УДК 519.68; 519.17 doi 10.26089/NumMet.v19r108

РЕАЛИЗАЦИЯ МОДЕЛИ АССОЦИАТИВНЫХ ВЫЧИСЛЕНИЙ НА GPU: БИБЛИОТЕКА БАЗОВЫХ ПРОЦЕДУР ЯЗЫКА STAR

Т. В. Снытникова[1]

Ассоциативные (контекстно адресуемые) параллельные процессоры типа SIMD с вертикальной обработкой информации ориентированы на решение задач нечисловой обработки данных. Моделирование работы таких систем описывается с помощью абстрактной модели типа SIMD (STAR-машины). На этой модели были разработаны эффективные алгоритмы для решения многих задач на графах. Однако из-за отсутствия широко распространенных ассоциативных архитектур эти алгоритмы не могли применяться на практике. С развитием графических ускорителей появилась возможность реализовывать ассоциативные параллельные модели без существенной потери эффективности. В качестве первого этапа реализации STAR-машины на графических ускорителях в виде библиотеки на CUDA были реализованы специфические для языка STAR типы данных и простейшие операции над ними. В настоящей статье приводится эффективная реализация на GPU библиотеки стандартных процедур языка STAR. Проведено сравнение времени работы данной реализации с временем работы процедур из стандартных библиотек (STL на CPU и CUDA thrust на GPU), выполняющих эти же операции. Планируется использовать представленную реализацию STAR-машины на GPU для решения задач на графах.

Ключевые слова: вертикальная обработка данных, модель ассоциативного параллельного процессора, графический ускоритель, высокопроизводительные вычисления.

1. Введение. Ассоциативные параллельные процессоры (АПП) типа SIMD (Single Instruction, Multiple Data) с вертикальной обработкой информации и с простейшими процессорными элементами (ПЭ) выполняют массовый параллельный поиск по содержимому памяти и используют двумерные таблицы в качестве базовой структуры данных. Такая архитектура ориентирована на решение задач нечисловой обработки. Сюда относятся теория графов, реляционные базы данных, базы знаний, экспертные системы, обработка сейсмических данных и др.

К классу ассоциативных параллельных архитектур относятся системы STARAN (Stellar Attitude Reference and Navigation) [1, 2], ASPRO (Airborne Associative Processor) [3, 4], IXM2 [5], Rutgers CAM (Content-Addressable Memory) [6, 7] и современная многопроцессорная система, основанная на ассоциативной памяти, — ATLAS Fast TracKer (FTK) [8, 9]. Все системы были спроектированы для решения конкретных задач, которые не могли быть решены на системах другой архитектуры: ASPRO — для задач контроля воздушного движения (в том числе использовался в радарах самолетов-разведчиков E-2 Hawkeye AWACS ВМС США), IXM2 — для систем машинного перевода (в частности, для перевода устной речи в режиме реального времени [10–12]), ATLAS FTK — триггер первого уровня детектора ATLAS Большого Адронного Коллайдера.

Кроме того, были разработаны модели ассоциативной параллельной обработки данных: STAR-машина [14], модели Поттера ASC (Aspect-Scale-Context) [15] и MASC (Multiple ASsociative Computing) [16].

Заметим, что АПП позволяют строить эффективные алгоритмы для различных приложений. Так, для STAR-машины были построены как ассоциативные параллельные версии ряда известных последовательных алгоритмов на графах, так и новые ассоциативные параллельные алгоритмы на графах, в том числе динамические. Для *ориентированных* графов разработаны ассоциативные версии алгоритма Уоршалла и алгоритма Флойда [17], алгоритмов Дейкстры [18] и Беллмана–Форда [19] и алгоритма Эдмондса [20]. Для *неориентированных* графов построены ассоциативные версии алгоритмов Краскала и Прима–Дейкстры [21] и алгоритма Габова [22]. Были построены ассоциативные параллельные алгоритмы для динамической обработки MST (Minimum Spanning Tree) [23–25], для динамической обработки транзитивного замыкания ориентированного графа [26], для динамической обработки подграфа кратчайших путей с одним стоком [27, 28].

[1] Институт вычислительной математики и математической геофизики СО РАН, просп. Лаврентьева, 6, 630090, Новосибирск; мл. науч. сотр., e-mail: snytnikovat@ssd.sscc.ru

Однако пока нет широко используемых ассоциативных архитектур, позволяющих реализовывать эти алгоритмы. Поэтому ведутся работы по реализации ассоциативных моделей на доступных архитектурах [29–32]. Одна из архитектур, подходящих для эффективной реализации ассоциативных параллельных моделей, — это графические ускорители (GPU). Эта архитектура относится к классу SIMD и доступна для вычислений.

В работе [32] приведена реализация на GPU типов данных и простейших операций языка STAR. Для обоснования ее эффективности там же была представлена реализация ассоциативной версии алгоритма Уоршалла, имеющая порядок сложности близкий к линейному вместо кубической сложности последовательной версии. В настоящей работе приводится эффективная реализация на GPU библиотеки стандартных процедур языка STAR. Под эффективностью понимается сохранение при реализации свойств ассоциативной параллельной модели.

Во втором разделе описывается модель STAR-машины и рассматриваются особенности моделирования. В третьем разделе описывается библиотека базовых ассоциативных процедур языка STAR. В четвертом разделе рассматриваются особенности реализации библиотеки на графических ускорителях. В пятом разделе приводится сравнение времени работы процедур библиотеки с аналогичными процедурами библиотек STL (Standard Template Library) и CUDA thrust.

2. Модель STAR-машины и особенности ее моделирования на GPU. STAR-машина — абстрактная модель типа SIMD (Single Instruction Multiple Data) с вертикальной обработкой данных была представлена в работах [14, 33].

STAR-машина состоит из следующих частей (рис. 1): последовательного устройства управления (ПУУ), в котором записаны программа и скалярные константы; матричной памяти и устройства ассоциативной обработки (УАО), состоящего из p одноразрядных процессорных элементов.

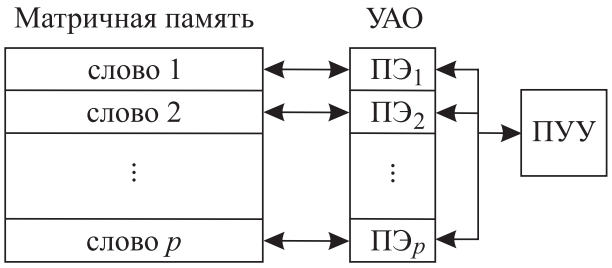

Рис. 1. Модель STAR-машины

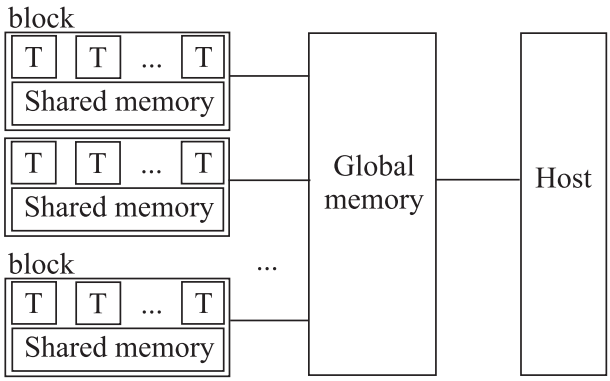

Рис. 2. Модель GPU

Входные данные, записанные в двоичном коде, помещаются в матричную память в виде двумерных таблиц, причем каждая единица данных хранится в отдельной строке и обрабатывается отдельным процессорным элементом. Строки каждой таблицы нумеруются сверху вниз, а столбцы — слева направо. В матричную память можно загружать несколько таблиц.

Устройство ассоциативной обработки представляется в виде совокупности h вертикальных одноразрядных регистров длины p. Вертикальный регистр можно представлять как массив, состоящий из одного столбца. Обработка информации происходит следующим образом. Из определенной таблицы в определенном порядке извлекаются ее одноразрядные столбцы и помещаются в вертикальные регистры устройства ассоциативной обработки, в котором выполняются побитовые операции. Результат выполнения любой операции записывается либо в некоторый регистр, либо в определенный столбец обрабатываемой таблицы, либо в матричную память.

В работе [15] Поттер выделил основные свойства ассоциативных параллельных моделей и архитектур, которым, в частности, удовлетворяет STAR-машина. Ключевыми для моделирования на других типах архитектур являются следующие свойства ассоциативных параллельных моделей.

1. Мелкозернистость: каждое слово данных обрабатывается отдельным ПЭ.

2. Синхронизация: последовательное устройство управления (ПУУ) передает инструкцию всем ячейкам; активные ячейки выполняют команду, полученную от ПУУ, в то время как неактивные ячейки принимают команду, но не выполняют ее.

3. Быстрый обмен результатом вычисления между ПЭ: ПУУ может дать команду выбранному ПЭ передать данные по шине, все другие ПЭ получают значение с шины.

На графических ускорителях можно смоделировать эти свойства следующим образом:

1) каждый ПЭ моделируется отдельным потоком вычислений в блоке (T на рис. 2), слово матричной памяти моделируется элементом массива данных в глобальной памяти (индекс элемента в массиве выражается через индекс потока/блока);

2) на графический ускоритель инструкции от процессора поступают пакетом (через вызов __global__-процедуры); в __global__- процедурах предусмотрена возможность синхронизации по потокам одного блока, но возможность синхронизации по блокам отсутствует; при необходимости синхронизации вычислений инструкции должны быть разнесены в разные __global__- процедуры, при этом накладные расходы на запуск __global__- процедур занимают около 15–20μc;

3) данные передаются в глобальную память, при этом возможны два варианта: через копирование данных на процессор (Host на рис. 2) или через передачу указателя на данные.

Для эффективной реализации STAR-машины на GPU должны выполняться следующие свойства:

— константное время выполнения глобальных операций: побитовые логические операции, доступ к столбцам и строкам матричной памяти как на чтение, так и на запись и другие базовые операции языка STAR [34]; в [32] приведена эффективная реализация базовых операций языка STAR и показано, что базовые операции, не требующие синхронизации по данным, выполняются за константное время;

— ПУУ может выбрать старшую ячейку из множества активных ячеек за единицу времени; реализация данной операции критична к синхронизации по данным; в [32] приведена реализация этой базовой операции с временно́й сложностью $O\Big(\lceil \log_{64}(N) \rceil\Big)$, где N — число строк таблицы;

— базовые операции поиска (=, <, >, min, max) и арифметические операции выполняются за время, пропорциональное числу битовых столбцов в таблице, а не числу ее строк; эти операции выполняются процедурами библиотеки стандартных процедур языка STAR [33]; особенности их реализации приводятся ниже.

Таким образом, будет показано, что с помощью графических ускорителей можно реализовать STAR-машину с сохранением всех ее основных свойств.

3. Описание библиотеки базовых ассоциативных процедур языка STAR. Библиотека базовых ассоциативных процедур широко используется в ассоциативных алгоритмах обработки графов, использующих представление графов в виде списка ребер или матрицы весов. Ее эффективная реализация позволяет предполагать, что реализации ассоциативных алгоритмов на GPU будут производить вычисления быстрее неассоциативных параллельных алгоритмов.

Библиотека включает процедуры для нечисловой обработки, процедуры для числовой обработки и процедуры копирования данных.

3.1. Процедуры для нечисловой обработки данных. Эти процедуры используют управляющий слайс (битовый столбец), в котором отмечены позиции анализируемых строк соответствующей матрицы.

Процедуры для нахождения позиции строк заданной матрицы T, в которых записан минимальный и максимальный элемент: MIN(T, X, Z) и MAX(T, X, Z).

Процедуры, которые по заданной матрице T, по образцу v и по управляющему слайсу X находят позиции строк, удовлетворяющих условию поиска: MATCH(T, X, v, Z) для ROW(j, T)=v, LESS(T, X, v, Y) для ROW(j, T)<v и GREAT(T, X, v, Y) для ROW(j, T)>v.

Процедура GEL(T, v, Y, Z) обобщает процедуры LESS и GRAET: в слайсе Y отмечены позиции строк, которые больше образца, а в слайсе Z — меньше образца.

Процедуры, которые по двум матрицам T и F и по управляющему слайсу X находят позиции строк, удовлетворяющих условию поиска: HIT(T, F, X, Z) для ROW(j, T)=ROW(j, F), SETMIN(T, F, X, Z) для ROW(j, T)<ROW(j, F) и SETMAX(T, F, X, Z) для ROW(j, T)>ROW(j, F).

3.2. Процедуры для числовой обработки. Процедуры ADDV(T, F, X, R) для построчного сложения матриц и ADDC(T, X, v, R) для добавления к строкам матрицы T двоичного слова v (строки матрицы R, которые отмечены ‘0’ в слайсе X, состоят из нулей). Процедура ADDC1(T, X, v, R) отличается от

ADDC тем, что в строки матрицы R, которые отмечены ‘0’ в слайсе X, записывает соответствующие строки матрицы T. Аналогично определены процедуры вычитания SUBTV(T, R, X, R), SUBTC(T, X, v, R) и SUBTC1(T, X, v, R).

3.3. Процедуры копирования. Процедура CLEAR(n, T) записывает нули во все n столбцов матрицы T. Процедура WCOPY(v, X, F) записывает двоичное слово v в те строки матрицы F, которые отмечены ‘1’ в слайсе X. Остальные строки матрицы F состоят из нулей. Процедура WMERGE(v, X, F) записывает двоичное слово v в те строки матрицы F, которые отмечены ‘1’ в слайсе X, при этом остальные строки матрицы F не меняются. Процедура TCOPY(T, F) копирует все столбцы матрицы T в соответствующие столбцы матрицы F. Размеры матриц совпадают. Процедура TCOPY1(T, j, h, F) выделяет из матрицы T j-ю полосу шириной h и копирует ее в матрицу F. TCOPY2(T, j, h, F) копирует матрицу T в j-ю полосу матрицы F. Процедура TMERGE(T, X, F) записывает строки матрицы T, отмеченные ‘1’ в слайсе X, в соответствующие строки матрицы F. Остальные строки матрицы F не меняются.

4. Особенности реализации библиотеки базовых ассоциативных процедур. Поскольку в STAR-машине и в GPU данные обрабатываются по-разному, то в реализации используется особое представление данных. Кроме того, при реализации ассоциативных процедур на GPU необходимо оптимизировать алгоритмы по двум характеристикам: частота синхронизации и независимость данных по столбцам.

Рис. 3. Представление данных

4.1. Представление данных в реализации STAR-машины. На рис. 3 показаны представление структуры vector<int> из 64-х элементов в десятичном и бинарном виде и представление этих же данных для реализации STAR-машины. В структуре vector<int> каждое из целых чисел массива представляет собой 32-разрядную бинарную строку. Для использования в реализации STAR-машины такое представление не подходит, поскольку обработка происходит не по строкам, а по столбцам. Поэтому каждый столбец из 64-х битов составляет одно 64-разрядное целое число (количество 64-разрядных слов для представления столбца рассчитывается по его длине). За счет такого вертикального представления данных пустые разряды можно не включать в таблицу для экономии памяти и уменьшения времени работы, так как количество столбцов обычно входит в качестве параметра в оценку временно́й сложности алгоритма. Вертикальное представление дает также возможность использовать слова произвольной ширины.

4.2. Оптимизация библиотеки базовых процедур под архитектуру GPU. Базовые алгоритмы можно разделить на следующие группы по способу работы с таблицами.

I. В алгоритме используются базовые операции, критичные к синхронизации: MIN и MAX. Алгоритмы этой группы при реализации разбиваются на несколько __global__-процедур.

II. В алгоритме не используются следующие базовые операции, критичные к синхронизации: MATCH, LESS, GREAT, GEL, HIT, SETMIN и SETMAX. Алгоритмы этой группы могут быть представлены в виде __device__-процедур. Это позволяет встраивать их в реализации других алгоритмов без накладных расходов, снижающих производительность.

III. Арифметические алгоритмы: ADDV, ADDC, ADDC1, SUBTV, SUBTC и SUBTC1. Арифметические алгоритмы при вычислении используют слайс переноса на следующий разряд. Для проверки корректности вычислений необходимо убедиться, что после вычислений в нем нет единичных элементов. Для этого используется базовая операция, критичная к синхронизации. Таким образом, арифметические алгоритмы занимают промежуточное положение между I и II группой и представляются

в виде одной выполняющей действие __device__-процедуры, которая упакована в __global__-процедуру, после чего вызывается операция, проверяющая корректность вычисления.

IV. В алгоритме не используются базовые операции, критичные к синхронизации. Столбцы могут обрабатываться независимо друг от друга: CLEAR, WCOPY, WMERGE, TCOPY, TCOPY1, TCOPY2 и TMERGE. Для этой группы алгоритмов столбцы можно обрабатывать как поочередно (как во II группе алгоритмов), так и одновременно (двухуровневый параллелизм), поскольку нет зависимости по данным. В последнем случае при достаточном количестве вычислительных ресурсов процедуры выполняются за константное время.

5. Сравнение производительности библиотеки базовых процедур языка STAR с процедурами библиотек STL и CUDA thrust. Для оценки производительности реализации библиотеки базовых процедур проводится сравнение времени работы со временем работы аналогичных процедур из других библиотек. Для сравнения были выбраны STL — стандартная библиотека языка C++ и CUDA thrust — библиотека шаблонов C++ для CUDA.

Библиотека STL была разработана для того, чтобы предоставить пользователям надежный и эффективный инструментарий. Библиотека CUDA thrust представляется как мощная библиотека алгоритмов для параллельных расчетов и структур данных с использованием GPU. Поэтому они выбираются как эталон для реализации библиотеки базовых ассоциативных процедур: сравнение с библиотекой STL позволяет оценить накладные расходы алгоритмов, связанные с вычислением на графических ускорителях; сравнение с библиотекой CUDA thrust позволяет оценить возможные преимущества ассоциативных алгоритмов перед другими параллельными реализациями на GPU.

Все расчеты проводились на графической карте NVIDIA GEFORCE 920M и процессоре CORE i5.

Следует отметить следующее.

1. Ассоциативные алгоритмы используют другое представление данных.

2. Сравниваемые алгоритмы часто отличаются входными и выходными параметрами.

Во-первых, в ассоциативных алгоритмах в качестве параметра используется управляющий слайс, в котором отмечены позиции обрабатываемых строк. Поэтому алгоритмически нет различия между обработкой всего множества данных или какого-то их подмножества. В алгоритмах из библиотек STL [35] и CUDA thrust [36] время обработки всех строк может существенно отличаться от времени обработки какого-то их подмножества (предикатная форма).

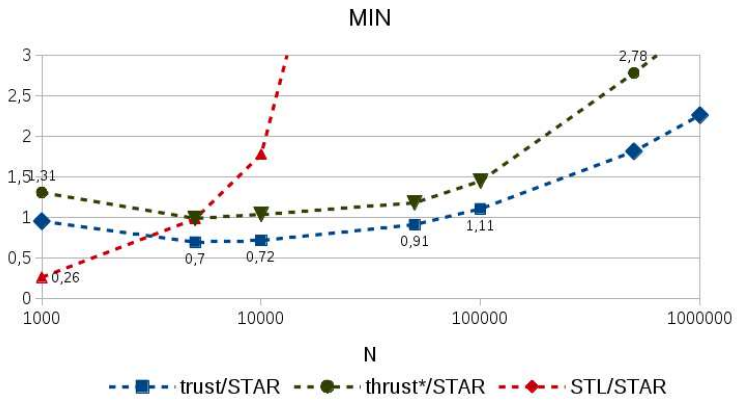

Рис. 4. Отношение времени работы реализаций

Во-вторых, в ассоциативных алгоритмах поиска в качестве выходных данных используется слайс, в котором отмечены все вхождения искомого элемента. Алгоритмы библиотек STL и CUDA thrust выдают только одно вхождение искомого элемента.

В этой связи рассматриваемое сравнение реализаций носит условный характер, тем не менее оно дает некоторое представление о производительности.

5.1. I группа алгоритмов: MIN. Из первой группы алгоритмов рассмотрим процедуру MIN. Теоретическая сложность реализации ассоциативного алгоритма: $O\left(h\cdot\lceil\log_{64}(N)\rceil\right)$, где h — количество столбцов обрабатываемой таблицы, N — количество строк.

Приведем сравнение производительности следующих реализаций:

- STAR — реализация ассоциативного алгоритма MIN;
- STL — последовательная реализация std::min_element() из библиотеки STL;
- thrust — реализация на CUDA thrust::min_element() библиотеки thrust;
- thrust* — реализация на CUDA с использованием библиотеки thrust выдает вектор, в котором помечены позиции всех минимальных элементов.

На рис. 4 показано отношение времени работы реализаций ко времени работы реализации ассоциативного алгоритма на данных разного размера (N — количество элементов массива/количество строк в

таблице). Из рисунка видно, что параллельные реализации дают выигрыш по времени при $N \geqslant 5\,000$. Реализации min и thrust_min_element* дают сравнимый результат на диапазоне $N \leqslant 50\,000$. Реализация min дает заметный выигрыш по производительности относительно thrust::min_element при $N \geqslant 100\,000$.

5.2. II группа алгоритмов: MATCH. Из алгоритмов II группы будем проводить сравнение на алгоритме MATCH. Оценки теоретической сложности ассоциативного алгоритма и его реализации на графическом ускорителе совпадают: $O(h)$, где h — ширина строки таблицы.

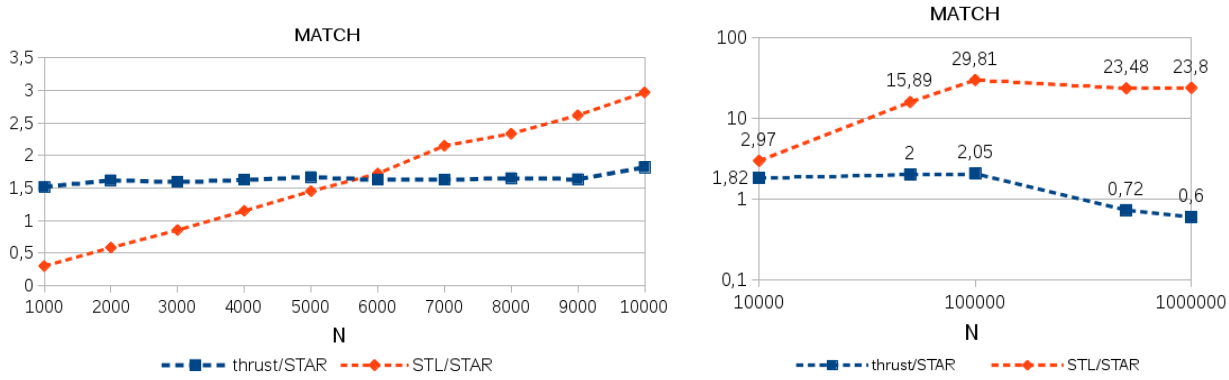

Рис. 5. Сравнение времени работы алгоритмов поиска

Приведем сравнение следующих реализаций:

- STAR — реализация ассоциативного алгоритма MATCH;
- STL — процедура std::find();
- thrust — процедура thrust::find().

Отметим, что алгоритмы библиотек STL и thrust передают указатель на одно из вхождений искомого элемента и поиск проводится по всем элементам массива. Из рис. 5 видно, что последовательный алгоритм из библиотеки STL работает быстрее для векторов небольшой размерности (до 6 000 элементов), но с ростом размерности значительно уступает параллельным реализациям.

В случае, когда все блоки могут обрабатываться одновременно ($N \leqslant 100\,000$ на GEFORCE 920M), реализация ассоциативного алгоритма MATCH работает в 1.5–2 раза быстрее, чем thrust::find().

5.3. III группа алгоритмов: SUBTV. Из группы арифметических алгоритмов рассмотрим процедуру поэлементного вычитания массивов. Теоретическая сложность ассоциативной процедуры SUBTV оценивается как $O(h)$, где h — ширина таблицы, и не зависит от длины таблицы N. Теоретическая сложность реализации на графическом ускорителе: $O\bigl(h + \lceil \log_{64}(N) \rceil\bigr)$.

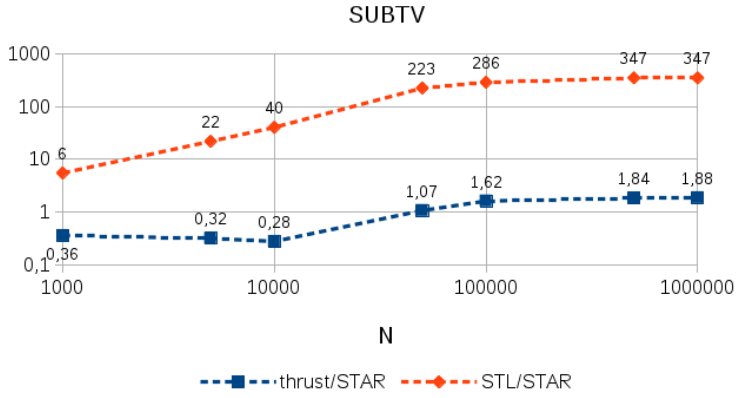

Рис. 6. Отношение времени работы реализации

Проводится сравнение производительности следующих реализаций:

- STAR — реализация ассоциативного алгоритма SUBTV;
- STL — процедура std::transform(...,std::minus<int>());
- thrust — процедура thrust::transform(...,thrust::minus<int>()).

В библиотеках STL и thrust есть процедура transform_if(...), предикатный аналог используемой процедуры, но время его работы отличается не существенно, поэтому рассматриваться не будет.

Из рис. 6 видно, что реализация ассоциативного алгоритма работает медленнее реализации из библиотеки thrust на данных при $N \leqslant 10\,000$, работает сравнимое время при $N \approx 50\,000$ и работает быстрее

на данных большего размера. Последовательная реализация сильно уступает в производительности параллельным.

5.4. IV группа алгоритмов: TMERGE. Для этой группы алгоритмов столбцы можно обрабатывать как поочередно, так и одновременно, поскольку нет зависимости по данным. Поэтому сравним оба варианта:

- tmarge1D: столбцы обрабатываются поочередно, как во II группе алгоритмов;
- tmarge2D: столбцы обрабатываются одновременно ($gridDim.y = h$).

Приведем сравнение со следующими реализациями:

- STL: std::copy() проводит копирование всех элементов;
- STL_if: copy_if проводит копирование только тех элементов, которые удовлетворяют некоторому предикату (данная процедура не была включена в библиотеку STL, ее реализация была взята из [35]);
- thrust: thrust::copy() проводит копирование всех элементов;
- thrust_if thrust::copy_if() проводит копирование только тех элементов, которые удовлетворяют некоторому предикату.

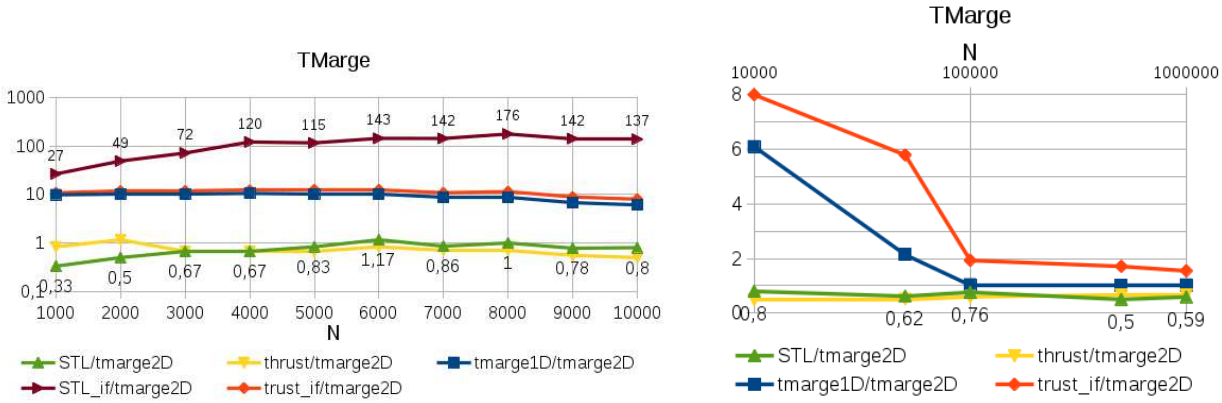

Рис. 7. Отношение времени работы реализаций

Из рис. 7 видно следующее. Реализация tmarge2D сравнима по времени с беспредикатными версиями стандартных библиотек STL и thrust (4–6 μc на векторах до 10 000 элементов) и существенно выигрывает в производительности у предикатных. Реализация с поочередным обрабатыванием столбцов tmarge1D сравнима в работе с thrust_if на векторах до 10 000 элементов (tmarge1D: 60 μc, thrust_if: 70–80 μc) и совпадает с tmerge2D на векторах более 100 000 элементов, когда ресурсов не достаточно для физически одновременного исполнения блоков.

По результатам проведенного сравнения можно сделать следующие выводы.

- Для векторов размерности более 5 000 элементов реализации представленных алгоритмов на GPU работают быстрее процедур библиотеки STL. Для векторов длиной 10 000–50 000 библиотека базовых ассоциативных процедур дает ускорение на 1–2 порядка по сравнению с библиотекой STL.
- Библиотека базовых ассоциативных алгоритмов зачастую выигрывает в производительности у CUDA thrust в 1.5–2 раза.

6. Заключение. Несмотря на то что ассоциативные параллельные системы не получили широкого применения, ведутся работы по разработке ассоциативных процессоров. Разработчики процессора для ATLAS FTK [37] планируют использовать процессор как для повседневных приложений (в камерах смартфонов и смарт-камерах), так и для научных (задачи физики высоких энергий, для медицинской визуализации, в системах обеспечения безопасности и др.). Поэтому актуальны как реализации моделей ассоциативных вычислений на доступных вычислительных архитектурах, так и разработка ассоциативных параллельных алгоритмов на ассоциативных моделях.

При реализации базовых операций модели STAR-машины на графических ускорителях удалось добиться выполнения свойств ассоциативных параллельных архитектур с незначительной потерей эффективности. Такие потери вызваны концептуальными различиями моделей в отношении синхронизации.

Построенная реализация базовых ассоциативных алгоритмов на GPU дает существенное ускорение по сравнению с библиотекой STL и, зачастую, выигрывает в производительности у библиотеки CUDA thrust. Это позволяет предполагать, что реализации ассоциативных версий алгоритмов на GPU будут выигрывать по производительности у параллельных реализаций алгоритмов.

Следует отметить, что для группы ассоциативных алгоритмов возможен двухуровневый параллелизм при достаточных вычислительных ресурсах: IV группа базовых процедур и алгоритм Уоршалла [32]. Для таких алгоритмов временна́я сложность понижается на два порядка по сравнению с последовательным выполнением.

Планируется использовать данную реализацию STAR-машины для вычисления как построенных ассоциативных параллельных алгоритмов [18, 23, 26–28, 38], так и новых ассоциативных алгоритмов при решении задач на графах.

список литературы

1. *Rudolph J.A.* A production implementation of an associative array processor: STARan // Proc. Fall Joint Computer Conf. Part I. New York: ACM Press, 1972. 229–241.
2. *Foster C.C.* Content addressable parallel processors. New York: Reinhold, 1976.
3. *Batcher K.E.* Bit-serial parallel processing systems // IEEE Transactions on Computers. **31**, N 5. 1982. 377–384.
4. *Uhr L.M.* Algorithm-structured computer arrays and networks: architectures and processes for images, precepts, models, information. Orlando: Academic, 1984.
5. *Higuchi T., Kitano H., Furuya T., Handa K., Kokubu A., Takahashi N.* IXM2: a parallel associative processor for knowledge processing // Proc. 9th Nat. Conf. on Artificial Intelligence. Vol. 1. Palo Alto: AAAI Press, 1991. 296–303.
6. *Smith D.E., Hall J.S., Miyake K.* Rutger's CAM2000 chip architecture. Technical Report LCSR-TR-196. New Brunswick: Rutgers Univ., 1993.
7. *Hsu C.-H., Smith D.E., Levy S.* Linear-C: a data-parallel extension to C. Technical Report LCSR-TR-273. New Brunswick: Rutgers Univ., 1996.
8. *Aad G., Abat E., Abdallah J., et al.* The ATLAS experiment at the CERN Large Hadron Collider // Journal of Instrumentation. **3**. 2008. doi 10.1088/1748-0221/3/08/S08003.
9. *Annovi A., Beretta M., Bossini E., Crescioli F., Dell'Orso M., Giannetti P., Piendibene M., Sacco I., Sartori L., Tripiccione R.* Associative memory design for the FastTrack processor (FTK) at ATLAS // Proc. 17th IEEE-NPSS Real Time Conference. IEEE Press, doi 10.1109/RTC.2010.5750451.
10. *Kitano H.* An implementation on the IXM2 associative memory // Speech-to-Speech Translation: A Massively Parallel Memory-Based Approach. Boston: Springer, 1994. 135–155.
11. *Kitano H., Higuchi T., Tomita M.* Massively parallel spoken language processing using a parallel associative processor IXM2 // Proc. First Int. Conf. on Spoken Language Processing. http://www.isca-speech.org/archive/icslp_1990.
12. *Oi K., Sumita E., Furuse O., Iida H., Higuchi T.* Real-time spoken language translation using associative processors // Proc. Fourth Conference on Applied Natural Language Processing. Stroudsburg: ACL Press, 1994. 101–106.
13. *Непомнящая А.Ш., Владыко М.А.* Сравнение моделей ассоциативного вычисления // Программирование. 1997. № 6. 41–50.
14. *Nepomniaschaya A.Sh.* Language STAR for associative and parallel computation with vertical data processing // Parallel Computing Technologies. Singapore: World Scientific, 1991. 258–265.
15. *Potter J.L.* Associative computing: a programming paradigm for massively parallel computers. Basel: Perseus Publ., 1991.
16. *Potter J., Baker J., Scott S., Bansal A., Leangsuksun C., Asthagiri C.* ASC: an associative-computing paradigm // Computer. 1994. **27**, N 11. 19–25.
17. *Nepomniaschaya A.S.* Solution of path problems using associative parallel processors // Proc. Int. Conf. on Parallel and Distributed Systems. New York: IEEE Press, 1997. 610–617.
18. *Nepomniaschaya A.S., Dvoskina M.A.* A simple implementation of Dijkstra's shortest path algorithm on associative parallel processors // Fundamenta Informaticae. 2000. **43**, N 1–4. 227–243.
19. *Nepomniaschaya A.S.* An associative version of the Bellman–Ford algorithm for finding the shortest paths in directed graphs // Lecture Notes in Computer Science. Vol. 2127 Berlin: Springer, 2001. 285–292.
20. *Nepomniaschaya A.S.* Efficient implementation of Edmonds' algorithm for finding optimum branchings on associative parallel processors // Proc. Eighth Int. Conf. on Parallel and Distributed Systems. New York: IEEE Press, 2001. doi 10.1109/ICPADS.2001.934794.
21. *Непомнящая А.Ш.* Сравнение алгоритмов Прима–Дейкстры и Краскала с помощью ассоциативного параллельного процессора // Кибернетика и системный анализ. 2000. № 2. 19–27.
22. *Nepomniaschaya A.S.* Representation of the Gabow algorithm for finding smallest spanning trees with a degree constraint on associative parallel processors // Lecture Notes in Computer Science. Vol. 1123. Berlin: Springer, 2001. 813–817.
23. *Nepomniaschaya A.S.* Associative parallel algorithms for dynamic edge update of minimum spanning trees // Lecture Notes in Computer Science. Vol. 2763. Berlin: Springer, 2003. 141–150.

24. *Nepomniaschaya A.* Associative parallel algorithm for dynamic reconstructing a minimum spanning tree after deletion of a vertex // Lecture Notes in Computer Science. Vol. 3606. Berlin: Springer, 2005. 159–173.
25. *Непомнящая А.Ш.* Ассоциативный параллельный алгоритм для динамической обработки минимального каркаса после добавления к графу новой вершины // Кибернетика и системный анализ. 2006. № 1. 19–31.
26. *Nepomniaschaya A.* Efficient implementation of the Italiano algorithms for updating the transitive closure on associative parallel processors // Fundamenta Informaticae. 2008. **89**, N 2–3. 313–329.
27. *Nepomniaschaya A.* Associative version of the Ramalingam decremental algorithm for dynamic updating the single-sink shortest paths subgraph // Lecture Notes in Computer Science. Vol. 5698. Berlin: Springer, 2009. 257–268.
28. *Непомнящая А.Ш.* Ассоциативная версия алгоритма Рамалингама для динамической обработки подграфа кратчайших путей после добавления к графу новой дуги // Кибернетика и системный анализ. 2012. № 3. 45–57.
29. *Walker R.A., Potter J.L., Wang Y., Wu M.* Implementing associative processing: rethinking earlier architectural decisions // Proc. 15th Int. Parallel and Distributed Processing Symposium. New York: IEEE Press, 2001. 2092–2100.
30. *Trahan J.L., Jin M., Chantamas W., Baker J.W.* Relating the power of the Multiple Associative Computing (MASC) model to that of reconfigurable bus-based models // Journal of Parallel and Distributed Computing. **70**, N 5. 2010. 458–466.
31. *Jin M.* Associative operations from MASC to GPU // Proc. 21st Int. Conf. on Parallel and Distributed Processing Techniques and Applications. Red Hook: CSREA Press, 2015. 388–393.
32. *Снытникова Т.В., Непомнящая А.Ш.* Решение задач на графах с помощью STAR-машины, реализуемой на графических ускорителях // Прикладная дискретная математика. 2016. **3**, № 33. 98–115.
33. *Nepomniaschaya A.S.* Basic associative parallel algorithms for vertical processing systems // Bulletin of the Novosibirsk Computing Center. 2009. № 9. 63–77.
34. *Nepomniaschaya A.S., Dvoskina M.A.* A simple implementation of Dijkstra's shortest path algorithm on associative parallel processors // Fundamenta Informaticae. 2000. **43**, N 1–4. 227–243.
35. *Страуструп Б.* Язык программирования C++. Специальное издание. М.: Бином, 2012.
36. Realeases-thrust/thrust-GitHub. https://github.com/thrust/thrust/releases.
37. *Masi S.* Periodic Report Summary 1 — FTK (Fast Tracker for Hadron Collider Experiments): Technical Report 324318 http://cordis.europa.eu/result/rcn/167746_en.html.
38. *Nepomniaschaya A.Sh., Snytnikova T.V.* Associative parallel algorithm for dynamic reconstruction of the shortest paths tree after insertion of a vertex // Bulletin of Novosibirsk Computing Center. 2006. N 24. 89–103.

Поступила в редакцию
21.11.2017

Implementation of an Associative-Computing Model on GPU: A Basic Procedure Library of the STAR Language

T. V. Snytnikova [1]

[1] *Institute of Computational Mathematics and Mathematical Geophysics, Siberian Branch of Russian Academy of Sciences; prospekt Lavrentyeva 6, Novosibirsk, 630090, Russia; Junior Scientist, e-mail: snytav@ssd.sscc.ru*

Received November 21, 2017

Abstract: The associative (content addressable) parallel processors of the SIMD type with vertical data processing are oriented on solving problems of non-numeric data processing. The simulation of such systems is described using an abstract SIMD-type model of a STAR machine. On the basis of this model, a number of efficient algorithms are developed to solve many graph problems. Since the associative architectures are not widely available, however, these algorithms cannot be used in practice. With advances in the production of GPU, the possibilities to implement the associative parallel models without significant loss of efficiency are increased. As the first stage in the implementation of the STAR-machine on GPU in the form of a CUDA library, specific data types and simple operations of the STAR language were developed. In this paper, we consider an efficient GPU implementation of the standard associative procedure library. The runtime of this implementation is compared with the runtime of similar procedures in the standard libraries (STL on CPU and CUDA thrust on GPU). We plan to use our library implementation to solve graph problems.

Keywords: vertical data processing, model of associative parallel processor, GPU, high-performance computing.

References

1. J. A. Rudolph, "A Production Implementation of an Associative Array Processor: STARAN," in *Proc. Fall Joint Computer Conf., Part I, Anaheim, USA, December 5–7, 1972* (ACM Press, New York, 1972), pp. 229–241.

2. C. C. Foster, *Content Addressable Parallel Processors* (Reinhold, New York, 1976).

3. K. E. Batcher, "Bit-Serial Parallel Processing Systems," IEEE Trans. Comput. **31** (5), 377–384 (1982).

4. L. M. Uhr, *Algorithm-Structured Computer Arrays and Networks: Architectures and Processes for Images, Precepts, Models, Information* (Academic, Orlando, 1984).

5. T. Higuchi, H. Kitano, T. Furuya, et al., "IXM2: A Parallel Associative Processor for Knowledge Processing," in *Proc. 9th Nat. Conf. on Artificial Intelligence, Anaheim, USA, July 14–19, 1991* (AAAI Press, Palo Alto, 1991), Vol. 1, pp. 296–303.

6. D. E. Smith, J. S. Hall, and K. Miyake, *Rutger's CAM2000 Chip Architecture*, Technical Report LCSR-TR-196 (Rutgers University, New Brunswick, 1993).

7. C.-H. Hsu, D. Smith, and S. Levy, *Linear-C: A Data-Parallel Extension to C*, Technical Report LCSR-TR-273 (Rutgers University, New Brunswick, 1996).

8. G. Aad, E. Abat, J. Abdallah, et al., "The ATLAS Experiment at the CERN Large Hadron Collider," J. Instr. **3** (2008). doi 10.1088/1748-0221/3/08/S08003

9. A. Annovi, M. Beretta, E. Bossini, et al., "Associative Memory Design for the FastTrack Processor (FTK) at ATLAS," in *Proc. 17th IEEE-NPSS Real Time Conference, Lisbon, Portugal, May 24–28, 2010*, IEEE Press, doi 10.1109/RTC.2010.5750451

10. H. Kitano, "An Implementation on the IXM2 Associative Memory," in *Speech-to-Speech Translation: A Massively Parallel Memory-Based Approach* (Springer, Boston, 1994), pp. 135–155.

11. H. Kitano, T. Higuchi, and M. Tomita, "Massively Parallel Spoken Language Processing uusing a Parallel Associative Processor IXM2," in *Proc. First Int. Conf. on Spoken Language Processing, Kobe, Japan, November 18–22, 1990*, http://www.isca-speech.org/archive/icslp_1990. Cited February 20, 2018.

12. K. Oi, E. Sumita, O. Furuse, et al., "Real-Time Spoken Language Translation Using Associative Processors," in *Proc. Fourth Conference on Applied Natural Language Processing, Stuttgart, Germany, October 13–15, 1994* (ACL Press, Stroudsburg, 1994), pp. 101–106.

13. A. Sh. Nepomniaschaya and M. A. Vladyko, "A Comparison of Associative Computation Models," Programmirovanie, No. 6, 41–50 (1997) [Program. Comput. Software **23** (6), 319–324 (1997)].

14. A. Sh. Nepomniaschaya, "Language STAR for Associative and Parallel Computation with Vertical Data Processing," in *Parallel Computing Technologies* (World Scientific, Singapore, 1991), pp. 258–265.

15. J. L. Potter, *Associative Computing: A Programming Paradigm for Massively Parallel Computers* (Perseus Publ., Basel, 1991).

16. J. Potter, J. Baker, S. Scott, et al., "ASC: An Associative–Computing Paradigm," Computer **27** (11), 19–25 (1994).

17. A. S. Nepomniaschaya, "Solution of Path Problems Using Associative Parallel Processors," in *Proc. Int. Conf. on Parallel and Distributed Systems, Seoul, South Korea, December 10–13, 1997* (IEEE Press, New York, 1997), pp. 610–617.

18. A. S. Nepomniaschaya and M. A. Dvoskina, "A Simple Implementation of Dijkstra's Shortest Path Algorithm on Associative Parallel Processors," Fundam. Inform. **43** (1–4), 227–243 (2000).

19. A. S. Nepomniaschaya, "An Associative Version of the Bellman–Ford Algorithm for Finding the Shortest Paths in Directed Graphs," in *Lecture Notes in Computer Science* (Springer, Berlin, 2001), Vol. 2127, pp. 285–292.

20. A. S. Nepomniaschaya, "Efficient Implementation of Edmonds' Algorithm for Finding Optimum Branchings on Associative Parallel Processors," in *Proc. 8th Int. Conf. on Parallel and Distributed Systems, Kyongju City, South Korea, June 29–29, 2001* (IEEE Press, New York, 2001), doi 10.1109/ICPADS.2001.934794

21. A. S. Nepomniaschaya, "Comparison of the Prim-Dijkstra and Kraskal Algorithms on an Associative Parallel Processor," Kibernet. Sist. Anal., No. 2, 19–27 (2000) [Cybernet. Systems Anal. **36** (2), 162–169 (2000)].

22. A. S. Nepomniaschaya, "Representation of the Gabow Algorithm for Finding Smallest Spanning Trees with a Degree Constraint on Associative Parallel Processors," in *Lecture Notes in Computer Science* (Springer, Berlin, 2001), Vol. 1123, pp. 813–817.

23. A. S. Nepomniaschaya, "Associative Parallel Algorithms for Dynamic Edge Update of Minimum Spanning Trees," in *Lecture Notes in Computer Science* (Springer, Berlin, 2003), Vol. 2763, pp. 141–150.

24. A. Nepomniaschaya, "Associative Parallel Algorithm for Dynamic Reconstruction of a Minimum Spanning Tree After Deletion of a Vertex," in *Lecture Notes in Computer Science* (Springer, Berlin, 2005), Vol. 3606, pp. 159–173.

25. A. Sh. Nepomniaschaya, "Associative Parallel Algorithm for Dynamic Update of a Minimum Spanning Tree after Addition of a New Node to a Graph," Kibernet. Sist. Anal., No. 2, 19–31 (2006) [Cybernet. Systems Anal. **36** (2), 162–169 (2006)].

26. A. Nepomniaschaya, "Efficient Implementation of the Italiano Algorithms for Updating the Transitive Closure on Associative Parallel Processors," Fundam. Inform. **89** (2–3), 313–329 (2008).

27. A. Nepomniaschaya, "Associative Version of the Ramalingam Decremental Algorithm for Dynamic Updating the Single-Sink Shortest-Paths Subgraph," in *Lecture Notes in Computer Science* (Springer, Berlin, 2009), Vol. 5698, pp. 257–268.

28. A. S. Nepomniaschaya, "Associative Version of the Ramalingam Algorithm for Dynamically Updating the Shortest-Path Subgraph after Inserting a New Edge into a Graph," Kibernet. Sist. Anal., No. 3, 45–57 (2012) [Cybernet. Systems Anal. **48** (3), 358–368 (2012)].

29. R. A. Walker, J. L. Potter, Y. Wang, and M. Wu, "Implementing Associative Processing: Rethinking Earlier Architectural Decisions," in *Proc. 15th Int. Parallel and Distributed Processing Symposium, San Francisco, USA, April 23–27, 2000* (IEEE Press, New York, 2001), pp. 2092–2100.

30. J. L. Trahan, M. Jin, W. Chantamas, and J. W. Baker, "Relating the Power of the Multiple Associative Computing (MASC) Model to That of Reconfigurable Bus-Based Models," J. Parallel Distrib. Comput. **70** (5), 458–466 (2010).

31. M. Jin, "Associative Operations from MASC to GPU," in *Proc. 21st Int. Conf. on Parallel and Distributed Processing Techniques and Applications, Las Vegas, USA, July 27–30, 2015* (CSREA Press, Red Hook, 2015), pp. 388–393.

32. T. V. Snytnikova and A. Sh. Nepomniaschaya, "Solution of Graph Problems by Means of the STAR-Machine Being Implemented on GPUs," Prikl. Diskr. Mat. **3** (33), 98–115 (2016).

33. A. S. Nepomniaschaya, "Basic Associative Parallel Algorithms for Vertical Processing Systems," Bull. Novosibirsk Comput. Center, No. 9, 63–77 (2009).

34. A. S. Nepomniaschaya and M. A. Dvoskina, "A Simple Implementation of Dijkstra's Shortest Path Algorithm on Associative Parallel Processors," Fundam. Inform. **43** (1–4), 227–243 (2000).

35. B. Stroustrup, *The C++ Programming Language, Special Edition* (Addison-Wesley, Reading, 2000; Binom, Moscow, 2012).

36. Realeases-thrust/thrust-GitHub. https://github.com/thrust/thrust/releases. Cited February 20, 2018.

37. S. Masi, *Periodic Report Summary 1 — FTK (Fast Tracker for Hadron Collider Experiments)*, Technical Report 324318. http://cordis.europa.eu/result/rcn/167746_en.html.

38. A. Sh. Nepomniaschaya and T. V. Snytnikova, "Associative Parallel Algorithm for Dynamic Reconstruction of the Shortest Paths Tree after Insertion of a Vertex," Bull. Novosibirsk Comput. Center, No. 24, 89–103 (2006).