ISSN 2079-3316 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ №4(35), 2017, с. 305–317

УДК 004.27

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

Выполнение задач сортировки на векторном процессоре с архитектурой управления потоком данных

Аннотация. Задача сортировки является широко распространённым вычислительным блоком, который используется во многих областях, и известно много алгоритмов, как последовательных, так и параллельных, решения этой задачи. Поэтому несколько известных алгоритмов решения задачи сортировки были выбраны для тестирования разрабатываемого в МСЦ РАН векторного процессора с архитектурой управления потоком данных и сравнения достигнутой при моделировании производительности с производительностью лучших современных процессоров.

Анализируются преимущества и недостатки архитектуры разрабатываемого векторного процессора, их учет для построения графов программ последовательных и параллельных алгоритмов решения задачи сортировки на данном процессоре и достигнутые при этом результаты.

Ключевые слова и фразы: суперкомпьютер, векторный процессор, архитектура управления потоком данных, оценка производительности, сортировка слиянием, битонная сортировка.

Введение

Сортировка является широко распространённым вычислительным блоком, и эффективность реализации сортировки критична в таких областях как, например, базы данных или визуализация. Сортировка также применяется в качестве библиотечных приложений для понижения сложности некоторых алгоритмов [1]. Таким образом, эффективная реализация сортировки на новых архитектурах процессора может увеличить производительность многих приложений.

Цель данной работы — показать на примере выполнения задач сортировки, что разрабатываемый в МСЦ РАН процессор с архитектурой управления потоком данных (потоковый процессор) за счет усиленной векторной обработки и параллельного выполнения нескольких

Работа выполнена при поддержке гранта РФФИ 17-07-01457 и с использованием вычислительных ресурсов МСЦ РАН.

© Н. И. Дикарев, Б. М. Шабанов, А. С. Шмелёв, 2017
© Межведомственный суперкомпьютерный центр РАН, 2017
© Программные системы: теория и приложения, 2017

DOI: 10.25209/2079-3316-2017-8-4-305-317

программных блоков может обеспечить значительно более высокую производительность по отношению к лучшим процессорам традиционной архитектуры.

1. Принцип работы потокового процессора и его преимущества и недостатки

В потоковом процессоре программой является ориентированный граф, узлами которого являются команды, а информация по дугам передается в виде токенов, содержащих значение операнда и его контекст. Этот контекст определяет номер команды приемника в графе программы, куда должен быть отправлен токен, а также определяет принадлежность команды к конкретному номеру итерации во вложенных циклах и запусках процедур, что позволяет выполнять их в параллель, используя один и тот же граф программы. Вне зависимости от места двухвходовой команды в графе она выдаётся на исполнение по прибытию на вход последнего из пары токенов операндов с одинаковым контекстом.

После вычисления результата в исполнительном устройстве (ИУ) новые токены со значением результата отправляются на входы последующих команд согласно графу программы, а использованные токены операндов уничтожаются. Такой принцип работы позволяет исключить конфликты информационной зависимости — основную причину ограничения производительности процессора традиционной архитектуры, в котором до выдачи команды на выполнение нужно проверить записаны ли в память (регистровый файл) результаты предыдущих команд, которые должна читать в качестве операндов выдаваемая команда.

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. А именно, устройство поиска готовых к выполнению команд — память поиска пар готовых операндов (ППП) может содержать до нескольких десятков работающих в параллель модулей, обеспечивающих работой такое же большое число ИУ. Поэтому потенциально потоковый процессор может обеспечить значительно более высокую производительность по отношению к лучшим процессорам традиционной архитектуры, но лишь при наличии достаточного параллелизма в программе.

Однако при выполнении цепочек из последовательных команд потоковому процессору требуется значительно большее время, поскольку к времени вычисления результата в ИУ, как в фон-неймановском процессоре, добавляется задержка, вносимая модулем ППП и двумя

коммутаторами в цепи прохождения токена от ИУ к ППП и обратно. Кроме того, в потоковом процессоре приходится выполнять в 2–3 раза больше команд на программах научных задач по сравнению с фоннеймановским процессором [2]. Одна из причин в том, что команда, как правило, 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.

Использование векторной обработки в разрабатываемом в МСЦ РАН векторном потоковом процессоре (ВПП) позволяет многократно уменьшить число выполняемых команд, поскольку одна векторная команда заменяет собой цикл с VL независимыми итерациями, где VL — длина вектора. Для записи результатов векторных команд и хранения массивов в разрабатываемом ВПП используется обычная линейно адресуемая память, содержащая два уровня. Это — 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. 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.

При этом большие массивы предлагается хранить в ПВ в виде «векторов-указателей», то есть, векторов, элементами которых являются указатели векторов подмассивов. Такое хранение массивов увеличивает время доступа к произвольному элементу массива, но позволяет одной командой «Формирование потока» (ФП) выдать токены с указателями, например, всех векторов строк из вектора-указателя матрицы для их последующей обработки в цикле. Тем самым удается векторизовать два вложенных цикла, исключив из тела цикла команду перехода, адресные и другие скалярные вычисления [3].

Далее будет рассмотрено использование команд ФП в графах программ сортировки. По результатам выполнения этих программ

на VHDL модели ВПП будут определены значения производительности, и дано их сравнение с производительностью процессоров фирмы Intel. Рассмотрим алгоритм сортировки слиянием, который содержит следующую последовательность шагов [4]:

(1) Сортируемый массив разбивается на части примерно равного размера.

(2) Каждая из получившихся частей сортируется отдельно, используя какой-то вспомогательный алгоритм сортировки.

(3) Упорядоченные отрезки массива по два соединяются в один отсортированный отрезок вдвое большего размера.

Пункт 3 повторяется до тех пор, пока все отрезки не образуют один отсортированный массив

2. Сортировка с использованием команд подсчёта совокупностей

Сортируемый массив чисел в соответствии с методом векторов-указателей хранится в памяти ВПП в виде векторов с длиной, не превышающей аппаратную длину вектора $\mathrm{VL}_{\max}$, упакованных в вектор-указатель массива. Поэтому на первом шаге описанного выше алгоритма сортировки массив, разбивается на вектора длиной $\mathrm{VL}_{\max}$, и элементы таких векторов можно сортировать в параллель на втором шаге сортировки. В качестве вспомогательного алгоритма для второго шага сортировки нами был выбран известный алгоритм, применявшийся в векторных процессорах CRAY и основанный на использовании команд подсчёта совокупностей. Это векторные команды редукции, преобразующие вектор в скаляр, в частности, для сортировки используется команда, подсчитывающая число единичных элементов в битовом векторе.

Используемый алгоритм представляет собой цикл по i от 0 до $(N-1)$, где N — длина вектора, в каждой итерации которого очередной элемент $a(i)$ сравнивается со всеми элементами сортируемого вектора A. Результатом векторной команды сравнения является бит вектор, принимающий значение 1 для тех элементов вектора, которые меньше элемента $a(i)$. Тогда число единичных элементов в бит векторе -результате команды подсчёта совокупности является номером этого элемента в отсортированном векторе. Например, для минимального элемента исходного вектора результатом команды сравнения будет бит вектор из всех нулей.

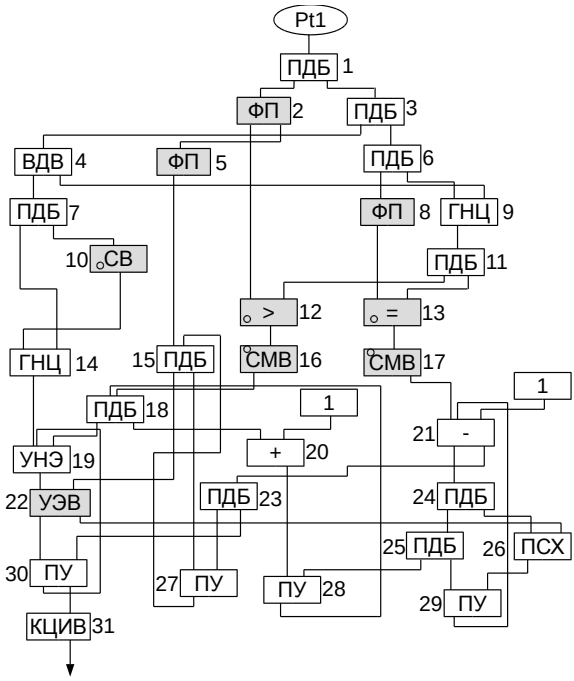

Рис. 1. Граф программы сортировки элементов вектора

Команда, подсчитывающая число единиц, выдаст 0, и этот элемент нужно записать в нулевой элемент отсортированного вектора. Записав элемент $a(i)$ по выданному командой подсчёта числа 1 номеру и выполнив N итераций цикла, на выходе получаем отсортированный вектор. На рис. 1 показан граф этой программы для ВПП, в котором команды, выполняемые в векторных ИУ (ВИУ), выделены более тёмным цветом.

Выполнение программы в ВВП начинается с прихода на вход команды дублирования (ПДБ) 1 токена с указателем Pt1 сортируемого вектора A, предварительно переписанного из ПВ в ЛПВ. Этот указатель содержит два параметра: адрес начального (0-го) элемента вектора A в ЛПВ и число его элементов (длину вектора). При выполнении команды 1 в СИУ формируются два токена с указателем Pt1, которые посылаются на входы команд 2 и 3. Команда ПДБ 3 увеличивает число токенов с указателем Pt1 до трёх, один из которых активирует выполнение команды «Выдать Длину Вектора» (ВДВ) 4, которая, в свою очередь, выдаёт два токена с длиной вектора в поле

данных. Как видно из графа программы на рис. 1, каждая из команд с одним входом вместо одного токена, поступившего на вход, может выдать 2 токена с выхода, увеличивая тем самым число команд, выполняемых в параллель.

К ещё большему параллелизму приводит выполнение в ВИУ команд ФП и в скалярных ИУ (СИУ) — команд «Групповое Начало Цикла» (ГНЦ), которые помимо формирования серии токенов по левому выходу команды ещё передают токен с входа на правый выход. Так каждая из команд ФП 2, 5 и 8 по указателю Pt1 со входа производит чтение элементов вектора A в ЛПВ и формирует N токенов из элементов $a(i)$, причем индекс i записан в поле контекста каждого токена.

Аналогично команды ГНЦ 9 и 14 размножают операнд на левом входе команды для последующего использования в теле цикла, то есть формируют последовательность из N токенов, со значениями индекса i от 0 до $(N-1)$ в поле контекста, как и у команд ФП, что позволяет выполнять в параллель итерации цикла. Заметим, что команда «Создать Вектор» (СВ) 10, выдаёт адрес свободного вектора в ЛПВ для записи элементов отсортированного вектора. Далее указатель этого вектора размножается командой ГНЦ 14.

Таким образом, команды ФП 2 и 8 вместе с командой ГНЦ 9 создают все необходимые токены для выполнения в параллель векторных команд сравнения 12, 13 и последующих команд «Сумма единичных элементов Вектора» (СМВ) 16, 17. Команды сравнения проверяют условие $a(i) < A$ и $a(i) = A$ во всех итерациях цикла, а команды СМВ 16 и 17 подсчитывают число элементов, удовлетворяющих этим условиям. В результате для всех i от 0 до $(N-1)$ команда СМВ 16 выдаёт номер элемента в отсортированном векторе, в который нужно записать значение $a(i)$.

Команда «Установить Элемент Вектора» (УЭВ) 19 записывает $a(i)$ в вектор, созданный командой СВ 10, в элемент, номер которого установлен командой «Установить Номер Элемента» (УНЭ) 19 (см. рис. 1). Токены операнды для команд УЭВ 22 (значение $a(i)$) и для команд УНЭ 19 (указатель вектора результата) формируют команды ФП 5 и ГНЦ 14 соответственно, причем для всех итераций цикла и с минимальными накладными расходами на организацию цикла. Наконец команда КЦИВ 31 выдает токен с указателем вектора результата на выход графа программы при условии, что индекс в поле контекста равен длине вектора N минус 1, то есть при выполнении последней итерации цикла.

Что же касается большинства других скалярных команд в рассматриваемом графе программы, то они необходимы, если в сортируемой

последовательности элементов $a(i)$ встречаются два или более одинаковых элемента. Число таких элементов выдаёт команда СМВ 17, и их нужно записать в последовательные элементы вектора результата, начиная с номера элемента, выданного командой СМВ 16. При этом в теле цикла должно быть ветвление по условию — число одинаковых элементов равно 0 или нет, и в случае неравенства организовать внутренний цикл с записью одного и того же значения в несколько элементов вектора результата. Но при выполнении ветвления каждое из значений данных, используемых в ветвях программы, необходимо направить в нужную ветвь с помощью отдельной команды переключателя, в то время как традиционном процессоре требуется лишь одна команда условного перехода.

Команда переключателя «Переход по Условию» (ПУ) направляет токен с левого входа команды на правый выход, если токен на управляющем (правом) входе равен 0, и на левый выход — если не равен 0. Таких команд ПУ в графе на рис. 1 четыре, плюс 2 арифметические команды (команды 20 и 21), плюс команды дублирования ПДБ 18, 23, 24, 25, плюс команда синхронизации ПСХ 26. Синхронизация нужна для последовательного выполнения итераций во внутреннем цикле, поскольку токены в этом цикле имеют одно и то же значение индекса в поле контекста.

Таким образом, команды из разных итераций основного цикла могут выполняться в параллель благодаря разным значениям индекса i в поле контекста, и на их фоне в тех итерациях, где $a(i) = a(k)$, формируется внутренний цикл с последовательным выполнением итераций. Параллелизм основного цикла раскручивается до тех пор, пока одно из ИУ не оказывается перегруженным, то есть не образуется очередь из готовых команд на входе ИУ. Как показало моделирование, в ВВП с тремя мощными ВИУ, которые обрабатывают вектора страницами по 32 элемента в такт, и с тремя СИУ, одно из которых выполняет двухвходовые команды, и два — одновходовые, таким узким местом в данной программе оказываются СИУ.

На рис. 2 показано среднее время выполнения рассматриваемой программы сортировки в пересчёте на 1 элемент в зависимости от длины сортируемого вектора для двух значений числа сортируемых векторов: 2 и 64 (для их последующего слияния). Не забываем, что программа сортировки одного вектора, показанная на рис. 1, является лишь вспомогательным алгоритмом на втором шаге сортировки слиянием.

Поэтому основная цель рассмотренного алгоритма — ускорить сортировку векторов длиной не более $\mathrm{VL}_{\max}$=256 элементов, из которых состоит сортируемый массив, перед выполнением их слияния

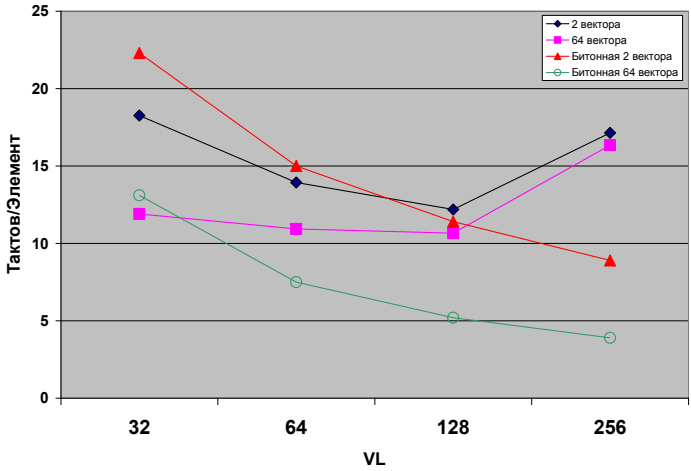

Рис. 2. Среднее время сортировки на 1 элемент с использованием команд редукции и битонного слияния

на третьем шаге. Соответственно на втором шаге сортируются все вектора, указатели которых входят в состав вектора указателя массива, и программа, показанная на рис. 1, выполняется в цикле. Перед входом в цикл вектор указатель массива копируется из ПВ в ЛПВ вместе с вектором данных, указатель которого записан в нулевой элемент вектора-указателя массива. Далее в каждой итерации цикла кроме последней выполняется сортировка элементов ранее записанного в ЛПВ вектора, а также копирование из ПВ в ЛПВ вектора для следующей итерации цикла.

Как видно из приведённых на рис. 2 зависимостей, задержка выборки из ПВ вектора-указателя массива и нулевого вектора данных, считавшаяся при моделировании равной 100 тактам, заметно увеличивает среднее удельное время сортировки 1 элемента лишь при малой длине вектора и сортировке двух векторов.

С увеличением длины вектора удельное время уменьшается, достигая минимума при VL=128, и возрастает при VL=256. Дело в том, что при VL_{max}=256 производительность данной программы ограничивается не СИУ, как при других значениях VL, а одним из трёх ВИУ, входящих в состав ВПП, которое выполняет команды редукции СМВ с производительностью 32 элемента в такт. Поскольку в каждой итерации основного цикла выполняются две команды СМВ (16 и 17 на рис. 1), то при VL_{max}=256 время выполнения этих двух команд равно 16 тактам, и время выполнения одной итерации будет всегда больше 16 тактов, что и подтверждается результатами моделирования.

Для сравнения лучшее время сортировки массива из N элементов на процессоре Intel Pentium, согласно [5], составляет в секундах $10 \times 10^{-9} N \times \log_2 N$, и при тактовой частоте процессора 2,4 ГГц время сортировки в пересчёте на один элемент равно $24 \log_2 N$ тактов. Следовательно, при $N = 128$ производительность ВПП выше, чем у Intel Pentium, в 13,8 раза и при $N = 256$ — в 11,2 раза.

3. Сортировка слиянием предварительно отсортированных векторов

Как уже упоминалось выше, на третьем шаге алгоритма сортировки слиянием упорядоченные отрезки массива по два объединяются в один отсортированный отрезок вдвое большего размера. Последовательный алгоритм такого объединения двух отсортированных подмассивов в один состоит в следующем [4]:

(*1*) На каждом шаге берём меньший из двух первых элементов подмассивов и записываем его в результирующий массив. Счётчики номеров элементов результирующего массива и подмассива, из которого взят элемент, увеличиваем на 1.

(*2*) Когда один из подмассивов закончился, добавляем все элементы второго подмассива в результирующий массив.

Применительно к ВПП такой алгоритм слияния двух предварительно отсортированных векторов предполагает, что их указатели хранятся в векторе-указателе массива, и на выходе из программы тот же указатель массива содержит указатели двух векторов, первый из которых содержит упорядоченные по возрастанию меньшие элементы двух исходных векторов, а второй — большие.

Понятно, что в графе этой программы тело цикла содержит ветвление по условию, которое увеличивает число выполняемых команд из-за наличия нескольких команд переключателя. Кроме того, тело цикла содержит цепочки из нескольких последовательно выполняемых скалярных команд, таких как увеличение на 1 номера элемента, чтение по этому номеру значения элемента в ЛПВ, его сравнение с элементом противоположного вектора и запись одного из этих элементов в вектор-результат в зависимости от результата команды сравнения.

Как отмечалось ранее, время выполнения таких цепочек команд, связанных зависимостью по данным, в потоковом процессоре значительно больше, чем в фон-неймановском процессоре, что и подтвердило моделирование. Так при времени выборки из ЛПВ равном 10 тактам, число тактов необходимых для слияния двух векторов по 32 элемента в пересчёте на 1 элемент составило 79,7 такта. При этом ограничивает производительность не пропускная способность СИУ, а большая задержка выполнения цепочек из последовательных команд.

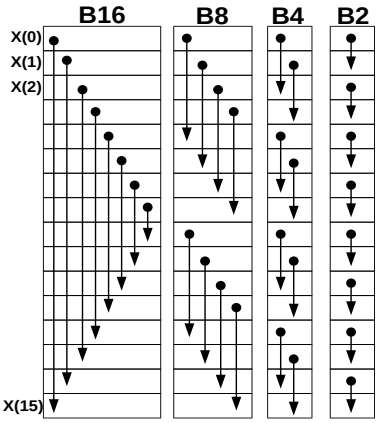

Рис. 3. Сеть компараторов для битонного слияния двух векторов

В этом случае можно повысить производительность ВПП, если слияние двух векторов вести не только по возрастанию элементов, начиная с наименьших значений, но и одновременно в противоположном направлении. Такая модифицированная программа позволила уменьшить число тактов на 1 элемент с 79,7 такта до 62,8 такта, и при увеличении длины векторов до 256 элементов — до 60,1 такта на элемент, что существенно больше, чем у Intel Pentium — 24 такта на элемент.

Таким образом, на последовательном алгоритме слияния ВПП в 2,5 раза уступает в производительности фон-неймановскому процессору, однако слияние двух векторов можно выполнить, используя параллельный и, более того, векторизуемый алгоритм. Это битонная сортировка — параллельный алгоритм, в основе которого лежит сортировочная сеть или сеть компараторов.

На рис. 3 показана сеть для слияния двух предварительно отсортированных векторов по 8 элементов в каждом. Эта сеть содержит 4 ступени компараторов B16, B8, B4, B2, каждая из которых попарно сравнивает 16 элементов на входе следующим образом.

Ступень компараторов Bn сравнивает между собой элементы $x(i)$ и $x(i + n/2)$ на входе ступени и упорядочивает их по возрастанию. Заметим, что элементы $x(i)$ и $x(i + n/2)$ на входе ступени B16 есть ни что иное, как элементы $a(i)$ двух сливаемых векторов $A0$ и $A1$ по 8 элементов в каждом, только перед подачей на вход команды векторного сравнения элементы вектора $A1$ должны быть переупорядочены по убыванию.

Такое переупорядочивание элементов вектора $A1$ выполняется в ВПП векторной командой «Упорядочить Вектор», которая элементы вектора с первого входа команды записывает в вектор-результат в соответствии с индексами у вектора на втором входе. В данном примере вектор индексов будет содержать следующие элементы: 7, 6, 5, 4, 3, 2, 1, 0. Аналогично переупорядочиваются и вектора с выхода ступени перед поступлением на вход следующей ступени.

Как показало моделирование, основное время на прохождение одной ступени сортировочной сети складывается из выполнения двух команд «Упорядочить Вектор» для сборки векторов операндов команды сравнения и переупорядочивания элементов этих векторов в результате слияния по маске — результату команды векторного сравнения. Таким образом, векторные команды выполняют большинство операций в этой программе, что обеспечивает высокую производительность её выполнения на ВПП.

Из приведённых на рис. 2 результатов видно, что при попарном слиянии 64 векторов среднее время слияния двух векторов по 256 элементов в пересчёте на 1 элемент составило 3,9 такта, что в 3,5 раза меньше чем в одном ядре процессора Intel Xeon Knights Landing [1]. Причина проста — производительность ВПП на векторной обработке от 4 до 8 раз выше в зависимости от типа команд по сравнению с ядром в Knights Landing.

Заключение

Моделирование времени выполнения задач сортировки на ВПП подтвердило, что на параллельных алгоритмах, таких как битонная сортировка и с использованием команд подсчёта совокупностей, производительность ВПП в 3–12 раз выше по сравнению с процессорами фирмы Intel. В то же время на чисто последовательных алгоритмах, таких как сортировка слиянием, ВПП уступает в производительности современным процессорам до 3 раз.

Список литературы

[1] B. Bramas. *Fast sorting algorithms using AVX-512 on Intel Knights Landing*, arXiv: 1704.08579 [cs.MS]. ↑ 305,315

[2] G. V. Papadopoulos, K. R. Traub. “Multithreading: a revisionist view of dataflow architectures”, *Proc. 18-th Ann. Symp. on Computer Architecture*, ISCA ’91 (Toronto, Ontario, Canada, May 27–30, 1991), ACM, New York, 1991, pp. 342–351. ↑ 307

[3] 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. ↑ 307

[4] https://ru.wikipedia.org/wiki/Сортировка_слиянием. ↑ 308,313
[5] М. В. Якобовский, *Введение в параллельные методы решения задач*, Учебное пособие, Издательство Московского университета, М., 2013, 328 с. ↑ 313

Рекомендовал к публикации *Программный комитет* Шестого национального суперкомпьютерного форума ***НСКФ-2017***

Пример ссылки на эту публикацию:

Н. И. Дикарев, Б. М. Шабанов, А. С. Шмелёв. «Выполнение задач сортировки на векторном процессоре с архитектурой управления потоком данных», *Программные системы: теория и приложения*, 2017, **8**:4(35), с. 305–317. URL: http://psta.psiras.ru/read/psta2017_4_305-317.pdf

Об авторах:

Николай Иванович Дикарев
Кандидат технических наук, ведущий научный сотрудник МСЦ РАН
e-mail: nic@jscc.ru

Борис Михайлович Шабанов
Кандидат технических наук, доцент, лауреат Государственной премии Российской Федерации в области науки и техники. Директор МСЦ РАН
e-mail: shabanov@jscc.ru

Александр Сергеевич Шмелёв
Научный сотрудник МСЦ РАН
e-mail: guest8993@rambler.ru

Nikolay Dikarev, Boris Shabanov, Aleksandr Shmelev. *Execution of sorting algorithms on vector dataflow processor.*

Abstract. Sorting is a common problem in computer science and there is a lot of algorithms, both sequential and parallel to solve this problem. Therefore, several well-known sorting algorithms were selected for testing Vector Dataflow Processor(VDP). In this paper we analyze advantages and disadvantages of the VDP for solving the sorting problem and compare performance based on the simulation results with the best traditional processors. (*In Russian*).

Key words and phrases: supercomputer, vector processor, dataflow architecture, performance evaluation, merge sort, bitonic sort.

References

[1] B. Bramas. *Fast sorting algorithms using AVX-512 on Intel Knights Landing*, arXiv: 1704.08579 [cs.MS].

[2] G. V. Papadopoulos, K. R. Traub. "Multithreading: a revisionist view of dataflow architectures", *Proc. 18-th Ann. Symp. on Computer Architecture*, ISCA '91 (Toronto, Ontario, Canada, May 27–30, 1991), ACM, New York, 1991, pp. 342–351.

[3] N. I. Dikarev, B. M. Shabanov, A. S. Shmelëv. "Vector dataflow processor: performance evaluation", *Izvestiya YuFU. Tekhnicheskiye nauki*, 2014, no.12(161), Tematicheskiy vypusk: Superkomp'yuternyye tekhnologii, pp. 36–46 (in Russian).

[4] *Merge sort*, From Wikipedia, the free encyclopedia, URL: https://en.wikipedia.org/wiki/Merge_sort

[5] M. V. Yakobovskiy, *Introduction to parallel methods for solving problems*, Uchebnoye posobiye, Izdatel'stvo Moskovskogo universiteta, M., 2013 (in Russian), 328 p.

Sample citation of this publication:

Nikolay Dikarev, Boris Shabanov, Aleksandr Shmelev. "Execution of sorting algorithms on vector dataflow processor", *Program systems: Theory and applications*, 2017, **8**:4(35), pp. 305–317. (*In Russian*).

URL: http://psta.psiras.ru/read/psta2017_4_305-317.pdf

© N. I. Dikarev, B. M. Shabanov, A. S. Shmelev, 2017
© Joint Supercomputer Center of RAS, 2017
© Program systems: Theory and Applications, 2017

DOI: 10.25209/2079-3316-2017-8-4-305-317