

О РЕАЛИЗАЦИИ НА GPU БАЗОВЫХ АССОЦИАТИВНЫХ ПРОЦЕДУР ЯЗЫКА STAR

Т. В. Снытникова, А. Ш. Непомнящая

Институт Вычислительной Математики и Математической Геофизики СО РАН, 630090, Новосибирск

УДК 519.68->519.17

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). В работе [1] построена библиотека стандартных процедур языка STAR. В работе [2] приведена эффективная реализация на GPU простейших операций языка STAR. В данной работе приводится эффективная реализация на GPU библиотеки стандартных процедур языка STAR. Проведено сравнение времени работы данной реализации с временем работы подобных алгоритмов из стандартных библиотек (STL на CPU и CUDA thrust на GPU). Планируется использовать представленную выше на GPU библиотеку стандартных процедур языка STAR для решения задач на графах.

Ключевые слова: вертикальная обработка данных, графический ускоритель, эффективность, высокопроизводительные вычисления.

Введение

Ассоциативные параллельные процессоры типа SIMD 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.

В работе [3] построена абстрактная модель типа SIMD (Star-машина), которая описывает работу ассоциативной архитектуры на микроуровне. Ассоциативные параллельные алгоритмы записываются в виде процедур на языке высокого уровня STAR. Для удобного проектирования и анализа ассоциативных алгоритмов в работе [1] были выделены базовые ассоциативные параллельные алгоритмы. Соответствующие процедуры составляют библиотеку стандартных процедур языка STAR.

В работе [2] приведена эффективная реализация на GPU типов данных и простейших операций языка STAR. В данной работе приводится эффективная реализация на GPU библиотеки стандартных процедур языка STAR.

1 Свойства Star-машины и особенности ее моделирования на GPU

В работе [4] Поттер выделил основные свойства ассоциативных параллельных моделей и архитектур, которым, в частности, удовлетворяет Star-машина (рис. 1). Ключевыми для моделирования на других типах архитектур являются следующие свойства ассоциативных параллельных моделей.

- Мелкозернистость: каждое слово данных обрабатывается отдельным ПЭ.
- 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.
- Быстрый обмен результатом вычисления между ПЭ: ПУУ может дать команду выбранному ПЭ передать данные по шине, все другие ПЭ получают значение с шины.

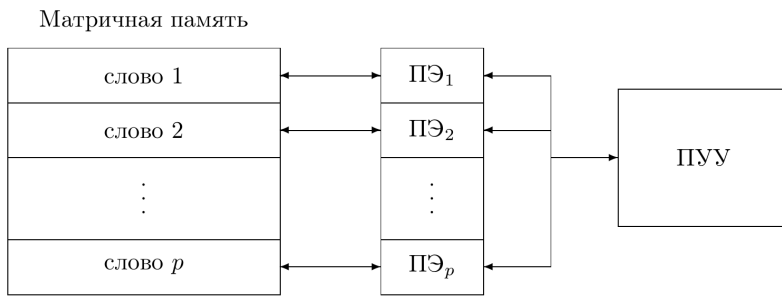


Рис. 1: Модель Star-машины

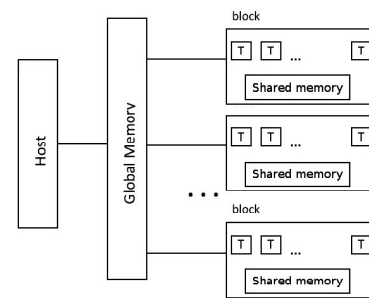


Рис. 2: Модель GPU

На графических ускорителях можно смоделировать эти свойства следующим образом:

- Каждый ПЭ моделируется отдельным потоком вычислений в блоке (Т на рис. 2), слово матричной памяти моделируется элементом массива данных в глобальной памяти (индекс элемента в массиве выражается через индекс потока/блока).
- 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 – 20 мс.
- Данные передаются в глобальную память. При этом возможно 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 [3]. В работе [2] 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 работе [2] приведена реализация этой базовой операции с временной сложностью $O(\lceil \log_{64}(N) \rceil)$, 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. Данные операции выполняются процедурами библиотеки стандартных процедур языка STAR [1]. Особенности их реализации приводятся ниже.

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

2 Описание библиотеки базовых ассоциативных процедур языка Star

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

Приведем процедуры, которые относятся к первой группе. 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 / максимальный элемент: $\text{MIN}(T, X, Z)$ и $\text{MAX}(T, X, Z)$, соответственно.

Процедуры, которые по заданной матрице T , по образцу v и по управляющему слайсу X находят позиции строк, удовлетворяющих условию поиска: $\text{MATCH}(T, X, v, Z)$ для $\text{ROW}(j, T)=v$, $\text{LESS}(T, X, v, Y)$ 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 $GREAT$: в слайсе Y отмечены позиции строк, которые больше образца, а в слайсе Z — меньше образца.

Процедуры, которые по двум заданным 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 $ROW(j, T) > ROW(j, F)$.

Приведем процедуры для числовой обработки. Процедуры построчного сложения матриц $ADDV(T, F, X, R)$ и добавления к строкам матрицы T двоичного слова v $ADDC(T, X, v, R)$. 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 $SUBTV(T, R, X, R)$, $SUBTC(T, X, v, R)$, $SUBTC1(T, X, v, R)$.

Также в библиотеку входят различные 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.

3 Особенности реализации библиотеки стандартных процедур и сравнение с STL, CUDA thrust

Базовые алгоритмы можно разделить на следующие группы по способу работы с таблицами.

- I В алгоритме используются базовые операции, критичные к синхронизации: MIN , MAX .
Алгоритмы этой группы при реализации разбиваются на несколько `__global__`-процедур.
- II В алгоритме не используются базовые операции, критичные к синхронизации: $MATCH$, $LESS$, $GREAT$, GEL , HIT , $SETMIN$, $SETMAX$.
Алгоритмы этой группы могут быть представлены в виде `__device__`-процедур. Это позволяет встраивать их в реализации других алгоритмов без накладных расходов на производительность.
- III Арифметические алгоритмы: $ADDV$, $ADDC$, $ADDC1$, $SUBTV$, $SUBTC$, $SUBTC1$.
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.
- IV В алгоритме не используются базовые операции, критичные к синхронизации. Столбцы могут обрабатываться независимо друг от друга: $CLEAR$, $WCOPY$, $WMERGE$, $TCOPY$, $TCOPY1$, $TCOPY2$, $TMERGE$.
Для данной 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.

4 Сравнение производительности библиотеки базовых процедур языка STAR с процедурами библиотек STL и CUDA thrust

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

Стоит отметить следующее:

	vector<int>	vector<int> бинарное представление	Реализация Star-машины
1	1804289	0b000000000011011100010000000001	0...0111011111000110000000000001
2	846930	0b0000000000011001110110001010010	0...001110011110111000101010010
3	1681692	0b0000000000110011010100100011100	0...0111001110101010010000111100
4	1714636	0b0000000000110100010100111001100	0...011101000101010011110011100
5	1957747	0b000000000011101110111101110011	0...011110111101111110111100111
...
60	1967513	0b0000000000111100000010110011001	0...01111100000001011100111001
61	1365180	0b0000000000101001101010010111100	0...01010011101010100101111100
62	1540383	0b000000000010111100000100011111	0...0101111100000010000111111
63	304089	0b00000000001001010001111011001	0...00010010101000111110111001
64	1303455	0b000000000010011111000111001111	0...0100111111100011110011111

Из 32 разрядов 11 разрядов пустые

Рис. 3: Представление данных

- Ассоциативные алгоритмы используют другое представление данных.
На рисунке 3 показаны представление структуры `vector<int>` из 64-х элементов для библиотек STL и CUDA thrust и представление этих же данных для реализации Star-машины. В структуре `vector<int>` каждое из целых чисел массива представляет собой 32-х 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-х разрядное целое число. За счет такого вертикального представления данных пустые разряды можно не включать в таблицу для экономии памяти и уменьшения времени работы.
- Сравнимые алгоритмы часто отличаются входными и выходными параметрами.
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 и CUDA thrust время обработки всех строк может существенно отличаться от времени обработки какого-то их подмножества (предикатная форма).
В ассоциативных алгоритмах поиска в качестве выходных данных также используется слайс, в котором отмечены все вхождения искомого элемента. Алгоритмы библиотек STL и CUDA thrust выдают только одно вхождение искомого элемента.

Поэтому рассматриваемое сравнение реализаций носит условный характер, тем не менее дает некоторое представление о производительности.

4.1 I группа алгоритмов: MIN

Из первой группы алгоритмов рассмотрим процедуру MIN. Теоретическая сложность реализации ассоциативного алгоритма $O(h \cdot \lceil \log_{64}(N) \rceil)$, где h — количество столбцов обрабатываемой таблицы, N — количество строк. На рис. 4 видна слабая зависимость времени счета ($\mu\text{с}$) от количества строк (N).

Приведем сравнение производительности следующих реализаций:

- **min** — реализация ассоциативного алгоритма MIN;
- **STL** — последовательная реализация `std::min_element()` из библиотеки *STL* [5];
- **thrust** — реализация на CUDA `thrust::min_element()` библиотеки thrust [6];
- **thrust*** — реализация на CUDA с использованием библиотеки thrust выдает вектор, в котором помечены позиции всех минимальных элементов.

На рисунке 5 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 $N \geq 5000$. Реализации *min* и *thrust_min_element** дают сравнимый результат на диапазоне $N \leq 50000$. Реализация *min* дает заметный выигрыш по производительности относительно *thrust::min_element* при $N \geq 100000$.

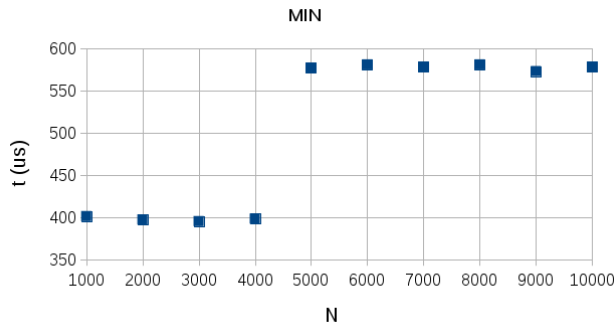


Рис. 4: Время работы с таблицами разной длины.

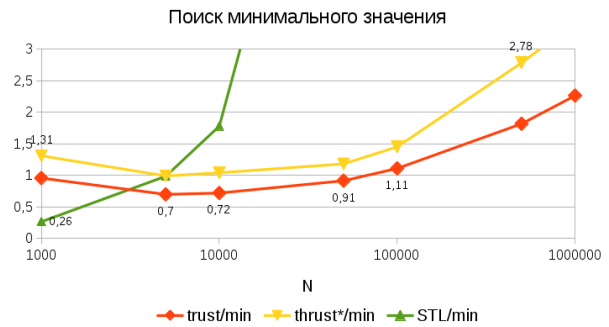


Рис. 5: Отношение времени работы реализации.

4.2 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.

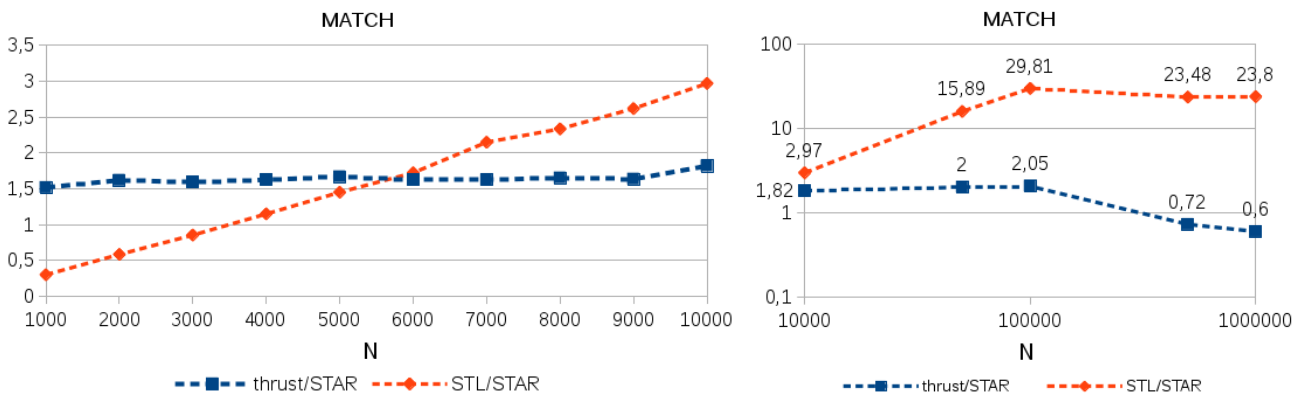


Рис. 6: Сравнение времени работы алгоритмов поиска.

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 рисунка 6 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 ($N \leq 100\,000$ на GEFORCE 920M), реализация ассоциативного алгоритма MATCH работает в 1,5 – 2 раза быстрее, чем `thrust::find()`.

4.3 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 — $O(h + \lceil \log_{64}(N) \rceil)$.

Проводится сравнение производительности следующих реализаций:

- **subtv** — 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.

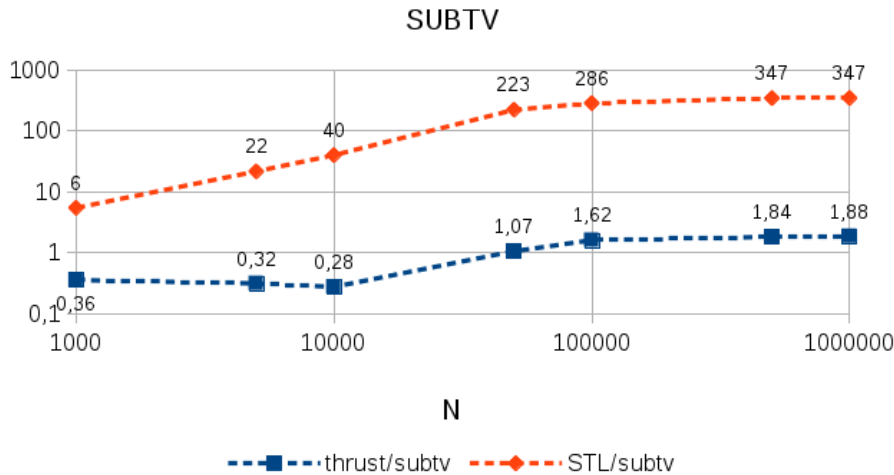


Рис. 7: Отношение времени работы реализации.

Из рисунка 7 видно, что реализация ассоциативного алгоритма работает медленнее реализации из библиотеки thrust на данных при $N \leq 10\,000$, работает сравнимое время при $N \approx 50\,000$ и работает быстрее на данных большего размера. Последовательная реализация сильно уступает в производительности параллельным.

4.4 IV группа алгоритмов: TMERGE

Для данной 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 [5]);
- **thrust**: `thrust::copy()` проводит копирование всех элементов;
- **thrust_if** `thrust::copy_if()` проводит копирование только тех элементов, которые удовлетворяют некоторому предикату.

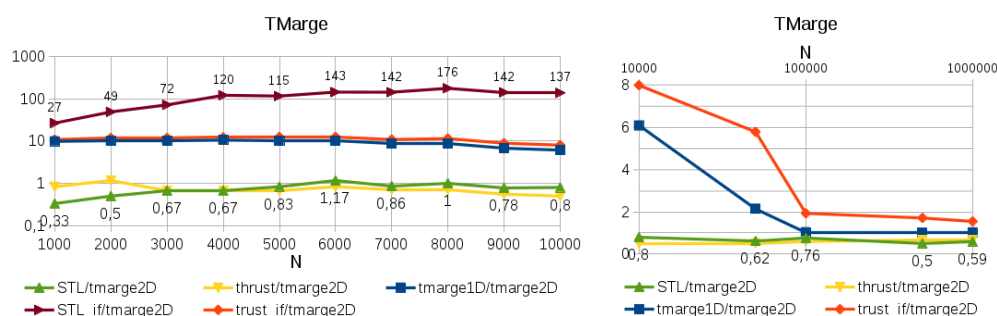


Рис. 8: Отношение времени работы реализации.

Из рисунка 8 видно следующее. Реализация tmerge2D сравнима по времени с беспредикатными версиями стандартных библиотек STL и thrust (4 – 6 μ s на векторах до 10 000 элементов) и существенно выигрывает в производительности у предикатных. Реализация с поочередным обрабатыванием столбцов tmerge1D сравнима в работе с thrust_if на векторах до 10 000 элементов (tmerge1D: 60 μ s, thrust_if: 70 – 80 μ s) и совпадает с tmerge2D на векторах более 100 000 элементов, когда не достаточно ресурсов для физически одновременного исполнения блоков.

5 Выводы

Star-машина использует существенно другое представление данных, поэтому механический перенос ассоциативных алгоритмов на ЭВМ стандартной архитектуры невозможен.

Построенная реализация базовых ассоциативных алгоритмов на GPU дает существенное ускорение по сравнению с библиотекой STL и, зачастую, выигрывает в производительности у библиотеки CUDA thrust.

В представленной реализации Star-машины есть потенциал для оптимизации, связанный с вычислением конфигурации блоков с учетом свободных ресурсов.

Для группы ассоциативных алгоритмов возможен двухуровневый параллелизм при достаточных вычислительных ресурсах: IV группа базовых процедур, алгоритм Уоршалла [2].

Список литературы

- [1] Nepomniaschaya A. S. Basic associative parallel algorithms for vertical processing systems // Bulletin of the Novosibirsk Computing Center. 2009. Vol. 9. P. 63–77.
- [2] Снытникова Т. В., Непомнящая А. Ш. Решение задач на графах с помощью Star-машины, реализуемой на графических ускорителях // Прикладная дискретная математика, 2016, №3(33), С. 98–115.
- [3] Nepomniaschaya A. S., Dvoskina M. A. A simple implementation of Dijkstra's shortest path algorithm on associative parallel processors // Fundamenta Informaticae. IOS Press, 2000. Vol. 43. P. 227–243.
- [4] Potter J. L. Associative Computing: A Programming Paradigm for Massively Parallel Computers. Perseus Publishing, 1991.
- [5] Страуструп Б. Язык программирования C++. Специальное издание. Бином, 2012.
- [6] Releases-thrust/thrust-GitHub. 2017. <https://github.com/thrust/thrust/releases>.

Татьяна Валентиновна Снытникова — мл. науч.сотр. Института
вычислительной математики и математической геофизики СО РАН;
e-mail: snytnikovat@ssd.sccc.ru;

Анна Шмилевна Непомнящая — к.ф.-м.н., ст. науч.сотр. Института вычислительной
математики и математической геофизики СО РАН;
e-mail: anep@ssd.sccc.ru.

Дата поступления — 25 мая 2017 г.