Опыт разработки гибридных версий решателей разреженных СЛАУ

Авторы:
Гризан С.А.
Кривов М.А.

Новосибирск, 2012

Постановка задачи

- Эллиптическое уравнение

$$\nabla \Phi(x, y, z) = f(x, y, z)$$

- Относительно произвольная область *S*

$$\bar{S} = G_1 \cup G_2$$

- Смешанные граничные условия

$$\Phi(x, y, z)|_{(x, y, z) \in G1} = g_1(x, y, z)$$

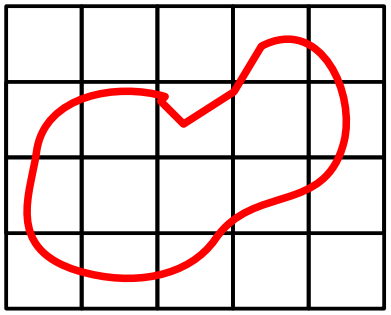
$$\frac{\partial \Phi(x, y, z)}{\partial \vec{n}}|_{(x, y, z) \in G2} = g_2(x, y, z)$$

Содержание

- **Выбор сеток**
- Решатель #1
- Решатель #2
- Выводы

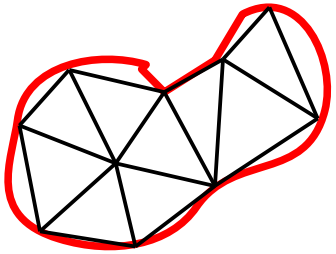
Выбор формата сеток

- Вариант #1: структурированная



```
double phi[6 * 5];
```

- Вариант #2: неструктурированная



```
struct vertex
{
    double value;
    int neighbours[N];
};

vertext grid[11];
```

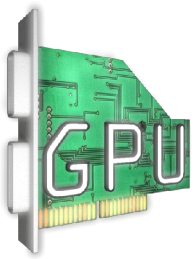
Структурированная сетка

- Плюсы:
  - Все соседи известны во время компиляции

    Меньше вычислений из-за более простой логики обсчёта *(на одну вершину)*
  - Возможность динамического разрезания

    Удастся эффективно задействовать *shared memory*
- Минусы:
  - Множество «лишних» узлов

    Требуется больше вычислений и памяти для достижения заданной точности *(для всех вершин)*



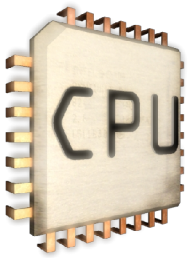
Лучше подходит для графических ускорителей!

Неструктурированная сетка

- Плюсы:
  - Нет «лишних» узлов

    Требуемая точность за минимальное количество вершин
- Минусы:
  - Сложный формат данных

    Требуется множество не-*coalesced* обращений к памяти



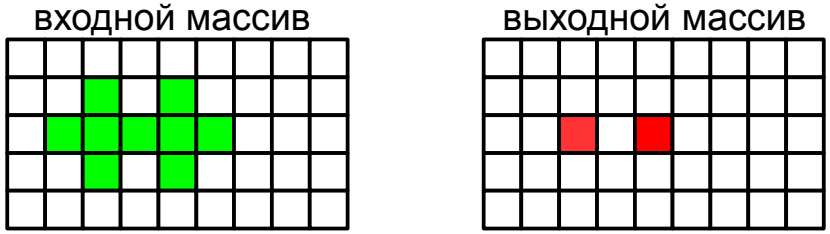
Лучше подходит для центральных процессоров!

Что будет быстрее?

«Медленные» вычисления на небольшой сетке
или
«быстрые» на большой?

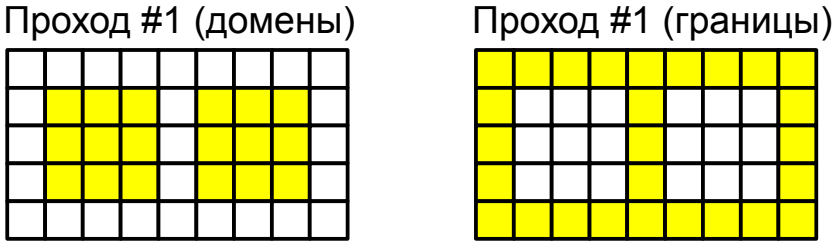
Тестовые реализации (метод Якоби для задачи Дирихле)

- Структурированная сетка
  - Shared memory



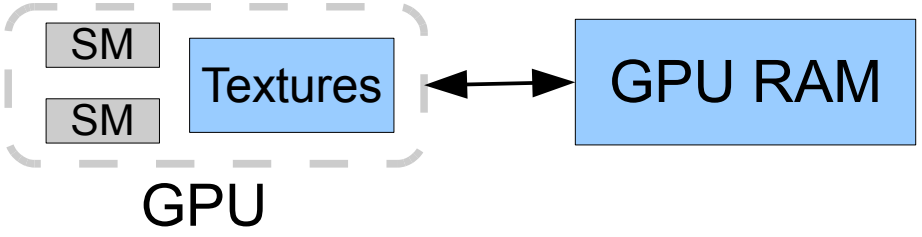
5 операций чтения -> 1 операция чтения

  - Two-pass kernel



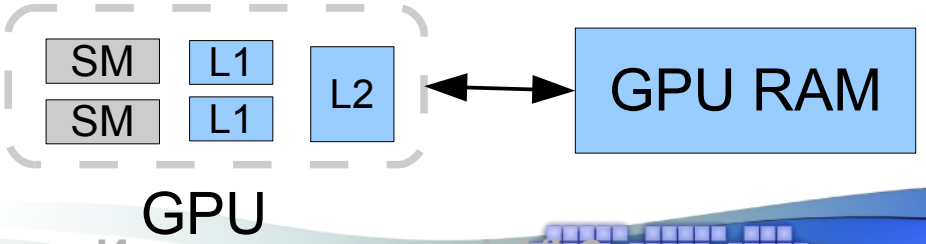
Отсутствие if'оа и не-coalesced доступа

- Неструктурированная сетка
  - Texture memory



Использование текстурного кэша

  - Кэши L1/L2



Использование L1/L2 кэша

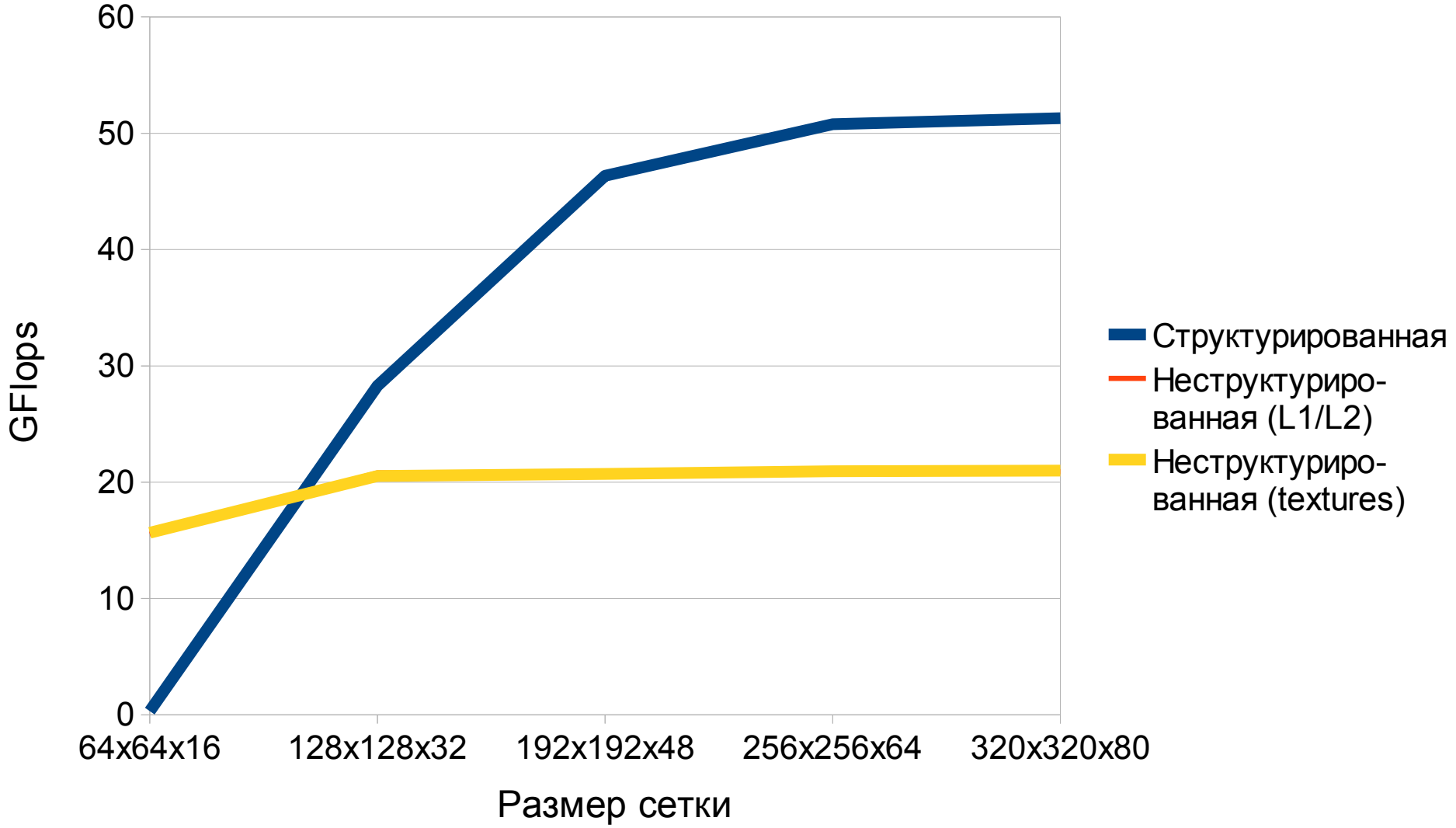
Результаты тестирования

NVidia Tesla C2050 + Intel Core 2 Quad Q6600

Сетка	Структурированная		Неструктурированная (упорядоченная)			Неструктурированная (перемешанная)		
	GPU	CPU	GPU (L1 кэш)	GPU (textures)	CPU	GPU (L1 кэш)	GPU (textures)	CPU
64x64x16	0,31	1,03	16,13	14,89	0,69	15,52	15,65	0,63
128x128x32	28,28	1,46	20,88	20,59	0,35	20,8	20,53	0,4
192x192x48	46,34	1,56	20,69	20,77	0,53	20,72	20,71	0,69
256x256x64	50,77	1,54	20,69	21,06	0,32	20,97	20,94	0,64
320x320x80	51,3	1,53	21	21,13	0,55	20,9	20,99	0,67

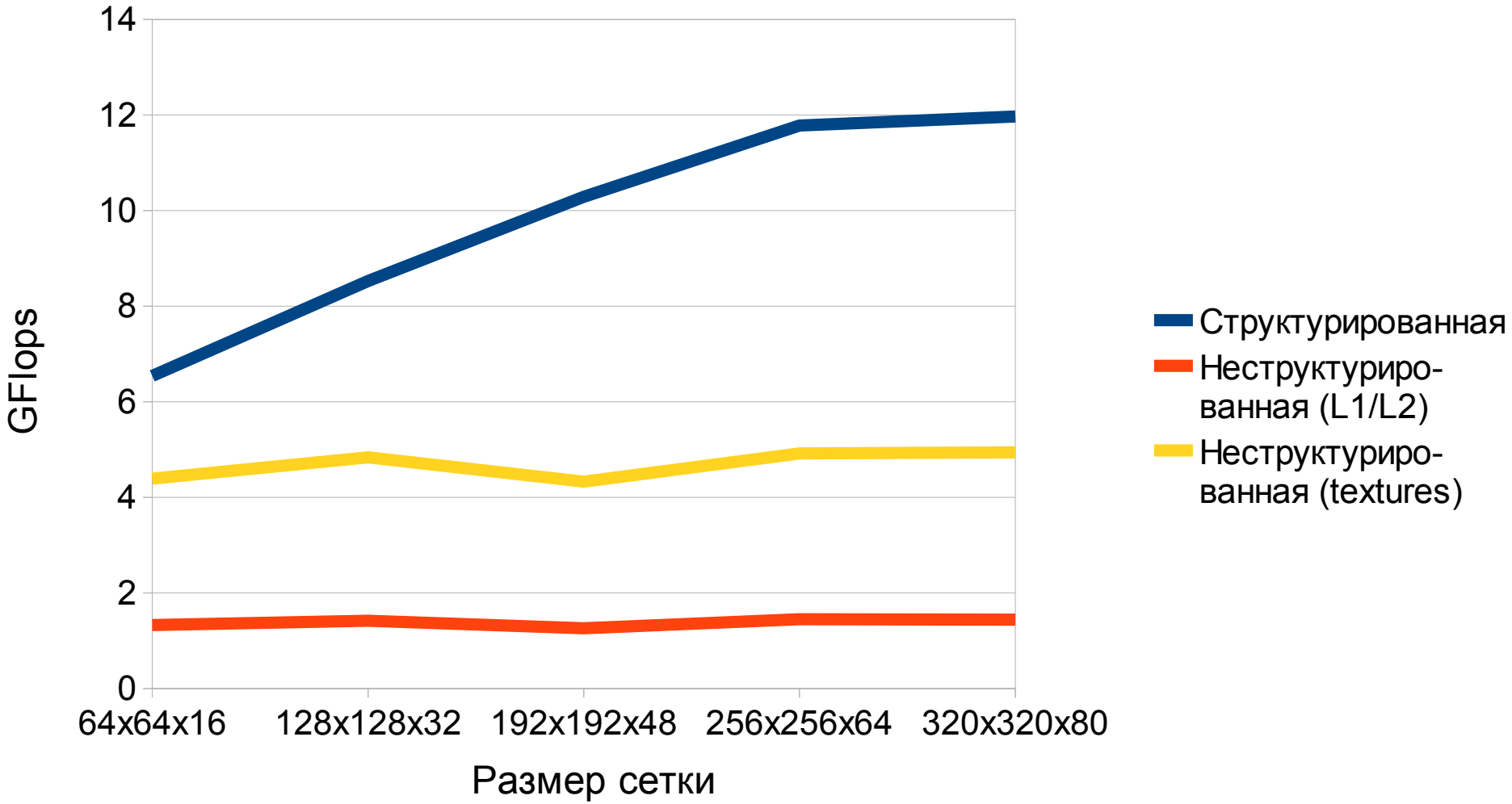
Все значения в «честных» GFlops'ax!

Сравнение форматов сеток

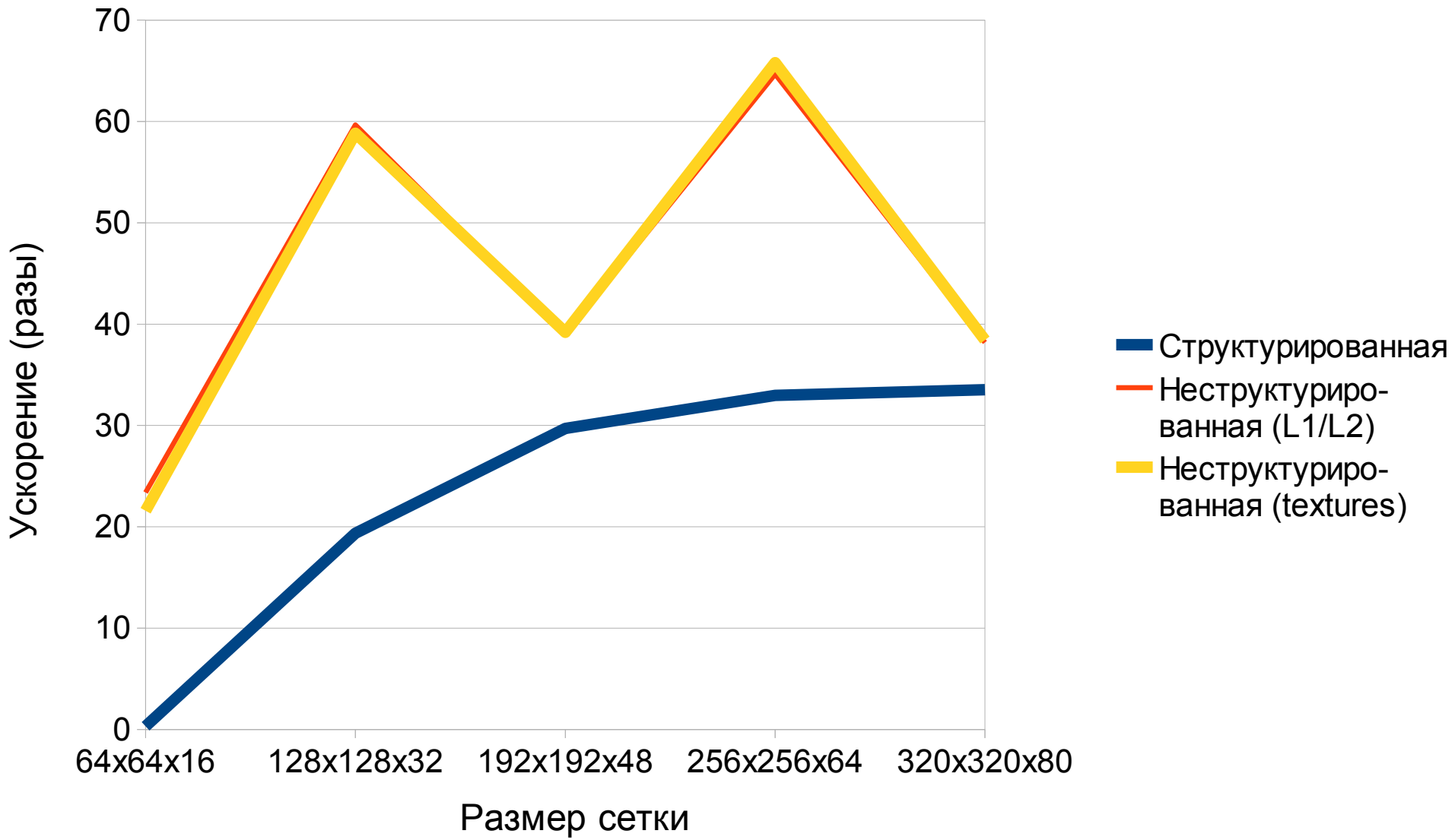


Сравнение форматов сеток

Устаревшая архитектура CUDA 1.3 (Tesla C1060, GeForce 2xx)



Ускорение относительно центрального процессора



Что будет быстрее?

«Медленные» вычисления на небольшой сетке!

Содержание

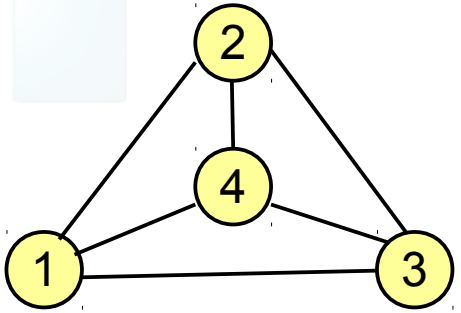
- Выбор сеток
- **Решатель #1**
- Решатель #2
- Выводы

Метод сопряженных невязок

- Оператор D: $D = A^* A$
- Условия:
- $\gamma_1 B \le A \le \gamma_2 B, \gamma_1 > 0,$
  $A = A^* > 0, B = B^* > 0$
- Итерационные параметры:
- $\tau_{k+1} = \frac{D\omega_k, z_k}{D\omega_k, \omega_k}$, k = 0, 1, …
- $\alpha_{k+1} = \left(1 - \frac{\tau_{k+1}}{\tau_k} \frac{(D\omega_k, z_k)}{D\omega_{k-1}, z_{k-1}} \frac{1}{\alpha_k}\right)^{-1},$
  k = 1, 2, …, $\alpha_1 = 1$

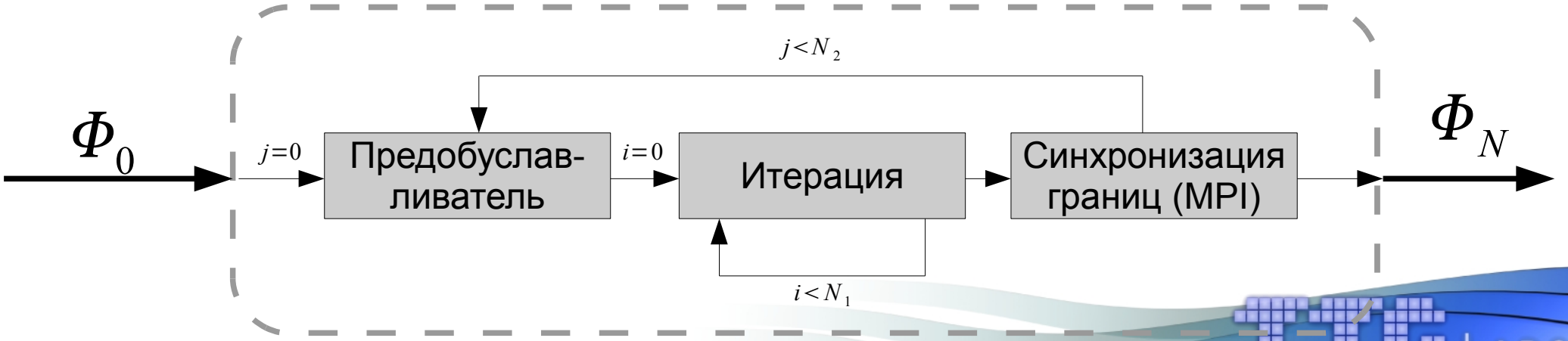
Исходная реализация

- Формат данных




```
double phi[4] = {0.0, 0.0, 0.0, 0.0};
int edge_start[6] = {1,2,3,4,4,4}
int edge_end[6] = {2,3,1,1,2,3}
```

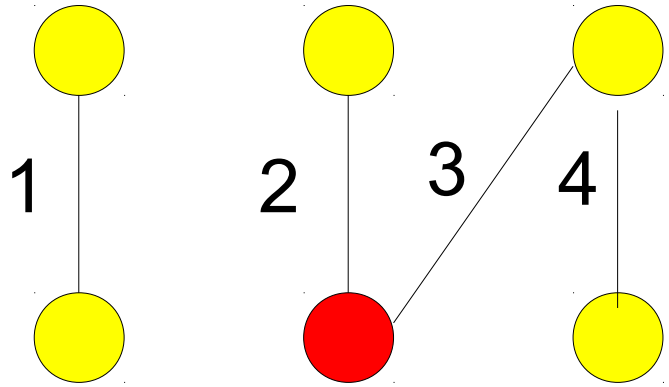
- Схема работы



Опыт портирования на CUDA

- Проблема

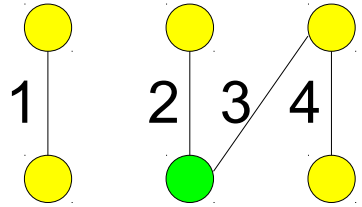
  Обход графа идёт по граням - при обращении к памяти будут data race'ы



Опыт портирования на CUDA

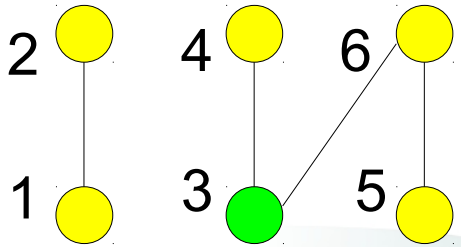
- Решение #1

  Использование атомарных операций

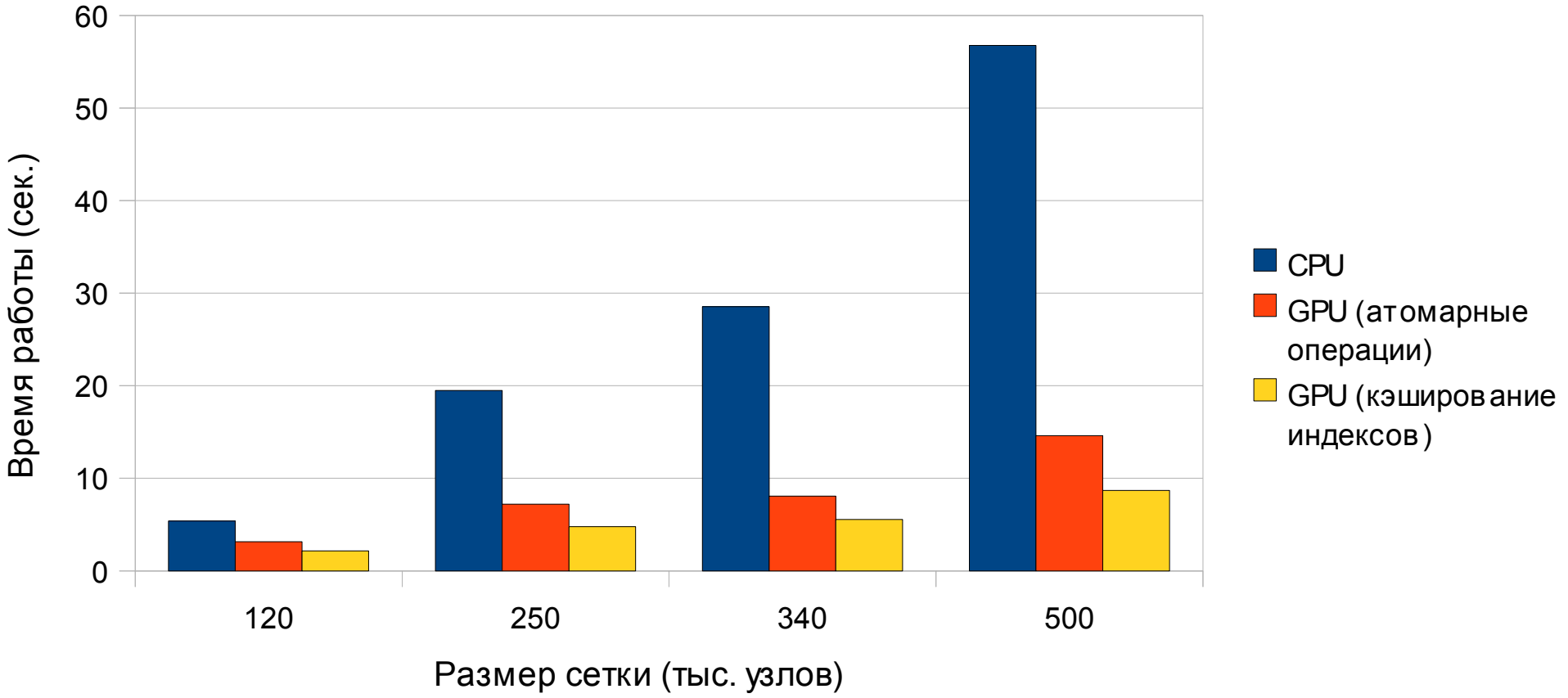


- Решение #2

  Использование вспомогательного массива с индексами и обход по вершинам

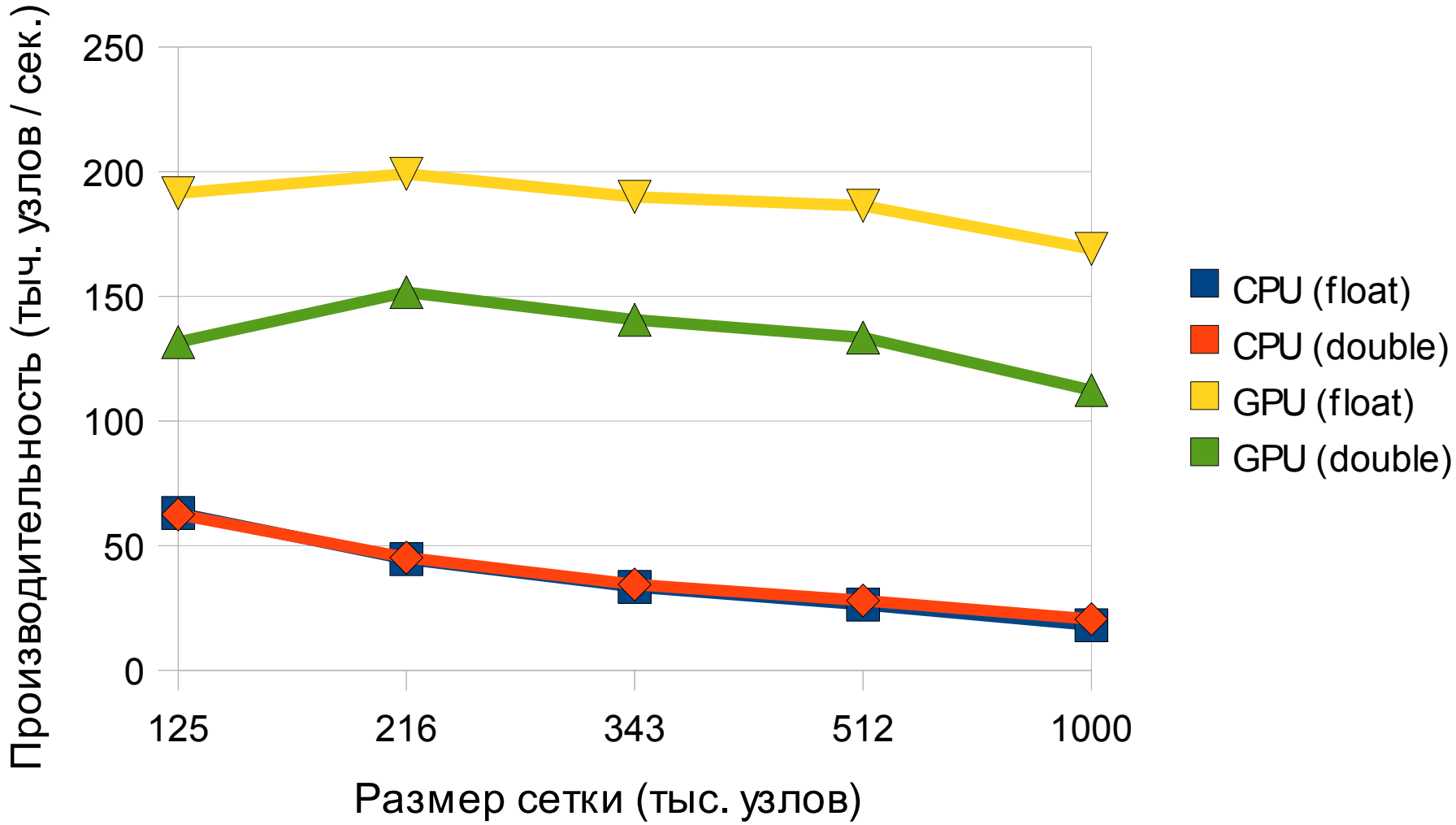


Тестирование предложенных решений

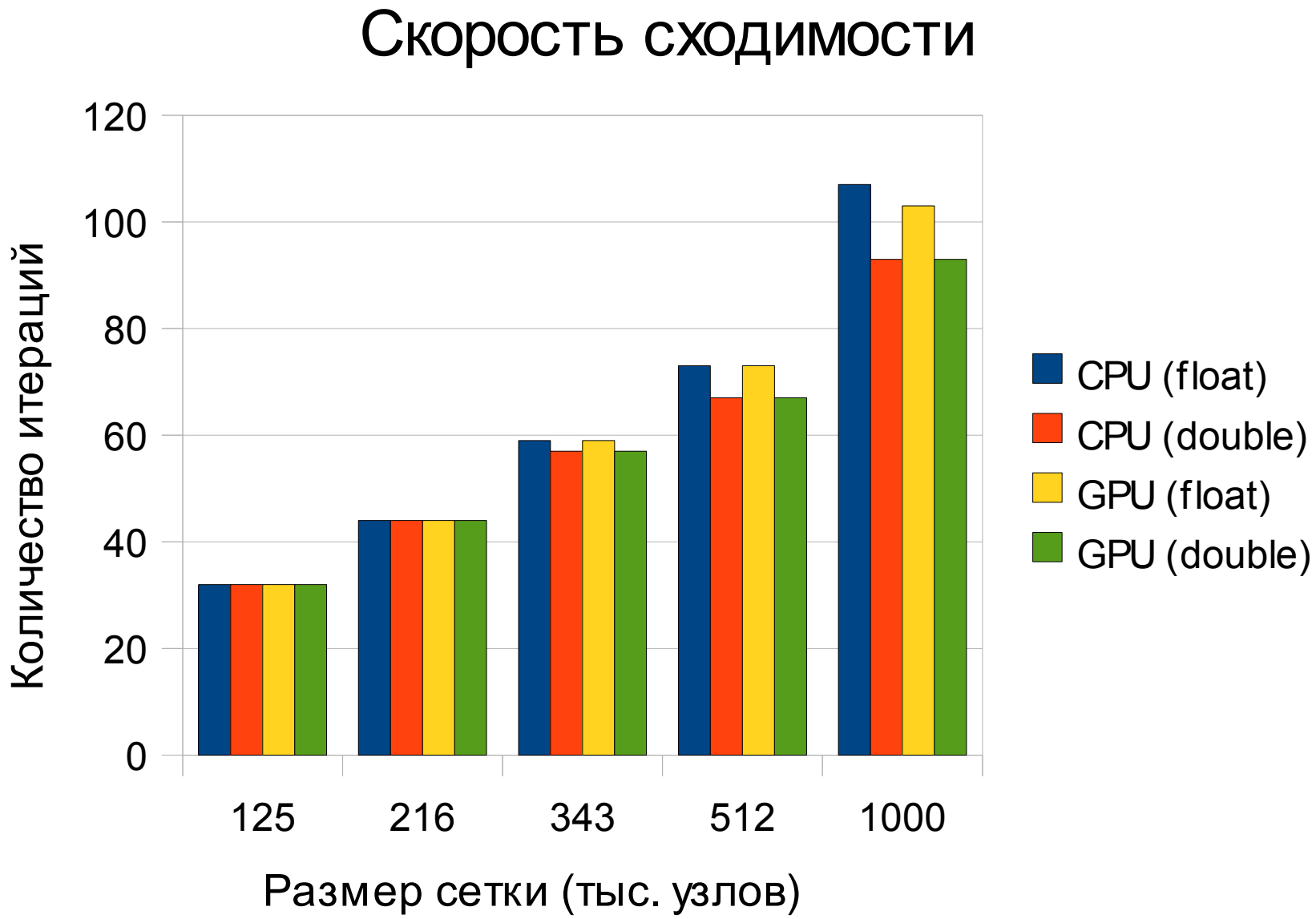


Вывод: лучше использовать дополнительные индексы

Тестирование (производительность)



Тестирование (скорость сходимости)



Содержание

- Выбор сеток
- Решатель #1
- **Решатель #2**
- Выводы

Метод DILU

- Разбиение:
- $A = D_A + L_A + U_A$
- Предобуславливатель:
- $M = (D + L_A)D^{-1}(D + U_A)$

- Пусть S = $\{(i,j): a_{ij} \neq 0\}$
- for i = 1, 2, …
- $set\ d_{ij} \leftarrow a_{ii}$
- for i = 1, 2, …
- $set\ d_{ii} \leftarrow \frac{1}{d_{ij}}$
- for j = i + 1, i + 2, …
- $if\ (i,j) \in S\ and\ (i,j) \in S\ then$
- $set\ d_{jj} \leftarrow d_{jj} - a_{ji}d_{ii}a_{ij}$

Опыт портирования на CUDA

- Проблема

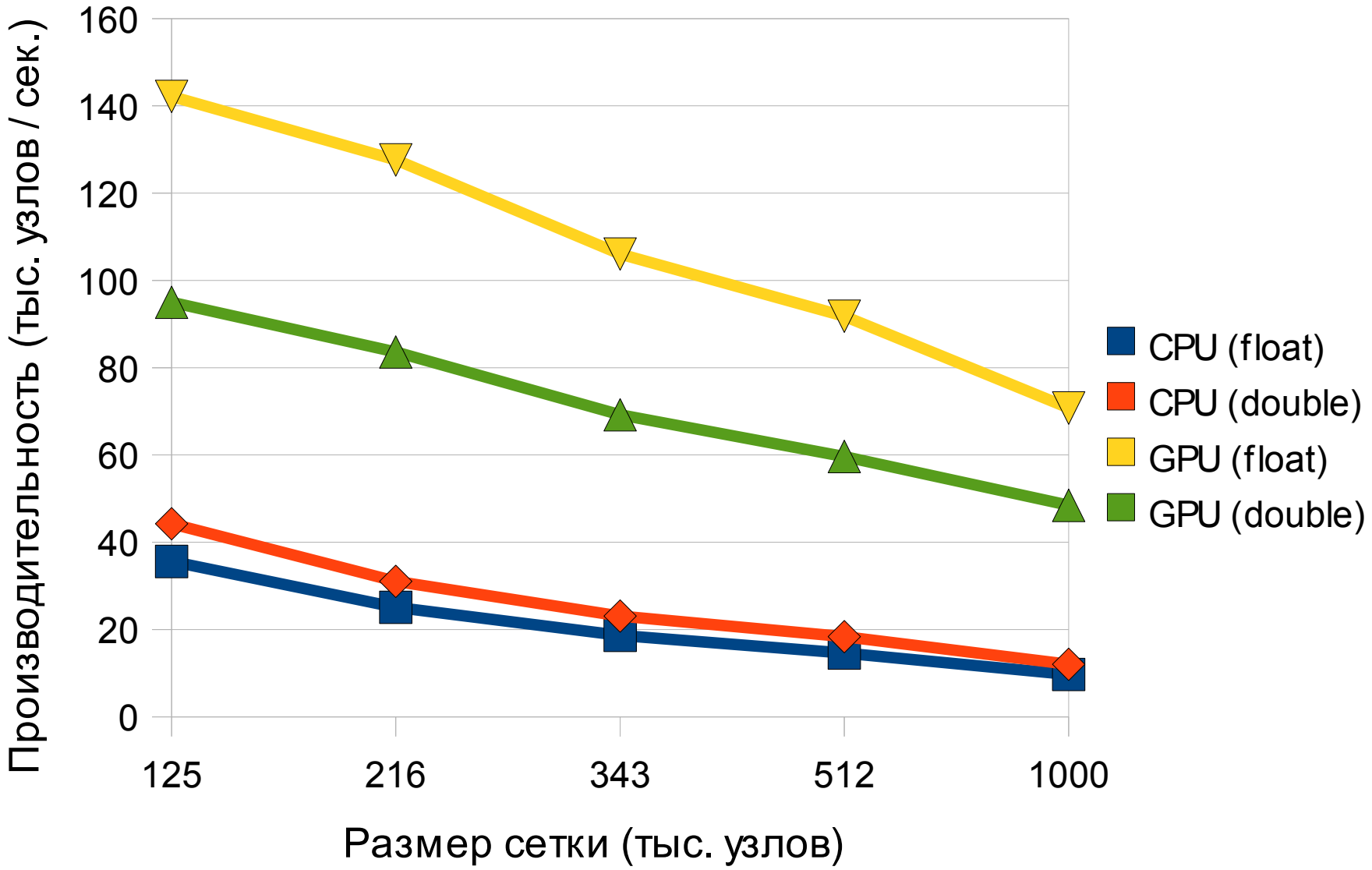
  Используется паттерн scan, который не может быть эффективно распараллелен на GPU

$$z_P = d_P^{-1}\left(f_P - (A\phi^k)_P - \sum_{S<P} L_{SP} z_S\right)$$

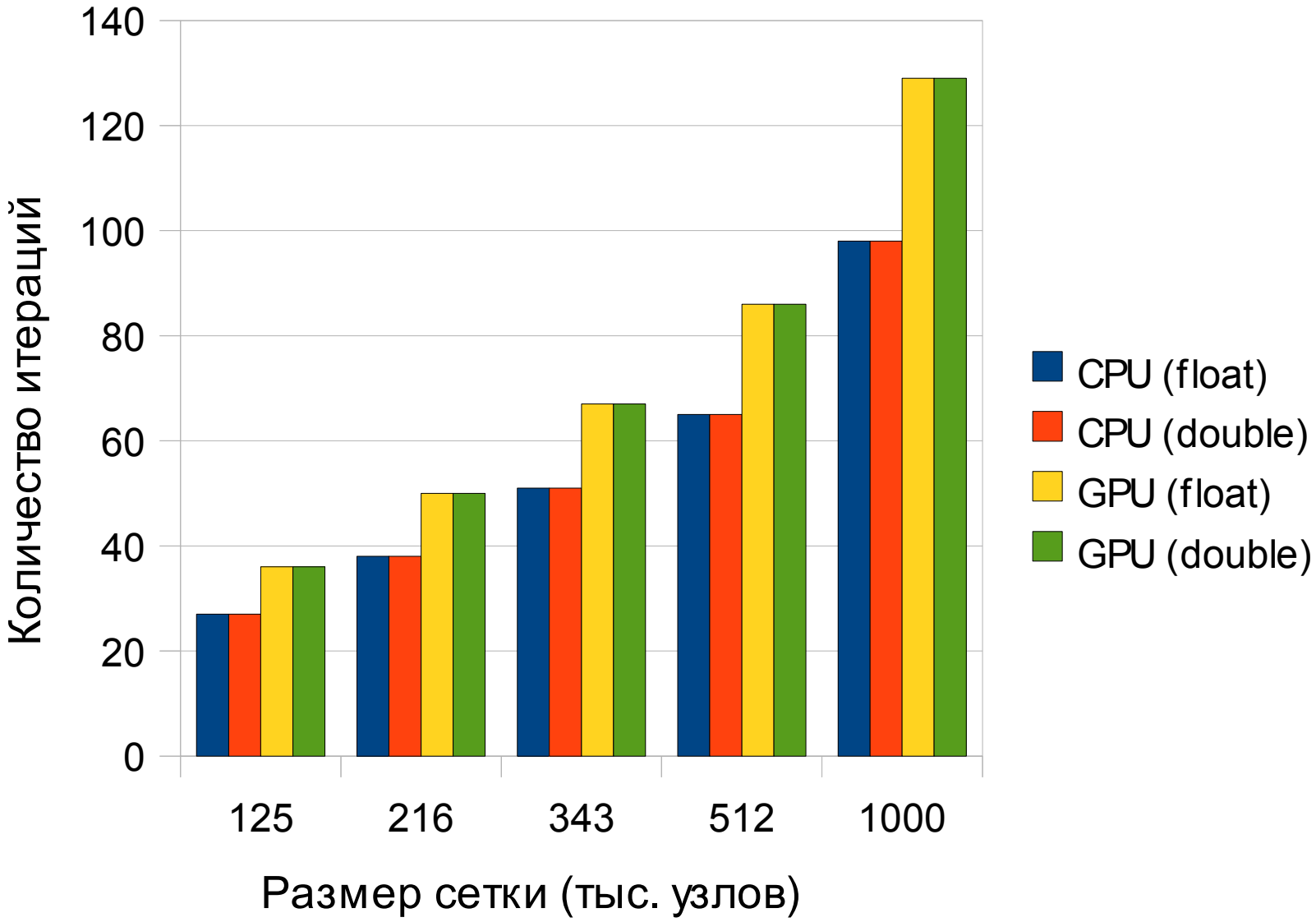
$$x_P = z_P - d_P^{-1} \sum_{S>P} U_{SP} x_S; \quad P = \overline{1, N}$$

$$\phi_P^{k+1} = \phi_P^{k+1} + x_P$$

Тестирование (производительность)



Тестирование (скорость сходимости)



Содержание

- Выбор сеток
- Решатель #1
- Решатель #2
- **Выводы**

Выводы

- Произведена оценка потери производительности при переходе от структурированных сеток к неструктурированным
- Рассмотрена эффективность применения ряда подходов, позволяющих повысить скорость обращений к памяти графического ускорителя при нерегулярном паттерне доступа
- Следующим этапом работ является
  - портирование логики подготовки данных
  - проведение дальнейшей оптимизации отдельных решателей
  - добавление поддержки систем с несколькими GPU.