Эффективная реализация модели ассоциативных вычислений на графических ускорителях.

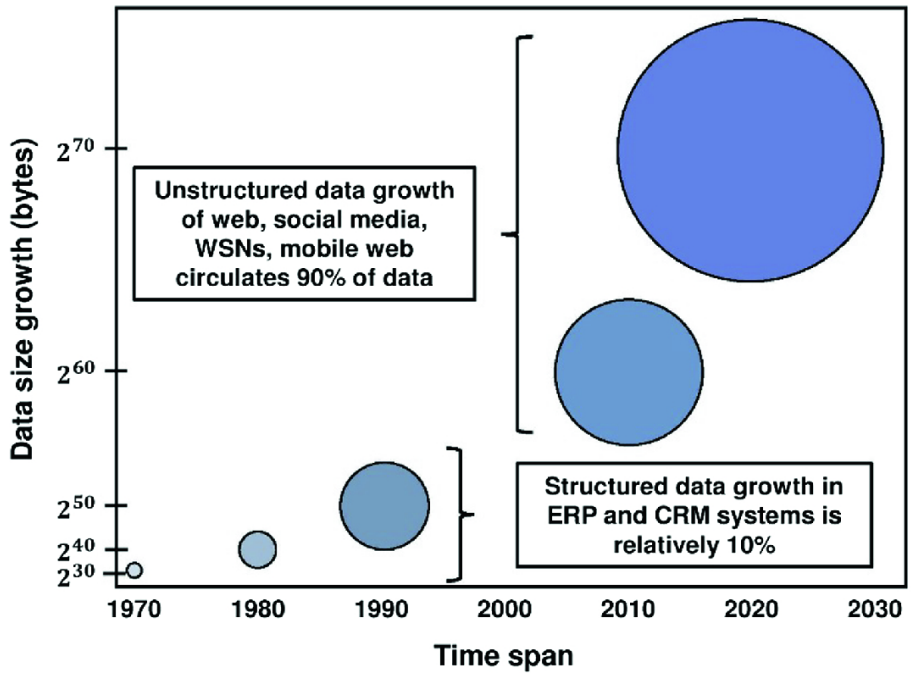
Т. В. Снытникова

05.13.11 - математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

Институт Вычислительной Математики
и Математической Геофизики СО РАН

Актуальность работы

- информация удваивается каждые 18 месяцев;
- большая часть не является структурированной (организованной в базы данных и знаний);



Алгоритмы поиска по несортированным данным и алгоритмы сортировки - узкое место для вычислительных машин фон-неймановского типа.

Ассоциативные параллельные модели и архитектуры:
время выполнения базовых операций поиска не зависит от числа строк.

Цель диссертационной работы

состоит в **создании программных средств для эффективной реализации ассоциативных параллельных алгоритмов на графических ускорителях**.

Для достижения поставленной цели были решены следующие задачи:

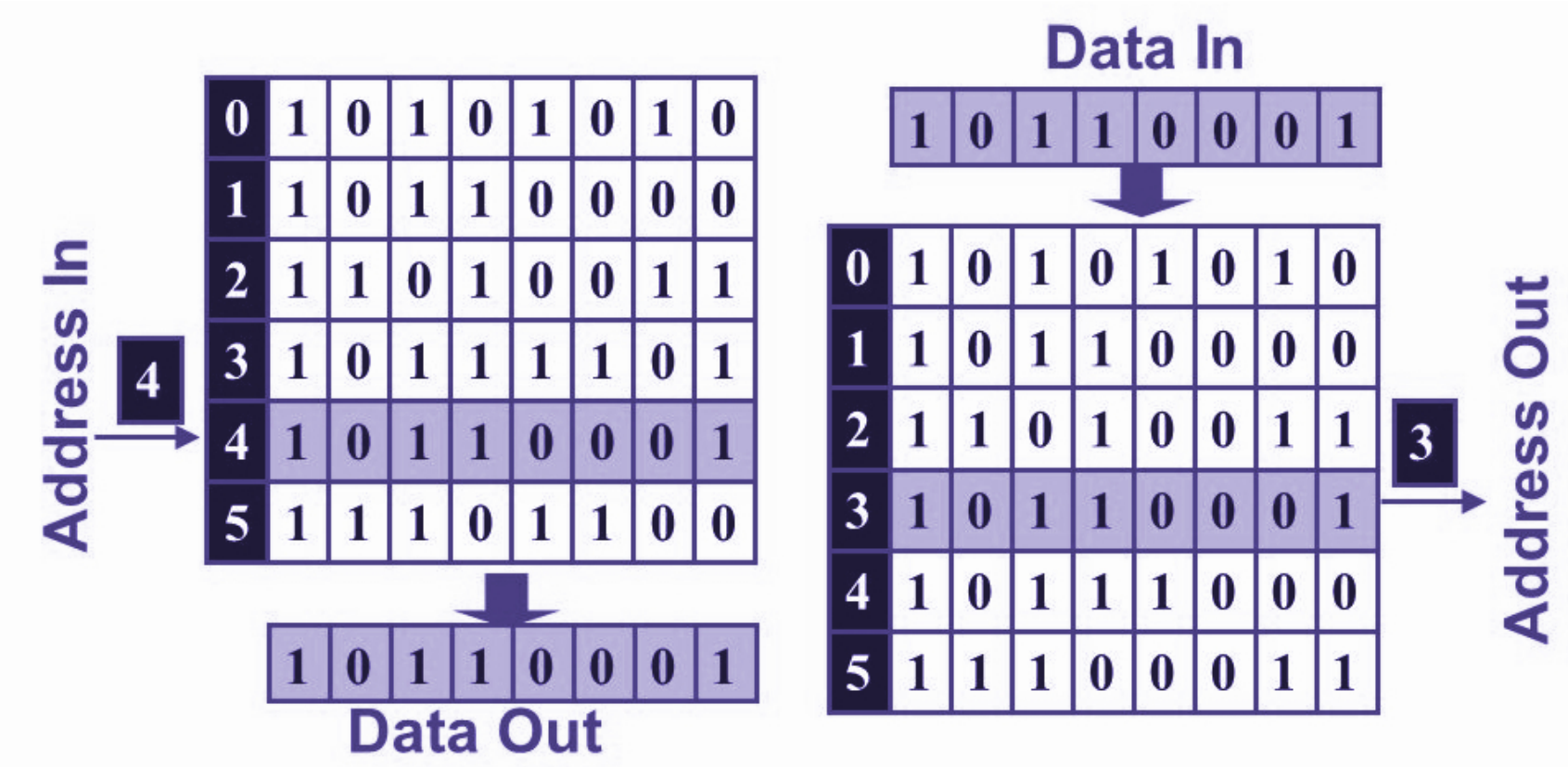
- построена реализация базовых операций языка Star на GPU;
- выделены операции языка Star, критичные к синхронизации;
- разработан модуль ввода/вывода данных;
- реализована библиотека стандартных процедур языка Star;
- доказана эффективность реализации Star-машины как оценкой теоретической сложности процедур реализации, так и практическим сравнением времени работы с временем работы аналогов;
- выработаны методы оптимизации ассоциативных алгоритмов для выполнения на графических ускорителях, учитывающие различия Star-машины и GPU.

На защиту выносятся следующие положения и результаты:

- выявлены операции языка Star, критичные к синхронизации (2);
- разработана реализация модели ассоциативных вычислений (Star-машина) на графических ускорителях (8);
- приведена классификация библиотеки стандартных процедур языка Star по способу обработки данных (2);
- выработаны методы оптимизации ассоциативных алгоритмов для выполнения на графических ускорителях, учитывающие различия Star-машины и GPU (8);
- эффективность реализации Star-машины доказана как оценкой теоретической сложности процедур реализации, так и практическим сравнением времени работы с временем работы аналогов. (10)

Глава I. Развитие ассоциативных параллельных моделей и архитектур: доступ к памяти.

- RAM (Random Access Memory, память с произвольным доступом);
- CAM (Content Addressable Memory, память с доступом по содержимому).



Использование: скоростные кэши, сетевые маршрутизаторы

Глава I. Развитие ассоциативных параллельных моделей и архитектур: принципы.

1. Свойства последовательного устройства управления (ПУУ): ПУУ передает инструкцию всем ячейкам за единицу времени; **активные ячейки выполняют команду, неактивные принимают команду но не выполняют ее;** ПУУ может активировать ячейки.
2. Константное время выполнения глобальных операций: побитовые логические операции, **доступ к столбцам и строкам матричной памяти как на чтение, так и на запись, выбор активной ячейки.**
3. Ассоциативные свойства: (следующие операции выполняются за единицу времени)
   - **передача данных выбранной ячейкой всем остальным;**
   - ПУУ выбирает старшую ячейку из множества активных ячеек.
   - **Базовые операции поиска ($=$, $<$, $>$, min, max) и арифметические операции выполняются за время, пропорциональное числу битовых столбцов в таблице, а не числу ее строк.**

Глава I. Развитие ассоциативных параллельных моделей и архитектур: архитектуры.

- **Staran (1972)** Первая коммерчески успешная версия ассоциативного процессора.
- **Aspro (1982)** Системы управления воздушными сообщениями США. По данным 1983 года использовалась в радарах палубных самолетов-разведчиков E-2 Hawkeye AWACS ВМС США.
- IXM2 (1990-1994) системы машинного перевода ASTRAL и EBMT, система TDMT для перевода устной речи в режиме реального времени;
- Rutgers CAM2000 (1993-1996) при поддержке NASA;
- **ATLAS FastTracker (2008-2016 и далее)** обслуживание детектора ATLAS LHC. Области дальнейшего применения: медицинская визуализация (томография и т. д.); системы видео-наблюдения, смарт-камеры; задачи высокоскоростной фильтрации данных; изучение зрения и других функций мозга.

Глава I. Развитие ассоциативных параллельных моделей и архитектур: модели.

model ASC и MASC (на базе процессора ASPRO);

model Star-машина.

Сходство моделей:

- архитектура ASC и Star-машины;
- ассоциативные свойства.

Отличия:

- в моделях ASC и MASC поиск ведется для байта или слова , для всех операций критична синхронизация;
- в Star-машине побитовая обработка, часть операций не критична к синхронизации (выполняются PE над своими данными).

Глава I. Развитие ассоциативных параллельных моделей и архитектур. Выводы.

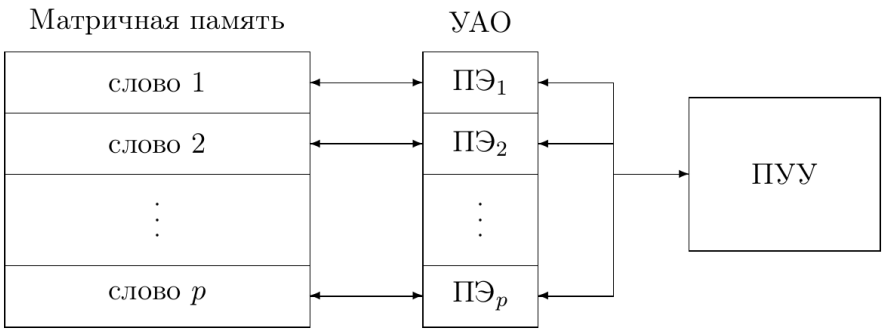
- Все представленные архитектуры были построены для решения конкретных задач, которые не могли быть эффективно решены на системах архитектуры PRAM (Parallel Random Access Machines).
- Разрабатываются новые устройства ассоциативной памяти.
- Ведется работа над тем, чтобы сделать АП пригодными для портативных устройств.

- Ведется разработка ассоциативных алгоритмов.

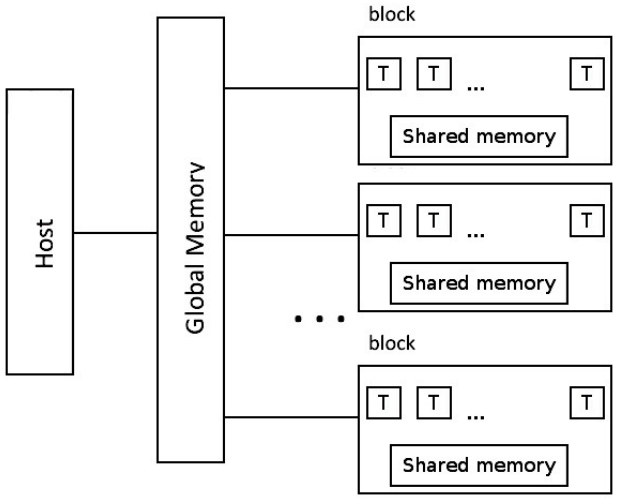
Отсюда следует актуальность эффективной программной реализации модели ассоциативных процессоров (STAR-машины) на доступной архитектуре.

Глава II. Раздел 1. Реализация STAR-машины на GPU: обоснование выбора архитектуры для реализации.

a) Модель Star-машины



b) Модель GPU



- класс SIMD (Single Instruction Multiple Data), массовый параллелизм $p \sim 128, 256, 1024$
- синхронизация на каждом шаге(инструкции поступают по одной, обмен результатами на каждом шаге).

Особенности GPU:

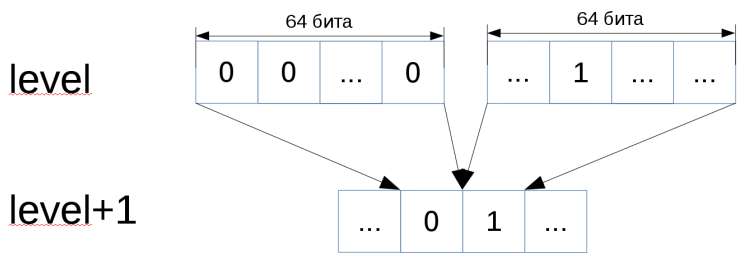
+ Более 6000 нитей вычислений, общая память).
+ 1D-3D нумерация блоков.
- __global__ и __device__ процедуры;
  Star-машина: критичные, не критичные к синхронизации.

Глава II. Раздел1. Реализация STAR-машины на GPU: сложности реализации и предложенные решения.

1. **Синхронизация операций всех ячеек, как активных, так и неактивных**. Выделение операций, критичных к синхранизации (использование __global__ -процедур для этого типа)
2. Константное время выполнения глобальных операций: побитовые логические операции, **доступ к столбцам и строкам матричной памяти как на чтение, так и на запись**.
3. Ассоциативные свойства: (следующие операции выполняются за единицу времени)
   - **передача данных выбранной ячейкой всем остальным;** использование глобальной памяти.
   - **выбор старшей ячейки из множества активных ячеек.** Сложность реализации операции $O(log_{64}(N))$.
   - Базовые операции поиска (=, <, >, min, max) и арифметические операции выполняются за время, пропорциональное числу битовых столбцов в таблице, а не числу ее строк. Является критерием эффективности реализации.

Глава II. Раздел 2. Реализация Star-машины на GPU: выбор старшей активной ячейки.

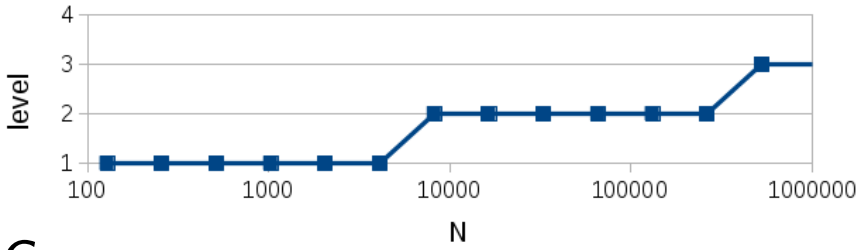
1 этап



2 этап

```
first_backward(LongPointer *d_v,int *d_first_non_zero,  int level)
{...   f[level+1] = 1;
       while(level >= 0)
       {
// вычисляется индекс ненулевого элемента для level
                int index = f[level+1]-1;
                u = dvl[index];
// вычисляет индекс ненулевого бита на текущем уровне
                f[level] = __ffsll(u) + index*SIZE_OF_LONG_INT;
                level--;
        }
// результат
     *d_first_non_zero = f[0];
}
```

Битовый столбец редуцируется в одну переменную типа Unsigned long long int за несколько итераций (level итераций для 64 level+1)



Сложность реализации операции $O(log_{64}(N))$, но для графа с числом вершин $N \leq 64^2(4096)$ требуется одна свертка. Для $N \leq 10^6$ требуется не более 3-х сверток.

Глава II. Реализация Star-машины на GPU: профилирование базовых операций

Операция	Время выполнения (μc)			
	100	1 000	3 000	5 000
ROW (чтение)	18,6 (18,5 – 19,7)	19, 5	19,6	19,8
ROW (запись)	4,1 (2,9 – 5,6)	3,9	4,3	4,1
or	1,9 (1,9 – 3,0)	2,2	2,0	2,1
FND	2,5 (2,3 – 2,8)	2,8	3,5	3,8
	3,2 (3,1 – 4,0)	3,2	3,3	3,4

Время выполнения базовых операций практически не зависит от размера данных, что соответствует теоретическим оценкам: $O(\log_{64}(n))$ для реализации операции *FND*, *SOME* и *STEP*, и константное время для реализации остальных операций.

Глава II. Реализация библиотеки стандартных ассоциативных алгоритмов

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

- есть базовые операции, критичные к синхронизации: MIN и MAX;
- нет базовых операций, критичных к синхронизации, управляющий слайс изменяется в процессе вычисления: поиск строк таблицы, совпадающих с образцом MATCH, сравнение строк с образцом GREAT, LESS, GEL, сравнение двух таблиц построчно SETMIN, SETMAX, HIT;
- арифметические алгоритмы: прибавление слова к строкам таблицы ADDC, построчное сложение таблиц ADDV, вычитание слова из строк таблицы SUBTC, построчное вычитание одной таблицы из другой SUBTV;
- нет базовых операций, критичных к синхронизации, и управляющий слайс не изменяется: CLEAR, TMERGE, WMERGE, WCOPY, TCOPY, TCOPY1, TCOPY2.

Глава II. Временные затраты стандартных ассоциативных алгоритмов

Оценка сложности базовых ассоциативных алгоритмов $O(h)$

Группа алгоритмов	Оценка сложности	Размер данных (одновременно)	Время μc	
			GF920m	k40
I	$O(h \cdot \lceil \log_{64}(N) \rceil)$	до 4 096 4 097 – 100 000	400 570 – 590	713 865 – 890
II	$O(h)$	100 000	58 – 61	71 – 73
III Арифм. алгоритмы	$O(h + \lceil \log_{64}(N) \rceil)$	50 000	36 – 41	52 – 58
IV	$O(1)$	10 000	6-10	8-9

Расчеты проводились на карте NVIDIA GeForce 920M и nks-30T (узел k40 (Kepler)).

Глава II. Реализации модели ассоциативных параллельных вычислений. Выводы.

- Реализованы базовые типы и операции языка STAR:
  - Время выполнения базовых операций практически не зависит от размера данных, что соответствует теоретическим оценкам: $O(\log_{64}(n))$ для реализации операции *FND*, *SOME* и *STEP*;
  - и константное время для реализации остальных операций.
- Реализована библиотека стандартных ассоциативных процедур:
  - ассоциативные алгоритмы оптимизированы под архитектуру GPU;
  - проведено сравнение времени работы процедур с аналогичными процедурами библиотек STL и CUDA thrust:
    - для векторов размерности более 5 000 элементов реализации представленных алгоритмов на GPU работают быстрее процедур библиотеки STL
    - созданная реализация базовых ассоциативных алгоритмов дает существенное ускорение по сравнению с библиотекой STL и, зачастую, выигрывает в производительности у библиотеки CUDA thrust.

Глава III. Ассоциативная обработка для решения задач теории графов

- Простые типы данных и операции, основанные на парадигмах ассоциативных вычислений, сделали модель универсальной;
- Позволяет строить параллельные алгоритмы для решения задач на графах:
  - обработка графа по вершинам, входящие/исходящие дуги обрабатываются параллельно;
  - полиномиальные задачи ($O(V^n) \to O(V^{n-1})$, $O(V+E) \to O(V)$);
  - динамические алгоритмы;
  - На практике ориентированны на графы $\approx 1000 - 10000$ вершин.

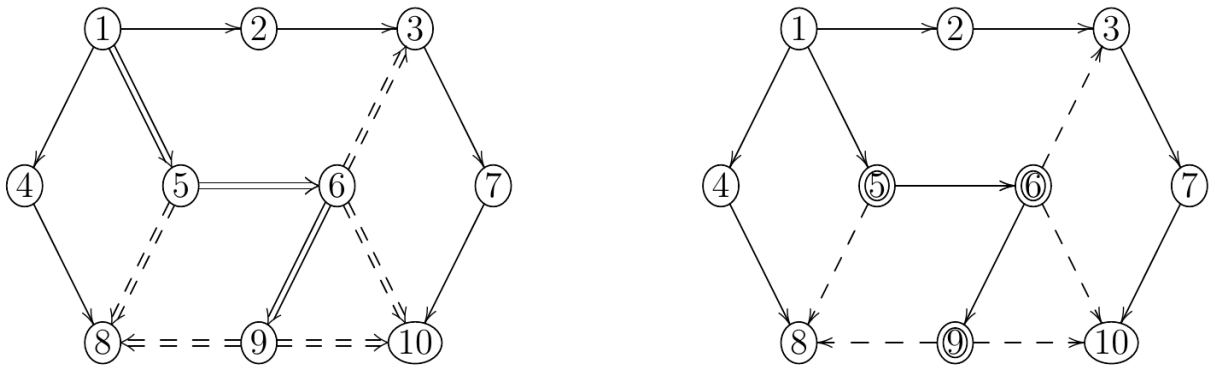


Рис.: Остовное дерево на множестве достижимых вершин после добавления дуги (1, 5). Последовательный и ассоциативный алгоритм.

Глава III. Star-алгоритмы для решения задач на графах. Реализация на GPU.

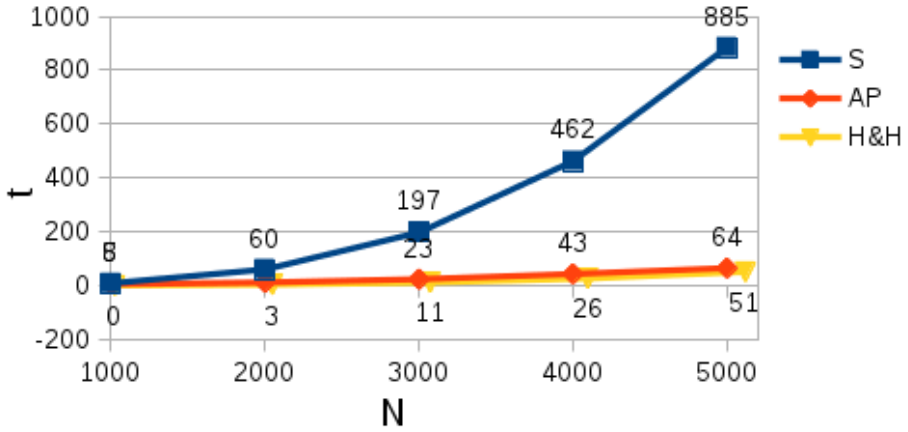
- Выработаны конкретные рекомендации для адаптации ассоциативных алгоритмов под исполнение на GPU;
- для алгоритма Уоршалла сделана оптимизация под исполнение на GPU;
- совместно с Непомнящей А. Ш. разработаны ассоциативные версии алгоритмов:
  - Инкрементальный алгоритм Дейкстры для динамической обработки дерева кратчайших путей после добавления новой дуги;
  - Инкементальный и декрементальный алгоритмы Рамалингама для решения проблемы достижимости в потоковых графах с одним источником;
- проведены численные эксперименты.

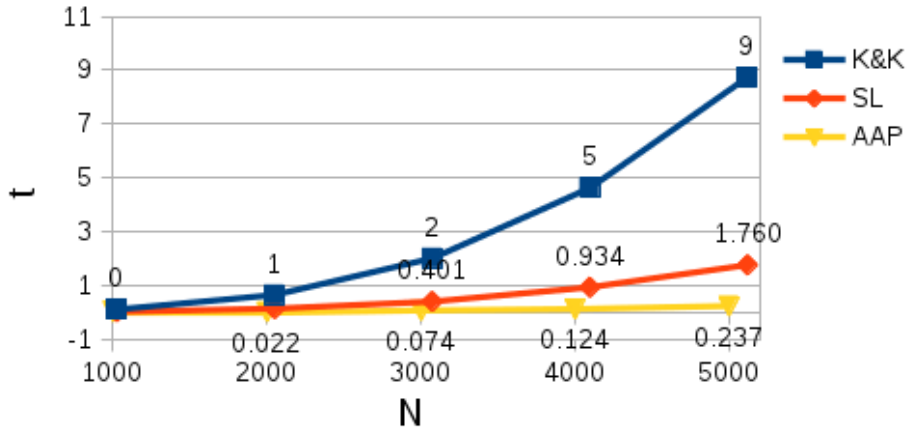
Глава III. Рекомендации для адаптации ассоциативных алгоритмов под выполнение на GPU.

Вследствие архитектурных особенностей Star-машины от GPU возникают несколько возможностей для оптимизации ассоциативных алгоритмов.

1. Определение точек синхронизации и уменьшение их числа: чтение/запись строк, циклы и условные операторы с предикатами SOME или ZERO.
2. Двухуровневый параллелизм. Проследить, есть ли зависимость по данным при обработке столбцов.
3. Уменьшение числа __global__-процедур.
   Для всех операторов, некритичных к синхронизации, и базовых алгоритмов II-IV групп можно использовать __device__-процедуры.

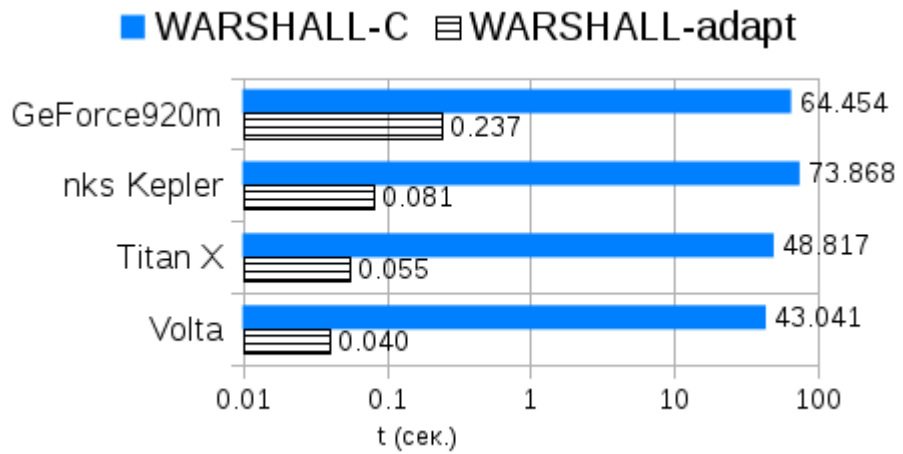
Глава III. Алгоритм Уоршалла. Сравнение времени работы различных реализаций на GPU





- **S** — последовательный алгоритм Уоршалла;
- **AP** — **Star-алгоритм WARSHALL-C**;
- **N&H** — реализация Нариньяри алгоритма Уоршалла;
- **AAP** — **адаптированный Star-алгоритм WARSHALL-adapt**;
- **K&K** — блочная реализация;
- **SL** — блочная реализация с использованием разделяемой памяти.

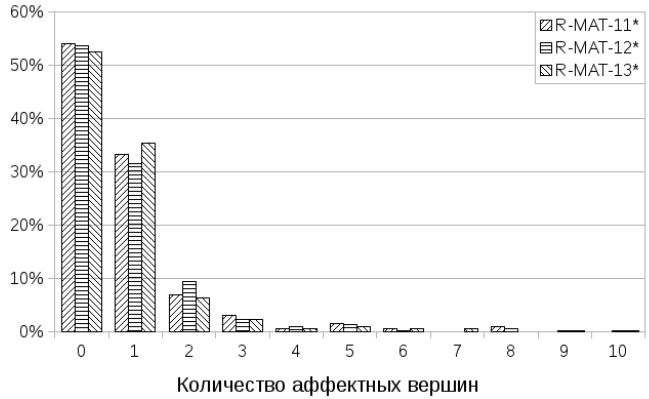
Глава III. Алгоритм Уоршалла. Сравнение времени работы на разных архитектурах графических ускорителей.



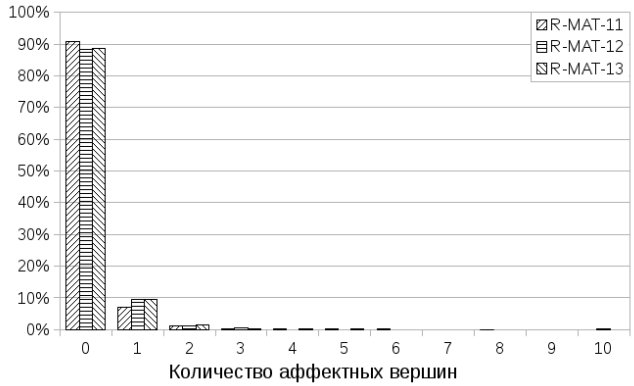
Расчет проводился на графе с 5000 вершин.

- **GeForce920m** — ноутбук;
- **nks Kepler** — кластер nks-30T ССКЦ (узел k40 Kepler);
- **Titan X** — кластер ФИТ НГУ;
- **Volta** — кластер НГУ;

Глава III. Инкрементальный алгоритм Дейкстры. Результаты вычислительного эксперимента.



Дуга с весом 0



Дуга со случайным весом

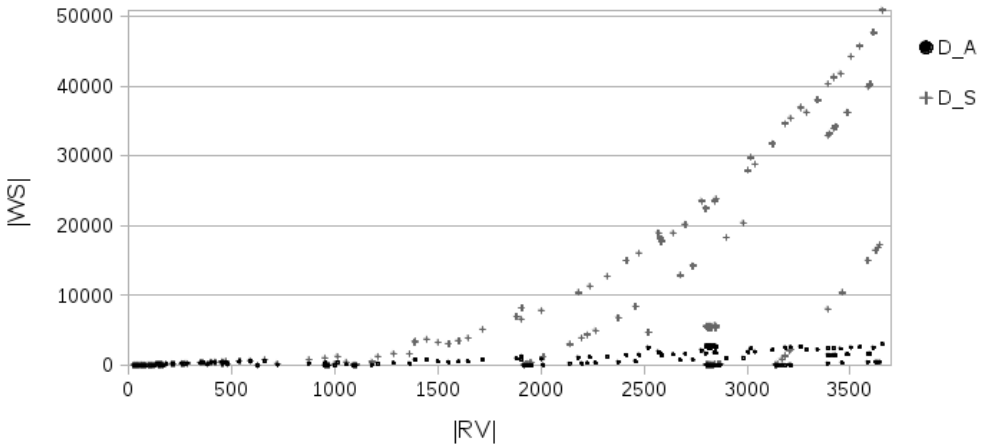
Сравнение времени работы статического и динамического ассоциативных алгоритмов на R-MAT-графах

Граф	n	DistSPT	InsertArcSPT*	k*	InsertArcSPT	k
R-MAT-11	2048	6,171	0,012	6/8	0,009	1/8
R-MAT-12	4096	9,643	0,017	5/10	0,012	3/10
R-MAT-13	8192	29,475	0,038	7/15	0,020	4/13

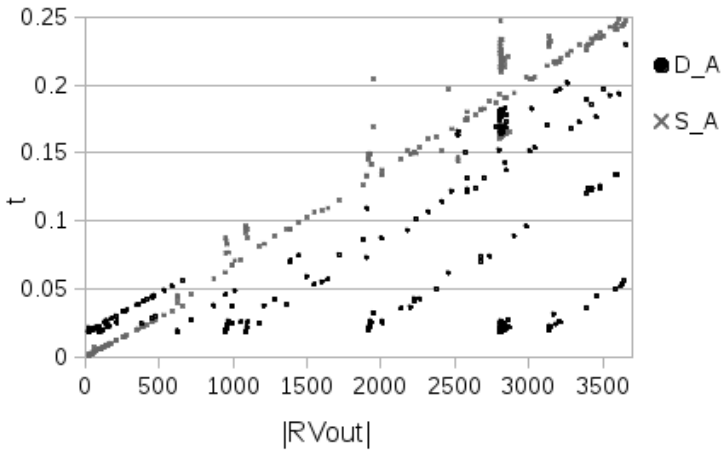
k — число аффектных вершин (max_t/max_k) Для последовательного алгоритма число обрабатываемых дуг до 2000 в первом режиме и до 100 во втором.

Глава III. Инкрементальный алгоритм Рамалингама. Результаты вычислительного эксперимента.

D_S - динамический последовательный алгоритм
D_A - динамический ассоциативный алгоритм
S_A - статический ассоциативный алгоритм



Динамические версии



Ассоциативные версии

Результаты тестирования показывают, что ассоциативный динамический алгоритм выполняется до десяти раз быстрее статического ассоциативного алгоритма, а также делает значительно меньше итераций, чем последовательный динамический алгоритм.

Глава III. Star-алгоритмы для решения задач на графах. Выводы.

- Разработаны и реализованы ассоциативные динамические алгоритмы, для которых не удалось обнаружить другие параллельные реализации.
  - дающие ускарение **на несколько порядков** по сравнению со статическими ассоциативными алгоритмами;
  - при этом число аффектных вершин в ассоциативном алгоритме от числа аффектных дуг в последовательном отличается в d раз, где d - средняя степень вершин в графе.
- В случае с алгоритмом Уоршала теоретическая сложность была снижена с $O(N^3)$ до $O(N)$. На практике алгоритм давал практически линейное время работы при том, что другие реализации выполнялись за квадратичное время.

Список публикаций I

- T. V. Snytnikova, A. V. Snytnikov. Implementation of the STAR-machine on GPU. Bulletin of Novosibirsk Computing Center, Computer Sience, Volume 39, 2016. P. 51–60.

Scopus Т. В. Снытникова, А. Ш. Непомнящая. Решение задач на графах с помощью STAR-машины, реализуемой на графических ускорителях // Прикладная дискретная математика. 2016. Vol. 3(33). P. 98–115.

ВАК Т.В. Снытникова. Реализация модели ассоциативных вычислений на GPU: библиотека базовых процедур языка STAR // Вычислительные методы и программирование. Новые вычислительные технологии. Т.19, выпуск 1, 2018, с. 85-95. DOI: 10.26089/NumMet.v19r108

ВАК Т. В. Снытникова. Развитие ассоциативных параллельных архитектур // Проблемы информатики. №2, 2019, с.36-50.

Список публикаций II

Scopus А.Ш. Непомнящая, Т.В. Снытникова. Ассоциативный параллельный алгоритм для динамической обработки дерева кратчайших путей после добавления новой дуги // Прикладная дискретная математика. 2019. № 46. С. 58-71.

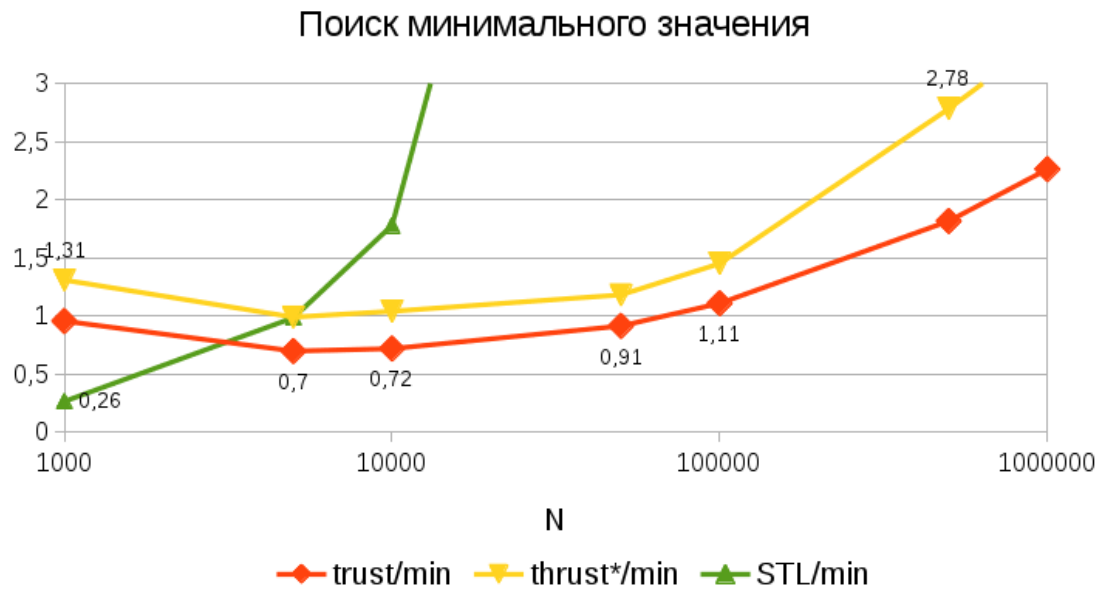
Scopus А. Ш. Непомнящая, Т.В. Снытникова. Ассоциативная версия инкементального алгоритма Рамалингама для решения проблемы достижимости в потоковых графах с одним источником // Вестник ТГУ УВТиИ, 2021. №54. С. 86–96. DOI: 10.17223/19988605/54/11

Спасибо за внимание!

I группа алгоритмов: на примере процедуры MIN. Сравнение с STL и CUDA thrust.

Производится сравнение производительности следующих реализаций:

- **min**;
- **STL:**
  std :: *min_element*();
- **thrust:**
  thrust :: *min_element*();
- **thrust*:** выдает все позиции минимального элемента.

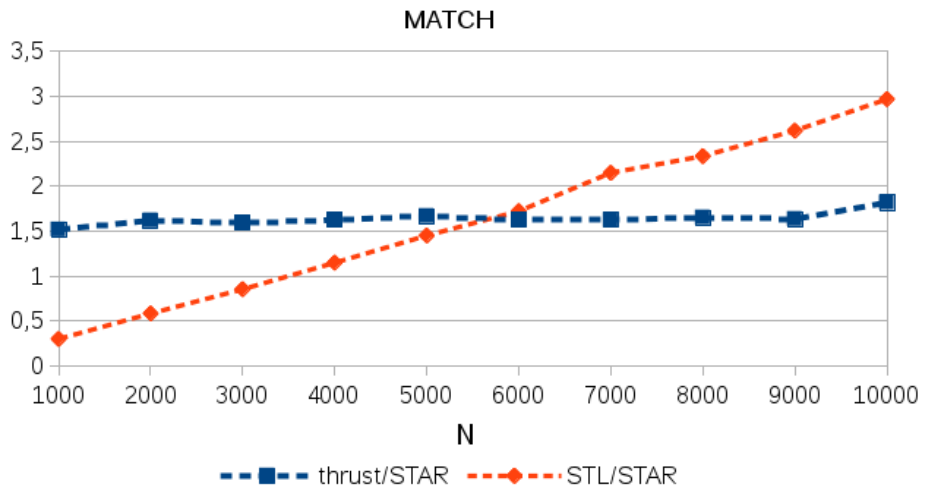


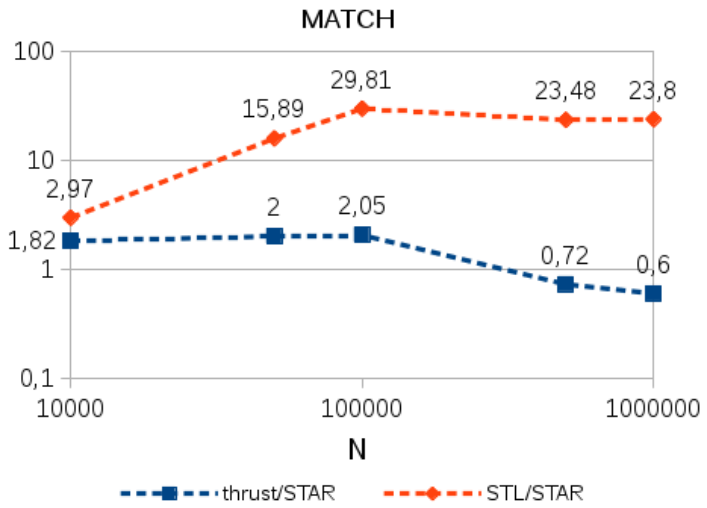
Реализация *min* дает заметный выигрыш по производительности относительно *thrust* при $N \geq 100\,000$.
Реализации *min* и *thrust*$*$ дают сравнимый результат на диапазоне $N \leq 50\,000$.

II группа алгоритмов: на примере процедуры MATCH. Сравнение с STL и CUDA thrust.

Производится сравнение производительности следующих реализаций:

- **STL:** *std* :: *find*();
- **thrust:** *thrust* :: *find*().



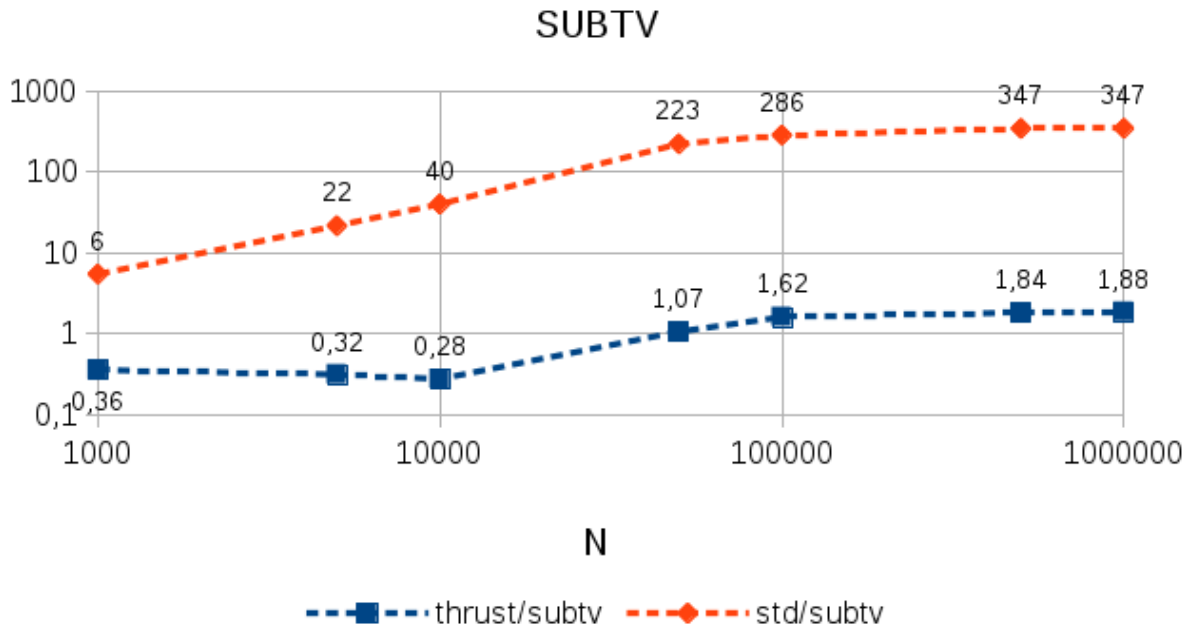


- Параллельные реализации выигрывают по производительности для векторов размерности более 4 000 - 6 000 элементов.
- **В случае, когда все блоки могут обрабатываться одновременно, *MATCH* работает в $1,5 - 2$ раза быстрее, чем thrust::find.**

III группа алгоритмов: на примере SUBTV. Сравнение с STL и CUDA thrust.

Производится сравнение производительности следующих реализаций:

- **std:** std::transform(...,std::minus<int>());
- **thrust:** thrust::transform(....thrust::minus<int>());



SUBTV работает быстрее, чем thrust на данных большего размера (более 100 000 элементов). Последовательная реализация сильно уступает в производительности параллельным.

IV группа алгоритмов: на примере процедуры TMERGE

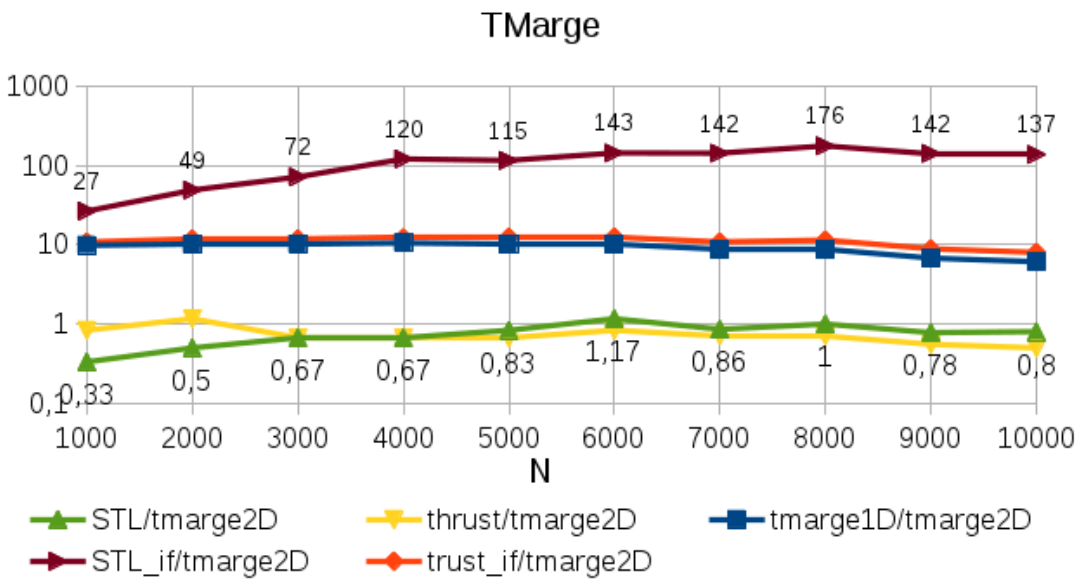
- **tmerge1D:** столбцы обрабатываются поочередно, как во II группе алгоритмов.
- **tmerge2D:** столбцы обрабатываются одновременно, $gridDim.y = h$ (**двухуровневый параллелизм**).

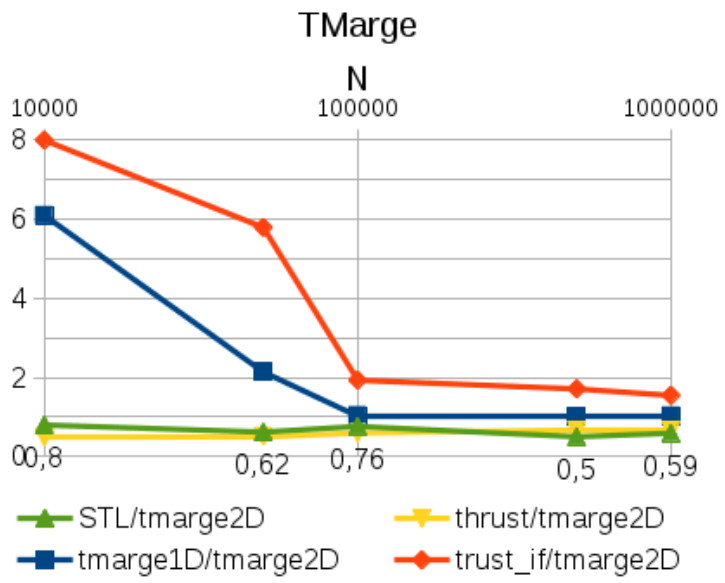
Производится сравнение со следующими реализациями:

- **STL:** std::copy() производит копирование всех элементов;
- **STL_if:** copy_if()[1] производит копирование только тех элементов, которые удовлетворяют некоторому предикату;
- **thrust:** thrust::copy() производит копирование всех элементов;
- **thrust_if:** thrust::copy_if() производит копирование только тех элементов, которые удовлетворяют некоторому предикату.

[1] Бьерн Страуструп. Язык программирования C++. Специальное издание.

IV группа алгоритмов: сравнение с STL и CUDA thrust.





- **tmerge2D сравнима в работе с беспредикатными версиями стандартных библиотек STL и thrust ($4-6\mu c$ на векторах до $10\,000$) и выигрывает в производительности у предикатных;**
- **tmerge1D сравнима в работе с thrust_if на векторах до $10\,000$ элементов ($60; 70-80\mu c$) и сравнивается с tmerge2D на векторах более $100\,000$ элементов.**

Паспорт специальности 05.13.11 Области исследований:

1 Модели, методы и алгоритмы проектирования и анализа программ и программных систем, их эквивалентных преобразований, верификации и тестирования.
2 Языки программирования и системы программирования, семантика программ.
3 Модели, методы, алгоритмы, языки и программные инструменты для организации взаимодействия программ и программных систем.
7 Человеко-машинные интерфейсы; модели, методы, алгоритмы и программные средства машинной графики, визуализации, обработки изображений, систем виртуальной реальности, мультимедийного общения.
8 Модели и методы создания программ и программных систем для параллельной и распределенной обработки данных, языки и инструментальные средства параллельного программирования.
9 Модели, методы, алгоритмы и программная инфраструктура для организации глобально распределенной обработки данных.
10 Оценка качества, стандартизация и сопровождение программных систем.