Гузик Вячеслав Филиппович – Южный федеральный университет; e-mail: gvf@dce.sfedu.ru; 347928, г. Таганрог, пер. Некрасовский, 44; тел.: +78634371737; кафедра вычислительной техники; зав. кафедрой; д.т.н.; профессор.

Гушанский Сергей Михайлович – кафедра вычислительной техники; к.т.н; доцент.

Кубраков Евгений Сергеевич – e-mail: kubrakov90@gmail.com; тел.: 89094268757; кафедра вычислительной техники; аспирант.

Guzik Vyacheslav Philipovich – Southern Federal University; e-mail: gvf@dce.sfedu.ru; 44, Nekrasovsky, Taganrog, 347928, Russia; phone: +78634371737; the department of computing; head of department; dr. of eng. sc.; professor.

Gushansky Sergey Michaylovich – the department of computing; cand. of eng. sc.; associate professor.

Kubrakov Evgeny Sergeevich – e-mail: kubrakov90@gmail.com; phone: +79094268757; the department of computing; postgraduate student.

УДК 004.27

Н.И. Дикарев, Б.М. Шабанов, А.С. Шмелев

ВЕКТОРНЫЙ ПОТОКОВЫЙ ПРОЦЕССОР: ОЦЕНКА ПРОИЗВОДИТЕЛЬНОСТИ*

Производительность одного процессорного ядра не растет уже более 10 лет, и увеличение производительности суперЭВМ осуществляется за счет роста числа процессоров. Проблема поддержания высокой эффективности вычислений с ростом числа процессорных ядер в суперЭВМ актуальна сейчас и будет только обостряться в дальнейшем при переходе суперЭВМ к экзафлопсному диапазону производительности. Цель данной работы – показать, что разрабатываемый векторный процессор с архитектурой управления потоком данных (ВПП) может обеспечить не только значительно более высокую производительность по отношению к лучшим процессорам традиционной архитектуры, но и поддерживать ее при работе с мелкоструктурным параллелизмом. Моделирование времени выполнения на программе перемножения матриц при изменении числа сумматоров и умножителей в ВПП показало, что его векторная производительность может быть поднята до 256 флоп в такт.

СуперЭВМ; векторный процессор; архитектура управления потоком данных; оценка производительности; мелкоструктурный параллелизм.

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

VECTOR DATAFLOW PROCESSOR: PERFORMANCE EVALUATION

Processor core performance is not increasing more than 10 years, so the only way to improve supercomputer performance is to use more cores. Efficient use of increasing core number in supercomputer is a vital issue when moving to Exascale. The aim of this work is to demonstrate that our Vector Dataflow Processor (VDP) performance is many times the conventional processor performance and it can sustain high performance on fine grained parallelism. VDP simulation results are presented on the basis of matrix multiplication execution time. These results are obtained for varied number of vector floating point adders and multipliers in VDP. From these experiments conclusions can be drawn that it is possible to raise VDP core vector performance to 256 flop per clock.

Supercomputer; vector processor; dataflow architecture; performance evaluation; fine grained parallelism.

* Работа выполнена при поддержке гранта РФФИ 13-07-00792.

Продолжающийся рост степени интеграции СБИС привёл к тому, что сейчас можно на одном кристалле реализовать до 16-ти универсальных процессоров (CPU), каждый из которых способен выполнять 4–6 команд в такт. Такой темп выдачи команд не меняется уже более 12-ти 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 достигло одного миллиона. 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.

Проблема поддержания высокой эффективности вычислений с ростом числа процессорных ядер в суперЭВМ актуальна сейчас и будет только обостряться в дальнейшем из-за необходимости использовать более мелкоструктурный параллелизм. Дело в том, что при переходе суперЭВМ к экзафлопсному диапазону производительности появляется возможность при моделировании использовать вместо упрощенных математических моделей реальных систем более сложные, учитывающие важные детали таких систем. При этом модели становятся нелинейными и со связанными структурами. Увеличивается временной диапазон моделирования, причем на разных диапазонах времени и пространства используются разные модели [1]. Все это приводит к необходимости работы с мелкоструктурным и нерегулярным параллелизмом, на котором современные процессоры и ускорители имеют низкую эффективность работы. Цель данной работы – на примере разрабатываемого в МСЦ РАН векторного процессора с архитектурой управления потоком данных (ВПП) показать, что процессор с данной архитектурой может обеспечить не только значительно более высокую производительность по отношению к лучшим CPU, но и поддерживать ее при работе с мелкоструктурным параллелизмом.

В настоящее время все выпускаемые серийно процессоры имеют фоннеймановскую архитектуру и, как уже отмечалось выше, их производительность достигла своего предела, поскольку ни тактовая частота, ни число команд, выполняемых в такт, уже давно не растут. Для достижения пропускной способности в 4 – 6 команд в такт в CPU 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 [2].

Одним из немногих способов повысить производительность процессора, не увеличивая числа команд, выдаваемых в такт, является использование векторных команд, поскольку каждая такая команда выполняет одну и ту же операцию над векторами из нескольких чисел. Системы команд современных микропроцессоров, как правило, имеют в своем составе команды обработки коротких векторов и комплект исполнительных устройств (ИУ) для одновременной обработки всех элементов вектора. Так, в последних микропроцессорах Xeon фирмы Intel используется расширение векторных команд AVX для работы с векторами длиной 256 бит, что позволяет обрабатывать вектора из четырех 64-разрядных чисел с плавающей запятой. Дальнейший рост производительности векторной обработки в CPU ограничен тем, что все элементы вектора должны находиться в пределах одной строки в кэш и

производительности скалярной и векторной обработки в процессоре должны быть сбалансированы. Так, нецелесообразно увеличивать производительность векторной обработки в процессоре более чем в 10 раз по отношению к скалярной производительности, поскольку в этом случае время работы скалярного блока на невекторизуемых участках программы будет доминировать в общем времени выполнения программы (следствие закона Амдала). В ускорителях Intel Xeon Phi длина вектора составляет восемь 64-разрядных слов, и векторная производительность ядра (16 флоп в такт) уже сейчас превышает скалярную производительность в 8 раз, т.е. близка к своему пределу. Как будет показано ниже, векторную производительность 1-го ядра ВПП можно поднять до 256 флоп в такт, причем не только за счет роста скалярной производительности, но и за счет повышения степени векторизации программ.

Препятствия на пути создания потокового процессора. Процессор с архитектурой управления потоком данных, или, более кратко, потоковый процессор, имеет гораздо более высокую потенциальную производительность за счет того, что параллелизм выполнения команд закладывается уже при составлении графа программы. Проекты по разработке такого процессора проводились в США, Японии, Англии и ряде других стран с конца 70-х до середины 90-х гг. прошлого века [3–5]. Однако ни один из этих проектов не был успешным из-за сложности практической реализации такого процессора.

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 (И), номера итерации (Т) и запуска процедуры (П), 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.

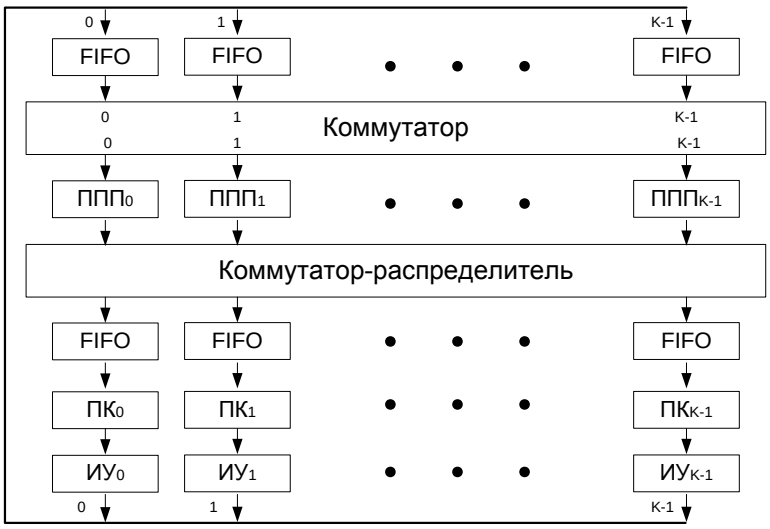

Рис. 1. Структурная схема потокового процессора

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 отсутствуют накладные расходы на переключение между потоками 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 фоннеймановским процессором [6], 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 в 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 [7]. 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 помо-

щью 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 снова приводит к появлению избыточных команд. Еще одной проблемой, связанной с введением в потоковый процессор линейно адресуемой памяти для хранения массивов, является медленное распределение ресурса этой памяти, 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 гг., когда степень интеграции СБИС значительно выросла, появились проекты повышения производительности процессора с нетрадиционной архитектурой за счет больших аппаратных затрат [8, 9], но они также не были успешными.

Векторный потоковый процессор. 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, реализованная на микросхемах динамической памяти, и быстрая локальная память векторов (ЛПВ) значительно меньшей емкости, размещенная на процессорном кристалле. Распределение ресурса ПВ и ЛПВ в ВПП реализовано на аппаратном уровне, и в качестве единицы фрагментации используется вектор с фиксированным числом слов VL_{max}=256, что позволяет быстро выделять место в ПВ или ЛПВ для записи результата векторной команды. 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. 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.

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. Рассмот-

рим в качестве примера выполнение программы перемножения матриц в ВПП. Основной объем вычислений в программе перемножения матриц A и B приходится на подпрограмму SGENV [10] 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).
```

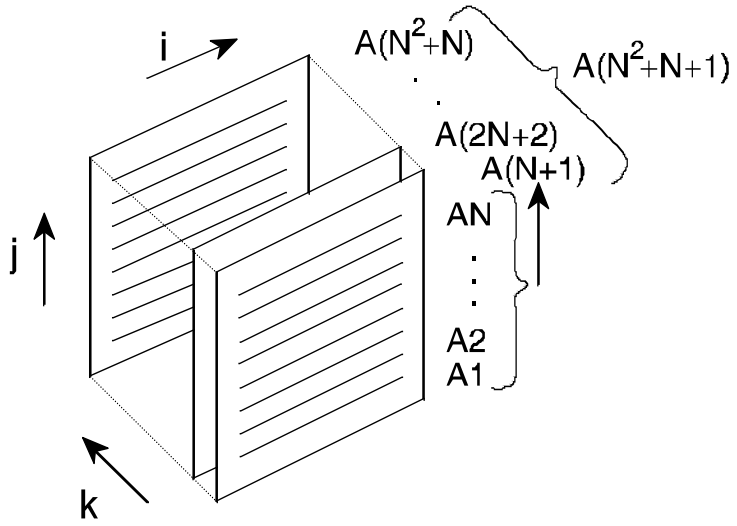

Рис. 2. Представление трехмерного массива векторами-указателями

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 подпрограммы SGENV для ВПП показан на рис. 3 в предположении, что размер матрицы N меньше или равен VL_{max} и столбцы матрицы A с указателями A1,A2,... AN переписаны из ПВ в ЛПВ вместе с указателем матрицы A – вектором A(N+1) до входа в подпрограмму. На вход графа поступают токен вектора A(N+1) с указателем матрицы A, подлежащий вычислению указатель вектора-столбца Cj с нулевыми значениями элементов и указатель j-го столбца матрицы B – Bj. Вектор Cj создается в ЛПВ, а элементы вектора Bj читаются непосредственно из ПВ, поскольку они используются лишь один раз – при вычислении столбца Cj, в отличие от столбцов A1,A2,... AN матрицы A, которые читаются N раз – при вызове подпрограммы SGENV со значениями j от 1 до N.

Команды 1 и 2 "Формирование Потока" (ФП) в графе на рис. 3 читают элементы вектора-указателя A(N+1) матрицы A из ЛПВ и вектора-столбца Bj из ПВ соответственно, формируя последовательность токенов со значениями элементов A1,A2,... AN и B(1,j),B(2,j),... B(N,j) для всех итераций цикла, проставляя в поле И номер итерации k в формуле выше. Тогда 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 зна-

чение 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.

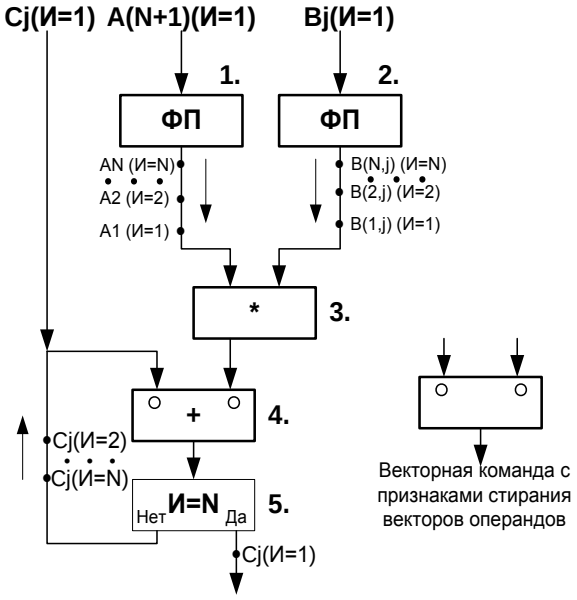

Рис. 3. Граф подпрограммы SGENV для выполнения в ВПП

Заметим, что в векторном процессоре традиционной архитектуры, например CRAY Y-MP, в каждой итерации цикла той же подпрограммы нужно выполнить 8 команд, из них 5 – скалярных, вместо 3 и 1 команды соответственно в ВПП. Значительное уменьшение общего числа выполняемых команд и особенно скалярных команд в ВПП достигается за счет использования команд ФП, которые позволяют исключить из тела цикла команды обращения к памяти по чтению векторов Ak и элементов B(k,j), а также команды вычисления начальных адресов для чтения векторов Ak и увеличения номера итерации k на 1.

Оценка производительности ВПП на программе перемножения матриц. Моделирование проводилось на VHDL модели ВПП уровня регистровых станций, что позволяет моделировать процесс выполнения программы с точностью до такта. Использовался блочный вариант этой программы, в котором матрица произвольного размера состоит из блоков размером NxN, где N не превышает 256, и обработка блоков ведется последовательно. В пределах блока подпрограмма SGENV вызывается N раз со значениями j от 1 до N, причем номер итерации j записывается в поле T контекста токена, что позволяет одновременно выполнять команды из нескольких итераций цикла по j. Как показало моделирование, команды ФП 1 и 2 в графе на рис. 3 обеспечивают полную загрузку работой умножителя, выполняющего команду 3, не допуская перерывов при переходе вычисления от одного столбца Cj к следующему. Однако наличие зависимости по данным между итерациями внутреннего цикла (по k), когда вектор Cj, вычисленный в первой итерации (И=1), используется в качестве операнда в следующей (И=2), приводит к простоям сумматора, выполняющего команду 4. Здесь в ВПП проявляется отме-

ченный выше известный недостаток потоковой архитектуры, заключающийся в низкой производительности выполнения последовательных команд. Так, по результатам моделирования ВПП период поступления команды 4 к сумматору в графе на рис. 3 составил 33 такта. В него входят не только задержка выборки векторов операндов из ЛПВ 10 тактов и время разгона конвейерного сумматора 5 тактов, но также 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).

Лишь возможность ВПП осуществлять поиск готовых команд в гораздо большем окне (в 100 раз) по сравнению с лучшими CPU позволяет не останавливать работу умножителя, выполняющего команду 3. В результате сумматор начинает выполнять команды 4 из нескольких итераций внешнего цикла (по j), что устраняет простои в его работе и производительность ВПП приближается к пиковой. При этом 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 минимально возможной до максимальной, равной производительности умножителя. Так, при пиковой производительности ВПП 64 флоп в такт, когда 32 конвейерных сумматора и умножителя с плавающей запятой в ВИУ обрабатывают векторы страницами по 32 элемента в такт, сумматор в начале пролога загружен работой лишь 8-ми тактов из 33-х. Но после того как умножитель переходит к вычислению столбца C_5, пролог заканчивается, и в ходе второго этапа сумматор имеет полную загрузку, выполняя в параллель команды из пяти итераций внешнего цикла. Второй этап заканчивается после того как умножитель выполнил последнюю команду 3 с И=256 и Т=256. Наконец в эпилоге работает только сумматор, и его нагрузка снижается с выполнения команд из пяти итераций внешнего цикла до одной.

За счет увеличения числа сумматоров и умножителей в ВИУ векторную производительность ВПП можно поднять до 128 и 256 флоп в такт. При этом время пролога и эпилога останется примерно тем же, что было при производительности ВПП 64 флоп в такт за счет роста числа выполняемых сумматором итераций внешнего цикла с 5-ти до 9-ти и 18-ти соответственно. Однако время основного (второго) этапа уменьшится и это приводит к снижению реальной производительности ВПП. Тот же эффект проявляется и при уменьшении размера обрабатываемых матриц. На рис. 4 приведены значения производительности ВВП, полученные по результатам прогона программы перемножения матриц на VHDL-модели ВПП и аналогичные результаты для вычислительного узла на микропроцессорах Intel Xeon E5 при выполнении той же программы из библиотеки Intel MKL на 1-, 8- и 16-ти процессорных ядрах.

Как видно из приведенных на рис. 4 зависимостей, при одной и той же пиковой производительности, равной 128 флоп в такт, реальная производительность ВПП ниже, чем у системы на CPU и составляет 89,6 % от пиковой. С ростом пиковой производительности ВПП до 256 флоп в такт его эффективность падает до 86,8 %, в то время как при 64 флоп в такт эффективность увеличивается до 95 %, что примерно соответствует эффективности системы на CPU. Вместе с тем ВПП обеспечивает значительно более высокую производительность при уменьшении размера обрабатываемых матриц. Как видно из зависимостей на рис. 4, чем больше пиковая производительность ВВП или системы на CPU, тем больше нужен размер матрицы $N_{1/2}$, на котором достигается производительность, равная половине от пиковой производительности. Однако для ВВП с пиковой производительностью 128 и 256 флоп в такт $N_{1/2}$ составляет примерно 60 и 100, а для систем на Intel Xeon E5 с пиковой производительностью 8, 64 и 128 флоп в такт $N_{1/2}$ составляет 650, 1500 и 2500 соответственно.

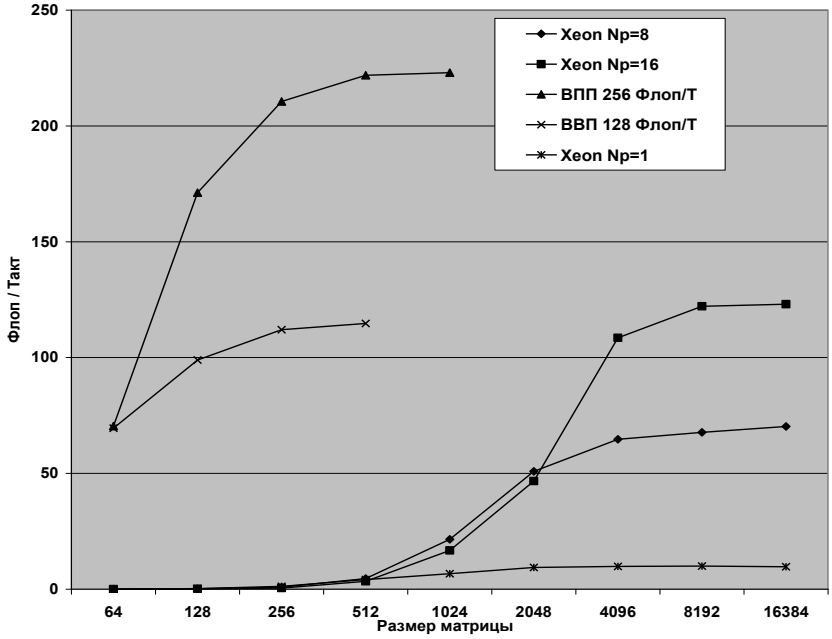

Рис. 4. Зависимость производительности от размера матрицы

Таким образом, разрабатываемый ВПП за счет более высокой степени векторизации программ и за счет поиска готовых команд в гораздо большем окне по сравнению с процессорами традиционной архитектуры имеет возможность не только повысить производительность до 256 флоп в такт, но и сохранять ее при значительно меньших размерах обрабатываемых массивов.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. The Opportunities and Challenges of Exascale Computing // Summary Report of the ASCA committee, U.S. Department of Energy, Fall 2010.
2. *Дикарев Н.И., Шабанов Б.М.* Архитектура процессора и ее влияние на производительность суперЭВМ // Программные продукты и системы. – 2007. – № 2 (78). – С. 2-5.
3. *Arvind and R.S.Nikhil.* Executing a program on the MIT tagged-token data-flow architecture // IEEE Trans. Comput. – 1990. – Vol. C-39, № 3. – P. 300-318.
4. Manchester Dataflow Research Project. http://intranet.cs.man.ac.uk/cnc/ progects/dataflow.html.
5. *Sakai S. et al*. An Architecture of a Dataflow Single-Chip Processor // Proc. 16-th Ann. Symp. on Computer Architecture. – 1989. – P. 252-273.
6. *Papadopoulos G.V., Traub K.R.* Multithreading: A revisionist view of dataflow architectures // Proc. 18-th Ann. Symp. on Computer Architecture. – 1991. – P. 342-351.
7. *Culler D.E. and Arvind*. Resource requirements of dataflow programs // Proc. 15-th Ann. Symp. on Computer Architecture. – 1988. – P. 141-150.
8. The WaveScalar Architecture. http://wavescalar.sc.washington.edu.
9. TRIPS Processor Architecture. http://www.cs.utexas.edu/users/cart/trips.
10. *Hake J.F. and Homberg W*. The impact of memory organization on the performance of matrix calculations // Parallel Computing. – 1991. – Vol. 17, № 2/3. – P. 311-327.

REFERENCES

1. The Opportunities and Challenges of Exascale Computing, Summary Report of the ASCA committee, U.S. Department of Energy, Fall 2010.

2. *Dikarev N.I., Shabanov B.M.* Arkhitektura protsessora i ee vliyanie na proizvoditel'-nost' superEVM [Processor architecture and its influence on the performance of supercomputers], Programmnye produkty i sistemy [Software products and systems], 2007, No. 2 (78), pp. 2-5.
3. *Arvind and R.S.Nikhil.* Executing a program on the MIT tagged-token data-flow architecture, *IEEE Trans. Comput*., 1990, Vol. C-39, No. 3, pp. 300-318.
4. Manchester Dataflow Research Project. Available at: http://intranet.cs.man.ac.uk/cnc/progects/dataflow.html.
5. *Sakai S. et al*. An Architecture of a Dataflow Single-Chip Processor, *Proc. 16-th Ann. Symp. on Computer Architecture,* 1989, pp. 252-273.
6. *Papadopoulos G.V., Traub K.R.* Multithreading: A revisionist view of dataflow architectures, *Proc. 18-th Ann. Symp. on Computer Architecture,* 1991, pp. 342-351.
7. *Culler D.E. and Arvind*. Resource requirements of dataflow programs, *Proc. 15-th Ann. Symp. on Computer Architecture,* 1988, pp. 141-150.
8. The WaveScalar Architecture. Available at: http://wavescalar.sc.washington.edu.
9. TRIPS Processor Architecture. Available at: http://www.cs.utexas.edu/users/cart/trips.
10. *Hake J.F. and Homberg W*. The impact of memory organization on the performance of matrix calculations, *Parallel Computing,* 1991, Vol. 17, No. 2/3, pp. 311-327.

Статью рекомендовал к опубликованию д.т.н., профессор И.И. Левин.

Дикарев Николай Иванович – Межведомственный суперкомпьютерный центр РАН (МСЦ РАН); e-mail: nic@jscc.ru; 119991, Москва, Ленинский пр., 32а; тел.: 84959386144; зав. лабораторией; к.т.н.

Шабанов Борис Михайлович – e-mail: shabanov@jscc.ru; тел.: 84951373361; зам. директора; к.т.н.; доцент.

Шмелев Александр Сергеевич – e-mail: guest8993@rambler.ru; тел.: 84959386144; н.с.

Dikarev Nikolay Ivanovich – Joint Supercomputer Center of the Russian Academy of Sciences (JSCC RAS); e-mail: nic@jscc.ru; 32a, Leninsky Prospect, Moscow, Russia, 119991; phone: +74959386144; head of laboratory; cand. of eng. sc.

Shabanov Boris Mikhaylovich – e-mail: shabanov@jscc.ru; phone: +74951373361; deputy director; cand. of eng. sc.; associate professor.

Shmelev Aleksandr Sergeevich – e-mail: guest8993@rambler.ru; phone: +74959386144; researcher.

УДК 004.272.43

А.В. Колодзей

КОМПРОМИСС «ВРЕМЯ/ПАМЯТЬ» В РЕКОНФИГУРИРУЕМЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ

В настоящее время является актуальной задача создания проблемно-ориентированных вычислительных систем (ПОВС), то есть предназначенных для решения конкретных классов задач. Одним из таких классов задач является обращение функций при помощи проведенных заранее предварительных вычислений. Решается задача достижения баланса между вычислительной производительностью и пропускной способностью системы памяти при использовании метода так называемых радужных таблиц. Даются оценки времени решения и необходимого объема памяти в зависимости от сложности задачи и конфигурации вычислительной системы. Полученные оценки позволяют выбрать оптимальное значение одного их основных параметров метода радужных цепочек – длины цепочки L в зависимости от текущей конфигурации вычислительной системы. Рост сложности решаемых задач может требовать модернизации ПОВС. Оказывается, если модернизацию планируется вести «экстенсивным» образом, когда относительный рост числа вычислительных