ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ СОРТИРОВКИ СЛИЯНИЕМ В МОДЕЛИ ВЫЧИСЛЕНИЙ С УПРАВЛЕНИЕМ ПОТОКОМ ДАННЫХ

А.В. Климов, М.Ж. Акжолов
Институт проблем проектирования в микроэлектронике РАН, Москва, Россия

Рассматривается эффективный мелкозернистый параллельный алгоритм сортировки, в основе которого лежит существующий классический алгоритм сортировки «слиянием» [1,2,3]. Алгоритм написан на языке DFL модели параллельной потоковой вычислительной системы (ППВС), проект которой восходит к работе [4] и развивается в работах [5,6].

Алгоритм сортировки выполняется поэтапно. На каждом этапе имеется несколько пар отсортированных порций, которые попарно сливаются в одну порцию, то есть количество порций уменьшается в два раза, а в каждой порции количество данных удваивается. Этот процесс продолжается, пока не останется одна порция. Слияние каждой пары порций происходит последовательно на одном вычислительном ядре (ВЯ), поэтому на заключительных этапах число используемых ядер сокращается. Начальный массив длины N подается как N порций по 1 элементу в каждой порции.

Программа сортировки, написанная на языке DFL, состоит из набора описаний узлов. На рис.1 приведен код основного узла, выполняющего слияние двух порций.

```
type Puck = { v:real32, j:int }
node Merge(v:real32,s:Puck) {l,k,i};
   var z:real32;
   begin
      if(v <= s.v) then
         begin
            z:=v;
            send s to AA.s{l,k,i+1};
         end
      else
         begin
            z:=s.v;
            send {i,a} to AA.s{l,k xor 1,s.j+1};
         end;
      send  z << 1 >> to AA.v{l+1,k div 2,i+s.j};
   end;
```

Рис. 1. Листинг основного узла программы на языке DFL

Данный узел имеет имя Merge и два входа: *v* – значение элемента, *s* – «шайба». Экземпляры узлов различаются контекстом, который в данном узле состоит из полей: *i* – номер элемента в порции, *k* – номер порции, *l* – номер этапа (level). Предполагается, что две сливаемые порции имеют соседние номера *k* и *k*+1, где *k* – четное. Элементы порции *k* поданы на входы *v* с номерами *i*=0,1,2,…K_k. В некоторый момент начальные части порций уже слиты и результат слияния, кроме последнего элемента, находится в выходной порции следующего уровня. На второй вход очередного элемента одной из порций направлена «шайба», содержащая последнее значение слитой части (поле *v*) и номер очередного элемента парной порции (поле *j*). Узел выбирает из своего значения и значения «шайбы» минимум и отправляет его в выходную порцию. Другое значение отправляется оператором send в качестве «шайбы» на очередной элемент этой же или

парной порции, в зависимости от того, какой именно элемент был выбран. В начале слияния на первый элемент одной из порций посылается начальная шайба со значением ниже минимального («искусственный минимум»). В конце каждой порции должен находиться «искусственный максимум».

Значения приходят на входы в виде токенов, которые создаются операторами send. Когда токены придут на все входы некоторого узла, узел сработает, и выполнится его программа. В результате появляются новые токены, направляемые на другие узлы.

Процесс слияния двух порций выполняется последовательно путем «перекидывания токена-шайбы» между этими порциями. Вначале, на этапе 0 имеется *N* порций по одному элементу в каждой порции. Поскольку при слиянии каждой пары порций делается 3 лишних сравнения, несколько первых этапов сделаны особо методом сортирующих сетей.

На рис.2 показана визуализация алгоритма на модели ППВС. Каждая линия соответствует токену, посылаемому одним узлом другому, причем пунктиром выражены пересылки токенов между разными вычислительными ядрами. Горизонтальные линии – это «шайбы», они всегда проходят внутри ядер. Цвет соответствует номеру этапа, к которому относится целевой узел. Количество вычислительных ядер n_p=4, количество сортируемых данных N=64. Вначале на каждом ядре имеется 16 элементов, и первые несколько этапов полностью выполняются внутри ядер без обмена информацией. От этапа к этапу количество порций уменьшается вдвое, что приводит с некоторого этапа недогрузке части ядер. В конце остается одна пара порций на вершине пирамиды, которая сливается в одном ядре. Этот вариант параллельного алгоритма слияния описан в [2], где также отмечается, что на последних этапах плохо использует вычислительные ресурсы.

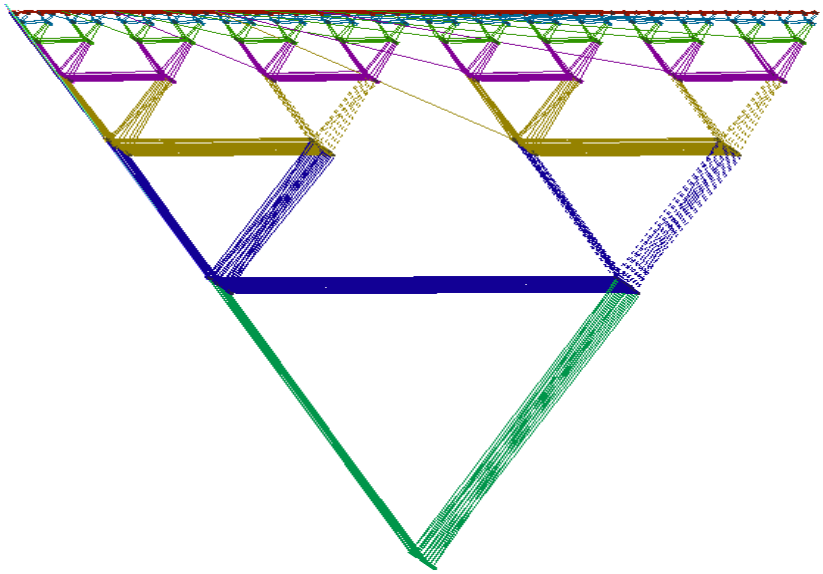

Рис. 2. Визуализация алгоритма по первому режиму

Для того чтобы эффективно использовать все ядра, работа алгоритма разбивается на два режима. Первый режим работает, пока пар порций хватает на все имеющиеся ядра. Начиная с некоторого этапа, переходим во второй режим, в котором каждая порция делится на две части (младшую и старшую) относительно некоторого общего среднего значения, а слияние частей производится в разных ядрах. Среднее определяется как значение медианы в первой порции из группы порций, а в остальных порциях группы положение середины ищется методом деления пополам. Аналогичная идея предложена в [3], но там она используется лишь для групп из двух порций в контексте рекурсивного метода «деления пополам». Поэтому там достигается полноценный параллелизм только на двух процессорах. Мы предлагаем делать то же для групп из любого числа порций. Задача

состоит в том, чтобы в конце работы элементы были распределены между процессорами монотонно по возрастанию их номеров и отсортированы в каждом процессоре.

В начале второго режима имеется одна группа (номер g=0) из $2n_p$ порций, по 2 порции в каждом процессоре. После разделения каждой порции группы на младшую и старшую части процесс попарного слияния порций продолжается как в первом режиме, но теперь он протекает независимо для младших и старших частей порций. Заметим, что все элементы старших частей группы больше всех элементов младших частей этой же группы. Поэтому результаты слияния младших частей (группа номер $2g$) посылаются по две порции на процессоры с номерами от p до $(p+q)/2$, а старших (группа номер $2g+1$) – с номерами от $((p+q)/2)+1$ до q, где $(p..q)$ – диапазон номеров процессоров, занимаемых исходной группой. После каждого этапа число групп удваивается, а число порций в каждой группе уменьшается вдвое. Номер группы g должен быть добавлен в контекст. Общее число порций остается неизменным, что позволяет продолжать загружать все процессоры. На последнее этапе имеется n_p групп, по 2 порции в каждой, которые сливаются и остаются каждая в своем процессоре. Цель сортировки достигнута.

Алгоритм хорошо масштабируется: удвоение количества вычислительных ядер дает почти двойное ускорение. Проблемы возникают в связи с неравномерностью деления порций на младшие и старшие: в результате размеры новых порций на этапах 2-го режима могут отклоняться от их постоянного среднего размера. Поэтому алгоритм будет хорошо работать только на хорошо перемешанных исходных массивах. В этом случае время будет близким к теоретическому минимуму O(N*log(N)). При применении к произвольному массиву полезно сначала выполнить сортировку по случайным тегам, прикрепленным к каждому элементу, а потом вторично отсортировать массив уже по значениям самих элементов. Это увеличит время вдвое, но не более того.

На рис.3 показана визуализация алгоритма, работающего на этапах L=0-2 по первому режиму, а затем с третьего этапа до конца (l=3-6) – по второму режиму. Алгоритм выполняется для N=64 элементов на модели ППВС с n_p=4 вычислительными ядрами.

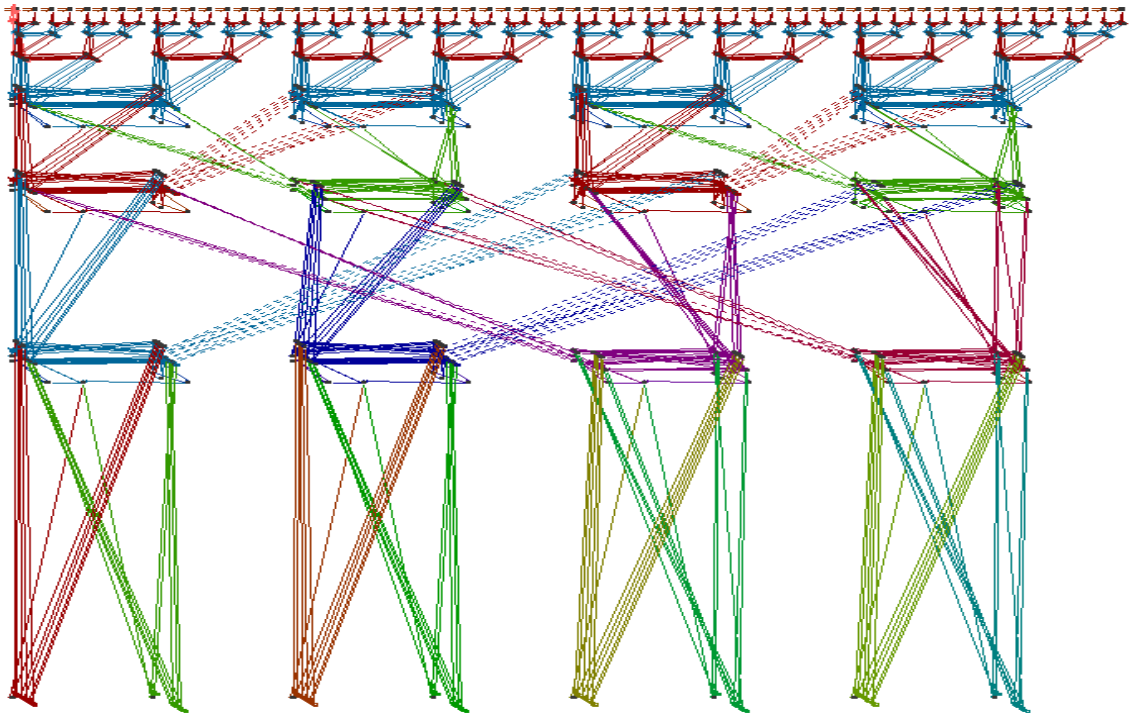

Рис. 3. Визуализация работы алгоритма в двух режимах

Привязку порций к процессорам обеспечивает функция распределения [6], устанавливающая зависимость номера физического процессора от контекста (а в общем случае и от имени узла). Хороший результат дает функция $p=((k/2)<<(L_P+1)+g^-)>>(L_N+l)$, где $L_N=\log_2(N)$, $L_P=\log_2(n_p)$, g^- – номер группы, записанный в двоичном виде в обратном порядке (при длине кода L_P+1), <<, >> – операции сдвига. В этом случае объем пересылок минимален. Передачи в другие ядра показаны пунктиром: они производятся только при передачах на следующий этап и только для половины новых порций. Важно, что каждое слияние пары порций происходит в одном ядре без задержек на пересылки. Задержки при передачах между этапами не играют большой роли, поскольку они выполняются асинхронно и параллельно с выполнением слияний текущего и следующего этапов.

Было проведено сравнение эффективности выполнения данного алгоритма с выполнением программы Integer Sort ($N=2^{23}$, класс A) из пакета NPB-3.0 на вычислительном комплексе МВС-100К МСЦ РАН с использованием технологии MPI. Оказалось, что она масштабируется лишь до 32 процессоров. Работа нашего алгоритма для ППВС с соответствующим количеством данных демонстрирует устойчивое масштабирование до 65 536 вычислительных ядер, превосходя более чем в 100 раз наилучший результат для МВС-100К.

Проведены эксперименты, исследующие зависимость времени прохождения задачи от количества ядер ППВС при различных объемах исходных массивов сортируемых чисел. Результаты этих расчетов приведены на рис. 4. По горизонтальной оси - количество ядер, по вертикальной оси – время выполнения программы (масштаб по обеим координатам логарифмический). На рис. 4 верхняя линия (1) соответствует сортировке N=32768 элементов, вторая линия (2) - N=16384, линия (3) - N=8192 и нижняя линия (4) - N=4096. При N = 4096 работа перестает улучшаться, начиная с 64 ядер, что соответствует 64 элементам на ядро.

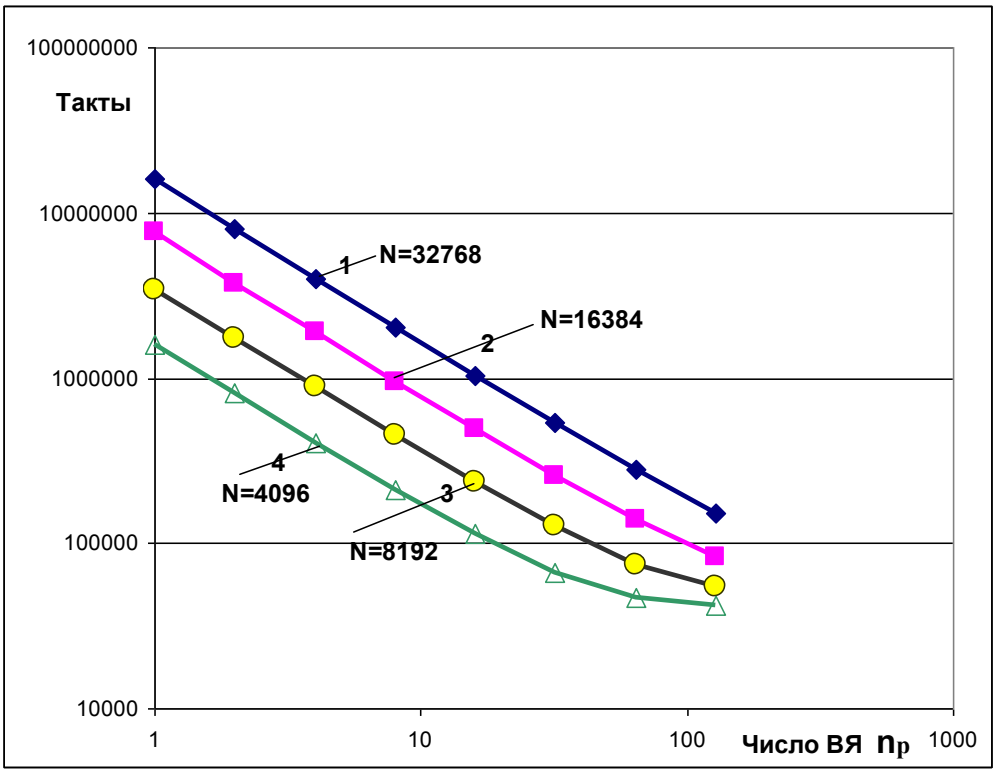

Рис. 4. Время прохождения задачи на ППВС в зависимости от числа ядер

На рис.5 приведены результаты расчетов из тестового пакета NPB программы «сортировка данных по алгоритму Integer Sort» при различных значениях исходных данных на суперкомпьютере МВС-100К.

1. Линия (1) соответствует значению $N=2^{27}$. Видно, что время выполнения программы уменьшается с увеличением количества процессоров до 256, в этом случае на каждый процессор приходится 2^{19}= 524288 чисел. При увеличении числа процессоров время выполнения программы увеличивается.
2. Линия (2) соответствует $N=2^{25}$. Время выполнения программы в этом случае уменьшается с увеличением числа процессоров до 128, в этом случае на каждый процессор приходится 2^{18}= 262144 числа. Дальнейшие увеличение числа процессоров приводит к увеличению времени выполнения программы.
3. Линия (3) соответствует $N=2^{23}$. Время выполнения программы уменьшается с увеличением числа процессоров до 32, на каждый процессор приходится 2^{18}= 262144 числа. Начиная с n_p= 64 процессоров и более, время выполнения программы увеличивается.

На этом же рис. 5 также приведены результаты по алгоритму Merge Sort на ППВС, полученные для аналогичных значений N путем экстраполяции результатов, приведенных на рис.4 (линия 4 – $N=2^{27}$, линия 5 – $N=2^{25}$, линия 6 - 2^{23}).

Сравнивая линии (6) и (3), видим, что до 32-х ядер результаты, полученные на ППВС, уступают результатам, полученным на МВС-100К. Но, начиная с 32 ядер и далее, программа для ППВС показывает лучшие результаты до 131072 ядер, когда на одно ядро приходится всего 64 числа, что более чем в 100 раз быстрее, чем время выполнения программы на MPI.

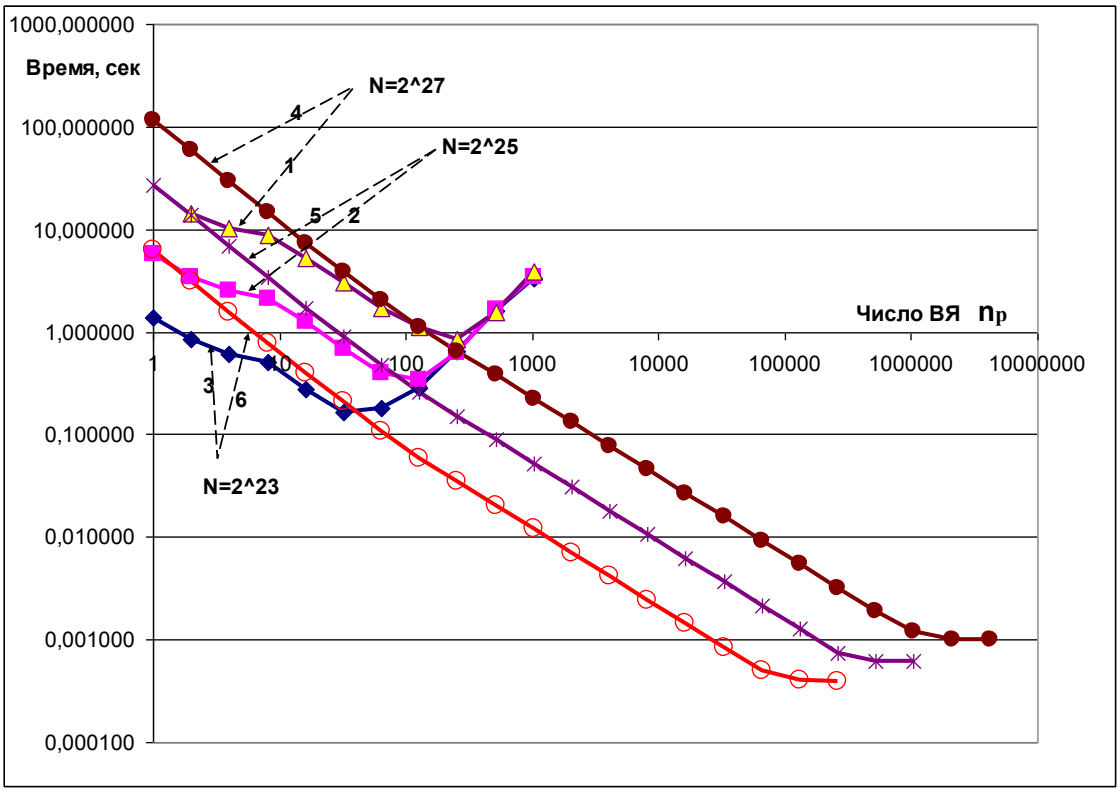

Рис. 5. Время прохождения задачи в зависимости от числа ядер

ЛИТЕРАТУРА

1. Дональд Э.Кнут. Искусство программирования, т.3. Сортировка и поиск 2-е изд.: Пер. с английского – М.: Издательский дом «Вильямс», 2001.
2. Якобовский М.В. Параллельные алгоритмы сортировки больших объемов данных. Москва, 2008, http://lira.imamod.ru/FondProgramm/Sort/ParallelSort.pdf.
3. Dzmitry Huba. Parallel merge sort. 2010. http://dzmitryhuba.blogspot.com/2010/10/parallel-merge-sort.html.
4. Бурцев В.С. Выбор новой системы организации выполнения высокопараллельных вычислительных процессов, примеры возможных архитектурных решений построения суперЭВМ. «Параллелизм вычислительных процессов и развитие архитектуры суперЭВМ», ИВВС РАН, Москва, с. 41-78.
5. Стемпковский А.Л., Левченко Н.Н., Окунев А.С, Цветков В.В. Параллельная потоковая вычислительная система — дальнейшее развитие архитектуры и структурной организации вычислительной системы с автоматическим распределением ресурсов // журнал "Информационные технологии" №10, 2008, с. 2-7.
6. Стемпковский А.Л., Климов А.В., Левченко Н.Н., Окунев А.С. Методы адаптации параллельной потоковой вычислительной системы под задачи отдельных классов // журнал «Информационные технологии и вычислительные системы», 2009. №3, с. 12-21.