Векторный процессор с архитектурой управления потоком данных: предварительный анализ[1]

Дикарев Н.И., к.т.н., Шабанов Б.М., к.т.н., Шмелев А.С.
Межведомственный Суперкомпьютерный центр РАН, 119991, Москва, Ленинский пр.,32-а,
Тел. 938-61-44,137-33-61, Факс. 938-59-51, E`mail: nic@jscc.ru, shabanov@jscc.ru

Vector Dataflow Processor: Preliminary Analysis

Dikarev N.I, Shabanov B.M., Shmelev A.S.

Продолжающийся рост степени интеграции СБИС привёл к тому, что сейчас можно на одном кристалле реализовать до шестнадцати процессоров (ядер), каждый из которых способен выполнять 4 – 6 команд в такт. Такой темп выдачи команд не меняется уже более 12 лет, а последние 7 лет не растет и тактовая частота процессора, т.е. производительность скалярной обработки одного процессора достигла своего предела. Поэтому дальнейшее увеличение производительности суперЭВМ осуществляется за счет роста числа процессоров (ядер) в суперЭВМ, число которых уже приближается к миллиону. Это накладывает серьёзные ограничения на класс задач, которые могут эффективно использовать столь большое число процессоров, поскольку эффект от распараллеливания по процессорам в суперЭВМ проявляется лишь при наличии параллелизма крупных программных блоков, в каждом из которых должно выполняться не менее нескольких десятков тысяч команд. При уменьшении объема вычислений и переходе через этот порог процессор начинает резко терять производительность, в первую очередь, из-за недостаточного накопления данных в КЭШах, что приводит к частым промахам и падению производительности (эффект «холодного КЭШа»). С другой стороны, при распараллеливании задачи на большее число процессоров увеличиваются потери времени на обмен данными и синхронизацию между ними, что при фиксированном объеме вычислений в задаче также ограничивает возможность дальнейшего роста числа процессоров в суперЭВМ. Очевидный способ решения данной проблемы, заключающийся в существенном увеличении производительности одного универсального процессора, в рамках фон-неймановской архитектуры, как показала практика, не может быть реализован. Однако мы считаем, что это можно осуществить за счет перехода на архитектуру управления потоком данных.

Процессор с архитектурой управления потоком данных позволяет достичь во много раз более высокую производительность по сравнению с процессором традиционной архитектуры. Проекты по разработке такого процессора проводились в США, Японии, Англии и ряде других стран с конца 70-х до середины 90-х годов прошлого века [1,2,3]. Однако ни один из этих проектов не был успешным из-за сложности практической реализации такого процессора. Цель данной работы – сравнить процессор традиционной фон-неймановской архитектуры с процессором с архитектурой управления потоком данных (потоковым процессором), показать преимущества и недостатки последнего. На примере разрабатываемого в МСЦ РАН векторного потокового процессора (ВПП) объяснить каким образом недостатки потоковой архитектуры процессора могут быть устранены, и оценить его производительность по результатам моделирования.

В настоящее время все выпускаемые серийно процессоры имеют фон-неймановскую архитектуру и, как уже отмечалось выше, их производительность достигла своего предела, поскольку ни тактовая частота, ни число команд, выполняемых в такт, последнее время уже не растут. При этом невозможность дальнейшего увеличения пропускной способности конвейера команд у современных микропроцессоров носит фундаментальный характер и определяется последовательной семантикой программ, свойственной фон-неймановской архитектуре процессора. А именно, порядок выполнения команд в программе задается централизованно с помощью счетчика команд, причем часто команды должны выполняться последовательно. Например, если результат команды используется в качестве операнда следующей за ней командой, или выполняется команда перехода, определяющая следующее состояние счетчика команд.

---

[1] Работа выполнена при частичной поддержке гранта РФФИ 13-07-00792.

Для достижения пропускной способности в 4 – 6 команд в такт в суперскалярных процессорах используется сложная аппаратура управления, позволяющая изменять исходный порядок следования команд в программе, выдавая команды по готовности их операндов и осуществляя поиск таких команд в «окне» из примерно 100 команд. Квадратичный рост аппаратных затрат в этой аппаратуре при увеличении числа команд, выдаваемых в такт, приводит к снижению тактовой частоты и росту потребляемой мощности, что и определяет нецелесообразность дальнейшего роста производительности на скалярной обработке у процессорного ядра [4].

Процессор с архитектурой управления потоком данных имеет гораздо более высокую потенциальную производительность за счет того, что параллелизм выполнения команд закладывается уже при составлении графа программы, и его не требуется выявлять с помощью сложной аппаратуры из последовательности команд, заданной счетчиком команд, как это имеет место в фон-неймановском процессоре. Программой в потоковых ЭВМ является граф, узлами которого являются команды, а информация по дугам передается в виде токенов, содержащих поле данных (значение операнда) и поле контекста передаваемых данных. Этот контекст, в частности, однозначно определяет, куда должен быть отправлен токен, т.е. содержит номер команды приемника в графе программы и номер операнда в этой команде. Как правило, разрешается использование команд с не более чем двумя операндами, тогда выдача команды на исполнение осуществляется по прибытию на ее входы последнего из пары токенов со значениями операндов. После вычисления результата в ИУ формируются новые токены со значением результата, которые отправляются на входы последующих команд согласно графу программы, а использованные токены операндов уничтожаются. Тем самым в потоковой ЭВМ работает принцип единственного присваивания, при котором выдача команды на выполнение определяется лишь наличием операндов на ее входах.

Таким образом, в потоковом процессоре в отличие от фон-неймановского отсутствует центральное устройство управления (счетчик команд), а параллелизм выполняемых команд определяется в динамике по приходу операндов на входы команд в децентрализованной схеме. На рис. 1 показана структурная схема потокового процессора, в которой устройство поиска готовых к выполнению команд – память поиска пар (ППП) готовых операндов выполнена в виде *K* работающих в параллель модулей, обеспечивающих работой такое же число ИУ. Для этого требуется лишь задать распределение команд из графа по модулям ППП, например, использовать младшие разряды номера команды для задания номера модуля, в котором осуществляется поиск. Однако, чаще всего, контекст токена содержит не только номер команды в графе программы, но и еще несколько полей, например в разрабатываемом ВПП, это поля индекса, номера итерации и запуска процедуры, которые также должны совпадать у команд, выдаваемых на выполнение из ППП. Это дает возможность одновременного выполнения различных итераций вложенных циклов и различных запусков процедур на одном и том же графе программы, хотя и усложняет реализацию ППП, поскольку она должна выполнять ассоциативный поиск. С другой стороны, дополнительные поля в контексте токена дают возможность использовать их и для задания номера модуля ППП, что делает более равномерным распределение токенов по модулям ППП.

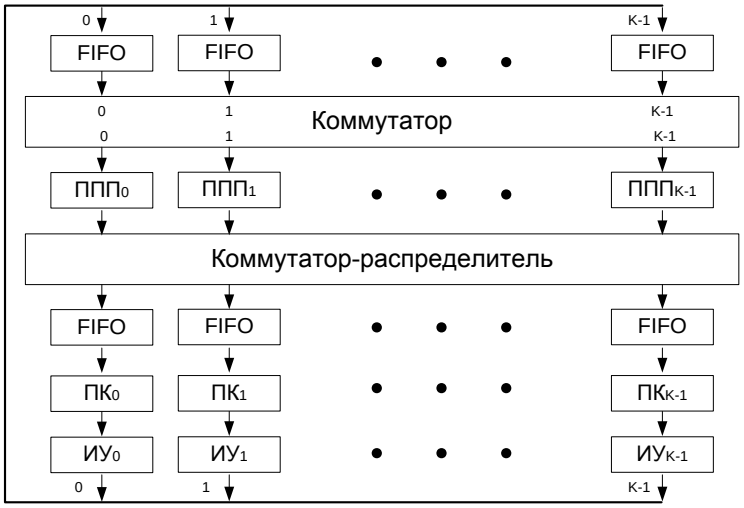

Рис. 1. Структурная схема потокового процессора

Направление токена со значением результата с выхода любого ИУ на вход нужного модуля ППП осуществляется в схеме с помощью коммутатора с прохождением через буфер FIFO на входе коммутатора. Буферы FIFO предотвращают блокировку выхода ИУ в случае конфликта за выход коммутатора, когда в одном такте несколько ИУ посылают токены в один и тот же модуль ППП. В этом случае на выход коммутатора проходит токен с одного из входов (с более высоким приоритетом), а остальные сохраняются в буфере FIFO на входе коммутатора для прохождения в последующие такты. Тем самым буферы FIFO используются для сглаживания неравномерности во времени прихода токенов по модулям ППП. Буферы FIFO должны быть достаточной емкости, поскольку переполнение любого из них из-за кольцевой структуры управления (см. рис. 1) с высокой вероятностью приводит к блокировке всех устройств в этой цепи, и процессор попадает в неработоспособное состояние.

Приход в ППП первого по времени токена операнда приводит к его записи в свободную ячейку ППП, а приход второго токена операнда вызывает выдачу команды на исполнение и освобождение ячейки ППП, занятой первым операндом. Тогда готовая команда в виде пакета из двух значений операндов вместе с контекстом передается с выхода ППП через коммутатор-распределитель на вход того специализированного ИУ или группы однотипных ИУ, где она будет выполняться, предварительно проходя через буфер FIFO и модуль памяти команд (ПК). Буферы FIFO, как и аналогичные буферы на входе коммутатора, предотвращают блокировку выхода коммутатора-распределителя, если ИУ занято обслуживанием предыдущей команды. Что же касается модулей ПК, то это обычные линейно адресуемые запоминающие устройства, из которых по номеру команды в графе читается её код операции и информация о том, сколько токенов со значением результата нужно сформировать и на входы каких команд их направить.

Такой принцип работы позволяет одновременно выполнять на потоковом процессоре команды из нескольких программ, и в отличие от фон-неймановского процессора у него нет аппаратуры, отвечающей за выполнение каждого из потоков команд. Необходимо лишь чтобы емкость модулей ПК была достаточной для хранения команд из разных программ, а также, чтобы не переполнялись модули ППП. Заметим, что модули ПК и ППП должны иметь высокое быстродействие, поскольку вносимая ими задержка вместе со временем вычисления результата в ИУ и задержкой прохождения токенов через два коммутатора определяет время выполнения команды в цепочке команд, связанных зависимостью данных. Если среднюю задержку выборки данных (командного слова) из ПК большой ёмкости при её реализации в виде кэш памяти можно приблизить к времени выборки из блока памяти малой ёмкости, то к ППП такой приём не применим. Дело в том, что эффективная работа кэш памяти основана на многократной выборке одних и тех же данных, например команд в цикле, а в ППП такая повторяемость обращений к одной и той же ячейке памяти отсутствует. Действительно, в каждую ячейку ППП имеет место всего два обращения по приходу каждого из двух токенов операндов команды, и последний из них вызывает выдачу команды на выполнение и освобождение ячейки памяти в ППП. Таким образом, повысить быстродействие модулей ППП можно только за счет

уменьшения их ёмкости, при этом нельзя допустить переполнения даже одного модуля ППП в процессоре, поскольку это приведёт к его неработоспособности.

Легко понять, что одной из главных причин неудачи проектов разработки потокового процессора была сложность практической реализации ППП, которая должна обладать большой ёмкостью, поскольку её переполнение недопустимо, высоким быстродействием, что несовместимо с большой емкостью, и еще осуществлять ассоциативный поиск. Вторая причина связана с использованием коммутатора, с помощью которого осуществляется пересылка токенов с выходов каждого из *K* ИУ на входы любого из *K* модулей ППП. Такой коммутатор должен иметь высокую пропускную способность по каждому из входов и одновременно малую задержку передачи до выхода, что при передаче пакетов малой длины (токенов) трудно выполнимо уже при $K > 16$. Если же учесть, что в потоковом процессоре приходится выполнять в 2 - 3 раза больше команд на программах научных задач по сравнению с фон-неймановским процессором [5], то оказывается, что преимущества в реальной производительности у него фактически нет. Заметим, что сам принцип работы потокового процессора приводит к появлению избыточности в числе выполняемых команд, поскольку, как правило, одна команда формирует не более двух токенов для передачи результата. Если же результат используется в качестве операнда более двух раз, то в граф программы приходится вводить команды дублирования, единственное назначение которых – указать на входы каких еще команд следует послать значение результата. Кроме того, при выполнении ветвления каждое из значений данных, используемых в ветвях программы, необходимо направить в нужную ветвь с помощью отдельной команды переключателя, в то время как традиционном процессоре требуется лишь одна команда условного перехода. Однако главная причина избыточности числа команд у потокового процессора связана с обработкой массивов данных.

В общем случае потоковому процессору не нужны другие устройства памяти кроме ППП. При создании массива A его элементы A(i) поступают в ППП в виде токенов с различными значениями индекса i в поле контекста. Каждый из этих элементов либо находит себе пару в ППП, и команда выдается на выполнение, либо записывается в ППП и ждет второго операнда. Однако при увеличении размера массивов естественный параллелизм у многих задач может быть столь велик, что обрабатывать все их элементы в параллель невозможно, поскольку для этого не хватит ресурсов ни в какой реальной системе [6]. Поскольку переполнение буферов готовых команд на входах ИУ или модулей ППП приводит процессор в тупиковую ситуацию (deadlock), то необходимо ограничение параллелизма. Тогда элементы массивов приходится хранить в ППП на входах команд синхронизации, с помощью которых выдается разрешение на продолжение вычислений, что и приводит к появлению большого числа избыточных команд. Кроме того, фактически статическое хранение массивов в ППП требует многократного увеличения ёмкости этой памяти, что недопустимо как из-за роста аппаратных затрат для осуществления ассоциативного поиска, так и снижения её быстродействия. Поэтому потоковый процессор часто содержит обычную память для хранения массивов, и эта функция снимается с ППП, но тогда для исключения конфликтов информационной зависимости необходима синхронизация обращений к его отдельным элементам по записи и чтению, что приводит к дополнительной аппаратуре и (или) к появлению избыточных команд. Еще одной проблемой, связанной с введением в потоковый процессор линейно адресуемой памяти для хранения массивов, является медленное выделение ресурса этой памяти под создаваемые массивы, поскольку обычно эту функцию выполняет операционная система. Таким образом, ни один из проектов разработки процессора с архитектурой управления потоком данных не достиг более высокой производительности при сопоставимых аппаратных затратах по сравнению с процессорами традиционной архитектуры, и эти проекты были закрыты. В середине 2000-х годов, когда степень интеграции СБИС значительно выросла, появились проекты достижения более высокой производительности у процессора нетрадиционной архитектуры за счет больших аппаратных затрат [7,8], но они также не были успешными.

Разрабатываемый в МСЦ РАН потоковый процессор является векторным, и само наличие векторной обработки позволяет в сотни раз уменьшить число выполняемых команд, поскольку одна векторная команда заменяет собой цикл с независимыми итерациями с числом итераций VL, где VL – длина вектора. Для записи результатов векторных команд и хранения массивов в разрабатываемом ВПП используется линейно адресуемая память, содержащая два уровня - память векторов (ПВ) большой емкости, реализованную на микросхемах динамической памяти,

и быструю локальную память векторов (ЛПВ) значительно меньшей емкости, размещенную на процессорном кристалле. Распределение ресурса ПВ и ЛПВ в ВПП предлагается реализовать на аппаратном уровне, и в качестве единицы фрагментации используется вектор с фиксированным числом слов VL_{max}=256. В этом случае входящее в состав ВПП устройство распределения памяти ведет список свободных векторов для ПВ и ЛПВ, и выделяет для записи результата векторной команды свободный вектор из затребованного списка. Тогда адрес начального элемента вектора вместе с фактической длиной VL, которая меньше или равна VL_{max}, и битом уровня памяти составляют аппаратный указатель (имя) вектора. Этот указатель однозначно определяет вектор для его использования в качестве операнда и передается в поле данных токена на входы последующих команд согласно графу программы. После выполнения последней из этих команд адрес вектора возвращается обратно в список свободных векторов. В результате размещение векторов и массивов в ПВ происходит динамически по мере их создания (уничтожения) и без участия операционной системы. Кроме того, посылая в ППП токены с указателем результата векторной команды после её выполнения можно гарантировать, что последующие команды будут читать уже записанные в ПВ элементы этого вектора. Тем самым, несмотря на использование в ВПП линейно адресуемой памяти работа с этой памятью приближена к принципу работы потокового процессора. Что же касается больших массивов, то их предлагается хранить в ПВ в виде «векторов указателей», т. е. векторов, элементами которых являются указатели векторов подмассивов, как это показано на рис. 2.

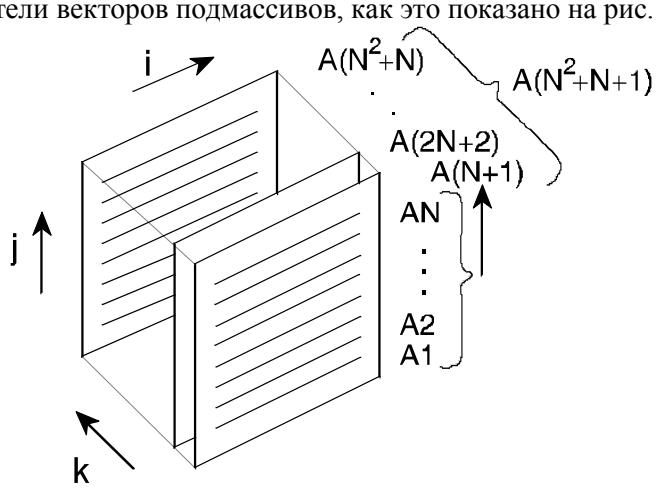

Рис. 2. Представление трехмерного массива векторами-указателями

Заметим, что нумерация векторов A1, A2, AN, A(N+1) и других на рис. 2 не имеет никакого отношения к физическим адресам этих векторов в ПВ. Как уже отмечалось выше, задача определения адреса для записи вектора результата в ПВ или ЛПВ решается аппаратно, путем ведения списков свободных векторов. Покажем, что при таком хранении массивов можно значительно сократить адресные и другие вычисления, выполняемые в скалярных ИУ (СИУ) ВПП, и соответственно повысить производительность векторных ИУ (ВИУ) относительно СИУ. Рассмотрим в качестве примера выполнение программы перемножения матриц в ВПП. Основной объем вычислений в программе перемножения матриц A и B приходится на подпрограмму SGENV [9] по вычислению элементов j-го столбца матрицы результата C:

```
DO 20 k=1, N
      DO 20 i=1, N
20     C(i,j)=C(i,j)+A(i,k)*B(k,j).
```

Внутренний цикл в этой подпрограмме исключается за счет использования векторных команд, и подпрограмму SGENV можно представить в следующем виде:

```
DO 20 k=1, N
20   Cj = Cj + Ak * B(k,j).
```

Граф рассматриваемой подпрограммы для ВПП при использовании метода векторов указателей показан на рис. 3 в предположении, что размер матрицы $N< VL_{max}$. На вход графа поступают подлежащий вычислению указатель вектора столбца Cj с нулевыми значениями элементов, указатель матрицы A – вектор A(N+1), содержащий указатели векторов столбцов

A1,A2,... AN, и указатель j-го столбца матрицы B - Bj. Как уже отмечалось выше, для возможности параллельного выполнения команд из разных итераций цикла контекст токена в ВПП имеет поля индекса (И), итерации (Т) и подпрограммы (П). Поле И содержит номер итерации самого внутреннего цикла (k в формуле выше), поэтому токены указателей, приходящие на вход графа на рис. 3, имеют поле И=1.

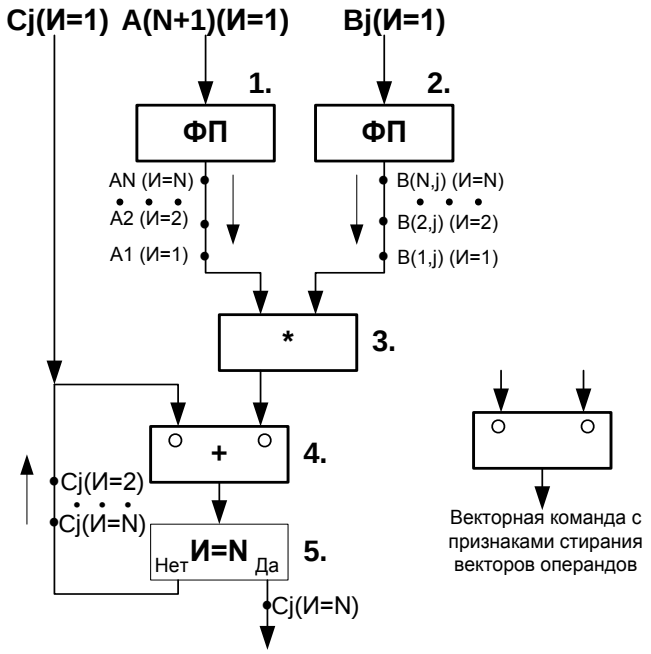

Рис. 3. Граф подпрограммы SGENV для выполнения в ВПП

Команды 1 и 2 "Формирование Потока" (ФП) в графе на рис. 3 читают в ПВ элементы вектора указателя A(N+1) матрицы A и вектора столбца Bj соответственно, формируя последовательность токенов со значениями элементов A1,A2,... AN и B(1,j),B(2,j),... B(N,j) для всех итераций цикла, выполняемого подпрограммой. Поэтому в каждой из N итераций цикла в ВПП выполняются лишь две векторные команды с плавающей запятой (команды 3 и 4 на рис. 3) и одна скалярная команда (команда 5). Команда 5 сравнивает номер текущей итерации k (поле И) в токене вектора Cj с длиной вектора N и осуществляет выход из цикла при k=N, а при невыполнении этого условия увеличивает на 1 значение И в токене результата для передачи Cj на вход команды сложения 4 в следующей итерации цикла. Признаки стирания на входах команды 4 указывают, что оба указателя вектора используются в качестве операнда последний (единственный) раз и после выполнения команды должны быть отправлены в список свободных векторов ЛПВ. Заметим, что вектора, представляющие промежуточные и коротко живущие результаты команд, как для команд 3 и 4 в графе, должны быть направлены компилятором на запись в ЛПВ. Освобождение ресурса памяти путем установки признаков стирания или выполнения команд "Удаление Вектора" также возложено на компилятор.

Сравним число команд, необходимых для выполнения подпрограммы SGENV, в ВПП с векторным процессором традиционной архитектуры, например, CRAY Y-MP. В CRAY Y-MP в каждой итерации цикла этой подпрограммы выполняется 8 команд, из них 5 – скалярных, а в ВПП – 3, и только 1 – скалярная. Значительное уменьшение общего числа выполняемых команд и особенно скалярных команд в ВПП достигается за счет формирования указателей вектора для всех итераций цикла векторными командами ФП, что позволяют исключить одиночные команды обращения к памяти по чтению векторов Ak и элементов B(k,j), а также команды вычисления адресов для этих обращений в традиционном процессоре.

Подпрограмма SGENV в программе перемножения матриц вызывается N раз со значениями j от 1 до N, и полный граф этой программы содержит еще внешний цикл, команды которого отправляют токены на входы графа на рис. 3 и принимают токен результат с его выхода. Эти команды, в частности, читают указатель Bj из вектора указателя матрицы B(N+1) и записывают указатель Cj с выхода графа в j-й элемент вектора указателя C(N+1). Запись номера итерации j в поле Т контекста токена дает возможность ВПП одновременно вычислять

несколько столбцов Cj. Моделирование программы перемножения матриц проводилось на VHDL модели разрабатываемого процессора уровня регистровых станций, что позволяет моделировать процесс выполнения программы с точностью до такта. Моделирование показало, что команды ФП 1 и 2 в графе на рис. 3 обеспечивают полную загрузку работой умножителя, выполняющего команду 3, не допуская перерывов даже при переходе вычисления от одного столбца Cj к следующему. Что же касается сумматора, выполняющего команду 4, то наличие зависимости по данным между итерациями внутреннего цикла, когда вектор Cj, вычисленный в первой итерации (И=1), используется в качестве операнда в следующей (И=2), приводит к большим задержкам. Так при обработке матриц размером 256x256 и пиковой производительности ВПП 64 флоп в такт, когда 32 конвейерных сумматора и умножителя с плавающей запятой в ВИУ обрабатывают вектора страницами по 32 элемента в такт, период поступления команд 4 к сумматору составил 33 такта. Этот период в ВПП определяют задержка выборки векторов операндов из ЛПВ 10 тактов, время разгона конвейерного сумматора 5 тактов, время выполнения команды 5 в СИУ – 2 такта и время на 2 прохода токена по кольцу управления (с выхода ВИУ на вход СИУ и обратно).

Таким образом, в ВПП проявляется известный недостаток потоковой архитектуры, заключающийся в низкой производительности выполнения последовательных команд, когда сумматор загружен работой лишь 8 тактов из 33. Однако ВПП способен увеличить загрузку сумматора за счет вычисления в параллель нескольких столбцов Cj с разными значениями поля T в контексте токена. Как отмечалось выше, по окончании формирования токенов операндов для команды 3 по вычислению первого столбца матрицы C, команды ФП загружают работой умножитель вычислением второго и последующих столбцов матрицы C. При этом нагрузка сумматора, выполняющего команду 4, возрастает, поскольку к нему начинают приходить команды из нескольких итераций внешнего цикла. В результате, как показало моделирование, процесс выполнения программы перемножения матриц в ВПП можно разделить на три этапа. Первый этап – пролог, в ходе которого умножитель работает с максимально возможной производительностью, а нагрузка сумматора возрастает с 24.2% до максимально возможной. Для матриц размером 256x256 и пиковой производительности ВПП 64 флоп в такт длительность пролога составила 10760 тактов. В ходе второго этапа сумматор имеет полную загрузку, выполняя в параллель команды из пяти итераций внешнего цикла. Второй этап длительностью 540380 тактов заканчивается после того как умножитель выполнил последнюю команду 3 с И=256 и T=256. Наконец в эпилоге работает только сумматор, и его нагрузка снижается с выполнения команд из пяти итераций внешнего цикла до одной. Эпилог заканчивается выдачей в устройство ввода-вывода указателя вычисленной матрицы C, и его длительность по результатам моделирования составила 8550 тактов. Тогда реальная производительность ВПП на данной программе равна 59.95 флоп в такт (93.7% от пиковой) при средней производительности на втором этапе 60.99 флоп в такт (95.3% от пиковой).

Из приведенного выше примера видно, что ВПП способен выполнять поиск команд с готовыми операндами в гораздо большем окне по сравнению с суперскалярными процессорами фон-неймановской архитектуры. Так в рассмотренной выше программе перемножения матриц максимальное число токенов в ППП, ожидающих прихода парного токена операнда, достигает 650, а планируемая суммарная емкость модулей ППП в ВПП составляет 8К токенов. Увеличение окна поиска готовых команд со 100 в фон-неймановском процессоре до 8К в ВПП позволяет выполнять в параллель команды не только из нескольких итераций внутреннего цикла программы, но и из итераций следующего вложенного цикла. В свою очередь, это позволяет не снижать производительность процессора при увеличении задержки выборки из памяти до нескольких сотен тактов по сравнению с 15 – 20 тактами, как это имеет место в лучших суперскалярных процессорах.

Производительность ВПП можно повысить за счет использования нескольких комплектов из сумматора и умножителя в ВИУ вместо одного комплекта в рассмотренном выше примере. Так при использовании четырех комплектов АЛУ и пиковой производительности ВПП 256 флоп в такт его производительность на программе перемножения матриц составила 210.8 флоп в такт (82.3% от пиковой производительности). Заметим, что лучшие процессоры традиционной архитектуры имеют производительность векторной обработки 8 – 16 флоп в такт на числах двойной точности. Таким образом, результаты моделирования ВПП подтверждают, что на процессоре с архитектурой управления потоком

данных можно получить производительность, по крайней мере, на порядок выше, чем у процессора традиционной архитектуры.

Литература

1. Arvind and R.S.Nikhil, Executing a program on the MIT tagged-token data-flow architecture. -IEEE Trans. Comput., vol. C-39, 1990, N 3, pp.300-318.
2. Manchester Dataflow Research Project. http://intranet.cs.man.ac.uk/cnc/progects/dataflow.html
3. Sakai S. et al, An Architecture of a Dataflow Single-Chip Processor. -Proc. 16-th Ann. Symp. on Computer Architecture, 1989, pp. 252-273.
4. Дикарев Н.И., Шабанов Б.М. Архитектура процессора и ее влияние на производительность суперЭВМ // Программные продукты и системы, №2 (78), -М., 2007, с.2-5.
5. G.V.Papadopoulos, K.R.Traub, Multithreading: A revisionist view of dataflow architectures. -Proc. 18-th Ann. Symp. on Computer Architecture, 1991, pp.342-351.
6. D.E.Culler and Arvind, Resource requirements of dataflow programs. -Proc. 15-th Ann. Symp. on Computer Architecture, 1988, pp.141-150.
7. The WaveScalar Architecture. http://wavescalar.sc.washington.edu
8. TRIPS Processor Architecture. http://www.cs.utexas.edu/users/cart/trips
9. J.F.Hake and W.Homberg, The impact of memory organization on the performance of matrix calculations. -Parallel Computing, vol. 17, 1991, N 2/3, pp. 311-327.