Программируем видеокарты

введение в основные виды GPGPU оптимизаций

Лукин Михаил
Михайленко Кристина

SUDO

Оглавление

- Немного о нас и о наших задачах
- Введение в GPGPU и обобщённая архитектура GPU
- OpenCL: сходства и различия с языком C
- Стандартные способы оптимизации ядер
- Задача поиска пути
- Подробнее об *occupancy* и *coalesced memory access*
- Выводы и список литературы

Немного о нас и о наших задачах

- Немного о нас и о наших задачах
- Введение в GPGPU и обобщённая архитектура GPU
- OpenCL: сходства и различия с языком C
- Стандартные способы оптимизации ядер
- Задача поиска пути
- Подробнее об *occupancy* и *coalesced memory access*
- Выводы и список литературы

SUDO

SUDO – это IT компания, специализирующаяся в области высокопроизводительных вычислений.

- Приоритетное направлением деятельности – перенос вычислений на GPGPU.
- Имеем опыт реализации крупных, в том числе государственных заказов.

Эволюция решений вычислительных задач

Типичный путь популярной compute bound (и не только) задачи:

CPU → GPU → FPGA → ASIC

Примеры:

- видеокодеки;
- Bitcoin;
- нейронные сети;
- трассировка лучей.

Задача о прокладке коммуникаций: постановка задачи

- Построение трасс для прокладки коммуникаций:
  - Построение графа из карты местности ~80 000 км² (город + область);
  - Точность:
    - Город: 10 см.
    - Область 1 м.
  - Поиск путей на графе.
- Время построения пути: 1 секунда.

Задача о прокладке коммуникаций: скриншот

Проект ТУ к заявке N° **123-456-789АБВ**

Информация о ТУ | Варианты трасс

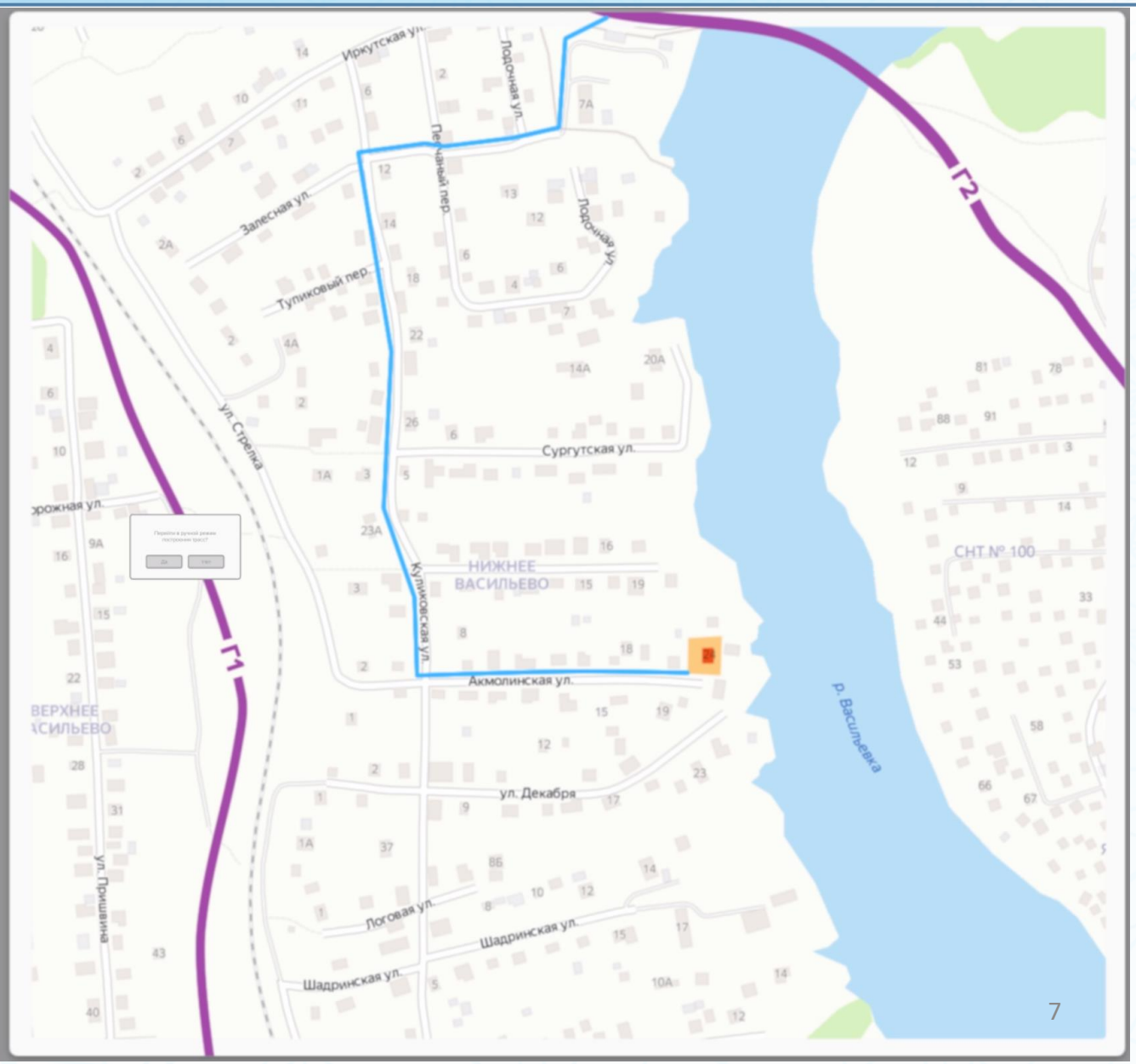
#	Название	ДУ	Давление, кПа	Материал	Длина, м	Точка врезки	Оценка
1	15-34	50	3	ПЭ	500	15-34а	454,000
2	15-34	50	3	ПЭ	950	ГРС-7	512,000
3	гм13	200	50	сталь	1220	гм13а	674`000
4	гм13-2	200	50	сталь	1350	гм13с	906`000
5	**тпг-4**	**200**	**3**	**сталь**	**1380**	**ГРС-1382**	**812`000**
6	88	50	50	ПЭ	2380	88-17.25	1`579`100

Дополнительные настройки

Состав работ (новое строительство)

#	Стоимость	Детали
1	599`000	Прокладка в траншее 2 сегмента 990 + 380 метров
2	213`000	Воздушный проход 1 сегмент 10 метров

Детали | Изменить трассу | Построить схему



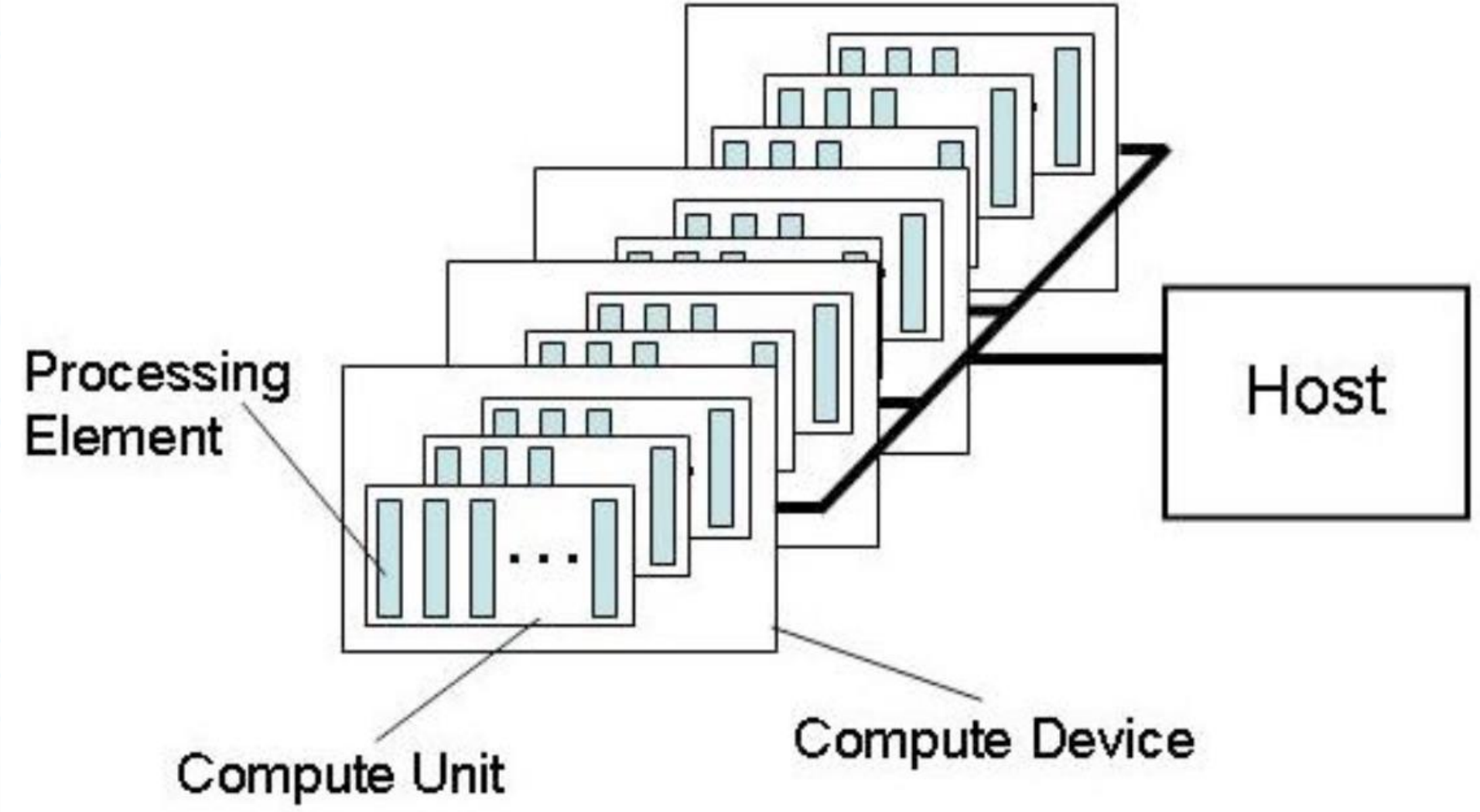
Введение в GPGPU и обобщённая архитектура GPU

- Немного о нас и о наших задачах
- **Введение в GPGPU и обобщённая архитектура GPU**
- OpenCL: сходства и различия с языком C
- Стандартные способы оптимизации ядер
- Задача поиска пути
- Подробнее об *occupancy* и *coalesced memory access*
- Выводы и список литературы

Технологии GPGPU

Технология GPGPU	Распространённость	Boilerplate code
OpenCL	Очень высокая	Средне
CUDA	Только Nvidia	Мало
Vulkan	Очень высокая	Очень много
Metal	Только Apple	Мало

Введение в OpenCL: платформа



*Khronos OpenCL Working Group. The OpenCL specification

Введение в OpenCL: примеры девайсов

NVIDIA GeForce RTX 3070

GA104	5888	184	96	8 GB	GDDR6	256 bit
GRAPHICS PROCESSOR	CORES	TMUS	ROPS	MEMORY SIZE	MEMORY TYPE	BUS WIDTH

Render Config	
Shading Units:	5888
TMUs:	184
ROPs:	96
SM Count:	46

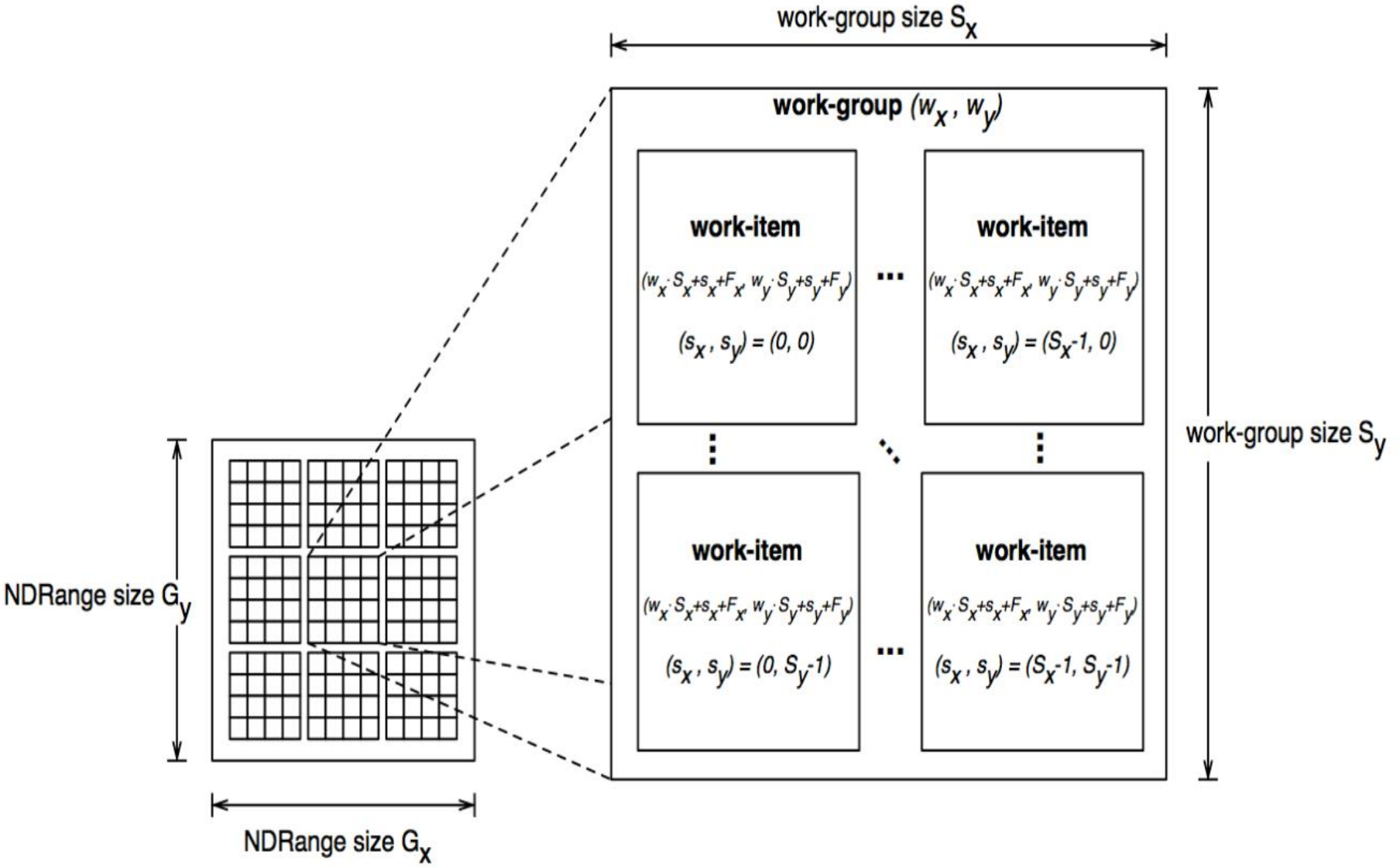
AMD Radeon RX 6800

Navi 21	3840	240	96	16 GB	GDDR6	256 bit
GRAPHICS PROCESSOR	CORES	TMUS	ROPS	MEMORY SIZE	MEMORY TYPE	BUS WIDTH

Render Config	
Shading Units:	3840
TMUs:	240
ROPs:	96
Compute Units:	60

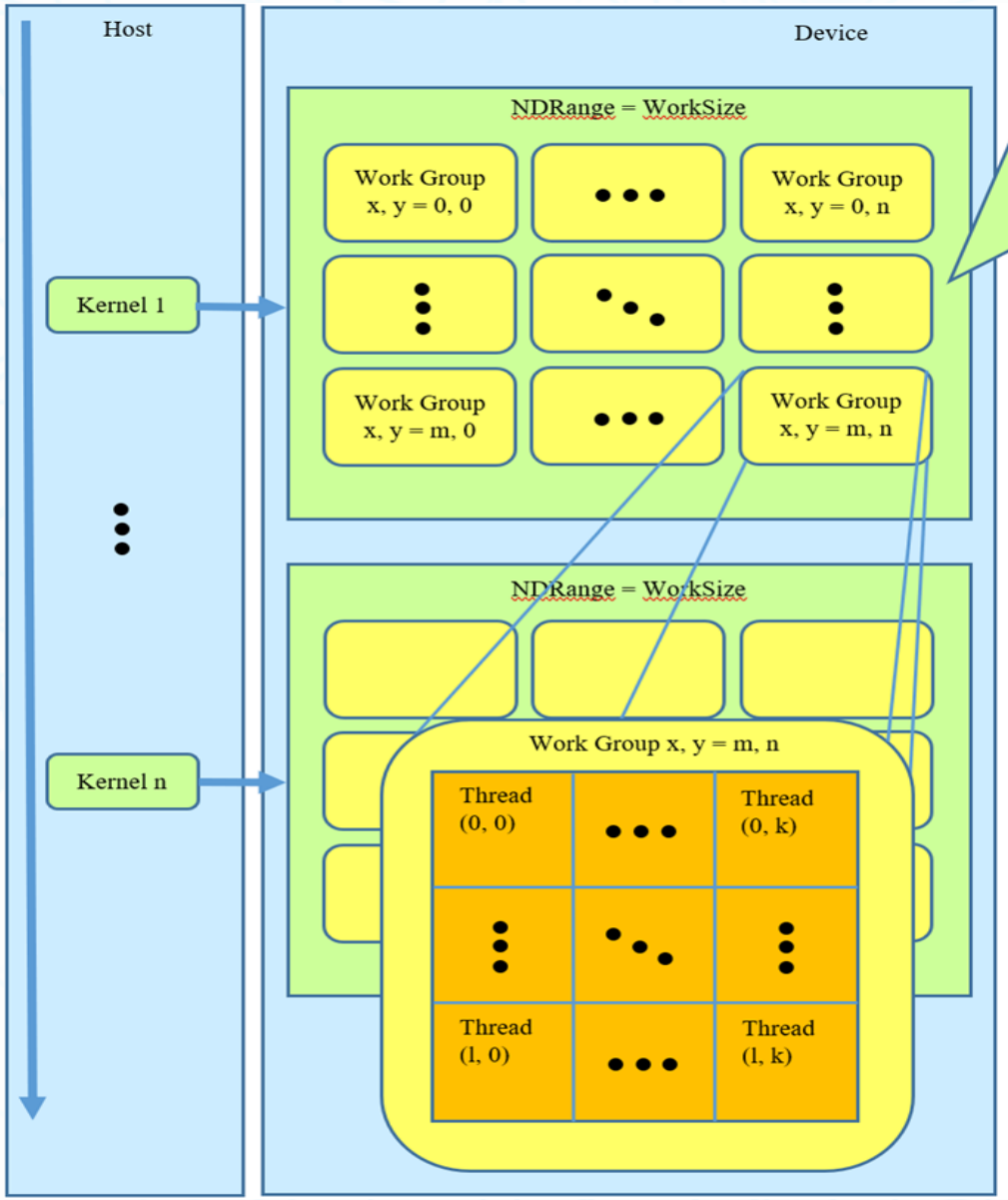
*https://www.techpowerup.com

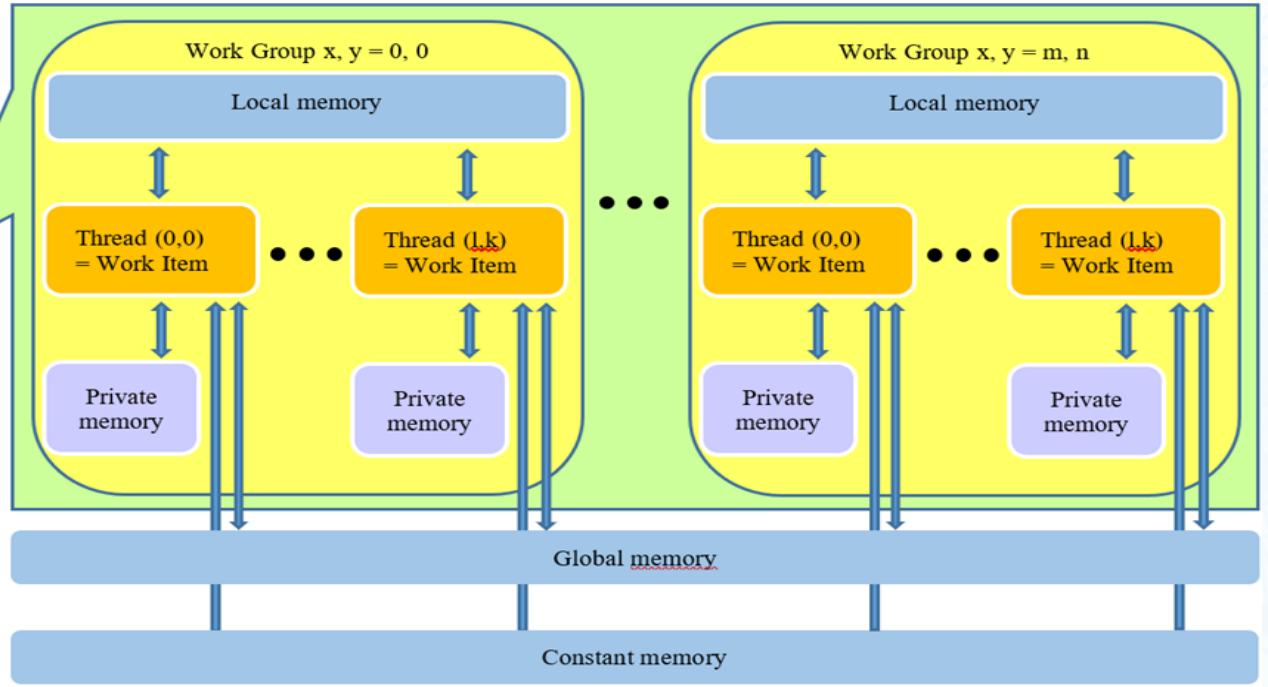
Введение в OpenCL: пространство потоков



*Khronos OpenCL Working Group. The OpenCL specification

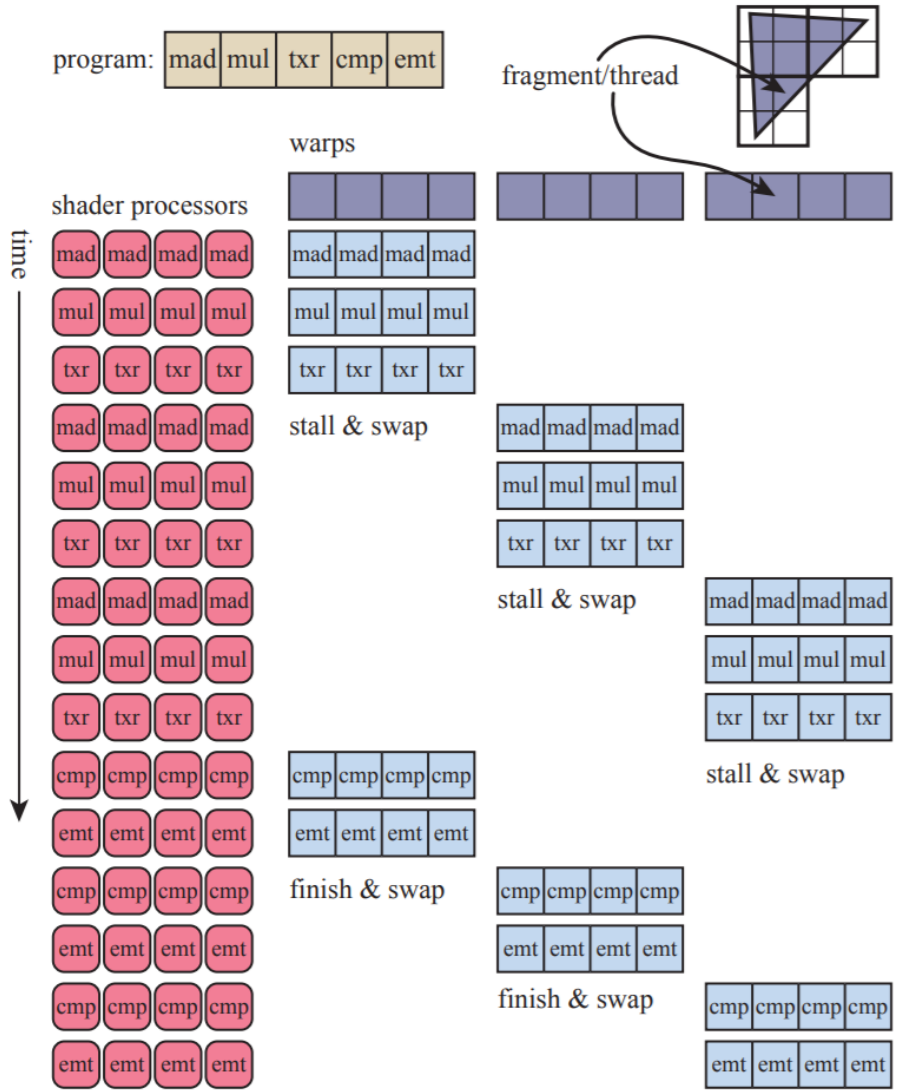
Введение в OpenCL: архитектура GPU





*https://doi.org/10.31799/1684-8853-2021-2-33-42

Введение в OpenCL: wavefront



*https://www.realtimerendering.com/

OpenCL: сходства и различия с языком C

- Немного о нас и о наших задачах
- Введение в GPGPU и обобщённая архитектура GPU
- **OpenCL: сходства и различия с языком C**
- Стандартные способы оптимизации ядер
- Задача поиска пути
- Подробнее об *occupancy* и *coalesced memory access*
- Выводы и список литературы

OpenCL – это диалект C

```
__kernel void add (
   __global float *res,
   __global float *a,
   __global float *b)
{
   int i = get_global_id(0);
   res[i] = a[i] + b[i];
}

void add (float *res, float *a, float *b, size_t n)
{
   for (size_t i = 0; i < n; i++)
   {
      res[i] = a[i] + b[i];
   }
}
```

Различия между OpenCL и C

Векторный литерал

```
uint4 u = (uint4)(1); // u == (1, 1, 1, 1)
uint3 v = (uint3)(2, u.xy); // v == (2, 1, 1)

float3 color = (0.1f, 0.1f, 0.5f);
// color == (0.5f, 0.5f, 0.5f) забыли векторный литерал, и у нас
оператор «запятая».
```

Векторный тип и булева логика

```
int b1 = true;  // b1 == 1

int2 b2 = true;  // b2 == (-1, -1)
```

https://habr.com/ru/post/345984/

Различия между OpenCL и C

Векторный литерал

```
uint4 u = (uint4)(1); // u == (1, 1, 1, 1)
uint3 v = (uint3)(2, u.xy); // v == (2, 1, 1)
```

~~float3 color = (0.1f, 0.1f, 0.5f);~~
// color == (0.5f, 0.5f, 0.5f) забыли векторный литерал, и у нас оператор «запятая».

```
float3 color = (float3)(0.1f, 0.1f, 0.5f); //правильно
```

Векторный тип и булева логика

```
int b1 = true;  // b1 == 1
int2 b2 = true; // b2 == (-1, -1)
```

https://habr.com/ru/post/345984/

Способы оптимизации ядер

- Немного о нас и о наших задачах
- Введение в GPGPU и обобщённая архитектура GPU
- OpenCL: сходства и различия с языком C
- **Стандартные способы оптимизации ядер**
- Задача поиска пути
- Подробнее об *occupancy* и *coalesced memory access*
- Выводы и список литературы

Стандартные способы оптимизации ядер

- Occupancy;
- Coalesced memory access;
- Банки памяти и strided index;
- Работа с кэшом данных;
- Кэш инструкций;
- Уменьшение дивергенции потоков.

Задача поиска пути

- Немного о нас и о наших задачах
- Введение в GPGPU и обобщённая архитектура GPU
- OpenCL: сходства и различия с языком C
- Стандартные способы оптимизации ядер
- **Задача поиска пути**
- Подробнее об *occupancy* и *coalesced memory access*
- Выводы и список литературы

Ещё раз формулировка задачи

- Построение трасс для прокладки коммуникаций:
  - Построение графа из карты местности ~80 000 км² (город + область);
  - Точность:
    - Город: 10 см.
    - Область 1 м.
  - Поиск путей на графе.
- Время построения пути: 1 секунда.

Поиск путей на графе

- Будем искать точные пути;
- CPU: алгоритм Дейкстры;
- GPU: алгоритм Беллмана-Форда.

Алгоритм Беллмана-Форда: реализация на CPU

```
void solve() {
 vector<double> d(n, INF);
 d[v] = 0f;
 for (;;) {
  bool any = false;
  for (int j = 0; j < m; ++j) {
   if (d[e[j].a] < INF) {
    if (d[e[j].b] > d[e[j].a] + e[j].cost) {
     d[e[j].b] = d[e[j].a] + e[j].cost;
     any = true;
    }
   }
  }
  if (!any) break;
 }
}
```

n – число вершин графа
m – число рёбер графа

*http://e-maxx.ru/algo/ford_bellman

Алгоритм Беллмана-Форда: реализация на OpenCL

```
__kernel void bellmanFordIter(
 uint nEdges,
 __global const uint2* restrict edges,
 __global const double* restrict weights,
 __global double* restrict d,
 __global uint* restrict changed )
{
 uint edgeId = get_global_id(0);
 if (edgeId >= nEdges) { return; }
 if (d[edges[edgeId].s0] < INF) {
  double relaxWeight = d[edges[edgeId].s0] + weights[edgeId];
  if (d[edges[edgeId].s1] > relaxWeight) {
   d[edges[edgeId].s1] = relaxWeight;
   *changed = 1;
  }
 }
}
```

Замеряем скорость на GPU без atomic

- AMD RX Vega 64: 1.39 с
- AMD Radeon VII: 0.69 с
- Nvidia RTX 3070: 0.71 с
- AMD RX 5700 XT: 1.63 с
- AMD RX 560: 4.32 с

Замеряем скорость на CPU

- Датасет https://snap.stanford.edu/data/roadNet-CA.html
- Bellman-Ford:
  - AMD Threadripper 2920x: 26.7 с;
  - AMD Ryzen 5 3600x: 20.4 с;
  - Intel i5 7400: 25.8 с.
- Dijkstra:
  - AMD Threadripper 2920x: 1.51 с;
  - AMD Ryzen 5 3600x: 1.37 с;
  - Intel i5 7400: 1.81 с.

Алгоритм Беллмана-Форда: добавляем атомарный минимум

```
__kernel void bellmanFordIter(
 uint nEdges,
 __global const uint2* restrict edges,
 __global const double* restrict weights,
 __global double* restrict d,
 __global uint* restrict changed )
{
 uint edgeId = get_global_id(0);
 if (edgeId >= nEdges) { return; }
 if (d[edges[edgeId].s0] < INF) {
  double relaxWeight = d[edges[edgeId].s0] + weights[edgeId];
  if (d[edges[edgeId].s1] > relaxWeight) {
   atomic_min_r64(&d[edges[edgeId].s1], relaxWeight);
   *changed = 1;
  }
 }
}
```

Замеряем скорость на GPU

- AMD RX Vega 64: 1.04 с;
- AMD Radeon VII: 0.6 с;
- Nvidia RTX 3070: 0.55 с;
- AMD RX 5700 XT: 1.33 с;
- AMD RX 560: 3.48 с.

Итого два ядра вместе

GPU	С atomic, с	Без atomic, с
AMD RX Vega	1.04	1.39
AMD Radeon VII	0.6	0.69
Nvidia RTX 3070	0.55	0.71
AMD RX 5700 XT	1.33	1.63
AMD RX 560	3.48	4.32

Почему так?

- Число итераций без atomic: ~900;
- Число итераций с atomic: ~700.

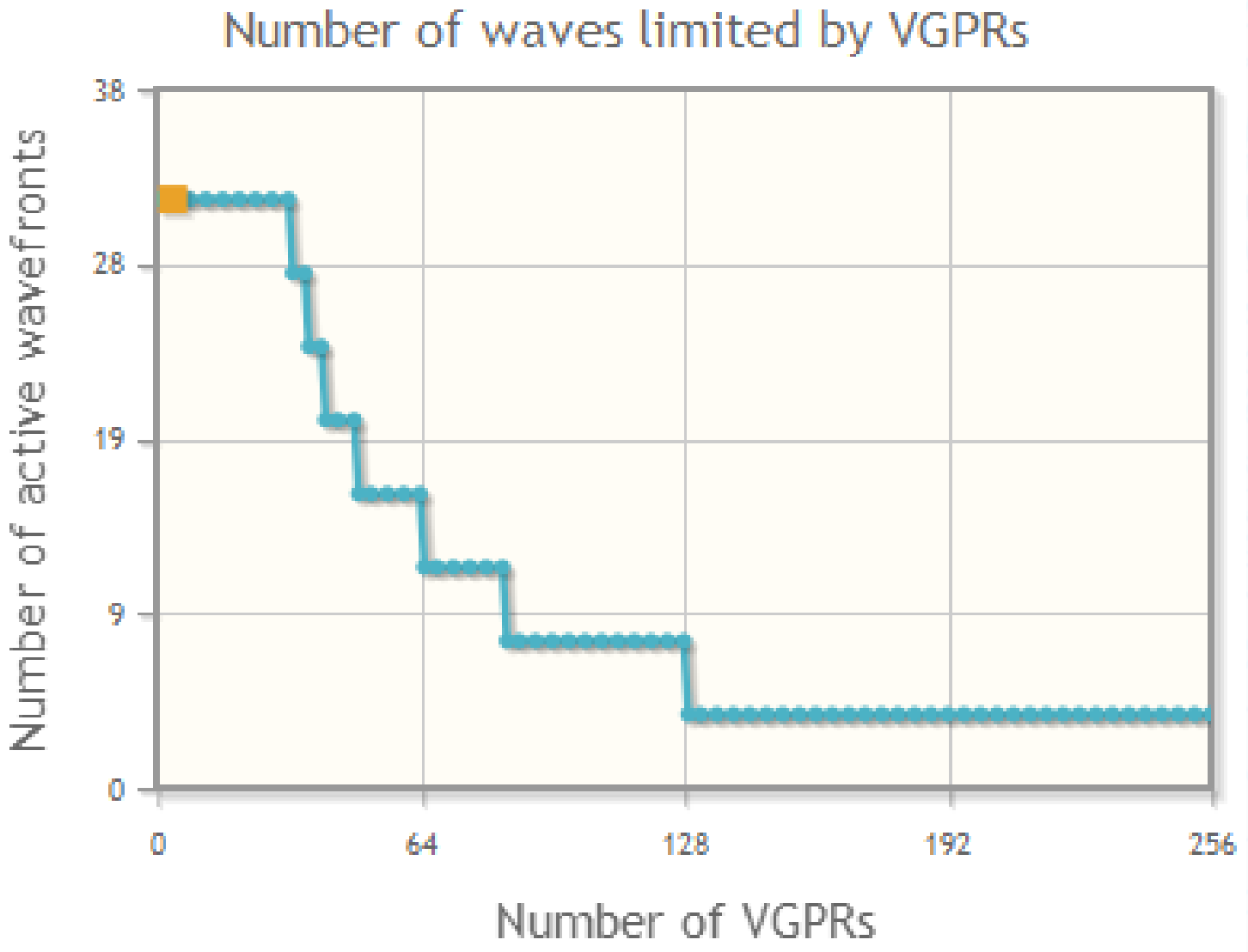
Попробуем ускорить

- Проверим occupancy;
- Попробуем увеличить coalesced memory access.

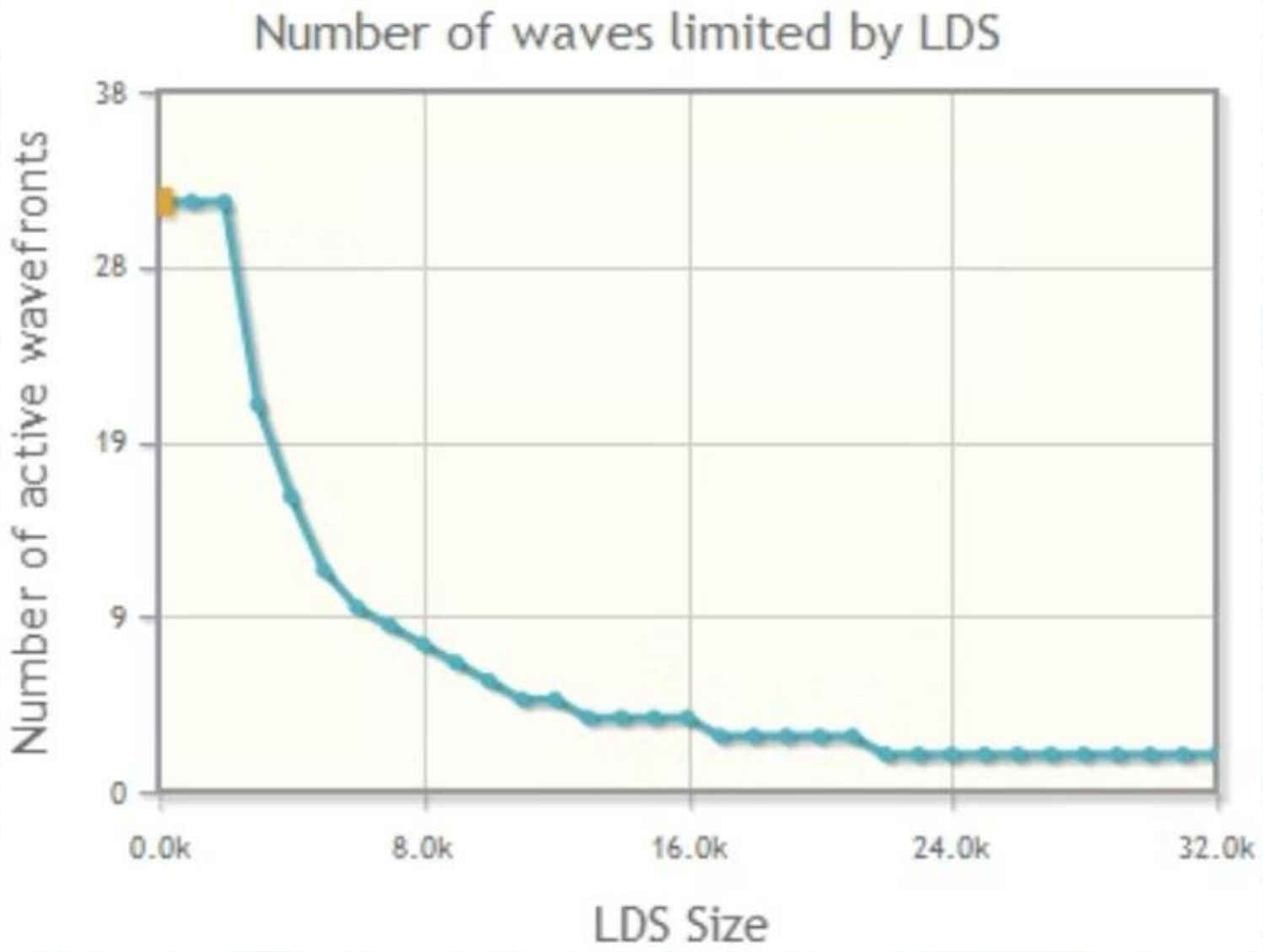
Подробнее об occupancy и coalesced memory access

- Немного о нас и о наших задачах
- Введение в GPGPU и обобщённая архитектура GPU
- OpenCL: сходства и различия с языком C
- Стандартные способы оптимизации ядер
- Задача поиска пути
- **Подробнее об *occupancy* и *coalesced memory access***
- Выводы и список литературы

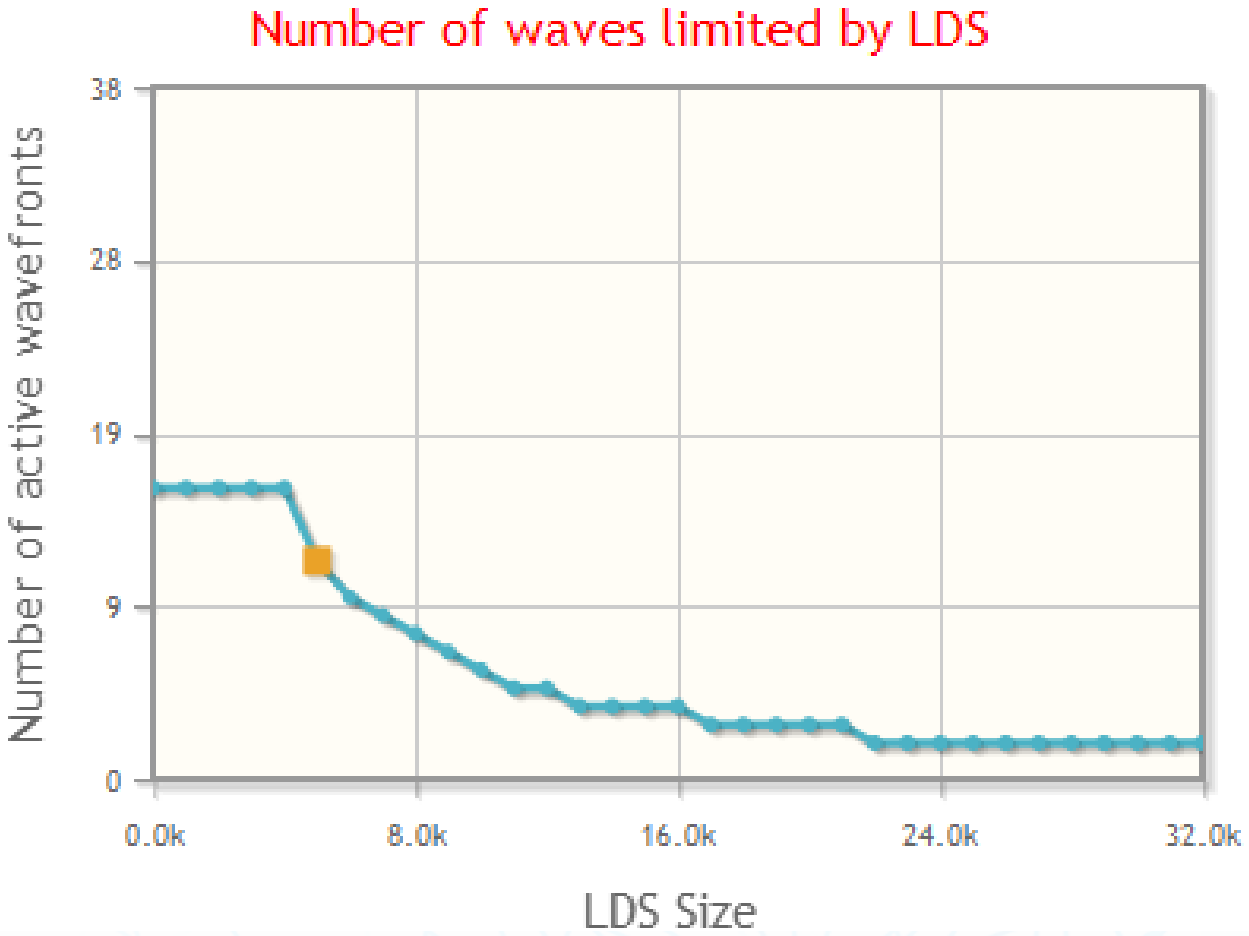
Occupancy

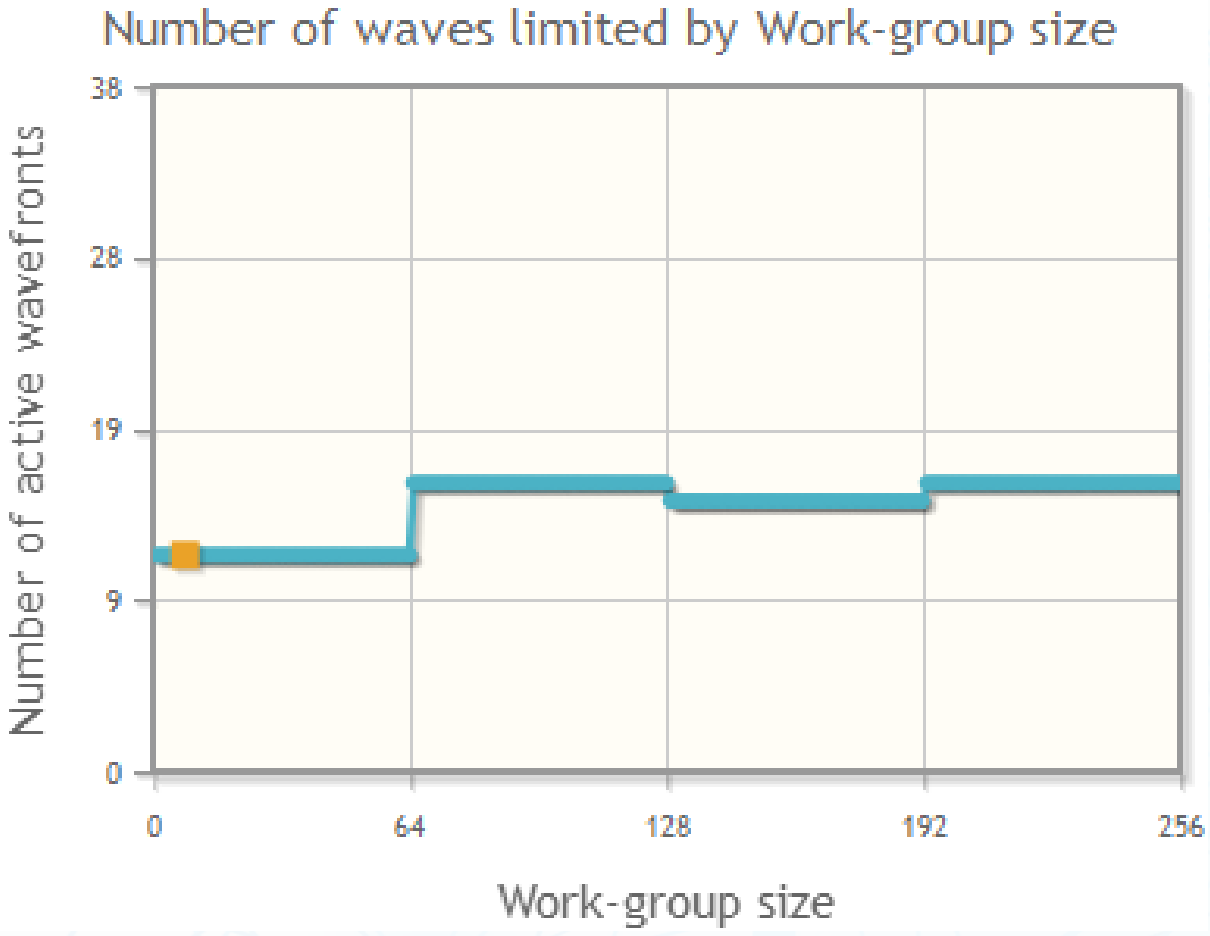

Number of waves limited by VGPRs
Number of active wavefronts
38
28
19
9
0
0
64
128
192
256
Number of VGPRs

Occupancy


Number of waves limited by LDS
Number of active wavefronts
38
28
19
9
0
0.0k
8.0k
16.0k
24.0k
32.0k
LDS Size

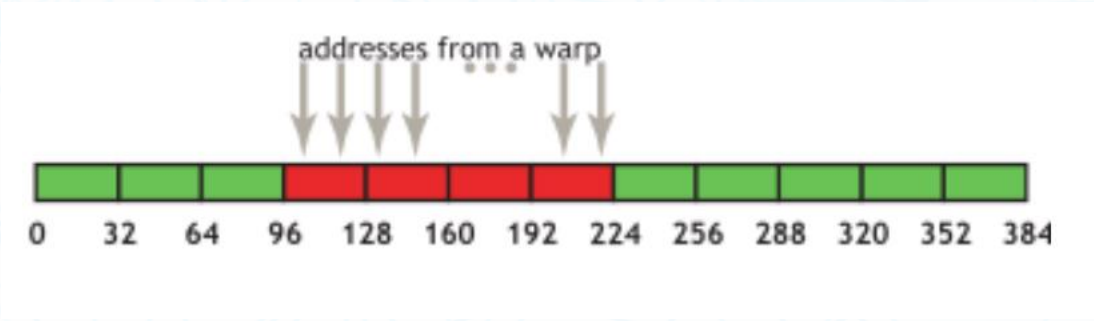
Occupancy


Number of waves limited by LDS
Number of active wavefronts
38
28
19
9
0
0.0k
8.0k
16.0k
24.0k
32.0k
LDS Size


Number of waves limited by Work-group size
Number of active wavefronts
38
28
19
9
0
0
64
128
192
256
Work-group size

Coalesced memory access

- Контроллер памяти на GPU достаёт данные из глобальной памяти большими кусками.
- Он умеет группировать запросы потоков из wavefront, чтобы выдать за один запрос сразу всем.



* https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html#coalesced-access-to-global-memory

Алгоритм Беллмана-Форда: реализация на OpenCL

```
__kernel void bellmanFordIter(
 uint nEdges,
 __global const uint2* restrict edges,
 __global const double* restrict weights,
 __global double* restrict d,
 __global uint* restrict changed )
{
 uint edgeId = get_global_id(0);
 if (edgeId >= nEdges) { return; }
 if (d[edges[edgeId].s0] < INF) {
  double relaxWeight = d[edges[edgeId].s0] + weights[edgeId];
  if (d[edges[edgeId].s1] > relaxWeight) {
   atomic_min_r64(&d[edges[edgeId].s1], relaxWeight);
   *changed = 1;
  }
 }
}
```

Попробуем сортировку по началу ребра

GPU	С atomic, с	Без atomic, с
AMD RX Vega	0.99	2.04
AMD Radeon VII	0.9	1.07
Nvidia RTX 3070	0.48	0.48
AMD RX 5700 XT	1.25	2.65
AMD RX 560	3.37	8.05

Попробуем сортировку по концу ребра

GPU	С atomic, с	Без atomic, с
AMD RX Vega	0.84	2.54
AMD Radeon VII	0.52	1.37
Nvidia RTX 3070	0.51	0.6
AMD RX 5700 XT	1.43	3.38
AMD RX 560	3.83	10.53

Алгоритм Дейкстры

CPU\Сортировка	Без сортировки	По началу ребра	По концу ребра
Threadripper	1.51	1.32	2.06
Ryzen 5 3600x	1.37	1.15	1.84
Intel i5 7400	1.81	1.48	2.53

Сводная таблица

GPU/CPU	Без сортировки, с	Сортировка начало, с	Сортировка конец, с
AMD RX Vega	1.04	0.99	0.84
AMD Radeon VII	0.6	0.52	0.9
Nvidia RTX 3070	0.55	0.48	0.51
AMD RX 5700 XT	1.33	1.25	1.43
AMD RX 560	3.48	3.37	3.83
Threadripper 2920x	1.51	1.32	2.06
Ryzen 5 3600x	1.37	1.15	1.84
Intel i5 7400	1.81	1.48	2.53

Список литературы

- Немного о нас и о наших задачах
- Введение в GPGPU и обобщённая архитектура GPU
- OpenCL: сходства и различия с языком C
- Стандартные способы оптимизации ядер
- Задача поиска пути
- Подробнее об *occupancy* и *coalesced memory access*
- **Выводы и список литературы**

Выводы

- Можно ускорить на GPU такие задачи, которые обычно решаются классическими однопоточными алгоритмами;
- Продемонстрированы способы оптимизации ядер на примере алгоритма на графе.

Список литературы

- Документация по OpenCL https://www.khronos.org/registry/OpenCL/
- Что ещё необходимо узнать про OpenCL C перед тем, как на нём писать https://habr.com/ru/post/345984/
- Курс лекций «Вычисления на видеокартах» https://compscicenter.ru/courses/video_cards_computation/
- Документация по CUDA https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html
- Введение в CUDA https://developer.nvidia.com/blog/even-easier-introduction-cuda/
- Задача поиска пути https://github.com/sudo-team-company/cppconf-path-finder

Что делать, если производительности всё равно не хватает

- Компилятор OpenCL не умеет подбирать оптимальное число регистров.
- Для локальных улучшений можно использовать ассемблерные вставки и интринсики (AMD недавно научились, Nvidia давно).
- Для глобальных улучшений можно полностью использовать ассемблер:
  - Пример 1: https://github.com/ROCmSoftwarePlatform/MIOpen
  - Пример 2: https://realhet.wordpress.com/gcn-asm-groestl-coin-kernel/

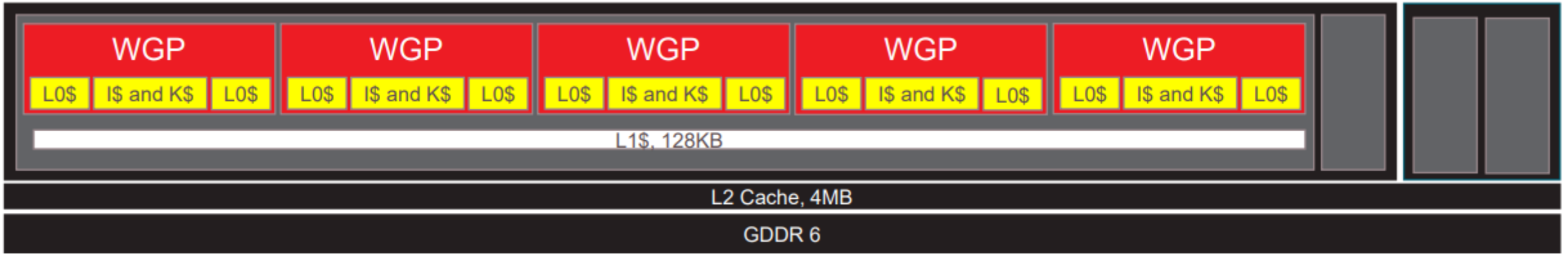
Регистры и scratch-регистры

- Если не хватает регистров, они выделяются в глобальной памяти (и обычно попадают в кэш);
- Также массив попадает в scratch-регистры, если его индексы нельзя вычислить в compile-time;
- Их наличие показывает профайлер (CodeXL, RGP, Nsight).

Архитектура GPU: кэш

Radeon™ RX Vega 64

Radeon™ RX 5700 XT



*https://gpuopen.com/wp-content/uploads/slides/GPUOpen_Let'sBuild2020_Optimizing%20for%20the%20Radeon%20RDNA%20Architecture.pdf

Memory bound задачи

- Часто бывает, что увеличение occupancy приводит к снижению производительности;
- Не забывать про coalesced memory access;
- Иногда бывает, что замедление куска кода ускоряет задачу в целом.

Банки памяти и strided index

- Локальная память состоит из банков.
- Если запросы из разных потоков попадают в один и тот же банк, то они сериализуются.
- Профайлер умеет показывать число конфликтов банков.
- Для разрешения конфликтов добавляют «дырки» в данные, в результате чего паттерн доступа к потокам меняется.

Instruction cache

- Размер ядра лучше делать таким, чтобы оно полностью влезало в кэш инструкций;
- Накладные расходы на разделение ядра на несколько могут быть околонулевыми.