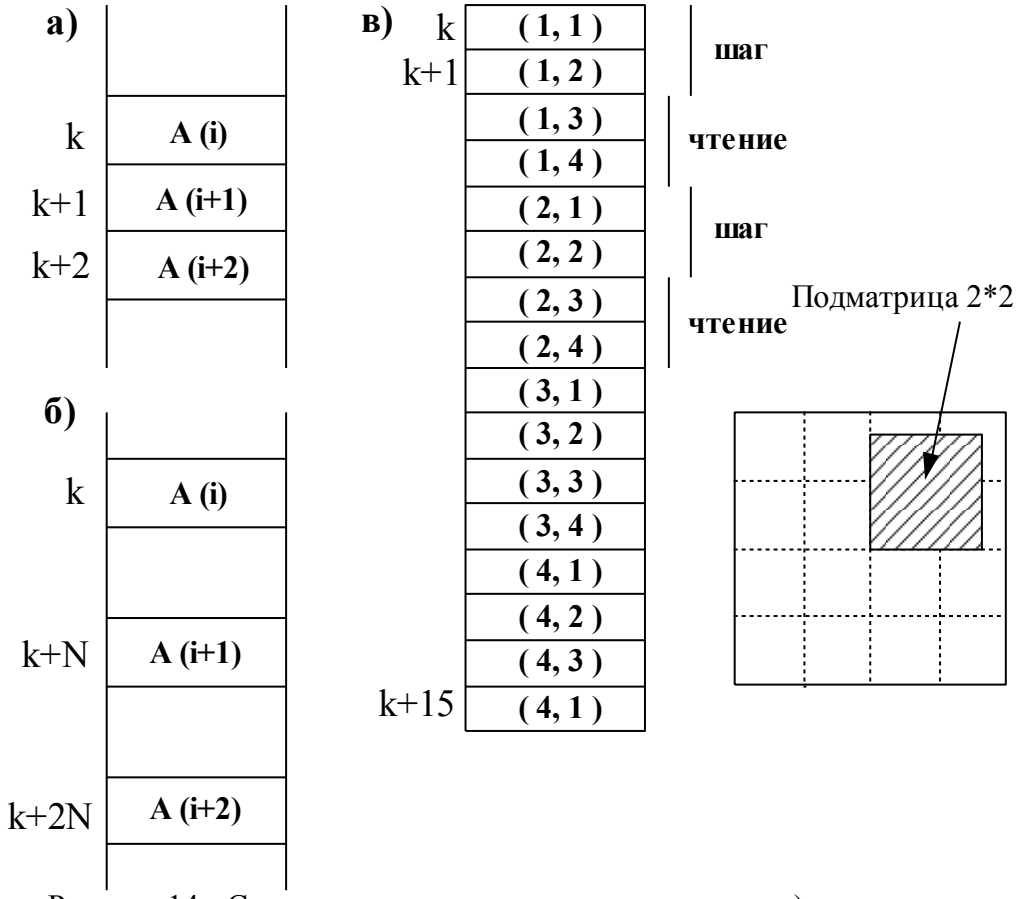

Рисунок 14 - Схемы распределения данных в памяти: а) последовательная, б) с регулярной структурой, в) в виде подматриц

Для векторных команд в большей степени, чем для скалярных, характерны такие понятия, как *состояние* и *обработка особых си*туаций. Поскольку в результате выполнения операции вырабатывается много величин, то с практической точки зрения невозможно и нежелательно держать состояния, т. е. признаки выполнения индивидуальных условий, в составе самой команды. Вместо этого вводится специальная векторная команда, с помощью которой состояния вырабатываются и записываются в явном виде в формате битовых векторов, о назначении и применении которых уже упоминалось. Например, векторная команда сравнения на равенство VCOMPEQ (X, Y, Z) вырабатывает битовый вектор Z следующему правилу:

$$Z(I) = \begin{cases} 1, & \textit{если } X(I) = Y(I) \\ 0, & \textit{если } X(I) \neq Y(I) \end{cases}.$$

Битовый вектор, сформированный в результате операции сравнения, используется для управления векторными командами.

Обработка особых ситуаций (ошибок) в векторных командах затруднительна из-за множественности операций. Обычно при появлении ошибки исполнение векторной команды либо прекращается, либо фиксируется сообщение об ошибке, а выполнение команды продолжается.

Поскольку векторные процессоры работают в соответствии с последовательно-групповой моделью вычислений, очередность команд устанавливается также, как и в обычных процессорах: счетчик команд указывает на следующую команду после получения очередного приращения или, в случае команды перехода – после загрузки в этот счетчик начального адреса новой ветви.

Операции редукции (вычисление скалярного произведения) требуют специальных команд, которые превращают векторы в скаляры. Эти команды называются *векторной редукцией.* В некоторых системах операции данного типа включены в систему команд. Так в системе Cyber 205 имеется ряд векторных макрокоманд для различных видов редукции. В тех ВКС, которые не имеют команд редукции, данная операция осуществляется иными

способами. Например, операция редукции в системе Cray-1 осуществляется с помощью обычных векторных команд и аппаратно реализованного счетчика элементов.

Целью программиста должно быть не получение правильного результата вычислений любой ценой, но получение правильного результата наибыстрейшим, оптимальным способом обязательно.

Если программа предназначена для однократного использования, то лучше написать её как можно проще, не оптимизируя её быстродействие и используемую память, чтобы потратить минимум усилий на тестирование и отладку.

Если программа предназначена для частого использования или время её работы будет гораздо больше времени её написания и отладки, то не следует жалеть труда на оптимизацию её быстродействия. Однако в результате программа, оптимальная для одного типа систем, окажется совсем не оптимальной для машин других типов.

Методы программирования для скалярных, векторных и параллельных систем коренным образом отличаются друг от друга.

Написание программы “с нуля” одинаково сложно (или одинаково просто) для машин любых типов. Этот способ является идеальным для разработки эффективных, высокопроизводительных векторных программ. Начинать надо с изучения специфики программирования для векторных систем, изучения алгоритмов, которые наиболее эффективно реализуются на ЭВМ данных типов. После этого надо проанализировать поставленную задачу и определить возможность применения векторизуемых алгоритмов для решения конкретной задачи. Возможно, что придется переформулировать какие-то части задачи, чтобы они решались с применением векторных или параллельных алгоритмов. Программа, специально написанная для ВКС, даст наибольшее ускорение при её векторизации.

Перенос готовой программы и оптимизация её для исполнения на машине другого типа может потребовать весьма существенных усилий по исправлению стиля написания программы. Опыт показывает, что проще всего адаптировать к исполнению на векторных машинах программы на ФОРТРАНе. Конструкции ФОРТРАНа для организации циклов и для работы с массивами менее разнообразны, чем в Си, а именно циклы по обработке массивов и являются главными объектами оптимизации при векторизации программ.

Для повышения эффективности программ для параллельных систем производят её распараллеливание. Рассмотрим отличия и сходства между распараллеливанием и векторизацией программ, что позволит лучше понять алгоритм векторизации программ для ВКС и способы его применения.

1) Сходство алгоритмов – *параллелизм данных*

Их архитектуры векторных и параллельных компьютеров следует самое важное свойство алгоритмов, которые могут быть эффективно исполнены на таких системах: эти алгоритмы должны включать одинаковые действия над некоторыми наборами данных (массивами), при этом результаты действий над любой частью данных не должны зависеть от результатов действий над другой их частью. Иначе говоря, последовательность вычисления величин не должна играть роль в данном алгоритме. Это есть параллелизм данных.

Простейшим примером такого алгоритма может служить следующий фрагмент программы:

```
real x(100), a(100), b(100)
do  i = 1, 100
    x(i) = real(i)**2 + 3.0
    a(i) = b(i) + x(i)
enddo
```

или в другой интерпретации

```
 real x(100), a(100), b(100)
do  i = 100, 1, -1
    x(i) = real(i)**2 + 3.0
    a(i) = b(i) + x(i)
enddo.
```

Порядок исполнения двух операторов находящихся в теле цикла не может быть изменен, но каждый i – тый элемент массивов a и x в могут быть вычислены независимо от других элементов. Следовательно, приведенная программа удовлетворяет основному требованию к векторизуемым алгоритмам – **для любого значения i вычисления можно проводить независимо.**

Предположим, что у нас имеется ВКС с векторными регистрами длиной 100 элементов. Тогда весь цикл по всем элементам массива записывается как линейная (не циклическая) последовательность команд, а каждая команда оперирует со всеми 100 элементами массива. Выигрыш в увеличении быстродействия программ (по отношению к быстродействию не векторизованных программ на той же ЭВМ) может ожидаться равным числу элементов в массиве.

При реализации данной программы на 100-процессорной параллельной машине, с возможностью доступа каждого процессора ко всем элементам данных, также как и в случае с ВКС, можно получить ускорение в 100 раз по сравнению с однопроцессорной ЭВМ.

2) Различие алгоритмов – *параллелизм действий*

Можно проследить аналогию в векторной и параллельной реализациях предыдущей программы. Каждая машинная команда вызывает действия сразу над 100 числами: в векторной программе явно выполняются операции над всеми элементами регистра, в параллельной программе каждый процессор выполняет более или менее синхронно одинаковые машинные команды, оперирует со своими собственными регистрами, и в результате выполняются действия одновременно со 100 числами.

Справедливо следующее утверждение: **алгоритм, который можно векторизовать, можно и распараллелить. Обратное утверждение не всегда верно.**

Теперь рассмотрим характеристики и преимущества, вносимые при векторизации программ.

1) Предельное быстродействие векторных программ

Для рассмотрения будем использовать ВКС с векторными регистрами. Векторный процессор выполняет математические операции сразу над всеми элементами векторного регистра. Если число элементов регистра равно 128, то операция над всеми 128 числами выполняется в векторном режиме так же быстро, как над одним числом в скалярном режиме. Это и есть теоретический предел повышения быстродействия программ при их векторизации. Однако необходимо учесть, что любая векторная операция требует больше машинных тактов для своего исполнения, чем такая же скалярная операция. С другой стороны, циклическое N-кратное исполнение скалярной команды для обработки массива требует исполнения еще нескольких команд, организующих собственно цикл. В результате исполнения векторной команды может оказаться эффективнее более, чем в 128 раз. Для простоты, будем считать, что предельное повышение эффективности векторных программ равно числу элементов в векторном регистре ЭВМ. Чаще всего это 128 или 256.

В любой программе существует две части – векторизуемая и невекторизуемая.

К невекторизуемой части относят:

- алгоритмы построения последовательностей заданных рекуррентным отношением, их нельзя векторизовать, т. к. каждый последующий элемент зависит от предыдущих и, соответственно, не может быть вычислен ни ранее, ни одновременно с предыдущим;
- ввод/вывод;
- вызов подпрограмм или функций;
- организация циклов;
- разветвленные алгоритмы;
- работа со скалярными величинами.

Это довольно широкий класс подзадач, он гораздо шире класса векторизуемых алгоритмов.

При векторизации программ на самом деле ускоряется выполнение только части программы (большей или меньшей). Поэтому каждую программу можно представить такой упрощенной схемой (рисунок 15), если собрать все векторизуемые и все скалярные части в единые блоки.

Рисунок 15 - Основные части программы

Для начала рассмотрим выполнение программы в полностью скалярном варианте. Обозначим время исполнения каждой из частей программы через *T1 ... T5*. Тогда полное время невекторизуемых частей программы будет равно

$$Tскал = T1 + T3 + T4 + T5,$$

а полное время работы программы

$$T' = Tскал + T2$$

Далее перетранслируем программу в режиме векторизации. Единственная часть программы, которая ускорит свое выполнение, будет вторая. Предположим, что мы достигли ускорения работы этой части в *N* раз. Тогда полной время работы всей программы составит

$$T'' = Tскал + T2/N,$$

а выигрыш в эффективности работы всей программы будет равен

$$P = \frac{T'}{T''} = \frac{Tсска + T2}{Tскал + T2/N},$$ что совсем не равно N.

Предположим, что *Tскал* = 1с, *T2* = 99с, а *N* = 100 (очень хороший показатель для 128-элементных векторных процессоров). Тогда эффективность *P* составит всего (1 + 99)/ (1 + 99/ 100) = 50 или 40% от предельно возможных 128 раз, а при *Tскал* = 2с, *T2* = 98с значение *P* составит всего (2 + 98)/ (2 + 98/ 100) = 34 (27%). Хотя само по себе увеличение быстродействия программы в 50 или даже 30 раз при её векторизации является очень большим (вычисления будут занимать одни сутки вместо одного месяца), но предельно возможное ускорение в 128 (или 256) раз не может быть, достигнуто на векторных ЭВМ. Практика показывает, что хорошим показателем увеличения быстродействия *P* можно считать уже значения 6 – 10, что соответствует времени выполнения скалярной части всего за 10 – 15% от полного времени исполнения программы (*T2* составляет 85 – 90%).

2) Дополнительные затраты на организацию векторных вычислений во время работы программы

Для работы на векторных ЭВМ наиболее удобными являются массивы с длиной, равной длине векторного регистра. Более того чаще всего число элементов вообще не кратно 128. Возникают трудности с обработкой длинных векторов в ВКС, работающих по принципу «регистр - регистр», так как длины векторов, непосредственно участвующих в операциях, фиксированы и определяются размерами векторных регистров. Для работы с векторами произвольной длины существует так называемый режим *векторного цикла*. Например, в системе Cray-1 при работе с векторами, содержащими более 64 элементов, используется векторный цикл, в котором управление осуществляется с помощью скалярных регистров параллельно с выполнением векторных операций, что позволяет исключить потери, связанные с организацией циклического перехода.

3) Ограниченное число векторных регистров

Число векторных регистров в процессоре обычно гораздо меньше, чем число скалярных (8 регистров). Это накладывает ограничения на сложность выражений (т.е. число массивов и скалярных переменных), стоящих в теле векторизуемого цикла. Требуемая перезагрузка регистров может стать фактором, существенно замедляющим программу.

4) Ограничения на используемые операторы в векторизуемых циклах

Существенным ограничением на конструкции, которые могут применяться в векторизуемых циклах, является использование только операторов присваивания и арифметических выражений. Никакие команды перехода (условные ветвления, вызовы подпрограмм и функций, циклические операторы или безусловные переходы) не могут быть использованы в теле векторизуемого цикла.

Из перечисленных запретов существует только два исключения:

- Использование встроенных в транслятор арифметических функций. Большинство таких функций реализуются в библиотеках языка, а некоторые транслируются в последовательности машинных команд. Обычно каждая функция имеет две реализации – скалярную и векторную.
- Использование условного оператора присваивания. Все арифметические операции (в т. ч. и само присваивание) будут выполняться не над всем векторным регистром, а только над теми его элементами, для которых было справедливо вычисленное логическое выражение (маскируемые операции). Команда сравнения устанавливает маску для каждого элемента вектора: “истина”, если элемент вектора меньше 0, и “ложь”, если элемент больше или равен нулю. Команда присваивания не затронет те элементы массива z, для которых маска равна “ложь”. Векторный процессор будет исполнять команды для вычисления арифметического выражения и команду присваивания, даже если не будет ни одного значения маски “истина”. Это важное примечание.

Рассмотрев систему команд ВКС и особенности программирования для ВКС можно сделать следующее заключение: системы команд и стиль программирования для ВКС существенным образом отличаются от привычного для большинства программистов, привыкших создавать последовательные программы. Знания же в области параллельного программирования являются новым направлением и еще редко применяются среди широкого круга пользователей, хотя в архитектуре всех современных процессоров заложены принципы векторно-конвейерной и параллельной обработки.

3 Векторно-конвейерный компьютер Cray C90: архитектура и особенности программирования

Особенностью ВКС являются, во-первых, КФУ и, во-вторых, набор векторных инструкций в системе команд. В отличие от традиционного подхода, векторные команды оперируют целыми массивами независимых данных, что позволяет эффективно загружать

доступные конвейеры. Типичным представителем данного направления является линия векторно-конвейерных компьютеров Cray компании Cray Research.

В качестве объекта детального рассмотрения возьмем компьютер Cray C90, в архитектуре которого есть все характерные особенности компьютеров данного класса. Его последователь Cray T90 имеет такую же структуру, отличаясь лишь некоторыми количественными характеристиками. Описание компьютера будет вестись с той степенью детальности, которая необходима для выделения ключевых особенностей и узких мест архитектуры.

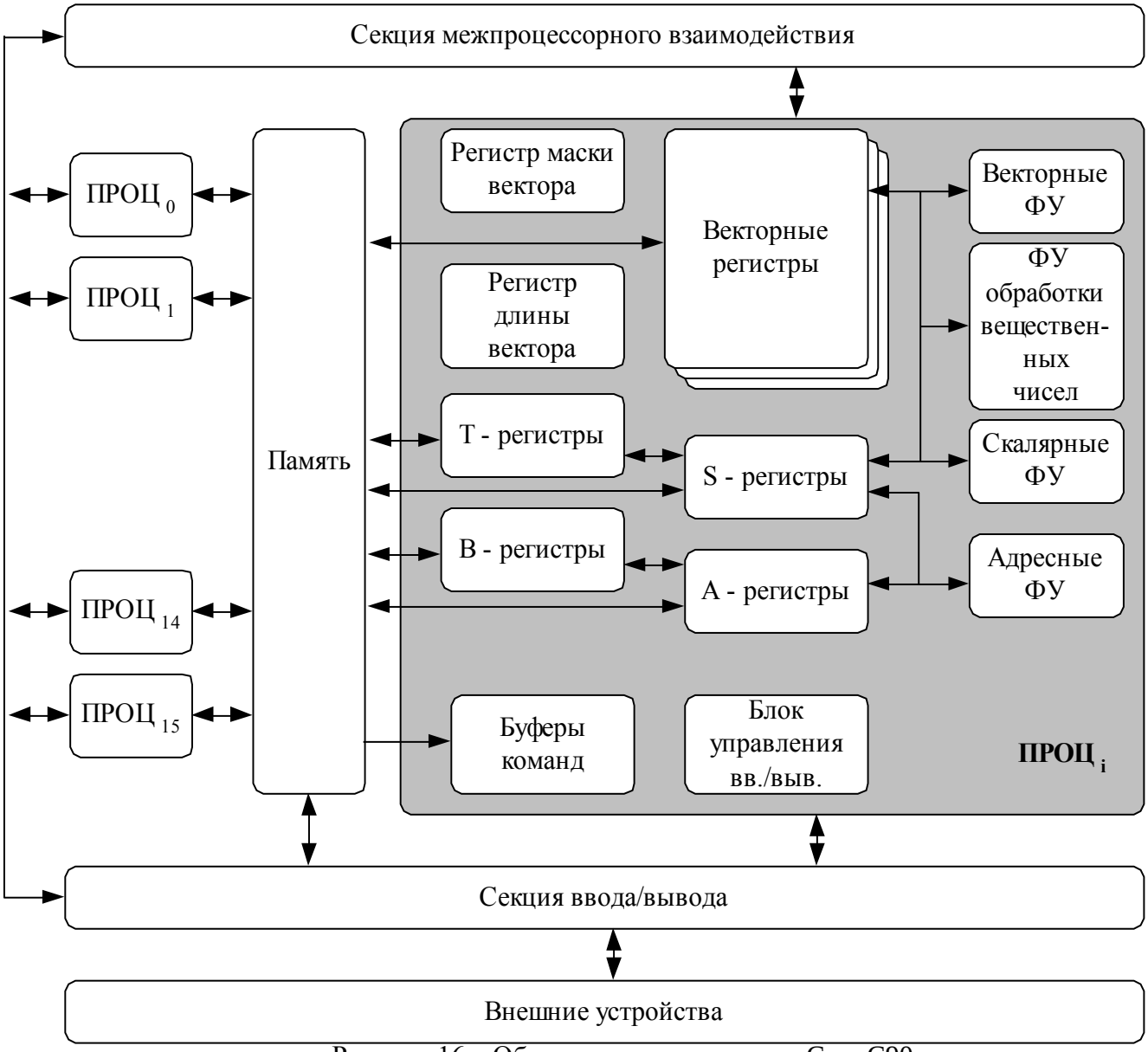

Рисунок 16 – Общая схема компьютера Cray C90

Итак, Cray C90 – это векторно-конвейерный компьютер, появившийся на рынке вычислительной техники в самом начале 90-х годов прошлого века. В максимальной конфигурации Cray C90 содержит 16 процессоров, работающих над общей памятью. Время такта компьютера равно 4.1 нс, что соответствует тактовой частоте почти 250 МГц. На рисунке 16 показана общая схема данного компьютера с более детальным представлением структуры одного процессора, поскольку все процессоры одинаковы.

Все процессоры компьютера Cray C90 не только одинаковы, но и равноправны по отношению ко всем разделяемым ресурсам: памяти, секции ввода/вывода и секции межпроцессорного взаимодействия.

1) Структура оперативной памяти

Оперативная память Cray C90 разделяется всеми процессорами и секцией ввода/вывода. Каждое слово памяти состоит из 80-ти разрядов: 64 разряда для хранения данных и 16

вспомогательных разрядов для коррекции ошибок. Для увеличения скорости выборки данных вся память разделена на множество банков, которые могут работать одновременно.

Каждый процессор имеет доступ к оперативной памяти через четыре порта с пропускной способностью два слова за один такт каждый. Один из портов всегда связан с секцией ввода/вывода, и, по крайней мере, один из портов всегда выделен под операцию записи. Подобная архитектура хорошо подходит для выполнения векторных операций с не более чем с двумя входными векторами.

В максимальной конфигурации реализовано расслоение памяти компьютера на 1024 банка: каждая из 8 секций разделена на 8 подсекций, а каждая подсекция на 16 банков (рисунок 17). Последовательные адреса идут с чередованием по каждому из данных параметров.

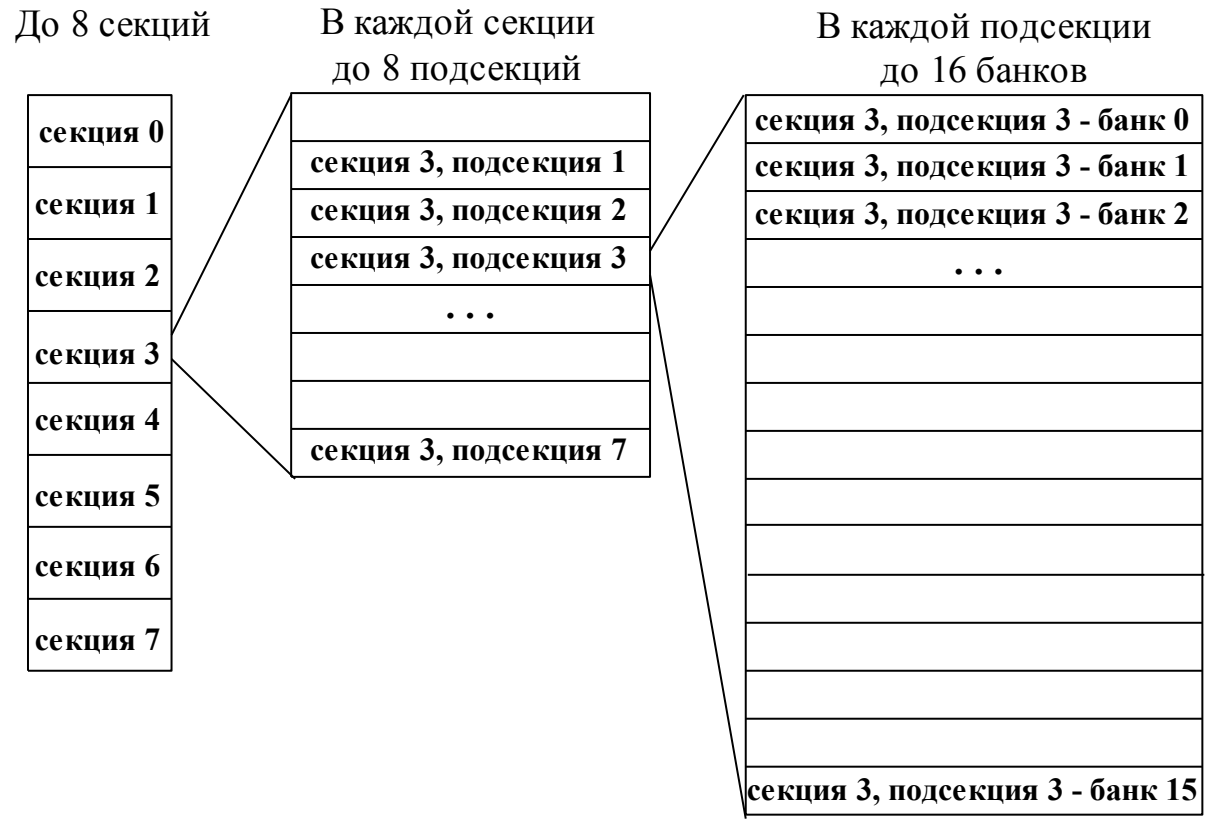

Рисунок 17 - Расслоение памяти компьютера Cray C90

При одновременном обращении к одной и той же секции возникает конфликт, который разрешается за один такт. В этом случае один из запросов продолжает обрабатываться, а другой просто блокируется на 1 такт. Если происходит одновременное обращение к одной и той же подсекции одной секции, время на разрешение конфликта уже может достигать 6 тактов. Ясно, что максимальное число конфликтов будет происходить при постоянном обращении к одной и той же подсекции одной и той же секции. Это заведомо произойдет при выполнении процессором векторной операции над данными, расположенными с шагом, кратным 64.

2) Секция ввода/вывода

Компьютер Cray C90 поддерживает три типа каналов для работы с внешними устройствами, которые различаются скоростью передачи данных:

- Low-speed (LOSP) channels – 6 Мбайт/с;
- High-speed (HISP) channels – 200 Мбайт/с;
- Very high-speed (VHISP) channels – 1800 Мбайт/с.

3) Секция межпроцессорного взаимодействия

Основное назначение данной секции заключается в передаче данных и управляющей информации между процессорами для синхронизации их совместной работы организации взаимодействия друг с другом. Секция межпроцессорного взаимодействия содержит

разделяемые регистры и семафоры, объединенные в одинаковые группы – кластеры. Каждый кластер состоит из восьми 32-разрядных разделяемых адресных регистров (SB), восьми 64-разрядных разделяемых скалярных регистров (ST) и 32 однобитовых семафоров. Число кластеров в системе определяется конфигурацией компьютера.

4) Структура процессора Cray C90

Все процессоры имеют одинаковую вычислительную секцию, состоящую из регистров и функциональных устройств (ФУ). Различные регистры и функциональные устройства могут хранить и обрабатывать три класса данных: адреса, скаляры и вектора.

а) Регистровая структура процессора

Каждый процессор имеет набор основных и набор промежуточных регистров. К *основным регистрам* относятся:

- адресные регистры A;
- скалярные регистры S;
- векторные регистры V.

Промежуточные регистры B и T, играющие роль промежуточного хранилища между памятью и основными регистрами, предусмотрены для регистров A и S соответственно. Все основные регистры связаны с памятью и функциональными устройствами, а регистры A и S имеют промежуточную связь с соответствующими промежуточными регистрами. Промежуточные регистры связаны только с памятью и основными регистрами, непосредственной связи с функциональными устройствами у них нет. Здесь же следует заметить, что и традиционной кэш-памяти в данном компьютере нет.

Адресные регистры. В структуре компьютера предусмотрено 8 адресных регистров в основном наборе A, и 64 регистра в промежуточном наборе B. Адресные регистры предназначены для хранения и вычисления адресов, индексации, указания величины сдвигов, числа итераций циклов и т.д. Все регистры данной группы имеют по 32 разряда.

Скалярные регистры. В основном наборе скалярных регистров S содержится 8 регистров, и еще 64 регистра в промежуточном наборе T. Эти регистры предназначены для хранения аргументов и результатов скалярной арифметики, но могут содержать операнды для векторных команд. Это значит, что скалярные регистры используются для выполнения как скалярных, так и векторных команд. Все скалярные регистры 64-разрядные.

Векторные регистры. Каждый векторный V-регистр может содержать до 128 64-разрядных слов. Всего в процессоре содержится 8 векторных регистров. Промежуточных регистров для набора V нет. Векторные регистры используются только для выполнения векторных команд.

Для поддержки выполнения векторных команд предусмотрено два дополнительных регистра VL и VM. *Регистр длины вектора* VL содержит реальную длину векторов, хранящихся в векторных регистрах и участвующих в векторной операции. Данный регистр содержит 8 разрядов. *Регистр маски вектора* VM состоит из 128 разрядов и позволяет выполнять векторную операцию не над всеми элементами входных векторов. Если разряд маски равен 1, то операция над соответствующими элементами будет выполнена, в противном случае – нет. Данная возможность исключительно полезна при векторизации фрагментов, содержащих условные операторы.

б) Функциональные устройства.

Все функциональные устройства у компьютера Cray C90 являются конвейерными. Число ступеней у них различно, однако каждая ступень каждого устройства всегда срабатывает за один такт. Это, в частности, означает, что при полной загрузке все устройства могут выдавать результат каждый такт. Кроме того, все функциональные устройства независимы и могут работать одновременно друг с другом.

Функциональные устройства данного компьютера делятся на четыре группы:

- адресные;
- скалярные;
- векторные;
- функциональные устройства для выполнения операций над вещественными числами.

Адресные функциональные устройства. Два адресных функциональных устройства предназначены для выполнения сложения/вычитания и умножения 32-разрядных целых чисел.

Скалярные функциональные устройства. Скалярных устройств в процессоре всего четыре. Они используются для целочисленного сложения/вычитания, логических поразрядных операций, для выполнения операций сдвига и нахождения числа нулей до первой единицы в слове. Скалярные устройства оперируют с 64-разрядными данными и предназначены для выполнения только скалярных команд.

Векторные функциональные устройства. Число векторных устройств в зависимости от конфигурации компьютера меняется от пяти до семи. В их число могут входить устройства для целочисленного сложения/вычитания, сдвига, логических поразрядных операций, устройство для нахождения числа нулей до первой единицы в слове, для умножения битовых матриц. Некоторые из перечисленных устройств могут быть продублированы. Все векторные функциональные устройства предназначены для выполнения только векторных команд.

Функциональные устройства для выполнения операций над вещественными числами. Три функциональных устройства для вещественной арифметики работают с 64-разрядными числами, представленными в форме с плавающей запятой. Они предназначены для сложения/вычитания, умножения и нахождения обратной величины числа. Отдельного устройства для выполнения явной операции деления вещественных чисел нет. Все устройства данной группы могут выполнять как векторные, так и скалярные команды.

Выполнение векторных операций в данном компьютере обладает интересной особенностью. Все конвейеры во всех векторных устройствах и устройствах для вещественной арифметики продублированы (рисунок 18). Элементы входных векторов с четными номерами всегда поступают на конвейер 0, а элементы входных векторов с нечетными номерами – на конвейер 1. В начальный момент времени нулевые элементы векторных регистров V_1 и V_2 поступают на первую ступень конвейера 0, и одновременно с этим первые элементы векторных регистров V_1 и V_2 поступают на первую ступень конвейера 1. На следующем такте результат первой ступени перемещается на вторую, а на первую ступень конвейеров 1 и 0 поступают вторые и третьи элементы векторных регистров V_1 и V_2. Аналогично раскладываются результаты в регистре V_3: с конвейера 0 они записываются в элементы с четными номерами, а с конвейера 1 – с нечетными. В результате функциональное устройство при максимальной загрузке на каждом такте выдает уже не один результат, а два.

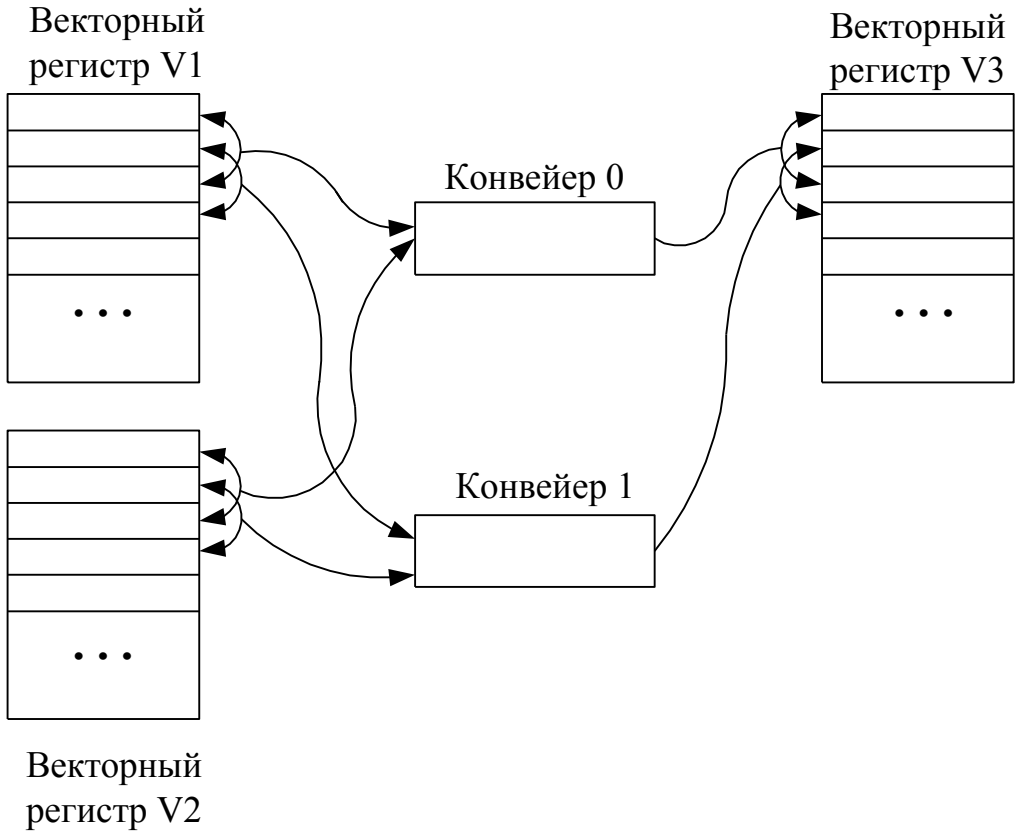

Рисунок 18 - Выполнение векторных операций в компьютере Cray C90

В скалярных операциях, использующих функциональные устройства для вещественной арифметики, работает только один конвейер 0.

Архитектура Cray C90 позволяет использовать регистр результатов одной векторной операции в качестве входного регистра для последующей векторной операции. Выход одной операции сразу подается на вход другой, причем последняя не обязана ждать завершения первой, а, начиная с некоторого момента, будет работать одновременно с ней. Подобная ситуация называется зацеплением векторных операций. Вообще говоря, глубина зацепления может быть любой, например, чтение векторов, выполнение операций сложения, выполнения операций умножений, запись векторов. Основное назначение зацепления состоит опять-таки в увеличении скорости обработки данных. В самом деле, предположим, что нам нужно выполнить операцию вида $A_1 = B_1 + C_1 \times d$, где каждый входной вектор содержит по n элементов. Пусть в нашем распоряжении есть функциональные устройства сложения и умножения, содержащие по n_1 и n_2 ступеней соответственно. Если выполнять исходную операцию традиционным способом, т.е. сначала векторную операцию умножения, а затем сложения, то вся операция будет реализована за $n_1 + n_2 + 2 + l - 2$ тактов. Если для той же операции воспользоваться режимом с зацеплением, то по сути получится один конвейер длиной $n_1 + n_2 + l - 1$. При больших значениях n время реализации операции по сравнению с обычным способом уменьшится почти в два раза. Эта ситуация схематично показана на рисунке 19.

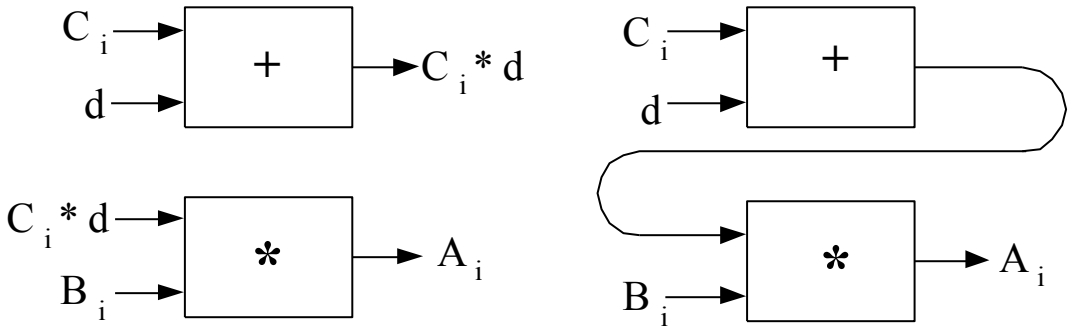

Рисунок 19 - Зацепление векторных операций

в) Секция управления процессора

Команды выбираются из оперативной памяти блоками и заносятся в буфера команд, откуда затем они выбираются для исполнения. Если необходимой для исполнения команды нет в текущем буфере, она ищется в других буферах. Если требуемой команды в буферах не оказалось, то происходит выборка очередного блока.

Такая основа архитектуры компьютера Cray C90. Нетрудно видеть, что идеи параллельной обработки пронизывают все ее составные части. Среди основных особенностей архитектуры, дающих заметный вклад в ускорение выполнения программ, можно назвать следующие.

- *Конвейеризация выполнения команд*. Все основные операции, выполняемые процессором, т.е. обращение в память, обработка команд и выполнение функциональными устройствами являются конвейерными.
- *Независимость функциональных устройств*. Функциональные устройства в Cray C90 являются независимыми, поэтому несколько операций могут выполняться одновременно.
- *Векторная обработка*. Векторная обработка увеличивает скорость и эффективность обработки за счет того, что обработка целого набора (вектора) данных выполняется одной командой. Скорость выполнения операций в векторном режиме может быть в 10-15 раз выше скорости скалярной обработки.
- *Зацепление функциональных устройств*. Возможность выполнения нескольких векторных операций в режиме «макроконвейера» дает дополнительный выигрыш в скорости их обработки.
- *Многопроцессорная обработка*. В максимальной конфигурации компьютер может содержать до 16 независимых процессоров. Эти процессоры могут быть использованы по-

разному. В частности, они могут выполнять несколько независимых программ, но могут и все быть назначены на выполнение одной программы.

Рассмотрев особенности архитектуры данного компьютера, перейдем к оценке его пиковой производительности Cray C90. Поскольку в этом вопросе, прежде всего, интересна скорость выполнения операций над вещественными числами, то нужно максимально загрузить ФУ для вещественной арифметики.

Для определения пиковой производительности компьютера будем задействовать только устройства умножения и сложения. Для получения максимальной производительности их нужно использовать в режиме с зацеплением (операция вида Ai = Bi + Ci * d). Если дополнительно учесть, что каждое такое устройство для выполнения векторной операции использует два внутренних конвейера, то система из двух устройств будет выдавать результат четырех операций за такт. Время такта компьютера равно 4,1 нс, поэтому пиковая производительность одного процессора Cray C90 составит почти 1Гфлопс или 10^9 операций в секунду. Если одновременно работают все 16 процессоров компьютера, то пиковая производительность увеличивается до 16 Гфлопс.

Для понимания того, как для данного компьютера нужно писать эффективные программы, нужно выделить те факторы, которые снижают его производительность на реальных программах.

Компьютер Cray C90 обладает векторно-конвейерной архитектурой. Основной выигрыш во времени можно получить за счет использования векторного режима обработки. Некоторый фрагмент в программе может быть обработан в векторном режиме, если для его выполнения могут быть использованы векторные команды из системы компьютера команд. Если весь фрагмент программы удалось заменить векторными командами, то говорят о его полной векторизации. В противном случае мы имеем дело с частичной векторизацией или невозможностью векторизации фрагмента вовсе. Процесс поиска подходящих фрагментов в программе и их замена векторными командами называется *векторизацией программы*.

Пример векторизуемого фрагмента может выглядеть так:

```
DO i = 1; n
  C(i) = A(i) + B(i)
END DO
```

Для данного фрагмента компилятор сгенерирует последовательность векторных команд:

- загрузка векторов A и B из памяти в векторные регистры;
- векторная операция сложения;
- запись содержимого векторного регистра в память.

Однако далеко не любой фрагмент программы можно векторизовать. Для этого необходимо выполнение двух условий. Первое – это наличие векторов-аргументов. Второе условие немного сложнее и заключается в том, что над всеми элементами векторов должны выполняться одинаковые, независимые операции, для которых существуют аналогичные векторные команды в системе команд компьютера.

Под вектором будем понимать упорядоченный набор однотипных данных, все элементы которого размещены в памяти компьютера с одинаковым смещением друг относительно друга. Примером векторов в программах служат:

- одномерные массивы;
- строки и столбцы матриц (расстояние между соседними элементами одного столбца матрицы равно единице, а между соседними элементами одной строки – размерности матрицы);
- диагональ квадратной матрицы также является примером вектора (расстояние между всеми ее элементами одинаково и равно размерности матрицы плюс единица).

Этот список можно продолжать и далее.

Кроме векторов, в векторизуемом фрагменте могут использоваться и простые переменные. Поскольку в системе команд компьютера Cray C90 есть векторные команды, в которых некоторые аргументы могут быть скалярами, то векторизация следующего фрагмента никаких проблем не вызовет:

```
DO i = 1; n
  B(i) = A(i) + s
END DO
```

Основными кандидатами для векторизации являются самые внутренние циклы всех циклических конструкций программы. Именно они задают перебор «одномерных» наборов данных, которые, в частности, могут быть векторами. Но работа с векторами еще не является достаточным условием для векторизации. Рассмотрим следующий пример:

```
DO i = 1; n
  A(i) = A(i-1) + B(i)
END DO
```

Вычисление i-го элемента массива A не может начаться, пока не вычислен предыдущий элемент. Но в таком случае теряет смысл использование конвейерной обработки! Мы не можем загрузить данные на первую ступень конвейера, пока результат не выйдет из устройства. В данном примере *существует зависимость между операциями*, которая и будет препятствовать векторизации. Именно поэтому мы говорили о возможности лишь при условии существования одинаковых и независимых операций.
Рассмотрим еще один пример:

```
DO i = 1; n
  A(i) = Funct (A(i), B(i))
END DO
```

В явном виде такой фрагмент не может быть векторизован. Компилятор не знает, какая операция соответствует вызову пользовательской функции Funct, и, следовательно, он не знает, на какие векторные команды можно заменить данный фрагмент. В некоторых случаях компилятор может получить дополнительную информацию, если проведет межпроцедурный анализ, выяснит содержание функции Funct и выполнит in-line подстановку.

Предстоит выяснить, какие особенности архитектуры необходимо учитывать при написании действительно эффективных программ для данного компьютера.

1) Анализ факторов, снижающих реальную производительность компьютеров, начнем с обсуждения закона Амдала. Одна из его интерпретаций сводится к тому, что время работы программы определяется ее самой медленной частью. В самом деле, предположим, что одна половина некоторой программы – это **сугубо последовательные вычисления**, которые невозможно векторизовать. Тогда вне зависимости от свойств другой половины, которая может быть идеально векторизована и, скажем, выполнена мгновенно, ускорения работы всей программы более чем в два раза мы не получим.

Влияние данного фактора надо оценивать с двух сторон. Во-первых, по природе самого алгоритма все множество операций программы Ω разбивается на последовательные операции Ω_1 и операции Ω_2, исполняемые в векторном режиме. Если доля последовательных операций велика, то программист сразу должен быть готов к тому, что большого ускорения он никакими средствами не получит и, быть может, следует уже на этом этапе подумать об изменении алгоритма.

Во-вторых, не следует сбрасывать со счетов и качество компилятора. Он вполне может не распознать векторизуемость отдельных конструкций, сгенерировать для них скалярный код и, тем самым, часть «потенциально хороших» операций из Ω_2 перенести в Ω_1.

2) Следующие два фактора, снижающие производительность компьютера на реальных программах, определяют принципиальную невозможность достижения пиковой производительности на векторно-конвейерных устройств.

а) **время разгона векторной команды**. Для инициализации векторной команды требуется несколько тактов, после чего начинается заполнение конвейера и лишь через некоторое время появляются первые результаты. Только после этого устройство каждый такт будет выдавать результат, и его производительность асимптотически будет приближаться к пиковой. Но лишь приближаться. Точного значения пиковой производительности никогда получено не будет.

б) **секционирование длинных векторов**. Вспомним устройство векторных регистров. Каждый регистр содержит 128 элементов. Для запуска векторной команды длинный входной вектор необходимо разбить на секции по 128 элементов. Секцию вектора, целиком расположенную в регистре, уже можно обрабатывать с помощью векторных команд. Такая технология работы с длинными векторами предполагает, что векторизуемый цикл исходной программы должен быть преобразован в двумерное гнездо циклов. Внешний цикл последовательно перебирает секции, а внутренний работает только с элементами текущей секции. Именно внутренний цикл реально и подлежит векторизации. На практике не стоит пугаться этой процедуры, поскольку она полностью выполняется компилятором.

На переход от обработки одной секции к другой требуется пусть и небольшое, но время, а это пять лишняя задержка и опять падение производительности. Если время разгона влияет только при старте, то секционирование немного снижает производительность во всех точках, кратных 128. Это снижение небольшое и с увеличением длины векторов его относительное влияние становится все меньше и меньше, асимптотически приближаясь к малой константе.

Для того, чтобы немного почувствовать влияние этих факторов, рассмотрим следующий фрагмент программы.

```
DO i = 1; n
  A(i) = B(i) *s + C(i)
END DO
```

В таблице 1 показана производительность компьютера Cray C90 на данном фрагменте в зависимости от длины входных векторов, т.е. от значения n. Необходимо отметить, что приведенные здесь и далее значения производительности на практике могут немного меняться в зависимости от текущей загрузки компьютера и некоторых других факторов. Эти значения нужно рассматривать скорее как ориентир, а не как абсолютный показатель.

Таблица 1 - Производительность Cray C90 на операции $A_i = B_i \times s + C_i$

Длина вектора	Производительность (Мфлопс)	Длина вектора	Производительность (Мфлопс)
1	7,0	150	413,2
2	14,0	256	548,0
4	27,6	257	491,0
16	100,5	512	659,2
32	181,9	1024	720,4
128	433,7	2048	768,0
129	364,3	8192	802,0

И время начального разгона, и секционирование относятся к накладным расходам на организацию векторных операций на конвейерных функциональных устройствах. Показанная зависимость явно стимулирует к работе с длинными векторами данных, т.к. с ростом длины вектора доля накладных расходов в общем времени выполнения операции быстро падает. Одновременно заметим, что очень короткие циклы выгоднее выполнять не в векторном режиме, а в скалярном, поскольку не будет необходимости тратить дополнительное время на инициализацию векторных команд.

3) **Конфликты при обращении в память** у компьютеров серии Cray C90 полностью определяются аппаратными особенностями организации доступа к оперативной памяти. Наибольшее время на разрешение конфликтов требуется при выборке данных с шагом 64, когда постоянно совпадают номера и секций, и подсекций. С другой стороны, выборка с любым нечетным шагом проходит без конфликтов вообще, и в этом смысле она эквивалентна выборке с шагом единица. Для оценки влияния конфликтов при доступе в память рассмотрим следующий пример:

```
DO i = 1, n x k, k
  A(i) = B(i)*s + C(i)
END DO
```

В данном примере происходит выполнение векторной операции $A_i = B_i \times s + C_i$ над векторами длины n в режиме с зацеплением. Значение переменной k определяет шаг, с которым данные выбираются из памяти. Рассматривая выше аналогичный пример для шага, равного единице, мы видели, насколько важным параметром для производительности является длина вектора. Чтобы этот параметр сейчас не мешал, мы сделаем число операций для любого числа значений k одинаковым, положив верхнюю границу цикла равной n x k. Производительность компьютера Cray C90 на данной операции n=1000 показана в таблице 2.

Таблица 2 - Влияние конфликтов в памяти на производительность Cray C90

Шаг по памяти	**Производительность (Мфлопс)**
1	705,2
2	444,6
4	274,6
8	142,8
16	84,5
32	44,3
64	22,6
128	22,6

Как мы видим, производительность падает катастрофически, достигая минимума с шага 64. А причина этому одна – структура фрагмента плохо соответствует особенностям архитектуры компьютера.

Еще одной неприятной стороной конфликтов при обращении к памяти является то, что проявляться они могут по-разному. В предыдущем примере конфликты возникали при использовании цикла с четным шагом. Но такая ситуация слишком очевидна, и опытного программиста она насторожит сразу. Вместе с тем, казалось бы, не должно быть никаких причин для беспокойства при работе фрагмента следующего вида:

```
DO i = 1; n
    DO j = 1; n
   DO  k = 1; n
       X(i,j,k) = X(i,j,k) + P(k,i)*Y(k,j)
   END DO
    END DO
END DO
```

Зависимости между итерациями цикла нет, используются одновременно операции сложения и умножения, внутренний цикл с параметром k легко векторизуется. Однако не все так безобидно, как кажется. Решающее значение имеет то, каким образом описан массив X. Предположим, что описание имеет вид:

```
DIMENSION X(40, 40, 1000)
```

По определению Фортрана массивы хранятся в памяти «по столбцам». Следовательно, при изменении последнего индексного выражения на единицу реальное смещение по памяти будет равно произведению размеров массива по предыдущим размерностям. Для нашего примера расстояние в памяти между соседними элементами X(i, j, k) и X(i, j, k+1) равно 40 x 40 = 1600. Но, используя другое разложение на множители, число 1600 можно представить произведением 25 x 64! Это число кратно наихудшему шагу для выборки из памяти, поэтому число конфликтов будет максимальным. Можно ли избавиться от конфликтов? Как ни странно, это сделать чрезвычайно легко. Достаточно лишь изменить описание массива, добавив единицу к двум первым размерностям:

```
DIMENSION X(41, 41, 1000)
```

Расстояние между соседними элементами X(i, j, k) и X(i, j, k+1) становится нечетным числом и конфликты исчезают. Немного увеличив объем массива, мы сделали выборку из памяти максимально эффективной. Точно такой же пример можно привести и для языка C, достаточно строки и столбцы поменять местами.

Еще один пример возможного появления конфликтов – это использование косвенной адресации. Рассмотрим следующий фрагмент:

```
DO i = 1; n
    XYZ( IX(i) ) = XYZ(IX(i)) + P(i) * Y(i)
END DO
```

В зависимости от того, к каким элементам массива XYZ реально происходит обращение, число конфликтов будет меняться. Их может не быть вовсе, если, например, IX(i) всегда равно i. Если же предположить, что IX(i) равно некоторому одному и тому же числу для всех i, то число конфликтов будет максимальным (на каждой итерации будет происходить обращение к одному и тому же элементу массива, т.е. будет обращение к одной и той же подсекции одной и той же секции).

4) Следующие три фактора, снижающие производительность Cray C90 определяются тем, что перед началом выполнения любой операции данные должны быть занесены в регистры. Для этого в архитектуре компьютера предусмотрены три независимых канала передачи данных, два из которых могут работать на чтение из памяти, а третий на запись. Такая структура хорошо подходит для операций, требующих не более двух входных векторов. Примером операции служит, в частности, операция $A_i = B_i \times s + D_i$ на которой компьютер может работать в режиме с зацеплением и, следовательно, показывать хорошие значения производительности.

Однако операции с тремя векторными аргументами, например, $A_i = B_i \times C_i + D_i$ уже не могут быть реализованы столь же эффективно. Часть времени будет неизбежно потрачено впустую на ожидание подкачки третьего аргумента для запуска операции с зацеплением. Это является прямым следствием **ограниченной пропускной способности тракта процессор-память** (memory bottleneck). С одной стороны, максимальная производительность достигается на операции с зацеплением, требующей три аргумента, а, с другой стороны, на чтение одновременно могут работать лишь два канала. В таблице 3 приведена производительность компьютера на указанной выше векторной операции, требующей вектора B, C, D, в зависимости от их длины.

Таблица 3 - Производительность Cray C90 на операции $A_i = B_i \times C_i + D_i$

Длина вектора	Производительность (Мфлопс)
10	57,0
100	278,3
1000	435,3
12801	445,0

Также предположим, что пропускная способность каналов не является узким место. В этом случае на предварительное занесение данных в регистры все равно требуется некоторое дополнительное время. Архитектура компьютера такова, что нам **необходимо использовать векторные регистры перед выполнением операций**. Как следствие, требуемые для этой операции чтения/записи будут неизбежно снижать общую производительность. Довольно часто влияние данного фактора можно заметно ослабить, если повторно используемые вектора один раз загрузить в регистры, выполнить все построенные на их основе выражения, а уже затем перейти к оставшейся части программы.

Рассмотрим следующий фрагмент программы:

```
DO j = 1; 120
   DO i = 1; n
   D(i) = D(i) + s*P(i,j-1) + t*P(i,j)
   END DO
END DO
```

Для векторизации внутреннего цикла фрагмента нет никаких препятствий. На каждой из 120 итераций по j для выполнения векторной операции требуется считать три входных вектора D(i), P(i,j-1) и P(i,j), и записать один входной D(i). Следовательно, за время работы всего фрагмента будет выполнено 120 x 3 = 360 операций чтения векторов и 120 операций записи.

Сделаем несложное эквивалентное преобразование данного фрагмента. Явно выпишем каждые две последовательные итерации цикла по j, приведя его к следующему виду:

```
DO j = 1, 120, 2
   DO i = 1, n
   D(i) = D(i) + s*P(i,j-1) + t*P(i,j) + s*P(i,j) + t*P(i,j+1)
   END DO
END DO
```

Теперь на каждой из 60 итераций внешнего цикла потребуется четыре входных вектора D(i), P(i,j-1), P(i,j), P(i,j+1) и, опять же, один выходной D(i). Суммарно, для нового варианта фрагмента будет выполнено 60 x 4 = 240 операций чтения и 60 операций записи. Выигрыш очевиден. Преобразование подобного рода носит название *раскрутки цикла*. Оно имеет максимальный эффект в том случае, когда на соседних итерациях цикла используются одни и те же данные. В таблица 4 показана производительность компьютера на данном фрагменте в зависимости от глубины раскрутки. Значение n равно 128.

Таблица 4 - Зависимость производительности Cray C90 от глубины раскрутки

Глубина раскрутки	Производительность (Мфлопс)
1	612,9
2	731,6
3	780,7
4	807,7

Теоретически, с увеличением глубины раскрутки растет и производительность, приближаясь в пределе к некоторому значению. Однако на практике максимальный эффект достигается где-то на первых шагах, а затем производительность либо остается примерно одинаковой, либо падает. Основная причина данного несоответствия теории и практики заключается в том, что компьютеры Cray C90 имеют сильно ограниченный набор векторных регистров: 8 регистров по 128 слов в каждом. Как правило, увеличение глубины раскрутки ведет к увеличению числа входных векторов. Так было и в нашем случае. Фрагмент в исходной форме требовал по три входных вектора на каждой итерации внешнего цикла. Раскрутка глубиной 2 привела к необходимости загрузки четырех векторов, для раскрутки на глубину 3 потребуется пять векторов и т.д. Каждый дополнительный вектор предполагает наличие дополнительного регистра, что с увеличением глубины раскрутки станет узким местом.

Теперь вспомним, что значение пиковой производительности вычислялось при условии одновременной работы всех функциональных устройств. Предположим, что некоторый алгоритм выполняет одинаковое число операций сложения и умножения. Пусть сначала должны выполниться все операции сложения, и лишь после этого операции умножения. В такой ситуации в каждый момент времени в компьютере будут задействованы только устройства одного типа. Более половины от пиковой производительности получить нельзя. Присутствующая **несбалансированность в использовании функциональных устройств** является серьезным фактором, сильно снижающим реальную производительность компьютера. Соответствующие данные можно найти в таблице 5.

Таблица 5 - Производительность Cray C90 на различных операциях

Длина вектора	**Производительность на операции (Мфлопс)**			
	$a_i = b_i + c_i$	$a_i = bi / ci$	$a_i = s / bi + t$	$a_i = s / bi \times t$
10	35,5	24,8	49,7	46,1
100	202,9	88,4	197,4	166,5
1000	343,8	117,2	283,8	215,9

В наборе функциональных устройств нет устройства деления. Для выполнения данной операции используется устройство вычисления обратной величины и устройство умножения. Отсюда сразу следует, что производительность фрагмента в терминах операций деления будет очень низкой. Это полностью подтверждает столбец таблицы 5, соответствующий операции $a_i = bi / ci$. Кроме того, использование деления вместе с операцией сложения немного выгоднее, чем с умножением. Это явно видно из последних двух столбцов таблицы.

5) Часто структура программы бывает такова, что в ней постоянно происходит **передача управления** из одной части в другую. Возможными причинами этому могут служить частое обращение к различным небольшим подпрограммам и функциям, либо просто запутанная структура управления из-за большого числа переходов. Следствием такой структуры станет частая перезагрузка буферов команд, и, следовательно, расположенных в нескольких буферах. Если же перезагрузка частая, т.е. фрагмент или программа обладают малой локальностью вычислений, то производительность может меняться в очень широких пределах.

На производительность реальных программ одновременно оказывают влияние в той или иной степени ВСЕ перечисленные выше факторы. В самом деле, программы не бывают векторизуемыми на все 100%. Всегда есть некоторая последовательная инициализация, ввод/вывод или что-то подобное. Вместе с этим, обязательно будет присутствовать какое-то число конфликтов в памяти, быть может легкая несбалансированность в использовании функциональных устройств. Для части операций может не хватать каналов чтения/записи, векторных регистров и т.д. по всем изложенным выше факторам. Не стоит сбрасывать со счетов и влияние компилятора, который вполне мог что-то векторизовать или что-то сделать не оптимально.

Предположим, что влияние каждого отдельного фактора в программе позволяет достичь 85% пиковой производительности. Примем самую простую модель, в которой суммарное влияние оценивается произведением коэффициентов для каждого фактора. Поскольку мы уже выделили 10 факторов, что дает суммарный эффект $0{,}85^{10}$ или менее, чем до 0,2 от пика! Если мы хотим добиться хорошей производительности данного компьютера, то необходимо принимать во внимание все указанные выше факторы одновременно, минимизируя их суммарное проявление в программе. Безусловно, это сделать можно, но это трудная работа. Работа, которую нельзя недооценивать, и большую часть которой лучше выполнить на этапе планирования алгоритма и составления программы.

Линия векторно-конвейерных компьютеров Cray занимает первые места по простоте достижения относительно высокой реальной производительности в сравнении с другими архитектурами. Все классы компьютеров обладают целым набором подобных факторов. Главное – это знать не только то, почему компьютер считает быстро, но и то, что в его архитектуре мешает достигать высокой производительности.