Лекция 16. Параллельные процессы и вычислительные системы

16.1 Векторные и векторно-конвейерные вычислительные системы

Хотя производительность ВС общего назначения неуклонно возрастает, по-прежнему остаются задачи, требующие существенно большей вычислительной мощности. К таким, прежде всего, следует отнести задачи моделирования реальных процессов и объектов, для которых характерна обработка больших регулярных массивов чисел в форме с плавающей запятой. Такие массивы представляются матрицами и векторами, а алгоритмы их обработки описываются в терминах матричных операций. Как известно, основные матричные операции сводятся к однотипным действиям над парами элементов исходных матриц, которые, чаще всего, можно производить параллельно. В универсальных ВС, ориентированных на скалярные операции, обработка матриц выполняется поэлементно и последовательно. При большой размерности массивов последовательная обработка элементов матриц занимает слишком много времени, что и приводит к неэффективности универсальных ВС для рассматриваемого класса задач. Для обработки массивов требуются вычислительные средства, позволяющие с помощью единой команды производить действие сразу над всеми элементами массивов — средства ***векторной обработки***.

В средствах векторной обработки под ***вектором*** понимается одномерный массив однотипных данных (обычно в форме с плавающей запятой), регулярным образом размещенных в памяти ВС. Если обработке подвергаются многомерные массивы, их также рассматривают как векторы. Такой подход допустим, если учесть, каким образом многомерные массивы хранятся в памяти ВМ. Пусть имеется массив данных *A,* представляющий собой прямоугольную матрицу размерности 4x5 (рис. 16.1.1)

a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
a_{31}	a_{32}	a_{33}	a_{34}	a_{35}
a_{41}	a_{42}	a_{43}	a_{44}	a_{45}

Рис. 16.1.1 Прямоугольная матрица данных

При размещении матрицы в памяти все ее элементы заносятся в ячейки с последовательными адресами, причем данные записаны строка за строкой или столбец за столбцом. С учетом такого размещения многомерных массивов в памяти вполне допустимо рассматривать их как векторы и ориентировать

соответствующие вычислительные средства на обработку одномерных массивов данных (векторов).

Действия над многомерными массивами имеют свою специфику, если рассмотренная в примере матрица расположена в памяти построчно, адреса последовательных элементов строки различаются на единицу, а для элементов столбца шаг равен пяти. При размещении матрицы по столбцам единице будет равен шаг по столбцу, а шаг по строке — четырем. В векторной концепции для обозначения шага, с которым элементы вектора извлекаются из памяти, применяют термин ***шаг по индексу*** (stride).

Еще одной характеристикой вектора является число составляющих его элементов — ***длина вектора***.

16.2 Векторный процессор

Векторный процессор — это процессор, в котором операндами некоторых команд могут выступать упорядоченные массивы данных — векторы. Векторный процессор может быть реализован в двух вариантах. В первом он представляет собой дополнительный блок к универсальной вычислительной машине (системе). Во втором — векторный процессор — это основа самостоятельной ВС.

Рассмотрим возможные подходы к архитектуре средств векторной обработки. Наиболее распространенные из них сводятся к трем группам:

- конвейерное АЛУ;
- массив АЛУ;
- массив процессорных элементов.

Последний вариант — один из случаев многопроцессорной системы, известной как *матричная ВС*. Понятие векторного процессора имеет отношение к двум первым группам. Эти две категории иллюстрирует рис. 16.2.2

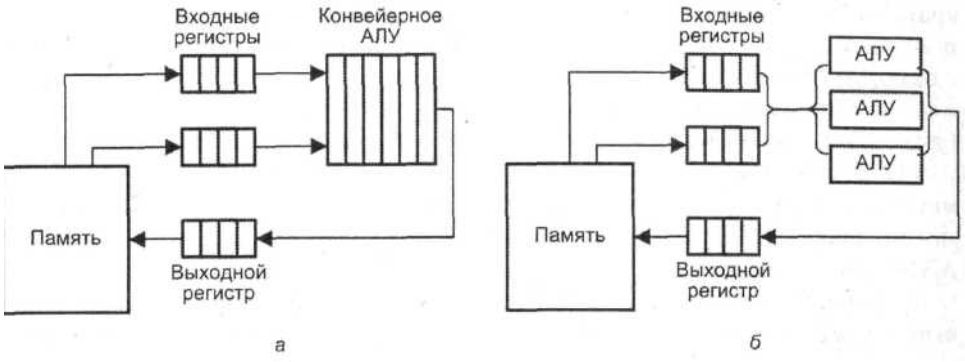

Рис. 16.2.2 Варианты векторных вычислений:
а — с конвейерным АЛУ; б — с несколькими АЛУ

Обобщенная структура векторного процессора приведена на рис. 16.2.3 На схеме показаны основные узлы процессора, без детализации некоторых связей между ними.

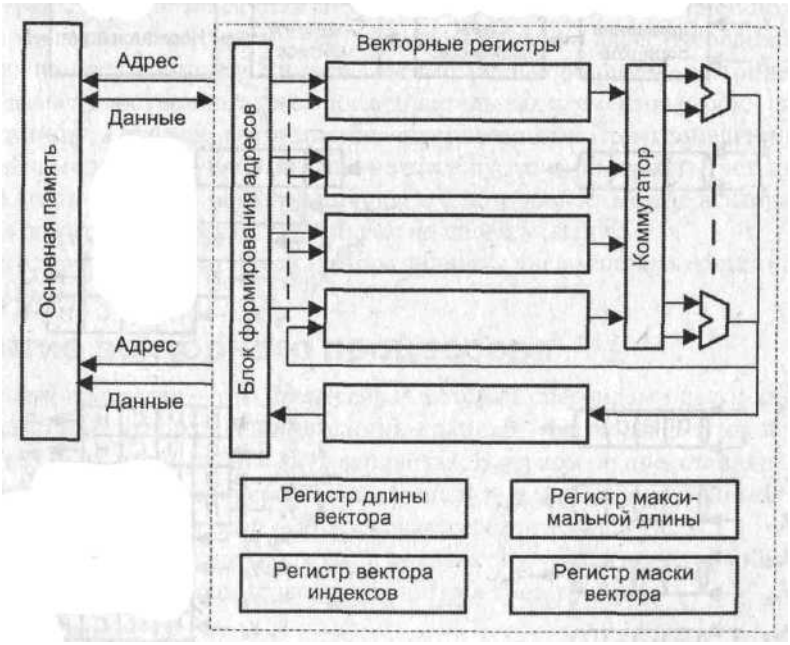

Рис. 16.2.3 Структура векторного процессора

Обработка всех *n* компонентов векторов-операндов задается одной ***векторной командой***. Общепринято, что элементы векторов представляются числами в форме с плавающей запятой (ПЗ). АЛУ векторного процессора может быть реализовано в виде единого конвейерного устройства, способного выполнять все предусмотренные операции над числами с ПЗ.

Кроме того, в состав ***векторной вычислительной системы*** обычно включают и скалярный процессор, что позволяет параллельно выполнять векторные и скалярные команды.

Для хранения векторов-операндов вместо множества скалярных регистров используют так называемые ***векторные регистры,*** которые представляют собой совокупность скалярных регистров, объединенных в очередь типа FIFO.

Система команд векторного процессора поддерживает работу с векторными регистрами и обязательно включает в себя команды:

- загрузки векторного регистра содержимым последовательных ячеек памяти, указанных адресом первой ячейки этой последовательности;
- выполнения операций над всеми элементами векторов, находящихся в векторных регистрах;
- сохранения содержимого векторного регистра в последовательности ячеек памяти, указанных адресом первой ячейки этой последовательности.

Важным элементом любого векторного процессора (ВП) является *регистр длины вектора*. Этот регистр определяет, сколько элементов фактически содержит обрабатываемый в данный момент вектор, то есть сколько индивидуальных операций с элементами нужно сделать. В некоторых ВП присутствует также *регистр максимальной длины вектора,* определяющий максимальное число элементов вектора, которое может быть одновременно обработано аппаратурой процессора. Этот регистр используется при разделении очень длинных векторов на сегменты, длина

которых соответствует максимальному числу элементов, обрабатываемых аппаратурой за один прием.

Достаточно часто приходится выполнять такие операции, в которых должны участвовать не все элементы векторов. Векторный процессор обеспечивает данный режим с помощью *регистра маски вектора.* В этом регистре каждому элементу вектора соответствует один бит. Установка бита в единицу разрешает запись соответствующего элемента вектора результата в выходной векторный регистр, а сброс в ноль — запрещает.

16.3 Структуры типа «память-память» и «регистр-регистр»

Принципиальное различие архитектур векторных процессоров проявляется в том, каким образом осуществляется доступ к операндам. При организации *«память-память»* элементы векторов поочередно извлекаются из памяти и сразу же направляются в функциональный блок. По мере обработки получающиеся элементы вектора результата сразу же заносятся в память. В архитектуре типа *«регистр-регистр»* операнды сначала загружаются в *векторные регистры,* каждый из которых может хранить сегмент вектора, например 64 элемента. Векторная операция реализуется путем извлечения операндов из векторных регистров и занесения результата в векторные регистры.

Преимущество ВП с **режимом «память-память»** состоит в возможности обработки длинных векторов, в то время как в процессорах типа «регистр-регистр» приходится разбивать длинные векторы на сегменты фиксированной длины. К сожалению, за гибкость режима «память-память» приходится расплачиваться относительно большими издержками, известными как *время запуска,* представляющее собой временной интервал между инициализацией команды и моментом, когда первый результат появится на выходе конвейера. Большое время запуска в процессорах типа «память-память» обусловлено скоростью доступа к памяти, которая намного больше скорости доступа к внутреннему регистру.

В вычислительных системах **типа «регистр-регистр»** векторы имеют сравнительно небольшую длину, но время запуска значительно меньше чем в случае «память-память». Этот тип векторных систем гораздо более эффективен при обработке коротких векторов, но при операциях над длинными векторами векторные регистры должны загружаться сегментами несколько раз.

16.4 Матричные вычислительные системы

Назначение ***матричных вычислительных систем*** во многом схоже с назначением векторных ВС — обработка больших массивов данных. В основе матричных систем лежит ***матричный процессор*** (array processor), состоящий из регулярного массива процессорных элементов (ПЭ). Организация систем подобного типа на первый взгляд достаточно проста. Они имеют общее управляющее устройство, генерирующее поток команд,

и большое число ПЭ, работающих параллельно и обрабатывающих каждый свой поток данных. Однако на практике, чтобы обеспечить достаточную эффективность системы при решении широкого круга задач, необходимо организовать связи между процессорными элементами так, чтобы наиболее полно загрузить процессоры работой. Именно характер связей между ПЭ и определяет разные свойства системы. Ранее уже отмечалось, что подобная схема применима и для векторных вычислений.

Между матричными и векторными системами есть **существенная разница**. Матричный процессор интегрирует множество идентичных функциональных блоков (ФБ), логически объединенных в матрицу и работающих в SIMD-стиле. Не столь существенно, как конструктивно реализована матрица процессорных элементов — на едином кристалле или на нескольких. Важен сам принцип — ФБ логически скомпонованы в матрицу и работают синхронно, то есть присутствует только один поток команд для всех. Векторный процессор имеет встроенные команды для обработки векторов данных, что позволяет эффективно загрузить конвейер из функциональных блоков. В свою очередь, векторные процессоры проще использовать, потому что команды для обработки векторов — это более удобная для человека модель программирования, чем SIMD.

Структуру матричной вычислительной системы можно представить в виде, показанном на рис. 16.4.

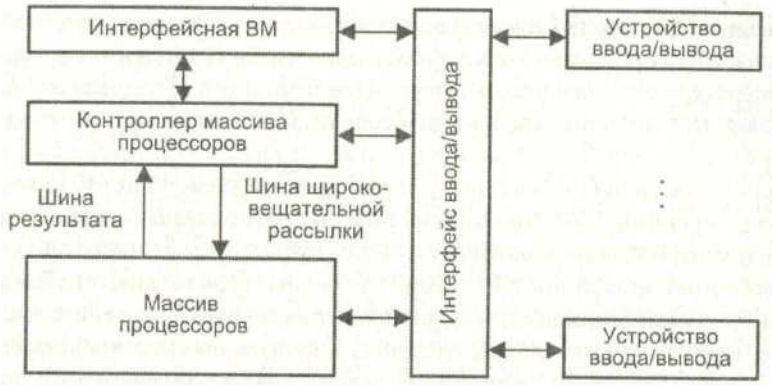

Рис. 16.4.1 Обобщенная модель матричной SIMD-системы

Собственно параллельная обработка множественных элементов данных осуществляется *массивом процессоров* (МПр). Единый поток команд, управляющий обработкой данных в массиве процессоров, генерируется *контроллером массива процессоров* (КМП). КМП выполняет последовательный программный код, реализует операции условного и безусловного переходов, транслирует в МПр команды, данные и сигналы управления. Команды обрабатываются процессорами в режиме жесткой синхронизации. Сигналы управления используются для синхронизации команд и пересылок, а также для управления процессом вычислений, в частности определяют, какие процессоры массива должны выполнять операцию, а какие — нет. Команды, данные и сигналы управления передаются из КМП в массив процессоров по *шине широковещательной рассылки.* Поскольку выполнение операций условного перехода зависит от результатов

вычислений, результаты обработки данных в массиве процессоров транслируются в КМП, проходя по *шине результата.*

Рассматривая массив процессоров, следует учитывать, что для хранения множественных наборов данных в нем, помимо множества процессоров, должно присутствовать и множество модулей памяти. Кроме того, в массиве должна быть реализована сеть взаимосвязей, как между процессорами, так и между процессорами и модулями памяти. Таким образом, под термином *массив процессоров* понимают блок, состоящий из процессоров, модулей памяти и сети соединений.

16.5 Массив процессоров

В матричных SIMD-системах распространение получили два основных типа архитектурной организации массива процессорных элементов.

В первом варианте, известном как архитектура типа *«процессорный элемент - процессорный элемент»* («ПЭ-ПЭ»), *N* процессорных элементов (ПЭ) связаны между собой сетью соединений (рис. 16.5.1*)*.

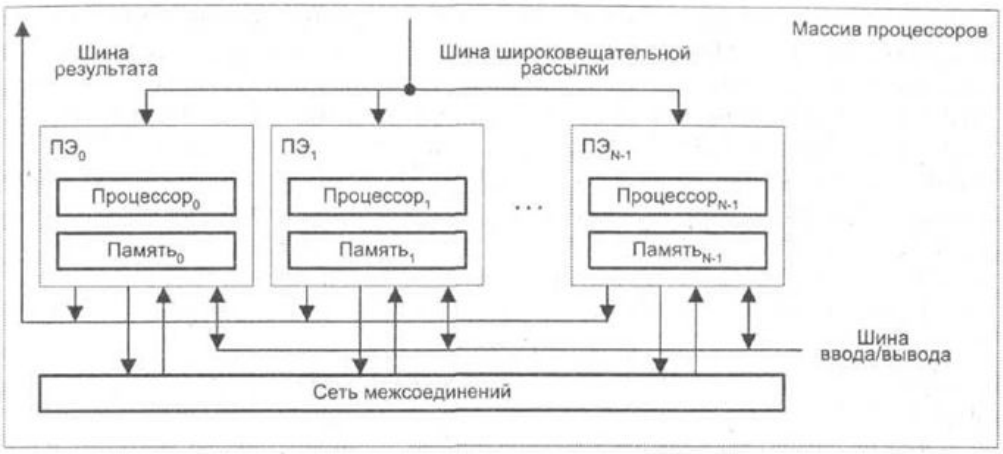

Рис. 16.5.1 Модель массива процессоров типа «процессорный элемент» - «процессорный - элемент»

Каждый ПЭ — это процессор с локальной памятью. Процессорные элементы выполняют команды, получаемые из КМП по шине широковещательной рассылки, и обрабатывают данные как хранящиеся в их локальной памяти, так и поступающие из КМП. Обмен данными между процессорными элементами производится по *сети соединений,* в то время как *шина ввода/вывода* служит для обмена информацией между ПЭ и устройствами ввода/вывода. Для трансляции результатов из отдельных ПЭ в контроллер массива процессоров служит *шина результата.*

Второй вид архитектуры — *«процессор-память»* — показан на рис. 16.5.2. В такой конфигурации двунаправленная сеть соединений связывает *N* процессоров с *M* модулями памяти. Процессоры управляются КМП через широковещательную шину. Обмен данными между процессорами осуществляется как через сеть, так и через модули памяти. Пересылка данных между модулями памяти и устройствами ввода/вывода обеспечивается шиной ввода/вывода. Для передачи данных из конкретного модуля памяти в КМП служит шина результата.

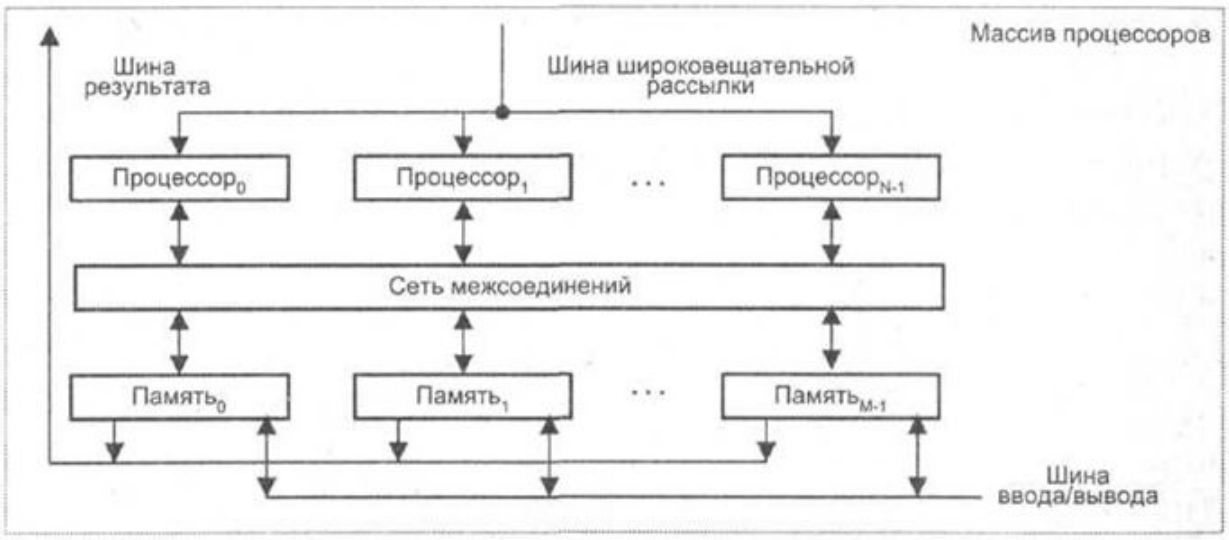

Рис. 16.5.2 Модель массива процессоров типа «процессорный элемент» – «процессорный - элемент»

Эффективность сетей взаимосвязей процессорных элементов во многом определяет возможную производительность всей матричной системы. Применение находят самые разнообразные топологии сетей.

16.6 Ассоциативные вычислительные системы

К классу SIMD относятся и так называемые ассоциативные вычислительные системы. В основе подобной ВС лежит ассоциативное запоминающее устройство, а точнее — *ассоциативный процессор,* построенный на базе такого ЗУ (ассоциативная память представляет собой ЗУ, где выборка информации осуществляется не по адресу операнда, а по отличительным признакам операнда). Запись в традиционное ассоциативное ЗУ также производится не по адресу, а в одну из незанятых ячеек.

Ассоциативный процессор (АП) можно определить как ассоциативную память, допускающую параллельную запись во все ячейки, для которых было зафиксировано совпадение с ассоциативным признаком. Эта особенность АП, носящая название *мультизаписи,* является первым отличием ассоциативного процессора от традиционной ассоциативной памяти. Считывание и запись информации могут производиться по двум срезам запоминающего массива — либо это все разряды одного слова, либо один и тот же разряд всех слов. При необходимости выделения отдельных разрядов среза лишние позиции допустимо маскировать. Каждый разряд среза в АП снабжен собственным процессорным элементом, что позволяет между считыванием информации и ее записью производить необходимую обработку, то есть параллельно выполнять операции арифметического сложения, поиска, а также эмулировать многие черты матричных ВС.

Таким образом, *ассоциативные ВС* или *ВС с ассоциативным процессором* есть не что иное, как одна из разновидностей параллельных ВС, в которых *n* процессорных элементов представляют собой простые устройства, как правило, последовательной поразрядной обработки. При этом каждое слово (ячейка) ассоциативной памяти имеет свое собственное устройство обработки данных

(сумматор). Операция осуществляется одновременно всеми *n* ПЭ. Все или часть элементарных последовательных ПЭ могут синхронно выполнять операции над всеми ячейками или над выбранным множеством слов ассоциативной памяти.

16.7 Вычислительные системы с систолической структурой

В фон-неймановских машинах данные, считанные из памяти, однократно обрабатываются в процессорном элементе, после чего снова возвращаются в память (рис. 16.7.1 а).

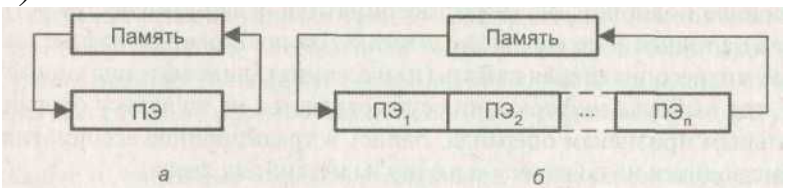

Рис. 16.7.1 Обработка данных в ВС: а — фон-неймановского типа; б — систолической структуры

Авторы идеи систолической матрицы Кунг и Лейзерсон предложили организовать вычисления так, чтобы данные на своем пути от считывания из памяти до возвращения обратно пропускались через как можно большее число ПЭ (рис. 16.7.1 б).

Если сравнить положение памяти в ВС со структурой живого организма, то по аналогии ей можно отвести роль сердца, множеству ПЭ — роль тканей, а поток данных рассматривать как циркулирующую кровь. Отсюда и происходит название *систолическая матрица* (систола — сокращение предсердий и желудочков сердца при котором кровь нагнетается в артерии). Систолические структуры эффективны при выполнении матричных вычислений, обработке сигналов, сортировке данных и т. д. В качестве примера авторами идеи был предложен линейный массив для алгоритма матричного умножения, показанный на рис. 16.7.2.

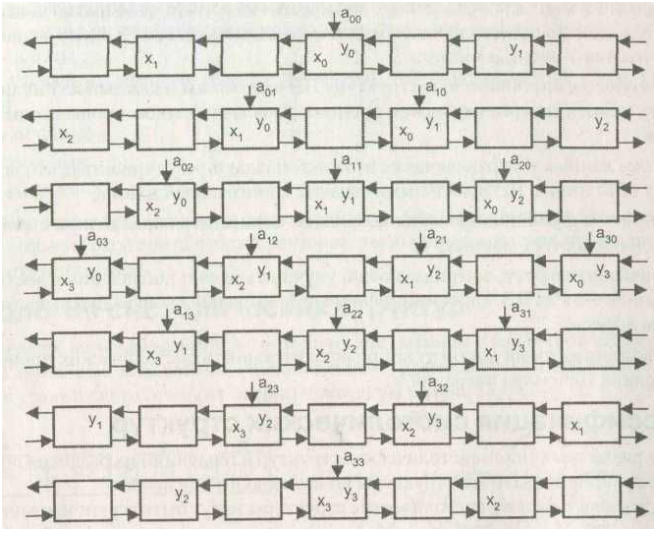

Рис. 16.7.2 Процесс векторного умножения матриц (п = 4)

В основе схемы лежит ритмическое прохождение двух потоков данных x_i, и y_i навстречу друг другу. Последовательные элементы каждого потока разделены одним тактовым периодом, чтобы любой из них мог встретиться с любым элементом встречного потока. Если бы они следовали в каждом периоде, то элемент x_i, никогда бы не встретился с элементами y_{i+1}, y_{i+3} ,... Вычисления выполняются параллельно в процессорных элементах, каждый из которых реализует один шаг в операции вычисления скалярного произведения (IPS, Inner Product Step) и носит название *IPS-элементпа* (рис. 16.7.3).

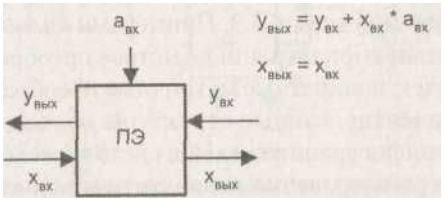

Рис. 16.7.3 Функциональная схема IPS-элемента

Значение $y_{вх}$, поступающее на вход ПЭ, суммируется с произведением входных значений $x_{вх}$ и $a_{вх}$. Результат выходит из ПЭ как $y_{вых}$. Значение $x_{вх}$, кроме того, для возможного последующего использования остальной частью массива транслируется через ПЭ без изменений и покидает его в виде $x_{вых}$.

Таким образом, *систолическая структура* — это однородная вычислительная среда из процессорных элементов, совмещающая в себе свойства конвейерной и матричной обработки и обладающая следующими особенностями:

- вычислительный процесс в систолических структурах представляет собой непрерывную и регулярную передачу данных от одного ПЭ к другому без запоминания промежуточных результатов вычисления;
- каждый элемент входных данных выбирается из памяти однократно и используется столько раз, сколько необходимо по алгоритму, ввод данных осуществляется в крайние ПЭ матрицы;
- образующие систолическую структуру ПЭ однотипны и каждый из них может быть менее универсальным, чем процессоры обычных многопроцессорных систем;
- потоки данных и управляющих сигналов обладают регулярностью, что позволяет объединять ПЭ локальными связями минимальной длины;
- алгоритмы функционирования позволяют совместить параллелизм с конвейерной обработкой данных;
- производительность матрицы можно улучшить за счет добавления в нее определенного числа ПЭ, причем коэффициент повышения производительности при этом линеен.

В настоящее время достигнута производительность систолических процессоров порядка 1000 млрд операций/с.

16.8 Классификация систолических структур

Анализ различных типов систолических структур и тенденций их развития позволяет классифицировать эти структуры по нескольким признакам.

По ***степени гибкости*** систолические структуры могут быть сгруппированы на:

специализированные;

алгоритмически ориентированные;

программируемые.

Специализированные структуры ориентированы на выполнение определенного алгоритма. Эта ориентация отражается не только в конкретной геометрии систолической структуры, статичности связей между ПЭ и числе ПЭ, но и в выборе типа операции, выполняемой всеми ПЭ. Примерами являются структуры, ориентированные на рекурсивную фильтрацию, быстрое преобразование Фурье для заданного количества точек, конкретные матричные преобразования.

Алгоритмически ориентированные структуры обладают возможностью программирования либо конфигурации связей в систолической матрице, либо самих ПЭ. Возможность программирования позволяет выполнять на таких структурах некоторое множество алгоритмов, сводимых к однотипным операциям над векторами, матрицами и другими числовыми множествами.

В программируемых систолических структурах имеется возможность программирования как самих ПЭ, так и конфигурации связей между ними. При этом ПЭ могут обладать локальной памятью программ, и хотя все они имеют одну и ту же организацию, в один и тот же момент времени допускается выполнение различных операций из некоторого набора. Команды или управляющие слова, хранящиеся в памяти программ таких ПЭ, могут изменять и направление передачи операндов.

Но ***разрядности процессорных элементов*** систолические структуры делятся на одноразрядные и многоразрядные.

В одноразрядных матрицах ПЭ в каждый момент времени выполняет операцию над одним двоичным разрядом; а в многоразрядных — над словами фиксированной длины.

По ***характеру локально-пространственных связей*** систолические структуры бывают: одномерные, двухмерные, трехмерные.

Выбор структуры зависит от вида обрабатываемой информации. Одномерные схемы применяются при обработке векторов, двухмерные — матриц, трехмерные — множеств иного типа.

16.9 Графический процессор

Графический процессор — отдельное устройство персонального компьютера или игровой приставки, выполняющее графический рендеринг. Современные графические процессоры очень эффективно обрабатывают и отображают компьютерную графику, благодаря специализированной

конвейерной архитектуре они намного эффективнее в обработке графической информации, чем типичный центральный процессор.

Графический процессор в современных видеоадаптерах применяется в качестве ускорителя трёхмерной графики, однако его можно использовать в некоторых случаях и для вычислений (GPGPU). Отличительными особенностями по сравнению с ЦП являются:

- архитектура, максимально нацеленная на увеличение скорости расчёта текстур и сложных графических объектов;
- ограниченный набор команд.

GPGPU (англ. ***G**eneral-**p**urpose **g**raphics **p**rocessing **u**nits* — «GPU общего назначения») — техника использования графического процессора видеокарты, который обычно имеет дело с вычислениями только для компьютерной графики, чтобы выполнять расчёты в приложениях для общих вычислений, которые обычно проводит центральный процессор. Это стало возможным благодаря добавлению программируемых шейдерных блоков и более высокой арифметической точности растровых конвейеров, что позволяет разработчикам ПО использовать потоковые процессоры для не-графических данных.

CUDA (англ. *Compute Unified Device Architecture*) — программно-аппаратная архитектура, позволяющая производить вычисления с использованием графических процессоров NVIDIA, поддерживающих технологию GPGPU (произвольных вычислений на видеокартах). Впервые появились на рынке с выходом чипа NVIDIA восьмого поколения — G80 и присутствует во всех последующих сериях графических чипов.

CUDA SDK позволяет программистам реализовывать на специальном упрощённом диалекте языка программирования Си алгоритмы, выполнимые на графических процессорах NVIDIA, и включать специальные функции в текст программы на Си. CUDA даёт разработчику возможность по своему усмотрению организовывать доступ к набору инструкций графического ускорителя и управлять его памятью, организовывать на нём сложные параллельные вычисления.

Первоначальная версия CUDA SDK была представлена 15 февраля 2007 года. В основе CUDA API лежит язык Си с некоторыми ограничениями. Для успешной трансляции кода на этом языке, в состав CUDA SDK входит собственный Си-компилятор командной строки **nvcc** компании Nvidia. Компилятор **nvcc** создан на основе открытого компилятора Open64 и предназначен для трансляции host-кода (главного, управляющего кода) и device-кода (аппаратного кода) (файлов с расширением .cu) в объектные файлы, пригодные в процессе сборки конечной программы или библиотеки в любой среде программирования, например в NetBeans.

Использует grid-модель памяти, кластерное моделирование потоков и SIMD инструкции. Применим в основном для высокопроизводительных графических вычислений и разработок NVIDIA-совместимого графического API. Включена возможность подключения к приложениям, использующим

OpenGL и Microsoft Direct3D 9. Создан в версиях для Linux, Mac OS X, Windows.

22 марта 2010 года nVidia выпустила CUDA Toolkit 3.0, который содержал поддержку OpenCL.[1]

Первая серия оборудования, поддерживающая CUDA SDK, G8x, имела 32-битный векторный процессор одинарной точности, использующий CUDA SDK как API (CUDA поддерживает тип double языка Си, однако сейчас его точность понижена до 32-битного с плавающей запятой). Более поздние процессоры GT200 имеют поддержку 64-битной точности (только для SFU), но производительность значительно хуже, чем для 32-битной точности (из-за того что SFU всего 2 на каждый потоковый мультипроцессор, а скалярных процессоров 8). Графический процессор организует аппаратную многопоточность, что позволяет задействовать все ресурсы графического процессора. Таким образом, открывается перспектива переложить функции физического ускорителя на графический ускоритель (пример реализации — nVidia PhysX). Также открываются широкие возможности использования графического оборудования компьютера для выполнения сложных неграфических вычислений: например, в вычислительной биологии и в иных отраслях науки.

По сравнению с традиционным подходом к организации вычислений общего назначения посредством возможностей графических API, у архитектуры CUDA отмечают следующие преимущества в этой области:

- Интерфейс программирования приложений CUDA (CUDA API) основан на стандартном языке программирования Си с некоторыми ограничениями. По мнению разработчиков, это должно упростить и сгладить процесс изучения архитектуры CUDA[2]
- Разделяемая между потоками память (shared memory) размером в 16 Кб может быть использована под организованный пользователем кэш с более широкой полосой пропускания, чем при выборке из обычных текстур
- Более эффективные транзакции между памятью центрального процессора и видеопамятью
- Полная аппаратная поддержка целочисленных и побитовых операций

Недостатки:

- Все функции, выполнимые на устройстве, не поддерживают рекурсии (в версии CUDA Toolkit 3.1 поддерживает указатели и рекурсию) и имеют некоторые другие ограничения
- Архитектуру CUDA поддерживает и развивает только производитель NVidia.