УДК 004.021, 004.27 DOI 10.31114/2078-7707-2018-3-16-23

Разработка и исследование алгоритма задачи перемножения разреженных матриц для параллельной потоковой вычислительной системы «Буран»

Д.Н. Змеев, А.С. Окунев

Институт проблем проектирования в микроэлектронике РАН, г. Москва, zmejevdn@ippm.ru, oku@ippm.ru

***Аннотация* — Для того чтобы повысить эффективность выполнения задачи перемножения разреженных матриц на традиционном суперкомпьютере (кластере) необходимо учитывать при программировании разные уровни параллелизма. Используя потоковую модель вычислений с динамически формируемым контекстом и архитектуру параллельной потоковой вычислительной системы можно обойти возникающие в этой связи проблемы. В статье описывается параллельный алгоритм задачи перемножения разреженных матриц для параллельной потоковой вычислительной системы, а также варианты его реализации на потоковом языке высокого уровня, в котором воплощается потоковая модель вычислений. На примере полученного алгоритма показаны отличия в подходах к составлению алгоритмов для потоковых программ. Эксперименты, проведенные на программном инструментальном комплексе системы, показали высокую эффективность потоковой вычислительной системы при решении задач, использующих разреженную структуру данных.**

***Ключевые слова* — перемножение разреженных матриц, потоковая модель вычислений, параллельная потоковая вычислительная система.**

I. Введение

Необходимость расчетов с использованием разреженных матриц возникает при решении оптимизационных задач, при численном решении дифференциальных уравнений в частных производных, в теории графов и во многих других научных и инженерных приложениях. При этом коэффициент заполненности матриц варьируется от миллионных долей процента до нескольких десятков процентов [1].

При работе с разреженными матрицами одной из актуальных и сложных задач (с точки зрения реализации) является операция перемножения этих матриц. Трудоемкость реализации эффективного алгоритма перемножения матриц обуславливается вариативностью структуры их разреженности, что, в свою очередь, влияет на способы хранения и обработки значений элементов матриц.

Эта проблема не так актуальна при реализации алгоритма перемножения разреженных матриц на однопроцессорных вычислительных системах. В этом случае применение последовательного алгоритма обеспечивает надежное, но медленное перемножение матриц с любой структурой разреженности и ограниченной размерностью. При переходе к вычислениям на многопроцессорных вычислительных системах последовательный алгоритм неприменим, и становится актуальной задача создания параллельного алгоритма, эффективно использующего ресурсы многопроцессорной вычислительной системы.

Традиционным подходом к решению этой задачи является создание параллельного алгоритма для вычислительных систем кластерного типа. Для этого применяются такие решения как MPI, CUDA, OpenCL и другие, которые используют императивную парадигму программирования, реализующую фоннеймановскую модель вычислений. Данный подход требует от программиста дополнительных знаний и существенных усилий, так как реализация эффективного параллельного алгоритма сопряжена с необходимостью использования разноуровневого параллелизма – на уровне команд, тредов, вычислительных узлов и на уровне взаимодействия между узлами.

В статье предлагается другой подход к распараллеливанию программ – применение потоковой модели вычислений [2] с динамически формируемым контекстом и реализующей её параллельной потоковой вычислительной системы (ППВС) «Буран» [3].

II. Параллельная потоковая вычислительная система

ППВС представляет собой многоядерную масштабируемую вычислительную систему, где множество вычислительных ядер соединено между собой коммуникационной сетью.

Архитектура ППВС реализует потоковую модель вычислений с динамически формируемым контекстом, в которой процесс вычислений инициируется по готовности данных [4].

Каждое вычислительное ядро ППВС состоит из процессора сопоставления, исполнительного устройства, блока хэш-функций и других устройств. Единицей данных, циркулирующих в системе, является токен. Токен – это структура, состоящая из ключа, данных и признаков. Ключ (в состав которого входит контекст и

номер программного узла) отражает положение данных в виртуальном адресном пространстве задачи, а признаки отвечают за правила сравнения токенов в процессоре сопоставления (основным блоком которого является ассоциативная память ключей). В результате совпадения ключей токенов формируются пакеты. Пакеты, поступая на исполнительное устройство, инициируют выполнение определенного кода программы, генерирующего новые токены. Новые токены, в свою очередь, через коммутационную среду, попадают в соответствующий процессор сопоставления, где формируются новые пакеты. Цикл повторяется до полного исчерпания данных, предназначенных для вычисления.

Главной особенностью системы является аппаратная поддержка распределения данных, которая обеспечивает эффективное использование вычислительных ресурсов. Распределение вычислений по ядрам выполняется при помощи блока хэш-функций, который вычисляет номер ядра для каждого токена на основе значения его ключа. Далее коммутационная среда по вычисленному номеру направляет токен в нужное ядро. Выбор функции распределения для конкретной задачи обеспечивает не только равномерную загрузку вычислительных ресурсов системы, но и минимизацию объемов передаваемых данных между вычислительными ядрами.

III. Разработка и описание алгоритма задачи перемножения матриц

Разработка алгоритмов задач в потоковой парадигме программирования для ППВС «Буран» отличается от разработки алгоритмов в традиционной парадигме программирования [5]. Программа для ППВС представляет собой ориентированный граф, где узлы выполняют роль функций в традиционных языках программирования, а дуги определяют направление передачи токенов, в которых содержатся, помимо данных, условия активации программных узлов.

В отличие от императивной парадигмы программирования (присущей практически всем современным языкам программирования) в потоковой парадигме вычисления начинаются по готовности данных, а не в соответствии с последовательностью инструкций.

Рассмотрим эти отличия на примере алгоритма перемножения матриц для традиционной вычислительной системы и для ППВС «Буран».

Классический последовательный алгоритм перемножения матриц заключается в расчете для каждого элемента результирующей матрицы *C* суммы произведений элементов строки матрицы *A* на элементы столбца матрицы *B*. При реализации данного алгоритма используется двойной цикл для перебора элементов матрицы *C* и вложенный в них цикл суммирования произведений. Сама по себе программа, реализующая этот алгоритм, не является сложной, но скорость ее работы очень низкая. Параллельная реализация данного алгоритма с применением библиотеки MPI [6] или с использованием CUDA [7] является гораздо более сложной, но обеспечивает более быстрое перемножение матриц на многопроцессорных системах.

Описанный выше алгоритм подходит как для перемножения полных матриц, так и для перемножения разреженных матриц (представленных в виде полных матриц). Однако для перемножения разреженных матриц существует множество более оптимальных алгоритмов, которые учитывают структуру разреженности и используют различные способы представления и хранения данных. При этом сложность кода программы прямо пропорциональна эффективности алгоритма.

Для разработки нового потокового алгоритма выделим этапы классического алгоритма. Первое, это перемножение элементов двух матриц. Второе, сложение этих произведений. Третье, выдача результата. Далее построим ориентированный граф с этими этапами в виде узлов и потоками данных между ними в виде дуг. Первый узел, выполняющий умножение, принимает два потока данных – элементы матриц *A* и *B*. Результат умножения направляется на второй узел, выполняющий операцию сложения. Второй поток данных на этот узел формируется самим же узлом, тем самым организуя накопление произведений. Окончательный результат суммирования произведений направляется на третий узел, обозначая тем самым завершение алгоритма.

В приведенном описании алгоритма отсутствует упоминание о способах хранения и механизмах совместного использования общих данных. Оба этих пункта решаются на аппаратном уровне без непосредственного участия программиста. Потоковая концепция вычислений по готовности данных не подразумевает работу с памятью в традиционном смысле – у программиста отсутствуют инструменты выделения памяти, записи в нее данных и чтения из нее. Вместо этого программист путем задания контекста и данного (операнда) токена организует потоки данных на те или иные входы узлов, а в самих узлах описывает правила изменения контекста и данных и указывает направление потока измененных данных. С помощью этого же механизма задания контекста, а также с помощью установки типа и параметров хэш-функции [8], решается вопрос с распределением данных по вычислительным ядрам и использованием общих данных в этих ядрах.

Все это делает описание алгоритма лишь частью общего алгоритма перемножения матриц. Не менее важной частью алгоритма является формирование начальных данных, а именно задание соответствий между значениями элементов и координатами матриц *A* и *B*. Для этого мы разделяем все элементы двух матриц на два потока. Первый поток, состоящий из элементов матрицы *A*, направляется на первый вход узла, в котором вычисляется произведение. Второй поток (элементы матрицы *B*), соответственно, на второй вход.

Следующим шагом является задание контекстов токенов для обоих потоков, причем значения контекстов должны быть такими, чтобы аппаратура, а именно процессор сопоставления, сравнивал контексты и фор-

мировал пакеты только для соответствующих друг другу элементов матриц. На первый взгляд, логичным будет разделение контекста на два поля для задания координат элемента. Однако в этом случае пакеты будут формироваться только элементами матриц с одинаковыми координатами, что является ошибкой. Правильным решением будет использование одного из свойств потоковой модели вычислений – маскирование полей контекста. Это свойство позволяет организовывать множественный отклик при сопоставлении одного токена с несколькими токенами с разными контекстами. В нашем случае, если для обоих потоков замаскировать весь контекст, кроме одного поля, и корректно расставить координаты элементов, то можно добиться желаемого результата.

Остается определить порядок заполнения полей контекстов значениями индексов элементов матриц *A* и *B*. Для этого введем следующие обозначения. Пусть *i* - индекс строки, *j* – индекс столбца, *n* – размер матрицы (для наглядности будем рассматривать квадратную матрицу). Согласно правилу перемножения матриц, для получения значения элемента результирующей матрицы *C(i,j)* требуется сложить произведения элементов *i*-ой строки матрицы *A* и *j*-го столбца матрицы *B*. Сформируем контексты обоих начальных потоков данных соответствующим образом: для каждой ячейки матрицы *A* в «открытом» поле контекста зададим номер строки, а для каждой ячейки матрицы *B* в том же поле – номер столбца. В ППВС «открытое» поле задается с помощью маски, где нулевой элемент в разряде маски открывает значимый разряд, а единица - закрывает незначимый разряд контекста. На этапе сопоставления производится сравнение только «открытых» полей контекстов двух токенов.

На финальном этапе разработки алгоритма выбираем наиболее подходящую хэш-функцию, которая распределяет данные только по «открытым» полям. Так как у нас в контексте есть только одно значимое «открытое» поле, то лучшим выбором является «функция распределения по полю» (FLD), которая работает следующим образом: все множество значений от *0* до *X* распределяется между ядрами равными частями по *x* значений, где *X* - максимальное значение поля, а *x* – число последовательных значений.

В качестве примера рассмотрим распределение матриц *A* и *B* размером 4*4 на 4 ядра. Равномерное распределение достигается при *x=1*. Таким образом, мы получаем, что в *0*-м ядре находятся все элементы *0*-й строки матрицы *A* и *0*-го столбца матрицы *B*, в *1*-м ядре – *1*-я строка матрицы *A* и *1*-й столбец матрицы *B* и так далее.

Однако в этом случае алгоритм не будет выполняться в полном объеме, так как, согласно условиям задачи, каждый элемент, например, *0*-й строки матрицы *A* должен взаимодействовать с соответствующим элементом каждого столбца матрицы *B*, которые, в свою очередь, из-за распределения оказались в разных вычислительных ядрах.

Решение данной проблемы может быть достигнуто копированием всей матрицы *B* во все вычислительные ядра. Осуществить подобное можно путем многократной посылки каждого столбца матрицы *B* со значениями «открытого» поля от *0* до *n-1*. Очевидным отрицательным последствием данного подхода является увеличение требуемого максимального объема ассоциативной памяти ключей (АПК) в процессоре сопоставления каждого вычислительного ядра. Для трех матриц (двух начальных и результирующей) это значение можно рассчитать по формуле:

$$M_k = \frac{2*N^2}{K} + N^2,$$

где M_k – требуемый объем АПК *k*-го ядра, *N* –размер матрицы, *K* – количество вычислительных ядер в системе (при $K \le N$).

Несмотря на то, что представленное решение позволяет решить поставленную задачу, его нельзя назвать удачным. Для поиска более эффективного решения стоит взглянуть на задачу не со стороны некой формулы и ее программной реализации, а со стороны данных, которые могут и должны взаимодействовать для получения требуемого результата.

С этой позиции при умножении двух матриц *j*-й элемент каждой строки матрицы *A* взаимодействует только с элементами *j*-ой строки матрицы *B*. То есть, задавая для каждой ячейки матрицы *A* в «открытом» поле контекста первого потока данных номер столбца, а для каждой ячейки матрицы *B* – номер строки, мы добиваемся того, что при распределении данных в одном ядре окажутся только те ячейки обеих матриц, произведения которых требуется для последующего вычисления значений ячеек результирующей матрицы. При этом требуемый объем АПК для каждого ядра вычисляется по формуле:

$$M_k = \frac{3*N^2}{K}.$$

Подробное рассмотрение способов решения задачи было призвано показать особенности программирования в потоковой парадигме. Причем большая часть описания не касалась непосредственно алгоритма решения задачи, а формировалась вокруг организации начальных данных. Это ключевой момент при создании потоковых программ. Можно отметить, что сам алгоритм фактически не зависит от тех данных, которые он обрабатывает, то есть, если при организации начальных потоков данных сформировать токены только для части матриц, алгоритм задачи будет выполнен и ее решение будет корректным, так как в самом алгоритме отсутствует контроль размеров и заполненность матриц.

Предложенный алгоритм задачи умножения двух матриц является универсальным и может быть использован для умножения двух полных матриц, двух раз-

реженных матриц, матрицы и вектора, а также для умножения матрицы на число.

IV. Варианты реализации алгоритма задачи перемножения матриц

Потоковый алгоритм перемножения матриц был запрограммирован на параллельном языке высокого уровня HPL (High-parallel language) и скомпилирован в систему команд ППВС «Буран». Структурно язык HPL состоит из набора узлов. Каждый узел состоит из заголовка и набора операторов, обрабатывающих входные данные.

На рис. 1 представлен код программы перемножения матриц на языке HPL.

```
type F1 = {k, j, i : int[2]};

node MUL (a, b : int64) F1; dgrouped;
  send a*b to SUM {0, k, i};

node SUM (a, b : int64) F1; symmetric;
  send a+b to SUM;

node RESULT (a : int64) F1;
  send a to HOST;
```

Рис. 1. Код программы на языке HPL

В начале программного кода описывается формат контекста, в котором указываются поля, обозначающие координаты ячеек матриц. Далее следует описание программных узлов. Первый узел MUL имеет два входа (обрабатывает два потока данных) и предназначен для умножения двух целочисленных значений и пересылке результата на узел SUM с измененным контекстом. Ключевое слово **dgrouped** в описании заголовка узла указывает на определенные правила взаимодействия токенов, направленных на этот узел, в процессоре сопоставления. Второй узел SUM также имеет два входа и выполняет операцию сложения двух целочисленных значений. Результат сложения направляется на узел SUM без указания входа. Это возможно благодаря указанию в заголовке узла ключевого слова **symmetric**, которое указывает процессору сопоставления на безразличность положения двух данных в формируемом пакете при взаимодействии двух токенов, направленных на этот узел. Третий узел RESULT перенаправляет данное со своего входа на ХОСТ. Так как в коде программы отсутствует явная посылка данных на этот узел, то для того, чтобы результирующая матрица была выдана на ХОСТ, по завершению вычислений, в систему подается глобальный токен, перенаправляющий все хранящиеся в АПК токены, относящиеся к узлу SUM, на узел RESULT. Эта операция поддерживается на аппаратном уровне системой команд процессора сопоставления.

Представленный код программы также, как и описываемый им алгоритм, является универсальным. В коде программы тип обрабатываемых данных обозначен как целочисленный. Для перемножения вещественных матриц достаточно заменить тип данных входов в заголовках узлов.

Размерность же данных и тип умножения матриц задаются начальными данными. Так, для перемножения двух полных матриц необходимо все элементы матрицы *A* послать на вход *a* узла MUL, а все элементы матрицы *B* – на вход *b*. Для перемножения двух разреженных матриц достаточно послать на те же входы узла MUL только ненулевые элементы матриц. Посылка нулевых элементов для этой задачи не обязательна, так как их участие в расчете результирующей матрицы не приводит к изменению значений ее элементов, а, следовательно, эффективность выполнения задачи будет повышена за счет сокращения затрат вычислительных ресурсов на операции умножения и сложения нулей. Формирование начальных данных для умножения матрицы на вектор полностью идентично подготовке данных для умножения двух полных матриц. А вот для умножения матрицы на число на вход *b* узла MUL необходимо послать глобальный токен с нулевым и полностью замаскированным контекстом.

Как можно заметить, управление начальными данными является неотъемлемой частью программирования задачи в потоковой парадигме. Более того, в самом коде программы отсутствует указание на стартовый узел – в потоковых программах отсутствует само это понятие. Это открывает определенные возможности для последующей оптимизации алгоритма.

```
type F1 = {k, j, i : int[2]};

node MUL (a, b : int64) F1; dgrouped;
  send <tSUM> a*b to RESULT {0, k, i};

node RESULT (a : int64) F1;
  send a to HOST;
```

Рис. 2. Код программы на языке HPL (вариант 2)

Например, при построении спецвычислителя на базе архитектуры ППВС «Буран» одной из задач может быть сокращение нагрузки на исполнительное устройство (ИУ) или полный отказ от него. Для этого необходимо перенести операции умножения и сложения из ИУ в процессор сопоставления (ПС). В ППВС это достигается путем применения специальных токенов, которые на аппаратном уровне заменяют целые цепочки вычислений.

Так, используя специальный токен «Суммирование», можно сократить код программы, удалив из него

описание узла SUM. На рис. 2 показан измененный код программы.

Посылаемый из узла MUL токен с кодом операции «Суммирование» (tSUM) инициирует в процессоре сопоставления специальный механизм обработки токенов. Данные всех токенов, направленных на узел RESULT с одинаковым контекстом, суммируются непосредственно в ПС. Пакет с результатом формируется только тогда, когда будут просуммированы все данные, относящиеся к соответствующей ячейке результирующей матрицы. Таким образом, посылка специального токена из узла MUL сразу на узел RESULT позволяет сократить на треть код программы и снизить нагрузку на ИУ.

Токен tSUM является достаточно универсальным и может применяться не только для рассматриваемой задачи. Для сокращения же оставшихся двух узлов, и в первую очередь узла MUL, невозможно использовать специальный токен с подобной степенью универсальности, так как помимо операции умножения в узле MUL осуществляется перестановка полей контекста. Вместо этого было предложено следующее решение. В блок специальных операций процессора сопоставлений добавляется новый тип токена «Умножение-сложение» (tMADD) с логикой работы, обеспечивающей при взаимодействии двух токенов tMADD формирование специального токена tSUM с соответственно измененным контекстом и результатом умножения двух значений данных провзаимодействовавших токенов.

Введение в систему команд спецвычислителя подобного токена, предназначенного только для выполнения задачи перемножения матриц, позволяет полностью отказаться от ИУ или сократить его до простого устройства, задача которого состоит только в формировании токена для выдачи результата на ХОСТ.

Подводя предварительные итоги, следует отметить, что все три описанных варианта программы реализуют один и тот же алгоритм. При этом все три реализации не зависят от числа и порядка поступления начальных данных. Также они не зависят от количества доступных вычислительных ядер, то есть на аппаратном уровне обеспечивается исполнение программы как на одном, так и на большом количестве ядер без перепрограммирования и повторной компиляции. Все это в полной мере можно отнести к любой программе для ППВС.

V. Эксперименты

Для оценки эффективности разработанного алгоритма и вариантов его реализации был проведен ряд экспериментов.

В первой части экспериментов было проведено сравнение HPL-программы (рис. 1) перемножения матриц с аналогичным по сложности реализации классическим параллельным алгоритмом. Такой реализацией является MPI программа перемножения двух матриц, где нулевые элементы также участвуют в вычислении. Принципиальный отказ от сравнения с параллельной программой, специально разработанной для вычисления разреженных матриц, объясняется несоизмеримыми затратами на разработку последней и применением в ней специальных схем организации данных, призванных ускорить процесс вычислений. В потоковой программе подобные схемы не используются.

Эксперимент проводился на суперкомпьютере кластерного типа «Ломоносов» [9]. HPL-программа запускалась на нем с помощью разработанного эмулятора [10] ППВС, входящего в состав программного инструментального комплекса [11]. На рис. 3 представлены результаты перемножения двух матриц размером 4096*4096 и коэффициентом заполненности K_3 (для HPL версии) 1, 5 и 10 процентов (значения выбраны на основании данных, приведенных в [1]).

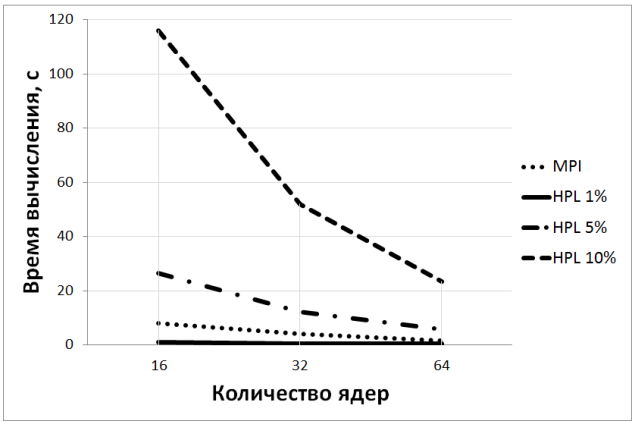

Рис. 3. Сравнение прохождения задачи перемножения разреженных матриц MPI-программы на кластере и HPL-программы на эмуляторе

Анализ полученных результатов позволяет сделать следующие выводы. Для обеих программ время их выполнения пропорционально уменьшается по мере увеличения вычислительных ресурсов. Таким образом, можно говорить об их хорошей масштабируемости. При этом время выполнения потоковой программы уменьшается прямо пропорционально уменьшению значения K_3. Причем, можно заметить, что при K_3 =1% время работы эмулятора значительно меньше времени выполнения MPI-программы. И это с учетом того, что основной объем его работы (в пределах 99.9±0.1%) расходуется на эмуляцию потоковой модели вычислений, то есть на работу, не связанную непосредственно с расчетами. Это происходит потому, что общее число операций при выполнении HPL-программы прямо зависит от количества данных, циркулирующих в системе. Поэтому для того, чтобы не обрабатывать нулевые элементы достаточно не вводить их в систему.

Сравнение прохождения программы на реальной системе (MPI-версия) с программой, исполнение которой эмулируется с помощью специального набора функций (HPL-версия), демонстрирует преимущество первого. Однако если заменить эмуляцию реальной аппаратурой, которая разрабатывается в настоящее

время, то преимущество HPL-версии будет значительно выше, особенно при низкой степени заполненности матриц.

Вторая часть экспериментов была посвящена оценке эффективности предпринятых шагов для улучшения работы потокового алгоритма. Сравнение проводилось между тремя программами: полностью программная реализация алгоритма (рис. 1), вариант реализации алгоритма с использованием специального токена «Суммирование» (рис. 2) и вариант, построенный на применении только специальных токенов.

Данная часть экспериментов проводилась на программной блочно-регистровой модели (ПБРМ), также входящей в состав программного инструментального комплекса. Отличие ПБРМ от эмулятора заключается в потактовом моделировании заданной архитектуры ППВС с явно установленными задержками (для всех элементов системы), максимально приближенными к реальной аппаратуре.

На рис. 4 показано во сколько раз ускоряется выполнение одной и той же задачи с одними и теми же начальными данными при переходе от полностью программной реализации алгоритма перемножения матрицы (вариант 1) к реализации с заменой одного программного узла на один специальный токен (вариант 2) и к реализации, полностью состоящей из специальных токенов.

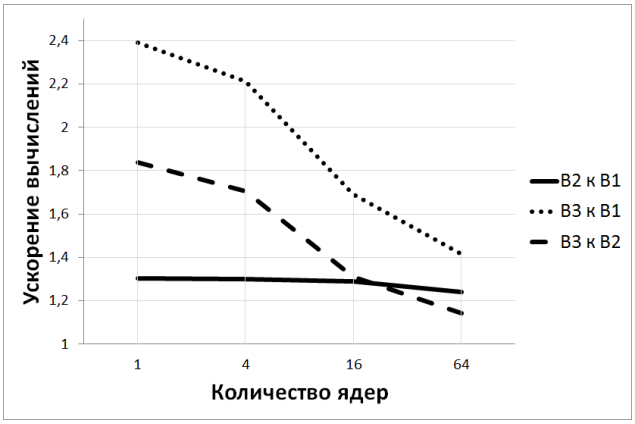

Рис. 4. Ускорение выполнения задачи перемножения матриц при переносе части вычислений из исполнительного устройства в процессор сопоставления. B1 – первый вариант программы (рис. 1), B2 – второй вариант программы (рис. 2), B3 – третий вариант программы (полностью аппаратная реализация)

Полученные результаты показывают, что перевод части программного кода в аппаратуру приводит к значительному ускорению выполнения программы. При этом для полностью аппаратной реализации программы это ускорение нелинейно – на небольшом количестве ядер время выполнения программы более чем в два раза быстрее. Эта неравномерность объясняется следующим. Работа вычислительного ядра представляет собой замкнутый цикл, в рамках которого производится конвейерная обработка токенов – от момента их поступления на вход ядра до момента передачи результатов их обработки на свое или другое ядро. Максимальный эффект от конвейеризации вычислений достигается при отсутствии разрывов между поступлениями данных на этот конвейер. Данное условие выполняется при распределении данных на небольшое количество вычислительных ядер. При максимальном распределении вычислений, когда на каждое ядро приходится минимально возможный объем данных, «недостаток» данных приводит к снижению эффекта от конвейеризации вычислений. Применение специальных токенов в потоковой программе демонстрирует ускорение выполнения задачи на архитектуре ППВС с любым количеством вычислительных ядер.

В целом эксперименты показали, что при аппаратной реализация ППВС «Буран» выигрыш при решении задач, использующих разреженную структуру данных, будет достигать несколько порядков, а специализация этой аппаратуры может повысить производительность еще в несколько раз.

VI. Заключение

При реализации эффективного параллельного алгоритма перемножения разреженных матриц на кластере возникает необходимость учета разных уровней параллелизма для оптимального использования ресурсов многопроцессорной вычислительной системы.

Это требует от программиста дополнительных знаний и существенных усилий. В статье предлагается другой подход – использование потоковой модели вычислений с динамически формируемым контекстом [12]. Базовые принципы архитектуры ППВС, реализующие данную модель, описаны во второй части данной статьи.

Основным результатом данной работы является следующее. Приведенный в статье алгоритм перемножения матриц для ППВС показал, что он является более простым в написании, реализации и более универсальным, чем аналогичная программа для традиционных вычислительных систем. Кроме того, данный алгоритм позволяет перемножать не только плотнозаполненные матрицы, но и разреженные. Можно отметить особенность данной программы: при неизменности кода программы время перемножения разреженных матриц существенно меньше времени перемножения полных матриц. Причем, чем выше коэффициент заполненности матриц, тем существеннее будет эта разница.

Эксперименты, проведенные в рамках данной работы, показали высокую эффективность архитектуры ППВС «Буран» при решении задач, использующих разреженную структуру данных, а также эффективность применения специальных токенов, которые позволяют добиться дополнительного ускорения.

Литература

[1] URL: https://sparse.tamu.edu/ (дата обращения: 18.04.2018)

[2] Климов А.В., Левченко Н.Н., Окунев А.С. Преимущества потоковой модели вычислений в

условиях неоднородных сетей // Журнал «Информационные технологии и вычислительные системы». 2012. № 2. С. 36-45.
[3] 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». 2008. №10. С. 2-7.
[4] Климов А.В., Левченко Н.Н., Окунев А.С., Стемпковский А.Л. Суперкомпьютеры, иерархия памяти и потоковая модель вычислений // Программные системы: теория и приложения: электрон. научн. журн. 2014. Т. 5. № 1(19). С. 15–36. URL: http://psta.psiras.ru/read/psta2014_1_15-36.pdf (дата обращения: 18.04.2018)
[5] Климов А. В., Левченко Н. Н., Окунев А. С., Змеев Д. Н. Стемпковский А. Л. Исследование возможности асинхронной реализации задачи молекулярной динамики на ППВС "Буран" // Журнал "Качество. Инновации. Образование". 2014. № 10. С. 46-51.
[6] ElEnin SA, ElSoud MA. Evaluation of Matrix Multiplication on an MPI Cluster // International Journal of Electric & Computer Sciences IJECS. 2011. V. 11. № 01. P. 50-57.
[7] URL: https://www.quantstart.com/articles/Matrix-Matrix-Multiplication-on-the-GPU-with-Nvidia-CUDA (дата обращения: 18.04.2018)
[8] Змеев Д.Н., Климов А.В., Левченко Н.Н. Средства распределения вычислений в ППВС «Буран» и варианты реализации блока выработки хэш-функций // Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС). М.: ИППМ РАН, 2016. № 2. С. 107-113.
[9] Воеводин Вл.В., Жуматий С.А., Соболев С.И., Антонов А.С., Брызгалов П.А., Никитенко Д.А., Стефанов К.С., Воеводин Вад.В. Практика суперкомпьютера "Ломоносов" // Открытые системы. Москва: Издательский дом "Открытые системы", 2012. № 7. С. 36-39.
[10] Змеев Д.Н., Окунев А.С., Левченко Н.Н., Климов А.В. Реализация параллельной модели вычислений с управлением потоком данных на кластерных суперкомпьютерах // Научный сервис в сети Интернет: все грани параллелизма: Труды Международной суперкомпьютерной конференции (23-28 сентября 2013 г., г. Новороссийск). М.: Изд-во МГУ, 2013. С. 375-377.
[11] Змеев Д.Н. Адаптация традиционных инструментов программирования для моделирования и выполнения задач в парадигме потока данных // Информационные технологии и математическое моделирование систем 2017. Труды международной научно-технической конференции. М.: Федеральное государственное бюджетное учреждение науки Центр информационных технологий в проектировании Российской академии наук, 2017. С. 118-121.
[12] Левченко Н.Н., Змеев Д.Н., Климов А.В., Окунев А.С., Стемпковский А.Л. Перспективы использования потоковой модели вычислений в высокопроизводительных вычислительных системах // Сборник трудов SoRuCom-2017. Четвертая Международная конференция «Развитие вычислительной техники в России и странах бывшего СССР: история и перспективы», 3–5 октября 2017 года, Москва, Зеленоград. 2017. С. 177 – 184.

Development and Investigation of Algorithm of Sparse Matrices Multiplication Task for the Parallel Dataflow Computing System "Buran"

D.N. Zmejev, A.S. Okunev

Institute for Design Problems in Microelectronics of RAS, Moscow, zmejevdn@ippm.ru, oku@ippm.ru

Abstract — **Operations with sparse matrices are relevant in solving scientific and engineering problems. One of these operations is the multiplication of sparse matrices. There are many algorithms for traditional computing systems that solve this task. A sequential algorithm, consisting of nested loops system, is inefficient and is only applicable for single-processor systems. Parallel algorithms are much more efficient and run on supercomputers and clusters. The creation of such algorithms requires certain knowledge of parallel programming and taking into account different levels of parallelism, and the creation of an effective program also requires complex system of data organization. Other approach for solving the task of sparse matrices multiplication is considered in the article - the application of the parallel dataflow computing system (PDCS) "Buran" that implements the dataflow computing model with a dynamically formed context. The process of developing an algorithm in dataflow programming paradigm, which is completely different from the traditional (imperative) one, is shown. The dataflow algorithm consists of two parts - the description of program nodes and the formation of initial data. Coding of program nodes is carried out using the high-level programming language HPL (High-parallel language). The resulting program code, as well as the algorithm described by it, is simple and universal. The program allows the multiplication of two full/sparse matrices or the multiplication of the matrix by a vector or number. The choice of multiplication objects is determined at the stage of initial data formation. When creating dataflow programs, hardware acceleration is possible, which allows reducing the code of the program or completely refusal the use of program nodes. This is also set up during the stage of initial data formation. The use of this programming approach allowed the creation of three dataflow programs: fully programmed implementation, fully hardware**

implementation and an intermediate version. The experimental part of the work was divided into two parts. In the first part, the dataflow program and the program created using traditional methods were compared. The comparison was performed on the "Lomonosov" cluster supercomputer and the results showed a high efficiency of the dataflow program when solving tasks with a sparse data structure and with equal programming complexity. In the second part of the experiments, three implementation variants of dataflow algorithm were compared. The results of the comparison showed the high efficiency of applying special approaches to programming, especially when creating small (in terms of the number of computational cores) special computers. In general, the article has demonstrated the following: the potential of using the PDCS "Buran" for solving tasks with a sparse data structure; the simplicity of programming; and the versatility and high efficiency of the created dataflow programs.

***Keywords* — sparse matrices multiplication, dataflow computing model, parallel dataflow computing system.**

REFERENCES

[1] The SuiteSparse Matrix Collection. Available at: https://sparse.tamu.edu/ (accessed: 18.04.2018).

[2] Klimov A.V., Levchenko N.N., Okunev A.S. Advantages of dataflow computing model in a non-homogeneous networks. Zhurnal «Informacionnye tehnologii i vychislitel'nye sistemy», 2012, no. 2, pp. 36-45 (in Russian).

[3] Stempkovskij A.L., Levchenko N.N., Okunev A.S., Cvetkov V.V. Parallel dataflow computing system - the further development of architecture and the structural organization of the computing system with automatic distribution of resources. Zhurnal «INFORMACIONNYE TEHNOLOGII», 2008, no. 10, pp. 2-7 (in Russian).

[4] Klimov A.V., Levchenko N.N., Okunev A.S., Stempkovskij A.L. Supercomputers, memory hierarchy and dataflow computing model. Programmnye sistemy: teorija i prilozhenija: jelektron. nauchn. Zhurn, 2014, vol. 5, no. 1(19), pp. 15–36 (in Russian). Available at: http://psta.psiras.ru/read/psta2014_1_15-36.pdf (accessed: 18.04.2018).

[5] Klimov A.V., Levchenko N.N., Okunev A.S., Zmeev D.N. Stempkovskij A.L. The research of the asynchronous implementation of molecular dynamics on pdcs "Buran". Zhurnal "Kachestvo. Innovacii. Obrazovanie", 2014, no. 10, pp. 46-51 (in Russian).

[6] ElEnin SA, ElSoud MA. Evaluation of Matrix Multiplication on an MPI Cluster. International Journal of Electric & Computer Sciences IJECS, 2011, vol. 11, no. 01, pp. 50-57.

[7] Matrix-Matrix Multiplication on the GPU with Nvidia CUDA. Available at: https://www.quantstart.com/articles/Matrix-Matrix-Multiplication-on-the-GPU-with-Nvidia-CUDA (accessed: 18.04.2018)

[8] Zmeev D.N., Klimov A.V., Levchenko N.N. The means for computation distribution in the PDCS "Buran" and the implementation variants of a block of hash-functions. Problemy razrabotki perspektivnyh mikro- i nanojelektronnyh sistem (MJeS), 2016, no. 2, pp. 107-113 (in Russian).

[9] Voevodin Vl.V., Zhumatij S.A., Sobolev S.I., Antonov A.S., Bryzgalov P.A., Nikitenko D.A., Stefanov K.S., Voevodin Vad.V. Practice of "Lomonosov" Supercomputer. Otkrytye sistemy. Moskva: Izdatel'skij dom "Otkrytye sistemy", 2012, no. 7, pp. 36-39 (in Russian).

[10] Zmeev D.N., Okunev A.S., Levchenko N.N., Klimov A.V. Implementation of dataflow computing model on cluster supercomputers. Nauchnyj servis v seti Internet: vse grani parallelizma: Trudy Mezhdunarodnoj superkomp'juternoj konferencii (23-28 sentjabrja 2013 g., g. Novorossijsk) – Proc. Int. Conf. “Scientific service on the Internet: all facets of parallelism”. Moscow: MSU, 2013, pp. 375-377 (in Russian).

[11] Zmeev D.N. Adaptation of traditional programming tools for modeling and executing tasks in the dataflow paradigm. Informacionnye tehnologii i matematicheskoe modelirovanie sistem 2017: Trudy mezhdunarodnoj nauchno-tehnicheskoj konferencii – Proc. Int. Conf. “Information Technologies and Mathematical Modeling of Systems 2017”. Moscow, 2017, pp. 118-121 (in Russian).

[12] Levchenko N.N., Zmeev D.N., Klimov A.V., Okunev A.S., Stempkovskij A.L. Prospects for using dataflow computing model in high-performance computing systems. Sbornik trudov SoRuCom-2017: Chetvertaja Mezhdunarodnaja konferencija «Razvitie vychislitel'noj tehniki v Rossii i stranah byvshego SSSR: istorija i perspektivy» – Proc. 4th Int. Conf. “Computer Technology in Russia and in the Former Soviet Union”. Moscow, 2017, pp. 177 – 184 (in Russian).