ISSN 2079-3316 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ т. 12, № 4(51), с. 65–83

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 (Quicksort) в векторном процессоре с архитектурой управления потоком данных (ВПП), в котором для хранения массивов используется метод векторов-указателей. Выявленный ранее на программе Quicksort недостаток хранения массивов с помощью векторов указателей был компенсирован введением команд split и fuse в систему команд процессора. Анализируется усовершенствованный граф программы Quicksort и результаты её моделирования на исходной и модернизированной системе команд ВПП. Производится сравнение результатов моделирования ВПП с производительностью процессорного ядра IntelSkylake.

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

Введение

Процессор с архитектурой управления потоком данных (потоковый процессор) в отличие от современных процессоров фон-неймановской архитектуры не имеет счётчика команд, и порядок их выполнения определяется целиком готовностью операндов у каждой команды [1]. Программой в таких процессорах служит граф, в котором команды являются узлами графа, а их результаты передаются по дугам на входы последующих команд с помощью токенов — пакетов, содержащих значение результата, номер команды приёмника и, как правило, ещё несколько полей, называемых контекстом. Эти поля позволяют искать

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

10.25209/2079-3316-2021-12-4-65-83

операнды для команд в разных итерациях цикла для их параллельного выполнения.

Готовность команд определяется в памяти совпадения потокового процессора, и приход последнего токена операнда приводит к выдаче команды из памяти совпадения на выполнение в исполнительное устройство (ИУ). При этом токены операнды после вычисления результата в ИУ уничтожаются, а значения результата, как отмечалось ранее, передаются по дугам графа на входы последующих команд. Это исключает в потоковом процессоре конфликты зависимости по данным RAW — основную причину снижения реальной производительности современных процессоров. Заметим лишь, что современные процессоры, выполняющие до 6 команд за один такт, используют аналогичный прием для повышения реальной производительности — выполнение команд по готовности операндов (out-of-order), а не по порядку, заложенному в программе (in-order). Только в процессорах традиционной архитектуры окно, в котором производится поиск готовых команд, составляет 80 — 200 команд [2], а в потоковом процессоре это вся программа.

За счёт поиска параллелизма во всей программе производительность потокового процессора может многократно превосходить производительность процессора традиционной архитектуры. Однако большой объем параллелизма уровня отдельных команд в потоковом процессоре приводит к необходимости использовать для передачи токенов массивно параллельную систему, состоящую из многих ИУ и модулей памяти совпадения. Например, японская потоковая ЭВМ Sigma-1 содержала 128 ИУ и 128 модулей памяти совпадения [3]. В результате основными проблемами создания конкурентоспособных ЭВМ с архитектурой управления потоком данных являются:

- Слишком мелкий параллелизм команд, который из-за большого объёма требует использовать в сети передачи токенов коммутаторы с большим числом входов и выходов.
- Для поиска команд с готовыми операндами требуется память совпадения большой ёмкости, в качестве которой в большинстве проектов потоковых ЭВМ использовалась ассоциативная память (АП) [3–5].
- Наконец, наличие коммутаторов и модулей АП увеличивает время при выполнении команд, связанных зависимостью по данным [5].

В разрабатываемом в МСЦ РАН векторном потоковом процессоре (ВПП) эти недостатки удалось преодолеть за счёт векторной обработки и использования оригинального метода хранения массивов данных — метода векторов-указателей. Программа быстрой сортировки (Quicksort) выявила недостаток метода векторов-указателей, связанный с распределённым характером хранения массивов в памяти ВПП [6]. Для его устранения в систему команд ВПП были введены split1, split0 и fuse. Моделирование начального варианта графа программы Quicksort выявило недостаток, проявившийся при увеличении размера сортируемого массива: неработоспособность на массивах из более 16 КиБ чисел [6].

В [7] был кратко описан усовершенствованный граф этой программы, устраняющий этот недостаток, и приведены результаты его моделирования. При этом использовалась та же неоптимизированная система команд, что и в [6]. А именно, бит наличия вектора на выходе команды fuse вычислялся с помощью скалярных команд, и не включался в состав указателя вектора, что увеличивало число выполняемых команд и задержку формирования бита наличия вектора.

Вклад статьи состоит в следующем:

- Предложено включить бит наличия вектора в состав указателя вектора, что позволило не только снизить задержку формирования бита, но и аппаратно обходить команду fuse в случае отсутствия вектора на одном из её входов.
- Дается более подробное описание как исходного, так и нового алгоритма разделения сортируемого массива на блоки в программе Quicksort.
- Предложено вести постраничную запись результатов чтения вектора по маске в сдвоенной команде split одновременно для элементов, отмеченных «1» и «0», в расслоённый на 32 модуля буфер для последующей записи результатов команды по последовательным адресам с разных сторон. Такое решение позволило вести постраничную запись элементов плотным массивом и избежать конфликтов при пересылке считанных данных через коммутатор. В результате пропускная способность сдвоенной команды split повысилась вдвое по сравнению с использованием команд split1 и split0.

Статья имеет следующую структуру: В разделах 1 и 2 преимущества и недостатки ВПП будут рассмотрены более подробно по результатам выполнения исходного и усовершенствованного вариантов графа

Quicksort с использованием исходной системы команд. В разделе 3 будет обоснована оптимизация системы команд ВПП, направленная на введение дополнительных функций при выполнении уже имеющихся команд в системе команд ВПП и использование вместо двух команд split1 и split0 одной сдвоенной команды split. Повышение производительности будет подтверждено результатами моделирования программы Quicksort в ВПП и приведено сравнение производительностей Intel Skylake и ВПП. Основные показатели достигнутого ускорения приводятся в Заключении.

1. Преимущества и недостатки ВПП по результатам моделирования программы Quicksort

Система команд ВПП помимо скалярных команд содержит ещё и векторные команды, которые выполняют однотипные операции над элементами векторов операндов. Векторная обработка позволяет значительно уменьшить как число команд, выполняемых в потоковом процессоре, так и число токенов, поступающих в АП [5]. В векторном процессоре с архитектурой управления потоком команд [5], также как и в ВПП, вектора хранятся в обычном запоминающем устройстве с выборкой по адресу, но вместо скалярных данных токены передают в АП указатели векторов для поиска готовых к выполнению векторных команд. Как в ВПП, так и в проекте [5], обработка длинных одномерных массивов ведётся фрагментами фиксированной длины, равной аппаратной длине вектора VLmax, которая в ВПП составляет 256 слов по 8 байт. Однако, если в [5] используется традиционный способ хранения массивов в памяти, при котором за выделение места в памяти под массив произвольного размера, также как и его удаление, отвечает операционная система, то в ВПП эта задача решается аппаратурой процессора.

В ВПП локальная память каждого процессорного ядра ёмкостью 1–2 МиБ, также как и общая оперативная память разбиты на фрагменты по 2 КиБ, каждый из которых хранит VLmax элементов одного вектора. В процессорном ядре ВПП есть аппаратура для ведения списка свободных (неиспользуемых) векторов в локальной памяти ядра, которая выдаёт адрес свободного вектора для записи результата, поступившей на выполнение векторной команды. Токены с указателем этого вектора передаются на входы тех команд, которые согласно графу программы будут читать его элементы в качестве операнда.

В потоковом процессоре токены операнды с указателем вектора в поле данных, также как и токены со скалярной переменной, уничтожаются после выполнения команды в ИУ. Если же значение операнда нужно подать на входы нескольких команд, то токен операнда, будь то вектор или скаляр, нужно размножить командой «дублирование». Однако в отличие от векторного процессора [5], удаление вектора из памяти (возвращение в список свободных в ВПП) производится после того как он будет прочитан в качестве операнда в последний раз, а не вместе с массивом, в котором он находится. Для этого может использоваться команда «уничтожить вектор» (УВ), но чаще операнд векторной команды просто помечается признаком уничтожения. Такой порядок работы с векторами позволяет приблизить процесс создания и уничтожения токенов с указателями вектора к принципу работы с токенами для скалярных переменных. Кроме того, это позволяет экономно расходовать ресурс локальной памяти в ВПП, выделяя вектора по мере необходимости и возвращая их в список свободных по мере использования [8].

Для хранения массивов, состоящих из многих векторов в ВПП, был предложен метод векторов-указателей, при котором указатели векторов, элементами которых являются данные, например строки матрицы, хранятся в качестве элементов вектора-указателя матрицы. Такую древовидную структуру можно использовать для хранения любых массивов данных. При этом из элементов вектора-указателя матрицы можно командой «формирование потока» (ФП) сформировать токены, содержащие указатели всех её строк, и использовать их для выполнения в цикле операций над строками матрицы. Заметим, что при традиционном способе хранения массивов в каждой итерации цикла помимо полезных операций над строками матрицы должны быть команды по организации цикла и вычислению адресов для векторов строк матрицы. Использование команд ФП позволяет исключить эти скалярные команды и в 2–3 раза уменьшить число выполняемых команд в ВПП, повысив долю векторных вычислений и, следовательно, эффективность векторной обработки в ВПП [9]. Моделирование программ умножения матриц, Stencil и битонической сортировки подтвердило более высокую реальную производительность по отношению к пиковой производительности у ВПП по сравнению с процессором Intel Skylake несмотря на то, что производительность одного ядра ВПП 256 флоп в такт, а у Intel Skylake — 32 флоп в такт [10].

Однако программа быстрой сортировки (Quicksort) выявила недостаток метода векторов-указателей, связанный с распределённым характером хранения массивов в памяти ВПП [6]. Программа Quicksort выполняет рекурсивное разделение сортируемого массива на блоки, уменьшая после каждого прохода размер блока вдвое. Для разделения массива используют векторные команды, производящие сравнение элементов вектора со значением ведущего элемента (порога) и формированием вектора маски, в котором «1» отмечены элементы меньше порога и «0» — больше или равные порогу. Затем под управлением вектора маски осуществляется разделение элементов исходного массива на два подмассива. В процессорах Intel с расширением AVX-512 для такого разделения есть команды записи в память тех элементов, которые отмечены «1» в векторе маске [11]. При этом несколько таких команд производят запись элементов по последовательным адресам, формируя в памяти два блока из элементов меньше порога и больше или равных порогу.

Для разделения элементов вектора по маске в систему команд ВПП были введены команды split1 и split0, которые из исходного вектора создают вектор из элементов, отмеченных «1» и «0» в векторе маски соответственно. Кроме того, чтобы не допускать образования всё более коротких векторов $\mathrm{V}_{<}$ и $\mathrm{V}_{\geqslant}$ после выполнения команд split1 и split0 в систему команд ВПП была введена команда fuse, которая присоединяет к элементам одного короткого вектора элементы другого вектора.

На рисунке 1 показан потоковый граф первого прохода по разделению на блоки в программе Quicksort с использованием команд split и fuse, из которого видно, что процесс разделения значительно сложнее и требует выполнения большего числа команд из-за невозможности слитного хранения массивов в памяти ВПП. На вход графа поступает вектор-указатель сортируемого массива, элементами которого являются указатели N векторов, подлежащих сортировке. Обработка ведётся в двух ветвях графа по $N/2$ векторов в каждой. Вектора результаты команд split1 $(\mathrm{V}_{<}, i)$ с индексами $i = 1, ..N/2$ из двух ветвей объединяются одной командой fuse, результаты команд split0 $(\mathrm{V}_{\geqslant}, i)$ — другой командой fuse. В результате, в левой ветви графа продолжают обрабатываться элементы меньше порога, в правой ветви — больше или равные порогу [6].

Дальнейшая обработка в каждой из ветвей графа заключается в анализе числа элементов у векторов, объединяемых командой fuse.

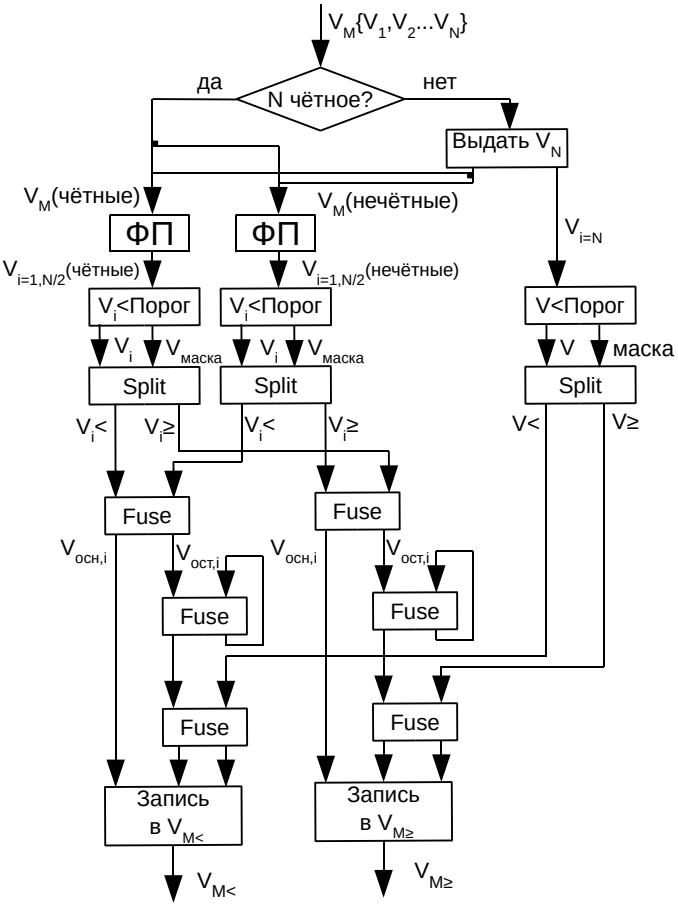

Рисунок 1. Потоковый граф разделения массива на блоки программой Quicksort для ВПП

Такой анализ производится скалярными арифметическими командами и командами сравнения, не показанными на рисунке 1. Если сумма числа элементов (длин векторов) у объединяемых командой fuse векторов больше аппаратной длины VLmax, то на правом выходе указатель вектора остатка $V_{ост}$, будет иметь длину, равную сумме длин векторов у операндов команды fuse минус VLmax. В противном случае остатка нет, и указатель вектора остатка $V_{ост}$ должен быть уничтожен командой УВ.

Указатели основных векторов $\mathrm{V}_{\mathsf{осн},i}$ с левого выхода команды fuse в каждой ветви, вне зависимости от того являются они полными или неполными, идут на запись в указатель массива результата $\mathrm{VM}_{<}$ и $\mathrm{VM}_{\geqslant}$ соответственно. А элементы векторов остатков ($\mathrm{V}_{\mathsf{ост},i}$), если такие вектора имеются, объединяются дополнительной командой fuse.

В рассматриваемом графе суммирование остатков показано упрощённо, поскольку для правильной работы двухвходовой команды, какой является команда fuse, в потоковом процессоре нужно либо подавать токены на оба входа, либо не подавать ни на один. Поэтому в графе есть блок, который начиная с первой итерации, анализирует, есть или нет ($\mathrm{V}_{\mathsf{ост},i}$). Первый пришедший вектор остаток запоминается для суммирования в следующей итерации с $i = i + 1$, и далее к нему присоединяются элементы последующих остатков в тех итерациях, когда они есть. Таким образом, не показанный блок управления подаёт на входы команды fuse сумму остатков и очередной остаток лишь в тех итерациях, когда вектор остаток формируется первой командой fuse.

Для такого подключения операндов к входам команды fuse используют команду «передача по условию» (ПУ). ПУ передаёт токен с первого входа на первый выход, если бит управления на втором входе команды равен «1», и на второй выход — если «0». После выхода из цикла по i вектор с накопленной суммой остатков сливается с помощью команды fuse с результатом разделения командами split1 и split0 вектора VN — последнего в массиве, если число векторов N было нечётным. Таким образом, в результате каждого прохода графа на рисунке 1 исходный массив VM делится на два блока $\mathrm{VM}_{<}$ и $\mathrm{VM}_{\geqslant}$, каждый из которых содержит $(N/2) + 1$ вектор из чисел меньше порога и больше или равных порогу.

Второй проход по разделению на блоки ничем не отличается от первого, только разделять нужно не один, а два блока и выставить для каждого из них своё значение порога. В ВПП, как и в большинстве проектов потоковых ЭВМ, для поиска команд с готовыми операндами использовалась ассоциативная память, которая позволяет искать совпадение не по одному номеру команды К в графе, а по контексту из нескольких целых чисел. В ВПП контекст помимо К содержит числа П, Т, и И. Поле И содержит номер итерации внутреннего цикла i — номер вектора в блоке, Т — номер блока. Это позволяет одновременно выполнять команды над векторами с разными значениями И и Т, выявляя максимальный параллелизм команд с готовыми операндами.

Если число векторов N в сортируемом массиве равно 2^k, то после k проходов число блоков на выходе становится равным N, и каждый из них содержит два вектора — $\mathrm{V}_{\mathsf{осн}}$ и $\mathrm{V}_{\mathsf{ост}}$ [6].

Моделирование сортировки массива из 32 КиБ чисел при равномерном распределении случайных чисел выявило ошибку в программе — накопленная сумма остатков превысила аппаратную длину вектора $\mathrm{VL}_{\max} = 256$, что не было предусмотрено первым вариантом графа программы. Соответственно потребовалась разработка усовершенствованного графа, устраняющего этот недостаток, который будет рассмотрен в следующем разделе.

После разделения на блоки, состоящие из двух векторов $\mathrm{V}_{\mathsf{осн}}$ и $\mathrm{V}_{\mathsf{ост}}$, выполненное программой Quicksort, слияние и упорядочивание элементов внутри блока производилось программой битонической сортировки. Такая комбинированная программа сортировки QS, как показало моделирование ВПП, превосходит битоническую сортировку по производительности, если число элементов в массиве больше, чем 1024 (4 вектора по 256 элементов) [6]. При этом производительность программы QS растёт с увеличением размера сортируемого массива, и при размере 16 КиБ элементов превосходит производительность ядра Intel Skylake в 7,4 раза на той же программе.

2. Усовершенствованная программа Quicksort и её выполнение в ВПП

Для устранения отмеченного выше недостатка, связанного с появлением вектора остатка $\mathrm{V}_{\mathsf{ост}}$ у команды fuse, суммирующей остатки, было предложено усовершенствовать программу Quicksort. А именно, вместо записи в указатель массива результата VM как полных, так и неполных векторов $\mathrm{V}_{\mathsf{осн},i}$ с выхода первой команды fuse в каждой ветви обработки (см. рисунок 1), записывать только полные вектора с длиной вектора $\mathrm{VL}{=}\mathrm{VL}_{\max}$. Тогда, если вектор $\mathrm{V}_{\mathsf{осн},i}$ не является полным, то этот вектор вместо записи в качестве элемента в указатель массива результата VM, направляется на вход второй команды fuse для присоединения к нему элементов накопленного вектора $\mathrm{V}_{\mathsf{нак}}$ из предыдущей итерации цикла по i. На рисунке 2 приведён потоковый граф одной из двух ветвей графа, начиная от команды fuse, которая на рисунке 1 объединяет вектора результаты команд split1 $\mathrm{V}_{<,i}$, и до записи векторов в указатель массива (блока) $\mathrm{VM}_{<}$. При входе в цикл

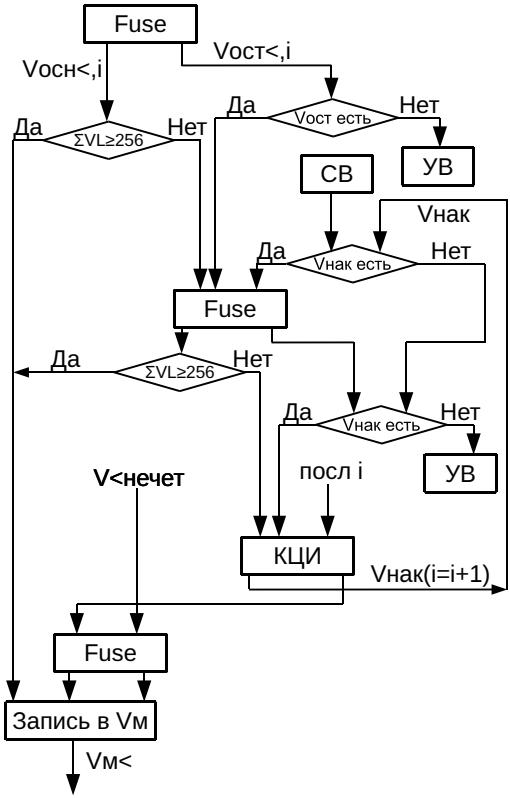

Рисунок 2. Усовершенствованный граф одной ветви разделения массива на блоки программы Quicksort

команда «создать вектор» (СВ) выделяет для $V_{нак}$ место в локальной памяти ВПП, и биту наличия присваивается значение «нет».

Но в случае отсутствия накопленного вектора $V_{нак}$ из предыдущей итерации по I команда ПУ направляет токены $V_{осн,i}$ или $V_{ост,i}$ с выхода первой команды fuse в обход второй команды fuse, на вход команды «конец цикла по итерации» (КЦИ).

Тогда в каждой итерации цикла по i в качестве $V_{нак}$ запоминается основной вектор $V_{осн}$ с первого выхода команды fuse, если этот вектор неполный, либо $V_{ост}$ — если есть вектор остаток на втором выходе команды fuse. Чтобы, как отмечалось в предыдущем разделе, вторая команда fuse правильно работала на каждом из её входов нужно

установить команды ПУ. Чтобы не усложнять граф на рисунке 2, команды ПУ на входах второй команды fuse, также как и блок управления, который вычисляя сумму длин векторов VL у операндов, сливаемых командой fuse, устанавливает бит наличия вектора для вектора $V_{ост}$ не показаны на графе.

Команда КЦИ в графе, приведённом на рисунке 2,служит для выхода из цикла по i. Она выдаёт пришедший на первый вход токен с указателем вектора $V_{нак}$ на правый выход команды, осуществляя выход из цикла, если номер итерации i, содержащийся в контексте токена, равен номеру последней итерации на втором входе команды. В противном случае токен указателя вектора $V_{нак}$ с входа команды КЦИ поступает на её левый выход, и номер итерации i в контексте токена с указателем вектора $V_{нак}$ увеличивается на 1.

Если число векторов N в указателе сортируемого массива нечётное, то при выходе из цикла вектор $V_{нак}$ с правого выхода команды КЦИ, как и вектор с суммой остатков в исходном графе программы Quicksort (см. рисунок 1) сливается командой fuse с результатом расщепления вектора VN. На этом проход по разделению массива на два блока $VM_{<}$ и $VM_{\geqslant}$ заканчивается, причём каждый из этих блоков содержит только полные вектора, за исключением последнего вектора, который может быть неполным.

Как видно из приведённого на рисунке 2 графа усовершенствованной программы Quicksort, в нем больше число проверок условий, чем в графе на рисунке 1. Если учесть, что итерации цикла по i с вектором $V_{нак}$ выполняются последовательно (см. рисунок 2), то было ожидаемо снижение производительности усовершенствованного алгоритма по сравнению с исходным. Однако моделирование показало, что разница в производительности при размере массива, равном 1 КиБ и 16 КиБ чисел с плавающей запятой двойной точности, оказалась меньше 5%. Это объясняется наличием зависимости по данным и у исходного алгоритма, поскольку суммирование остатков командой fuse имеет такую же зависимость по данным. Аналогично проверяется и условие, есть ли вектор $V_{ост}$ i на выходе первой команды fuse.

Моделирование показало, что программа, реализующая усовершенствованный алгоритм разделения массива на блоки, правильно работает при размере массива 32 КиБ чисел (128 векторов по 256 чисел). Однако на массиве из 64 КиБ чисел при выполнении одного из последних проходов число элементов у вектора, поступающего на вход команды split, может быть столь малым, что вектор маска —

результат сравнения этого вектора с порогом содержит все «1» или все «0». Это означает, что вектор на выходе команды split1 и split0 может отсутствовать, и его нужно пустить в обход первой команды fuse на вход команды УВ. В этом случае токен, который поступал на другой вход команды fuse, также нужно используя команду ПУ пустить в обход команды fuse. Конечно, это можно сделать, усложняя граф программы, но можно сделать и аппаратно за счёт введения дополнительных функций при выполнении уже имеющихся команд в системе команд ВПП.

3. Оптимизация системы команд ВПП для ускорения программы Quicksort

Такие дополнительные функции были введены для команд split и fuse, и заключались во включении в указатель вектора бита наличия вектора. У команды split1 этот бит устанавливался в «0», если вектор маска на входе команды содержит все «0», а у команды split0 — все «1», что означает отсутствие вектора результата. У команды fuse бит наличия необходим у указателя вектора $V_{ост}$ на втором выходе команды, поскольку этот вектор отсутствует, если сумма длин векторов у векторов операндов на входах команды fuse меньше или равна аппаратной длине вектора VL_{max}. Поскольку все элементы вектора на втором входе команды fuse были присоединены к элементам первого вектора, то вектор на выходе $V_{ост}$ оказывается пустым, и его нужно отправить на вход команды УВ, установив бит наличия в состояние «нет».

До введения бита наличия в указатель вектора его формирование, как отмечалось ранее, производилось скалярными арифметическими командами и командами сравнения, что увеличивало число выполняемых команд и задержку формирования этого бита. Отметим лишь, что для введения бита наличия в состав указателя вектора требуется минимум дополнительной аппаратуры, поскольку вычисление значений «да» или «нет» этого бита производилось в блоке формирования токенов, выдаваемых командами split1 и split0. Блок формирования токенов вычисляет число «1» в векторе маске у команды split, чтобы установить значения VL в указателе вектора на первом и втором выходах команды. При этом в поле данных токена размером 64 разряда с запасом размещаются адрес вектора в локальной памяти (21 разряд) ядра ВПП без номера элемента $N_{эл}$, который нужен для обращения по

чтению или записи одного элемента(8 разрядов) и длина вектора VL (9 разрядов). В оставшиеся 26 разрядов легко добавить не только бит наличия вектора, но и 3 бита проверки суммы VL у операндов команд split на $>$, $<$,$=$ относительно $\mathrm{VL}_{\max}$.

При этом введение в указатель вектора результата команд split бита наличия вектора позволило использовать этот бит для корректного выполнения команды fuse при отсутствии вектора на одном из входов этой команды. В случае прихода указателя вектора с битом наличия, равным «нет», нужно передать указатель другого вектора операнда на выход $\mathrm{V}_{\text{осн}}$ команды fuse без выполнения команды в ИУ. При этом указатель незначимого вектора операнда нужно направить на выход $\mathrm{V}_{\text{ост}}$, сохранив бит наличия «нет» для последующего уничтожения с помощью команды УВ.

Эти усовершенствования позволили без усложнения графа программы Quicksort не только восстановить её работоспособность при размере массива 64 КиБ чисел, но и уменьшить время выполнения при других размерах массива. Хотя наибольший вклад в уменьшение времени выполнения программы достигнут за счет разделения вектора по маске одной командой split вместо использования двух команд split1 и split0. В графе на рисунке 1 показан именно такой вариант команды split, у которой два выхода, на первом — результат команды split1, на втором — split0. В [6] отмечалось, что использовать одну команду split с созданием двух векторов возможно, но нужно будет менять управление в блоке выполнения специальных операций для записи двух векторов результатов одной векторной команды. На самом деле сложность заключалась в том, чтобы не обращаться дважды для чтения в локальной памяти векторов (ЛПВ) элементов разделяемого по маске вектора. Первый раз с выборкой элементов, отмеченных в векторе маске «1», второй раз — отмеченных «0». В этом случае пропускная способность блока специальных операций ВПП при выполнении такой команды split осталась бы такой же, как при выполнении двух команд split1 и split0.

Отметим, что блок выполнения специальных операций в ВПП, выполняющий команды split, fuse и другие, как и 4 векторных АЛУ, обрабатывает вектора страницами по 32 элемента в тридцати двух кольцах (lanes). Для пересылки данных между кольцами требуются 2 коммутатора, Один из них при выполнении командой split чтения по маске для пересылки считанных данных из одного кольца в другое кольцо для записи вектора результата. Эту задачу можно решить, если

элементы исходного вектора, отмеченные «1» в векторе маске, после чтения в ЛПВ записывать в один и тот же расслоённый по кольцам буфер, что и элементы, отмеченные «0». Для последующей раздельной выборки из буфера тех и других элементов для записи первого и второго результата команды split их нужно записывать в буфер с разных сторон. Элементы исходного вектора, отмеченные «1» в векторе маске, начиная с нулевого адреса в сторону увеличения, элементы, отмеченные «0» — в сторону уменьшения, начиная с адреса 255. В этом случае не будет конфликтов при прохождении элементов через коммутатор для записи в соответствующие модули буфера.

На рисунке 3 приведены значения удельного времени, полученные при моделировании программы Quicksort в зависимости от числа элементов n в сортируемом массиве, где n равно числу векторов N, умноженному на 256. Удельное время это время выполнения программы в тактах, делённое на число элементов в массиве. Приведено для двух вариантов графа программы: исходного, приведённого на рисунке 1, и усовершенствованного — на рисунке 2.

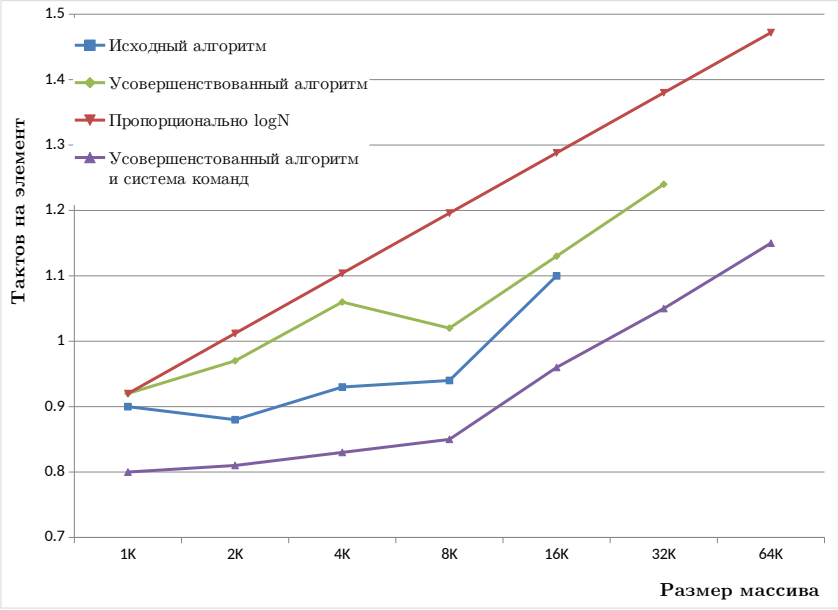

Рисунок 3. Удельное время исходного и нового алгоритмов программы Quicksort

При этом результаты моделирования усовершенствованного графа программы приведены как для системы команд, при которой проводилось моделирование исходного графа программы, так и для команд split и fuse с введением дополнительных функций. Ещё на рисунке 3 приведена зависимость объёма вычислений в программе Quicksort (на 1 элемент) от числа элементов n в сортируемом массиве — пропорционально $\log n$.

Как видно из представленных на рисунке 3 зависимостей, удельное время исходного алгоритма меньше, чем усовершенствованного (производительность выше). Это объясняется большей сложностью нового алгоритма — большим числом проверок условий в теле цикла по i, как уже отмечалось выше. Оптимизация системы команд позволила уменьшить удельное время выполнения усовершенствованного варианта программы Quicksort в среднем на 21%.

Заключение

Таким образом, усовершенствованный алгоритм и оптимизация системы команд ВПП позволили устранить недостаток более раннего алгоритма и не только сохранить работоспособность программы Quicksort при увеличении размера сортируемого массива до 64 КиБ элементов, но и повысить её производительность. Вклад программы Quicksort в общее время комбинированной программы сортировки (включающей битоническую сортировку в пределах каждого блока) уменьшается с 50% времени выполнения при размере массива 1 КиБ элементов до 33% при размере массива 64 КиБ элементов. В результате уменьшение времени выполнения программы Quicksort на 21% сокращает общее время выполнения комбинированной сортировки на 7%.

Если комбинированная программа сортировки выполнялась ранее в 7,4 раза быстрее аналогичной на Intel Skylake [6], то модернизированный вариант графа и системы команд, описанные выше, позволяют увеличить разницу до 7,9 раз. Таким образом, несмотря на значительное усложнение графа программы сортировки производительность её выполнения в ВПП почти в 8 раз выше по сравнению с процессором традиционной архитектуры.

Список литературы

[1] A. H. Veen. “Dataflow machine architecture”, *ACM Computing Surveys*, **18**:4, pp. 365–396. doi ↑65

[2] Д. Л. Хеннеси, Д. А. Паттерсон. *Computer Architecture: A Quantitative*

Approach, ред. А. К. Ким, Техносфера, М., 2016, ISBN 978-5-94836-413-1%, 936 с. ↑66

[3] T. Yuba, K. Hiraki, T. Shimada, S. Sekiguchi, K. Nishida. “578–585”, *Proceedings of the 1987 Fall Joint Computer Conference on Exploring technology: today and tomorrow*, ACM ’87 (December 1987, Dallas, Texas, USA), IEEE Computer Society Press, Washington, DC, US, ISBN 978-0-8186-0811-7. ↑66

[4] J. Gurd, W. Bohm, Yong Meng Teo. “Performance issues in dataflow machines”, *Future Generations Computer Systems*, **3**:4 (1987), pp. 285–297. ↑66

[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. doi ↑66,68,69

[6] 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*, **11**:4(47) (2020), с. 55–71. doi ↑67,70,73,77,79

[7] Н. И. Дикарев, А. С. Шмелёв. «Выполнение программы сортировки в векторном dataflow процессоре», *ИТНОУ: Информационные технологии в науке, образовании и управлении*, 2021, №1(17), с. 86–91. ↑67

[8] Н. И. Дикарев, Б. М. Шабанов, А. С. Шмелёв. «Реализация памяти структур данных в векторном потоковом процессоре», *Труды НИИСИ РАН*, **9**:6 (2019), с. 156–160. URL ↑69

[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. ✻ ↑69

[10] 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. doi ↑69

[11] 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. doi ↑70

Поступила в редакцию 12.12.2021
Переработана 24.12.2021
Опубликована 30.12.2021

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*, 2021, **12**:4(51), с. 65–83. 10.25209/2079-3316-2021-12-4-65-83
http://psta.psiras.ru/read/psta2021_4_65-83.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 Quicksort program execution in the vector dataflow processor (VDP), which uses the pointer vectors method to store arrays. Earlier revealed deficiency of pointer vectors method was compensated by the introduction of split and fuse commands into the processor instruction set. In this article we analyze improved graph of the Quicksort program and the results of its simulation on the original and modernized instruction sets of VDP. We also compare simulation results with performance of Intel Skylake processor core.

Key words and phrases: vector processor, dataflow architecture, sorting algorithms, parallelism, vector performance, instruction set.

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

References

[1] A. H. Veen. "Dataflow machine architecture", *ACM Computing Surveys*, **18**:4, pp. 365–396. ↑65

[2] J. L. Hennessy, D. A. Patterson. *Computer Architecture: A Quantitative Approach*, 5th edition, Morgan Kaufmann, 2011, ISBN 978-8178672663, 856 pp. ↑66

[3] T. Yuba, K. Hiraki, T. Shimada, S. Sekiguchi, K. Nishida. "578–585", *Proceedings of the 1987 Fall Joint Computer Conference on Exploring technology: today and tomorrow*, ACM '87 (December 1987, Dallas, Texas, USA), IEEE Computer Society Press, Washington, DC, US, ISBN 978-0-8186-0811-7. ↑66

[4] J. Gurd, W. Bohm, Yong Meng Teo. "Performance issues in dataflow machines", *Future Generations Computer Systems*, **3**:4 (1987), pp. 285–297. ↑66

[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. ↑66,68,69

[6] N. I. Dikarev, B. M. Shabanov, A. S. Shmelëv. "Advantages and disadvantages of using the pointer vector method in a vector dataflow processor", *Program Systems: Theory and Applications*, **11**:4(47) (2020), pp. 55–71 (in Russian). ↑67,70,73,77,79

[7] N. I. Dikarev, A. S. Shmelëv. "The vector-pointer method and its use in the dataflow processor", *ITNOU: Informatsionnyye tekhnologii v nauke, obrazovanii i upravlenii*, 2021, no. 1(17), pp. 86–91 (in Russian). ↑67

10.25209/2079-3316-2021-12-4-65-83

[8] N. I. Dikarev, B. M. Shabanov, A. S. Shmelëv. "Structure store implementation in vector dataflow processor", *Trudy NIISI RAN*, **9**:6 (2019), pp. 156–160 (in Russian). ↑69

[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). ↑69

[10] 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). ↑69

[11] 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. ↑70

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*, 2021, **12**:4(51), pp. 65–83. (*In Russian*).

10.25209/2079-3316-2021-12-4-65-83

http://psta.psiras.ru/read/psta2021_4_65-83.pdf