Тема доклада

Параллельная реализация операций над множествами в задаче обработки знаний

Синцов С.В.

Белорусский государственный университет информатики и радиоэлектроники
Республика Беларусь, г. Минск

План доклада

1. Введение (постановка задачи)
2. Существующие решения
3. Разработка алгоритмов
4. Реализация алгоритмов
5. Эксперимент
6. Заключение

1 Введение

Общее назначение – параллельная обработка информации в системах, основанных на знаниях.

В частности:

- реализация операций поддержки теоретико-модельной семантики в языках представления и обработки знаний (OWL, SCP);
- обеспечение для систем, основанных на знаниях, гарантий по производительности (затратам времени и памяти), что важно при разработке систем, управляемых знаниями.

Актуальность обусловлена необходимостью развития интеллектуальных систем, существованием физических ограничений и распространением параллельных вычислительных архитектур (суперскалярных, векторных, многоядерных, многопроцессорных) в качестве аппаратной основы систем, основанных на знаниях.

1 Постановка задачи

Цель: разработка и реализация программных средств параллельных базовых операций над множествами.

Задачи:

1. **Обзор существующих решений.**
2. Спецификация базовых теоретико-множественных операций модели представления и обработки знаний в виде унифицированных семантических сетей с базовой теоретико-множественной интерпретацией.
3. **Разработка и реализация алгоритмов параллельных операций пересечения, объединения и разности множеств.**
4. **Экспериментальная проверка реализованных алгоритмов.**

1 Постановка задачи

В докладе будут рассмотрены

- операции пересечения и объединения множеств (а также мультимножеств – множеств с кратными элементами);
- не деструктивные операции, т.е. операции не разрушающие структуру своих операндов;
- среднезернистый параллелизм в форме независимых ветвей и векторных операций для вычислительных машин с общей (разделяемой) памятью.

2 Существующие решения

Работы предшественников

1	Multisets / S. Baxter // NVIDIA Research. https://nvlabs.github.io/moderngpu/sets.html. – 2013.	Базовые операции над мультимножествами. CUDA реализация.
2	Efficient Lists Intersection by CPU-GPU Cooperative Computing / Di Wu, F. Zhang, Naiyong Ao, et al. // IPDPS. – 2010.	Пересечение множеств без кратных элементов. CUDA реализация.
3	Time-Space Optimal Parallel Set Operations / X.Guan, M.A.Langston // PARBASE'90. – 1990.	Математическая модель без экспериментальных результатов
4	Fast Set Operations Using Treaps / G.E. Blelloch, M. Reid-Miller // SPAA'98. – 1998.	Данные закодированы деревьями. Потоки ОС.
5	Just Join for Parallel Ordered Sets / G.E. Blelloch, D. Ferizovic, Y.Sun // SPAA'16. – 2016.	Данные закодированы деревьями. Потоки ОС.

3 Разработка алгоритмов

На входе алгоритмов пересечения и объединения
множества **A** и **B**, $Length(B) \geq Length(A)$, закодированные в виде **упорядоченных массивов** A и B целых положительных m-битных **чисел**, причём кратные элементы с кратностью k дублируются в массиве k раз:

$A=$ | 1 | 1 | 4 | 5 | 7 | 7 | 7 | 8 |

$B=$ | 1 | 1 | 1 | 2 | 2 | 7 | 8 | 9 |

На выходе – множество (**A**∩**B** или **A**∪**B**), закодированное в виде упорядоченного массива целых положительных m-битных чисел:

A∩**B**→ | 1 | 1 | 7 | 8 |

A∪**B**→ | 1 | 1 | 1 | 2 | 2 | 4 | 5 | 7 | 7 | 7 | 8 | 9 |

3 Пересечение мультимножеств

IntersectParallel(⟨*A*,*B*⟩)

Шаг 1. Выполнить поиск минимального и максимального индексов вхождения каждого элемента массива *A* в массив *B*:
⟨*L*, *U*⟩ ← *RangeBinarySearch*(⟨*A*, *B*⟩).

Шаг 2. Отметить в массиве *C* минимальные индексы равных элементов массива *A*:

$$C[i] \leftarrow \begin{cases} i \,\middle|\, A\left[\max\left(\{0\} \cup \{i-1\}\right)\right] < A[i] \\ 0 \,\middle|\, A\left[\max\left(\{0\} \cup \{i-1\}\right)\right] = A[i] \end{cases}$$

Шаг 3. Вычислить максимумы
C ← *MaxScatter*(*C*):

$k \leftarrow ((\sim 0) >> (clz(Length(A)) - 1)) - 1$
пока $(k > 0)$:
  если $(i \geq k)$, *то* $C[i] \leftarrow \max(\{C[i]\} \cup \{C[i+k]\})$
  $k >> 1$

Шаг 4. Вычислить

$$D[i] \leftarrow \begin{cases} 1 \,\middle|\, i - C[i] < U[i] - L[i] \\ 0 \,\middle|\, i - C[i] \geq U[i] - L[i] \end{cases}$$

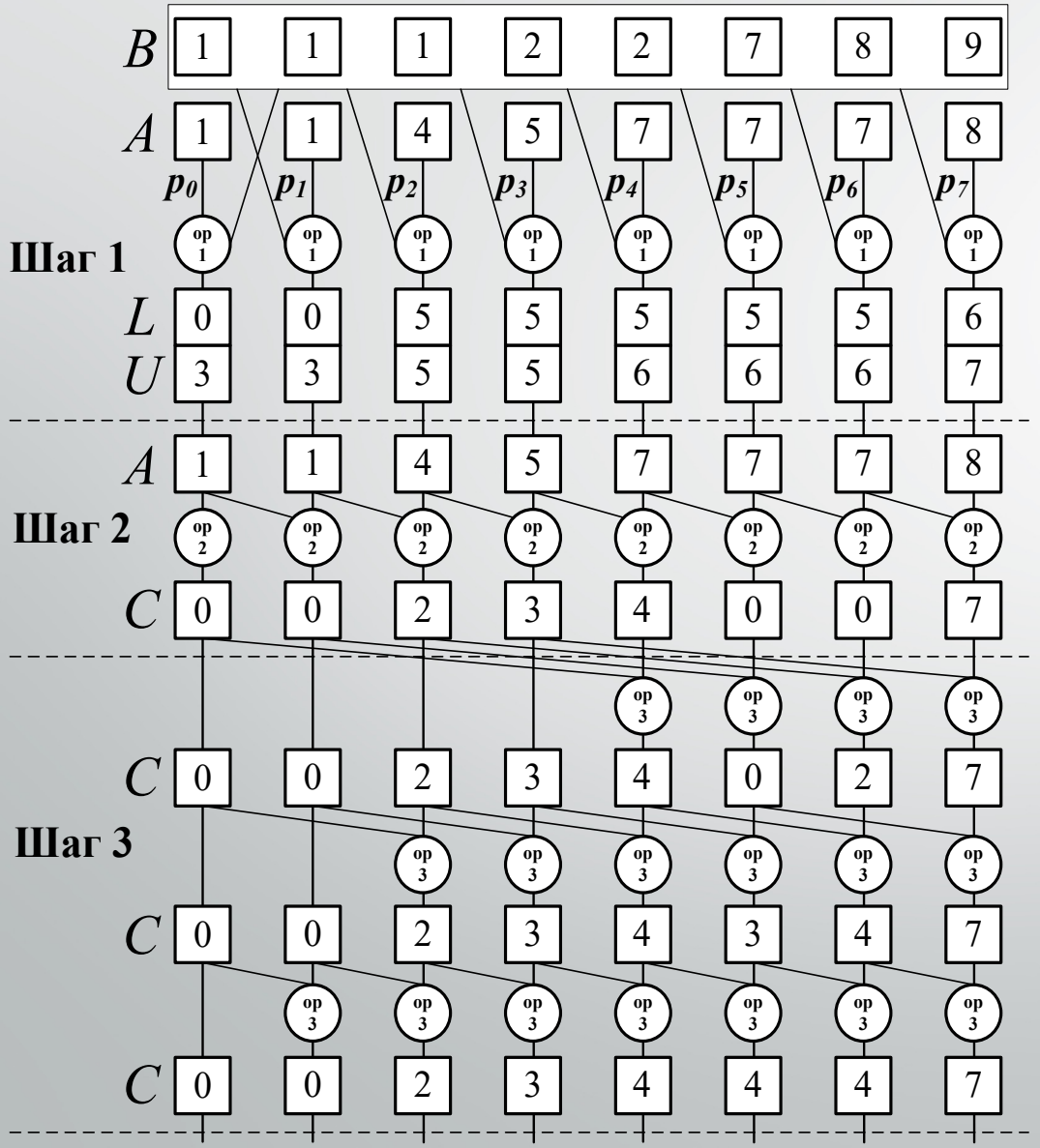
Шаг 5. Вычислить массив *E* префиксных сумм массива *D*: *E* ← *PrefixSum*(*D*).

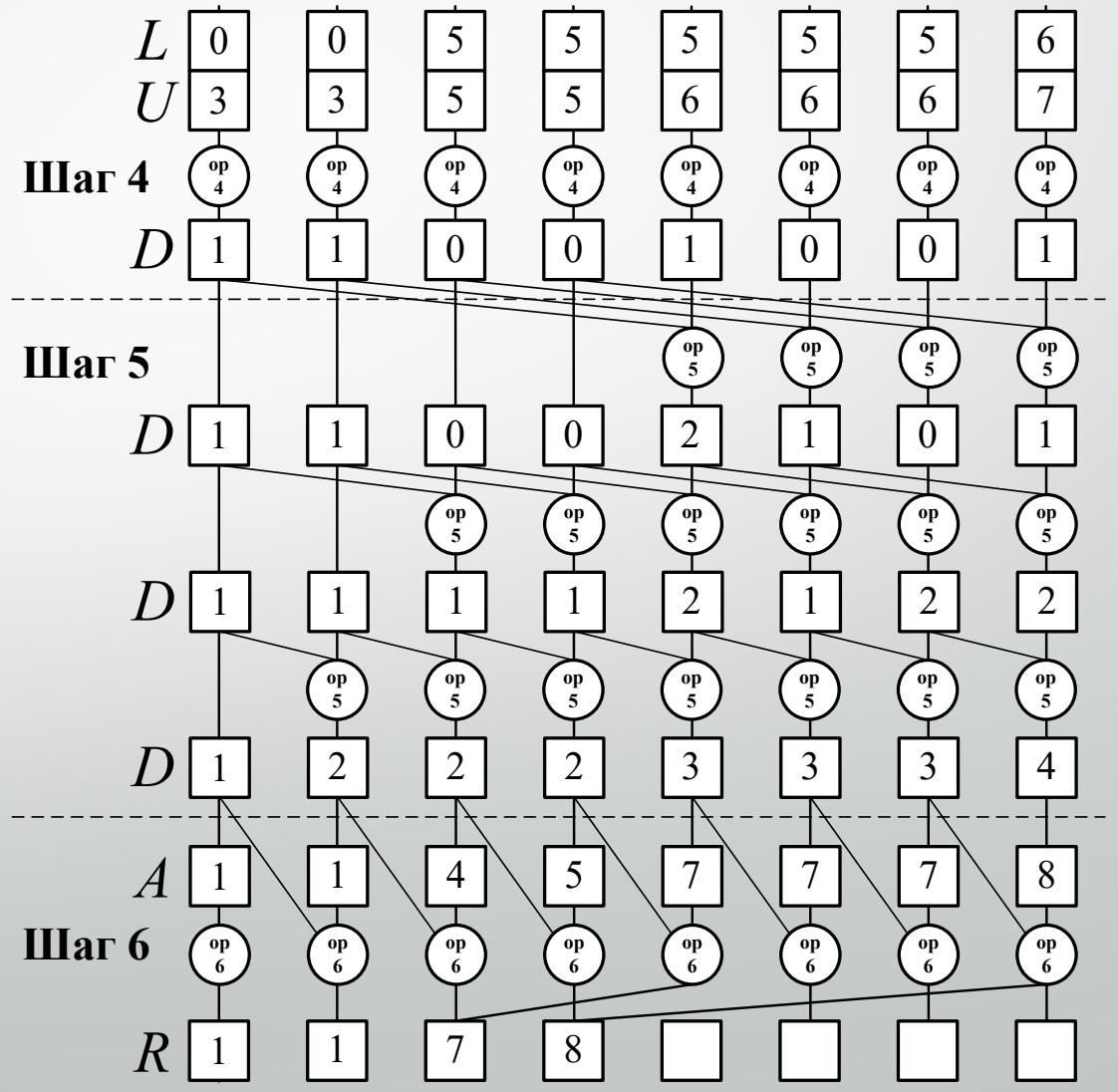
Шаг 6. Вычислить:
если $(D[i] = 1)$, *то* $R[E[i] - 1] \leftarrow A[i]$.

Шаг 7. Возвратить ⟨*R*, *E*[*Length*(*A*) − 1]⟩.

3 Пересечение мультимножеств

Информационный граф





3 Объединение мультимножеств

UniteParallel(⟨*A*,*B*⟩)

Шаг 1. Выполнить поиск минимального и максимального индексов вхождения каждого элемента массива *A* в массив *B*:
⟨*L*, *U*⟩ ← *RangeBinarySearch*(⟨*A*, *B*⟩).

Шаг 2. Отметить в массиве *C* минимальные индексы равных элементов массива *A*:

$$C[i] \leftarrow \begin{cases} i \,\middle|\, A\left[\max\left(\{0\} \cup \{i-1\}\right)\right] < A[i] \\ 0 \,\middle|\, A\left[\max\left(\{0\} \cup \{i-1\}\right)\right] = A[i] \end{cases}$$

Шаг 3. Вычислить максимумы *C* ← *MaxScatter*(*C*):

$k \leftarrow ((\sim 0) >> (clz(Length(A)) \& 1)) \& 1$
пока $(k > 0)$:
  если $(i \geq k)$, *то* $C[i] \leftarrow \max(\{C[i]\} \cup \{C[i-k]\})$
  $k >> 1$

Шаг 4. Вычислить

$$D[i] \leftarrow \begin{cases} 0 \,\middle|\, i - C[i] < U[i] - L[i] \\ 1 \,\middle|\, i - C[i] \geq U[i] - L[i] \end{cases}$$

Шаг 5. Вычислить массив *E* префиксных сумм массива *D*: *E* ← *PrefixSum*(*D*).

Шаг 6. Выполнить для каждого элемента массива *E*: *если* $(((i \neq 0) \& (E[i] \neq 1)) \vee ((i \neq 0) \& (E[i] \neq E[i-1] + 1)))$, *то* $G[i] \leftarrow E[i]$, *иначе* $G[i] \leftarrow U[i]$, *иначе* $G[i] \leftarrow 0$.

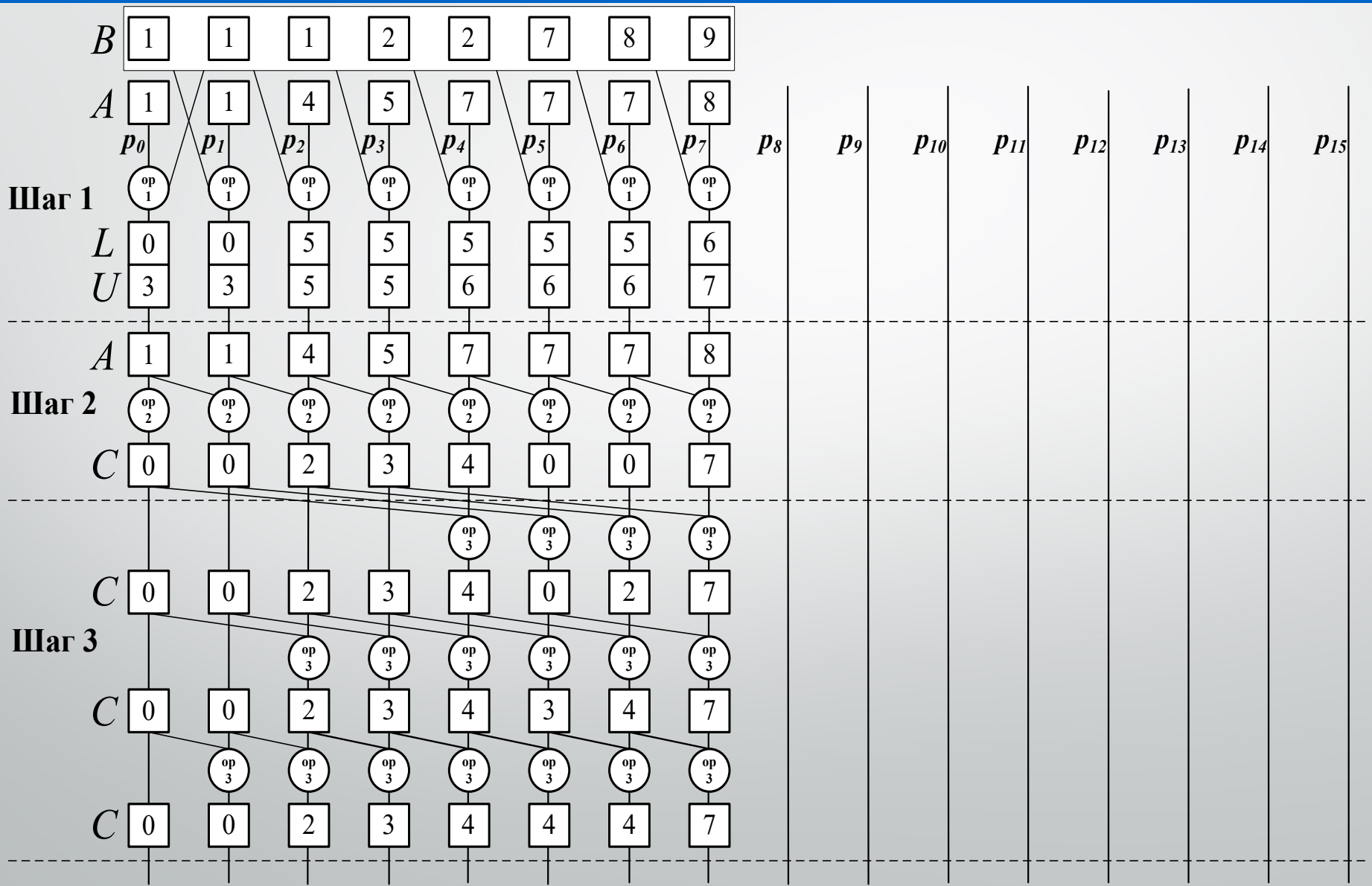
3 Объединение мультимножеств

UniteParallel(⟨*A*,*B*⟩)

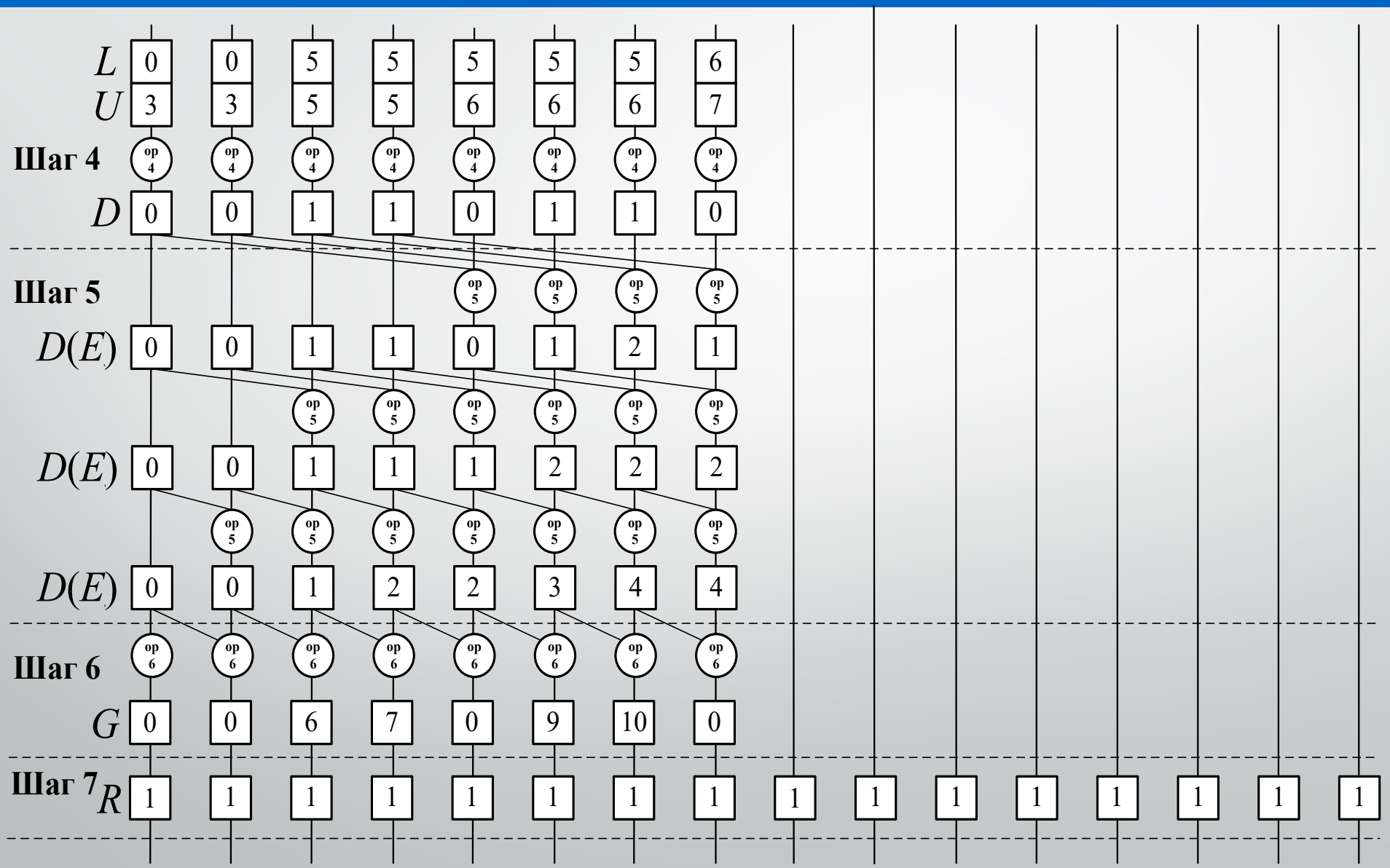
Шаг 7. Вычислить
для i от 0 до Length(*A*)□*Length*(*B*)□1: *R*□*i*□□1.
Шаг 8. Выполнить для каждого элемента массива *G*: *если* (*G*□*i*□□0), *то R*□*G*□*i*□□1□ □ 0.
Шаг 9. Вычислить массив *H* префиксных сумм массива *R*:
H □ *PrefixSum*(*R*); *H*□□1□ □ 0.
Шаг 10. Выполнить для каждого элемента (*i*≥0) массива *H*:
если ((*H*□*i*□≤*Length*(*B*))□(*H*□*i*□1□□□*H*□*i*□)),
то R□*i*□□ *B*□*H*□*i*□□1□.
После этого массив *R* содержит все элементы массива *B*.

Шаг 11. Выполнить для каждого элемента массива *A*:
если ((*i*□*Length*(*A*))□(*G*□*i*□□0)),
то R□G□*i*□□1□ □ *A*□*i*□.
После этого массив *R* содержит и элементы массива *A*.
Шаг 12. Возвратить
⟨*R*, *E*□*Length*(*A*)□1□ □ *Length*(*B*)⟩.

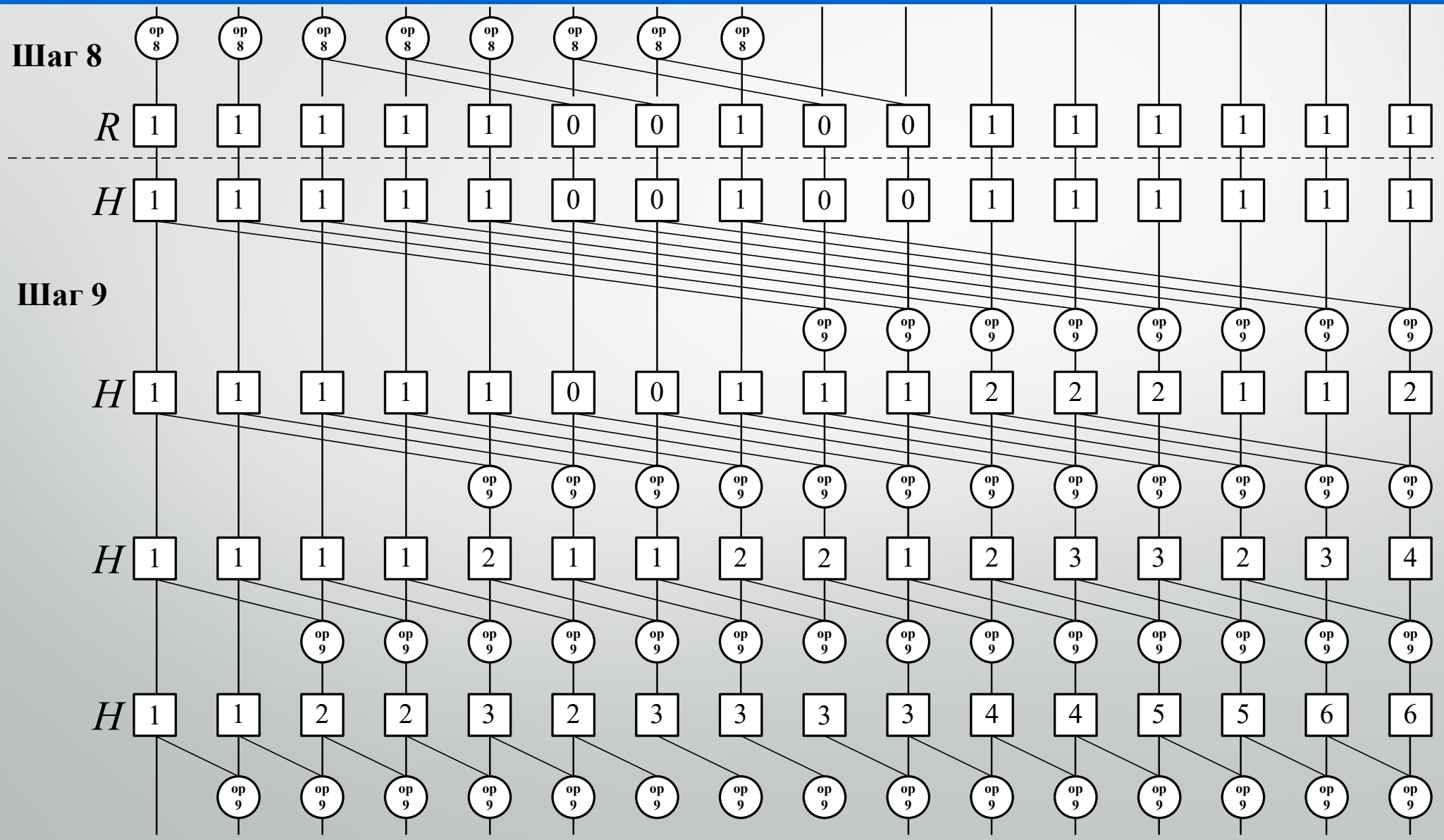
3 Объединение мультимножеств
Информационный граф



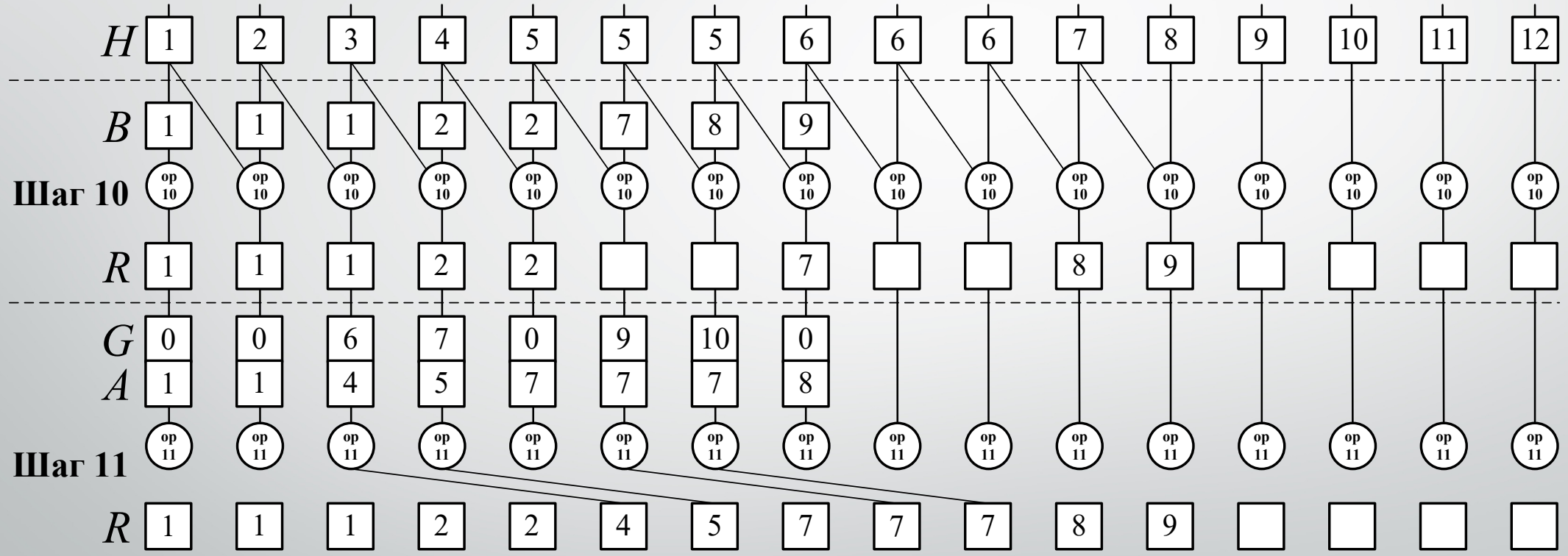
3 Объединение мультимножеств
Информационный граф



3 Объединение мультимножеств
Информационный граф



3 Объединение мультимножеств
Информационный граф



4 Реализация алгоритмов

Для реализации был выбран инструмент OpenCL 1.2.

Особенности платформы OpenCL:

- Основана на открытом стандарте.
- Позволяет работать с различными типами устройств гетерогенной вычислительной архитектуры.
- Позволяет запускать одну и ту же программу на различных типах устройств.

5 Эксперимент

Запуск программ производился на вычислительных архитектурах

- CPU (Intel Core i7 3520M CPU)
- GPU (Intel HD Graphics 4000)

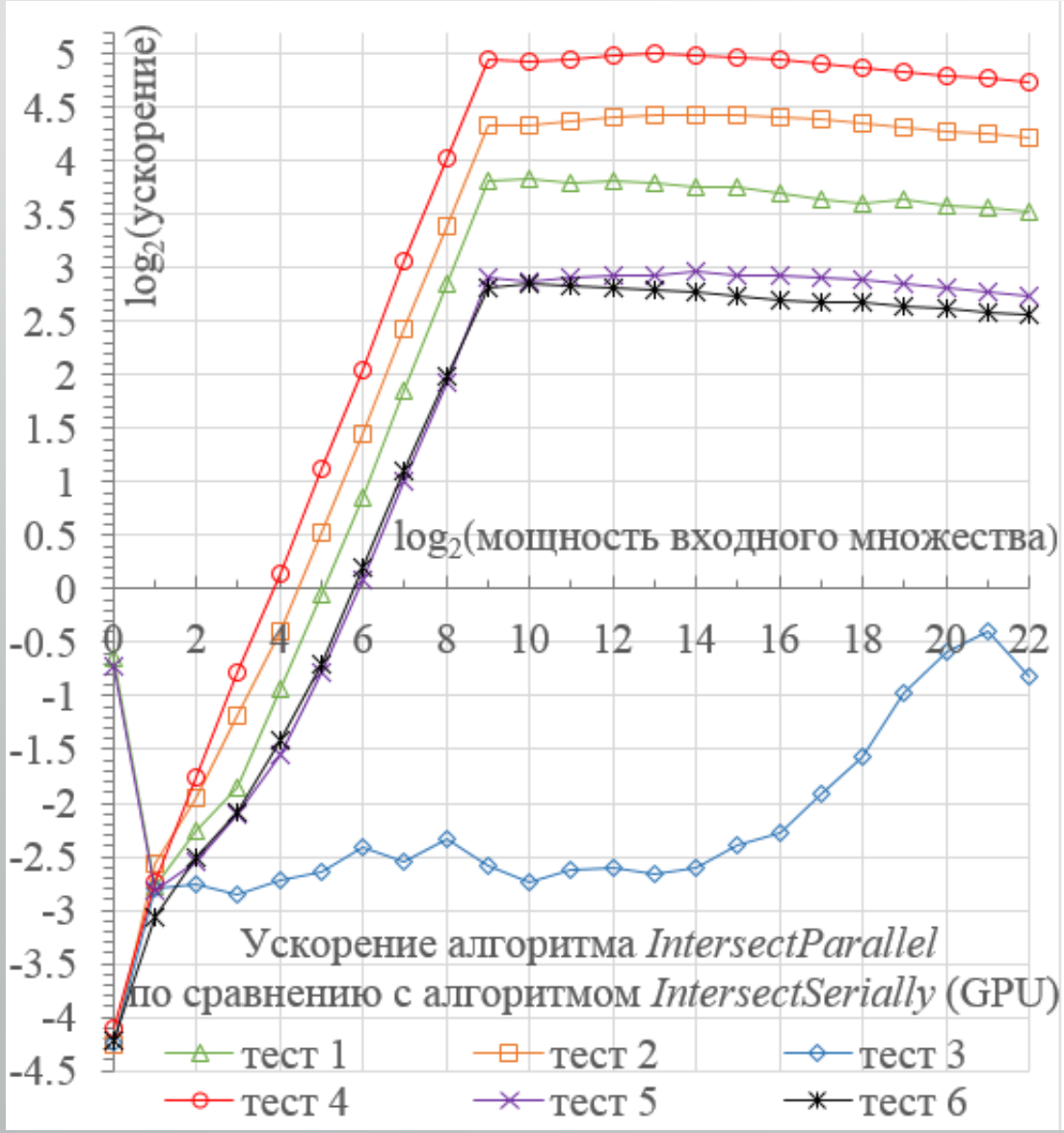
Использовались входные массивы шести типов:

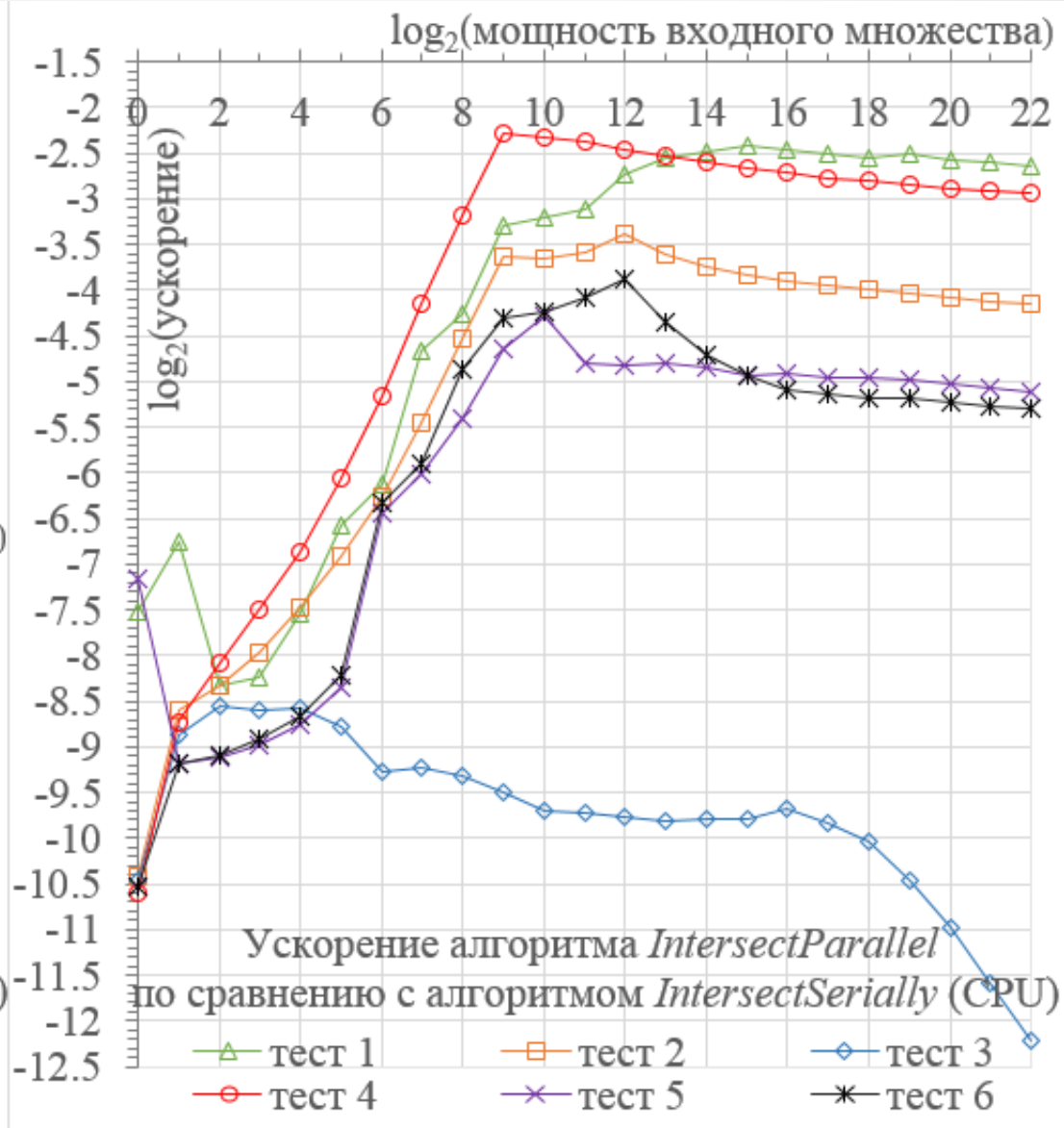
1. каждый $A[i]$ и $B[j]$ – до упорядочения есть псевдослучайная величина, равномерно распределённая на $[0, \max(Length[A], Length[B])/2]$;
2. $A[1,3,5,\ldots]$, $B[0,2,4,\ldots]$;
3. A и B не содержат кратных элементов и любой $A[i]$ больше любого $B[j]$;
4. $A[0,1,4,5,8,9,\ldots]$, $B[2,3,6,7,10,11,\ldots]$;
5. $A[1,2,3,\ldots]$, $B[1,2,3,\ldots]$;
6. $A[0,0,0,\ldots]$, $B[0,0,0,\ldots]$.

5 Эксперимент

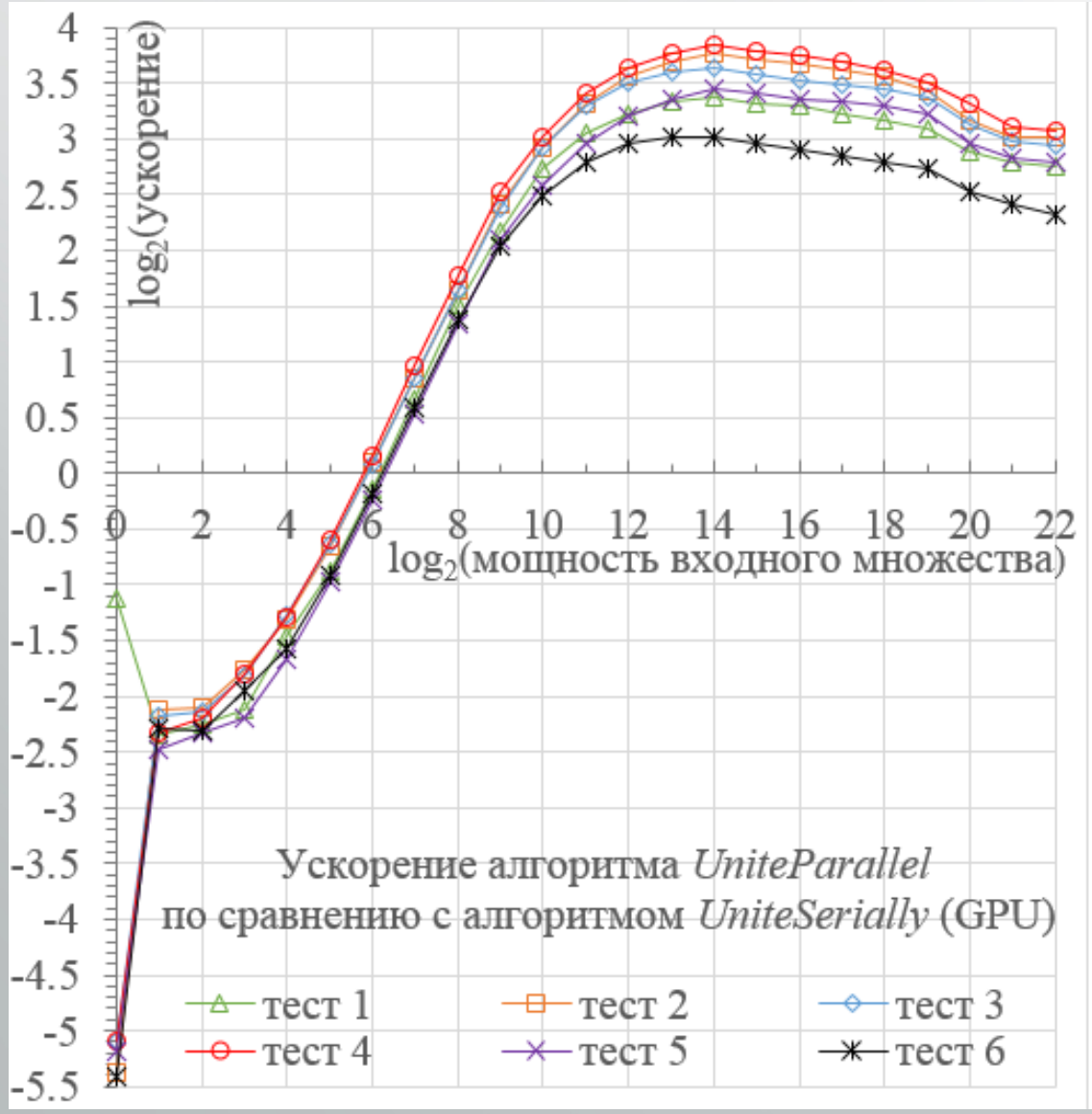
- Сравнение алгоритмов параллельных операций проводилось с алгоритмами последовательных операций *IntersectSerially* и *UniteSerially*.
- В основе последовательных реализаций – алгоритм слияния упорядоченных списков.
- OpenCL программа операции *IntersectParallel* (*UniteParallel*) исполнялась только устройством GPU, одной рабочей группой и максимально возможным количеством рабочих элементов.
- Программа операции *IntersectSerially* (*UniteSerially*) исполнялась
  1) в виде OpenCL гранулы устройством GPU, одной рабочей группой и одним рабочим элементом;
  2) устройством CPU в виде однопоточной программы, написанной на языке C++.

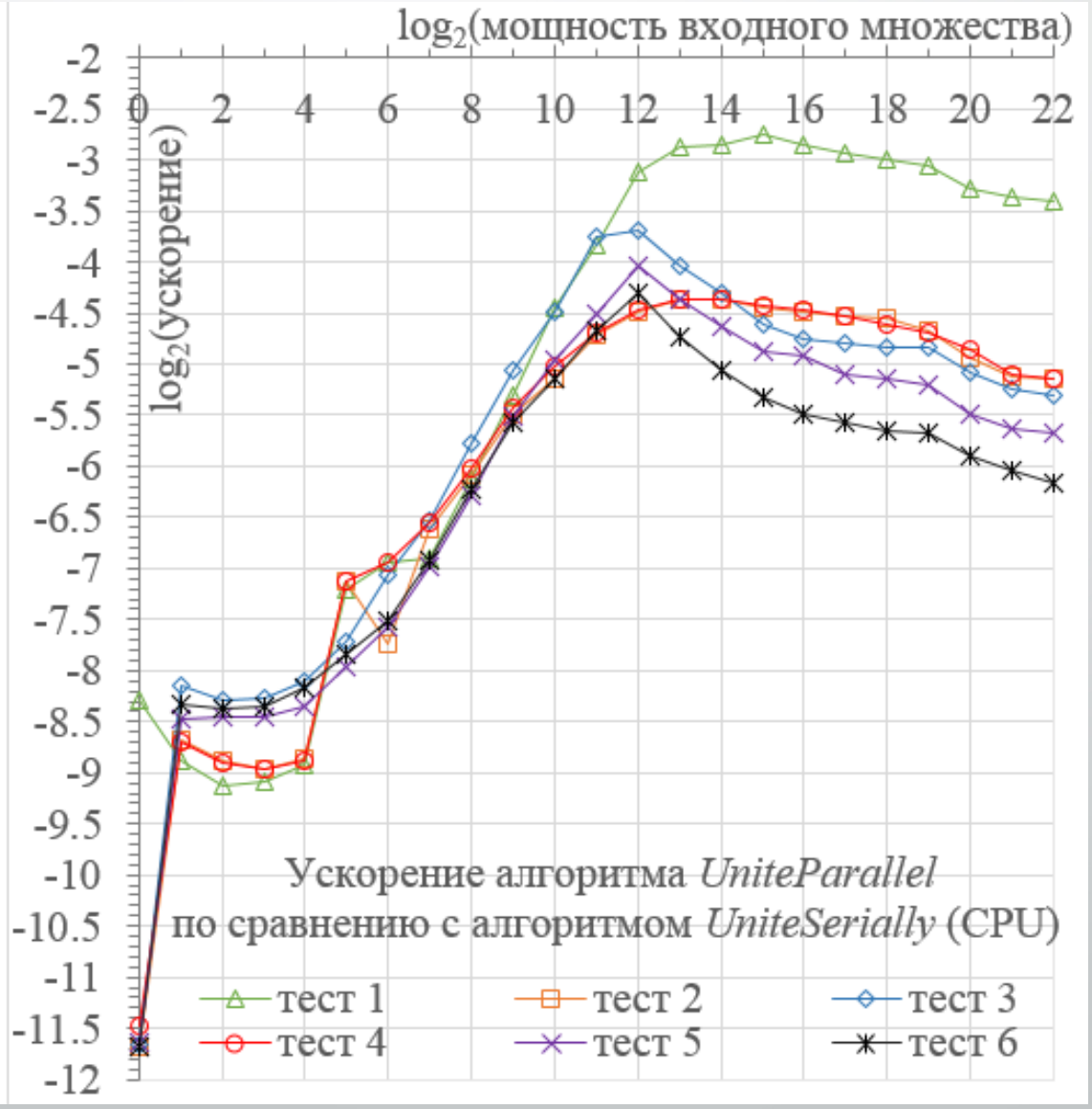
5 Эксперимент





5 Эксперимент





6 Заключение

- Параллельные реализации алгоритмов пересечения и объединения **масштабируются** с ростом числа процессоров.
- **Пропускная способность** операции пересечения (объединения) оказывается гораздо **ниже** пропускной способности операции пересечения (объединения), выполненной последовательным алгоритмом на устройстве CPU.
- В **лучшем случае** алгоритм *IntersectParallel* (*UniteParallel*) опережает *IntersectSerially* (*UniteSerially*) (GPU) в **десятки раз**, хотя и не достигает значений идеальной теоретической временной оценки.
- В **худшем** же **случае** *IntersectParallel* **неэффективен**: например, для входных данных типа 3 операция пересечения не совершает полезной работы, но на её исполнение затрачивается значительное процессорное время.
- **Велика скрытая константа**: текущая модель и архитектура памяти создают препятствия в виде «бутылочного горлышка».
- Появляется **возможность разгрузки** устройства **CPU**.

Спасибо за внимание!