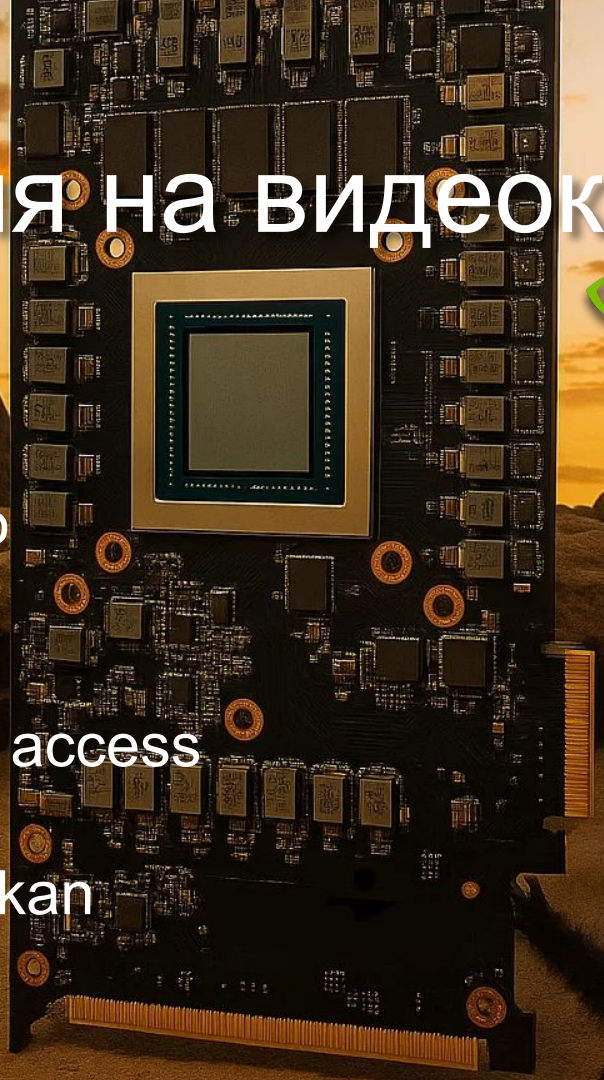
CS Space
Клуб технологий и науки

Вычисления на видеокартах



Vulkan
OpenCL™
NVIDIA
CUDA

Лекция 2

- Архитектура GPU
- Модель массового параллелизма
- code divergence
- coalesced memory access
- Код кернела: OpenCL CUDA Vulkan

polarnick239@gmail.com
Николай Полярный

План лекции

Большая часть из лекции “CS Space: Видеокарты: что они могут?”

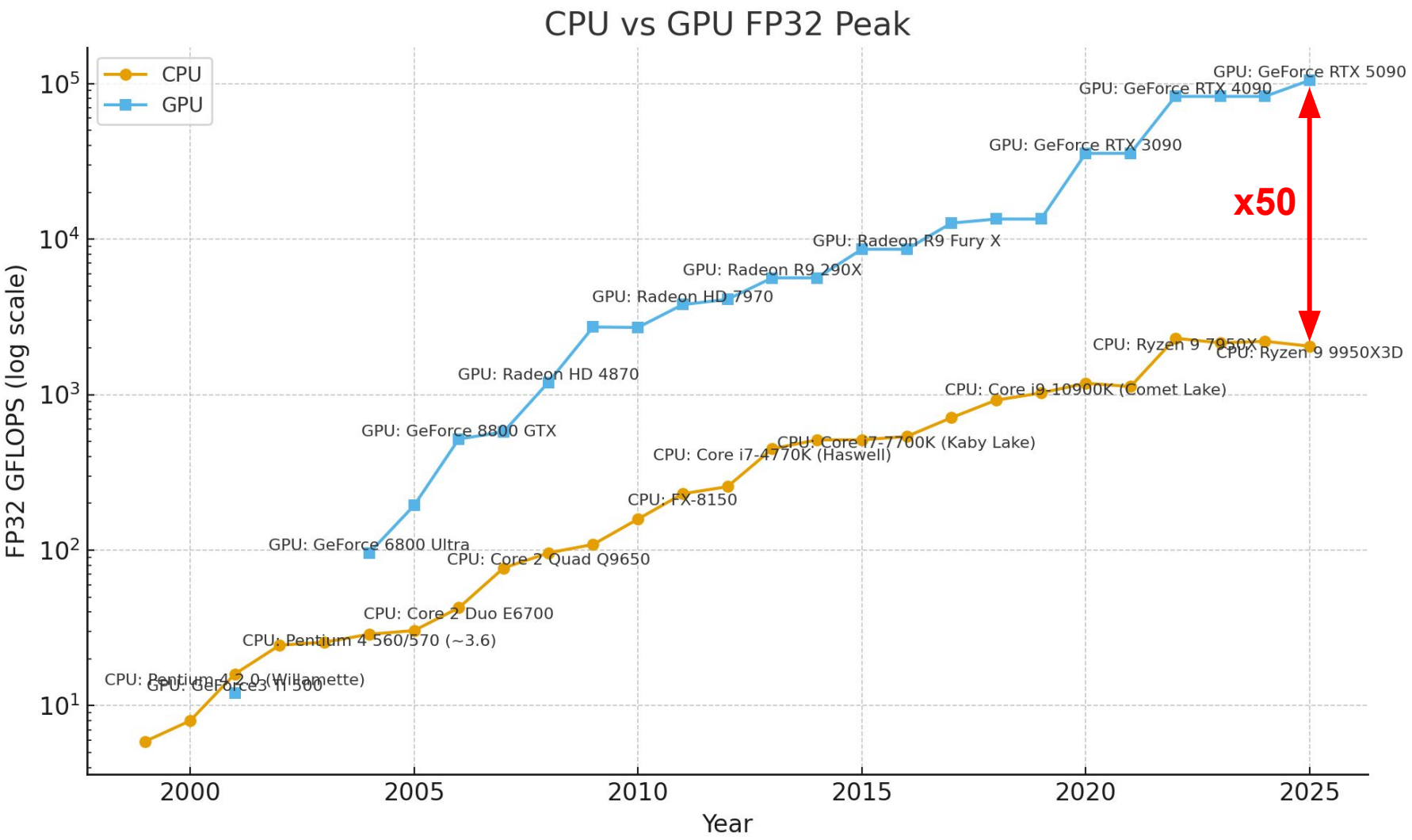
1) **Вычисления**:
   shared instruction pointer, code divergence
2) **Работа с памятью**:
   hyper-threading, latency hiding, occupancy, registers pressure/spilling, coalesced memory access pattern, cache lines, local/shared memory
3) **Общая картина ЭВМ-архитектуры**: CPU - RAM - PCI-E - VRAM - GPU
4) **Модель вычислений массового параллелизма**
5) **Профилирование и оптимизация:** NVIDIA Nsight, SoL анализ

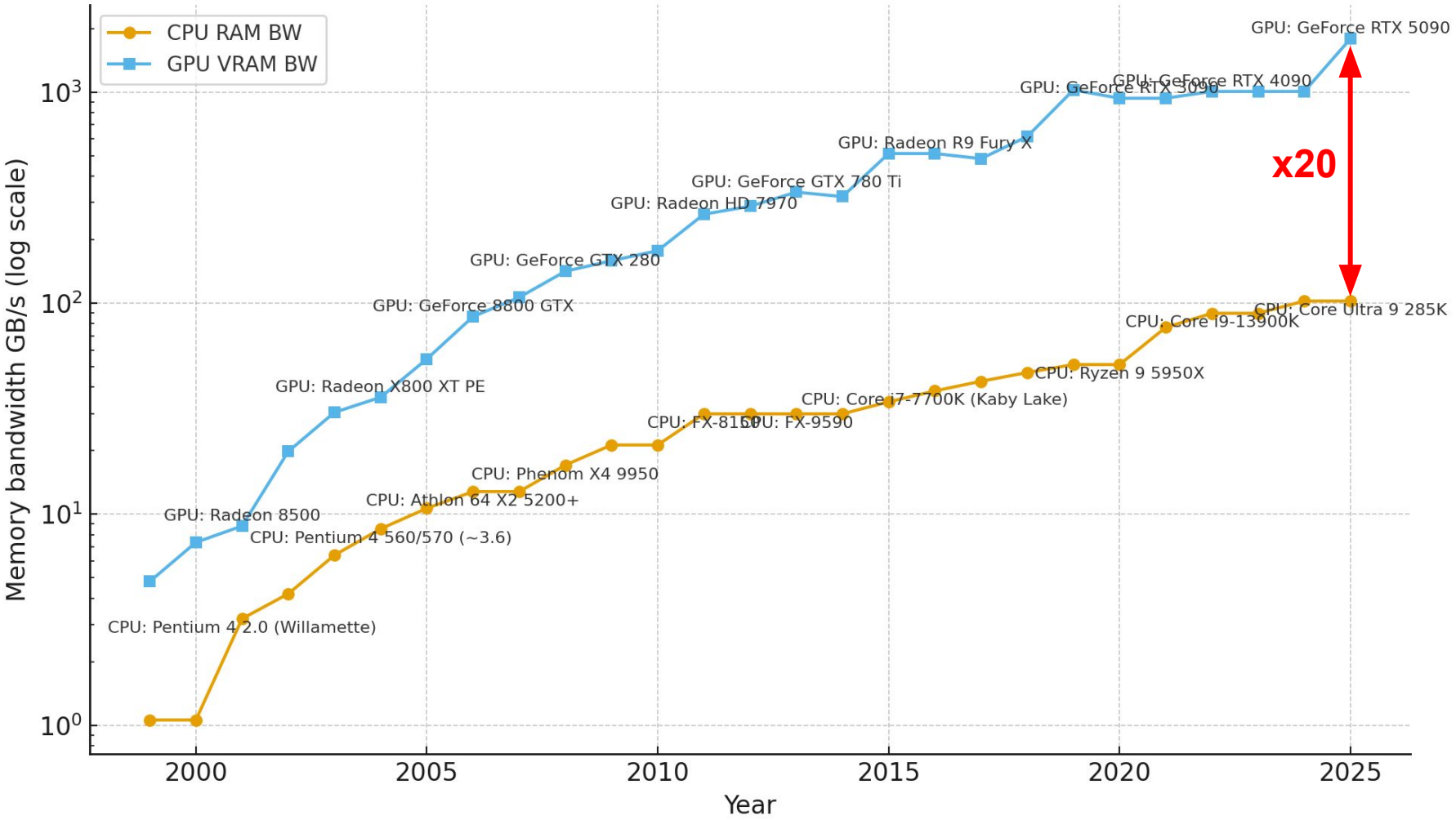
И обсудим как выглядит код на видеокарте:

6) **Пример кода:** A+B на C++ (OpenMP), OpenCL, CUDA, Vulkan (GLSL)

Глава 1: Вычисления

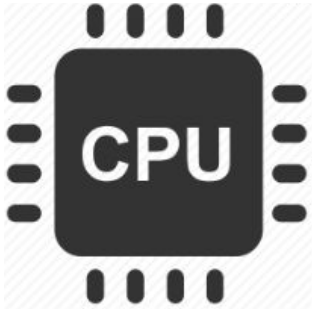
shared instruction pointer, code divergence


CPU vs GPU FP32 Peak
CPU
GPU
FP32 GFLOPS (log scale)
Year
10^1
10^2
10^3
10^4
10^5
2000
2005
2010
2015
2020
2025
GPU: GeForce RTX 5090
GPU: GeForce RTX 4090
GPU: GeForce RTX 3090
GPU: Radeon R9 Fury X
GPU: Radeon R9 290X
GPU: Radeon HD 7970
GPU: Radeon HD 4870
GPU: GeForce 8800 GTX
GPU: GeForce 6800 Ultra
GPU: GeForce3 Ti 500
x50
CPU: Ryzen 9 9950X
CPU: Ryzen 9 9950X3D
CPU: Core i9-10900K (Comet Lake)
CPU: Core i7-7700K (Kaby Lake)
CPU: Core i7-4770K (Haswell)
CPU: FX-8150
CPU: Core 2 Quad Q9650
CPU: Core 2 Duo E6700
CPU: Pentium 4 560/570 (~3.6)
CPU: Pentium 4 2.0 (Willamette)


CPU vs GPU Memory Bandwidth
CPU RAM BW
GPU VRAM BW
Memory bandwidth GB/s (log scale)
Year
10^0
10^1
10^2
10^3
2000
2005
2010
2015
2020
2025
GPU: GeForce RTX 5090
GPU: GeForce RTX 4090
GPU: GeForce RTX 3090
GPU: Radeon R9 Fury X
GPU: GeForce GTX 780 Ti
GPU: Radeon HD 7970
GPU: GeForce GTX 280
GPU: GeForce 8800 GTX
GPU: Radeon X800 XT PE
GPU: Radeon 8500
CPU: Core Ultra 9 285K
CPU: Core i9-13900K
CPU: Ryzen 9 5950X
CPU: Core i7-7700K (Kaby Lake)
CPU: FX-9590
CPU: FX-8150
CPU: Phenom X4 9950
CPU: Athlon 64 X2 5200+
CPU: Pentium 4 560/570 (~3.6)
CPU: Pentium 4 2.0 (Willamette)
x20

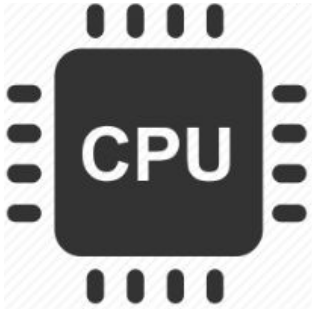
Архитектура

$100 \cdot 10^9$ FLOPS (**F**loating-point operations **p**er **s**econd)

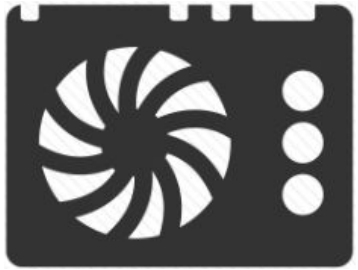


Архитектура

$100 \cdot 10^9$ FLOPS (**F**loating-point operations **p**er **s**econd)



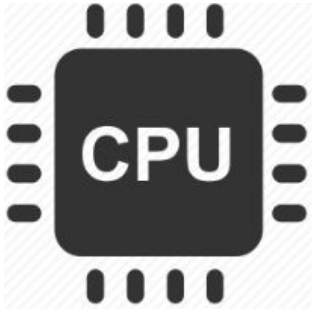
$100 \cdot 10^{12}$ FLOPS (**x1000 раз больше**)



GPU

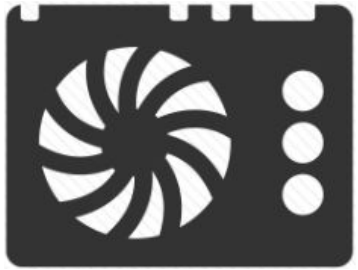
Архитектура

$100 \cdot 10^9$ FLOPS



40 GB/s memory bandwidth

$100 \cdot 10^{12}$ FLOPS (**x1000 раз больше**)

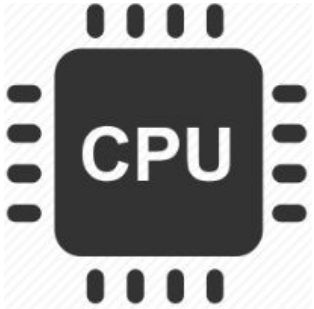


1000 GB/s (**x25 раз больше**)

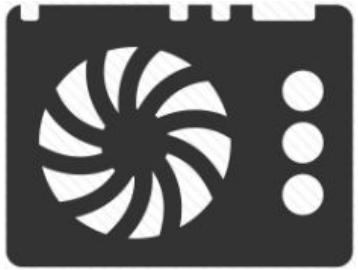
GPU

Архитектура

$100 \cdot 10^9$ FLOPS



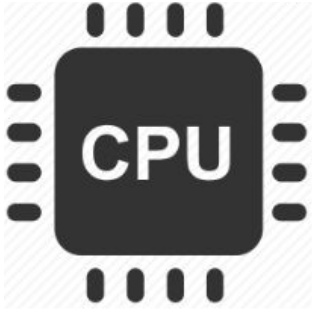
$100 \cdot 10^{12}$ FLOPS



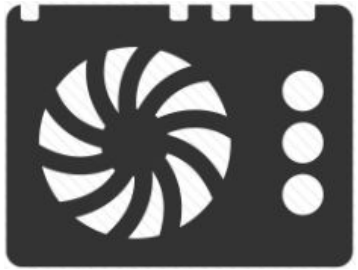
GPU

Архитектура

$100 \cdot 10^9$ FLOPS



$100 \cdot 10^{12}$ FLOPS

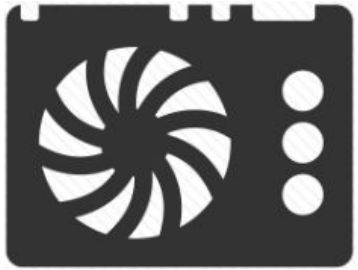


GPU

Архитектура

Как уместить ТЫСЯЧИ ядер?
На CPU ведь это почему-то невозможно!

$100 \cdot 10^{12}$ FLOPS



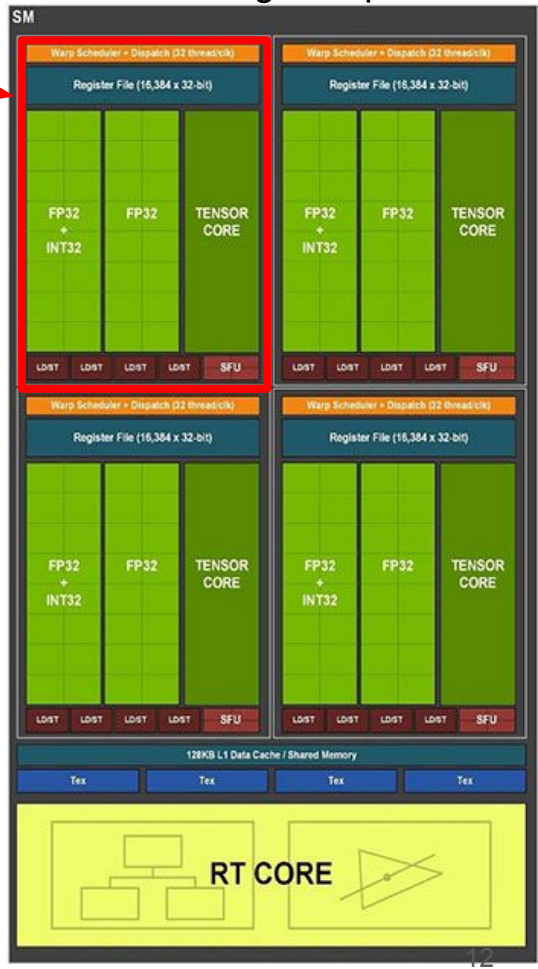
GPU

Архитектура

warp

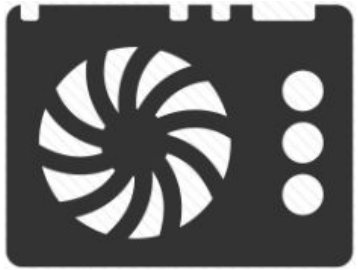
32 слабых медленных
CUDA ядер

SM - Streaming Multiprocessor



Как уместить ТЫСЯЧИ ядер?
На CPU ведь это почему-то невозможно!

$100 \cdot 10^{12}$ FLOPS



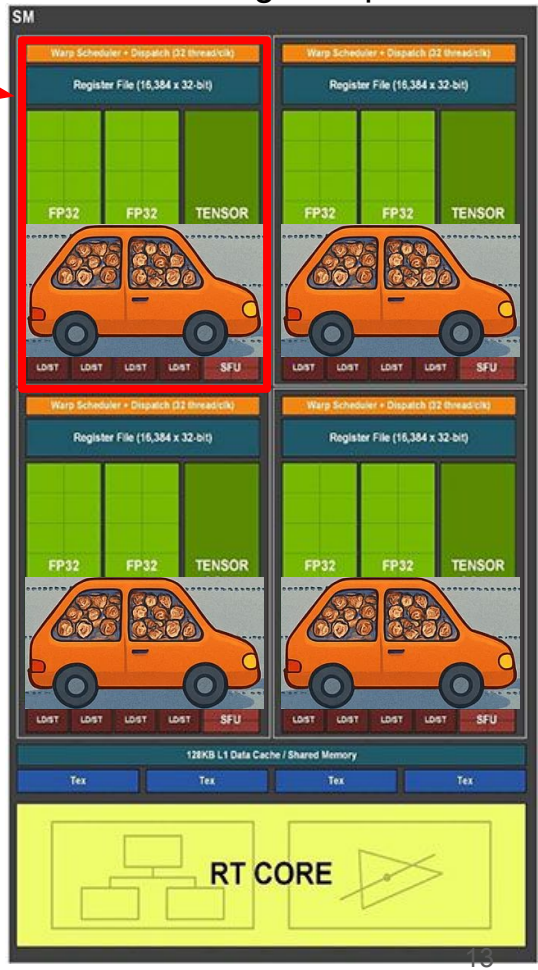
GPU

Архитектура

warp

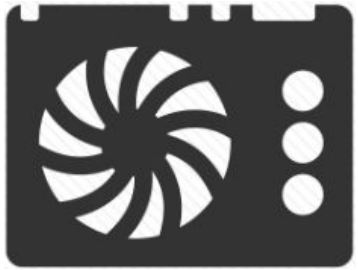
32 слабых медленных
CUDA ядер

SM - Streaming Multiprocessor



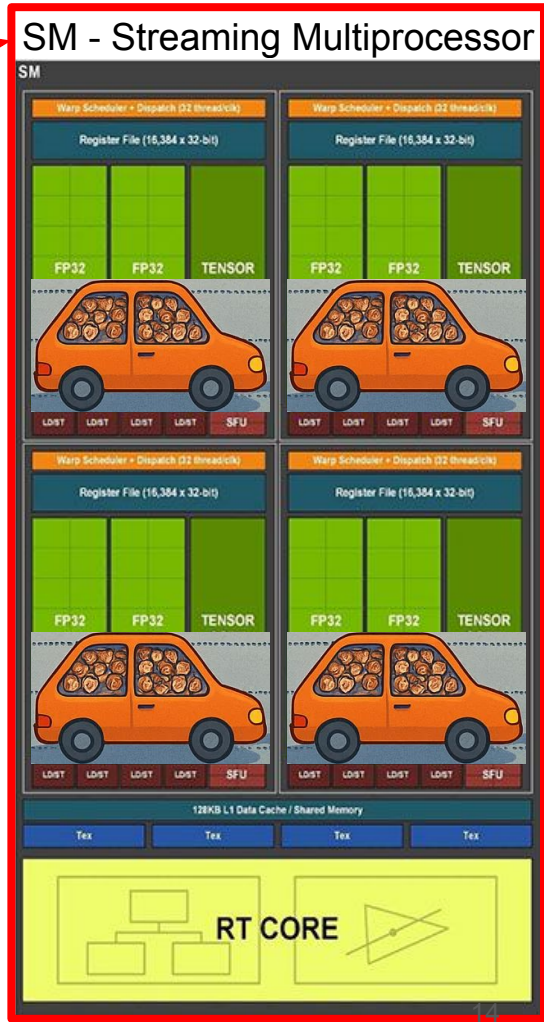
Как уместить ТЫСЯЧИ ядер?
На CPU ведь это почему-то невозможно!

$100 \cdot 10^{12}$ FLOPS



GPU

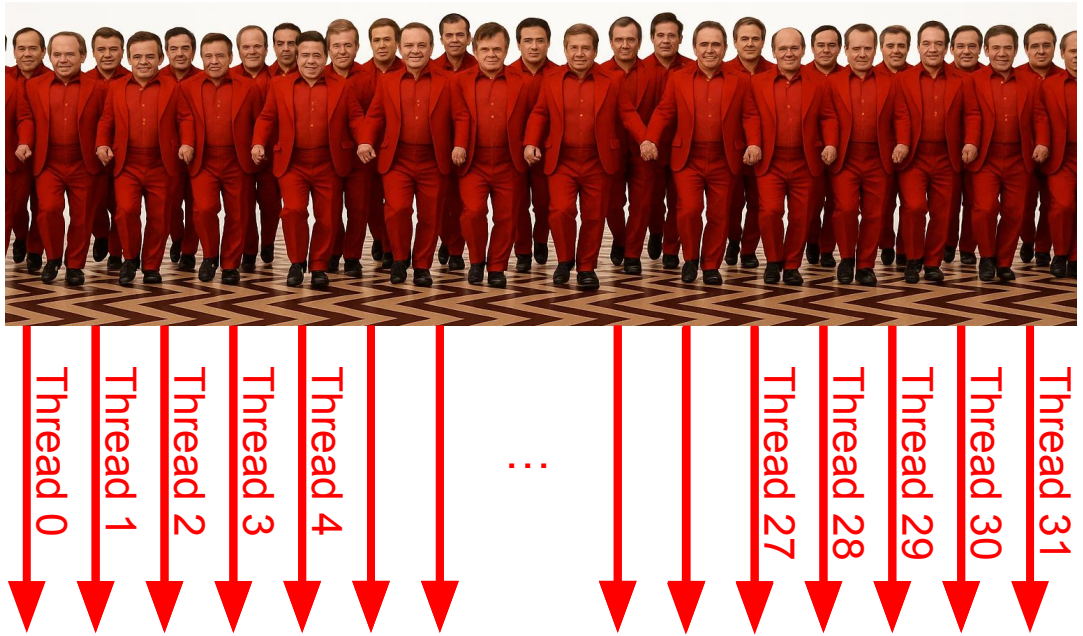
RTX 3090: **10496 CUDA cores** = 82 SM · 4 warps · 32 ALUs



https://www.techpowerup.com/review/nvidia-geforce-ampere-architecture-board-design-gaming-tech-software/3.html

CPU ядро

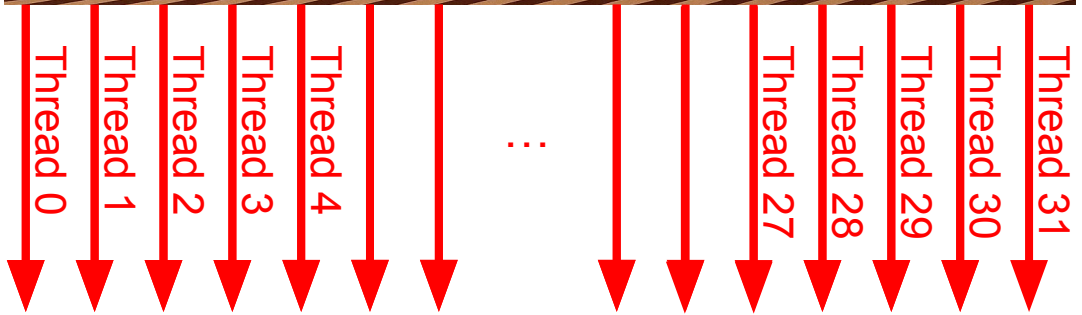
GPU warp



Как уместить ТЫСЯЧИ ядер?
На CPU ведь это почему-то невозможно!

Архитектура

GPU warp

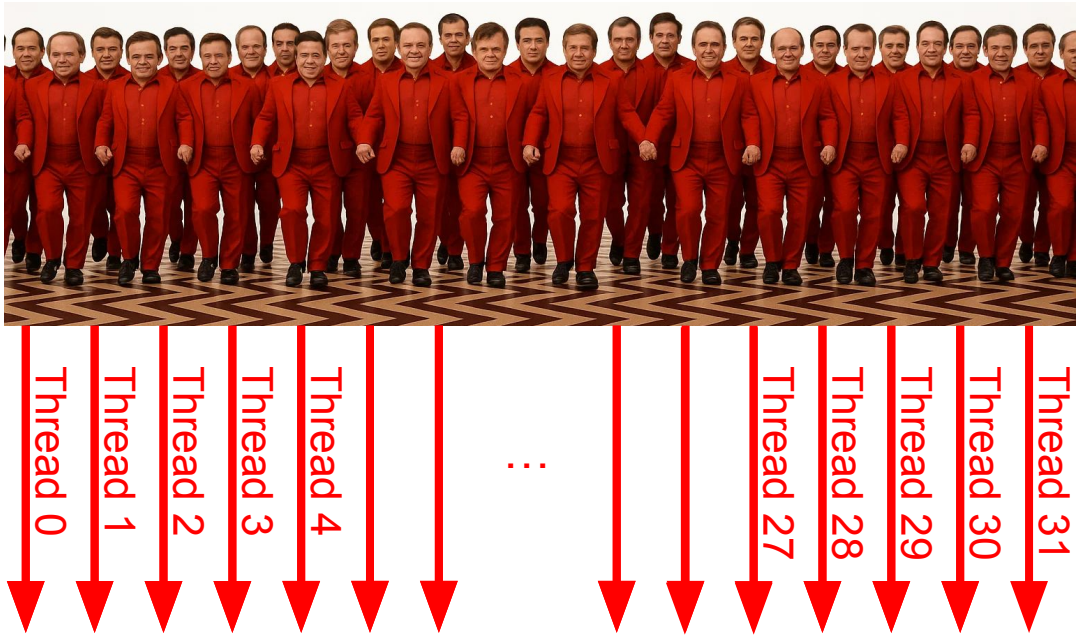


Один Instruction pointer
на все потоки warp-а!

```
239  int warpThreadID = threadIdx.x % 32;
240  int result = 0;
241  if (warpThreadID < 16) {
242      result = dataA[warpThreadID];
243  } else {
244      result = dataB[warpThreadID];
245  }
```

Архитектура

GPU warp

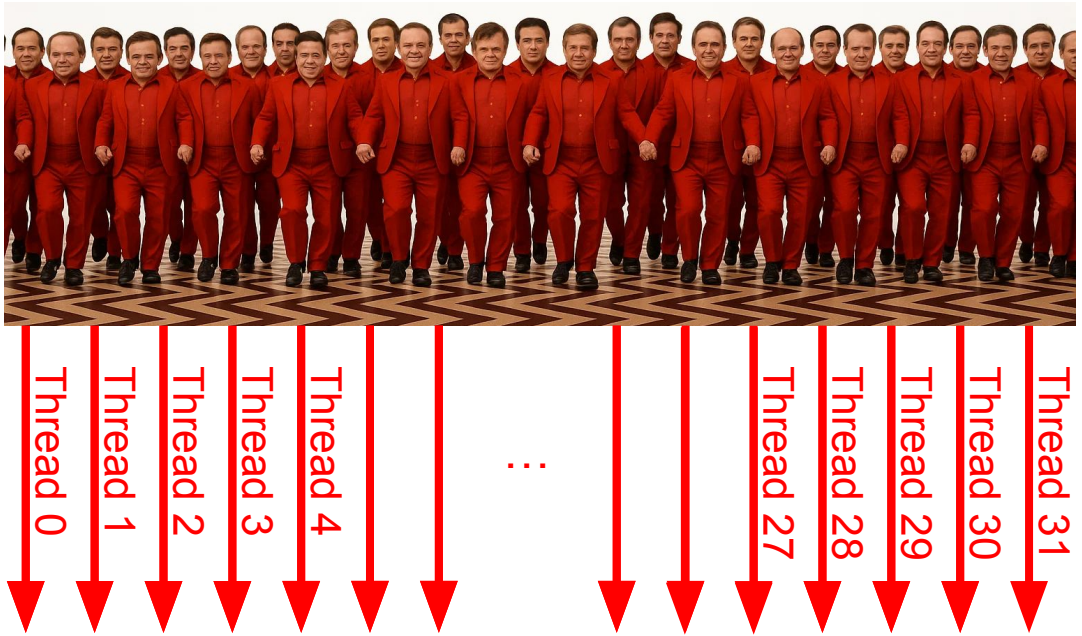


Один Instruction pointer
на все потоки warp-а!

```
239    int warpThreadID = threadIdx.x % 32;
240    int result = 0;
241    if (warpThreadID < 16) {
242        result = dataA[warpThreadID];
243    } else {
244        result = dataB[warpThreadID];
245    }
```

Архитектура

GPU warp

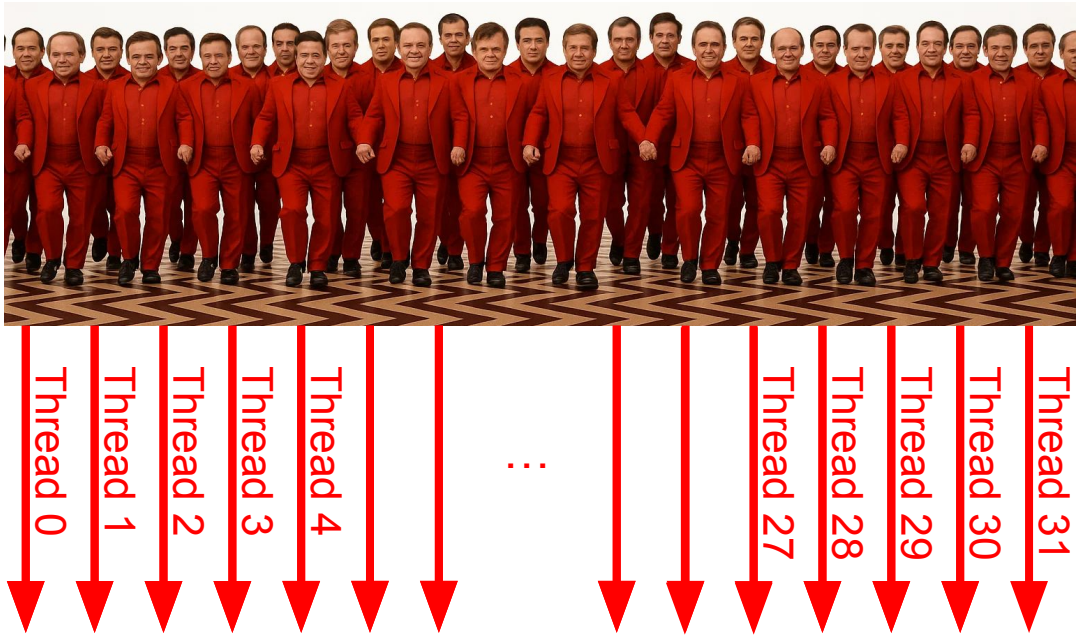


Один Instruction pointer
на все потоки warp-а!

```
239  int warpThreadID = threadIdx.x % 32;
240  int result = 0;
241  if (warpThreadID < 16) {
242      result = dataA[warpThreadID];
243  } else {
244      result = dataB[warpThreadID];
245  }
```

Архитектура

GPU warp

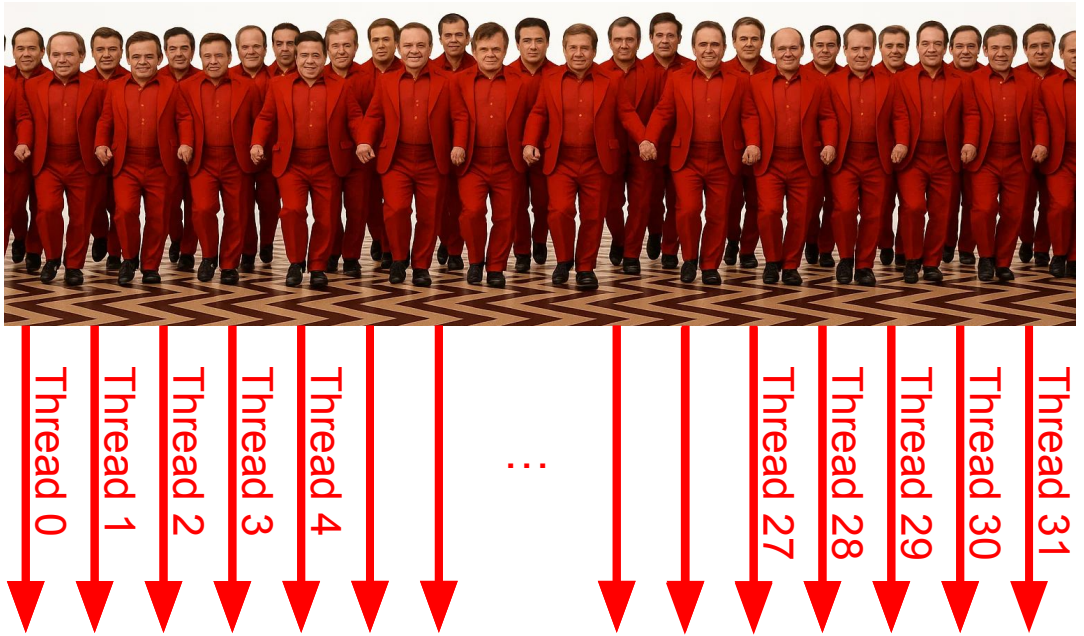


Один Instruction pointer
на все потоки warp-а!

```
239  int warpThreadID = threadIdx.x % 32;
240  int result = 0;
241  if (warpThreadID < 16) {
242      result = dataA[warpThreadID];
243  } else {
244      result = dataB[warpThreadID];
245  }
```

Архитектура

GPU warp

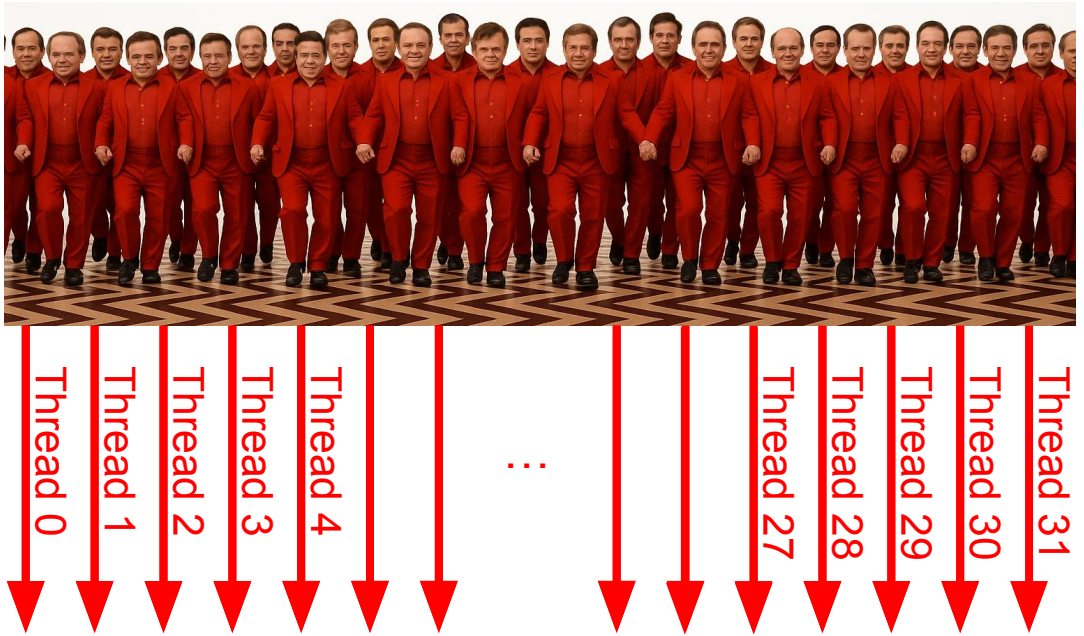


Один Instruction pointer
на все потоки warp-а!

```
239  int warpThreadID = threadIdx.x % 32;
240  int result = 0;
241  if (warpThreadID < 16) {
242      result = dataA[warpThreadID];
243  } else {
244      result = dataB[warpThreadID];
245  }
```

Архитектура

GPU warp

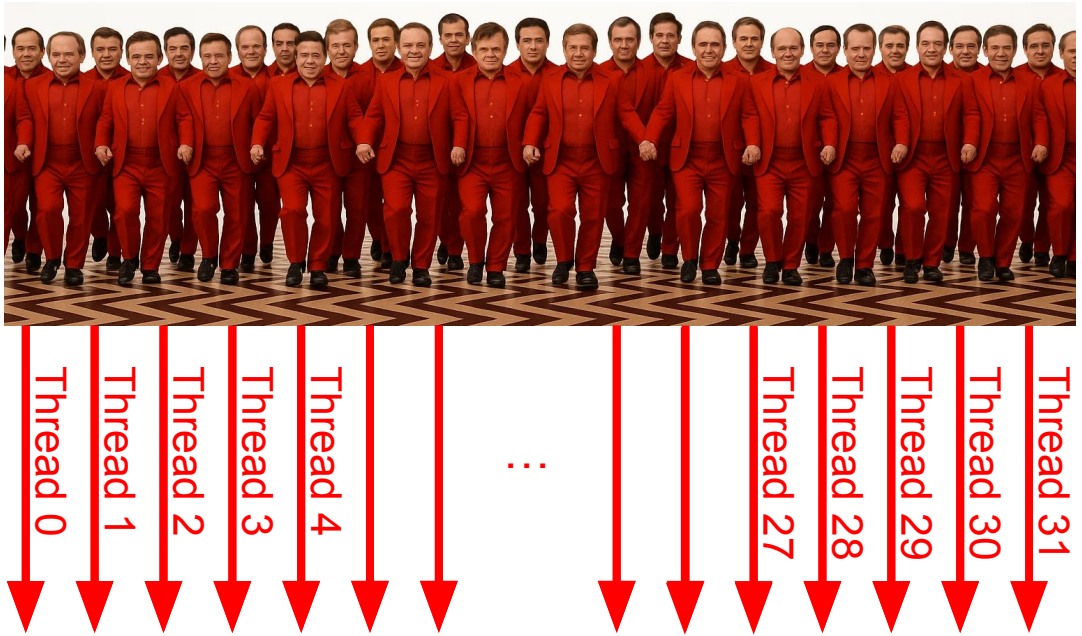


Один Instruction pointer
на все потоки warp-а!

```
239  int warpThreadID = threadIdx.x % 32;
240  int result = 0;
241  if (warpThreadID < 16) {
242      result = dataA[warpThreadID];
243  } else {
244      result = dataB[warpThreadID];
245  }
```

Архитектура

GPU warp



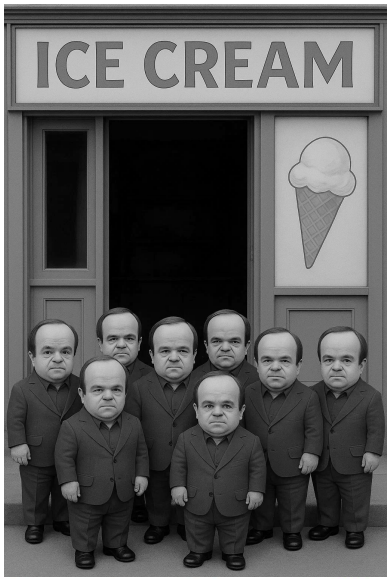
Один Instruction pointer
на все потоки warp-а!

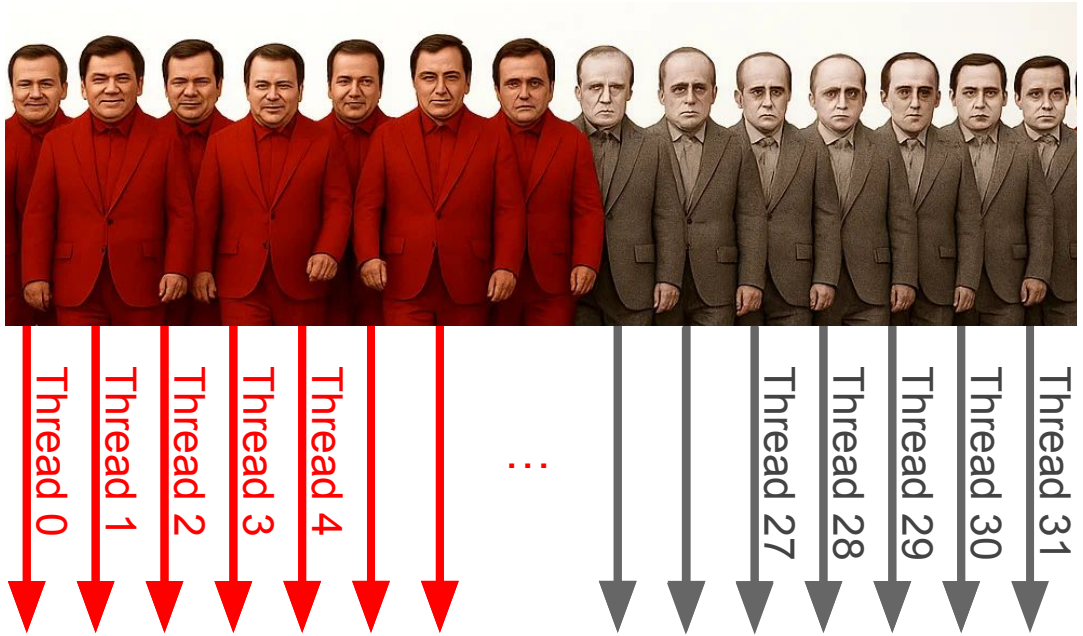
```
239     int warpThreadID = threadIdx.x % 32;
240     int result = 0;
241     if (warpThreadID < 16) {
242         result = dataA[warpThreadID];
243     } else {
244         result = dataB[warpThreadID];
245     }
```

Выходит зайдем в **обе ветки if**-а?

Архитектура

GPU warp

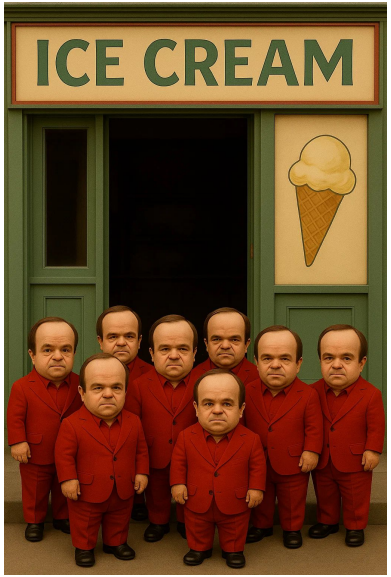




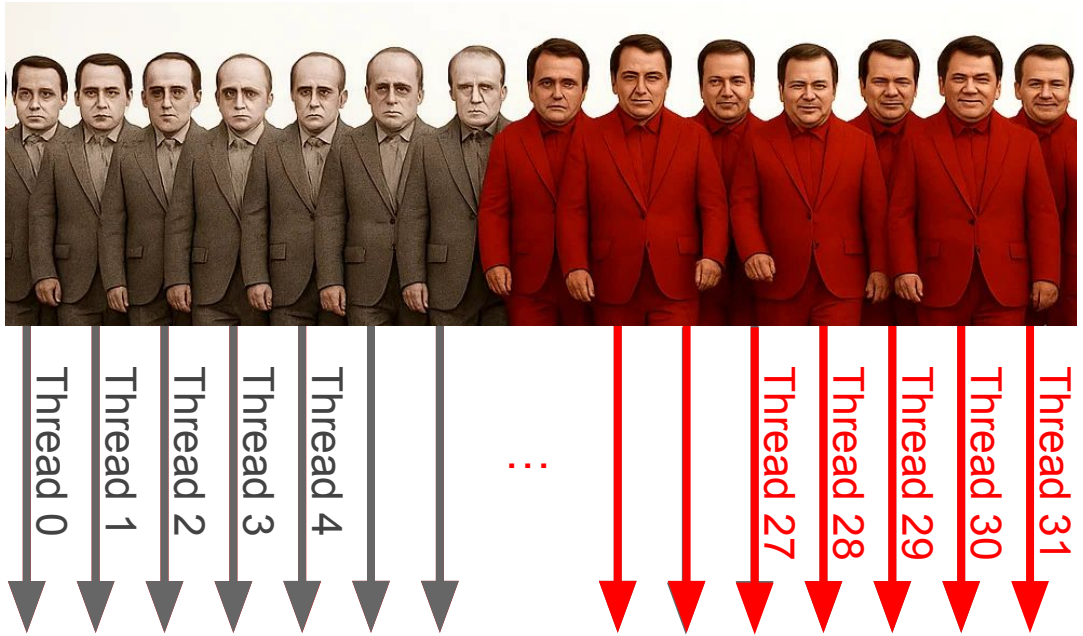
Один Instruction pointer
на все потоки warp-а!

```
239     int warpThreadID = threadIdx.x % 32;
240     int result = 0;
241     if (warpThreadID < 16) {
242         result = dataA[warpThreadID]; // exec mask: [++++ ++++ ++++ ++++ ---- ---- ---- ----]
243     } else {
244         result = dataB[warpThreadID]; // exec mask: [---- ---- ---- ---- ++++ ++++ ++++ ++++]
245     }
```

Архитектура



GPU warp

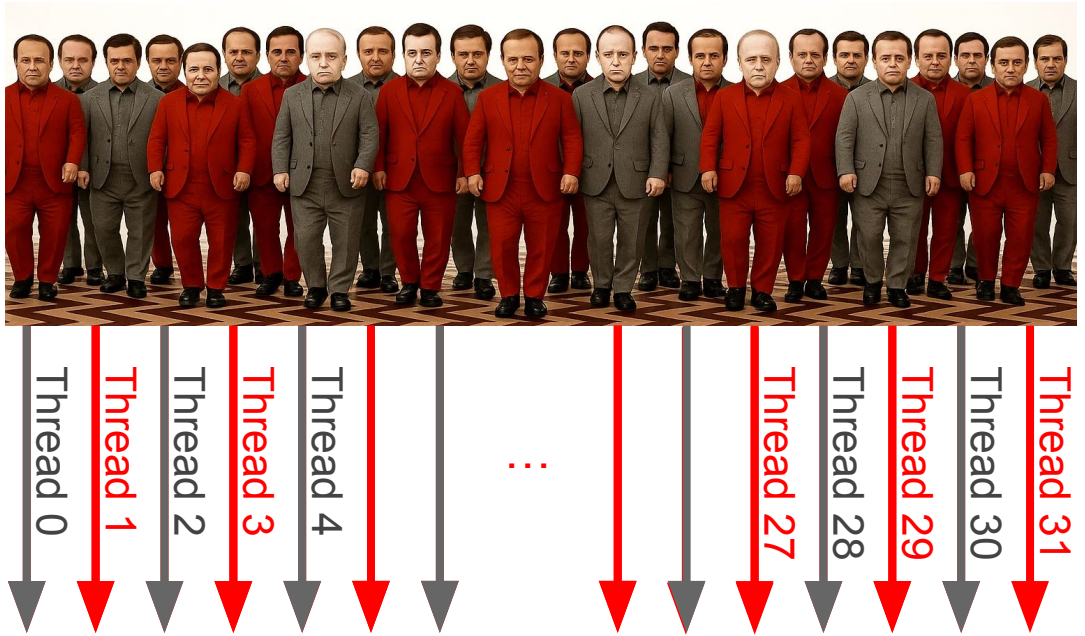


Один Instruction pointer
на все потоки warp-а!

```
239  int warpThreadID = threadIdx.x % 32;
240  int result = 0;
241  if (warpThreadID < 16) {
242      result = dataA[warpThreadID]; // exec mask: [++++ ++++ ++++ ++++ ---- ---- ---- ----]
243  } else {
244      result = dataB[warpThreadID]; // exec mask: [---- ---- ---- ---- ++++ ++++ ++++ ++++]
245  }
```

Архитектура

GPU warp

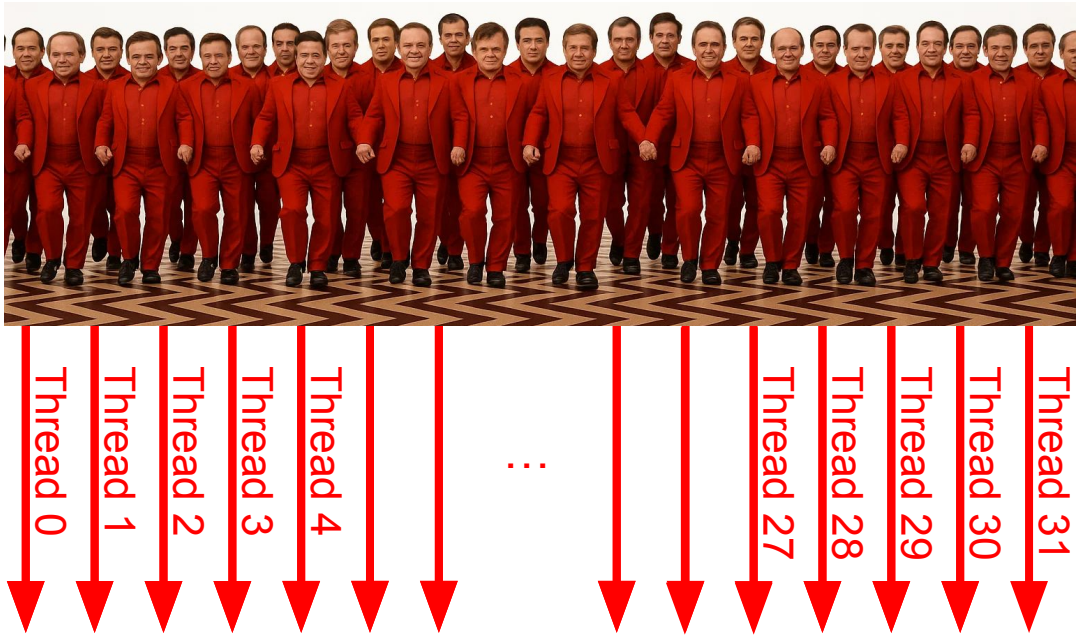


Один Instruction pointer
на все потоки warp-а!

```
239  int warpThreadID = threadIdx.x % 32;
240  int result = 0;
241  if (warpThreadID % 2 == 0) {
242      result = dataA[warpThreadID]; // exec mask: [+-+- +-+- +-+- +-+- +-+- +-+- +-+- +-+-]
243  } else {
244      result = dataB[warpThreadID]; // exec mask: [-+-+ -+-+ -+-+ -+-+ -+-+ -+-+ -+-+ -+-+]
245  }
```

Архитектура

GPU warp



Один Instruction pointer
на все потоки warp-а!

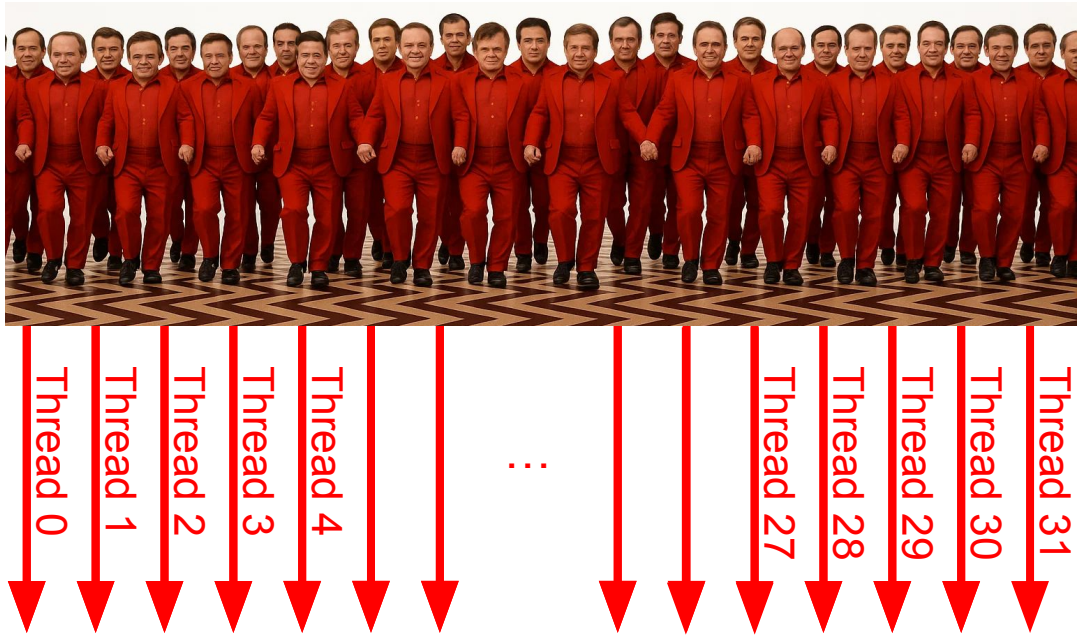
```
241  if (predicate1) {
242      if (predicate2) {
243          if (predicate3) {
244              // A1
245          } else {
246              // A2
247          }
248      } elif (predicate4) {
249          // A3
250      }
251  } else {
252      // A4
253  }
```

Сколько “времени” будет работать warp?

Архитектура

```
241  if (predicate1) {
242      if (predicate2) {
243          if (predicate3) {
244              // A1
245          } else {
246              // A2
247          }
248      } elif (predicate4) {
249          // A3
250      }
251  } else {
252      // A4
253  }
```

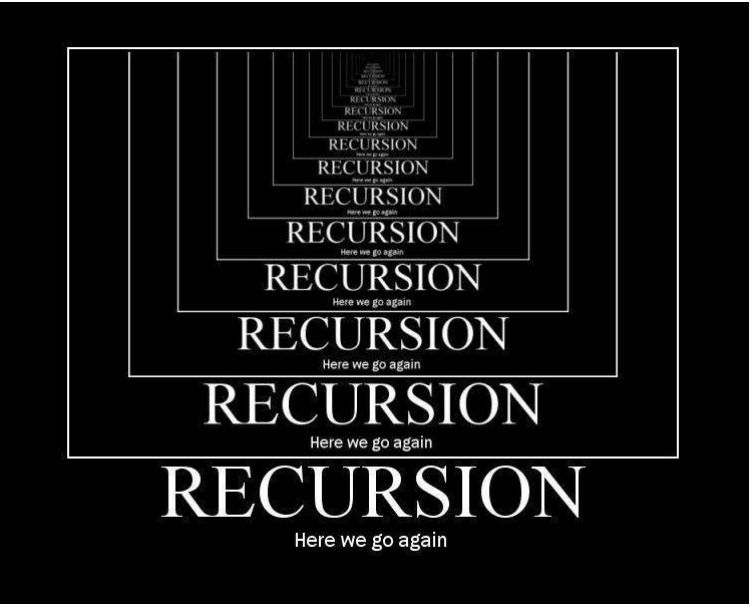
GPU warp



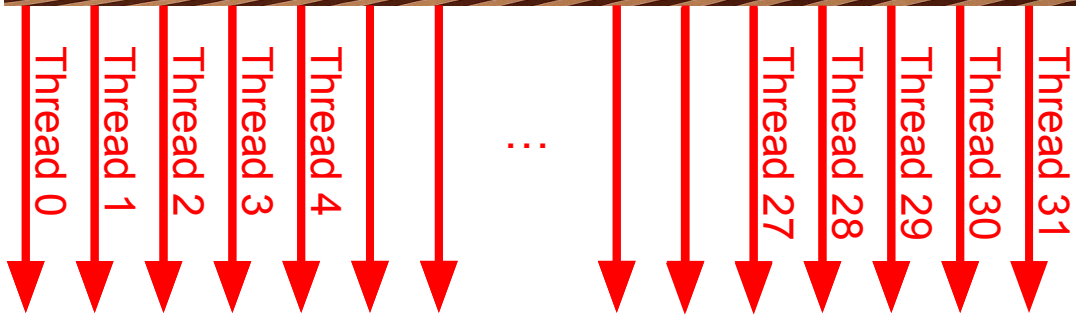
Один Instruction pointer
на все потоки warp-а!

Сколько “времени” будет работать warp?
A1 + A2 + A3 + A4 при **code divergence**

Архитектура



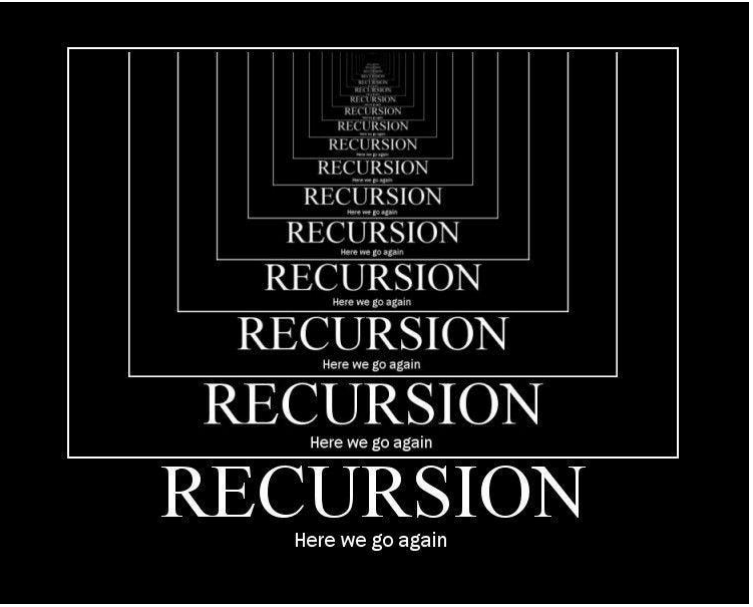
GPU warp



Один Instruction pointer
на все потоки warp-а!

Сколько “времени” будет работать достаточно глубокая рекурсия?

Архитектура



GPU warp

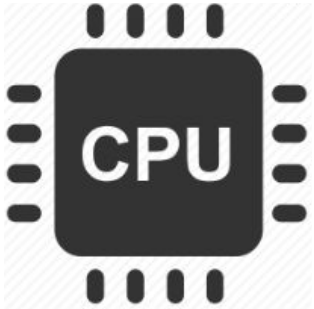
Thread 0 Thread 1 Thread 2 Thread 3 Thread 4 … Thread 27 Thread 28 Thread 29 Thread 30 Thread 31

Один Instruction pointer
на все потоки warp-а!

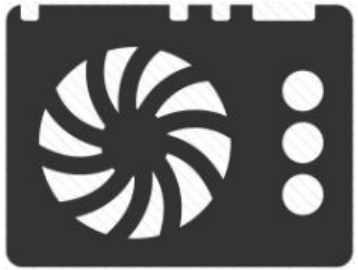
Глава 2: Работа с памятью

hyper-threading, latency hiding, occupancy, registers pressure/spilling,
coalesced memory access pattern, cache lines, local/shared memory, bank conflicts

Архитектура



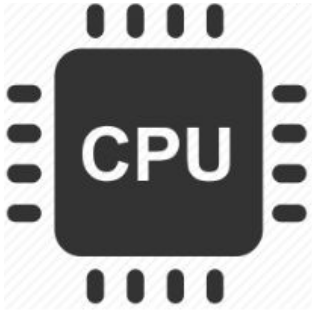
40
GB/s



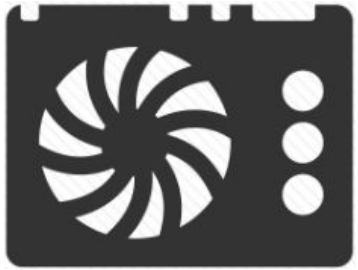
1000
GB/s

GPU

Архитектура



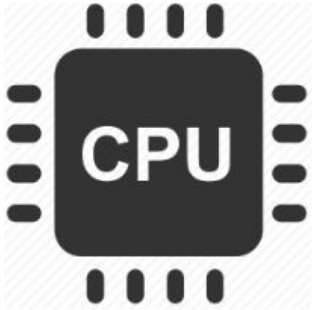
40
GB/s



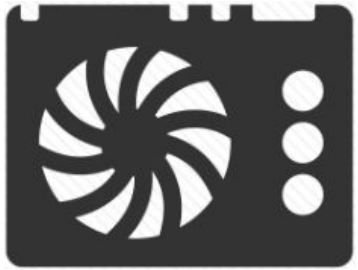
1000
GB/s

GPU

Архитектура



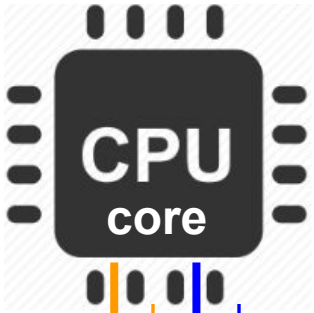
40 GB/s

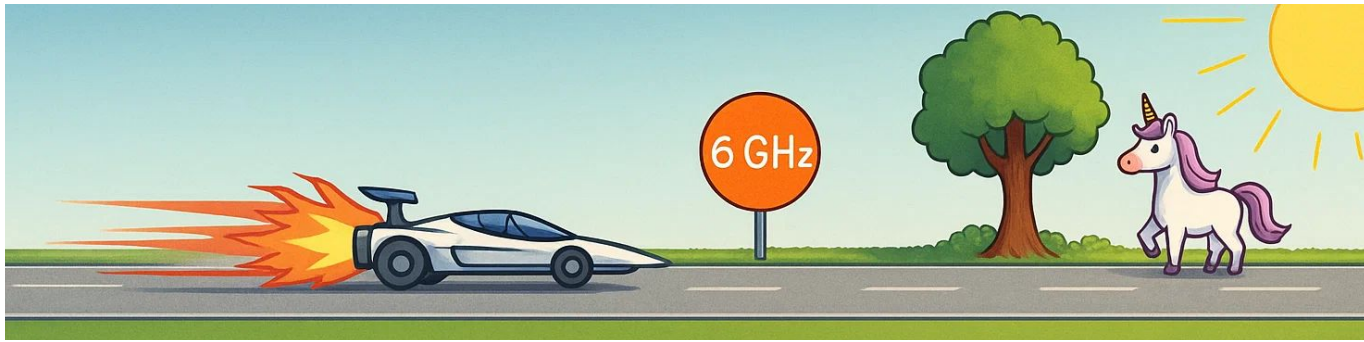


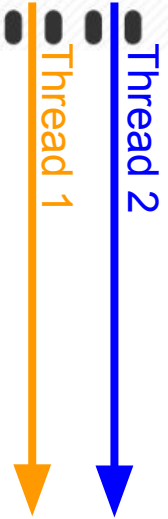
1000 GB/s

GPU

Архитектура CPU: многопоточность



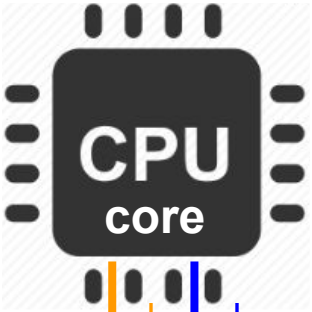


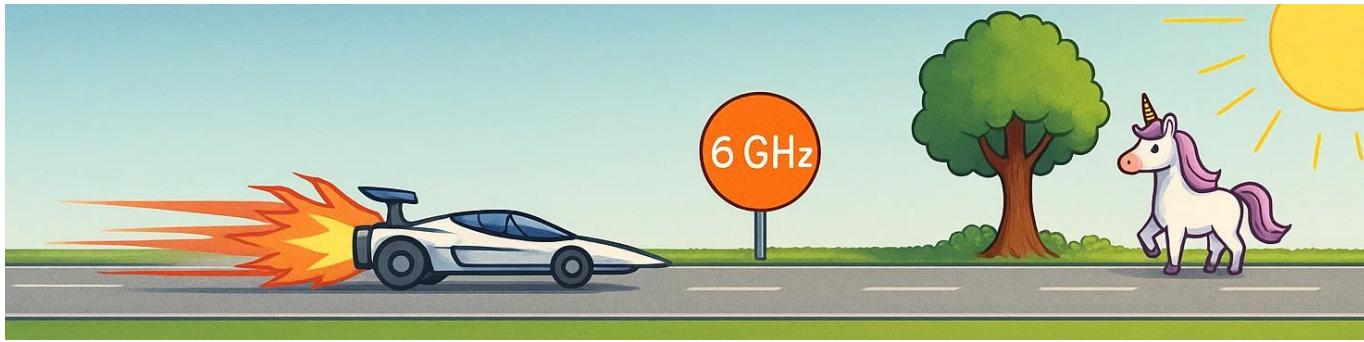


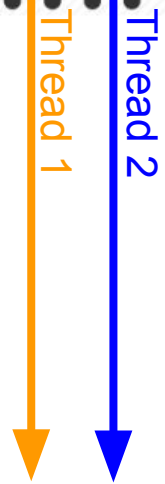
Многопоточность благодаря переключению контекста!

Что нужно перещелкнуть в состоянии ЦПУ ядра для другого потока?

Архитектура CPU: многопоточность



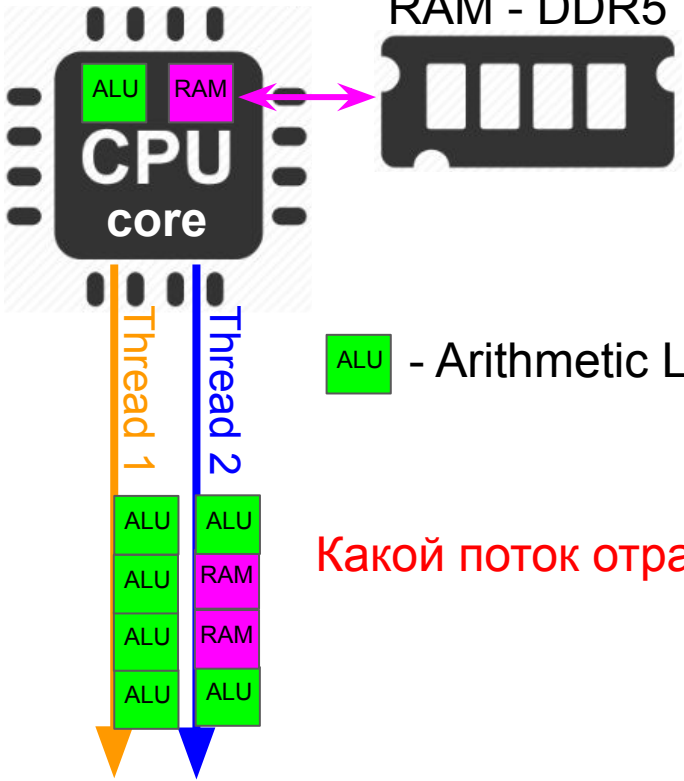




Многопоточность благодаря переключению контекста!

Context switch:
- Обновляет Instruction pointer (указатель на строку кода)
- Подгружает значения регистров процессора

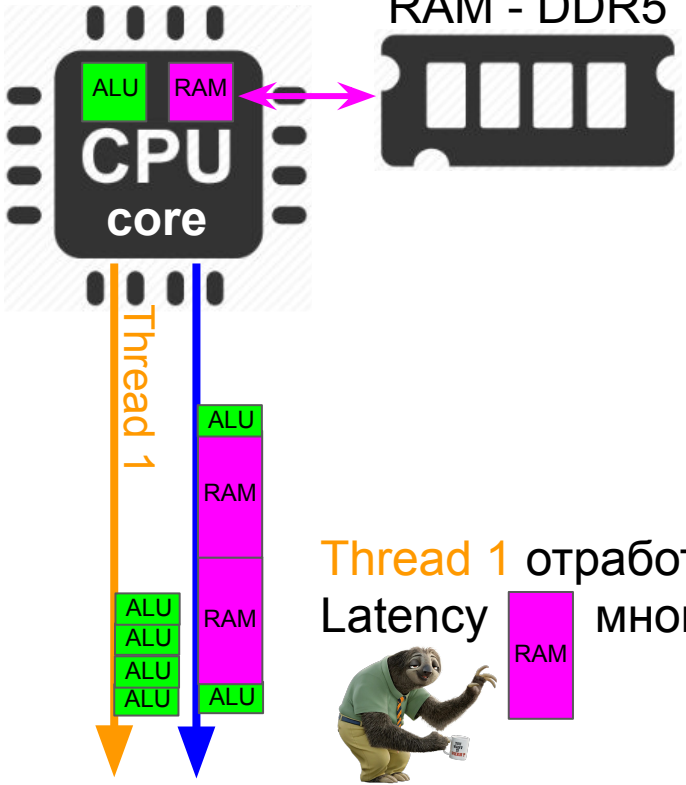
Архитектура CPU: Hyper-Threading, SMT

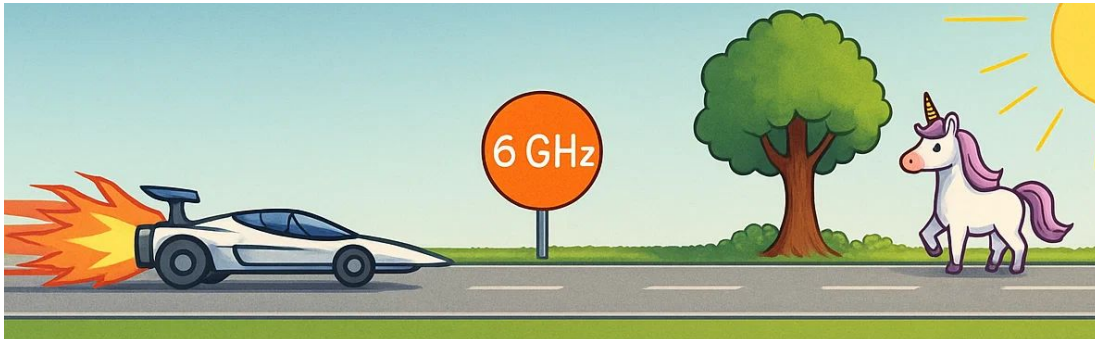


ALU - Arithmetic Logical Unit

Какой поток отработает быстрее?

Архитектура CPU: Hyper-Threading, SMT

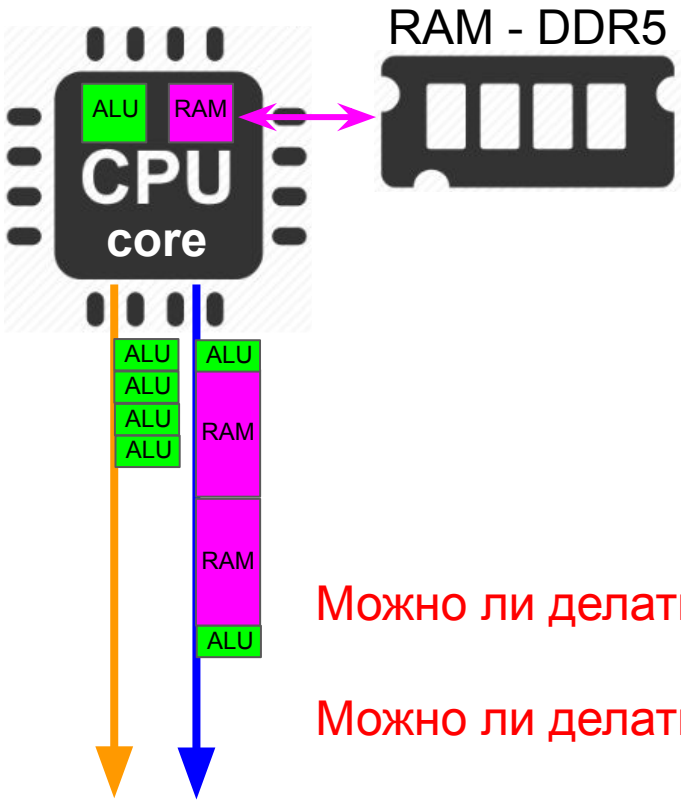




Thread 1 отработает быстрее!

Latency RAM много больше чем у ALU

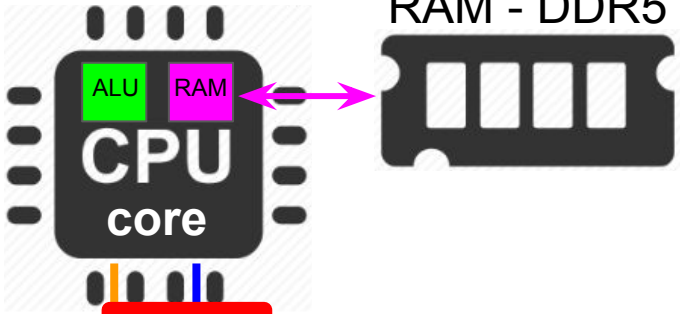
Архитектура CPU: Hyper-Threading, SMT

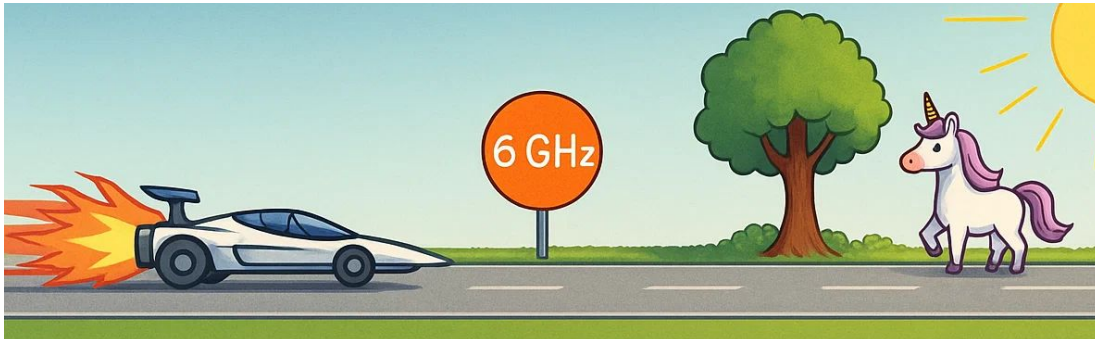


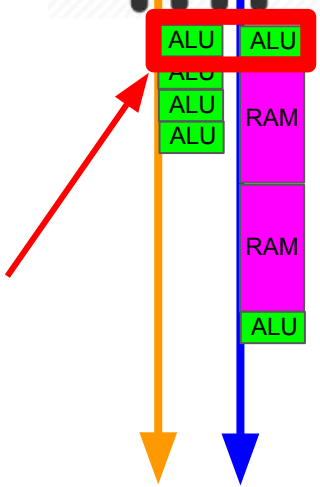
Можно ли делать ALU пока ждем RAM ?

Можно ли делать ALU пока ждем ALU ?

Архитектура CPU: Hyper-Threading, SMT



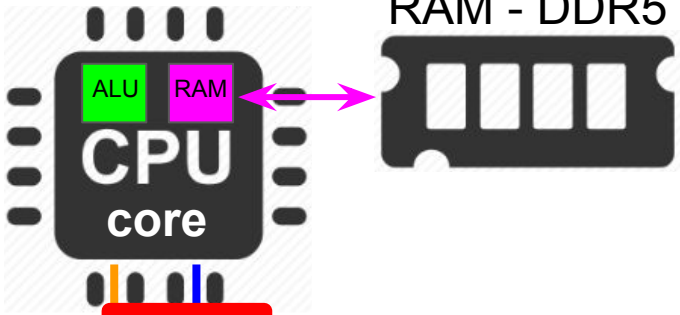


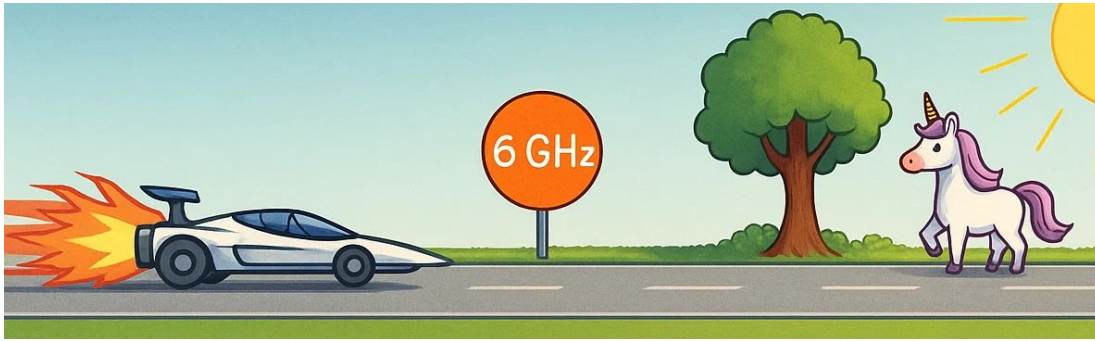


Можно ли делать ALU пока ждем RAM ?

Можно ли делать ALU пока ждем ALU ?

Архитектура CPU: Hyper-Threading, SMT

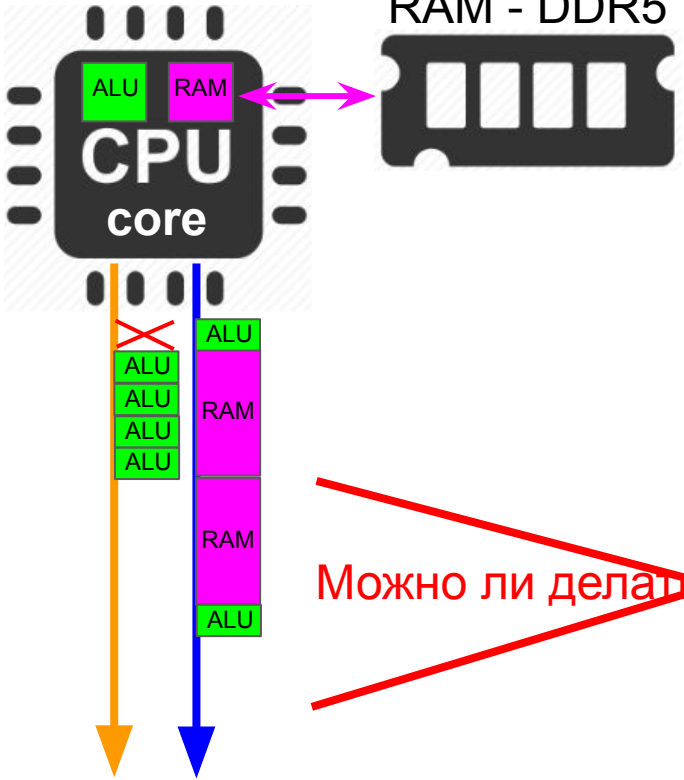


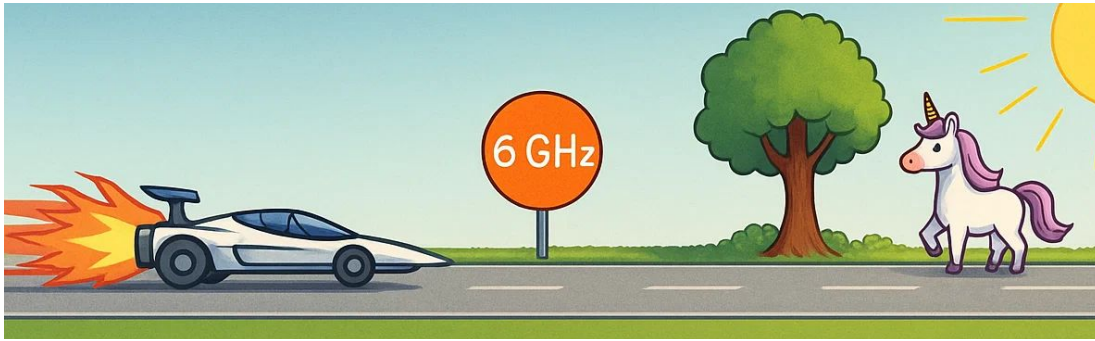


Можно ли делать ALU пока ждем RAM ?

Можно ли делать ALU пока ждем ALU ?

Архитектура CPU: Hyper-Threading, SMT

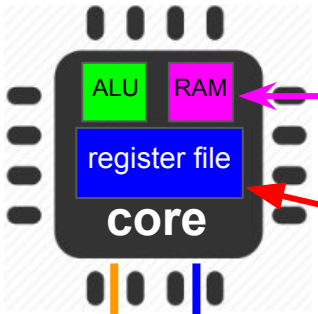




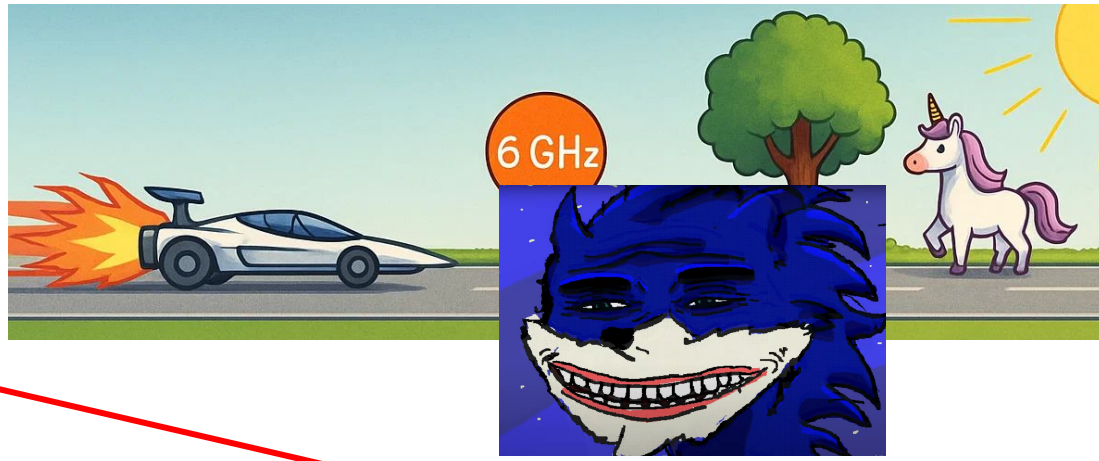
Можно ли делать ALU пока ждем RAM ?

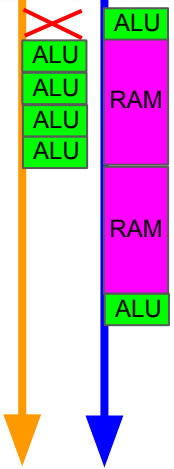
Context switch - дорого!
Долго подгружает регистры!

Архитектура CPU: Hyper-Threading, SMT



RAM - DDR5



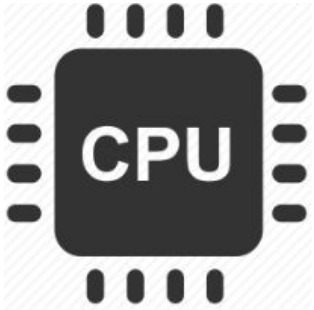


Давайте держать два множества регистров!

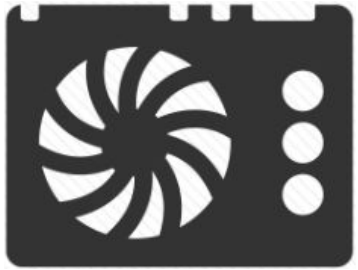
Можно делать ALU пока ждем RAM !

Это и есть SMT и HT!

Архитектура GPU



40 GB/s



1000 GB/s

Архитектура GPU

warp

32 слабых медленных CUDA ядер

SM - Streaming Multiprocessor

https://www.techpowerup.com/review/nvidia-geforce-ampere-architecture-board-design-gaming-tech-software/3.html

Архитектура GPU

warp

32 слабых медленных CUDA ядер

SM - Streaming Multiprocessor

AMD

wavefront

64 threads

NVIDIA

warp

32 threads

https://www.techpowerup.com/review/nvidia-geforce-ampere-architecture-board-design-gaming-tech-software/3.html

Архитектура GPU

warp

32 слабых медленных CUDA ядер

SM - Streaming Multiprocessor

AMD

wavefront

64 threads

Compute Units

(CU)

NVIDIA

warp

32 threads

Streaming Multiprocessor

(SM)

https://www.techpowerup.com/review/nvidia-geforce-ampere-architecture-board-design-gaming-tech-software/3.html

Архитектура GPU

warp

32 слабых медленных
CUDA ядер

SM - Streaming Multiprocessor

https://www.techpowerup.com/review/nvidia-geforce-ampere-architecture-board-design-gaming-tech-software/3.html

Архитектура GPU

warp

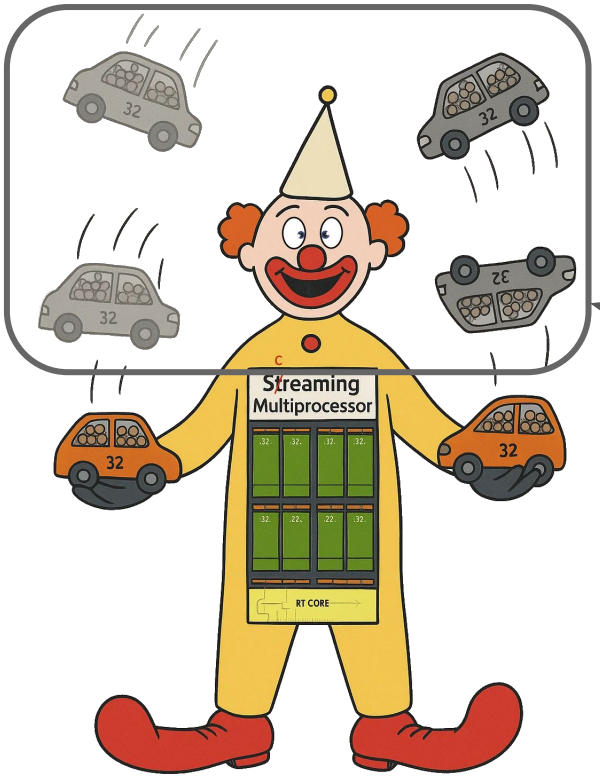
32 слабых медленных CUDA ядер

Register File для быстрого жонглирования warp-ами! (быстрый “context switch”)

SM - Streaming Multiprocessor

https://www.techpowerup.com/review/nvidia-geforce-ampere-architecture-board-design-gaming-tech-software/3.html

Архитектура GPU



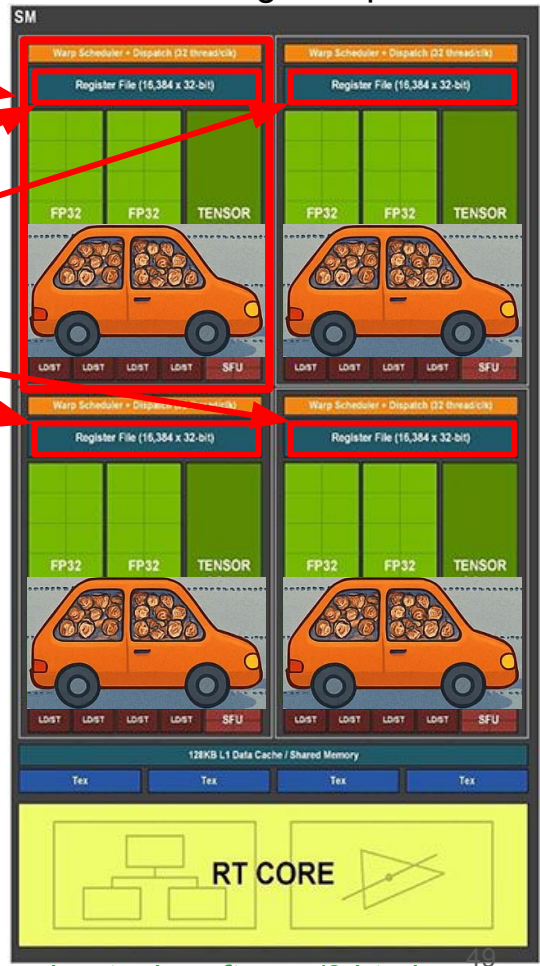
warp

32 слабых медленных CUDA ядер

Register File для быстрого жонглирования warp-ами! (быстрый “context switch”)

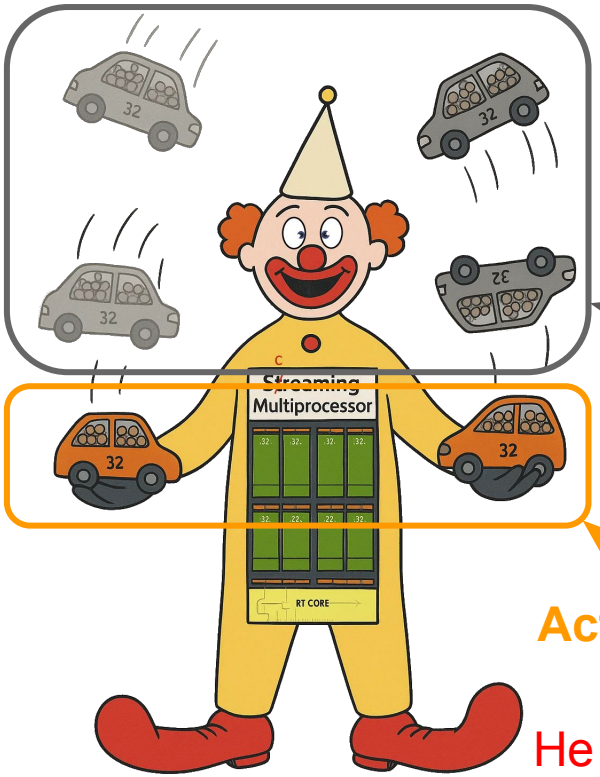
Inactive warps: ждут когда придут данные из VRAM

SM - Streaming Multiprocessor



https://www.techpowerup.com/review/nvidia-geforce-ampere-architecture-board-design-gaming-tech-software/3.html

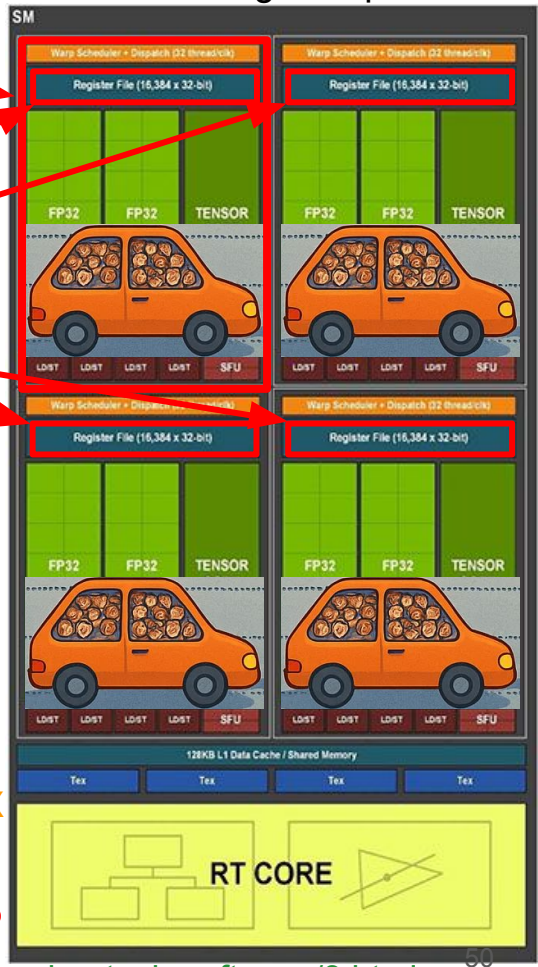
Архитектура GPU



warp

32 слабых медленных CUDA ядер

SM - Streaming Multiprocessor



Register File для быстрого жонглирования warp-ами! (быстрый “context switch”)

Inactive warps: ждут когда придут данные из VRAM

Active warps: данные в регистрах
Работаем-работаем! 😎

Не напоминает ли вам это что-то?

https://www.techpowerup.com/review/nvidia-geforce-ampere-architecture-board-design-gaming-tech-software/3.html

ULTRA SUPER MEGA GIGA TITANIC-HYPER-THREADING

Архитектура GPU

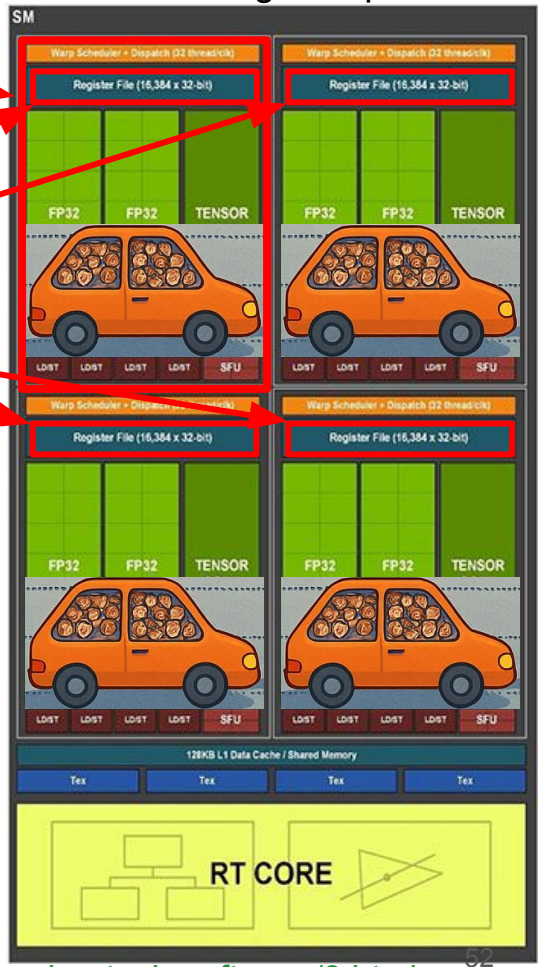
warp

32 слабых медленных CUDA ядер

Register File для быстрого жонглирования warp-ами! (быстрый “context switch”)

Occupancy = насколько много доступно warp-ов для жонглирования.

SM - Streaming Multiprocessor



https://www.techpowerup.com/review/nvidia-geforce-ampere-architecture-board-design-gaming-tech-software/3.html

Архитектура GPU

warp

32 слабых медленных CUDA ядер

Register File для быстрого жонглирования warp-ами! (быстрый “context switch”)

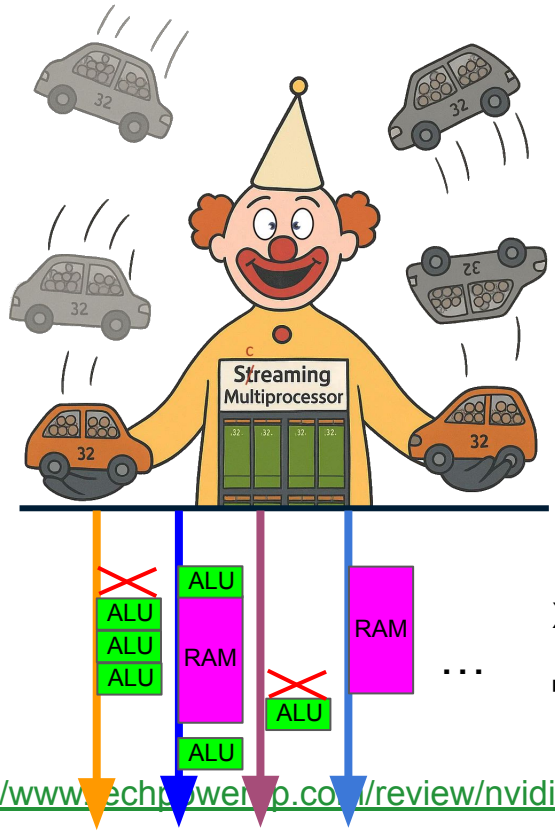
Occupancy = насколько много доступно warp-ов для жонглирования.

Если occupancy высокая, то что?

SM - Streaming Multiprocessor

https://www.techpowerup.com/review/nvidia-geforce-ampere-architecture-board-design-gaming-tech-software/3.html

Архитектура GPU



warp

32 слабых медленных CUDA ядер

Register File для быстрого жонглирования warp-ами! (быстрый “context switch”)

Occupancy = насколько много доступно warp-ов для жонглирования.

Если occupancy высокая, то **хорошо скрывается latency доступа к памяти** т.к. всегда найдется готовый warp!

SM - Streaming Multiprocessor

https://www.techpowerup.com/review/nvidia-geforce-ampere-architecture-board-design-gaming-tech-software/3.html

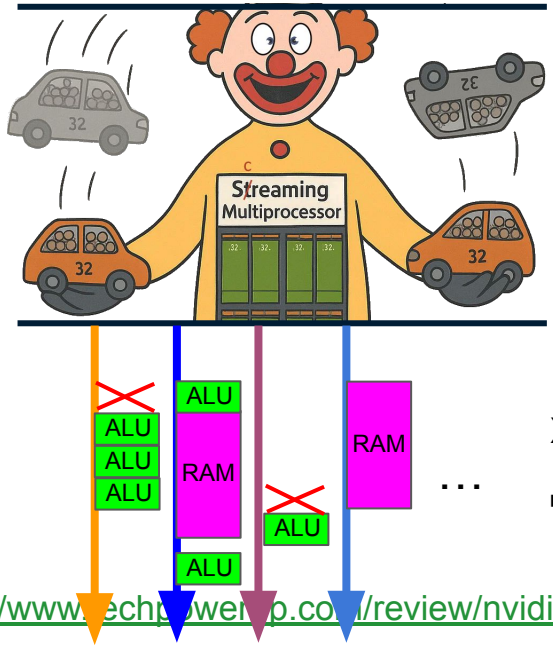
Архитектура GPU

warp

32 слабых медленных CUDA ядер

SM - Streaming Multiprocessor

А почему у SM может быть разное число warp-ов?



Register File для быстрого жонглирования warp-ами! (быстрый “context switch”)

Occupancy = насколько много доступно warp-ов для жонглирования.

Если occupancy высокая, то **хорошо скрывается latency доступа к памяти** т.к. всегда найдется готовый warp!

https://www.techpowerup.com/review/nvidia-geforce-ampere-architecture-board-design-gaming-tech-software/3.html

Архитектура GPU

warp

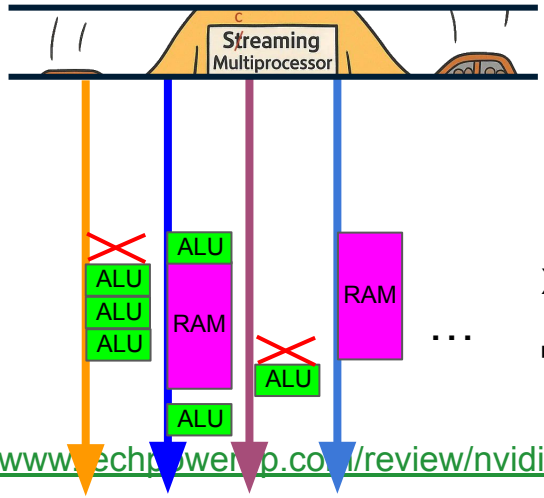
32 слабых медленных CUDA ядер

SM - Streaming Multiprocessor

А почему у SM может быть разное число warp-ов?

Register File для быстрого жонглирования warp-ами! (быстрый “context switch”)

Registers Pressure!



Occupancy = насколько много доступно warp-ов для жонглирования.

Если occupancy высокая, то **хорошо скрывается latency доступа к памяти** т.к. всегда найдется готовый warp!

https://www.techpowerup.com/review/nvidia-geforce-ampere-architecture-board-design-gaming-tech-software/3.html

Архитектура GPU

warp

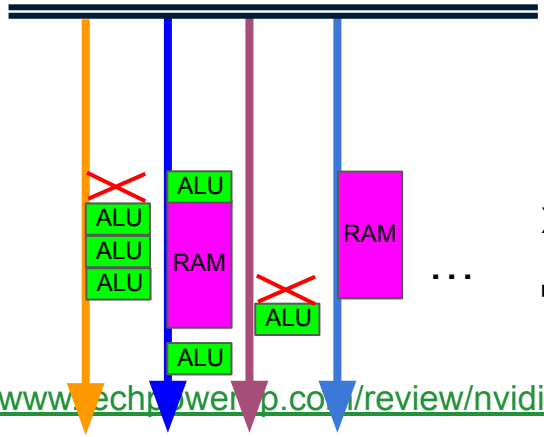
32 слабых медленных CUDA ядер

SM - Streaming Multiprocessor

А почему у SM может быть разное число warp-ов?

Register File для быстрого жонглирования warp-ами! (быстрый “context switch”)

Registers Pressure!
Вплоть до **Registers Spilling**!



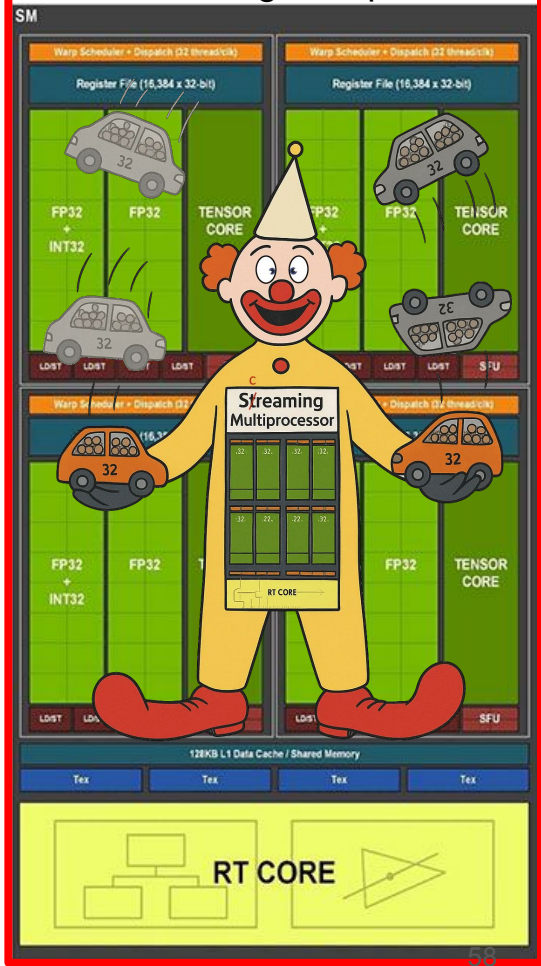
Occupancy = насколько много доступно warp-ов для жонглирования.

Если occupancy высокая, то **хорошо скрывается latency доступа к памяти** т.к. всегда найдется готовый warp!

https://www.techpowerup.com/review/nvidia-geforce-ampere-architecture-board-design-gaming-tech-software/3.html

RTX 3090: **10496 CUDA cores** = 82 SM · 4 warps · 32 ALUs

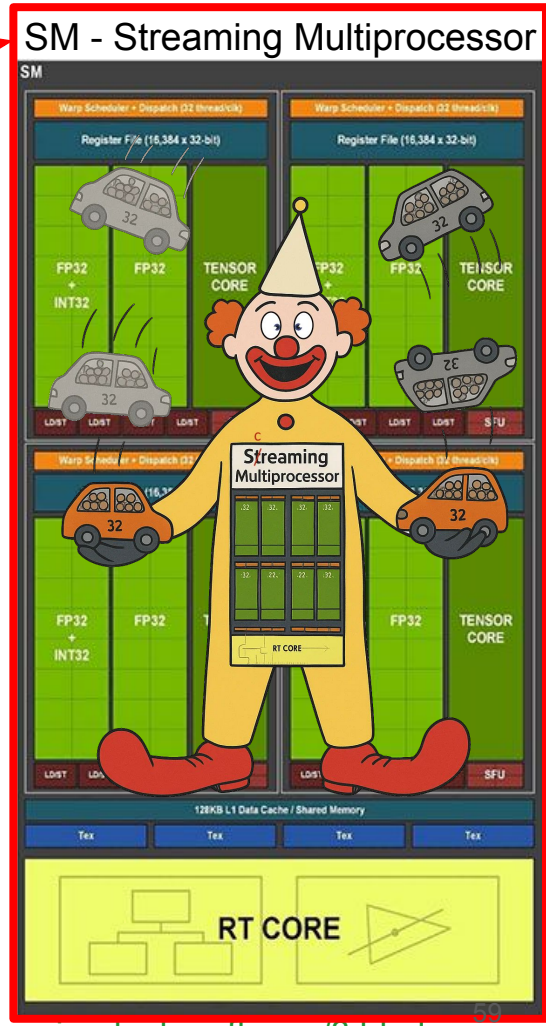
SM - Streaming Multiprocessor



https://www.techpowerup.com/review/nvidia-geforce-ampere-architecture-board-design-gaming-tech-software/3.html

RTX 3090: **10496 CUDA cores** = 82 SM · 4 warps · 32 ALUs

Куда сбежали два клоуна?

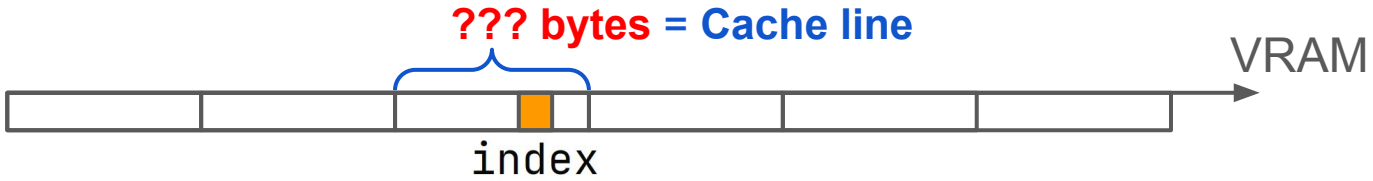


https://www.techpowerup.com/review/nvidia-geforce-ampere-architecture-board-design-gaming-tech-software/3.html

Архитектура VRAM (**coalesced** memory access pattern)


```
float value = data[index];
```

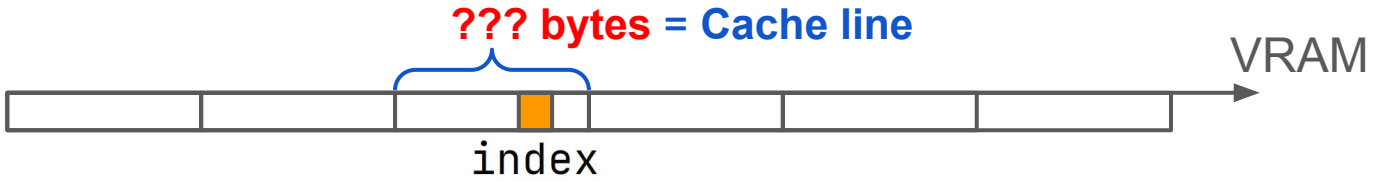
Архитектура VRAM (**coalesced** memory access pattern)




```
float value = data[index];
```

Какого размера вы бы решили сделать **Cache line**?

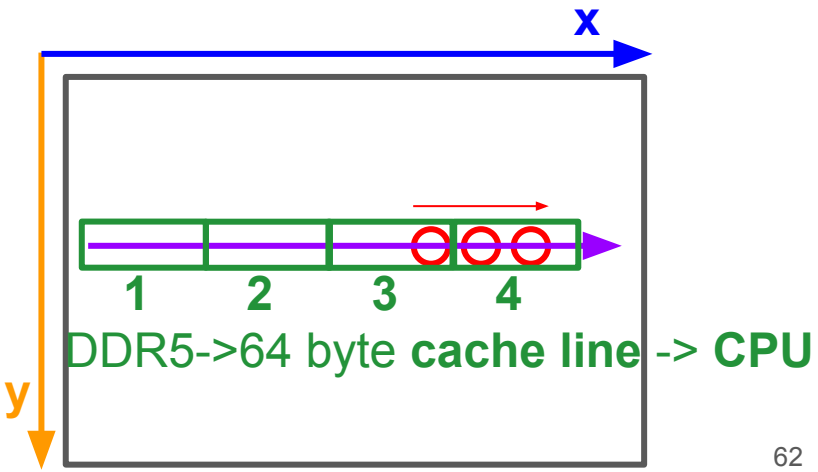
Архитектура VRAM (**coalesced** memory access pattern)



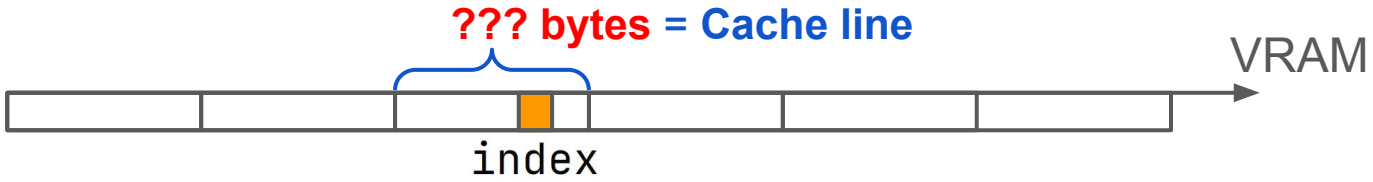

```
float value = data[index];
```

Какого размера вы бы решили сделать **Cache line**?

Напоминание про CPU:



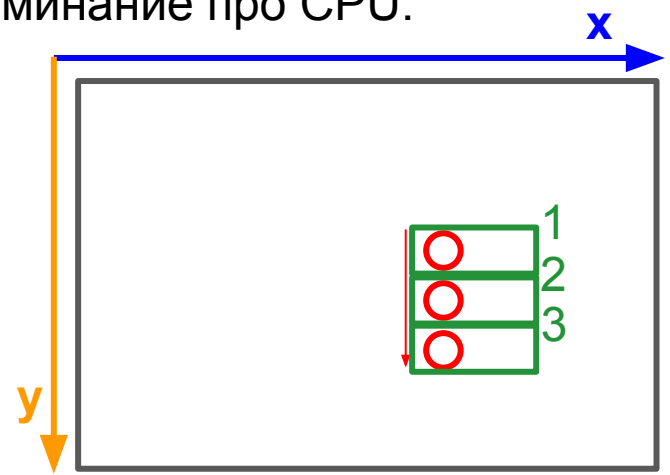
Архитектура VRAM (**coalesced** memory access pattern)

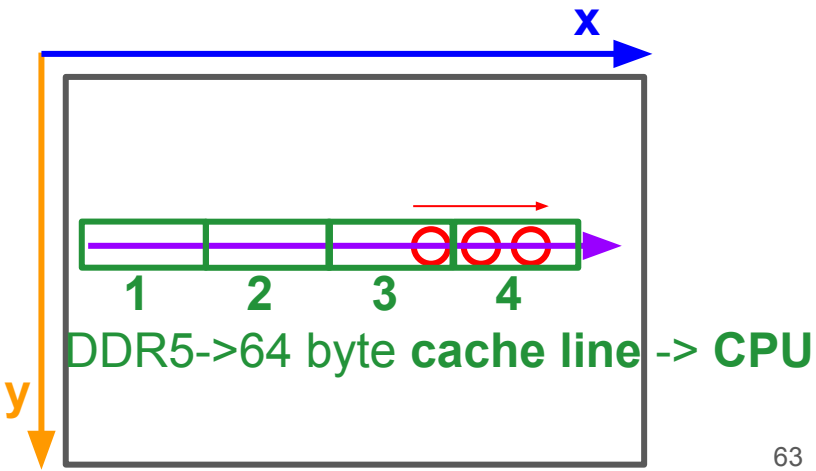



```
float value = data[index];
```

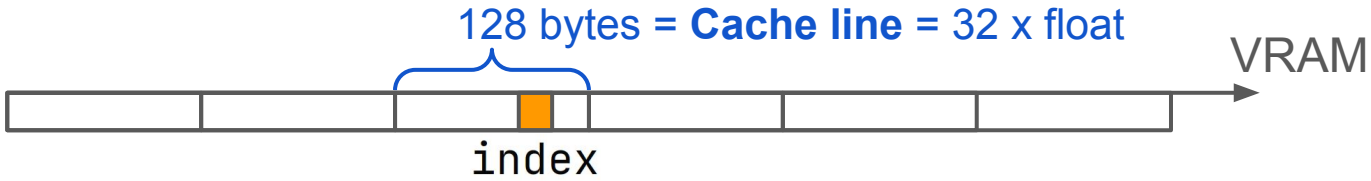
Какого размера вы бы решили сделать **Cache line**?

Напоминание про CPU:



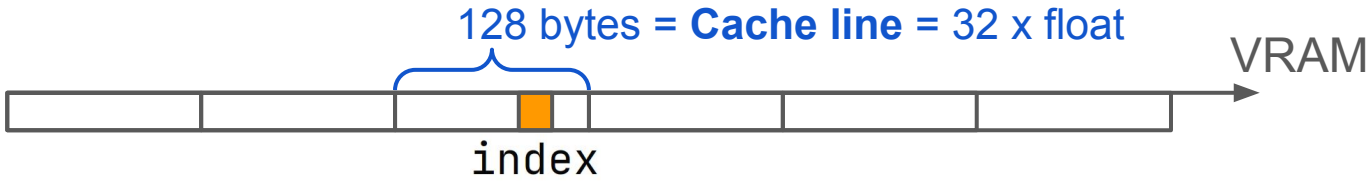


Архитектура VRAM (**coalesced** memory access pattern)



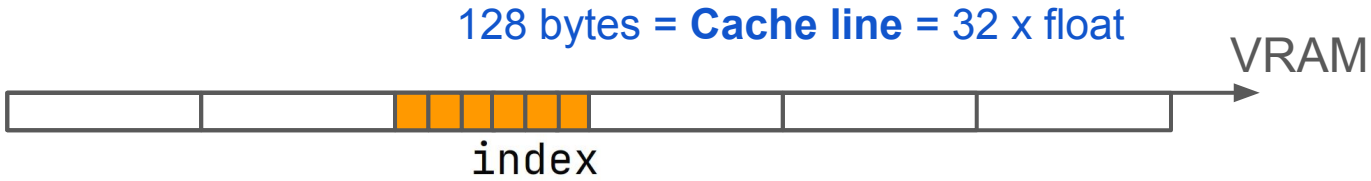

```
float value = data[index];
```

Архитектура VRAM (**coalesced** memory access pattern)




```
float value = data[index];
```

Архитектура VRAM (**coalesced** memory access pattern)

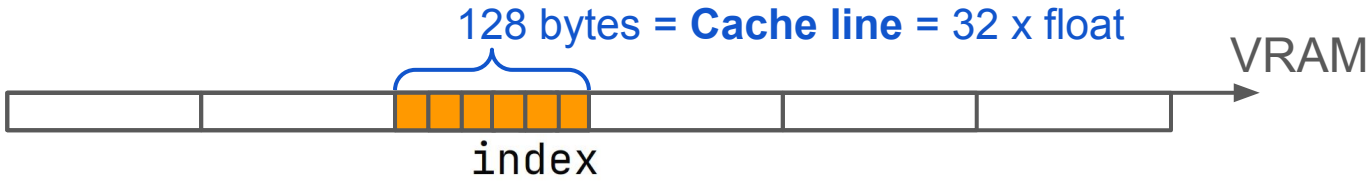



```
float value = data[index];
```

index	cache lines count
[1024+**0**; 1024+**32**)	???

Сколько cache line-ов чтобы покрыть заказ?
Сколько потребуется **транзакций** из VRAM?

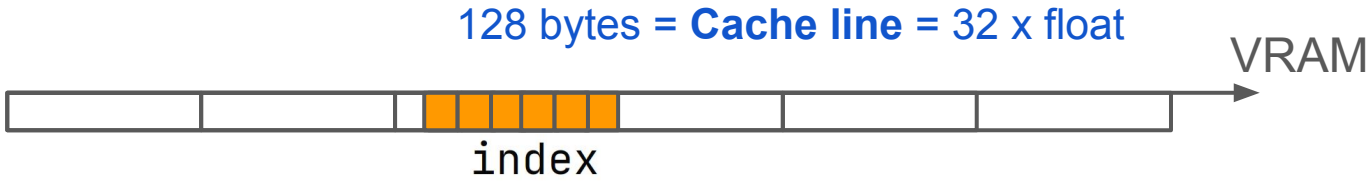
Архитектура VRAM (**coalesced** memory access pattern)




```
float value = data[index];
```

index	cache lines count
[1024+**0**; 1024+**32**)	1 транзакция (**coalesced**)

Архитектура VRAM (**coalesced** memory access pattern)

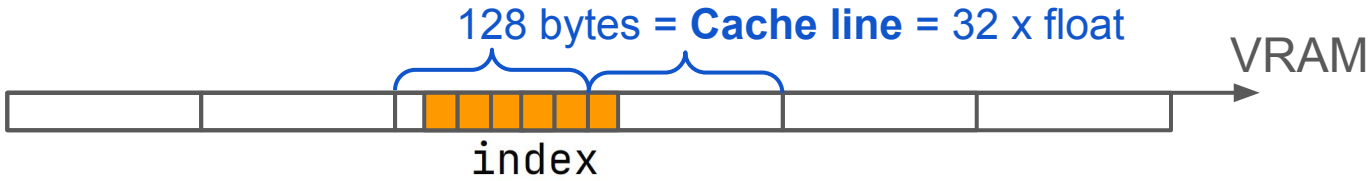



```
float value = data[index];
```

index	cache lines count
[1024+**0**; 1024+**32**)	1 транзакция (**coalesced**)
[1024+**1**; 1024+**33**)	???

Сколько cache line-ов чтобы покрыть заказ?
Сколько потребуется **транзакций** из VRAM?

Архитектура VRAM (**coalesced** memory access pattern)




```
float value = data[index];
```

index	cache lines count
[1024+**0**; 1024+**32**)	1 транзакция (**coalesced**)
[1024+**1**; 1024+**33**)	2 транзакции (**coalesced**)

Насколько просела достигнутая полезная пропускная способность VRAM?

Архитектура VRAM (**coalesced** memory access pattern)

128 bytes = **Cache line** = 32 x float


```
float value = data[index];
```

index	cache lines count
[1024+**0**; 1024+**32**)	1 транзакция (**coalesced**)
[1024+**1**; 1024+**33**)	2 транзакции (**coalesced**)
{1024 + **i*32**}	???

Сколько cache line-ов чтобы покрыть заказ?
Сколько потребуется **транзакций** из VRAM?

Архитектура VRAM (**coalesced** memory access pattern)

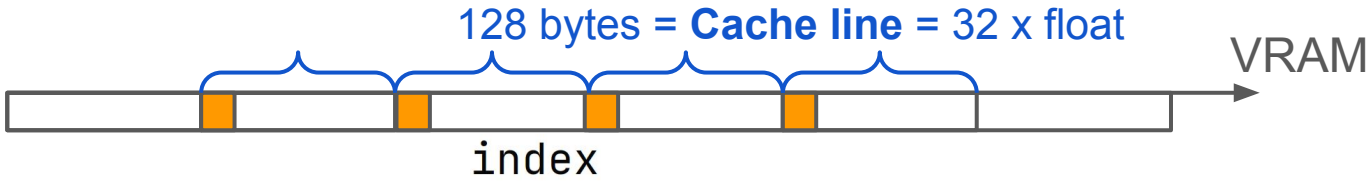
128 bytes = **Cache line** = 32 x float


```
float value = data[index];
```

index	cache lines count
[1024+**0**; 1024+**32**)	1 транзакция (**coalesced**)
[1024+**1**; 1024+**33**)	2 транзакции (**coalesced**)
{1024 + **i*32**}	???

Сколько cache line-ов чтобы покрыть заказ?
Сколько потребуется **транзакций** из VRAM?

Архитектура VRAM (**coalesced** memory access pattern)

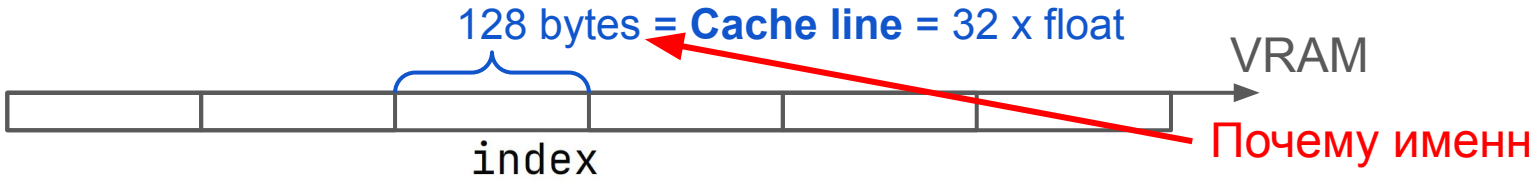



```
float value = data[index];
```

index	cache lines count
[1024+**0**; 1024+**32**)	1 транзакция (**coalesced**)
[1024+**1**; 1024+**33**)	2 транзакции (**coalesced**)
{1024 + **i*32**}	32 транзакции (**uncoalesced**)

Насколько просела достигнутая полезная пропускная способность VRAM?

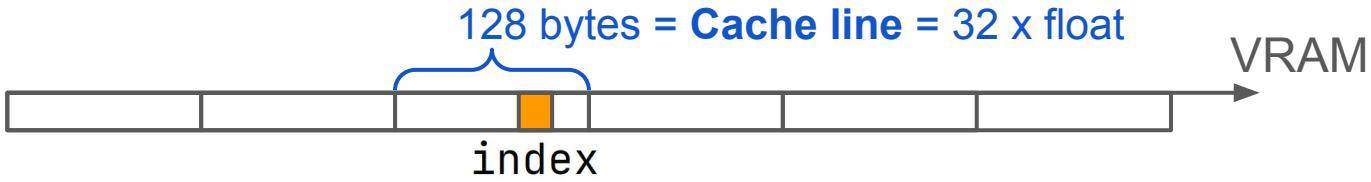
Архитектура VRAM (**coalesced** memory access pattern)



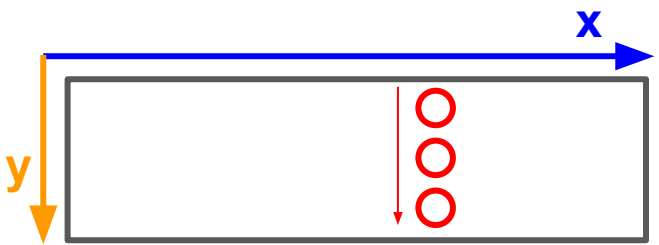

```
float value = data[index];
```

index	cache lines count
[1024+**0**; 1024+**32**)	1 транзакция (**coalesced**)
[1024+**1**; 1024+**33**)	2 транзакции (**coalesced**)
{1024 + **i*32**}	32 транзакции (**uncoalesced**)

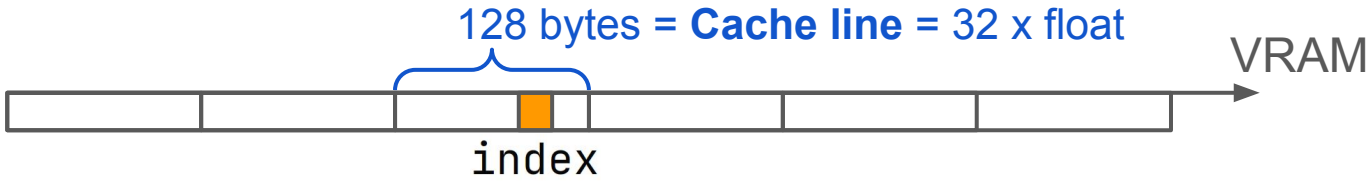
Архитектура VRAM (**coalesced** memory access pattern)




```
sum = 0;
for (int x = 0; x < width; ++x) {
    for (int y = 0; y < height; ++y) {
        sum += image[y * width + x];
    }
}
```

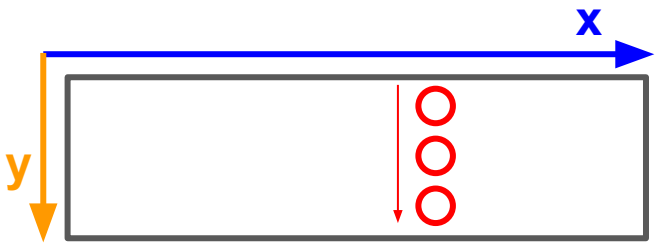


Архитектура VRAM (**coalesced** memory access pattern)

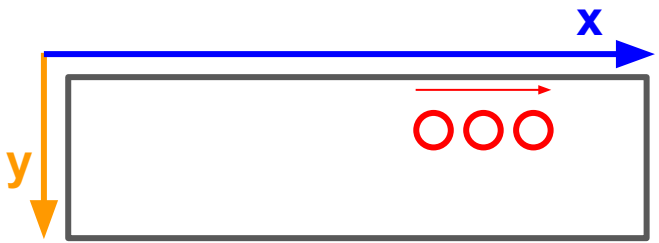


Но это однопоточный код, как в него добавить массовый параллелизм?

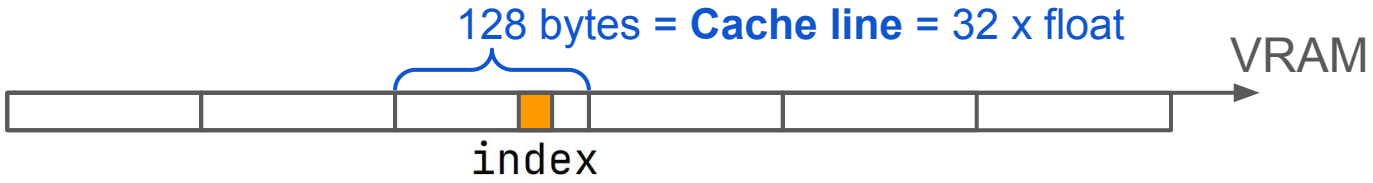
```
sum = 0;
for (int x = 0; x < width; ++x) {
    for (int y = 0; y < height; ++y) {
        sum += image[y * width + x];
    }
}
```




```
sum = 0;
for (int y = 0; y < height; ++y) {
    for (int x = 0; x < width; ++x) {
        sum += image[y * width + x];
    }
}
```

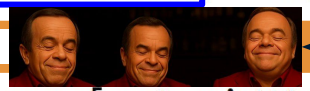


Архитектура VRAM (**coalesced** memory access pattern)

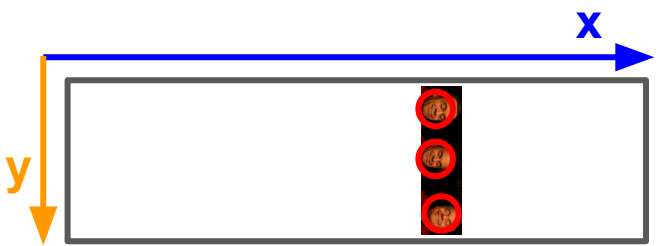


Но это однопоточный код, как в него добавить массовый параллелизм?

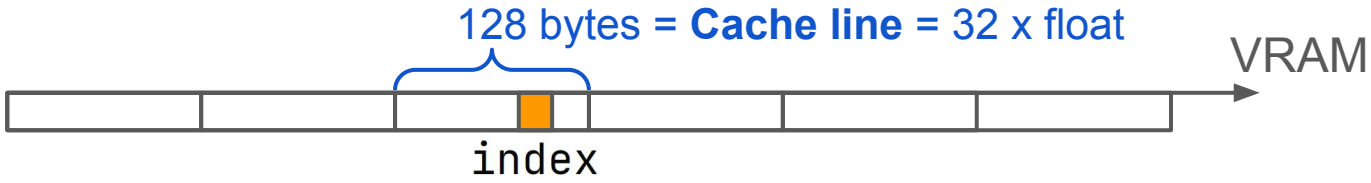
```
sum = 0;
for (int x = 0; x < width; ++x) {
    y =
    sum += image[y * width + x];
}
}
```



Номер потока в ***warp***

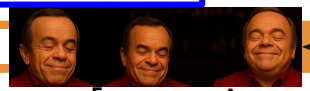


Архитектура VRAM (**coalesced** memory access pattern)

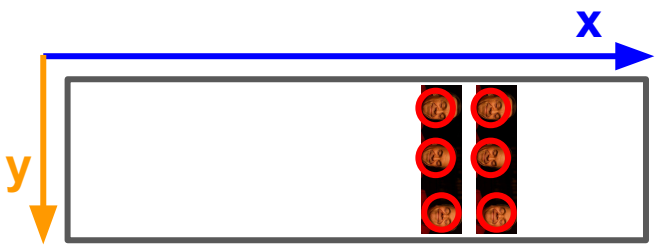


Но это однопоточный код, как в него добавить массовый параллелизм?

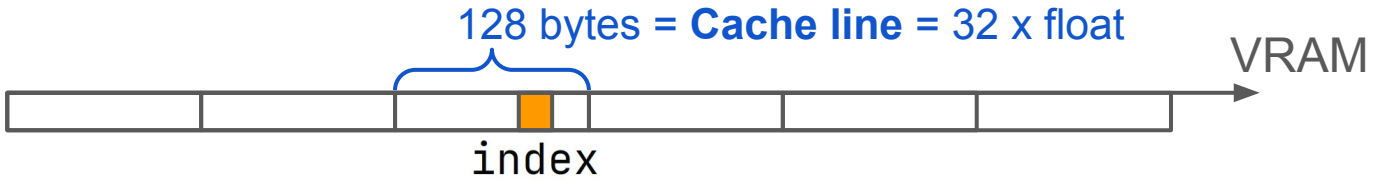
```
sum = 0;
for (int x = 0; x < width; ++x) {
    y =
    sum += image[y * width + x];
}
}
```



Номер потока в ***warp***

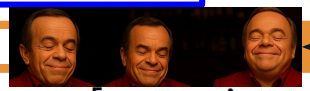


Архитектура VRAM (**coalesced** memory access pattern)

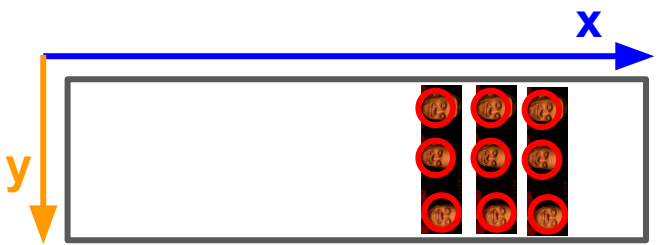


Но это однопоточный код, как в него добавить массовый параллелизм?

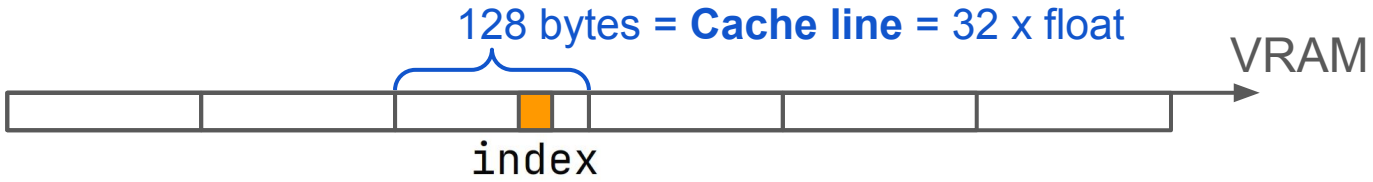
```
sum = 0;
for (int x = 0; x < width; ++x) {
    y =
    sum += image[y * width + x];
}
}
```



Номер потока в ***warp***



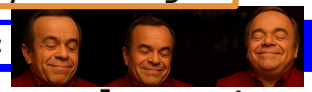
Архитектура VRAM (**coalesced** memory access pattern)

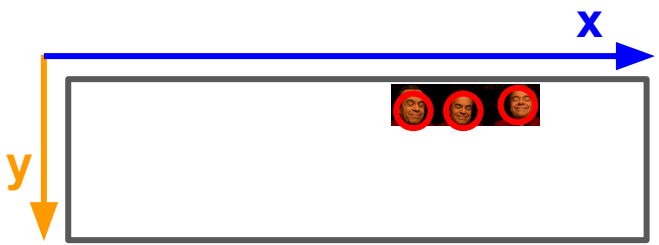


Но это однопоточный код, как в него добавить массовый параллелизм?

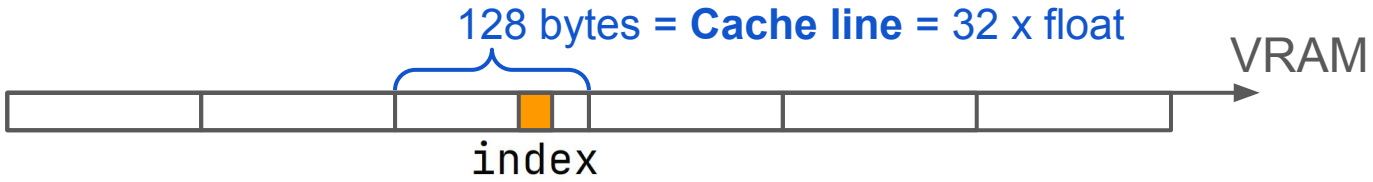
Номер потока в ***warp***

```
sum = 0;
for (int y = 0; y < height; ++y) {
    x =                                    {
        sum += image[y * width + x];
    }
}
```





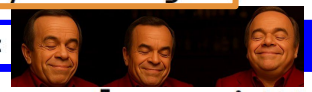
Архитектура VRAM (**coalesced** memory access pattern)

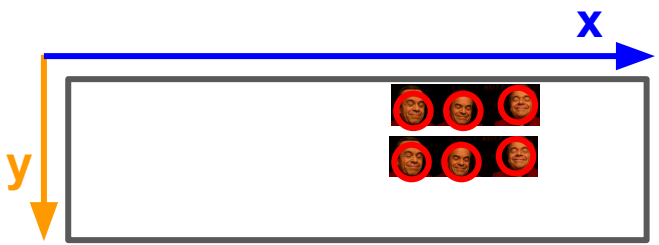


Но это однопоточный код, как в него добавить массовый параллелизм?

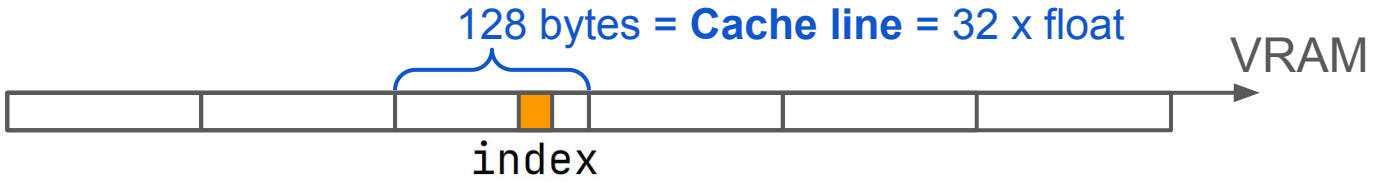
Номер потока в ***warp***

```
sum = 0;
for (int y = 0; y < height; ++y) {
    x =                                    {
        sum += image[y * width + x];
    }
}
```





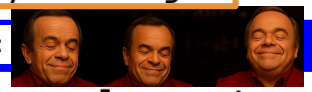
Архитектура VRAM (**coalesced** memory access pattern)

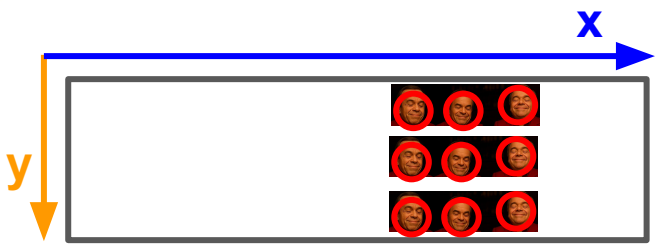


Но это однопоточный код, как в него добавить массовый параллелизм?

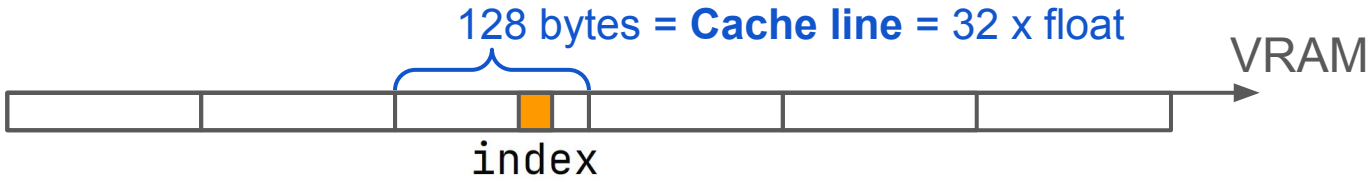
Номер потока в ***warp***

```
sum = 0;
for (int y = 0; y < height; ++y) {
    x =                                  {
        sum += image[y * width + x];
    }
}
```



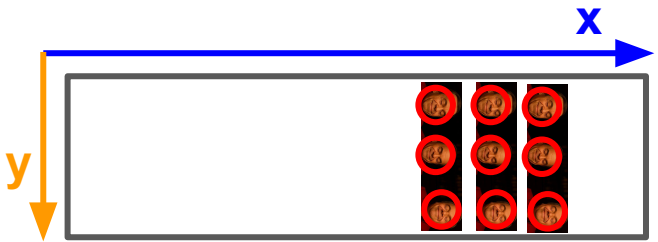


Архитектура VRAM (**coalesced** memory access pattern)

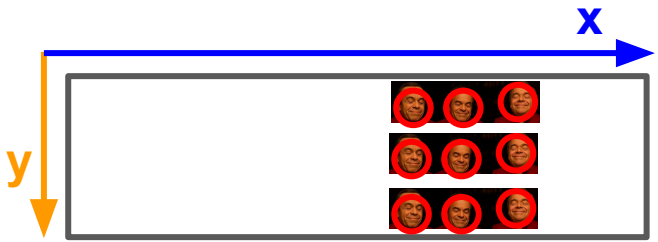


Какой код выбрали бы вы?

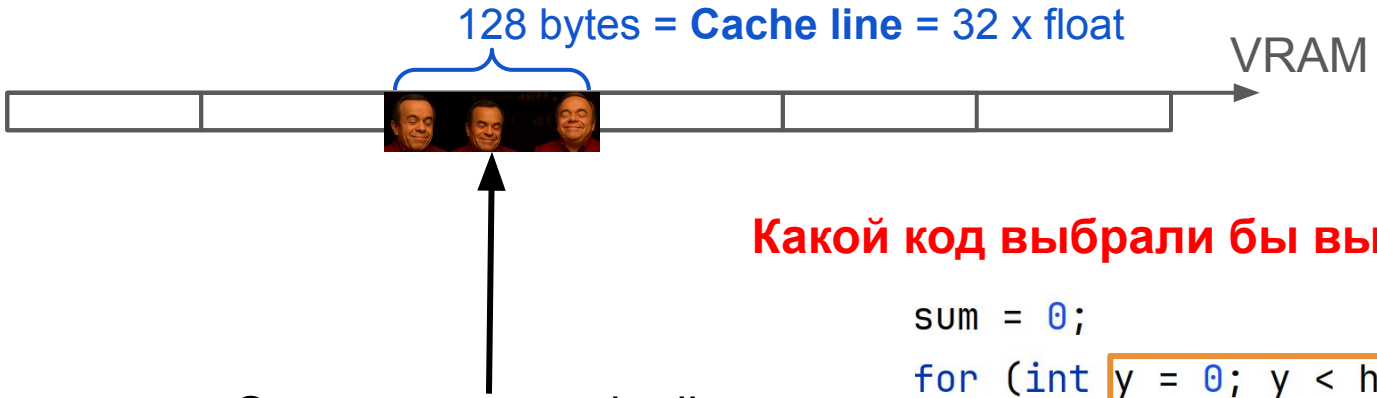
```
sum = 0;
for (int x = 0; x < width; ++x) {
    y =
        sum += image[y * width + x];
    }
}
```




```
sum = 0;
for (int y = 0; y < height; ++y) {
    x =                                  {
        sum += image[y * width + x];
    }
}
```



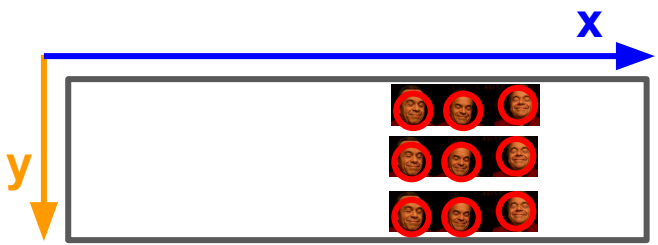
Архитектура VRAM (**coalesced** memory access pattern)



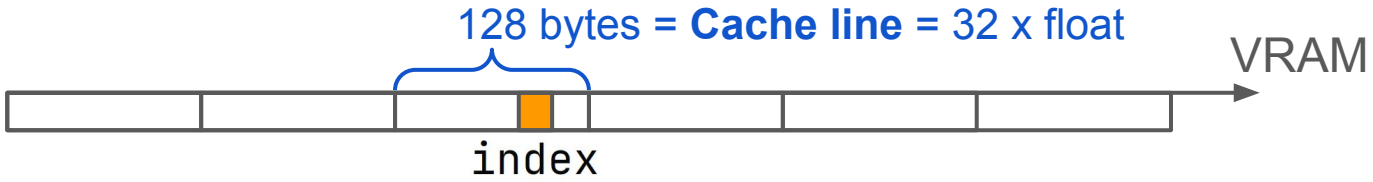
Они загрузят cache line
и распределят в рамках ***warp***-а
байты, произойдет ***broadcast***

Какой код выбрали бы вы?

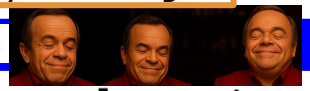
```
sum = 0;
for (int y = 0; y < height; ++y) {
    x =                                    {
        sum += image[y * width + x];
    }
}
```

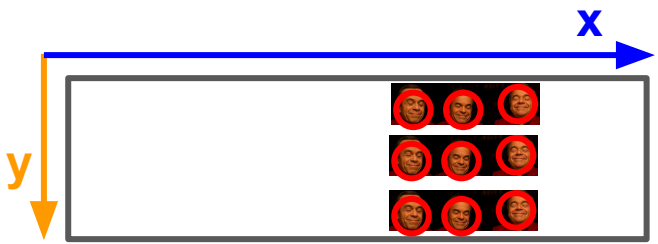


Архитектура VRAM (**coalesced** memory access pattern)

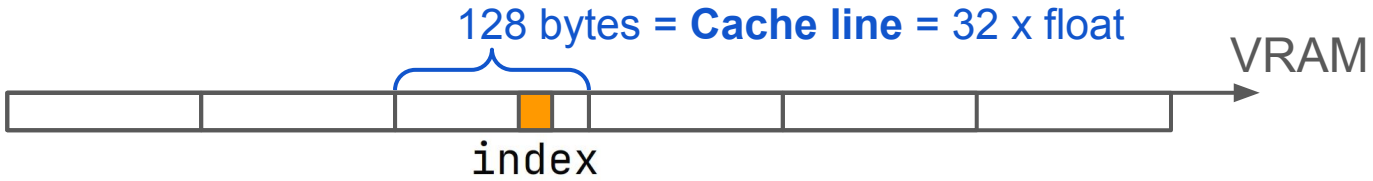


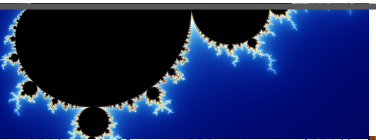
SIMT (thread)

```
sum = 0;
for (int y = 0; y < height; ++y) {
    x =  {
        sum += image[y * width + x];
    }
}
```



Архитектура VRAM (**coalesced** memory access pattern)





SIMD (data)

```
__m256i maskAll = _mm256_setzero_si256();
__m256i iters = _mm256_setzero_si256();
__m256 xs = _mm256_set1_ps(0.0f);
__m256 ys = _mm256_set1_ps(0.0f);
for (iteration = 0; iteration < MAX_ITERAT
    __m256 xsn = _mm256_add_ps(_mm256_sub
    __m256 ysn = _mm256_add_ps(_mm256_mul
    xs = _mm256_add_ps(_mm256_andnot_ps((
    ys = _mm256_add_ps(_mm256_andnot_ps((

    maskAll = (__m256i) _mm256_or_ps(_mm256_cmp_ps(_mm256_add_ps(_mm256_mul_ps(xs, xs), _mm256_mul_ps(ys,
    iters = _mm256_add_epi16(iters, _mm256_andnot_si256(maskAll, _mm256_set1_epi16(1)));
    int mask = _mm256_movemask_epi8(maskAll);
    if (mask == (int) 0xffffffff) {
        break;
    }
```

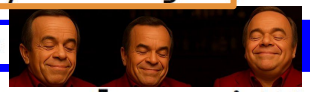
short b0	short b1	short b2	short b3

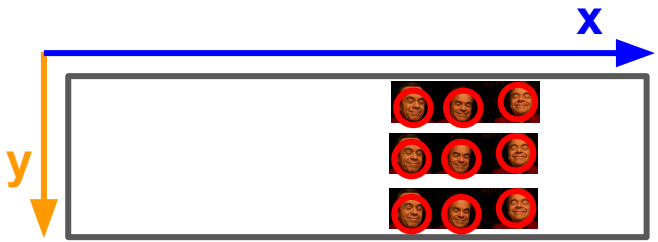
+

short c0	short c1	short c2	short c3

SIMT (thread)

```
sum = 0;
for (int y = 0; y < height; ++y) {
    x =                                    {
        sum += image[y * width + x];
    }
}
```



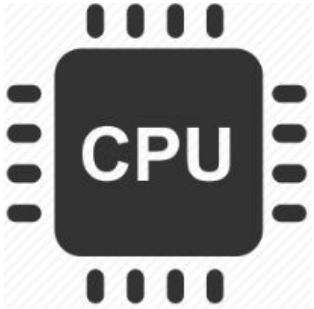


Глава 3: Общая картина ЭВМ-архитектуры

CPU - RAM - PCI-E - VRAM - GPU

Архитектура

$100 \cdot 10^9$ FLOPS

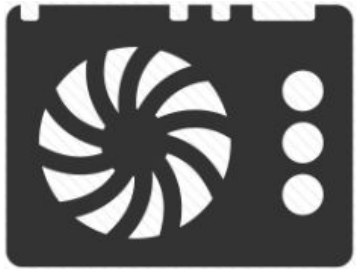


40 GB/s

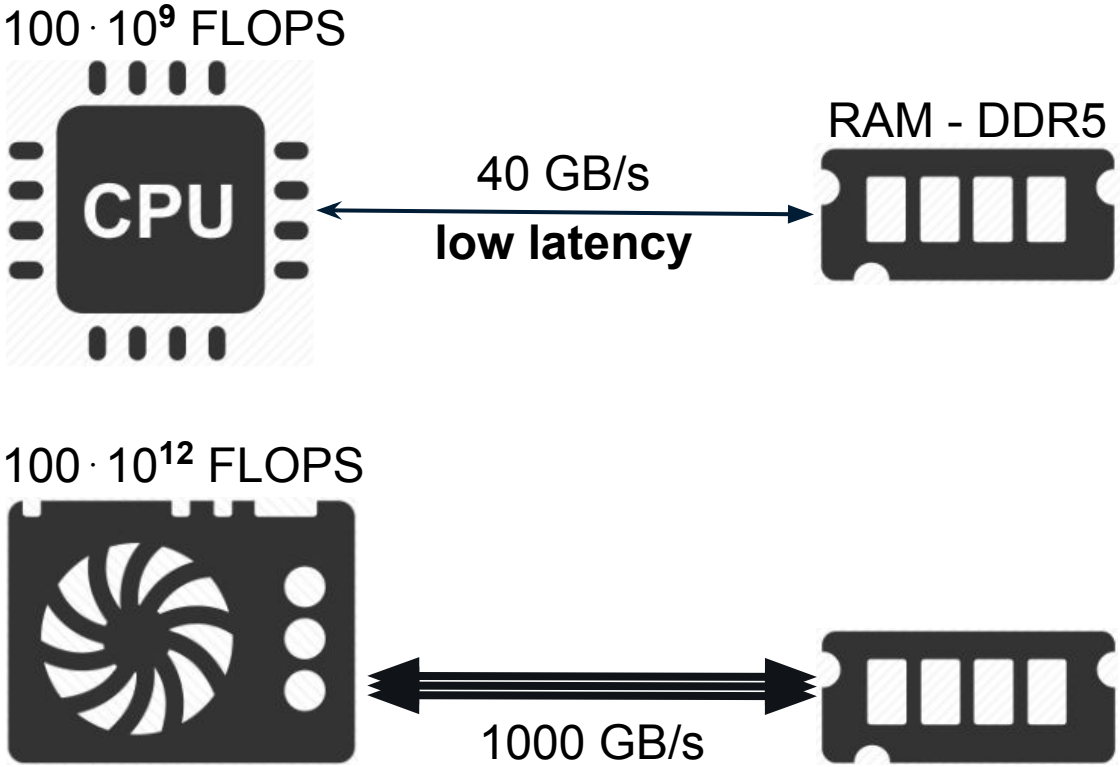
low latency

RAM - DDR5

$100 \cdot 10^{12}$ FLOPS



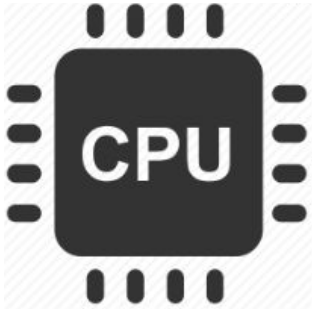
Архитектура



Архитектура

Где лучше исполнять OS?
(например реагировать на клики пользователя)
Какие есть метрики качества?

$100 \cdot 10^9$ FLOPS

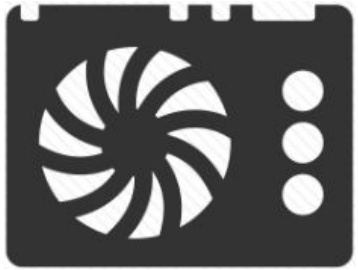


40 GB/s

low latency

RAM - DDR5

$100 \cdot 10^{12}$ FLOPS



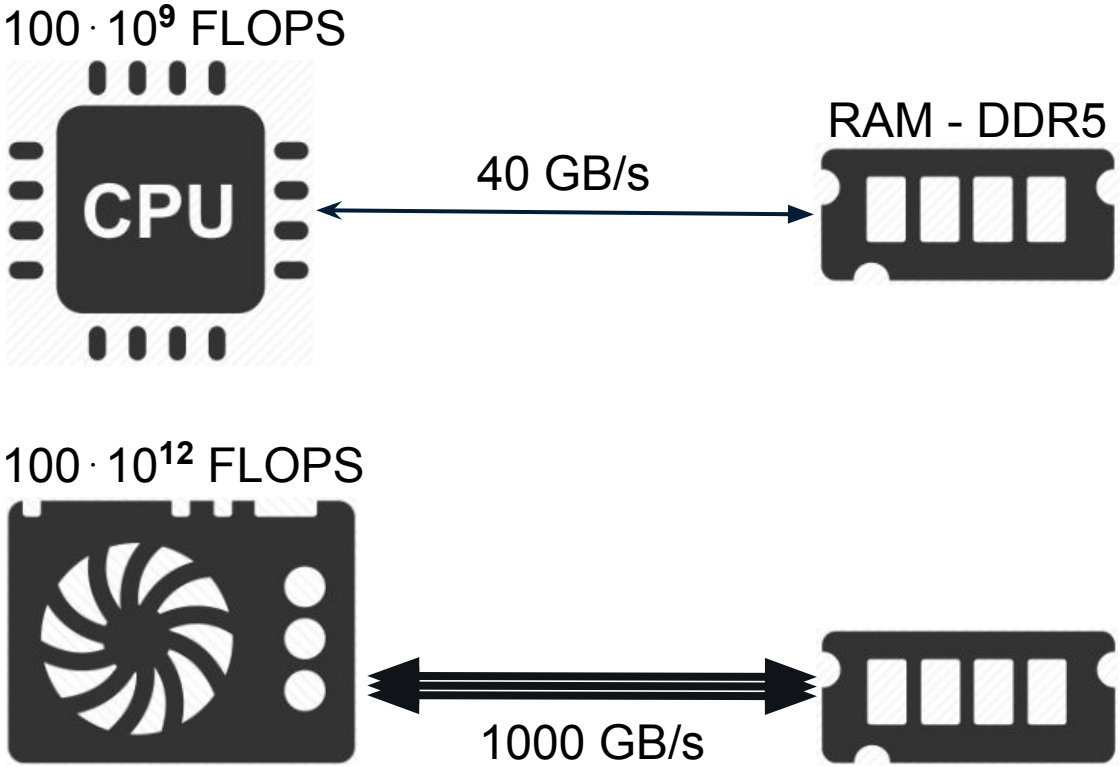
1000 GB/s

(**x25 раз больше**)

VRAM - GDDR6 или HBM (**high bandwidth**)

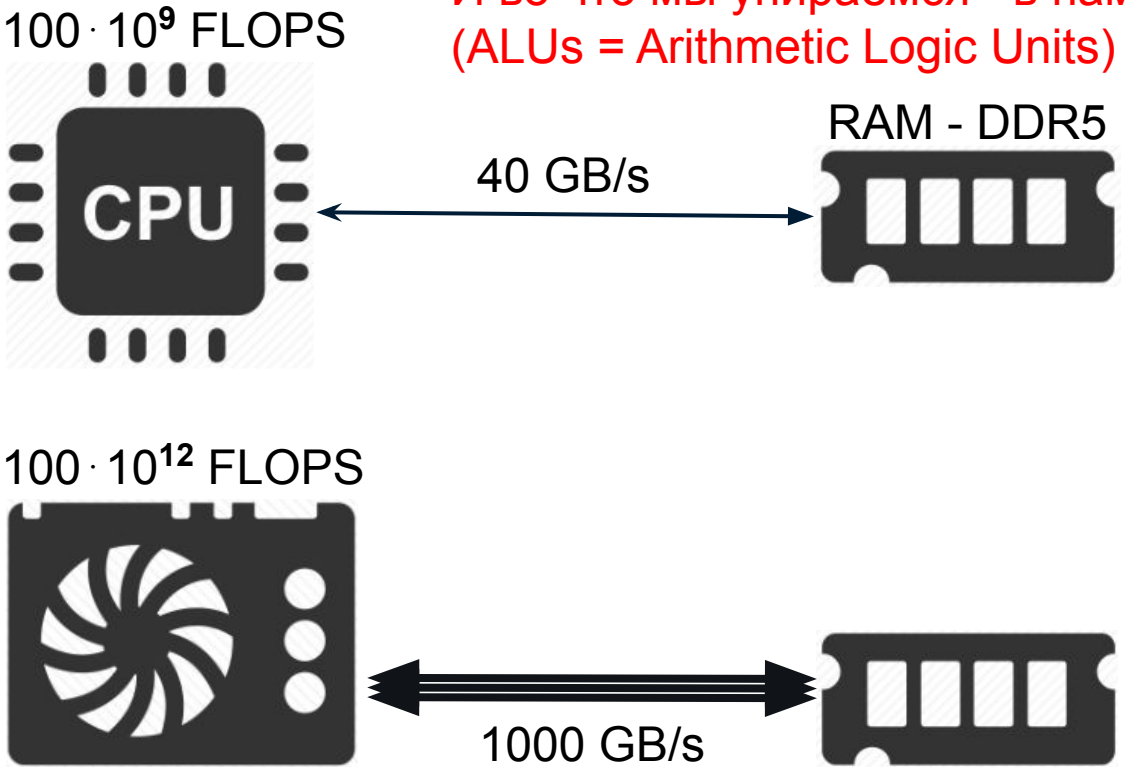
Архитектура

Где быстрее сложить два массива чисел?



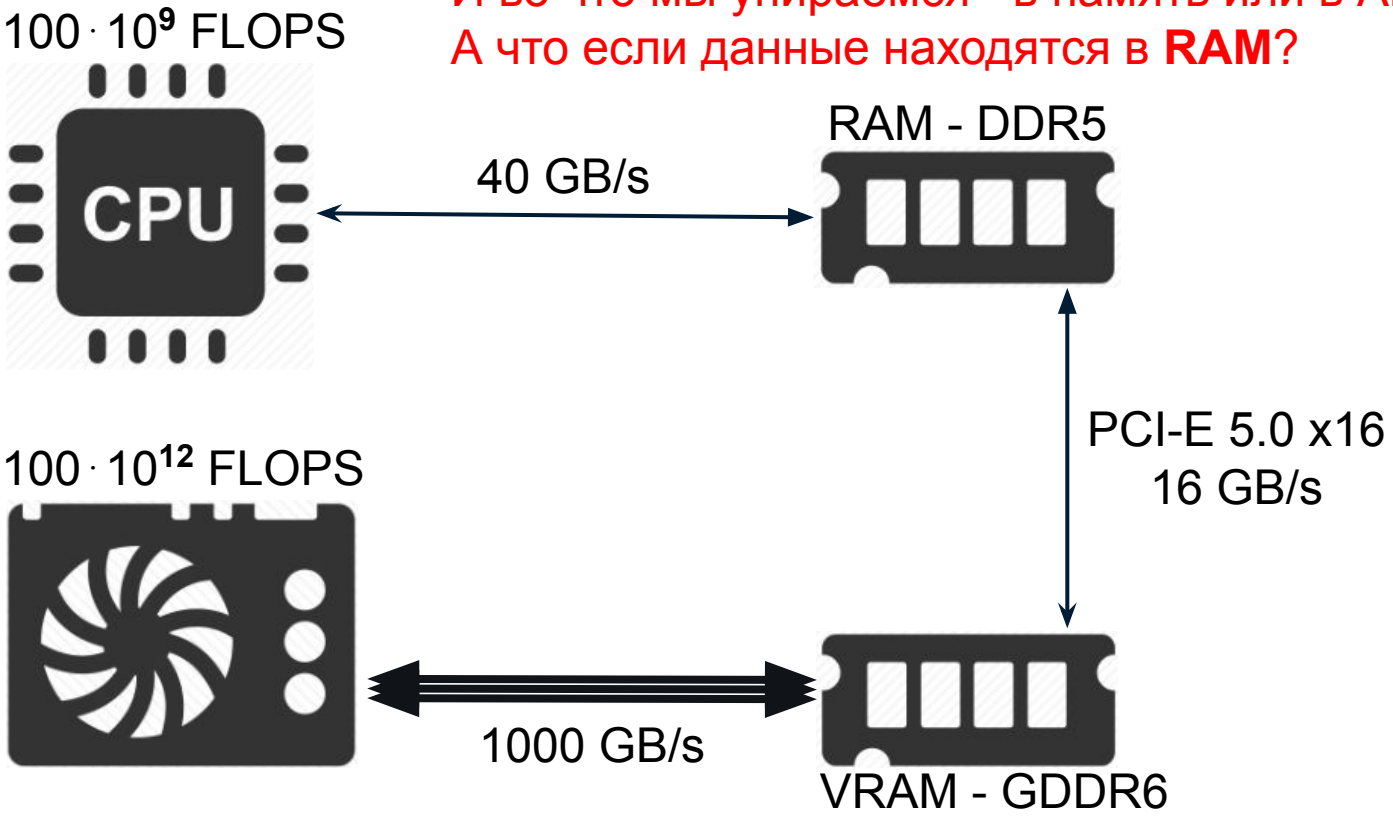
Архитектура

Где быстрее сложить два массива чисел?
И во что мы упираемся - в память или в ALUs?
(ALUs = Arithmetic Logic Units)



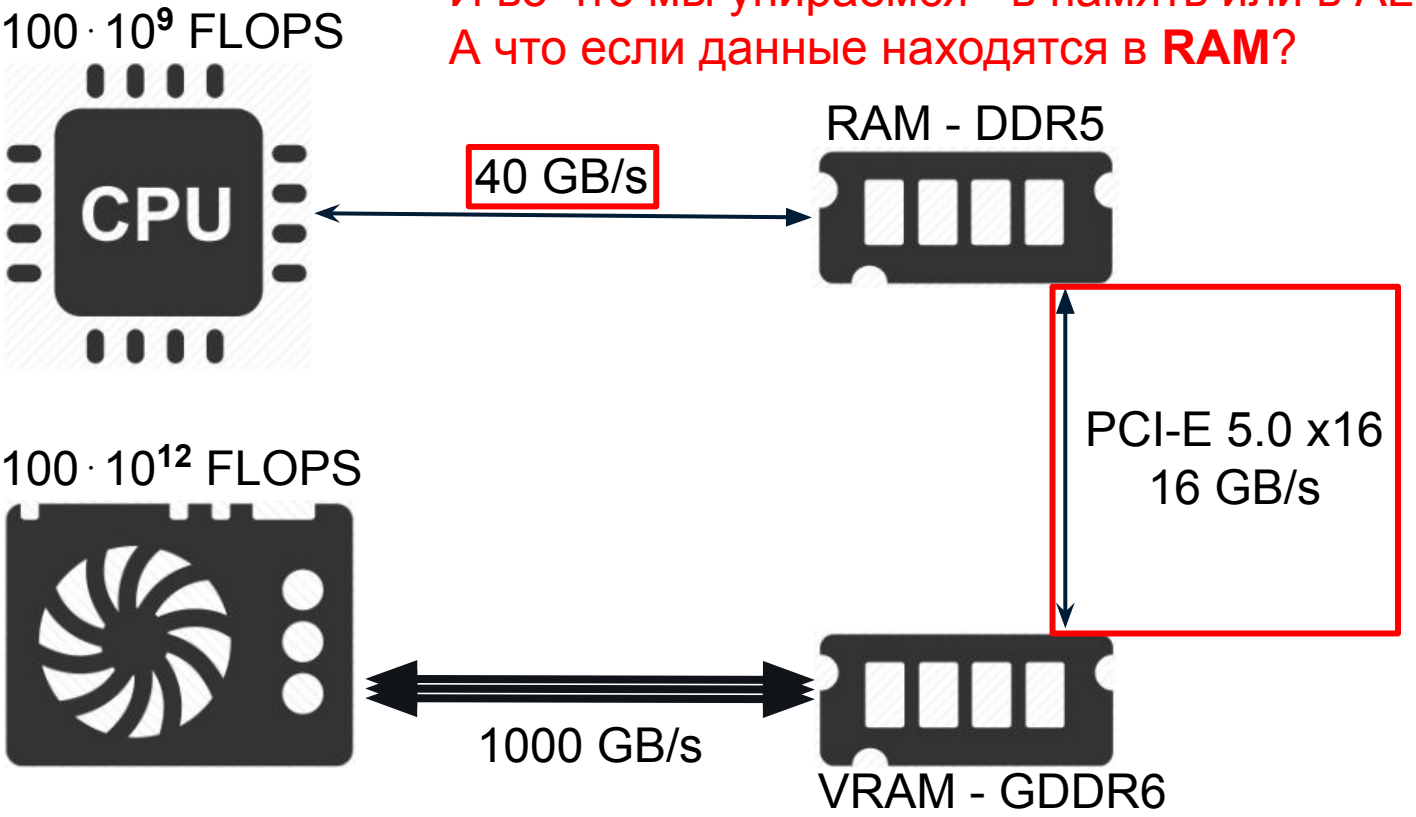
Архитектура

Где быстрее сложить два массива чисел?
И во что мы упираемся - в память или в ALUs?
А что если данные находятся в **RAM**?

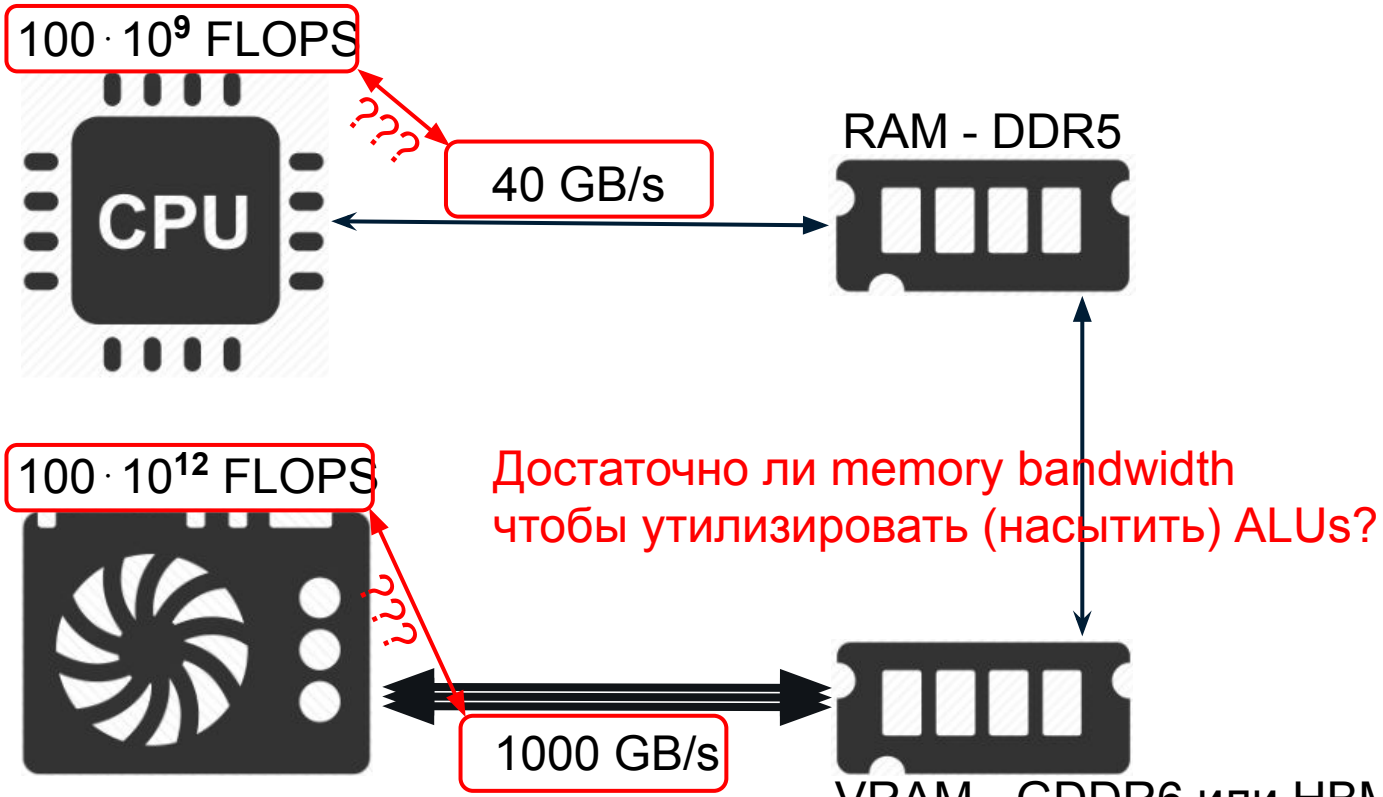


Архитектура

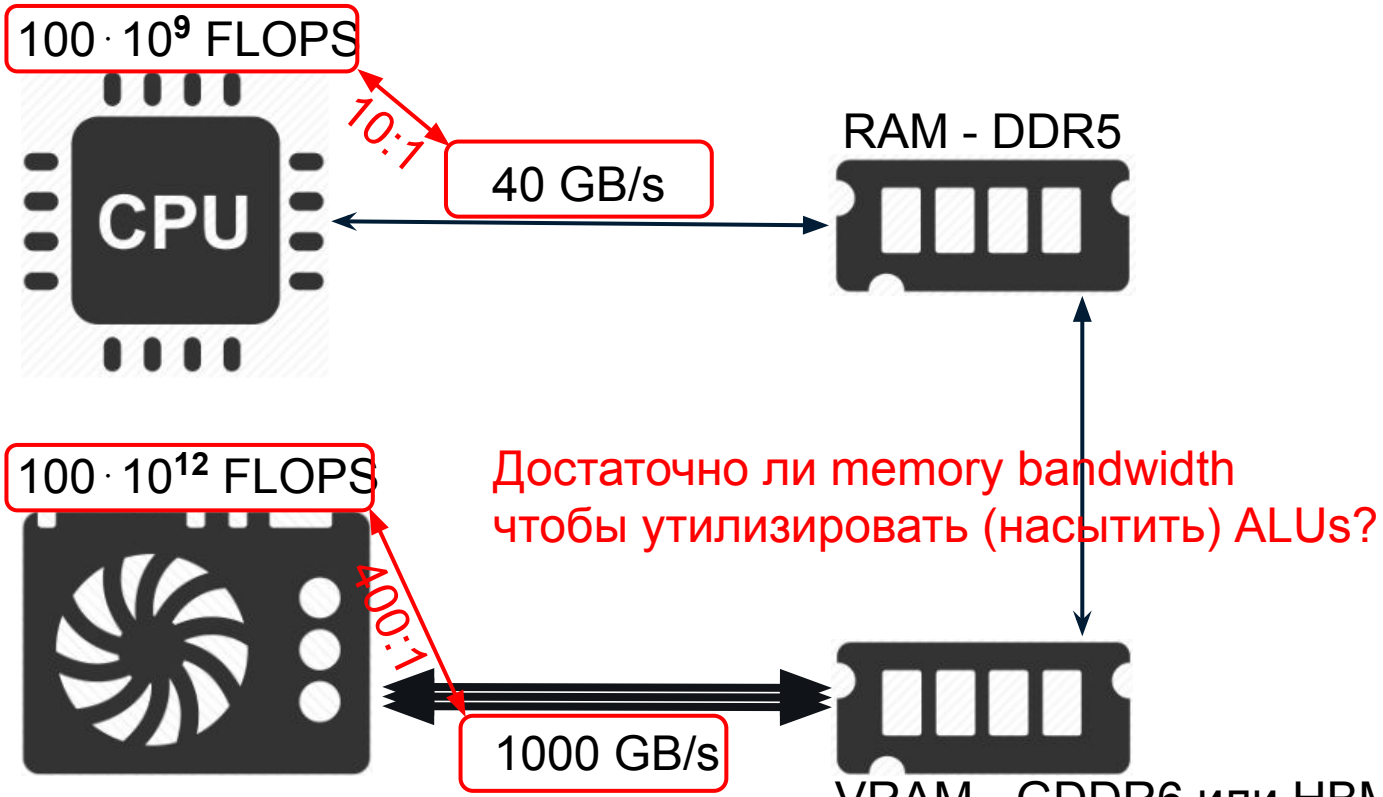
Где быстрее сложить два массива чисел?
И во что мы упираемся - в память или в ALUs?
А что если данные находятся в **RAM**?



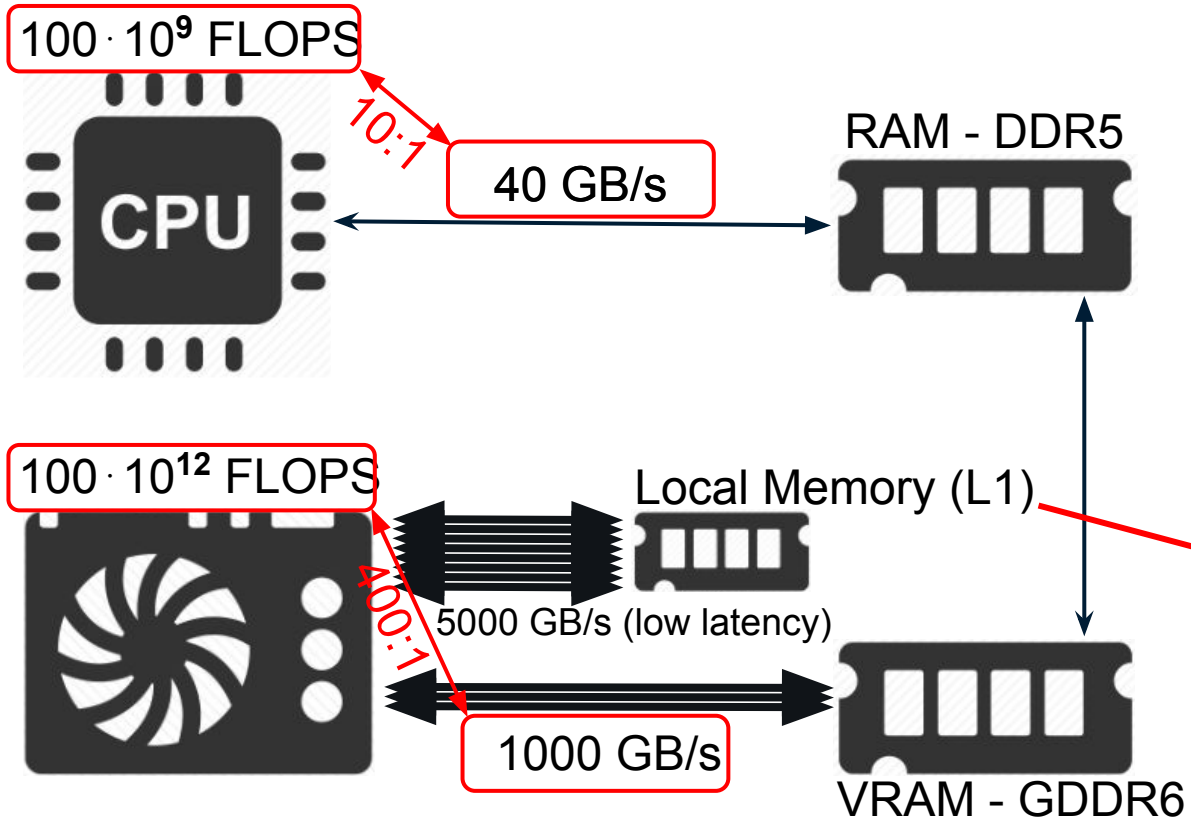
Архитектура



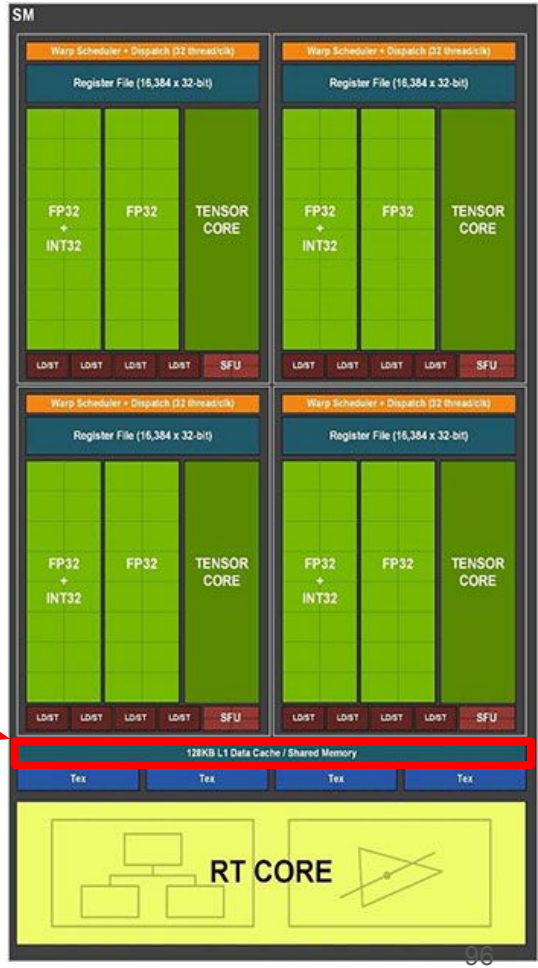
Архитектура



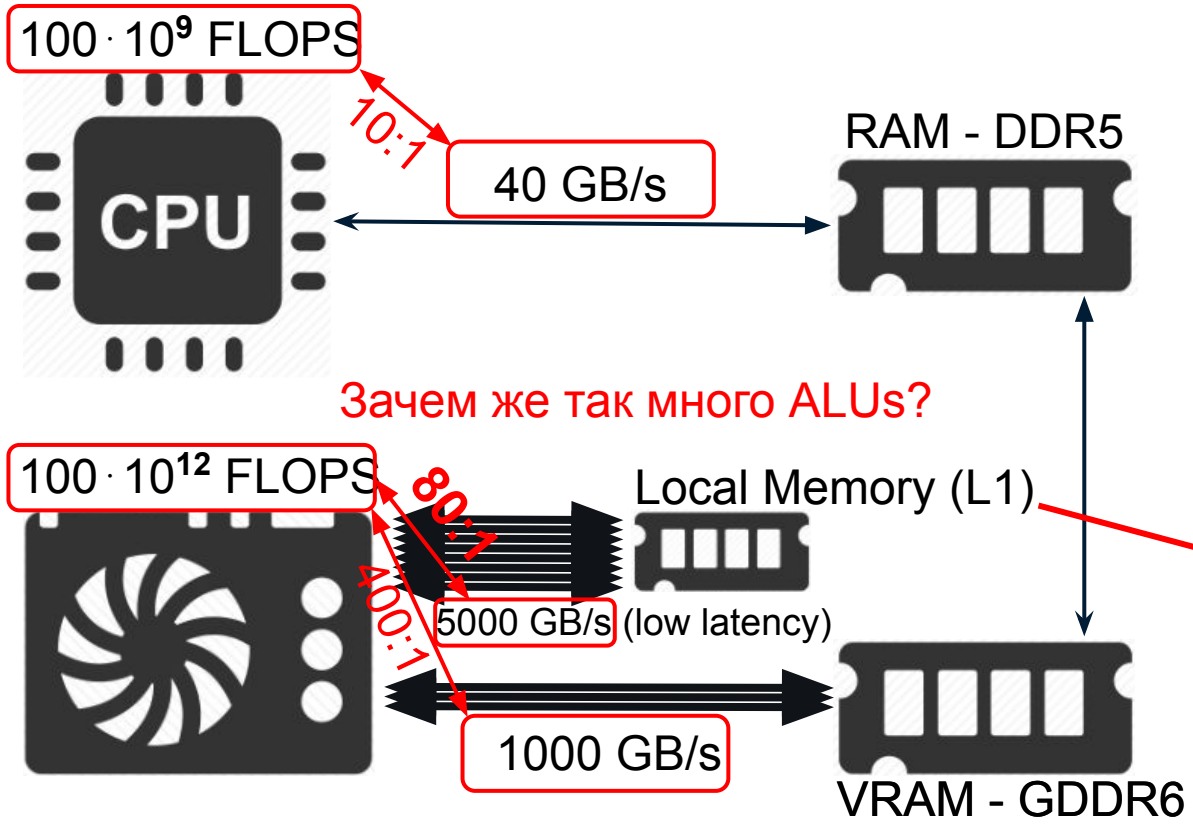
Архитектура



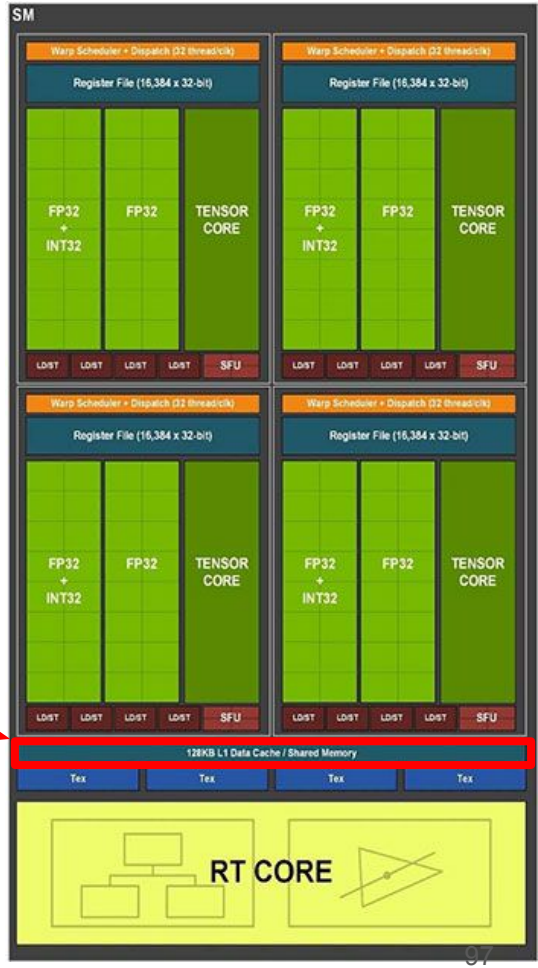
SM - Streaming Multiprocessor



Архитектура



SM - Streaming Multiprocessor



Архитектура

$100 \cdot 10^9$ FLOPS

CPU

40 GB/s

RAM - DDR5

Можно ли ее использовать так же как VRAM?

$100 \cdot 10^{12}$ FLOPS

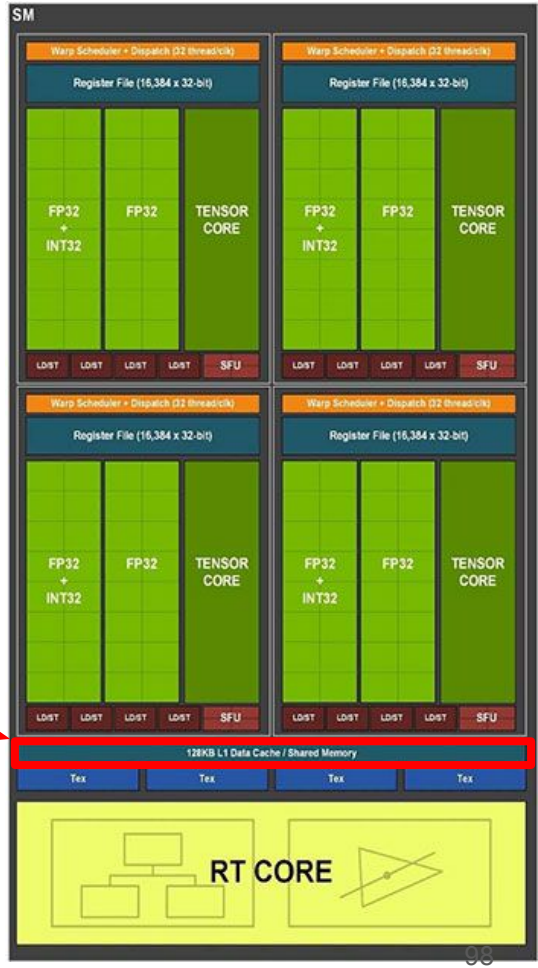
Local Memory (L1)

5000 GB/s (low latency)

1000 GB/s

VRAM - GDDR6

SM - Streaming Multiprocessor



$100 \cdot 10^{12}$ FLOPS

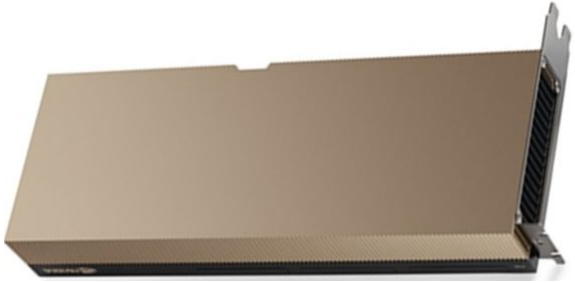
Local Memory (L1)

5000 GB/s (low latency)

1000 GB/s

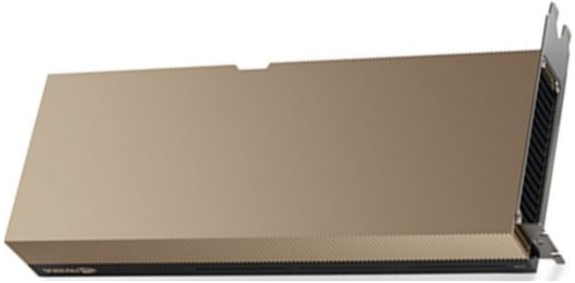
VRAM - GDDR6

Архитектура GPU



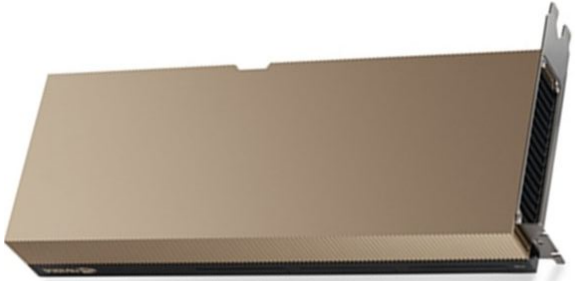
1) Десятки тысяч ядер (много **FLOPs**):
   - сгруппированы по 32 в **warp**-ы (по 64 в **wavefront**-ы у AMD)

Архитектура GPU



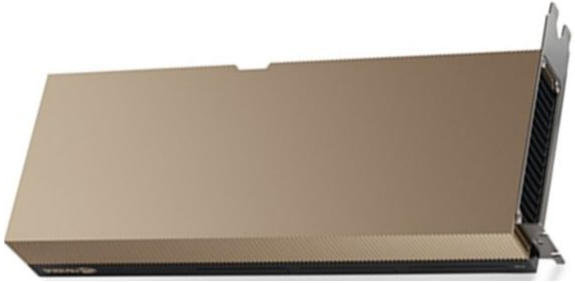
1) Десятки тысяч ядер (много **FLOPs**):
   - сгруппированы по 32 в **warp**-ы (по 64 в **wavefront**-ы у AMD)
   - но слабые ядра

Архитектура GPU



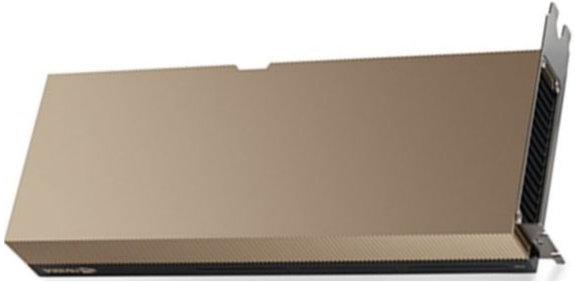
1) Десятки тысяч ядер (много **FLOPs**):
   - сгруппированы по 32 в **warp**-ы (по 64 в **wavefront**-ы у AMD)
   - но слабые ядра
   - но у warp-а общий **instruction pointer** (опасность **code divergence**)

Архитектура GPU



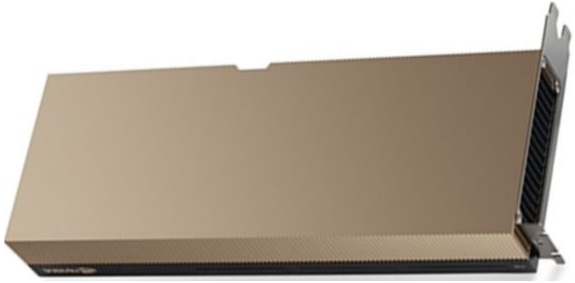
1) Десятки тысяч ядер (много **FLOPs**):
   - сгруппированы по 32 в **warp**-ы (по 64 в **wavefront**-ы у AMD)
   - но слабые ядра
   - но у warp-а общий **instruction pointer** (опасность **code divergence**)
   - активен **warp** который не ждет данных из **VRAM** (при высокой **occupancy**)

Архитектура GPU



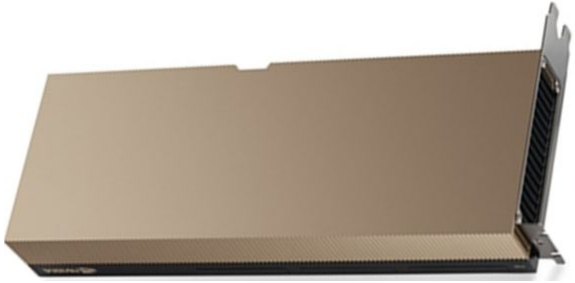
1) Десятки тысяч ядер (много **FLOPs**):
   - сгруппированы по 32 в **warp**-ы (по 64 в **wavefront**-ы у AMD)
   - но слабые ядра
   - но у warp-а общий **instruction pointer** (опасность **code divergence**)
   - активен **warp** который не ждет данных из **VRAM** (при высокой **occupancy**)
2) Огромная пропускная способность **VRAM:**

Архитектура GPU



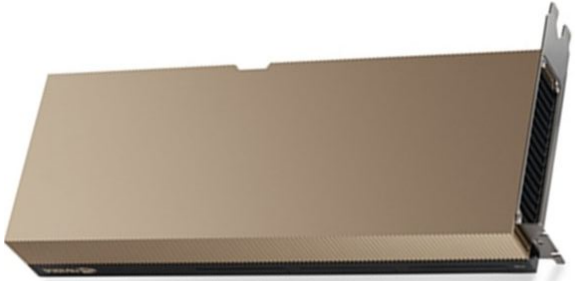
1) Десятки тысяч ядер (много **FLOPs**):
   - сгруппированы по 32 в **warp**-ы (по 64 в **wavefront**-ы у AMD)
   - но слабые ядра
   - но у warp-а общий **instruction pointer** (опасность **code divergence**)
   - активен **warp** который не ждет данных из **VRAM** (при высокой **occupancy**)
2) Огромная пропускная способность **VRAM:**
   - но большая задержка (**latency**)

Архитектура GPU



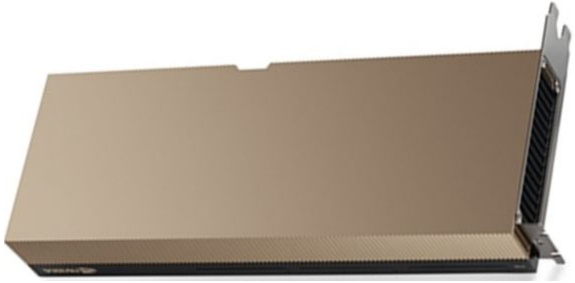
1) Десятки тысяч ядер (много **FLOPs**):
   - сгруппированы по 32 в **warp**-ы (по 64 в **wavefront**-ы у AMD)
   - но слабые ядра
   - но у warp-а общий **instruction pointer** (опасность **code divergence**)
   - активен **warp** который не ждет данных из **VRAM** (при высокой **occupancy**)
2) Огромная пропускная способность **VRAM:**
   - но большая задержка (**latency**)
   - скрывается за счет **сверх-SMT**/Hyper-Threading (при высокой **occupancy**)

Архитектура GPU



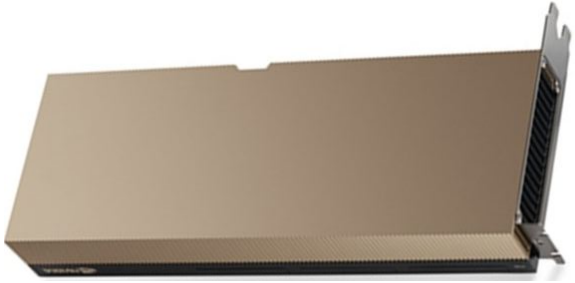
1) Десятки тысяч ядер (много **FLOPs**):
   - сгруппированы по 32 в **warp**-ы (по 64 в **wavefront**-ы у AMD)
   - но слабые ядра
   - но у warp-а общий **instruction pointer** (опасность **code divergence**)
   - активен **warp** который не ждет данных из **VRAM** (при высокой **occupancy**)
2) Огромная пропускная способность **VRAM:**
   - но большая задержка (**latency**)
   - скрывается за счет **сверх-SMT**/Hyper-Threading (при высокой **occupancy**)
   - транзакции разбиты на 128-байтные **cache line**

Архитектура GPU



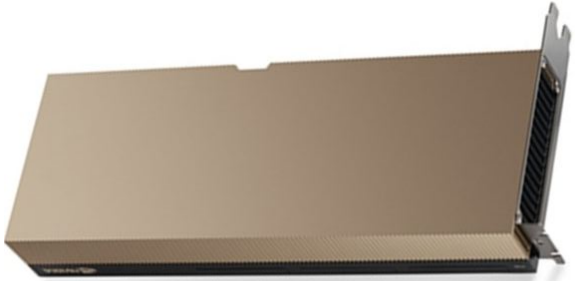
1) Десятки тысяч ядер (много **FLOPs**):
   - сгруппированы по 32 в **warp**-ы (по 64 в **wavefront**-ы у AMD)
   - но слабые ядра
   - но у warp-а общий **instruction pointer** (опасность **code divergence**)
   - активен **warp** который не ждет данных из **VRAM** (при высокой **occupancy**)
2) Огромная пропускная способность **VRAM:**
   - но большая задержка (**latency**)
   - скрывается за счет **сверх-SMT**/Hyper-Threading (при высокой **occupancy**)
   - транзакции разбиты на 128-байтные **cache line**
   - но при **un-coalesced** memory access pattern - малая проп. способность

Архитектура GPU



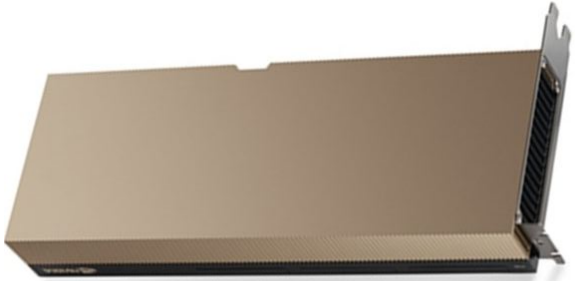
1) Десятки тысяч ядер (много **FLOPs**):
   - сгруппированы по 32 в **warp**-ы (по 64 в **wavefront**-ы у AMD)
   - но слабые ядра
   - но у warp-а общий **instruction pointer** (опасность **code divergence**)
   - активен **warp** который не ждет данных из **VRAM** (при высокой **occupancy**)
2) Огромная пропускная способность **VRAM:**
   - но большая задержка (**latency**)
   - скрывается за счет **сверх-SMT**/Hyper-Threading (при высокой **occupancy**)
   - транзакции разбиты на 128-байтные **cache line**
   - но при **un-coalesced** memory access pattern - малая проп. способность
3) Локальная память (**local/shared memory**):

Архитектура GPU



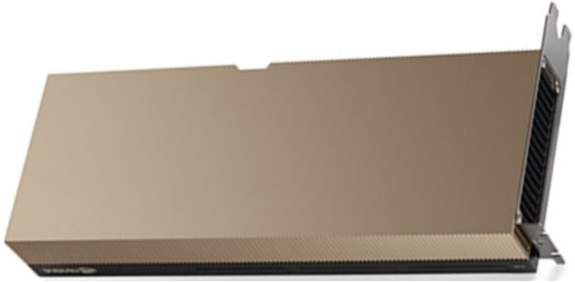
1) Десятки тысяч ядер (много **FLOPs**):
   - сгруппированы по 32 в **warp**-ы (по 64 в **wavefront**-ы у AMD)
   - но слабые ядра
   - но у warp-а общий **instruction pointer** (опасность **code divergence**)
   - активен **warp** который не ждет данных из **VRAM** (при высокой **occupancy**)
2) Огромная пропускная способность **VRAM:**
   - но большая задержка (**latency**)
   - скрывается за счет **сверх-SMT**/Hyper-Threading (при высокой **occupancy**)
   - транзакции разбиты на 128-байтные **cache line**
   - но при **un-coalesced** memory access pattern - малая проп. способность
3) Локальная память (**local/shared memory**):
   - **титаническая** пропускная способность + низкая задержка

Архитектура GPU



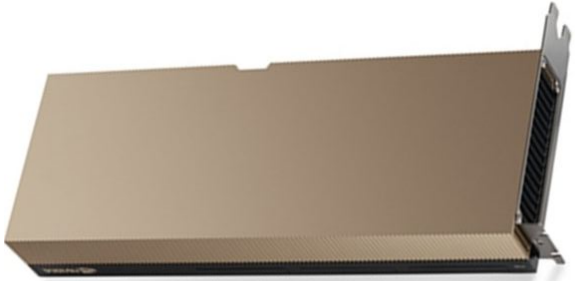
1) Десятки тысяч ядер (много **FLOPs**):
   - сгруппированы по 32 в **warp**-ы (по 64 в **wavefront**-ы у AMD)
   - но слабые ядра
   - но у warp-а общий **instruction pointer** (опасность **code divergence**)
   - активен **warp** который не ждет данных из **VRAM** (при высокой **occupancy**)
2) Огромная пропускная способность **VRAM:**
   - но большая задержка (**latency**)
   - скрывается за счет **сверх-SMT**/Hyper-Threading (при высокой **occupancy**)
   - транзакции разбиты на 128-байтные **cache line**
   - но при **un-coalesced** memory access pattern - малая проп. способность
3) Локальная память (**local/shared memory**):
   - **титаническая** пропускная способность + низкая задержка
   - нет проблемы с coalesced-паттерном

Архитектура GPU



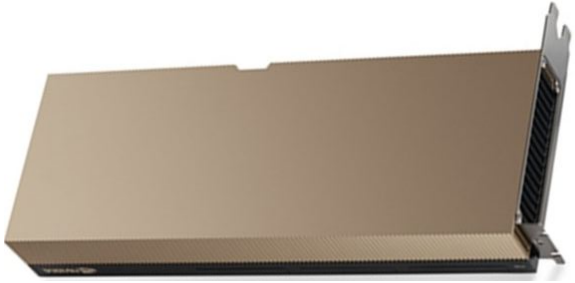
1) Десятки тысяч ядер (много **FLOPs**):
   - сгруппированы по 32 в **warp**-ы (по 64 в **wavefront**-ы у AMD)
   - но слабые ядра
   - но у warp-а общий **instruction pointer** (опасность **code divergence**)
   - активен **warp** который не ждет данных из **VRAM** (при высокой **occupancy**)
2) Огромная пропускная способность **VRAM:**
   - но большая задержка (**latency**)
   - скрывается за счет **сверх-SMT**/Hyper-Threading (при высокой **occupancy**)
   - транзакции разбиты на 128-байтные **cache line**
   - но при **un-coalesced** memory access pattern - малая проп. способность
3) Локальная память (**local/shared memory**):
   - **титаническая** пропускная способность + низкая задержка
   - нет проблемы с coalesced-паттерном
   - **но?**

Архитектура GPU



1) Десятки тысяч ядер (много **FLOPs**):
   - сгруппированы по 32 в **warp**-ы (по 64 в **wavefront**-ы у AMD)
   - но слабые ядра
   - но у warp-а общий **instruction pointer** (опасность **code divergence**)
   - активен **warp** который не ждет данных из **VRAM** (при высокой **occupancy**)
2) Огромная пропускная способность **VRAM:**
   - но большая задержка (**latency**)
   - скрывается за счет **сверх-SMT**/Hyper-Threading (при высокой **occupancy**)
   - транзакции разбиты на 128-байтные **cache line**
   - но при **un-coalesced** memory access pattern - малая проп. способность
3) Локальная память (**local/shared memory**):
   - **титаническая** пропускная способность + низкая задержка
   - нет проблемы с coalesced-паттерном
   - но **небольшая** + **локальная** для каждого ~~**warp**~~ **WorkGroup=Block**

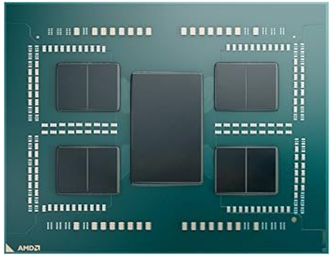
Архитектура GPU



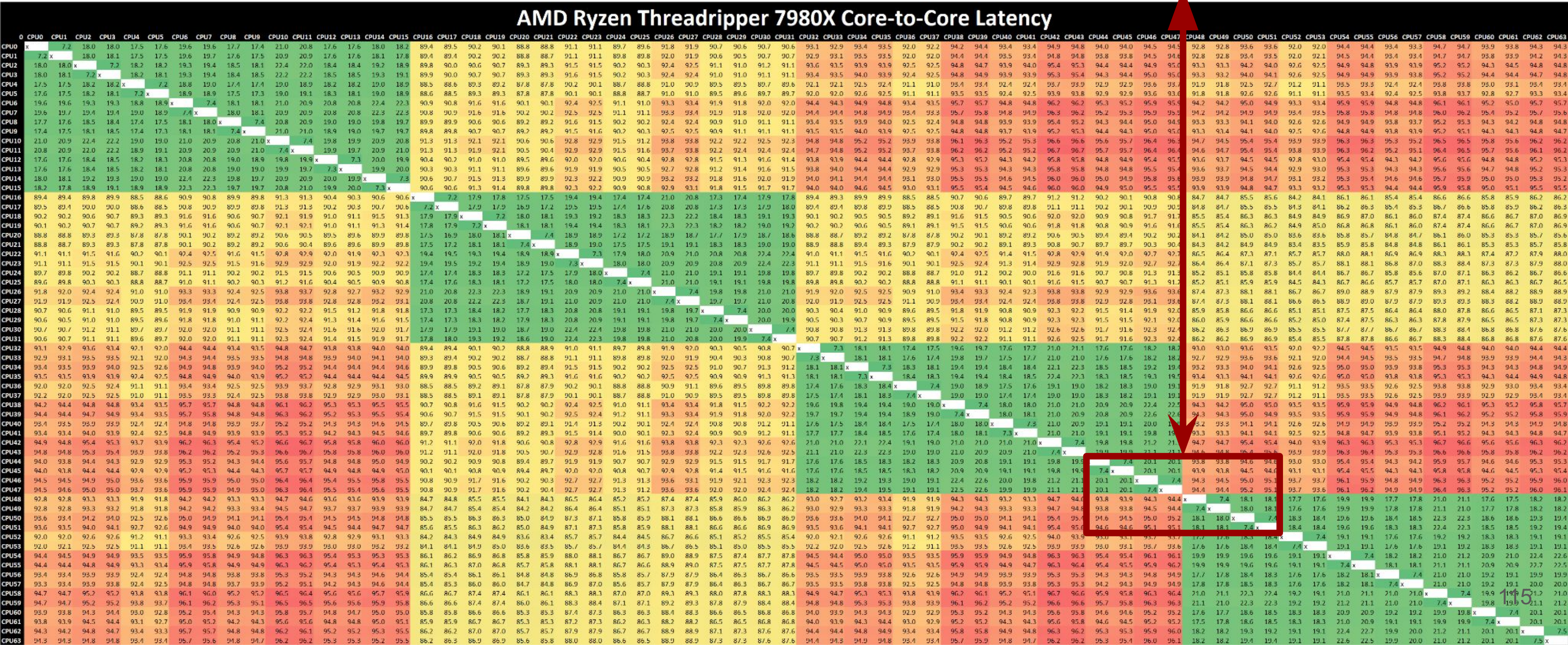
1) Десятки тысяч ядер (много **FLOPs**):
   - сгруппированы по 32 в **warp**-ы (по 64 в **wavefront**-ы у AMD)
   - но слабые ядра
   - но у warp-а общий **instruction pointer** (опасность **code divergence**)
   - активен **warp** который не ждет данных из **VRAM** (при высокой **occupancy**)
2) Огромная пропускная способность **VRAM:**
   - но большая задержка (**latency**)
   - скрывается за счет **сверх-SMT**/Hyper-Threading (при высокой **occupancy**)
   - транзакции разбиты на 128-байтные **cache line**
   - но при **un-coalesced** memory access pattern - малая проп. способность
3) Локальная память (**local/shared memory**):
   - **титаническая** пропускная способность + низкая задержка
   - нет проблемы с coalesced-паттерном
   - но **небольшая** + **локальная** для каждого ~~**warp**~~ **WorkGroup=Block**

Позже мы обсудим ***bank conflicts***

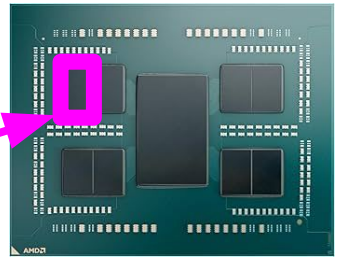
И в этом секрет успеха! Рецепт к **масштабируемости**!



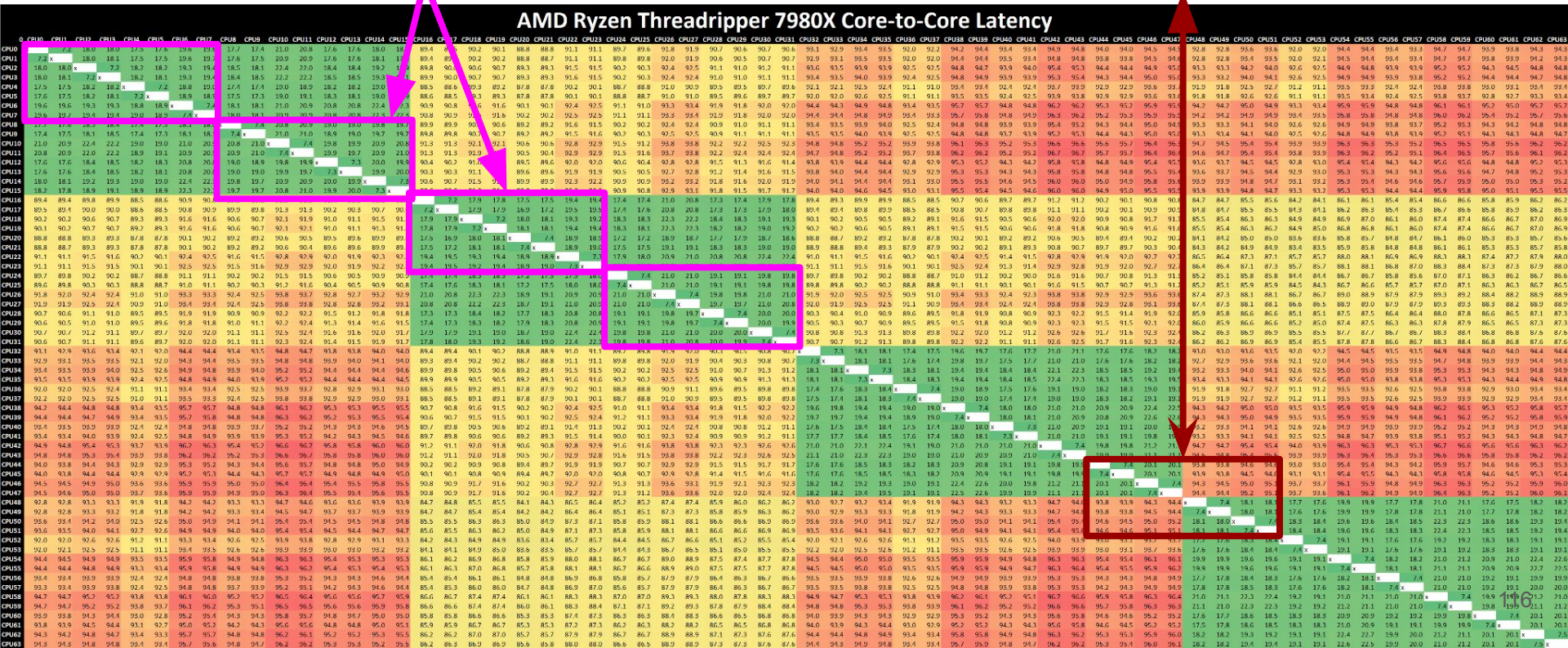
x	7.4	20.1	20.1	93.8	93.8	94.6	94.6
7.4	x	20.1	20.1	93.9	93.8	94.5	94.6
20.1	20.1	x	7.4	94.3	94.5	95.0	95.1
20.1	20.1	7.4	x	94.4	94.4	95.2	95.1
93.8	93.9	94.3	94.4	x	7.4	18.1	18.1
93.8	93.8	94.5	94.4	7.4	x	18.0	18.1
94.6	94.5	95.0	95.2	18.1	18.0	x	7.4
94.6	94.6	95.1	95.1	18.1	18.1	7.4	x



64 ядра = 8 x Чиплетов (по 8 ядер)

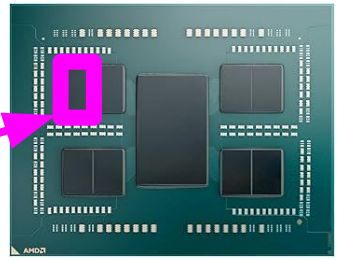


x	7.4	20.1	20.1	93.8	93.8	94.6	94.6
7.4	x	20.1	20.1	93.9	93.8	94.5	94.6
20.1	20.1	x	7.4	94.3	94.5	95.0	95.1
20.1	20.1	7.4	x	94.4	94.4	95.2	95.1
93.8	93.9	94.3	94.4	x	7.4	18.1	18.1
93.8	93.8	94.5	94.4	7.4	x	18.0	18.1
94.6	94.5	95.0	95.2	18.1	18.0	x	7.4
94.6	94.6	95.1	95.1	18.1	18.1	7.4	x



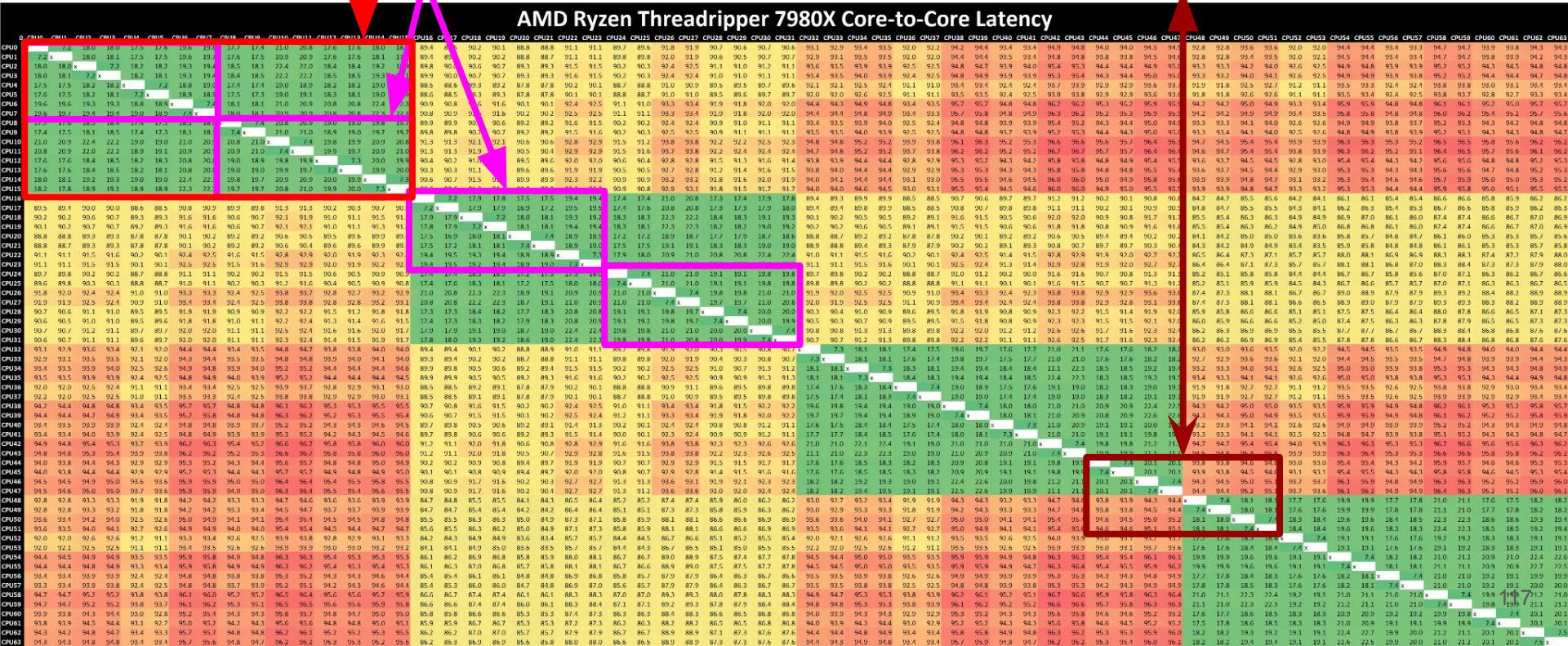
Но почему задержка маленькая группами по 16 ядер?

64 ядра = 8 x Чиплетов (по 8 ядер)



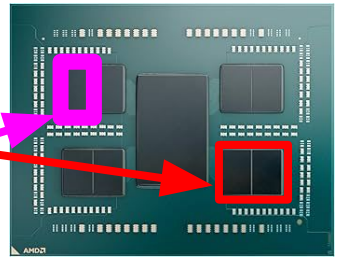
x	7.4	20.1	20.1	93.8	93.8	94.6	94.6
7.4	x	20.1	20.1	93.9	93.8	94.5	94.6
20.1	20.1	x	7.4	94.3	94.5	95.0	95.1
20.1	20.1	7.4	x	94.4	94.4	95.2	95.1
93.8	93.9	94.3	94.4	x	7.4	18.1	18.1
93.8	93.8	94.5	94.4	7.4	x	18.0	18.1
94.6	94.5	95.0	95.2	18.1	18.0	x	7.4
94.6	94.6	95.1	95.1	18.1	18.1	7.4	x

AMD Ryzen Threadripper 7980X Core-to-Core Latency



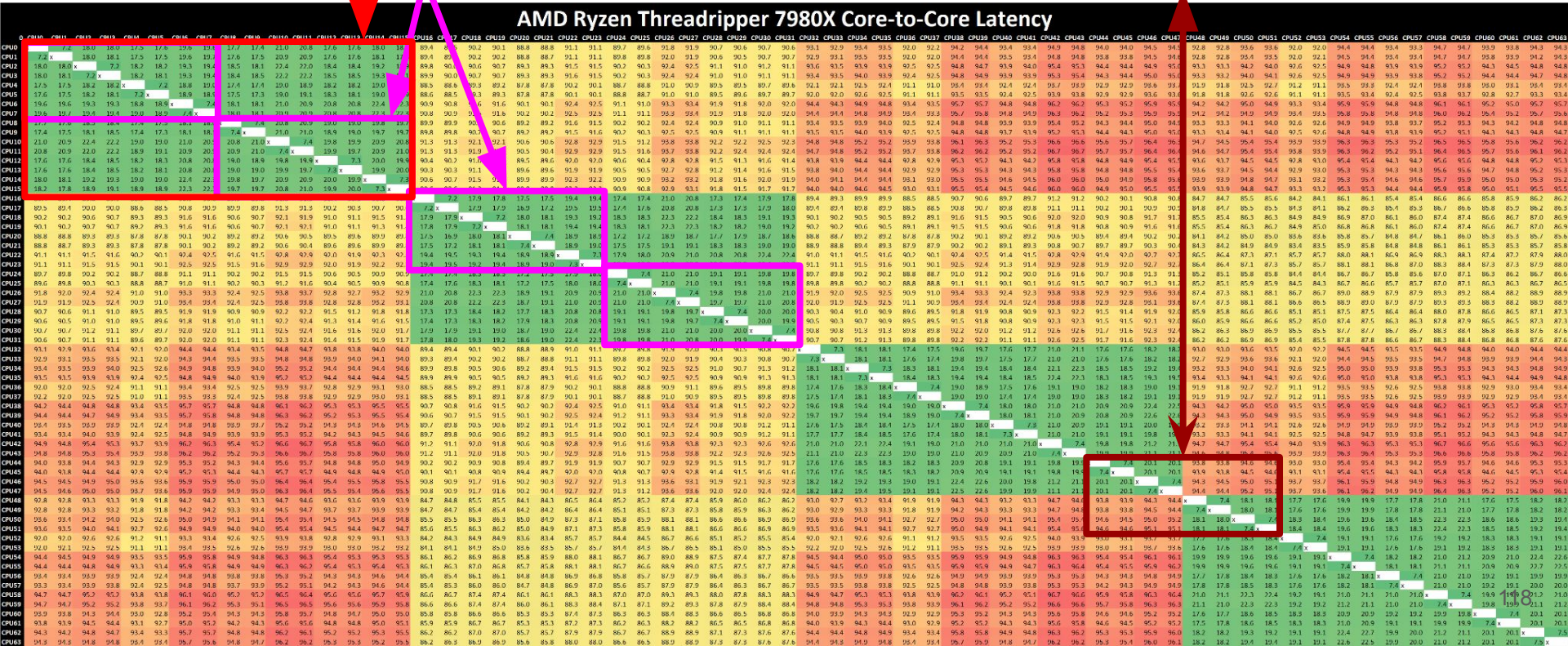
Но почему задержка маленькая группами по 16 ядер?

64 ядра = 8 x Чиплетов (по 8 ядер)



x	7.4	20.1	20.1	93.8	93.8	94.6	94.6
7.4	x	20.1	20.1	93.9	93.8	94.5	94.6
20.1	20.1	x	7.4	94.3	94.5	95.0	95.1
20.1	20.1	7.4	x	94.4	94.4	95.2	95.1
93.8	93.9	94.3	94.4	x	7.4	18.1	18.1
93.8	93.8	94.5	94.4	7.4	x	18.0	18.1
94.6	94.5	95.0	95.2	18.1	18.0	x	7.4
94.6	94.6	95.1	95.1	18.1	18.1	7.4	x

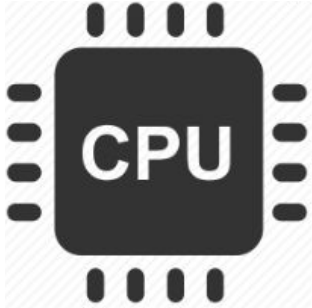
AMD Ryzen Threadripper 7980X Core-to-Core Latency

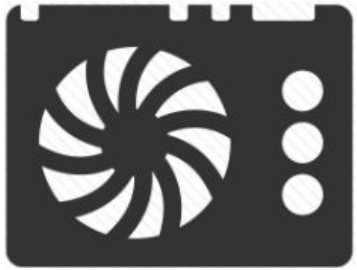


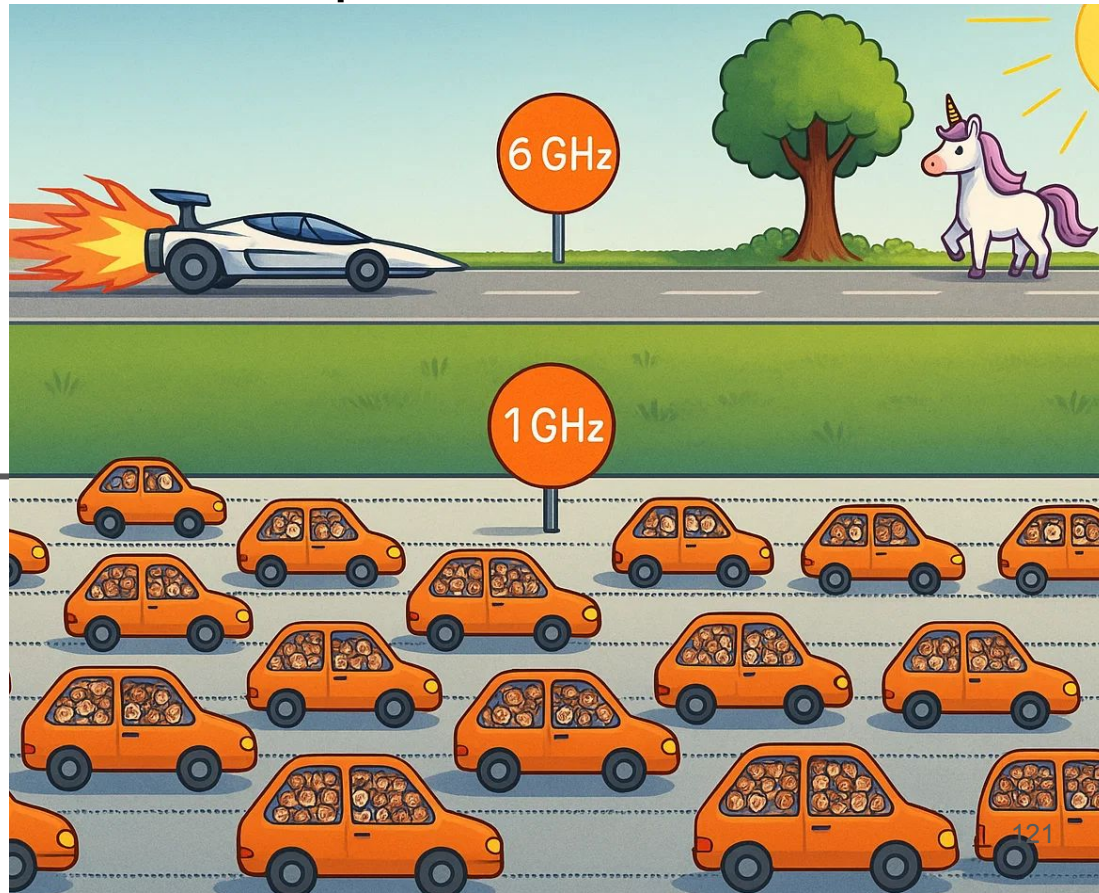

Перерыв!
Streaming
Pubiprocestor
VIDIA

Глава 4: Модель вычислений массового параллелизма

Модель вычислений массового параллелизма

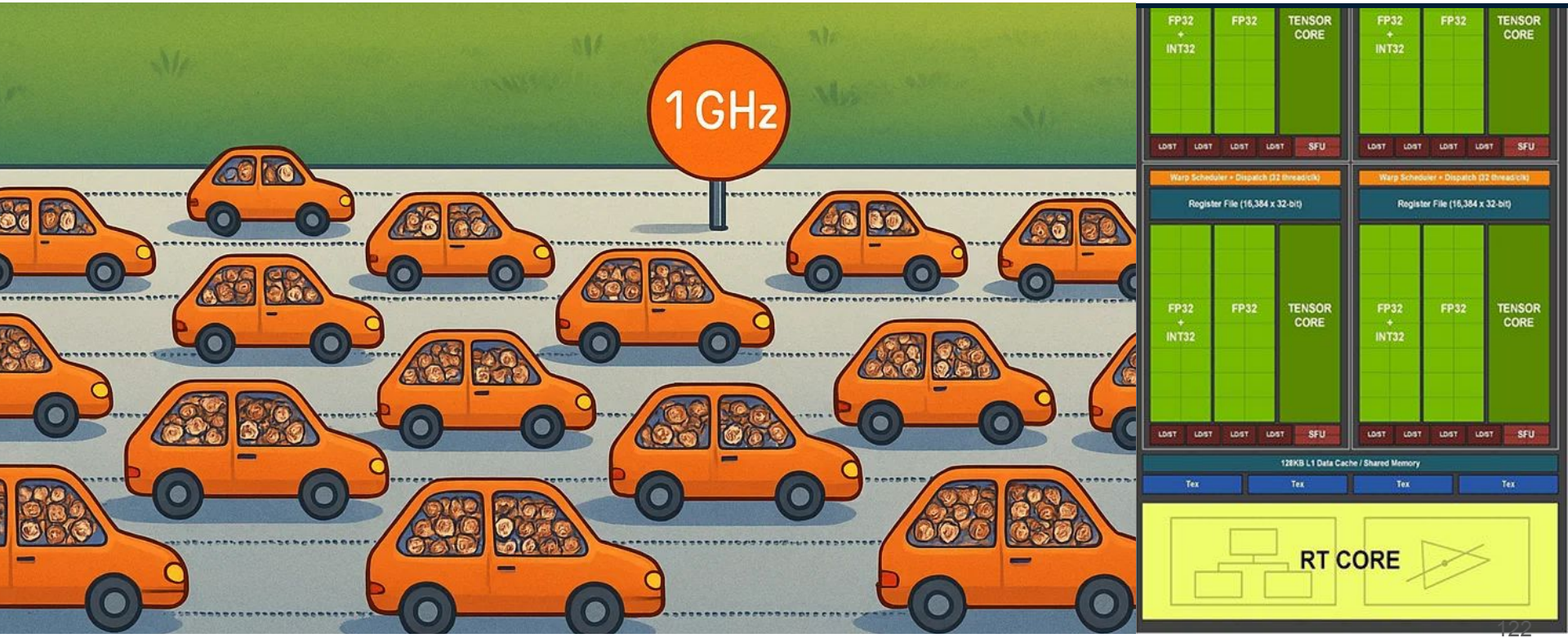






Модель вычислений массового параллелизма

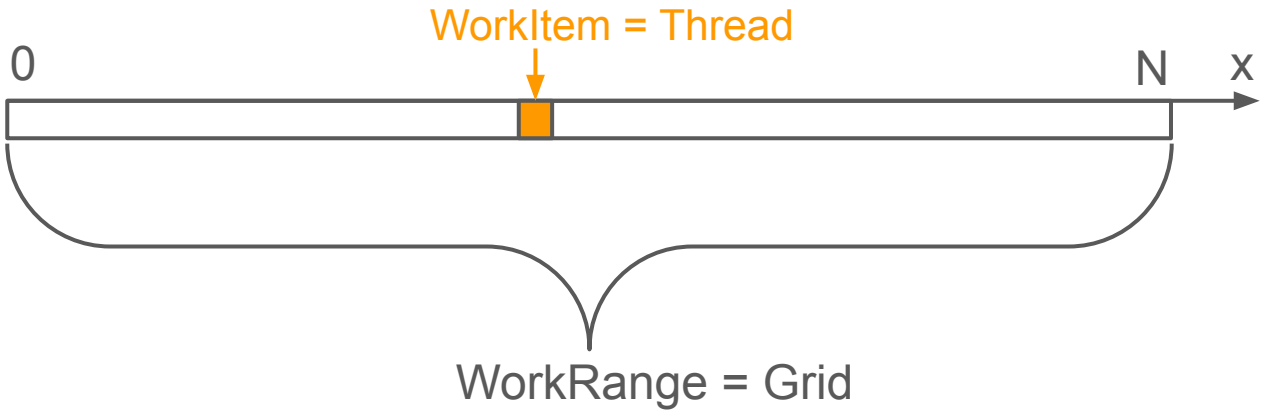
SM - Streaming Multiprocessor



Модель вычислений массового параллелизма

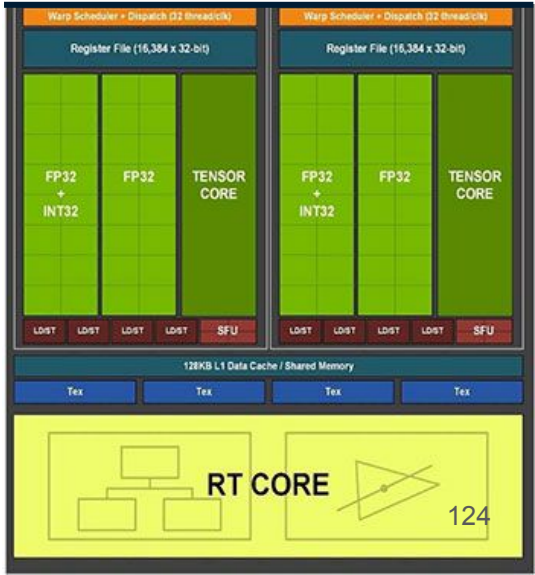
SM - Streaming Multiprocessor

Модель вычислений массового параллелизма

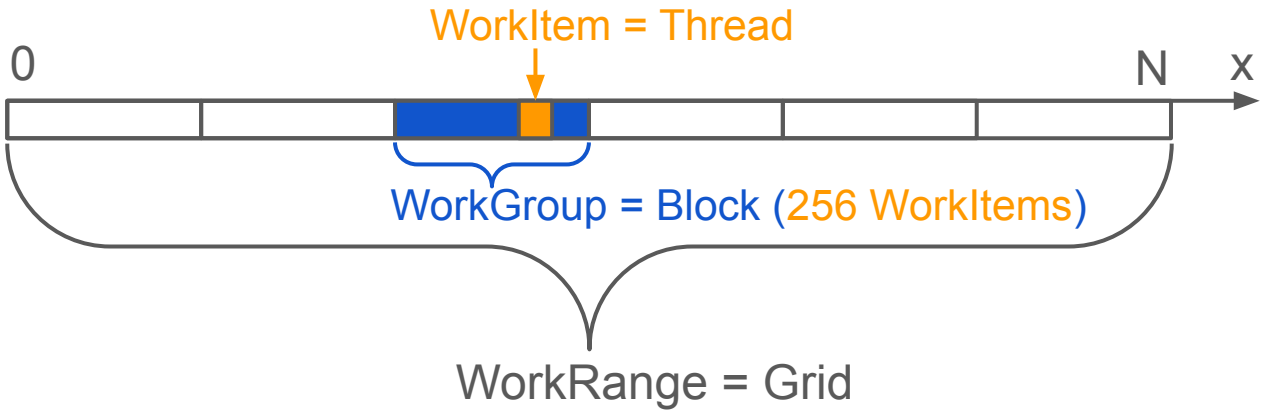


Много SM

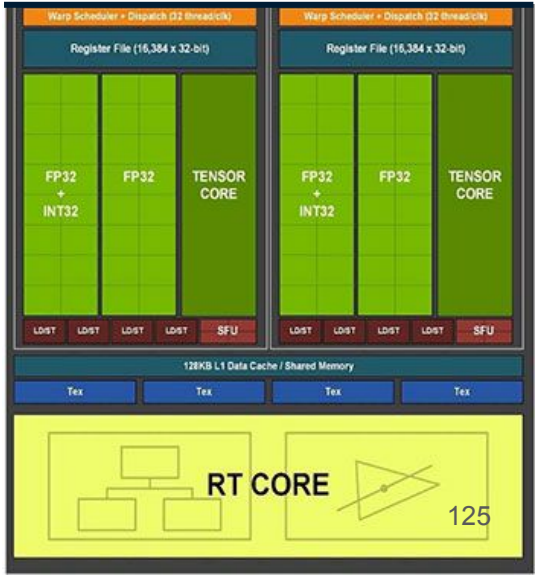
SM - Streaming Multiprocessor



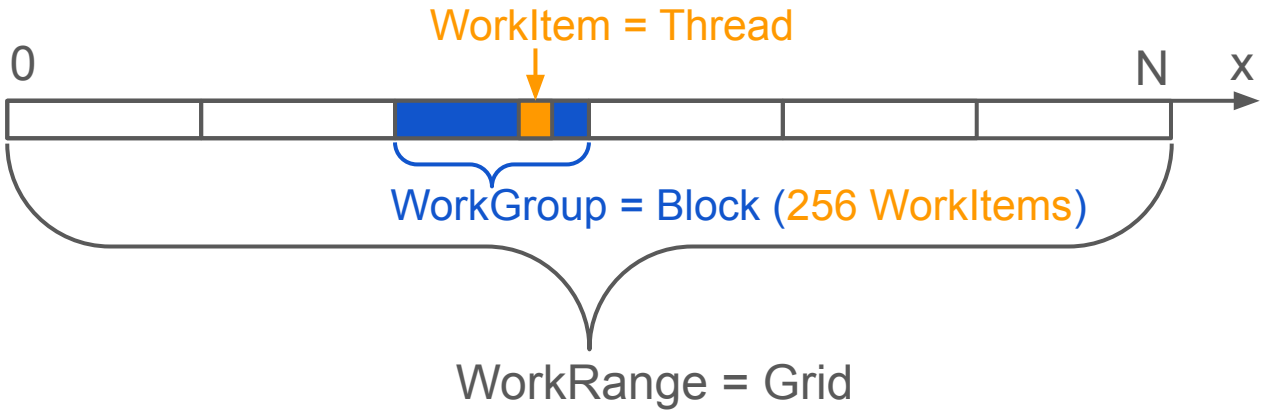
Модель вычислений массового параллелизма



SM - Streaming Multiprocessor

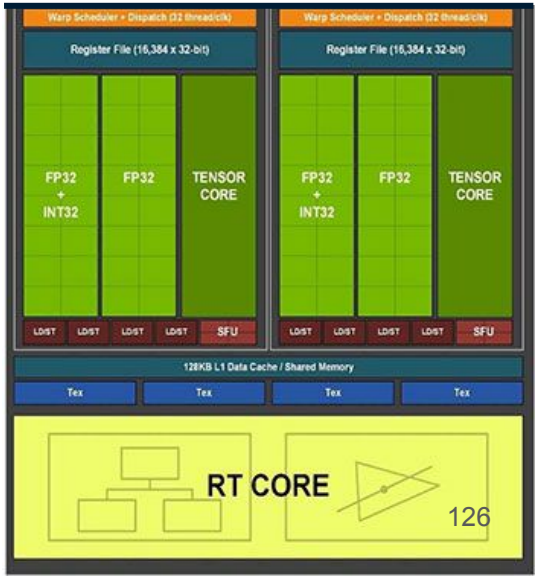


Модель вычислений массового параллелизма

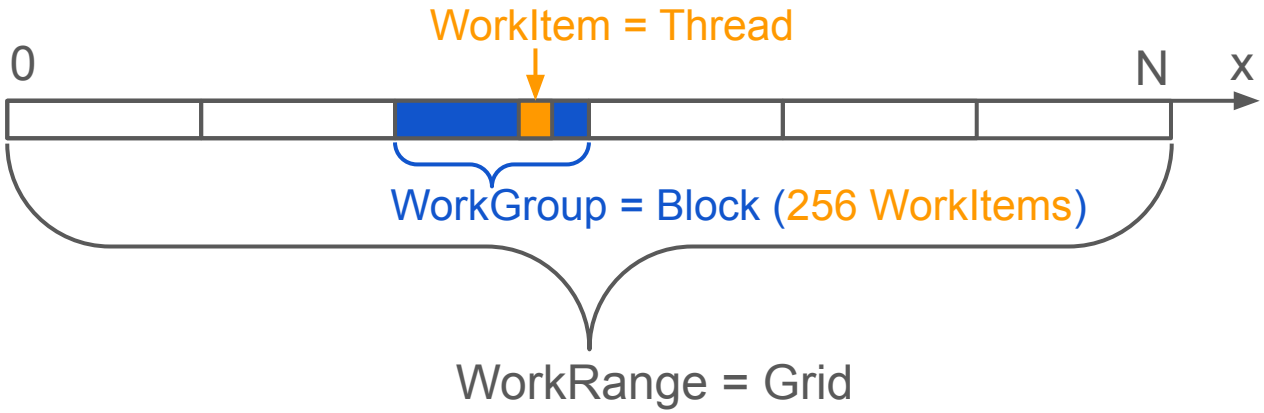


Но ведь в одном warp 32 потока!
(у **AMD**: 64 потока в wavefront)
Как так?

SM - Streaming Multiprocessor

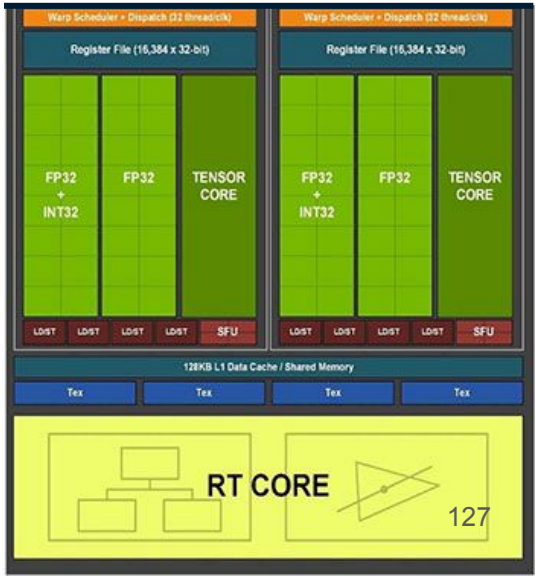


Модель вычислений массового параллелизма

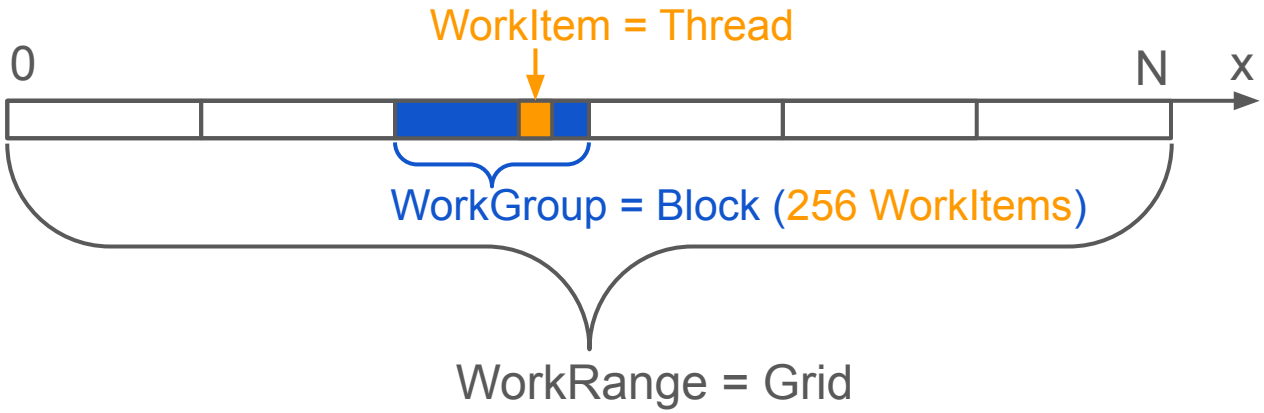


WorkGroup = Block (256 WorkItems) = 8 x Warps (32 threads)

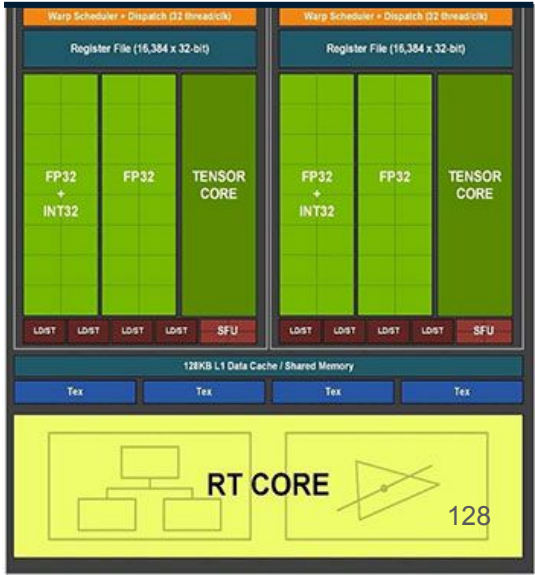
SM - Streaming Multiprocessor



Модель вычислений массового параллелизма



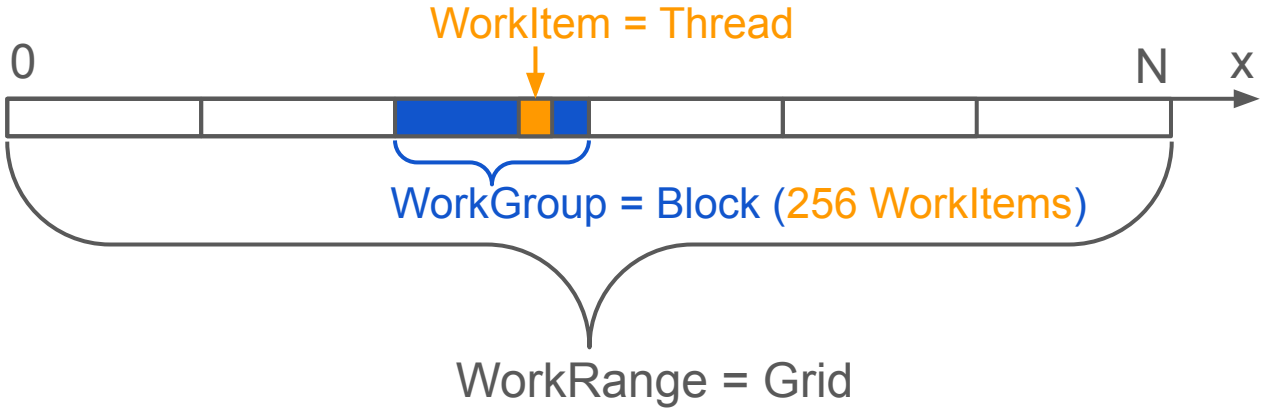
SM - Streaming Multiprocessor



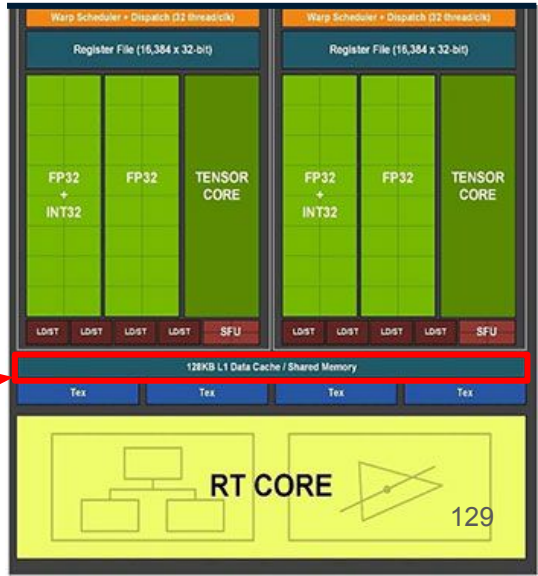
WorkGroup = Block (256 WorkItems) = 8 x Warps (32 threads)

Зачем нам контроль над **WorkGroups**?
Зачем на уровне API знать про **warps**?

Модель вычислений массового параллелизма



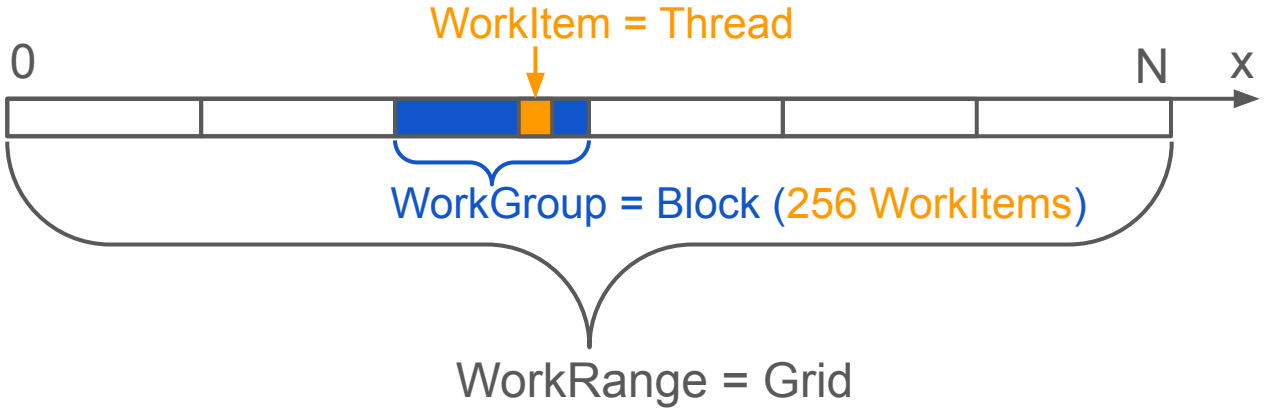
SM - Streaming Multiprocessor



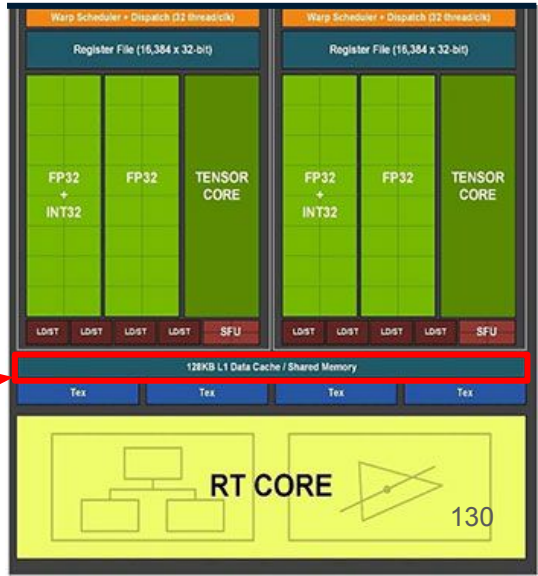
WorkGroup = Block (256 WorkItems) = 8 x Warps (32 threads)

- WorkItems коммуницируют через VRAM
- WorkItems в рамках одной WorkGroup общаются эффективнее - через **Shared/Local Memory (L1)**

Модель вычислений массового параллелизма



SM - Streaming Multiprocessor

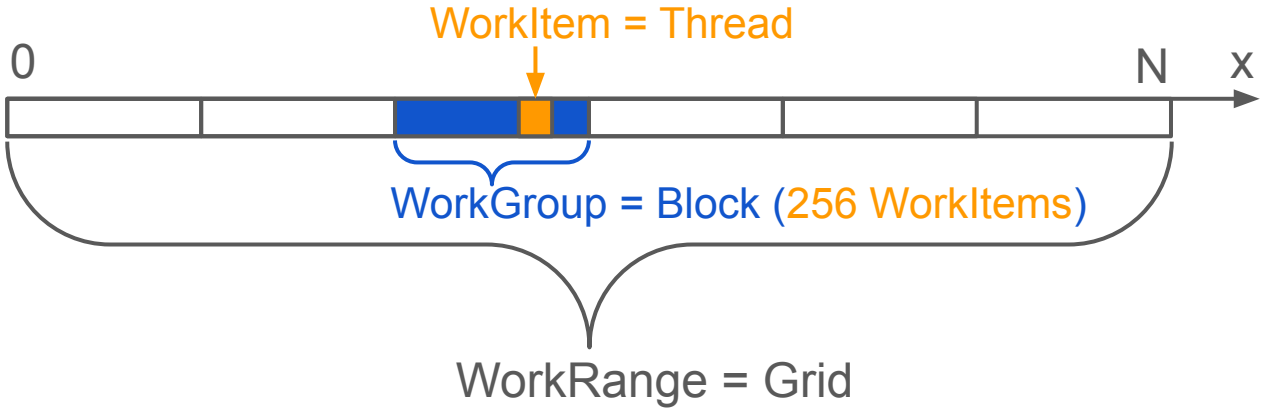


WorkGroup = Block (256 WorkItems) = 8 x Warps (32 threads)

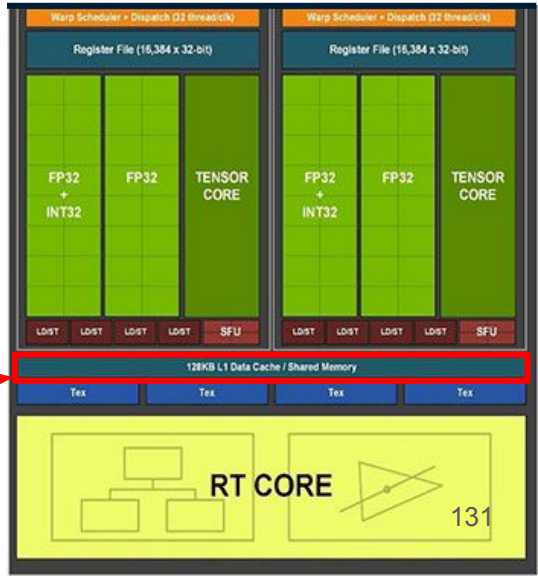
- WorkItems коммуницируют через VRAM
- WorkItems в рамках одной WorkGroup общаются эффективнее - через **Shared/Local Memory (L1)**

Могут ли **warp**-ы одной **WorkGroup** исполняться на разных SM?

Модель вычислений массового параллелизма



SM - Streaming Multiprocessor

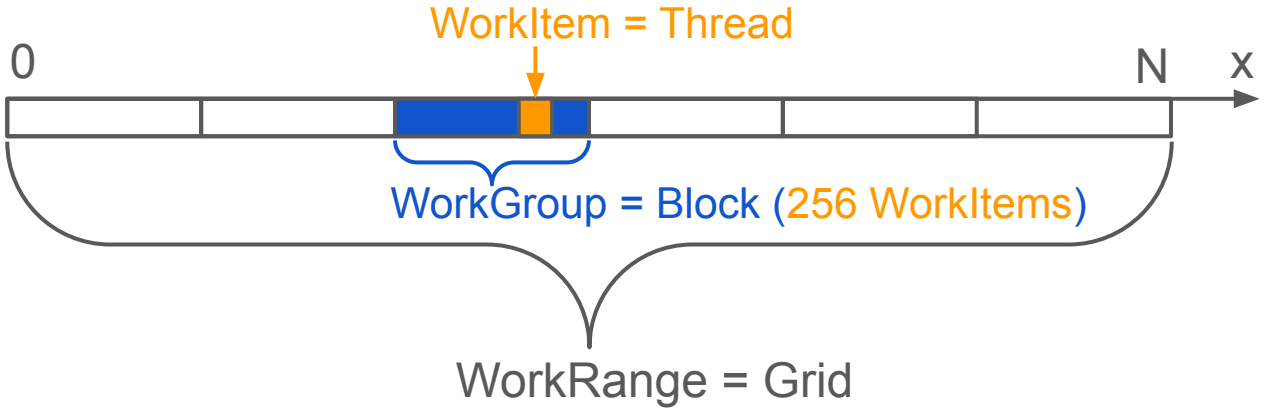


WorkGroup = Block (256 WorkItems) = 8 x Warps (32 threads)

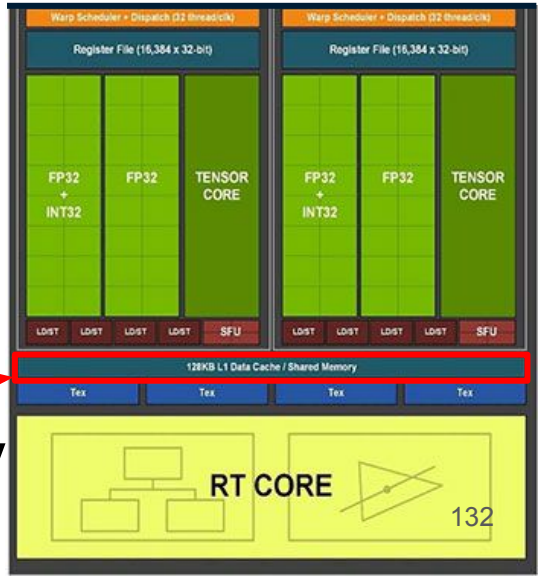
- WorkItems коммуницируют через VRAM
- WorkItems в рамках одной WorkGroup общаются эффективнее - через **Shared/Local Memory (L1)**

А если один такой поток записал в **Local Memory** число - увидит ли другой поток этой **WorkGroup**?

Модель вычислений массового параллелизма



SM - Streaming Multiprocessor

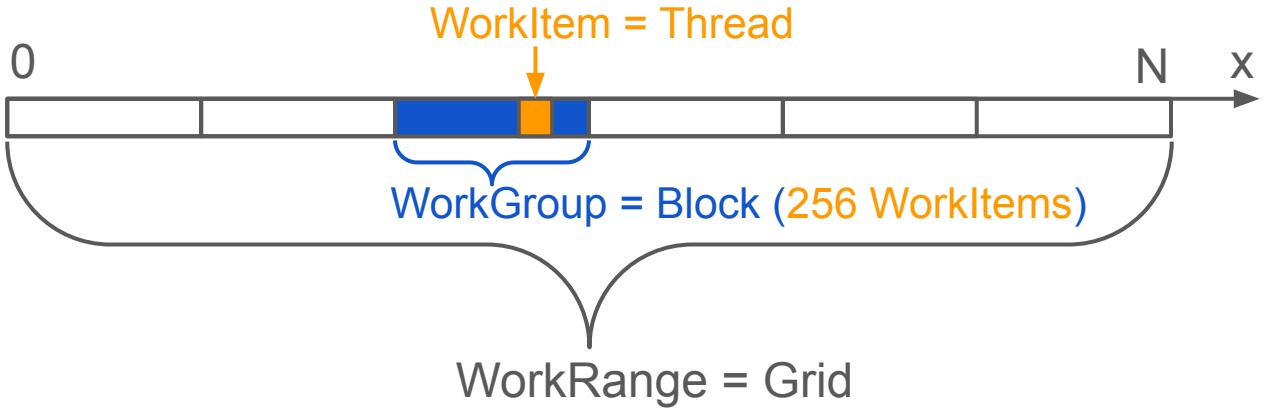


WorkGroup = Block (256 WorkItems) = 8 x Warps (32 threads)

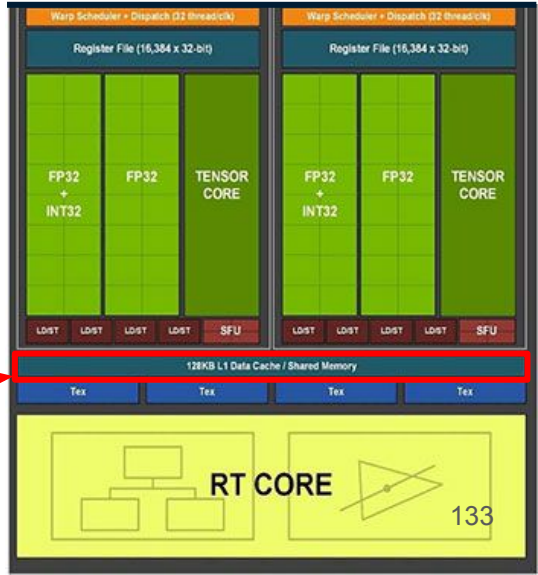
- WorkItems коммуницируют через VRAM
- WorkItems в рамках одной WorkGroup общаются эффективнее - через **Shared/Local Memory (L1)**

А если один такой поток записал в **Local Memory** число - увидит ли другой поток этой **WorkGroup**? **memory barrier!**

Модель вычислений массового параллелизма



SM - Streaming Multiprocessor

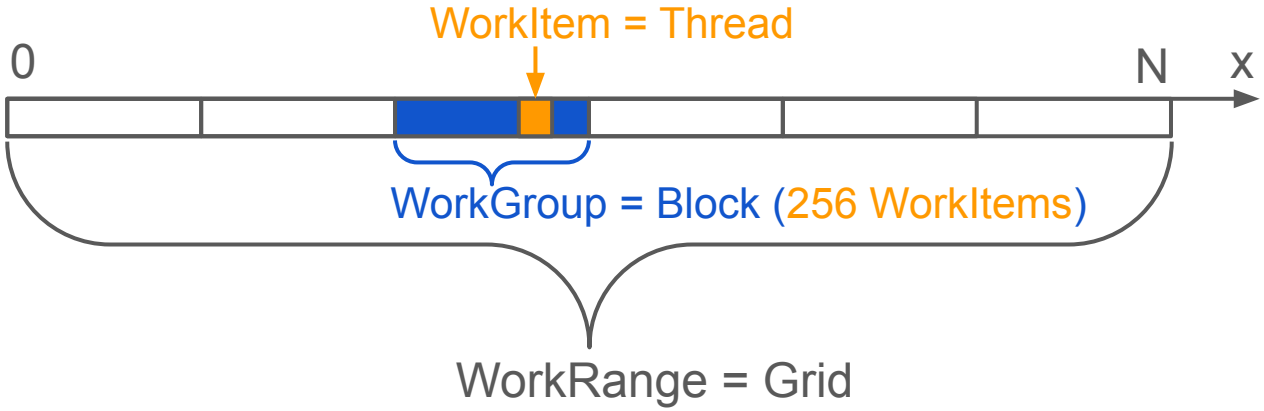


WorkGroup = Block (256 WorkItems) = 8 x Warps (32 threads)

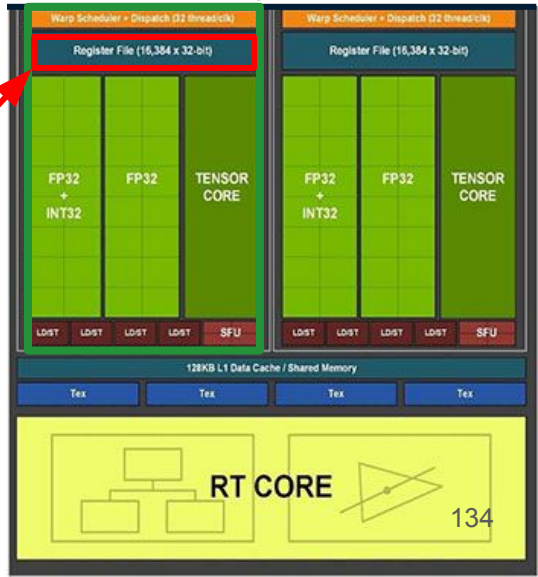
- WorkItems коммуницируют через VRAM
- WorkItems в рамках одной WorkGroup общаются эффективнее - через **Shared/Local Memory (L1)**

Могут ли потоки одного warp-а общаться еще эффективнее?

Модель вычислений массового параллелизма



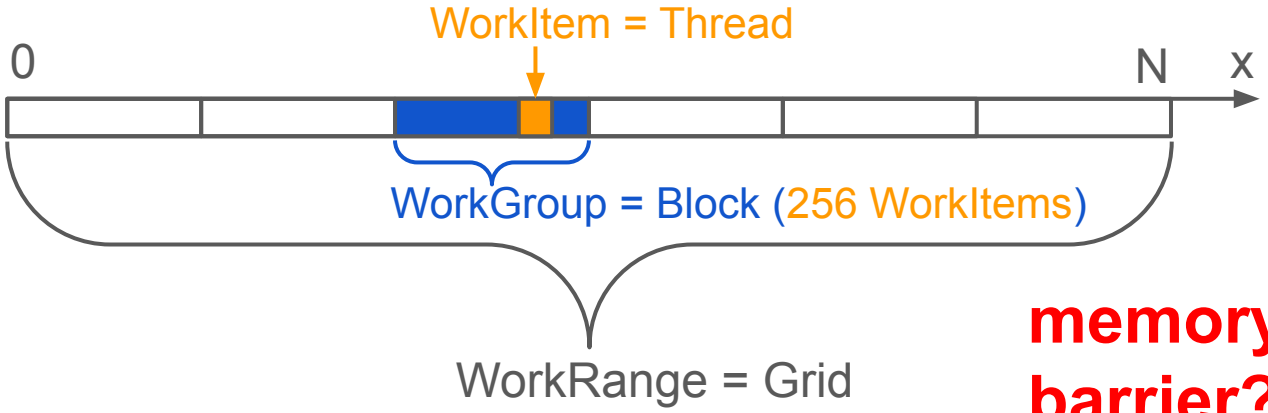
SM - Streaming Multiprocessor



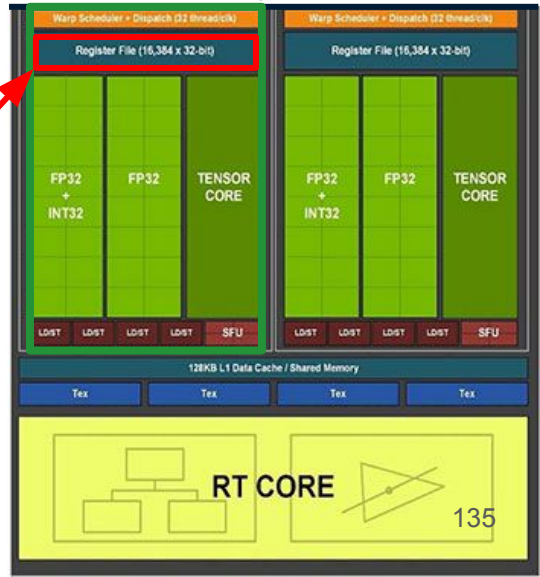
WorkGroup = Block (256 WorkItems) = 8 x Warps (32 threads)

- WorkItems коммуницируют через VRAM
- WorkItems в рамках одной WorkGroup общаются эффективнее - через **Shared/Local Memory (L1)**
- WorkItems в рамках одного warp-а могут подглядывать друг другу в **регистры (*shuffle instr.*)**

Модель вычислений массового параллелизма



SM - Streaming Multiprocessor



memory barrier?

WorkGroup = Block (256 WorkItems) = 8 x Warps (32 threads)

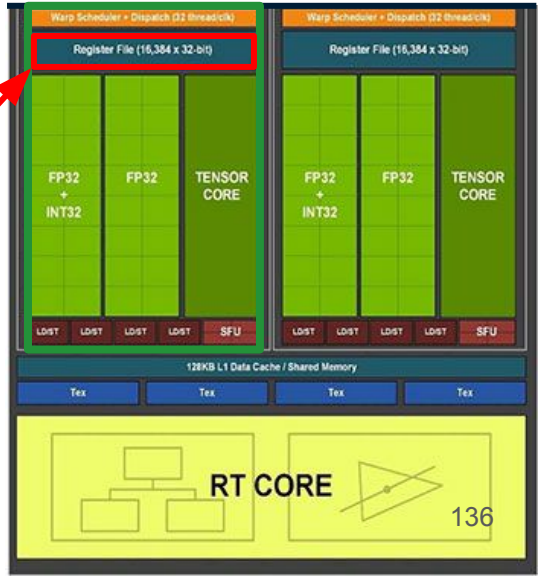
- WorkItems коммуницируют через VRAM
- WorkItems в рамках одной WorkGroup общаются эффективнее - через **Shared/Local Memory (L1)**
- WorkItems в рамках одного warp-а могут подглядывать друг другу в **регистры (*shuffle instr.*)**

GLSL (Graphics Library Shading Language)

dFdx(...), **dFdy**(...) — return the partial derivative of an argument with respect to x or y

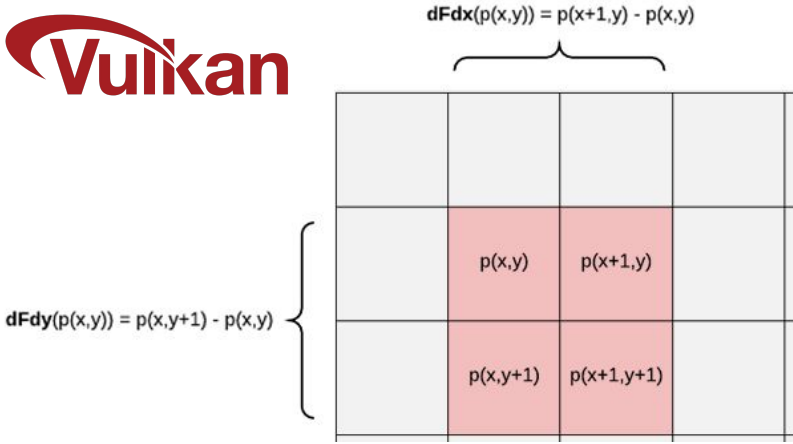
- WorkItems коммуницируют через VRAM
- WorkItems в рамках одной WorkGroup общаются эффективнее - через **Shared/Local Memory (L1)**
- WorkItems в рамках одного warp-а могут подглядывать друг другу в **регистры** (***shuffle instr.***)

SM - Streaming Multiprocessor

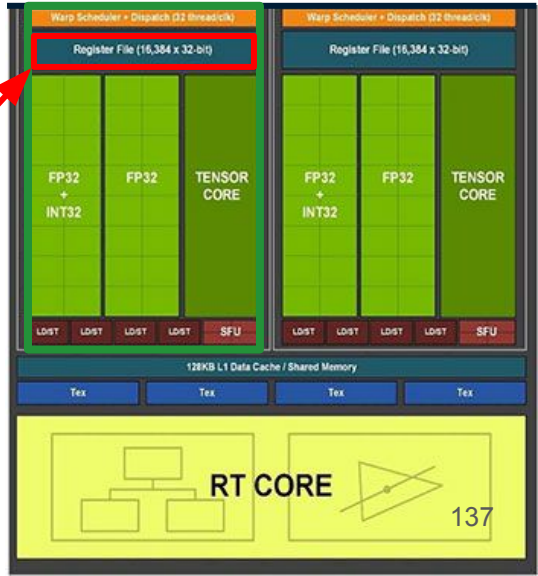


GLSL (Graphics Library Shading Language)

dFdx(...), **dFdy**(...) — return the partial derivative of an argument with respect to x or y



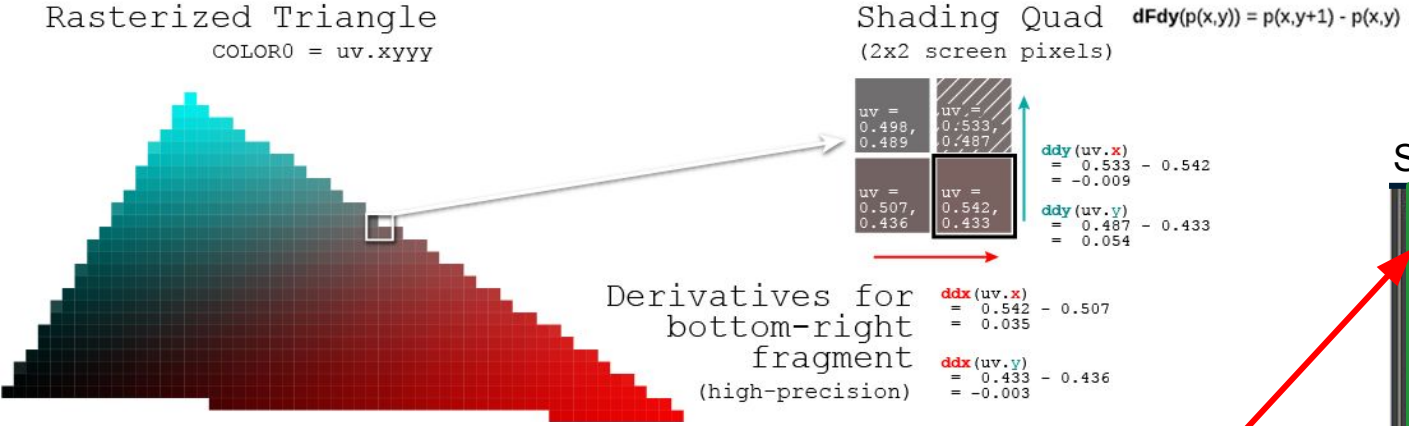
SM - Streaming Multiprocessor

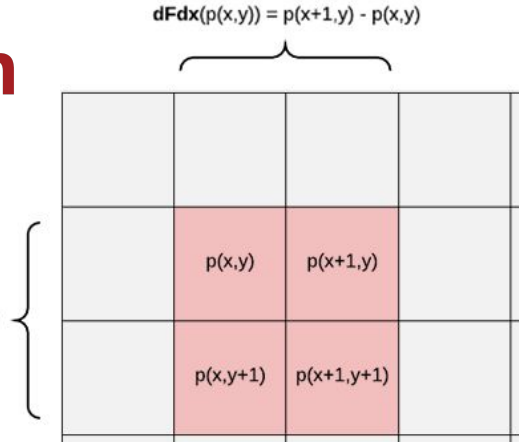


- WorkItems коммуницируют через VRAM
- WorkItems в рамках одной WorkGroup общаются эффективнее - через **Shared/Local Memory (L1)**
- WorkItems в рамках одного warp-а могут подглядывать друг другу в **регистры** (***shuffle instr.***)

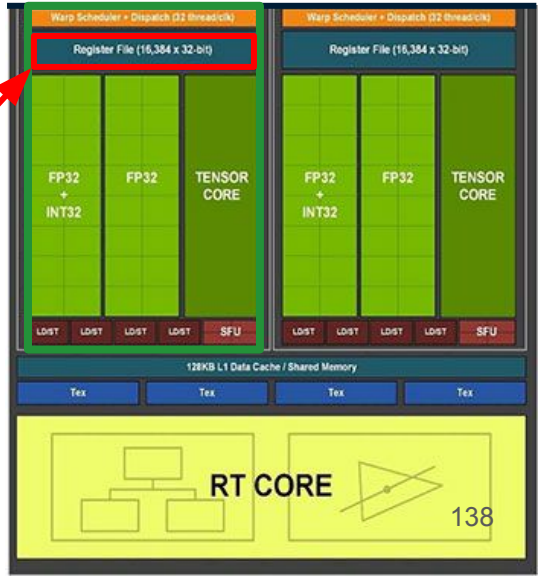
GLSL (Graphics Library Shading Language)

dFdx(...), **dFdy**(...) — return the partial derivative of an argument with respect to x or y



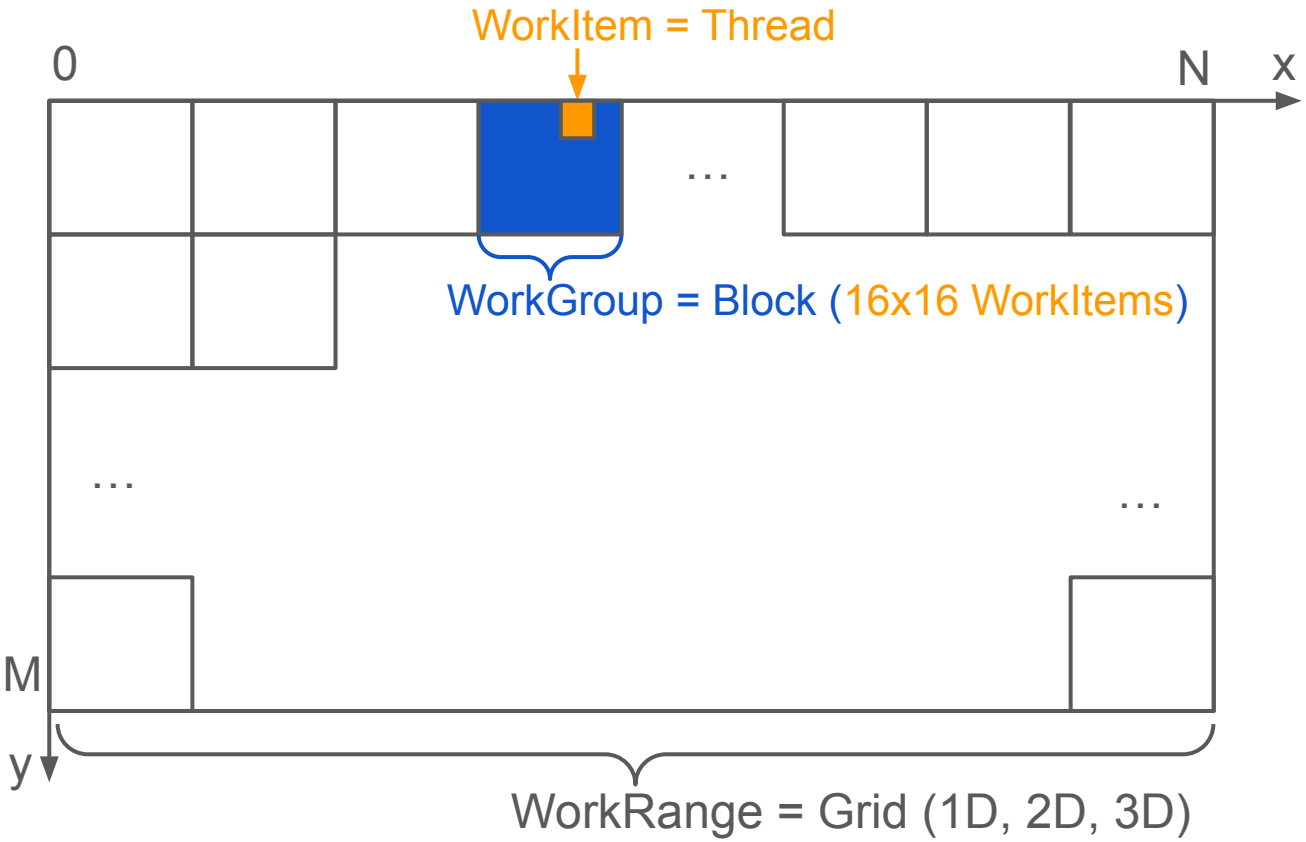


SM - Streaming Multiprocessor



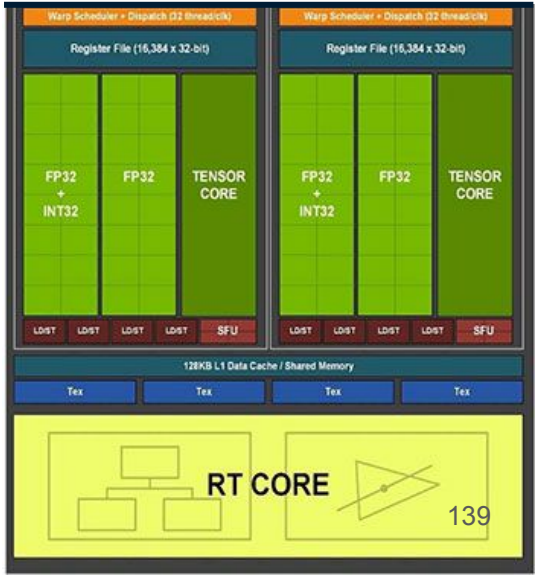
- WorkItems коммуницируют через VRAM
- WorkItems в рамках одной WorkGroup общаются эффективнее - через **Shared/Local Memory (L1)**
- WorkItems в рамках одного warp-а могут подглядывать друг другу в **регистры** (***shuffle instr.***)

Модель вычислений массового параллелизма

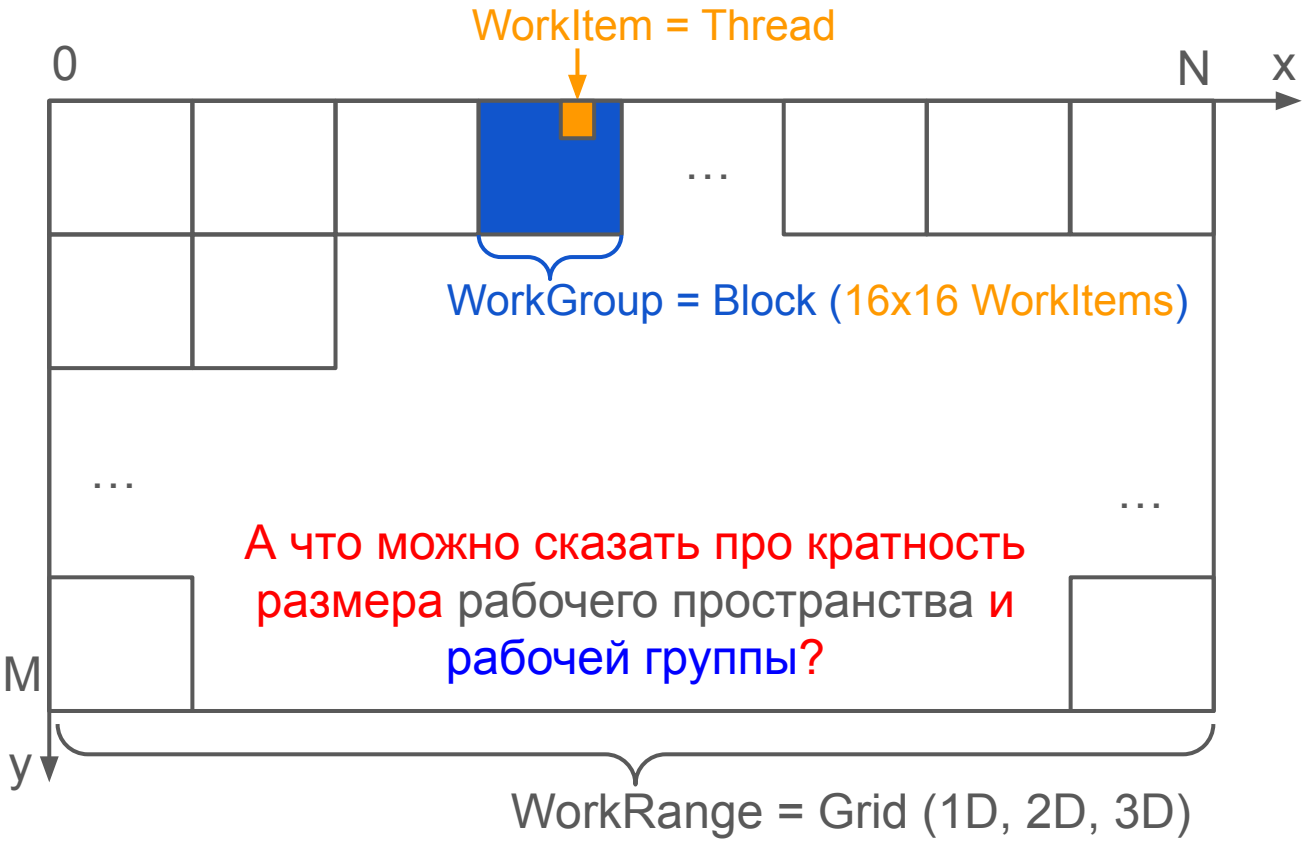


Много SM

SM - Streaming Multiprocessor

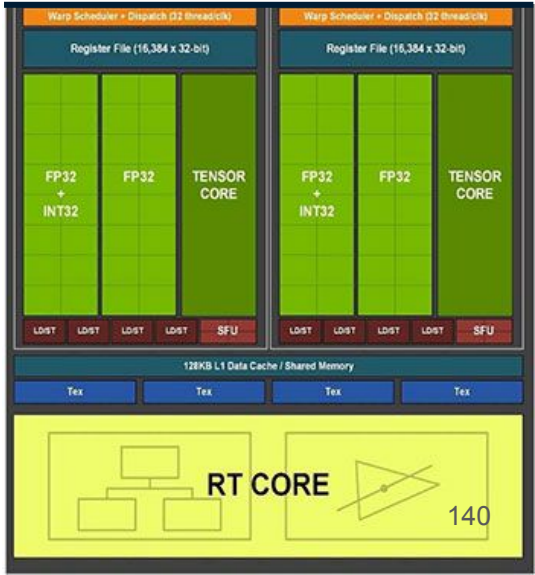


Модель вычислений массового параллелизма



А что можно сказать про кратность размера рабочего пространства и рабочей группы?

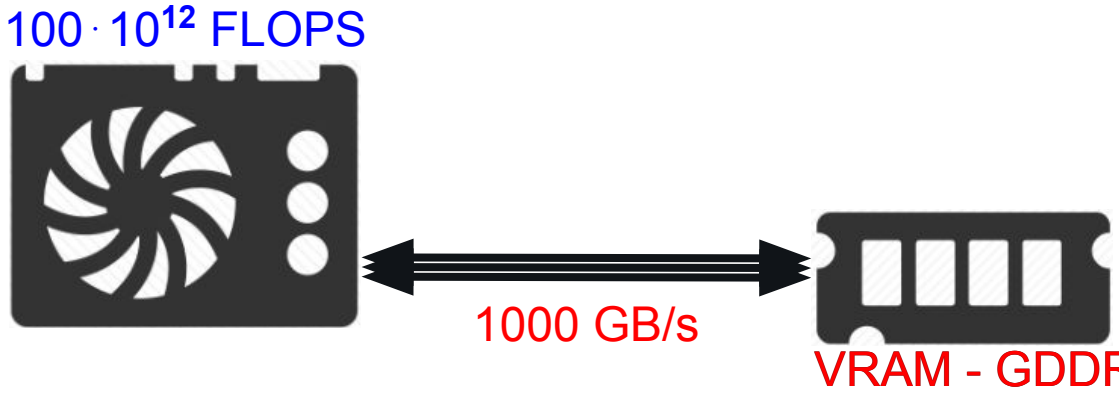
SM - Streaming Multiprocessor



Глава 5: Профилирование и оптимизация

МНОГО вычислений? (**ALU**)

МНОГО читать+писать? (**VRAM**)

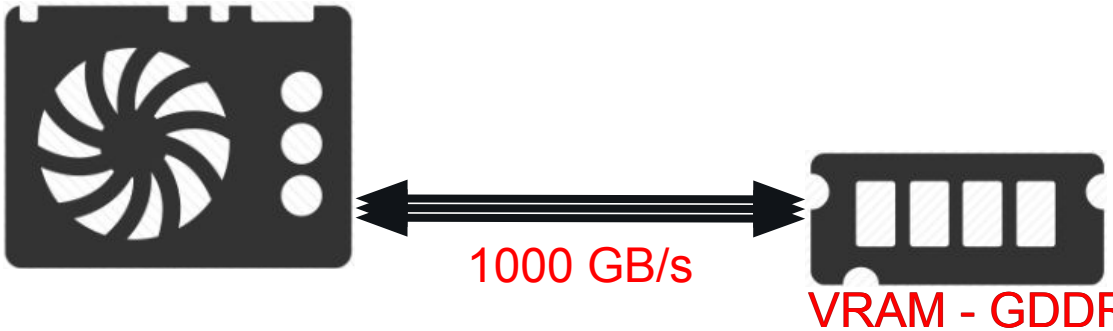


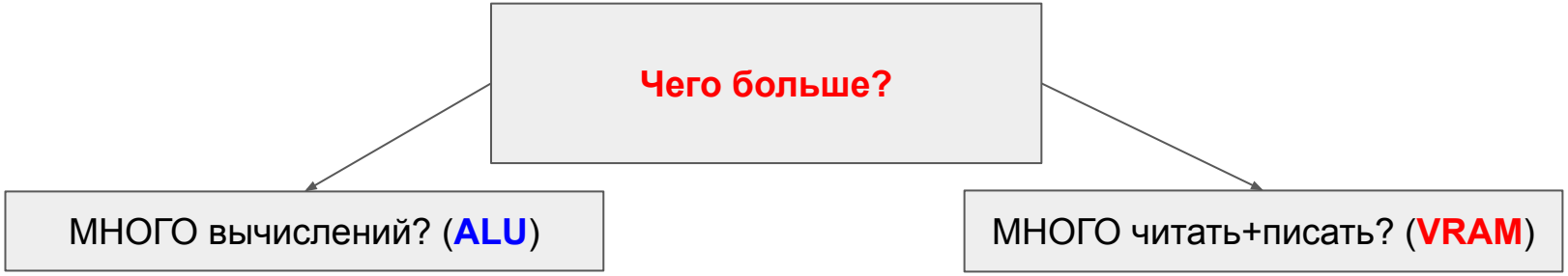
Чего больше?
Как сравнить яблоки и апельсины?

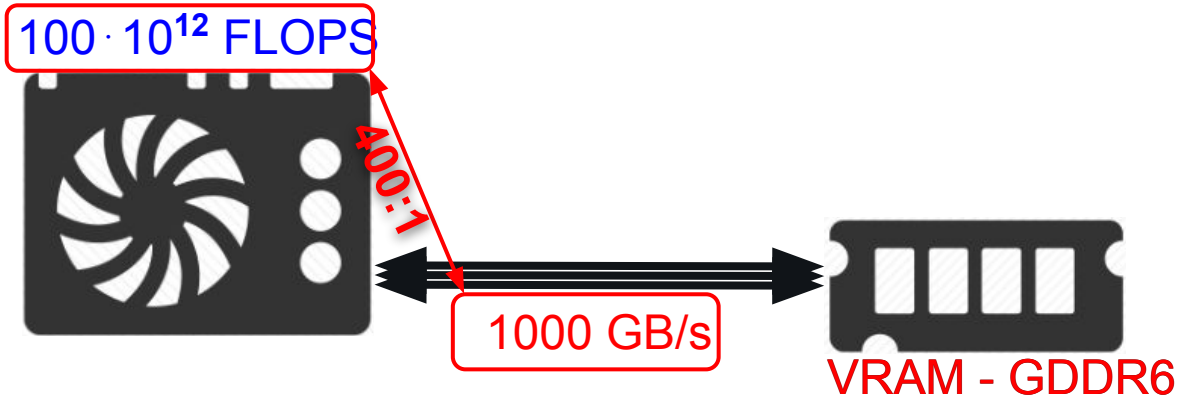
МНОГО вычислений? (**ALU**)

МНОГО читать+писать? (**VRAM**)

$100 \cdot 10^{12}$ FLOPS




Чего больше?
МНОГО вычислений? (ALU)
МНОГО читать+писать? (VRAM)


100 · 10^12 FLOPS
400:1
1000 GB/s
VRAM - GDDR6

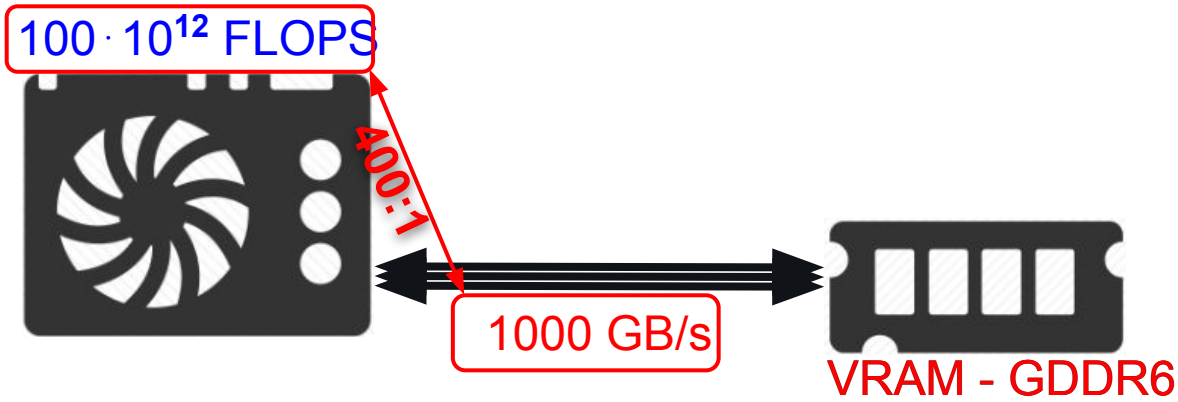
Чего больше?
Какая пропорция
ALU:VRAM?

Если **ALU:VRAM** > 400:1

МНОГО вычислений? (**ALU**)

Если **ALU:VRAM** < 400:1

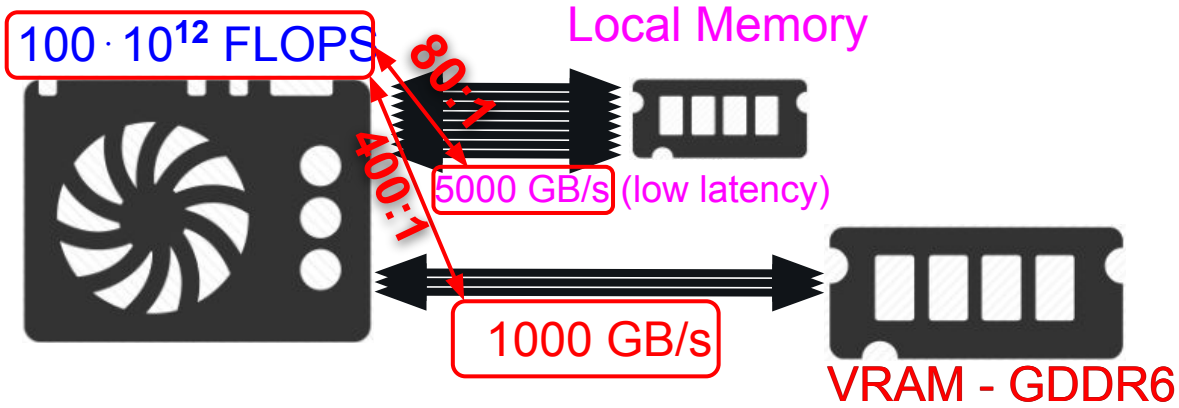
МНОГО читать+писать? (**VRAM**)



Чего больше?
Какая пропорция
ALU:VRAM?

Если **ALU:VRAM** > 400:1 → МНОГО вычислений? (**ALU**)

Если **ALU:VRAM** < 400:1 → МНОГО читать+писать? (**VRAM**)



Чего больше?
Какая пропорция
ALU:VRAM?

Если **ALU:VRAM** > 400:1

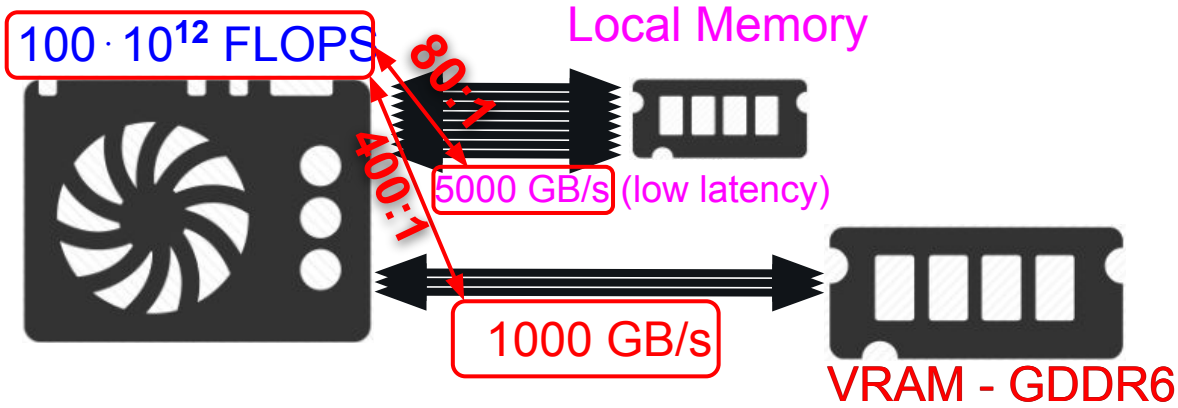
Если **ALU:VRAM** < 400:1

МНОГО вычислений? (**ALU**)

МНОГО читать+писать? (**VRAM**)

Если утилизируется
>70% ALU FLOPs

compute bound



Чего больше?
Какая пропорция
ALU:VRAM?

Если **ALU:VRAM** > 400:1

Если **ALU:VRAM** < 400:1

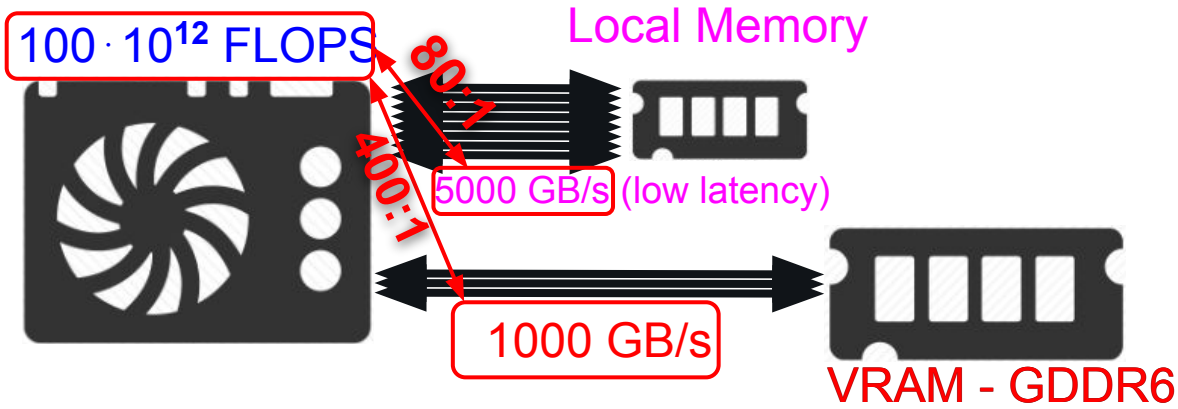
МНОГО вычислений? (**ALU**)

МНОГО читать+писать? (**VRAM**)

Если утилизируется
>70% ALU FLOPs

compute bound

Устранить
code
divergence



Чего больше?
Какая пропорция
ALU:VRAM?

Если **ALU:VRAM** > 400:1

Если **ALU:VRAM** < 400:1

МНОГО вычислений? (**ALU**)

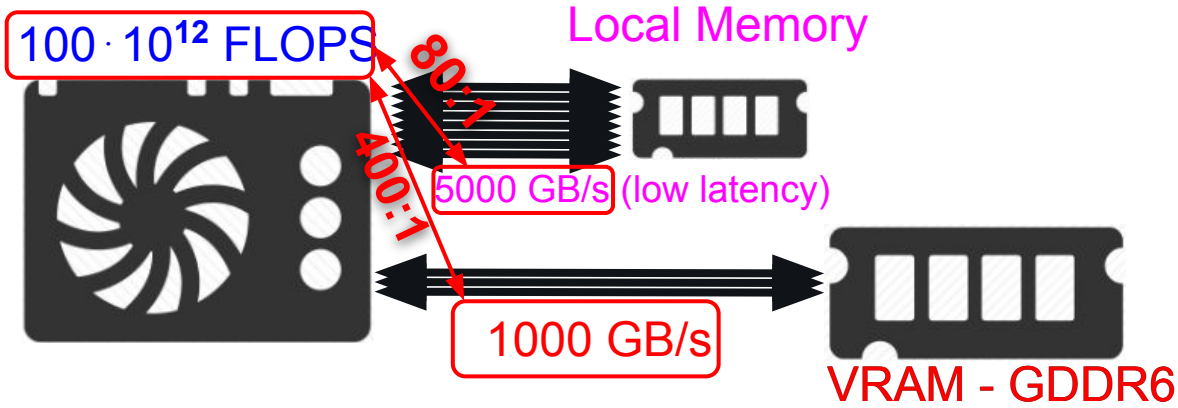
Низкая утилизация ALU

МНОГО читать+писать? (**VRAM**)

Если утилизируется
>70% ALU FLOPs

compute bound

Устранить
code
divergence



Чего больше?
Какая пропорция
ALU:VRAM?

Если **ALU:VRAM** > 400:1

Если **ALU:VRAM** < 400:1

МНОГО вычислений? (**ALU**)

Низкая утилизация ALU

МНОГО читать+писать? (**VRAM**)

Если утилизируется
>70% ALU FLOPs

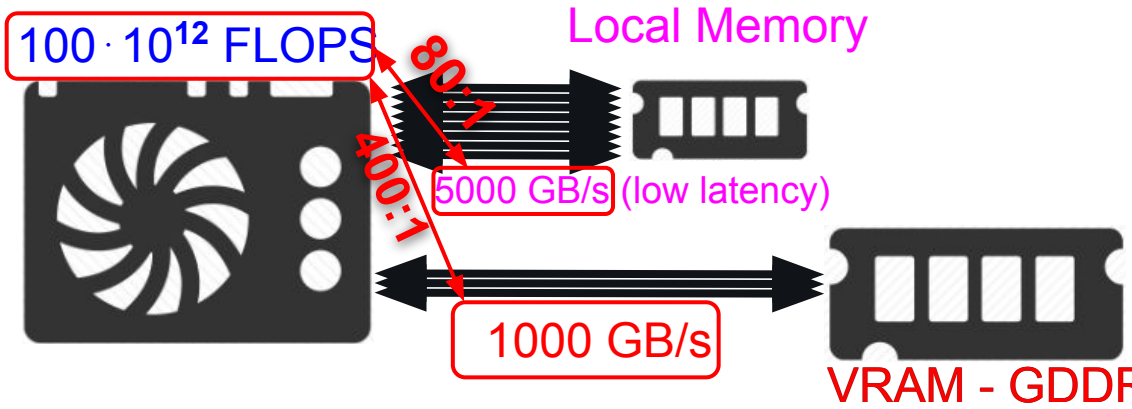
compute bound

Устранить
code
divergence

Если утилизируется
>70% пропускной
способности VRAM

memory bound

Можно ли что-то сделать?



Чего больше?
Какая пропорция
ALU:VRAM?

Если **ALU:VRAM** > 400:1

Если **ALU:VRAM** < 400:1

МНОГО вычислений? (**ALU**)

Низкая утилизация ALU

МНОГО читать+писать? (**VRAM**)

Если утилизируется
>70% ALU FLOPs

compute bound

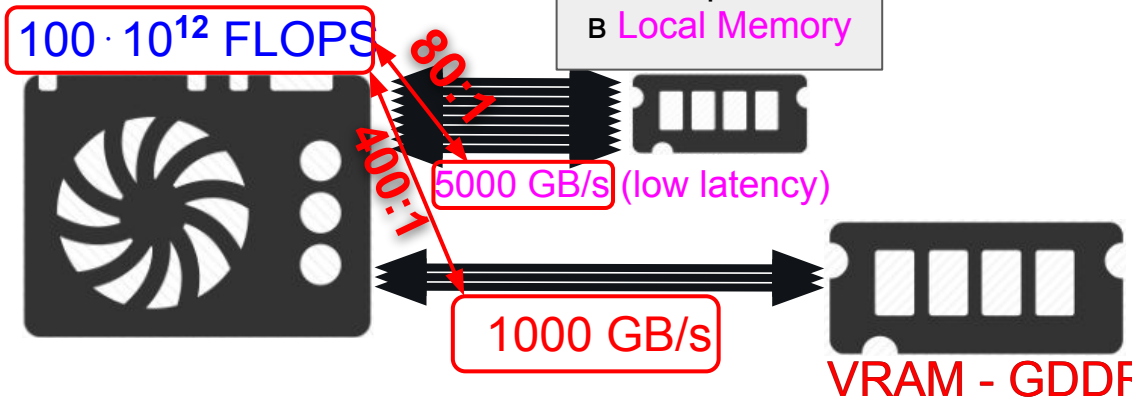
Устранить
code
divergence

Если утилизируется
>70% пропускной
способности VRAM

memory bound

Можно переиспользовать данные?

Кэшировать
в Local Memory



Чего больше?
Какая пропорция
ALU:VRAM?

Если **ALU:VRAM** > 400:1

Если **ALU:VRAM** < 400:1

МНОГО вычислений? (**ALU**)

Низкая утилизация ALU

МНОГО читать+писать? (**VRAM**)

Если утилизируется
>70% ALU FLOPs

compute bound

Устранить
code divergence

Если утилизируется
>70% пропускной
способности VRAM

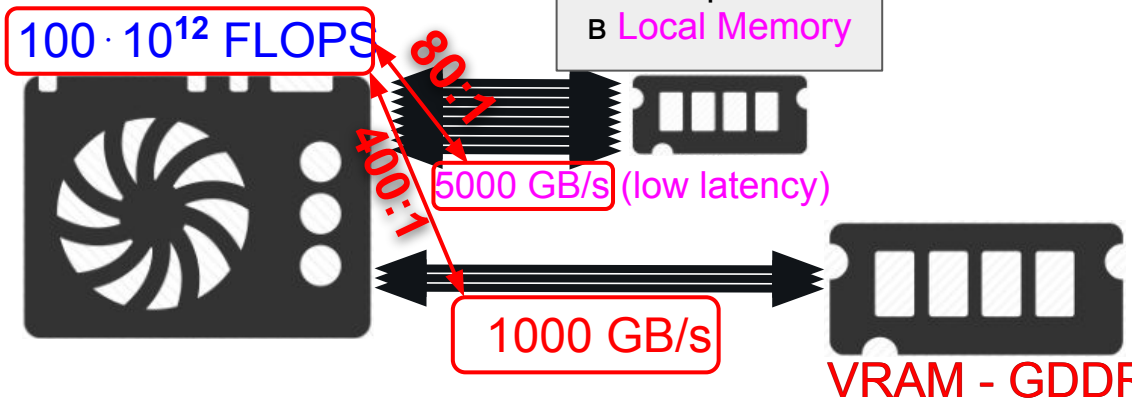
memory bound

Низкая утилизация
проп. способоности VRAM

Почему?

Можно переиспользовать данные?

Кэшировать
в Local Memory



Чего больше?
Какая пропорция
ALU:VRAM?

Если **ALU:VRAM** > 400:1

Если **ALU:VRAM** < 400:1

МНОГО вычислений? (**ALU**)

Низкая утилизация ALU

МНОГО читать+писать? (**VRAM**)

Если утилизируется
>70% ALU FLOPs

compute bound

Устранить
code divergence

Если утилизируется
>70% пропускной
способности VRAM

memory bound

Низкая утилизация
проп. способоности VRAM

Почему?

Можно переиспользовать данные?

Кэшировать
в Local Memory

Сделать паттерн
доступа **coalesced**

Чего больше?
Какая пропорция
ALU:VRAM?

Если **ALU:VRAM** > 400:1

Если **ALU:VRAM** < 400:1

МНОГО вычислений? (**ALU**)

Низкая утилизация ALU

МНОГО читать+писать? (**VRAM**)

Если утилизируется
>70% ALU FLOPs

compute bound

Устранить
code divergence

Если утилизируется
>70% пропускной
способности VRAM

memory bound

Низкая утилизация
проп. способоности VRAM

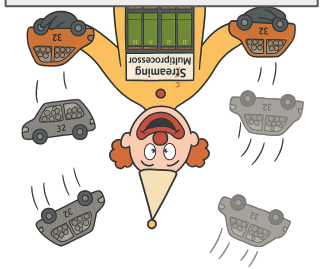
Почему?

Можно переиспользовать данные?

Кэшировать
в Local Memory

Сделать паттерн
доступа **coalesced**

Увеличить
occupancy



Чего больше?
Какая пропорция
ALU:VRAM?

Если **ALU:VRAM** > 400:1

Если **ALU:VRAM** < 400:1

МНОГО вычислений? (**ALU**)

Низкая утилизация ALU

МНОГО читать+писать? (**VRAM**)

Если утилизируется
>70% ALU FLOPs

compute bound

Устранить
code divergence

Если утилизируется
>70% пропускной
способности VRAM

memory bound

Низкая утилизация
проп. способоности VRAM

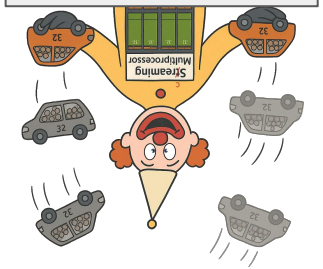
Почему?

Можно переиспользовать данные?

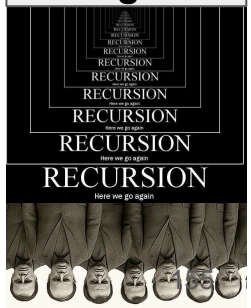
Кэшировать
в Local Memory

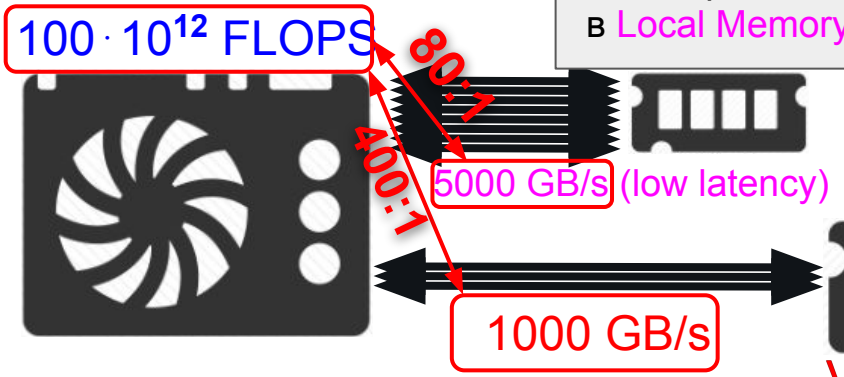
Сделать паттерн
доступа **coalesced**

Увеличить
occupancy



Устранить
code divergence





Чего больше?
Какая пропорция
ALU:VRAM?

Если **ALU:VRAM** > 400:1

Если **ALU:VRAM** < 400:1

МНОГО вычисл[illegible]й? (**ALU**)

Низкая утилизация ALU

МНОГО читать+писа[illegible] (**VRAM**)

Если утилизируется
>70% ALU FLOPs

Устра[illegible]ть
code
[illegible]ergence

Если утилизируется
>70% пропускной
способности VRAM

Низкая утилизация
проп. способоности VRAM

compute

[illegible]чему?

NVIDIA Nsight

$100 \cdot 10^{12}$ FLOPS

80:1

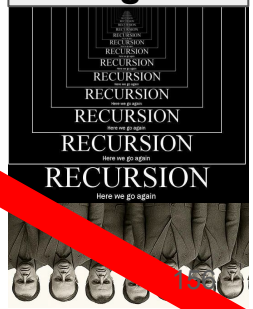
40:1

в L[illegible]al Memory

доступа **coalesc[illegible]**

[illegible]ить
occupancy

Устранить
code
divergence

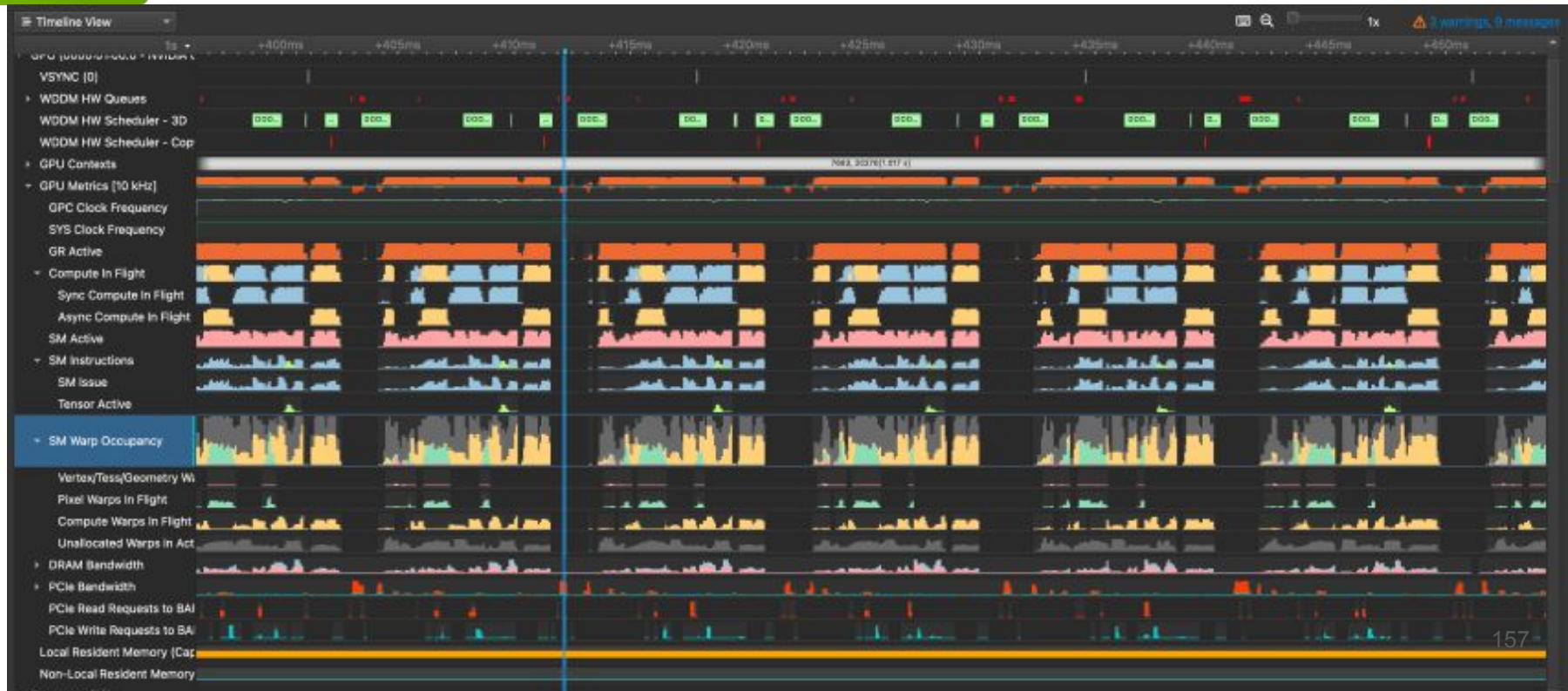


5000 GB/s (low latency)

1000 GB/s

VRAM - GDDR6

NVIDIA Nsight



NVIDIA Nsight Compute

Debug Profile Tools Window Help

Disconnect Terminate Profile Kernel Baselines Metric Details

addConstDouble3.ncu-rep addConstDouble.ncu-rep Untitled 1

	Result	Time	GPU	SM Frequency	CC
ouble3	539 - addConstDouble3 (4096, 1, 1)x(256, 1, 1)	101.73 usecond	0 - NVIDIA TITAN V	1.17 cycle/nsecond	7.0

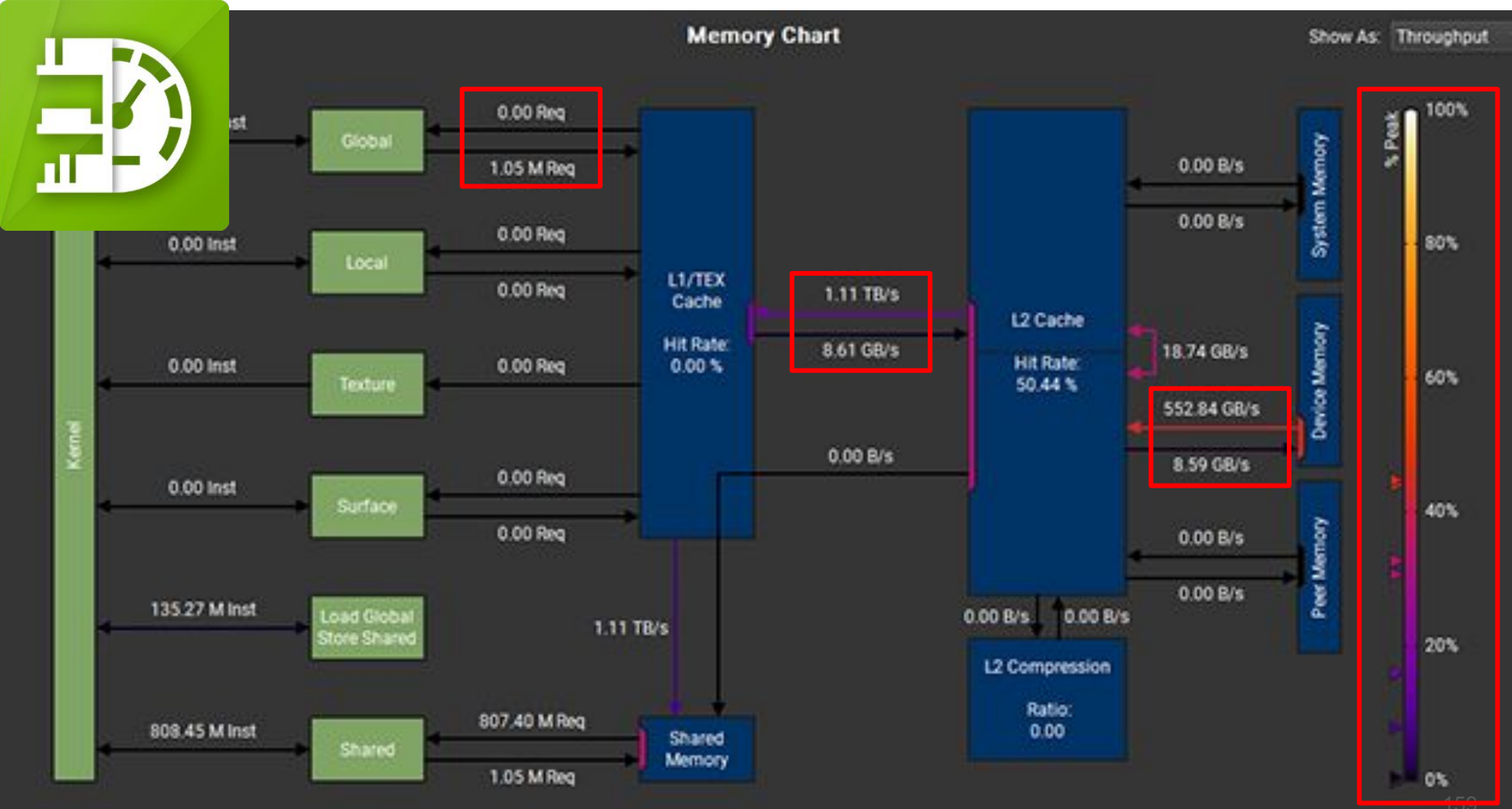
Report	Result	Time	GPU	SM Frequency	CC
addConstDouble	539 - addConstDouble (12288, 1, 1)x(256, 1, 1)	84.35 usecond	0 - NVIDIA TITAN V	1.16 cycle/nsecond	7.0

coalescedGlobalAccesses.cu Find... Instructions Executed

Source: uncoalescedGlobalAccesses.cu Find... Navigation: Instructions Executed

#	Source	Live arp Stall Sampling Registers (All Samples)		Instructions Executed
32	* global memory and generates an output array of double3 in global			
33	*/			
34				
35	#include <stdio.h>			
36	#include <cuda_runtime_api.h>			
37				
38	#define BLOCK_SIZE 256			
39				
40	#define RUNTIME_API_CALL(apiFuncCall)			
41	do {			
42	cudaError_t _status = apiFuncCall;			
43	if (_status != cudaSuccess) {			
44	fprintf(stderr, "%s:%d: error: function %s failed with err			
45	__FILE__, __LINE__, #apiFuncCall, cudaGetErrorStrir			
46	exit(EXIT_FAILURE);			
47	}			
48	} while (0)			
49				
50	__global__ void addConstDouble3(int numElements, double3 *d_in, d	1	0.64%	10.00%
51	{			
52	int index = blockIdx.x * blockDim.x + threadIdx.x;	3	1.58%	15.00%
53	if (index < numElements)	2	0.07%	10.00%
54	{			
55	double3 a = d_in[index];	11	31.08%	25.00%
56	a.x += k;	9	14.92%	5.00%
57	a.y += k;	9	1.78%	5.00%
58	a.z += k;	9	2.15%	5.00%
59	d_out[index] = a;	9	32.90%	20.00%
60	}			
61	}	1	14.88%	5.00%
62				
63	int main (int argc, char *argv[])			
64	{			
65	// Error code to check return values for CUDA calls			
66	cudaError_t err = cudaSuccess;			
67	double constK = 10.0;			
68				
69	int kernelOption = 0;			
70	if (argc > 1)			

#	Source	Live arp Stall Sampling Registers (All Samples)		Instructions Executed
32	* global memory and generates an output array of double3 in global			
33	*/			
34				
35	#include <stdio.h>			
36	#include <cuda_runtime_api.h>			
37				
38	#define BLOCK_SIZE 256			
39				
40	#define RUNTIME_API_CALL(apiFuncCall)			
41	do {			
42	cudaError_t _status = apiFuncCall;			
43	if (_status != cudaSuccess) {			
44	fprintf(stderr, "%s:%d: error: function %s failed with err			
45	__FILE__, __LINE__, #apiFuncCall, cudaGetErrorStrir			
46	exit(EXIT_FAILURE);			
47	}			
48	} while (0)			
49				
50	__global__ void addConstDouble(int numElements, double *d_in, dou	1	0.42%	14.29%
51	{			
52	int index = blockIdx.x * blockDim.x + threadIdx.x;	3	1.31%	21.43%
53	if (index < numElements)	2	0.37%	14.29%
54	{			
55	d_out[index] = d_in[index] + k;	7	93.47%	42.86%
56	}			
57	}	1	4.44%	7.14%
58				
59	int main (int argc, char *argv[])			
60	{			
61	// Error code to check return values for CUDA calls			
62	cudaError_t err = cudaSuccess;			
63	double constK = 10.0;			
64				
65	int kernelOption = 0;			
66	if (argc > 1)			
67	{			
68	kernelOption = atoi(argv[1]);			
69	}			
70				


Memory Chart
Show As: Throughput
Kernel
0.00 Inst
Global
0.00 Req
1.05 M Req
0.00 Inst
Local
0.00 Req
0.00 Req
0.00 Inst
Texture
0.00 Req
0.00 Inst
Surface
0.00 Req
0.00 Req
135.27 M Inst
Load Global
Store Shared
808.45 M Inst
Shared
807.40 M Req
1.05 M Req
L1/TEX
Cache
Hit Rate:
0.00 %
1.11 TB/s
8.61 GB/s
0.00 B/s
1.11 TB/s
Shared
Memory
L2 Cache
Hit Rate:
50.44 %
0.00 B/s
0.00 B/s
18.74 GB/s
552.84 GB/s
8.59 GB/s
0.00 B/s
0.00 B/s
0.00 B/s
0.00 B/s
L2 Compression
Ratio:
0.00
System Memory
Device Memory
Peer Memory
% Peak
100%
80%
60%
40%
20%
0%

2 - 566 - mergeRanksAndIndicesKer | Clear Baselines | Apply Rules | Occupancy Calculator | Source Comparison | Copy as Image

	Time	Cycles	GPU	SM Frequency	Process	Attributes
geRanksAndIndicesKernel (64, 1, 1)x(256, 1, 1)	4.19 us	4,326	0 - NVIDIA RTX A4500	1.03 cycle/ns	[762349] CuMergeSort	
geSortSharedKernel (4096, 1, 1)x(512, 1, 1)	555.10 us	5,79,537	0 - NVIDIA RTX A4500	1.04 cycle/ns	[762349] CuMergeSort	
erateSampleRanksKernel (64, 1, 1)x(256, 1, 1)	26.72 us	27,984	0 - NVIDIA RTX A4500	1.05 cycle/ns	[762349] CuMergeSort	

GPU Speed Of Light Throughput

GPU Throughput Chart

High-level overview of the throughput for compute and memory resources of the GPU. For each unit, the throughput reports the achieved percentage of utilization with respect to the theoretical maximum. Breakdowns show the throughput for each individual sub-metric of Compute and Memory to clearly identify the highest contributor. High-level overview of the utilization for compute and memory resources of the GPU presented as a roofline chart.

Compute (SM) Throughput [%]	10.84 (-75.36%, z=-0.61)	Duration [us]	4.19 (-98.56%, z=-0.75)
Memory Throughput [%]	7.88 (-87.90%, z=-1.23)	Elapsed Cycles [cycle]	4,326 (-98.58%, z=-0.75)
L1/TEX Cache Throughput [%]	10.84 (-79.01%, z=-0.82)	SM Active Cycles [cycle]	2,192.52 (-99.26%, z=-0.74)
L2 Cache Throughput [%]	7.88 (-51.10%, z=-0.75)	SM Frequency [cycle/ns]	1.03 (-1.32%, z=-1.38)
DRAM Throughput [%]	5.53 (-83.26%, z=-1.08)	DRAM Frequency [cycle/ns]	7.55 (-0.53%, z=-1.39)

GPU Throughput

PM Sampling

Timeline view of PM metrics sampled periodically over the workload duration. Data is collected across multiple passes. Use this section to understand how workload behavior changes over its runtime.

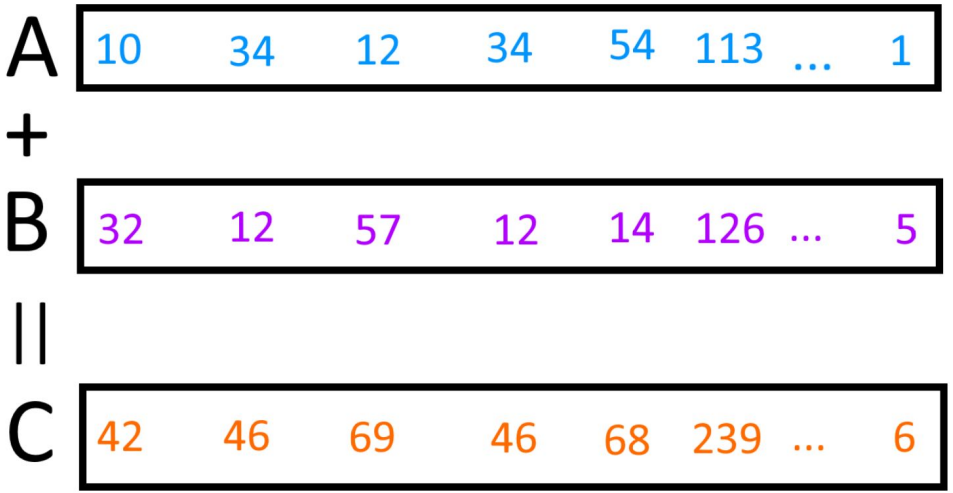
Baselines

Difference Bars: Green/Red

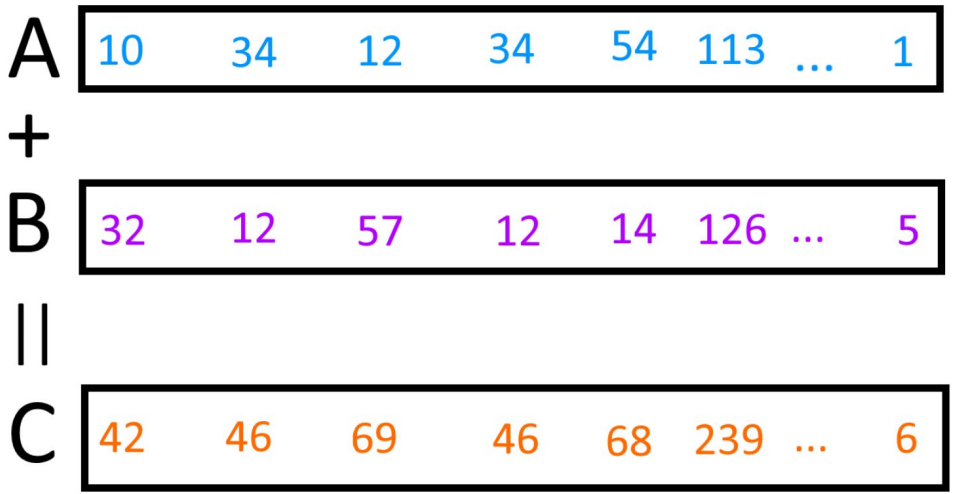
	Name	Launch	Report	Time	Cycles	Registers	GPU	SM Frequency	CC	Process
☑	mergeSort	560 - mergeSortSharedKernel (4096, ...	/home/prabhanshuc/Documents/me...	555.10 us	5,79,537	17	0 - NVIDIA RTX ...	1.04 cycle/ns	8.6	[762349] CuMergeSort
☑	generateSample	563 - generateSampleRanksKernel (6...	/home/prabhanshuc/Documents/me...	26.72 us	27,984	18	0 - NVIDIA RTX ...	1.05 cycle/ns	8.6	[762349] CuMergeSort
☑	mergeRanks	566 - mergeRanksAndIndicesKernel (...	/home/prabhanshuc/Documents/me...	4.19 us	4,326	16	0 - NVIDIA RTX ...	1.03 cycle/ns	8.6	[762349] CuMergeSort


Глава 6: Как выглядит код
OpenMP, OpenCL, CUDA, Vulkan (GLSL)

Пример: A + B

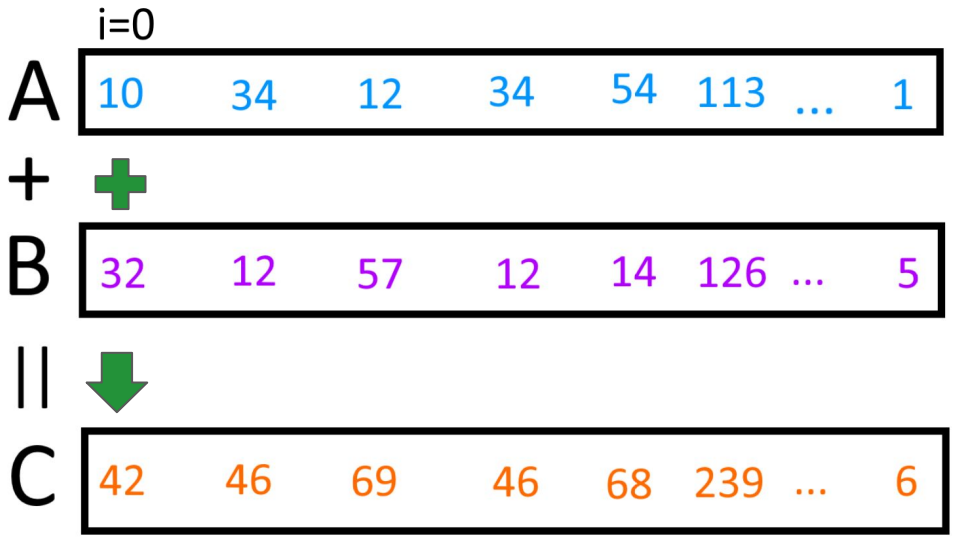


Пример: A + B



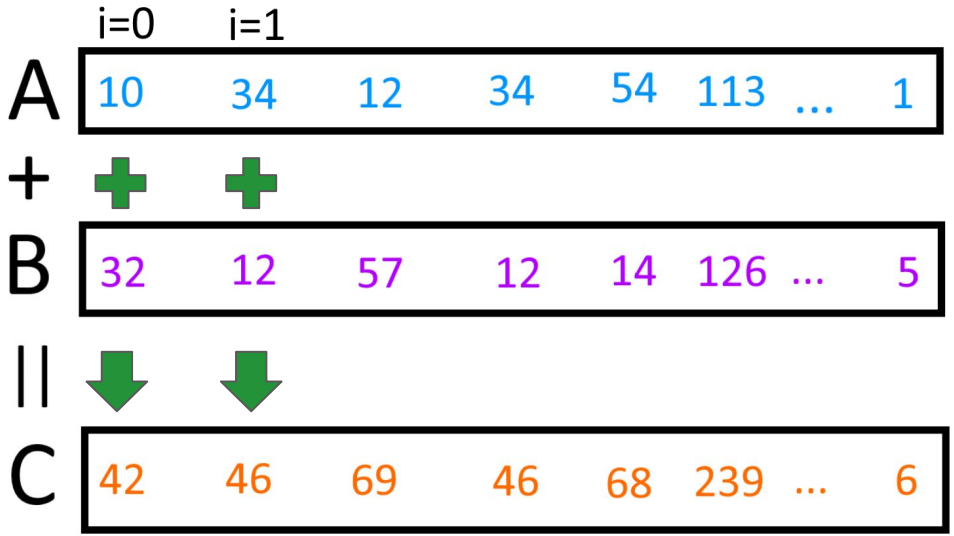

```
void solveCPU(int[] a, int[] b, int c[], int n) {
    for (int i = 0; i < n; ++i) {
        int sum = a[i] + b[i];
        c[i] = sum;
    }
}
```

Пример: A + B



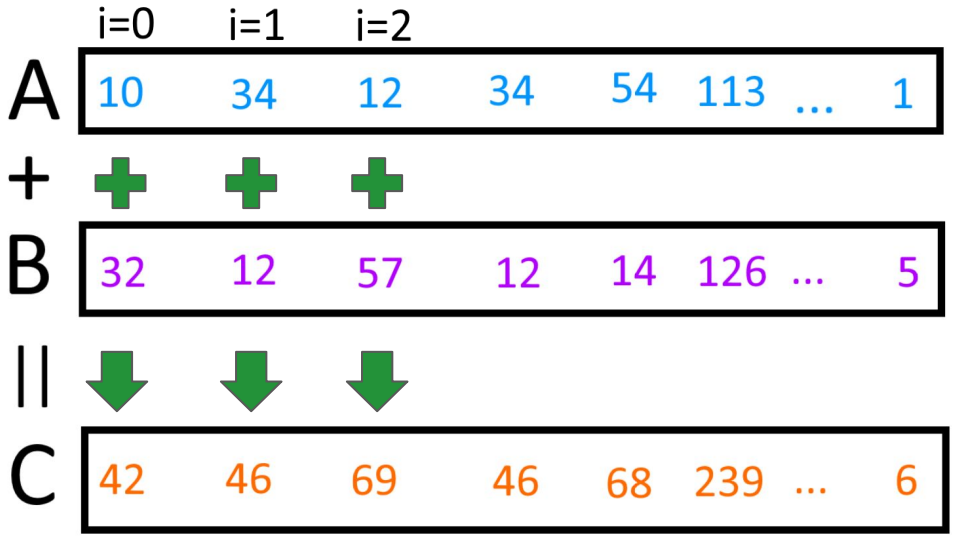

```
void solveCPU(int[] a, int[] b, int c[], int n) {
    for (int i = 0; i < n; ++i) {
        int sum = a[i] + b[i];
        c[i] = sum;
    }
}
```

Пример: A + B



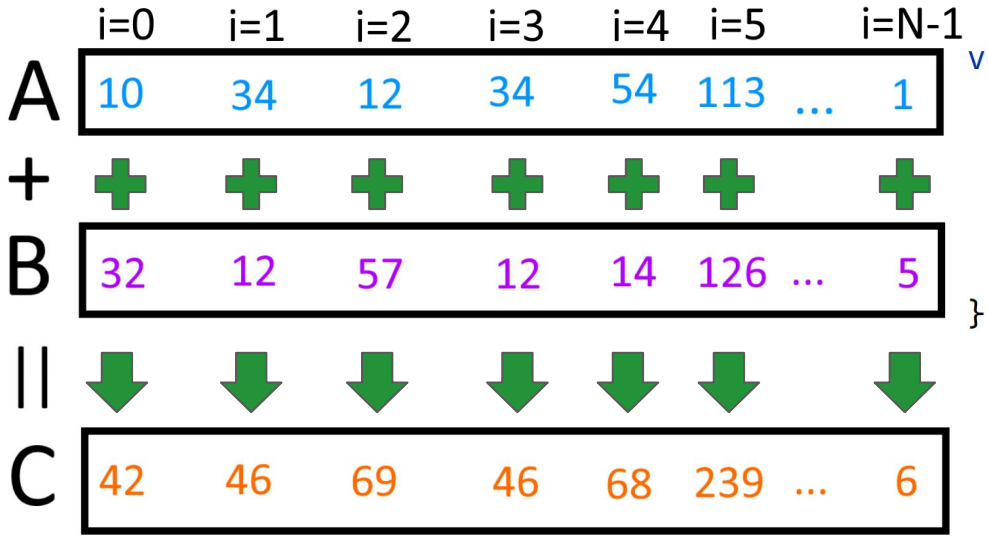

```
void solveCPU(int[] a, int[] b, int c[], int n) {
    for (int i = 0; i < n; ++i) {
        int sum = a[i] + b[i];
        c[i] = sum;
    }
}
```

Пример: A + B




```
void solveCPU(int[] a, int[] b, int c[], int n) {
    for (int i = 0; i < n; ++i) {
        int sum = a[i] + b[i];
        c[i] = sum;
    }
}
```

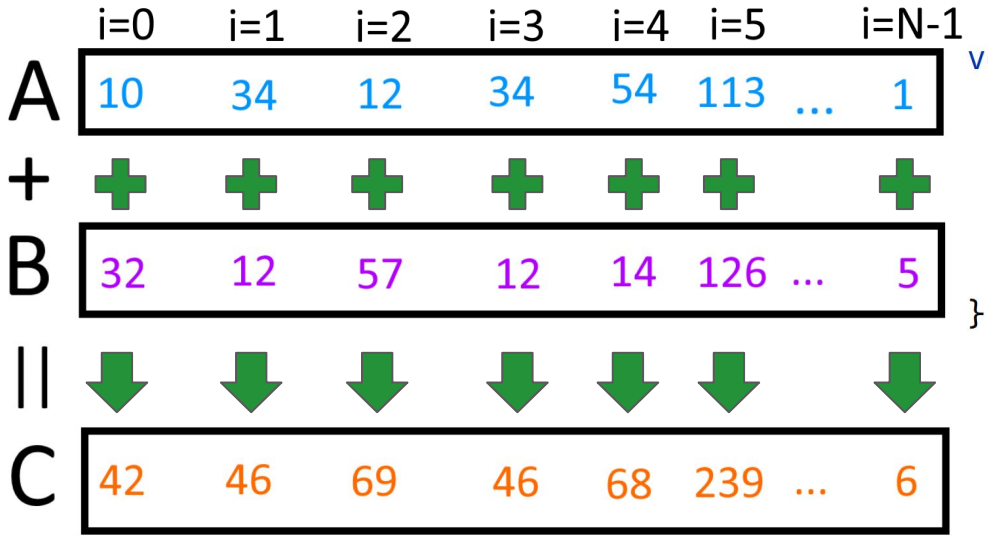
Пример: A + B




```
void solveCPU(int[] a, int[] b, int c[], int n) {
    for (int i = 0; i < n; ++i) {
        int sum = a[i] + b[i];
        c[i] = sum;
    }
}
```

Какая асимптотика?

Пример: A + B




```
void solveCPU(int[] a, int[] b, int c[], int n) {
    for (int i = 0; i < n; ++i) {
        int sum = a[i] + b[i];
        c[i] = sum;
    }
}
```

O(N)

Пример: A + B

A	10	34	12	34	54	113	...	1
+								
B	32	12	57	12	14	126	...	5
||								
C	42	46	69	46	68	239	...	6

NVIDIA RTX 4090 - 16 тысяч ядер!

OpenCL™ NVIDIA CUDA Vulkan

Пример: A + B

A	10	34	12	34	54	113	...	1
+								
B	32	12	57	12	14	126	...	5
||								
C	42	46	69	46	68	239	...	6

```
void solveCPU(int[] a, int[] b, int c[], int n) {
    for (int i = 0; i < n; ++i) {
        int sum = a[i] + b[i];
        c[i] = sum;
    }
}
```

O(N)

```
void solveGPU(int[] a, int[] b, int c[], int n) {
    const int i = get_global_id(0);
    c[i] = b[i] + a[i];
}
```

Супер многопоточно!

NVIDIA RTX 4090 - 16 тысяч ядер!

OpenCL™ NVIDIA CUDA Vulkan

Пример: A + B

A	10	34	12	34	54	113	...	1
+								
B	32	12	57	12	14	126	...	5
||								
C	42	46	69	46	68	239	...	6

```
void solveCPU(int[] a, int[] b, int c[], int n) {
    for (int i = 0; i < n; ++i) {
        int sum = a[i] + b[i];
        c[i] = sum;
    }
}
```

O(N)

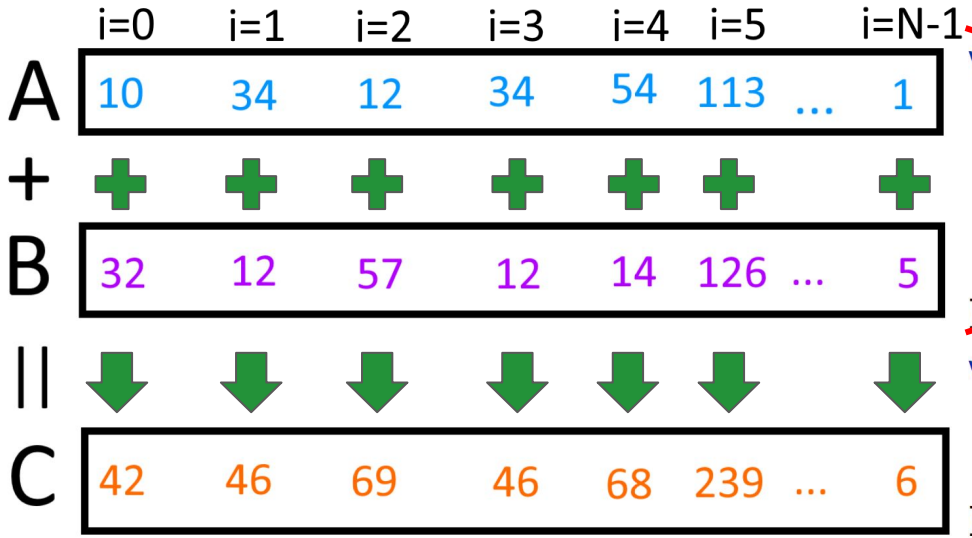
```
void solveGPU(int[] a, int[] b, int c[], int n) {
    const int i = get_global_id(0);
    c[i] = b[i] + a[i];
}
```

Какая асимптотика?

NVIDIA RTX 4090 - 16 тысяч ядер!

OpenCL™ NVIDIA CUDA Vulkan

Пример: A + B




```
void solveCPU(int[] a, int[] b, int c[], int n) {
    for (int i = 0; i < n; ++i) {
        int sum = a[i] + b[i];
        c[i] = sum;
    }
}
```

O(N)

```
void solveGPU(int[] a, int[] b, int c[], int n) {
    const int i = get_global_id(0);
    c[i] = b[i] + a[i];
}
```

O(N / 16384)

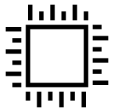
NVIDIA RTX 4090 - 16 тысяч ядер!

OpenCL™ NVIDIA CUDA Vulkan

Пример: A + B (N=100.000.000)

```
for (size_t i = 0; i < n; ++i) {
    cs[i] = as[i] + bs[i];
}
```

Пример: A + B (N=100.000.000)

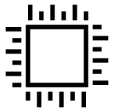


CPU - Intel 13700K

```
for (size_t i = 0; i < n; ++i) {
    cs[i] = as[i] + bs[i];
}
```

```
a + b median time: 0.068 sec (+-0.00481664)
a + b median RAM bandwidth: 16.4351 GB/s
```

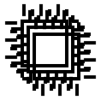
Пример: A + B (N=100.000.000)



CPU - Intel 13700K

```
for (size_t i = 0; i < n; ++i) {
    cs[i] = as[i] + bs[i];
}
```

```
a + b median time: 0.068 sec (+-0.00481664)
a + b median RAM bandwidth: 16.4351 GB/s
```

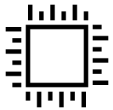


CPU - Intel 13700K

```
#pragma omp parallel for
for (ptrdiff_t i = 0; i < n; ++i) {
    cs[i] = as[i] + bs[i];
}
```

```
a + b median time: 0.047 sec (+-0.00153623)
a + b median RAM bandwidth: 23.7784 GB/s
```

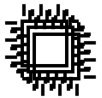
Пример: A + B (N=100.000.000)



CPU - Intel 13700K

```
for (size_t i = 0; i < n; ++i) {
    cs[i] = as[i] + bs[i];
}
```

```
a + b median time: 0.068 sec (+-0.00481664)
a + b median RAM bandwidth: 16.4351 GB/s
```



CPU - Intel 13700K

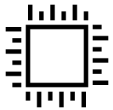
```
#pragma omp parallel for
for (ptrdiff_t i = 0; i < n; ++i) {
    cs[i] = as[i] + bs[i];
}
```

```
a + b median time: 0.047 sec (+-0.00153623)
a + b median RAM bandwidth: 23.7784 GB/s
```

Почему разница такая незначительная?

Как проверить гипотезу?
Как спровоцировать значительную разницу?

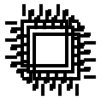
Пример: **100*cos(A + B)** (N=100.000.000)



CPU - Intel 13700K

```
for (size_t i = 0; i < n; ++i) {
    cs[i] = 100.0*cos( Left: as[i] + bs[i]);
}
```

```
median time: 1.178 sec (+-0.0235009)
median RAM bandwidth: 0.948716 GB/s
```



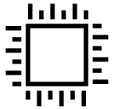
CPU - Intel 13700K

```
#pragma omp parallel for
for (ptrdiff_t i = 0; i < n; ++i) {
    cs[i] = 100*cos( Left: as[i] + bs[i]);
}
```

```
median time: 0.109 sec (+-0.0121988)
median RAM bandwidth: 10.2531 GB/s
```

x10.9

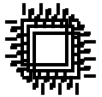
Пример: A + B (N=100.000.000)



CPU - Intel 13700K

```
for (size_t i = 0; i < n; ++i) {
    cs[i] = as[i] + bs[i];
}
```

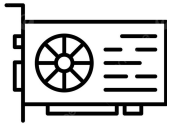
```
a + b median time: 0.068 sec (+-0.00481664)
a + b median RAM bandwidth: 16.4351 GB/s
```



CPU - Intel 13700K

```
#pragma omp parallel for
for (ptrdiff_t i = 0; i < n; ++i) {
    cs[i] = as[i] + bs[i];
}
```

```
a + b median time: 0.047 sec (+-0.00153623)
a + b median RAM bandwidth: 23.7784 GB/s
```

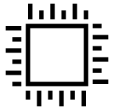


GPU - NVIDIA RTX 4090

```
__kernel void aplusb(__global const float* a,
                     __global const float* b,
                     __global       float* c,
                     unsigned int n)
{
    const unsigned int index = get_global_id( dimindx: 0);
    if (index >= n)
        return;

    c[index] = a[index] + b[index];
}
```

```
a + b median time: 0.002 sec (+-0.000632456)
a + b median VRAM bandwidth: 558.794 GB/s
```

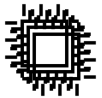
Пример: A + B (N=100.000.000)



CPU - Intel 13700K

```
for (size_t i = 0; i < n; ++i) {
    cs[i] = as[i] + bs[i];
}
```

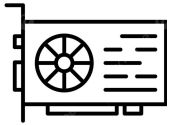
```
a + b median time: 0.068 sec (+-0.00481664)
a + b median RAM bandwidth: 16.4351 GB/s
```



CPU - Intel 13700K

```
#pragma omp parallel for
for (ptrdiff_t i = 0; i < n; ++i) {
    cs[i] = as[i] + bs[i];
}
```

```
a + b median time: 0.047 sec (+-0.00153623)
a + b median RAM bandwidth: 23.7784 GB/s
```



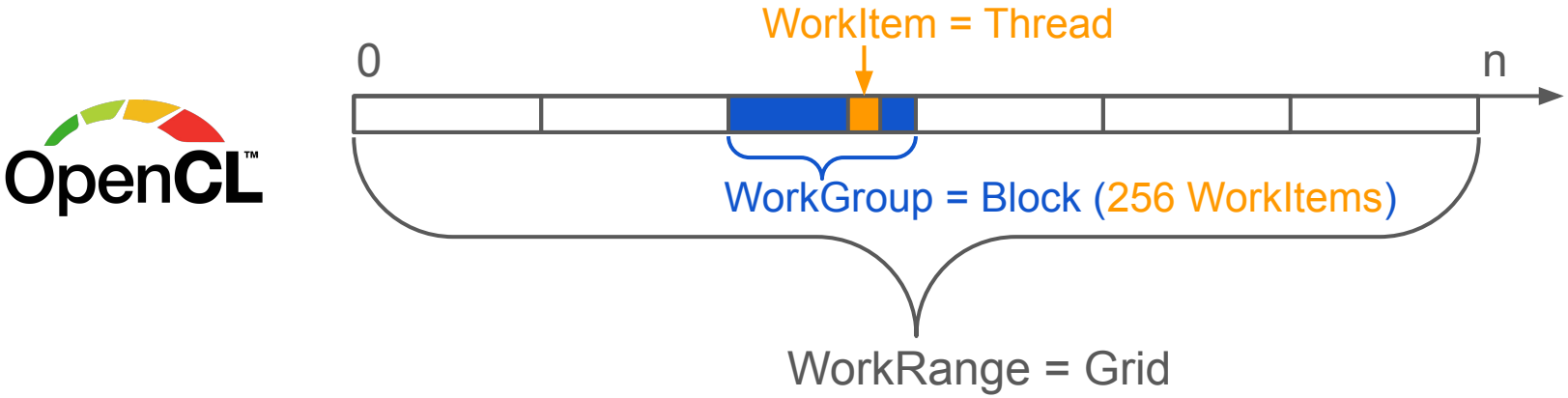
GPU - NVIDIA RTX 4090

```
__kernel void aplusb(__global const float* a,
                     __global const float* b,
                     __global       float* c,
                     unsigned int n)
{
    const unsigned int index = get_global_id( dimindx: 0);
    if (index >= n)
        return;

    c[index] = a[index] + b[index];
}
```

x23.5

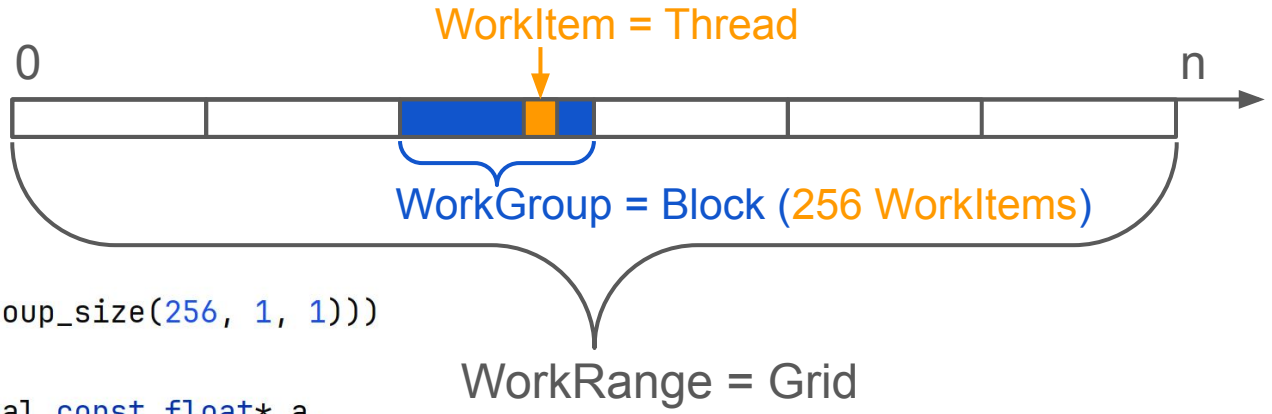
```
a + b median time: 0.002 sec (+-0.000632456)
a + b median VRAM bandwidth: 558.794 GB/s
```

Пример: A + B (N=100.000.000)



Пример: A + B (N=100.000.000)

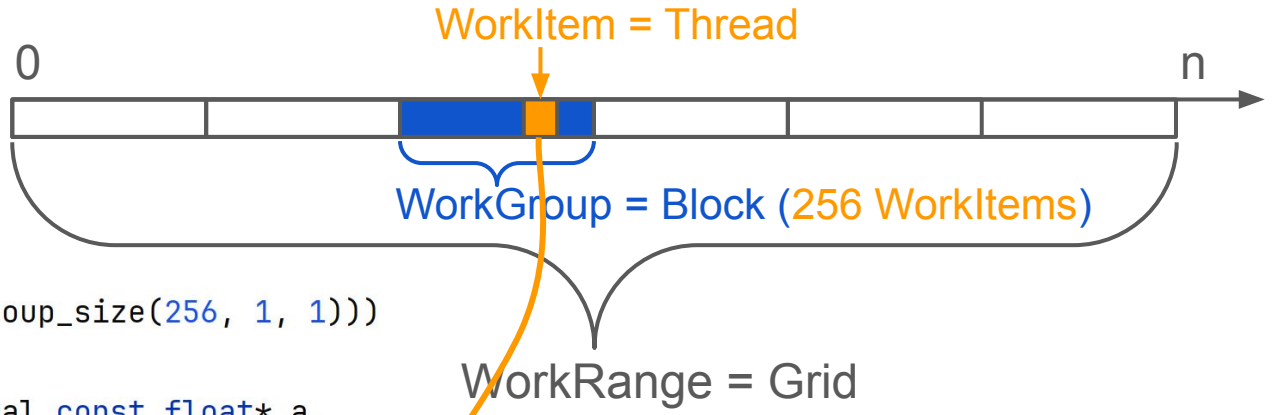
```
#ifdef __CLION_IDE__
#include <libgpu/opencl/cl/clion_defines.cl>
#endif

#line 5
```




```
__attribute__((reqd_work_group_size(256, 1, 1)))

__kernel void aplusb(__global const float* a,
                     __global const float* b,
                     __global       float* c,
                     unsigned int n)
{
    const unsigned int index = get_global_id( dimindx: 0);
    if (index >= n)
        return;

    c[index] = a[index] + b[index];
}
```

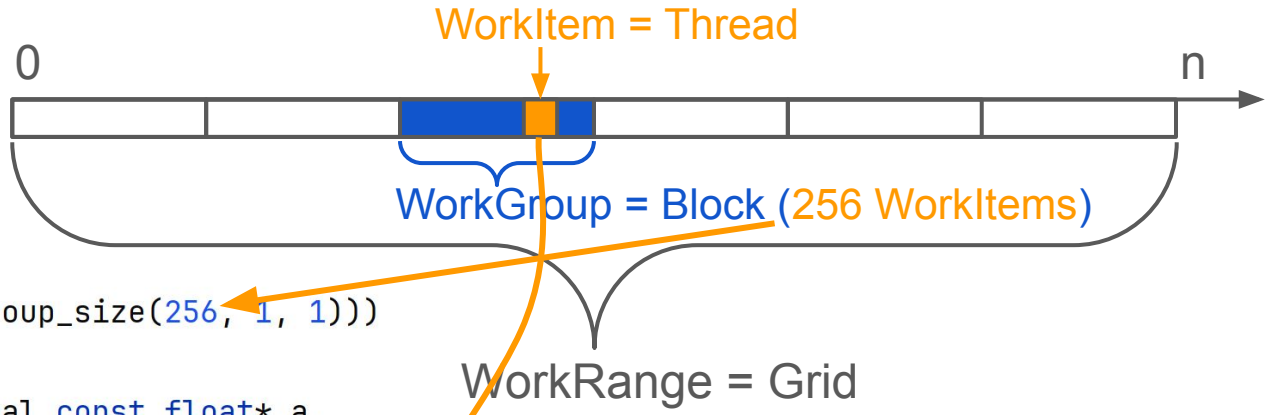
Пример: A + B (N=100.000.000)

```
#ifdef __CLION_IDE__
#include <libgpu/opencl/cl/clion_defines.cl>
#endif

#line 5

__attribute__((reqd_work_group_size(256, 1, 1)))

__kernel void aplusb(__global const float* a,
                     __global const float* b,
                     __global       float* c,
                     unsigned int n)
{
    const unsigned int index = get_global_id( dimindx: 0);
    if (index >= n)
        return;

    c[index] = a[index] + b[index];
}
```



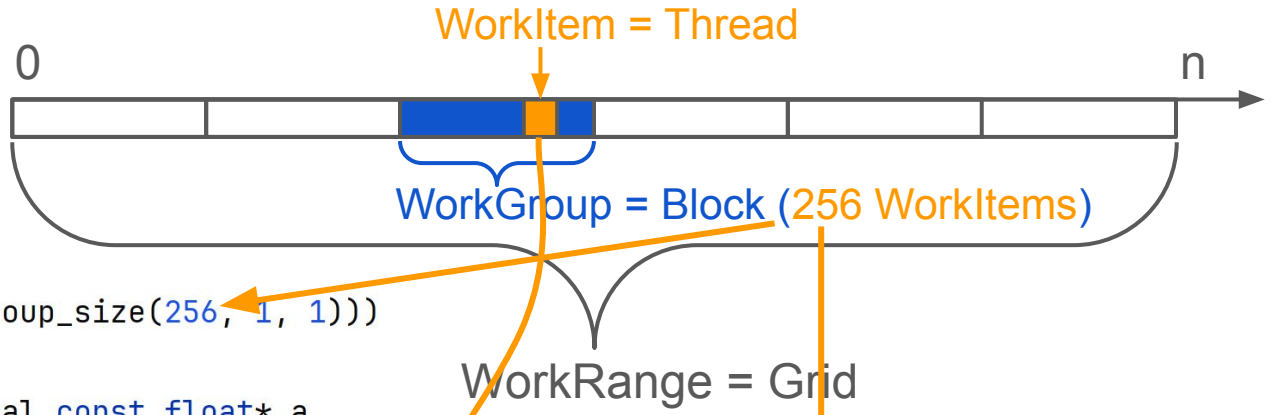
Пример: A + B (N=100.000.000)

```
#ifdef __CLION_IDE__
#include <libgpu/opencl/cl/clion_defines.cl>
#endif

#line 5

__attribute__((reqd_work_group_size(256, 1, 1)))

__kernel void aplusb(__global const float* a,
                     __global const float* b,
                     __global       float* c,
                     unsigned int n)
{
    const unsigned int index = get_global_id( dimindx: 0);
    if (index >= n)
        return;

    c[index] = a[index] + b[index];
}
```



Пример: A + B (N=100.000.000)

```
#ifdef __CLION_IDE__
#include <libgpu/opencl/cl/clion_defines.cl>
#endif

#line 5

__attribute__((reqd_work_group_size(256, 1, 1)))

__kernel void aplusb(__global const float* a,
                     __global const float* b,
                     __global       float* c,
                     unsigned int n)
{
    const unsigned int index = get_global_id( dimindx: 0);
    if (index >= n)
        return;

    c[index] = a[index] + b[index];
}
```

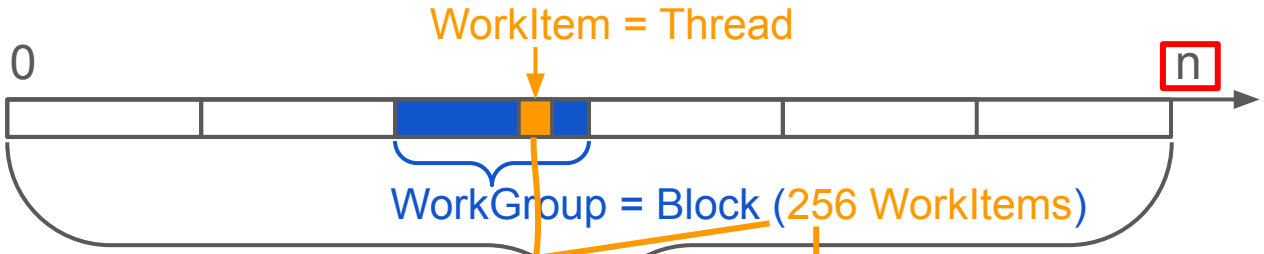


HOST-side (CPU, C++)

```
gpu::WorkSize workSize(GROUP_SIZE, n);
ocl_aplusb.exec(workSize, a_gpu, b_gpu, c_gpu, n);
```

Пример: A + B (N=100.000.000)

```
#ifdef __CLION_IDE__
#include <libgpu/opencl/cl/clion_defines.cl>
#endif

#line 5
```



А где задается размер рабочего пространства?

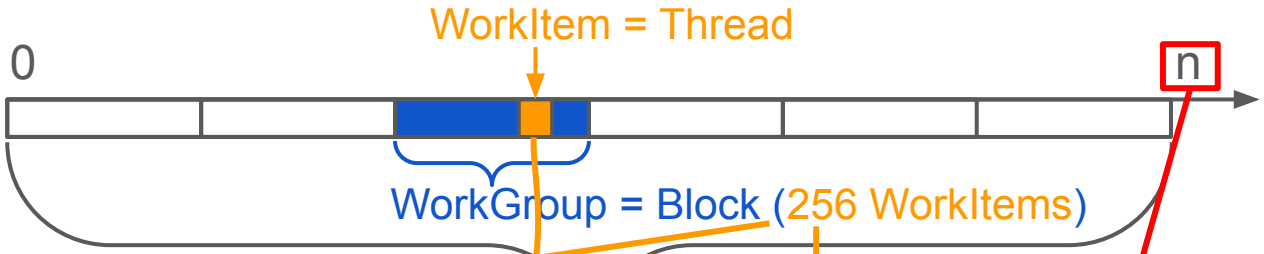
```
__attribute__((reqd_work_group_size(256, 1, 1)))

__kernel void aplusb(__global const float* a,
                     __global const float* b,
                     __global       float* c,
                     unsigned int n)
{
    const unsigned int index = get_global_id( dimindx: 0);
    if (index >= n)
        return;

    c[index] = a[index] + b[index];
}
```

HOST-side (CPU, C++)

```
gpu::WorkSize workSize(GROUP_SIZE, n);
ocl_aplusb.exec(workSize, a_gpu, b_gpu, c_gpu, n);
```

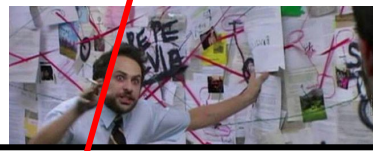
Пример: A + B (N=100.000.000)

```
#ifdef __CLION_IDE__
#include <libgpu/opencl/cl/clion_defines.cl>
#endif

#line 5
```



А где задается размер рабочего пространства?

```
__attribute__((reqd_work_group_size(256, 1, 1)))

__kernel void aplusb(__global const float* a,
                     __global const float* b,
                     __global       float* c,
                     unsigned int n)
{
    const unsigned int index = get_global_id( dimindx: 0);
    if (index >= n)
        return;

    c[index] = a[index] + b[index];
}
```

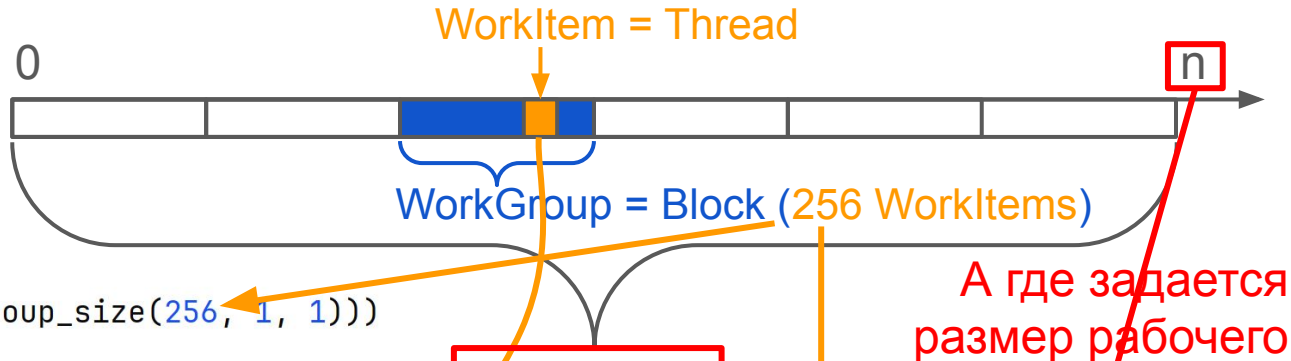


HOST-side (CPU, C++)

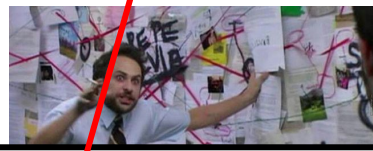
```
gpu::WorkSize workSize(GROUP_SIZE, n);
ocl_aplusb.exec(workSize, a_gpu, b_gpu, c_gpu, n);
```

Пример: A + B (N=100.000.000)

```
#ifdef __CLION_IDE__
#include <libgpu/opencl/cl/clion_defines.cl>
#endif

#line 5

__attribute__((reqd_work_group_size(256, 1, 1)))

__kernel void aplusb(__global const float* a,
                     __global const float* b,
                     __global       float* c,
                     unsigned int n)
{
    const unsigned int index = get_global_id( dimindx: 0);
    if (index >= n)
        return;

    c[index] = a[index] + b[index];
}
```



А где задается размер рабочего пространства?

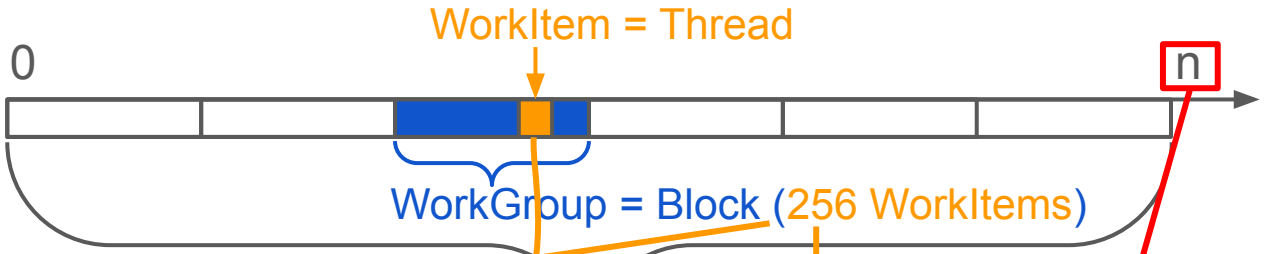


HOST-side (CPU, C++)

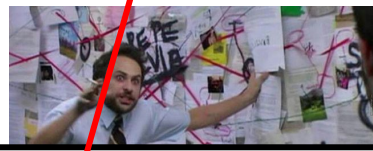
```
gpu::WorkSize workSize(GROUP_SIZE, n);
ocl_aplusb.exec(workSize, a_gpu, b_gpu, c_gpu, n);
```

Пример: A + B (N=100.000.000)

```
#ifdef __CLION_IDE__
#include <libgpu/opencl/cl/clion_defines.cl>
#endif

#line 5

__attribute__((reqd_work_group_size(256, 1, 1)))

__kernel void aplusb(__global const float* a,
                     __global const float* b,
                     __global       float* c,
                     unsigned int n)
{
    const unsigned int index = get_global_id( dimindx: 0);
    if (index >= n)
        return;

    c[index] = a[index] + b[index];
}
```



А где задается размер рабочего пространства?



```
HOST-side (CPU, C++)
gpu::WorkSize workSize(GROUP_SIZE, n);
ocl_aplusb.exec(workSize, a_gpu, b_gpu, c_gpu, n);
```

Пример: A + B (N=100.000.000)

OpenCL

```
#ifdef __CLION_IDE__
#include <libgpu/opencl/cl/clion_defines.cl>
#endif

#line 5


__attribute__((reqd_work_group_size(256, 1, 1)))

__kernel void aplusb(__global const float* a,
                     __global const float* b,
                     __global       float* c,
                     unsigned int n)
{
    const unsigned int index = get_global_id( dimindx: 0);
    if (index >= n)
        return;

    c[index] = a[index] + b[index];
}
```

NVIDIA CUDA

```
blockIdx.x * blockDim.x + threadIdx.x;


__global__ void aplusb(const float* a,
                       const float* b,
                       float* c,
                       unsigned int n)
{
    const unsigned int index = blockId
    if (index >= n)
        return;

    c[index] = a[index] + b[index];
}
```

Пример: A + B (N=100.000.000)


```
1   __global__ void aplusb(const float* a,
2                          const float* b,
3                          float* c,
4                          unsigned int n)
5   {
6       const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;
7       if (index >= n)
8           return;
9
10      c[index] = a[index] + b[index];
11  }
```

WorkGroup = **Block** (256 WorkItems)

Пример: A + B (N=100.000.000)

HOST-side (CPU, C++)

```
void cuda_aplusb(const gpu::WorkSize &workSize,
                 const float* a, const float* b, float* c, unsigned int n,
                 cudaStream_t stream)
{
    const unsigned int blockSize = 256;
    const unsigned int gridSize = (n + blockSize - 1) / blockSize;
    aplusb<<<gridSize, blockSize, 0, stream>>>(a, b, c, n);
    CUDA_CHECK_KERNEL(stream);
}
```

```
__global__ void aplusb(const float* a,
                       const float* b,
                       float* c,
                       unsigned int n)
{
    const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;
    if (index >= n)
        return;

    c[index] = a[index] + b[index];
}
```

WorkGroup = **Block** (256 WorkItems)

Пример: A + B (N=100.000.000)

HOST-side (CPU, C++)

```
void cuda_aplusb(const gpu::WorkSize &workSize,
                 const float* a, const float* b, float* c, unsigned int n,
                 cudaStream_t stream)
{
    const unsigned int blockSize = 256;
    const unsigned int gridSize = (n + blockSize - 1) / blockSize;
    aplusb<<<gridSize, blockSize, 0, stream>>>(a, b, c, n);
    CUDA_CHECK_KERNEL(stream);
}
```

NVIDIA CUDA

А где задается размер рабочего пространства?

```
__global__ void aplusb(const float* a,
                       const float* b,
                       float* c,
                       unsigned int n)
{
    const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;
    if (index >= n)
        return;

    c[index] = a[index] + b[index];
}
```

WorkGroup = **Block** (256 WorkItems)

Пример: A + B (N=100.000.000)

HOST-side (CPU, C++)

```
void cuda_aplusb(const gpu::WorkSize &workSize,
                 const float* a, const float* b, float* c, unsigned int n,
                 cudaStream_t stream)
{
    const unsigned int blockSize = 256;
    const unsigned int gridSize = (n + blockSize - 1) / blockSize;
    aplusb<<<gridSize, blockSize, 0, stream>>>(a, b, c, n);
    CUDA_CHECK_KERNEL(stream);
}
```

А где задается размер рабочего пространства?

```
__global__ void aplusb(const float* a,
                       const float* b,
                       float* c,
                       unsigned int n)
{
    const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;
    if (index >= n)
        return;

    c[index] = a[index] + b[index];
}
```

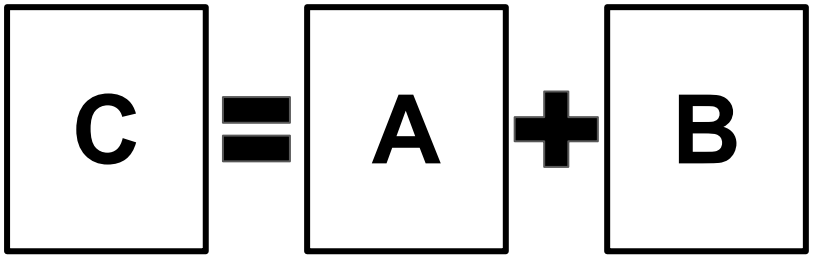
WorkGroup = **Block** (256 WorkItems)

Пример: A + B (N=100.000.000)

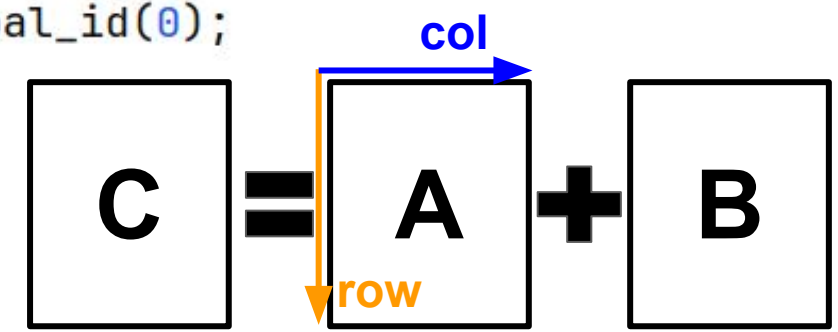
GLSL (Graphics Library Shading Language)


```
#version 450

#include <libgpu/vulkan/vk/common.vk>

layout (std430, binding = 0) readonly   buffer AsIn     {   uint as[];  };
layout (std430, binding = 1) readonly   buffer BsIn     {   uint bs[];  };
layout (std430, binding = 2) writeonly  buffer CsOut    {   uint cs[];  };

layout (push_constant) uniform PushConstants {
    uint n;
} params;

layout (local_size_x = 256) in;

void main()
{
    const uint index = gl_GlobalInvocationID.x;
    if (index >= params.n)
        return;

    cs[index] = as[index] + bs[index];
}
```

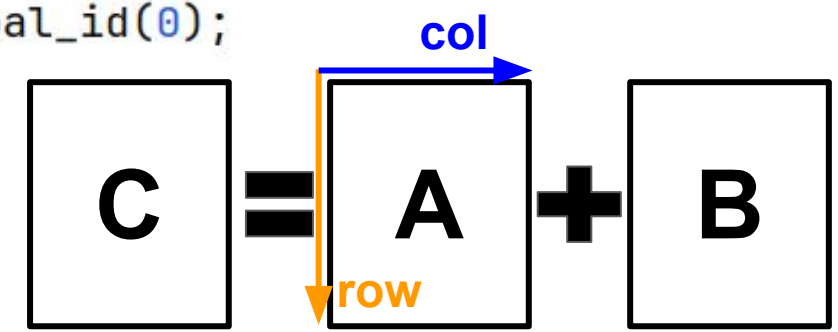
```
__kernel void aplusb_matrix(__global const uint* a,
                            __global const uint* b,
                            __global       uint* c,
                                unsigned int width,
                                unsigned int height)
{
    const unsigned int index = get_global_id(0);

    for (int k = 0; k < width; ++k) {
        unsigned int col = k;
        unsigned int row = index;
        c[row * width + col] = a[row * width + col]
                             + b[row * width + col];
    }
}
```



```
__kernel void aplusb_matrix(__global const uint* a,
                            __global const uint* b,
                            __global       uint* c,
                                unsigned int width,
                                unsigned int height)
{
    const unsigned int index = get_global_id(0);

    for (int k = 0; k < width; ++k) {
        unsigned int col = k;
        unsigned int row = index;
        c[row * width + col] = a[row * width + col]
                             + b[row * width + col];
    }
}
```



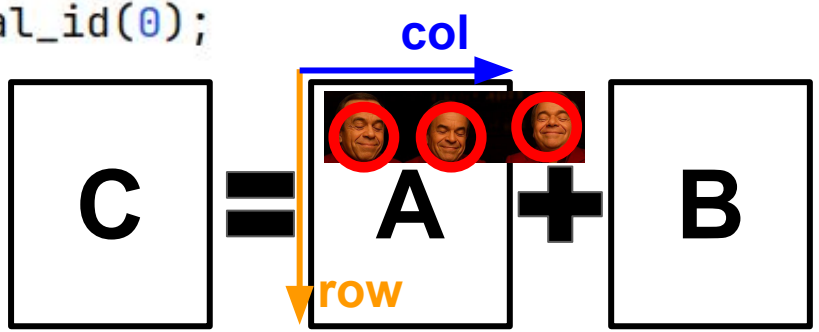
```
__kernel void aplusb_matrix(__global const uint* a,
                            __global const uint* b,
                            __global       uint* c,
                                unsigned int width,
                                unsigned int height)
{
    const unsigned int index = get_global_id(0);

    for (int k = 0; k < width; ++k) {
        unsigned int col = k;
        unsigned int row = index;
        c[row * width + col] = a[row * width + col]
                             + b[row * width + col];
    }
}
```



coalesced memory access?

```
__kernel void aplusb_matrix(__global const uint* a,
                            __global const uint* b,
                            __global       uint* c,
                                unsigned int width,
                                unsigned int height)
{
    const unsigned int index = get_global_id(0);

    for (int k = 0; k < width; ++k) {
        unsigned int col = k;
        unsigned int row = index;
        c[row * width + col] = a[row * width + col]
                             + b[row * width + col];
    }
}
```

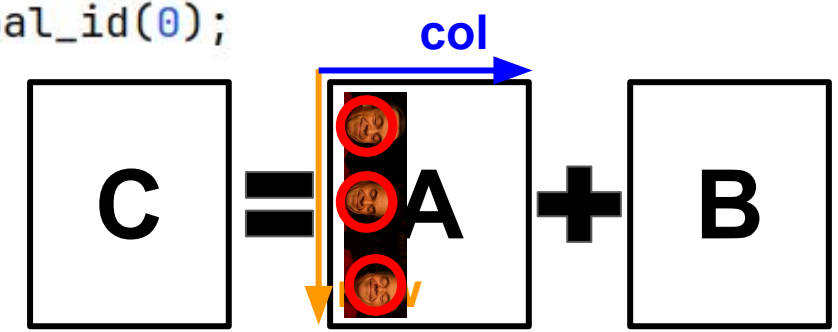
Как выглядит ***warp***?



coalesced memory access?

```
__kernel void aplusb_matrix(__global const uint* a,
                            __global const uint* b,
                            __global       uint* c,
                                unsigned int width,
                                unsigned int height)
{
    const unsigned int index = get_global_id(0);

    for (int k = 0; k < width; ++k) {
        unsigned int col = k;
        unsigned int row = index;
        c[row * width + col] = a[row * width + col]
                             + b[row * width + col];
    }
}
```

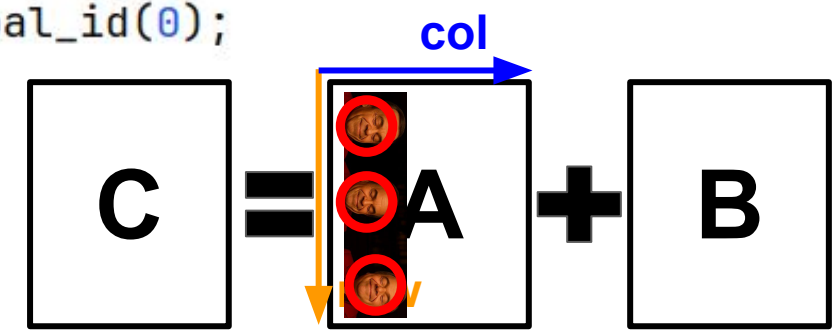
Как выглядит ***warp***?



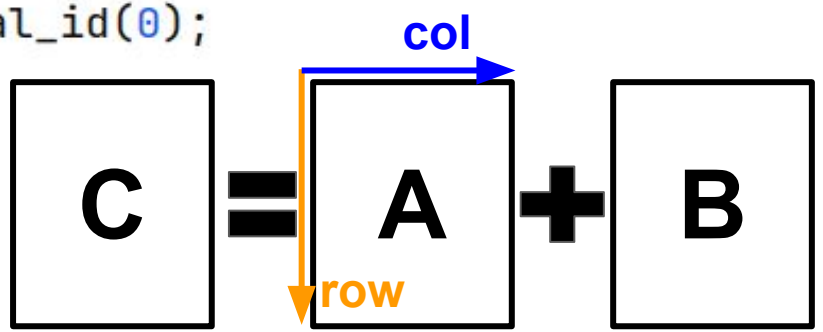
coalesced memory access?

```
__kernel void aplusb_matrix(__global const uint* a,
                            __global const uint* b,
                            __global       uint* c,
                                unsigned int width,
                                unsigned int height)
{
    const unsigned int index = get_global_id(0);

    for (int k = 0; k < width; ++k) {
        unsigned int col = k;
        unsigned int row = index;
        c[row * width + col] = a[row * width + col]
                             + b[row * width + col];
    }
}
```

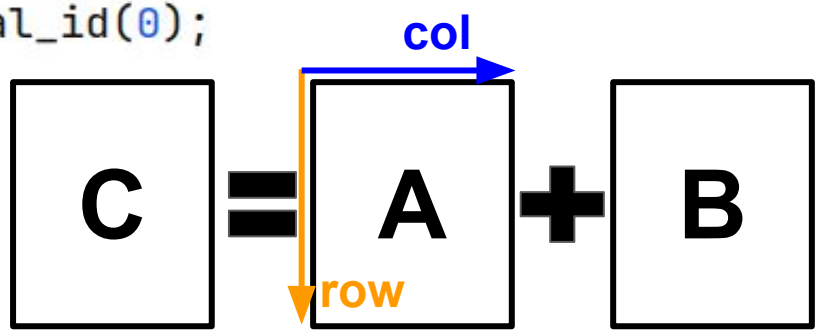
HE *coalesced memory access*!!!



```
__kernel void aplusb_matrix(__global const uint* a,
                            __global const uint* b,
                            __global       uint* c,
                                unsigned int width,
                                unsigned int height)
{
    const unsigned int index = get_global_id(0);

    for (int k = 0; k < height; ++k) {
        unsigned int col = index;
        unsigned int row = k;
        c[row * width + col] = a[row * width + col]
                             + b[row * width + col];
    }
}
```



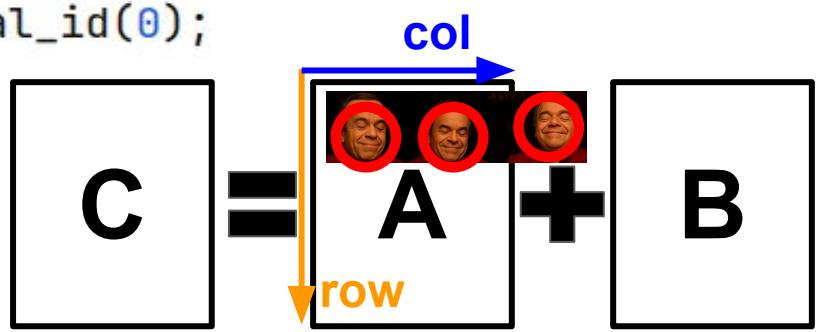
```
__kernel void aplusb_matrix(__global const uint* a,
                            __global const uint* b,
                            __global       uint* c,
                                unsigned int width,
                                unsigned int height)
{
    const unsigned int index = get_global_id(0);

    for (int k = 0; k < height; ++k) {
        unsigned int col = index;
        unsigned int row = k;
        c[row * width + col] = a[row * width + col]
                             + b[row * width + col];
    }
}
```



coalesced memory access?

```
__kernel void aplusb_matrix(__global const uint* a,
                            __global const uint* b,
                            __global       uint* c,
                                unsigned int width,
                                unsigned int height)
{
    const unsigned int index = get_global_id(0);

    for (int k = 0; k < height; ++k) {
        unsigned int col = index;
        unsigned int row = k;
        c[row * width + col] = a[row * width + col]
                             + b[row * width + col];
    }
}
```

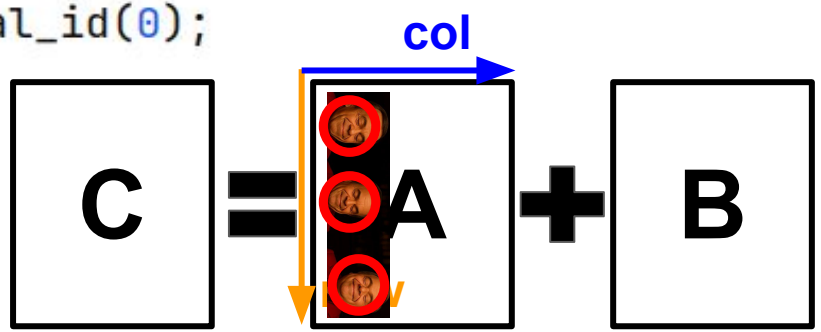
Как выглядит ***warp***?



coalesced memory access?

```
__kernel void aplusb_matrix(__global const uint* a,
                            __global const uint* b,
                            __global       uint* c,
                                unsigned int width,
                                unsigned int height)
{
    const unsigned int index = get_global_id(0);

    for (int k = 0; k < height; ++k) {
        unsigned int col = index;
        unsigned int row = k;
        c[row * width + col] = a[row * width + col]
                             + b[row * width + col];
    }
}
```

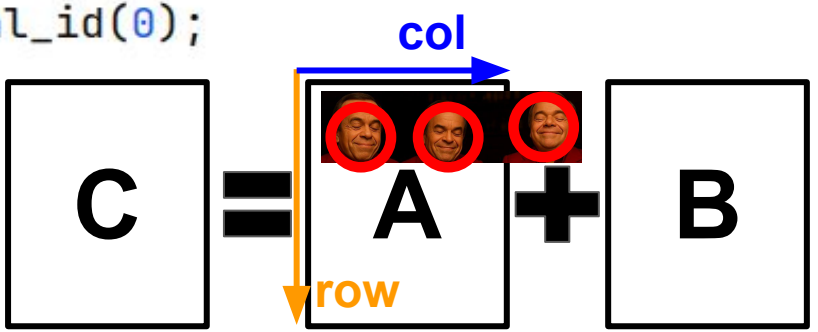
Как выглядит ***warp***?



coalesced memory access?

```
__kernel void aplusb_matrix(__global const uint* a,
                            __global const uint* b,
                            __global       uint* c,
                                unsigned int width,
                                unsigned int height)
{
    const unsigned int index = get_global_id(0);

    for (int k = 0; k < height; ++k) {
        unsigned int col = index;
        unsigned int row = k;
        c[row * width + col] = a[row * width + col]
                             + b[row * width + col];
    }
}
```

coalesced memory access!!!




CS Space
Клуб технологий и науки
Vulkan
NVIDIA
CUDA
OpenCL™
Вопросы?
@UnicornGlade
@PolarNick239
polarnick239@gmail.com
Николай Полярный
Activate Ubuntu
Go to Settings to activate Ubuntu.
206