ISSN 2079-3316 **ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ** **т. 11**, № 4(47), с. 55–71

ББК 3971.32-043:22.151.511
ГРНТИ 50.09.33, 50.33.04
УДК 004.272.25:004.272.44

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

Преимущества и недостатки использования метода векторов указателей в векторном потоковом процессоре

Аннотация. Статья посвящена анализу выполнения программы быстрой сортировки (QS) в векторном процессоре с архитектурой управления потоком данных, в котором для хранения массивов используется метод векторов-указателей. Анализируется выявленный на программе QS недостаток хранения массивов с помощью векторов указателей и предложен способ решения этого недостатка введением команд split и fuse в систему команд процессора. Несмотря на значительное усложнение графа и числа выполняемых команд в программе QS, введение в систему команд ВПП новых команд split и fuse позволило достичь на этой программе до 7.4 раз более высокой производительности по сравнению с процессорным ядром Intel Skylake.

Ключевые слова и фразы: векторный процессор, архитектура управления потоком данных, программа сортировки, параллелизм уровня команд, мелкоструктурный параллелизм, векторная производительность.

Введение

Одним из существенных недостатков процессора традиционной (фон-неймановской) архитектуры является невозможность повысить производительность одного ядра. Это проявляется в нецелесообразности как дальнейшего увеличения числа команд, выдаваемых на выполнение за один такт, так и тактовой частоты процессора [1]. Поэтому производительность суперкомпьютера можно повысить лишь путем увеличения числа процессорных ядер в нём, либо за счет использования ускорителей с ещё большим числом ядер. Это приводит к снижению реальной производительности высокопроизводительных систем на параллельных участках кода из-за увеличения накладных расходов на синхронизацию и обмен данными в системе с увеличением числа процессорных ядер, не говоря уже о последовательных участках кода.

Работа выполнена в МСЦ РАН в рамках Государственного задания по теме 0065-2019-0016. В исследованиях использовался суперкомпьютер МВС-10П.

В МСЦ РАН разрабатывается процессор с архитектурой управления потоком данных (более кратко потоковый процессор), в котором благодаря присущему этой архитектуре параллелизму выполнения команд реальная производительность процессорного ядра может быть увеличена в 8 - 10 раз [2].

Процесс выполнения команд в потоковом процессоре коренным образом отличается от процессора традиционной архитектуры, поскольку команды выполняются по готовности операндов, а не по порядку заданному компилятором или программистом в коде программы. Программой в потоковом процессоре является граф, в котором узлами являются команды, а дуги показывают на входы каких команд передать результаты выполненной команды [3]. Напомним, что в процессоре традиционной архитектуры программа - это список команд в оперативной памяти. Из неё команды читаются и идут на выполнение последовательно путем приращения адреса в счетчике команд.

Готовность операндов в потоковом процессоре определяется в специальной памяти — памяти совпадения, которая хранит приходящие на входы команды пакеты (токены) со значениями операндов до прихода последнего операнда. После чего пакет с готовыми операндами выдаётся из памяти совпадения на выполнение в исполнительном устройстве, и после вычисления результата в нём формируются токены для отправки результата на входы последующих команд, а операнды выполненной команды уничтожаются.

Уничтожение операндов приводит к увеличению числа выполняемых команд в потоковом процессоре, поскольку число токенов, создаваемых командой ограничено (обычно не более двух). Тогда, если результат команды нужно подать на входы более чем двух команд, приходится выполнять команды дублирования для размножения числа токенов. Такие команды дублирования составляют до трети от всех выполняемых команд в потоковом процессоре [4]. Однако это гарантирует отсутствие конфликтов информационной зависимости — основную причину снижения производительности в процессоре традиционной архитектуры. Средой для передачи операндов в этом процессоре являются регистры в регистровых файлах или слова в оперативной памяти, и перед выполнением команды нужно проверять их готовность. А именно, записаны ли результаты предыдущих команд по тем адресам, из которых читает операнды текущая команда.

Таким образом, наличие памяти совпадения в процессоре с архитектурой управления потоком данных позволяет выявлять параллелизм уровня команд в динамике (в процессе выполнения программы), и

одновременно является причиной тех недостатков, из-за которых ни один из проектов разработки потокового процессора не был доведён до промышленного выпуска. К ним относятся - слишком мелкий параллелизм (уровня отдельных команд), что приводит к увеличению числа циркулирующих токенов в цепях управления процессора, а повышение пропускной способности этой аппаратуры в N раз ведёт к росту аппаратных затрат от $N \log N$ до N^2 раз из-за наличия коммутаторов. Кроме того, прохождение через память совпадения увеличивает задержку выполнения команд, связанных между собой зависимостью по данным. Это приводит к низкой производительности потокового процессора на последовательных участках кода [4],[5]. Наконец, рост числа передаваемых токенов требует увеличения ёмкости памяти совпадения для их хранения, а это увеличивает аппаратные затраты потокового процессора и снижает его быстродействие.

Следует также учесть, что в большинстве проектов потокового процессора для построения памяти совпадения использовалась не обычная линейно адресуемая память, а ассоциативная память [3],[6]. Такая память позволяет упростить граф программы, поскольку поиск готовых команд можно вести не только по номеру команды в графе, но и по совпадению дополнительных признаков, таких как номер итерации в цикле. Однако это лишь усложняет реализацию памяти совпадения. Отметим, что перечисленные выше проблемы аппаратной реализации относятся в основном к проектам разработки универсального потокового процессора, поскольку специализированные процессоры, в частности, в области цифровой обработки сигналов не только разрабатывались, но и выпускались [7].

В разрабатываемом в МСЦ РАН векторном потоковом процессоре (ВПП) проблемы, с которыми сталкивались разработчики универсального процессора с архитектурой управления потоком данных, удалось в значительной степени решить благодаря введению векторной обработки и использованию метода векторов указателей для хранения массивов. В разделе 1 метод векторов указателей будет рассмотрен более подробно, как и те преимущества, которые даёт его использование в ВПП. В разделе 2 будет рассмотрена программа быстрой сортировки (QS), при разработке графа которой проявился недостаток метода векторов указателей. Будет показано, что его можно если не устранить совсем, то компенсировать введением в систему команд ВПП новых векторных команд, что будет подтверждено результатами моделирования.

1. Использование метода векторов указателей в ВПП

Введение векторных команд в систему команд процессора позволяет обрабатывать не одиночные значения данных, как при скалярной обработке, а одномерные массивы (вектора). Это позволяет в десятки раз уменьшить число выполняемых команд, и применительно к потоковому процессору - число циркулирующих в нём токенов [5]. Для хранения векторов в [5] используется обычное запоминающее устройство с произвольной выборкой. Вместо значения данных токены передают в память совпадения указатели векторов, содержащие адрес вектора в запоминающем устройстве (в памяти) и число элементов вектора (VL) для поиска готовых к выполнению векторных команд. Заметим, что, чем больше максимальное число элементов, обрабатываемых одной векторной командой, (аппаратная длина вектора VLmax), тем меньше векторных команд требуется для выполнения цикла по обработке элементов массива по сравнению с числом команд для выполнения того же цикла в скалярном режиме.

При этом в скалярном режиме нужно ещё учитывать накладные расходы по управлению циклом, то есть команды, осуществляющие подсчёт числа выполненных итераций и выход из цикла, в то время как в векторном режиме эти функции выполняет аппаратура. В результате, сокращение числа выполненных команд и применительно к потоковому процессору — числа циркулирующих токенов — может быть весьма значительным. Так, в ВПП аппаратная длина вектора VLmax равна 256 элементам по 64 разряда в каждом, соответственно число токенов на векторизуемых участках программы снижается в сотни раз.

В ВПП, так же как и в [5], для хранения векторов в памяти используются вектора фиксированного размера, равного аппаратной длине вектора. Однако в отличие от проекта [5], в ВПП используется не слитное расположение векторов в памяти, как это принято при традиционном способе хранении массивов, а аппаратное выделение по одному адресу вектора из списка неиспользуемых векторов для записи результата каждой выполненной векторной команды. Токены с указателем вектора результата передаются на входы последующих команд согласно графу программы, причем использовать вектор в качестве операнда можно многократно. Требуется лишь, чтобы та команда, которая будет читать вектор в последний раз, после выполнения вернула указатель вектора в список свободных векторов. Эта задача решается компилятором, который помечает вход команды признаком уничтожения вектора.

Такой оборот токенов с указателями векторов в потоковом процессоре максимально приближен к обращению токенов для скалярных

переменных, поскольку они создаются по мере вычисления результатов команд, и уничтожаются по мере их использования. Дополнительное преимущество такого решения — экономное расходование ресурса локальной памяти для хранения векторов, расположенной на процессорном кристалле, и имеющей сравнительно небольшую ёмкость, но высокое быстродействие.

Второе преимущество аппаратного распределения памяти — не нужно при создании массива выделять для него место в памяти требуемого размера, поскольку это медленный процесс, выполняемый операционной системой. При этом для хранения массивов, состоящих из многих векторов, в ВПП используется метод векторов указателей, при котором большие массивы данных хранятся в виде древовидной структуры из векторов, элементами которых являются указатели векторов подмассивов, как это показано на рисунке 2 для трехмерного массива. Здесь указатель трехмерного массива содержит в качестве элементов указатели матриц, которые, в свою очередь, содержат указатели строк, и лишь указатели строк — самого нижнего уровня этой структуры — содержат значения данных. Отметим, что вектора $A1$, $A2$, …, AN, показанные на рисунке 1, расположены не по последовательным адресам в памяти, как при традиционном способе хранения массивов, а это случайные адреса, выданные из списка неиспользуемых векторов.

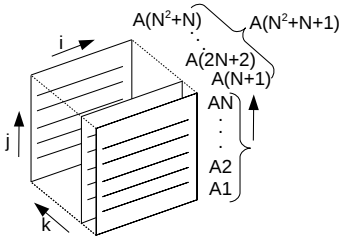

Рисунок 1. Трехмерный массив, представленный с помощью векторов указателей.

Конечно, такое решение помимо преимуществ имеет и свои недостатки. Так, вектора указатели, как и вектора с данными, занимают место в памяти, и ёмкость памяти должна быть больше по сравнению с традиционным способом хранения массивов. Однако применительно к ВПП такое увеличение ёмкости можно считать пренебрежимо малым. Действительно для матрицы, состоящей из векторов $A1$, $A2$, …, AN, как это показано на рисунке 1, в памяти будет выделено место под указатель матрицы $A(N+1)$, но при $N = 256$ увеличение ёмкости составит 0,4%, и для трехмерного массива — примерно ту же величину.

Другой недостаток — это увеличение времени выборки одиночных элементов массива, поскольку для выборки конкретного элемента требуется выполнить несколько последовательных обращений к памяти вместо нескольких арифметических операций для вычисления адреса элемента при традиционном способе хранения многомерных массивов. Однако при выполнении научно-технических задач, требующих большого объёма операций с плавающей запятой, обращения к одиночным элементам массивов встречаются редко, и этот недостаток также можно считать несущественным.

Отметим ещё одно преимущество использования метода векторов указателей в потоковом процессоре. С помощью команды «Формирование потока» (ФП) можно сформировать поток токенов из содержащихся в векторе указателе матрицы элементов, которыми являются указатели векторов строк этой матрицы, и использовать их в качестве операндов для выполнения векторной операции над всеми строками матрицы. При этом в ВПП не требуются команды по организации цикла, осуществляющие вычисление адресов для выборки каждого из векторов строк матрицы, которые необходимы при традиционном способе хранения массивов.

Аналогичное решение, снижающее число команд при выполнении циклов, использовалось в Манчестерском проекте потоковой машины (MDFM) [8] и в проекте [5], прототипом которого является MDFM. В [5] это векторная команда PRLV, формирующая поток последовательных адресов для чтения векторов, составляющих массив, из памяти в векторные регистры и последующего выполнения над ними арифметических операций. При этом, как и у команды ФП в ВПП, в [5] поток токенов, формируемый командой PRLV, имеет разные значения индекса в контексте токена, что позволяет одной командой в графе выполнять заданную операцию во всех итерациях цикла.

Использование векторных команд и метода векторов указателей в ВПП позволило на программе умножения матриц уменьшить число токенов в памяти совпадения в несколько сотен раз по сравнению со скалярной MDFM. Так, при выполнении этой программы с размером матрицы 35x35 максимальное число токенов в памяти совпадения MDFM составило 39500 [8], в то время как в ВПП — 550 при размере матрицы 128x128. При этом реальная производительность ВПП на этой программе при размере матрицы 128x128 составляет 108 флоп в такт или 84% от пиковой производительности 128 флоп в такт. Заметим, что это в десятки раз больше чем у вычислительного узла из 16 ядер процессора Intel Xeon ES-2690 с той же пиковой производительность [2].

Даже по сравнению с векторными процессорами традиционной архитектуры метод векторов указателей позволяет уменьшить число выполняемых команд в ВПП в 2 — 3 раза, причем в основном за счет сокращения числа скалярных команд [9]. Это позволяет повысить производительность векторной обработки в ВПП по отношению к скалярной производительности без потери эффективности выполнения программ [2].

К сожалению, разработка графа программы QS выявила недостаток метода векторов указателей, связанный с разбиением массива на отдельные вектора вместо слитного представления массивов в памяти при традиционном способе хранения массивов.

2. Программа быстрой сортировки и её выполнение в ВПП

Выполнение программы QS основано на разделении сортируемого массива на блоки, что достигается сравнением его элементов со значением ведущего элемента или порога. Элементы сортируемого массива сначала с помощью векторных команд чтения перемещаются из памяти в регистровый файл кусками, равными аппаратной длине вектора, и затем сравниваются со значением порога. По вектору маски — результату векторной команды сравнения производится разделение исходных векторов на части, состоящие из элементов меньше или равных порогу, и — больших порога, как это показано на рисунке 2 для двух векторов. Далее производится соединение частей вектора из элементов, отмеченных «1» (и аналогично «0») в векторе маски, и перемещением этих частей из регистрового файла в память и записью по последовательным адресам. В результате, в памяти формируются два блока из элементов исходного массива, разделенных на части с помощью ведущего элемента.

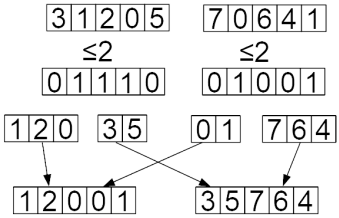

Рисунок 2. Разделение исходного массива на части с помощью ведущего элемента.

В процессорах Intel с векторным расширением AVX-512 есть специальные команды записи в память элементов вектора, отмеченных

«1» в векторе маске. Запись результатов этих команд при их применении к векторам в регистровом файле производится в оперативную память по последовательным адресам, образуя в памяти непрерывный массив из чисел меньше или равных порогу.

Аналогично записываются в память и элементы исходного вектора со значениями больше порога. При этом нет необходимости, как при использовании метода векторов указателей, вводить специальную векторную команду, которая разделяют вектор на части, и другую — для объединения элементов двух коротких векторов в один длинный вектор.

Такие команды split, разделяющие вектор на два коротких вектора по вектору маски (второму операнду команды), и fuse — «сливающие» два вектора в один, были введены в систему команд ВПП, а VHL описание блока выполнения специальных операций в ВПП было дополнено соответствующей аппаратурой. Для разделения пришлось использовать две команды split1 и split0, различающиеся сборкой по «1» и «0» в векторе маске. (Использовать одну команду split с созданием двух векторов в принципе возможно, но нужно будет менять управление в блоке выполнения специальных операций для записи двух векторов результатов одной векторной команды.)

Следует подчеркнуть, что описанный выше один проход по разделению массива на два блока в программе QS повторяется многократно, и после каждого прохода число блоков становится в 2 раза больше, а число элементов в каждом блоке примерно в 2 раза меньше. Такое дробление массива приводит к уменьшению длины вектора, и после некоторого числа проходов векторная обработка перестаёт быть эффективной.

В работе [**10**] было показано, что при малых размерах массива более эффективной является программа битонной сортировки, которая упорядочивает элементы массива, используя векторные команды сравнения и перестановки. Поскольку число операций сравнения и перестановки в этой программе растёт как $O(N \log^2 N)$, где N — размер сортируемого массива, то при больших значениях N лучше использовать программу быстрой сортировки. В результате, более высокую производительность имеет комбинированная программа QS, которая после разделения входного массива на блоки достаточно малого размера программой быстрой сортировки, затем производит сортировку элементов в каждом блоке с помощью битонной сортировки. Именно для этой комбинированной программы QS в [**10**] были приведены результаты для процессорного ядра Intel Skylake.

В [2] мы сравнили эти результаты с производительностью ВПП на программе битонной сортировки, показав, что одно ядро ВПП имеет значительно более высокую производительность по сравнению с Intel Skylake при сортировке массива из 4K чисел с плавающей запятой. Далее мы покажем, что и эти результаты можно улучшить, используя тот же комбинированный подход для выполнения сортировки в ВПП.

На рисунке 3 приведена блок схема графа программы быстрой сортировки, которая в ВПП выполняет один проход по разделению массива сортируемых данных на блоки с помощью команд split1, split0 и fuse.

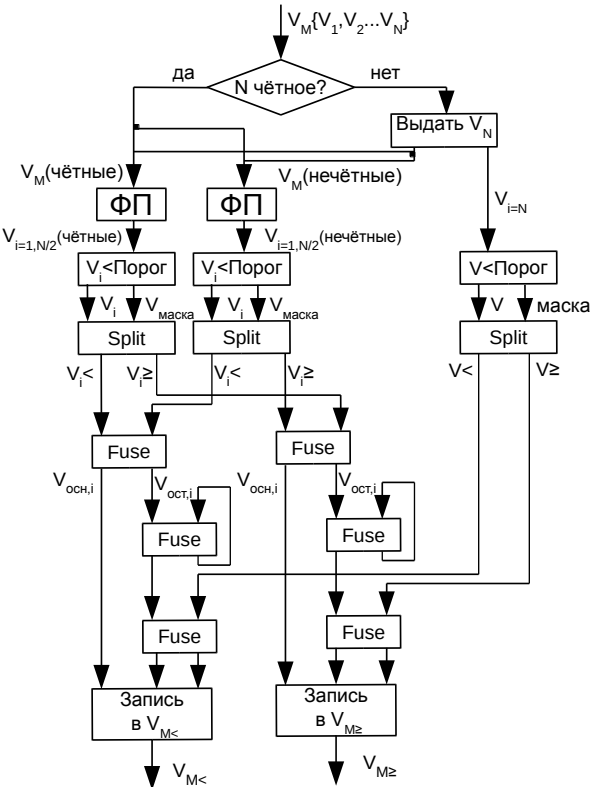

Рисунок 3. Блок схема графа программы по разделению массива на блоки с использованием команд split и fuse.

Считалось, что сортируемый массив находится в локальной памяти ВПП и состоит из N векторов, указатели которых являются элементами

вектора указателя массива Vм на входе графа. При моделировании длина векторов Vi задавалась равной аппаратной длине вектора VLmax, причём для упрощения алгоритма считалось, что число векторов N в сортируемом массиве не превышает VLmax.

Как видно из рисунка 3, обработка векторов ведётся не по одному, а парами. Цель обработки векторов парами — не допускать уменьшения длины вектора VL за счет объединения двух коротких векторов после разделения командой split в один длинный вектор (возможно с остатком) с помощью команды fuse. Для этого исходный массив Vм разделяется на два подмассива, и токены с указателями сортируемых векторов, созданные с помощью команды ФП из указателя подмассива, поступают на обработку в свою ветвь графа. В каждой ветви токены с указателями сортируемых векторов, сформированные командой ФП, поступают на вход команды векторного сравнения и далее на вход команды split, которая с помощью вектора маски (результату команды сравнения) разделяет вектор на два вектора меньшего размера. Указатель вектора на первом выходе команды split в блок схеме на рисунке 3 содержит элементы меньше порога, на втором выходе — больше или равные порогу.

Чтобы не усложнять рисунок 3, на нём показана одна команда split, просто первый выход этой команды на самом деле является выходом команды split1, а второй выход — split0. Вектора с выхода команды split1 из двух ветвей обработки с одинаковыми индексами i, сливаются командой fuse с целью получить один вектор исходной длины VLmax. Те же операции выполняются и для токенов, созданных командой split0.

Конечно, при случайных значениях элементов у сортируемых векторов это происходит нечасто. Чаще оказывается, что сумма длин, сливаемых командой fuse векторов меньше аппаратной длины вектора VLmax, либо больше VLmax. Здесь нужно пояснить, как работает команда fuse.

Эта команда присоединяет к элементам первого вектора операнда элементы второго операнда, причем в обратном порядке, то есть, начиная с элемента с индексом VL-1. Тогда, если сумма длин, сливаемых командой fuse векторов, больше VLmax, то с первого выхода этой команды будет выдан токен с адресом первого вектора операнда и VL=VLmax, который на рисунке 3 обозначен как основной вектор Vосн, а со второго выхода — токен вектора остатка Vост. Указатель вектора остатка Vост, содержащий остаток элементов второго из сливаемых векторов, не вошедших в основной вектор, имеет тот же адрес, что и у второго вектора операнда. Заметим, что при переносе

в обратном порядке элементов второго операнда у команды fuse начальные элементы этого вектора остаются на своих местах, и нужно лишь установить новую длину в указателе вектора остатка Vост, равную сумме длин, сливаемых векторов, минус VLmax.

Если же сумма длин, сливаемых командой fuse векторов, меньше или равна VLmax, то длина основного вектора Vосн будет равна сумме длин сливаемых векторов. При этом токен со второго выхода команды fuse нужно уничтожить, то есть адрес второго из сливаемых векторов отправить в список неиспользуемых векторов с помощью команды уничтожить вектор.

В конце прохода выполняются команды записи, которые указатели основных векторов Vосн,i с первого выхода команды fuse записывают в качестве элементов в указатель массива результата. Для левой ветви графа, как это показано на рисунке 3, запись производится в указатель массива VM<, предварительно выданный из списка свободных векторов. И аналогично для правой ветви — в указатель массива VM>=.

Вектора остатки Vост,i в каждой ветви графа при их наличии сливаются дополнительной командой fuse ещё в один вектор, указатель которого поступает на первый вход ещё одной команды fuse. Эта команда присоединяет к вектору, содержащему сумму остатков, результат разделения вектора V_N командой split, в случае если в указателе исходного массива Vм число векторов N было нечётным. Тогда вектор V_N, являющийся последним элементом в указателе исходного массива Vм, не находит себе пары и обрабатывается отдельно. При этом токены результаты с первого и второго выхода последней команды fuse в каждой из ветвей графа также записываются в качестве элементов в указатель массива VM< (VM≥) с номерами $N/2+1$ и $N/2+2$ соответственно.

При выполнении первого прохода при чётном числе векторов N в сортируемом массиве вектор V_N отсутствует, последняя команда fuse не выполняется, и число векторов в каждом из подмассивов VM< и VM≥ равно $N/2+1$. Тогда в следующем проходе разделяемые массивы будут состоять из нечётного числа векторов. При равномерном распределении чисел в сортируемом массиве, как показало моделирование, последняя команда fuse не формирует вектор остатка Vост. Тогда в каждом из последующих проходов число подмассивов на выходе рассматриваемого графа увеличивается вдвое относительно входа, и каждый из них содержит в два раза меньше указателей основных векторов с числами меньше и больше порога соответственно, и в дополнение к ним ещё по одному указателю вектора из чисел того же диапазона.

Если число векторов N в сортируемом массиве равно 2^k, то после k проходов число подмассивов становится равным N, а число основных векторов Vосн в каждом из них уменьшается до одного. В дополнение к нему указатель каждого подмассива содержит ещё один дополнительный вектор. Этот вектор состоит в основном из элементов, полученных в результате разделения вектора V_N командой split. Как уже отмечалось выше, вектор V_N является последним элементом в указателе подмассива уже после первого прохода в рассматриваемой программе, и этот вектор, как это показано на рисунке 3, не находит себе пары и обрабатывается отдельно.

Тогда после выполнения k проходов по рекурсивному разделению массива из N векторов на блоки (подмассивы) в результате сравнения с ведущим элементом, на выходе из программы быстрой сортировки будет N блоков, содержащих два указателя вектора Vосн и Vост в каждом блоке. Далее производится объединение элементов векторов Vосн и Vост в каждом блоке и разделение блоков на блоки из одного Vосн и блоки из двух векторов (Vосн и Vост) после проведенного объединения. Затем в каждом из блоков элементы в пределах вектора упорядочиваются по величине с помощью программы битонной сортировки, и завершается выполнение программы QS упорядочиванием элементов векторов Vосн и Vост в тех блоках, которые содержат вектора остатки. Такое упорядочивание элементов в каждом из двух векторов выполняется с помощью битонной сортировки, и все указатели векторов с упорядоченными по возрастанию значениями элементов в них упаковываются в вектор указатель отсортированного массива.

Описанная комбинированная программа сортировки QS была отлажена, и результаты её выполнения на VHDL модели ВПП в зависимости от размера массива приведены на рисунке 4. Там же для сравнения приведены значения производительности ВПП на программе битонной сортировки из [**2**] и производительности процессорного ядра Intel Skylake из [**10**].

Значения производительности измерены в тактах общего времени выполнения программы, приходящихся на один элемент массива. При этом, чем меньше число тактов удельного времени, тем выше производительность. Как видно из данных, приведенных на рисунке 4, производительность ВПП на программе QS превышает производительность ВПП на программе битонной сортировки на размерах массива больше, чем 2K чисел с плавающей запятой (или 8 векторов по 256 элементов). При этом разрыв в производительности одного ядра ВПП по сравнению с Intel Skylake растет с увеличением размера массива, и

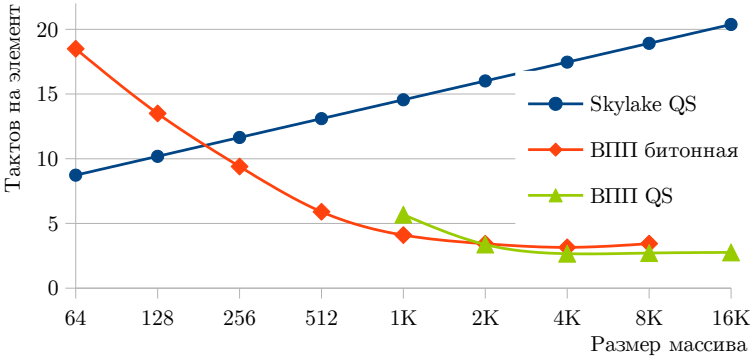

Рисунок 4. Производительность одного ядра ВПП и Intel Skylake на программе QS в зависимости от размера сортируемого массива

на размере в 16K чисел достигает 7.4 раза. После этого эффект от увеличения загрузки исполнительных устройств в ВПП с ростом объёма выполняемых операций исчерпывается, и производительность ВПП приближается к максимальной. Дальше удельное время должно начать расти и выйти на тот же темп, что у Skylake. При моделировании выйти на него не удалось, поскольку при размере массива 32K чисел (128 векторов по 256 элементов) была превышена аппаратная длина вектора у команды fuse, производящей суммирование остатков, что не было предусмотрено при разработке алгоритма.

Заключение

Несмотря на значительное усложнение графа и числа выполняемых команд в программе QS, введение в систему команд ВПП новых команд split и fuse позволило достичь на этой программе до 7.4 раза более высокой производительности по сравнению с процессорным ядром Intel Skylake.

Отметим что, эти результаты моделирования ВПП получены на первоначальном варианте программы QS, который не только не подвергался оптимизации, но и не работает на массивах с числом элементов больше 16K. Доработка планируется с целью повысить производительность ВПП за счет оптимизации времени выполнения программы, а также исследовать его работу на разных типах и уровнях параллелизма, включая параллелизм крупных блоков.

Список литературы

[1] Д. Л. Хеннеси, Д. А. Паттерсон. *Компьютерная архитектура. Количественный подход*, ред. А. К. Ким, Техносфера, М., 2016, ISBN 978-5-94836-413-1, 936 с. ↑55

[2] Н. И. Дикарев, Б. М. Шабанов, А. С. Шмелёв. «Мелкоструктурный параллелизм и более высокая производительность процессорного ядра: преимущества векторного потокового процессора», *Программные системы: теория и приложения*, **10**:4(43) (2019), с. 201–217. ↑56,60,61,63,66

[3] Arvind, R. S. Nikhil. "Executing a program on the MIT tagged-token data-flow architecture", *IEEE Transactions on Computers*, **39**:3 (1990), pp. 300–318. ↑56,57

[4] G. V. Papadopoulos, K. R. Traub. "Multithreading: A revisionist view of dataflow architectures", *Proc. 18-th Ann. Symp. on Computer Architecture* (30 May 1991, Toronto, Canada), 1991, pp. 342–351. ↑56,57

[5] W. M. Miller, W. A. Najjar, A. P. Wim Böhm. "A model for dataflow based vector execution", *Proceedings of the 8th international conference on Supercomputing*, ICS'94 (July 1994), 1994, pp. 11–22. ↑57,58,60

[6] A. R. Hurson, K. M. Kavi. "Dataflow computers: their history and future", Wiley Encyclopedia of Computer Science and Engineering, ed. B. Wah, John Wiley & Sons, Inc., 2008, 12 pp. ↑57

[7] H. Terada, S. Miyata, M. Iwata. "DDMP's: self-timed super-pipelined data-driven multimedia processors", *Proceedings of the IEEE*, **97**:2 (1999), pp. 282–296. ↑57

[8] J. Gurd, W. Bohm, Yong Meng Teo. "Performance issues in dataflow machines", *Future Generations Computer Systems*, **3**:4, pp. 285–297. ↑60

[9] Н. И. Дикарев, Б. М. Шабанов, А. С. Шмелёв. «Векторный потоковый процессор: оценка производительности», *Известия ЮФУ. Технические науки*, 2014, №12 (161), Тематический выпуск: Суперкомпьютерные технологии, с. 36–46. ↑61

[10] B. Bramas. "A novel hybrid Quicksort algorithm vectorized using AVX-512 on Intel Skylake", *International Journal of Advanced Computer Science and Applications*, **8**:10 (2017), pp. 337–344. ↑62,66

Поступила в редакцию	12.12.2020
Переработана	24.12.2020
Опубликована	28.12.2020

Рекомендовал к публикации *к.ф.-м.н. С. А. Романенко*

Пример ссылки на эту публикацию:

Н. И. Дикарев, Б. М. Шабанов, А. С. Шмелёв. «Преимущества и недостатки использования метода векторов указателей в векторном потоковом процессоре». *Программные системы: теория и приложения*, 2020, **11**:4(47), с. 55–71. 10.25209/2079-3316-2020-11-4-55-71
http://psta.psiras.ru/read/psta2020_4_55-71.pdf

Об авторах:

Николай Иванович Дикарев

Кандидат технических наук, ведущий научный сотрудник МСЦ РАН. Научные интересы: высокопроизводительные вычислительные системы, архитектура управления потоком данных, векторные процессоры, параллельная и векторная обработка.

0000-0002-7857-3250
e-mail: nic@jscc.ru

Борис Михайлович Шабанов

Доктор технических наук, доцент, лауреат Государственной премии Российской Федерации в области науки и техники. Директор МСЦ РАН.

0000-0002-5238-366X
e-mail: shabanov@jscc.ru

Александр Сергеевич Шмелёв

Научный сотрудник МСЦ РАН. Научные интересы: Суперкомпьютеры, архитектура управления потоком данных, векторные процессоры.

0000-0002-1941-7792
e-mail: guest8993@rambler.ru

CSCSTI 50.09.33, 50.33.04
UDC 004.272.25:004.272.44

Nikolay I. Dikarev, Boris M. Shabanov, Aleksandr S. Shmelev. *Advantages and disadvantages of using the pointer vector method in a vector dataflow processor.*

Abstract. The article is devoted to the analysis of the quick sort (QS) program execution in a vector dataflow processor (VDP), which uses the pointer vectors method to store arrays. The deficiency of pointer vectors method revealed in the QS program is analyzed and a method for solving this disadvantage is proposed by introducing split and fuse commands into the processor instruction set. Despite the significant complication of the graph and increase the overall number of commands executed in the QS program, the introduction of new split and fuse commands into VDP command system made it possible to achieve up to 7.4× faster performance than Intel Skylake processor core.

Key words and phrases: vector processor, dataflow architecture, sorting program, instruction level parallelism, fine-grained parallelism, vector performance.

2020 *Mathematics Subject Classification*: 65Y05; 68Q10, 08-04

References

[1] J. L. Hennessy, D. A. Patterson. *Computer Architecture: A Quantitative Approach*, 5th edition, Morgan Kaufmann, 2011, ISBN 978-8178672663, 856 pp. ↑55

[2] N. I. Dikarev, B. M. Shabanov, A. S. Shmelëv. "Fine-grained parallelism and higher core performance: advantages of vector dataflow processor", *Program Systems: Theory and Applications*, **10**:4(43) (2019), pp. 201–217 (in Russian). ↑56,60,61,63,66

[3] Arvind, R. S. Nikhil. "Executing a program on the MIT tagged-token data-flow architecture", *IEEE Transactions on Computers*, **39**:3 (1990), pp. 300–318. ↑56,57

[4] G. V. Papadopoulos, K. R. Traub. "Multithreading: A revisionist view of dataflow architectures", *Proc. 18-th Ann. Symp. on Computer Architecture* (30 May 1991, Toronto, Canada), 1991, pp. 342–351. ↑56,57

[5] W. M. Miller, W. A. Najjar, A. P. Wim Böhm. "A model for dataflow based vector execution", *Proceedings of the 8th international conference on Supercomputing*, ICS'94 (July 1994), 1994, pp. 11–22. ↑57,58,60

[6] A. R. Hurson, K. M. Kavi. "Dataflow computers: their history and future", Wiley Encyclopedia of Computer Science and Engineering, ed. B. Wah, John Wiley & Sons, Inc., 2008, 12 pp. ↑57

[7] H. Terada, S. Miyata, M. Iwata. "DDMP's: self-timed super-pipelined data-driven multimedia processors", *Proceedings of the IEEE*, **97**:2 (1999), pp. 282–296. ↑57

[8] J. Gurd, W. Bohm, Yong Meng Teo. "Performance issues in dataflow machines", *Future Generations Computer Systems*, **3**:4, pp. 285–297. ↑60

[9] N. I. Dikarev, B. M. Shabanov, A. S. Shmelëv. “Fast sorting algorithms for vector dataflow processor”, *Izvestiya YuFU. Tekhnicheskiye nauki*, 2014, no. 12 (161), Tematicheskiy vypusk: Superkomp’yuternyye tekhnologii, pp. 36–46 (in Russian). ↑61

[10] B. Bramas. “A novel hybrid Quicksort algorithm vectorized using AVX-512 on Intel Skylake”, *International Journal of Advanced Computer Science and Applications*, **8**:10 (2017), pp. 337–344. ↑62,66

Sample citation of this publication:

Nikolay I. Dikarev, Boris M. Shabanov, Aleksandr S. Shmelev. “Advantages and disadvantages of using the pointer vector method in a vector dataflow processor”. *Program Systems: Theory and Applications*, 2020, **11**:4(47), pp. 55–71. (*In Russian*).

10.25209/2079-3316-2020-11-4-55-71

http://psta.psiras.ru/read/psta2020_4_55-71.pdf