Технология CUDA для высокопроизводительных вычислений на кластерах с GPU

Лихогруд Николай
n.lihogrud@gmail.com

Часть седьмая

CUDA-потоки

Когда их стоит использовать?

cudaStream

- Последовательность команд для GPU (запуски ядер, копирования памяти и т.д.), исполняемая строго последовательно
  - следующая команда выполняется после полного завершения предыдущей

cudaStream

- Пользователь сам создает потоки и распределяет команды по ним
- По-умолчанию, все команды помещаются в «Default Stream», равный нулю

cudaStream

- Только команды из разных потоков, отличных от потока по-умолчанию, могут выполняться параллельно

- Пользователь сам задет необходимую синхронизацию между командами из разных потоков (при наличии зависимостей)
  - В общем случае, порядок выполнения команд из разных потоков не определен

Асинхронное копирование

Когда возвращается управление хостовой нити

	Host->device		Device->host		host-host	dev-dev
	pageable	**pinned**	pageable	**pinned**		
memcpy	После копирования в буфер*	После полного завершения	После полного завершения	После полного завершения	После полного завершения	сразу
memcpyAsync	После копирования в буфер	**сразу**	После полного завершения	**сразу**	После полного завершения	сразу

*В начале работы неявно вызывается cudaDeviceSynchronize

Параллельное выполнение команд

Параллельная работа хоста и устройства

- Ядра выполняются асинхронно
- Копирование между pinned-памятью и памятью устройства при помощи `cudaMemcpyAsync` также выполняется асинхронно

=> Добиться параллельной работы хоста и устройства достаточно просто!

Параллельная работа хоста и устройства

Пример:

```
cudaMallocHost(&aHost, size);
cudaMemcpyAsync( aDev, aHost, size,
          cudaMemcpyHostToDevice);
kernel<<<grid, block>>>(aDev, bDev);
cudaMemcpyAsync( bHost,
          cudaMemcpyHostToDevice);
doSomeWorkOnHost();
```

doSomeWorkOnHost будет выполняться параллельно с копированиями и выполнением ядра

Параллельное выполнение команд на GPU

- Команды из разных потоков, отличных от потока по-умолчанию, могут исполняться параллельно
  - В зависимости от аппаратных возможностей

- Возможные случаи:
  - Параллельные копирование и выполнение ядра
  - Параллельные выполнение ядер
  - Параллельные копирования с хоста на устройство и с устройства на хост

Копирование & выполнение ядра

- Если `cudaDeviceProp::asyncEngineCount > 0` устройство может выполнять параллельно копирование и счет ядра
  - Хостовая память долна быть page-locked

```
cudaMallocHost(&aHost, size);
cudaStreamCreate(&stream1);
cudaStreamCreate(&stream2); cudaMemcpyAsync(
aDev, aHost, size,
          cudaMemcpyHostToDevice, stream1);
kernel<<<grid, block, 0, stream2>>>(…);
```

Параллельное выполнение ядер

- Если cudaDeviceProp::concurrentKernels > 0 устройство может выполнять ядра параллельно

```
cudaStreamCreate(&stream1);
cudaStreamCreate(&stream2); kernel1<<<grid,
block, 0,
                              stream1>>>(data_1);
kernel2<<<grid, block, 0,
                              stream2>>>(data_2);
```

Копирование в обе стороны

- Если cudaDeviceProp::asyncEngineCount == 2 устройство может выполнять параллельно копирование в обе стороны и счет ядра

```
cudaMallocHost(&aHost, size);
cudaMallocHost(&bHost, size);
// создать потоки
cudaMemcpyAsync( aDev, aHost, size,
          cudaMemcpyHostToDevice, stream1);
cudaMemcpyAsync( bHost, bDev, size,
          cudaMemcpyDeviceToHost, stream2);
kernel<<<grid, block, 0, stream3>>>(…);
```

Обозначения

- Ядро = код для GPU, __global void kernel(…) {}
- Размер ядра = длительность вычисления отдельной нитью
- Большое, сложное ядро = ядро большого размера
- Выполнение ядра = выполнение ядра на некотором гриде
- Запуск ядра = команда запуска ядра на гриде
- Длительный запуск ядра = запуск ядра, который долго выполняется
  - Время вычисления ~ размер ядра * размер грида

Примеры

- Рассмотрим классическую схему работы с GPU:
  - Копирование входных данных на GPU
  - Выполнение ядра
  - Копирование результатов обратно на хост
- Необходима синхронизация между командами, т.е. следующая выполняется после завершения предыдущей

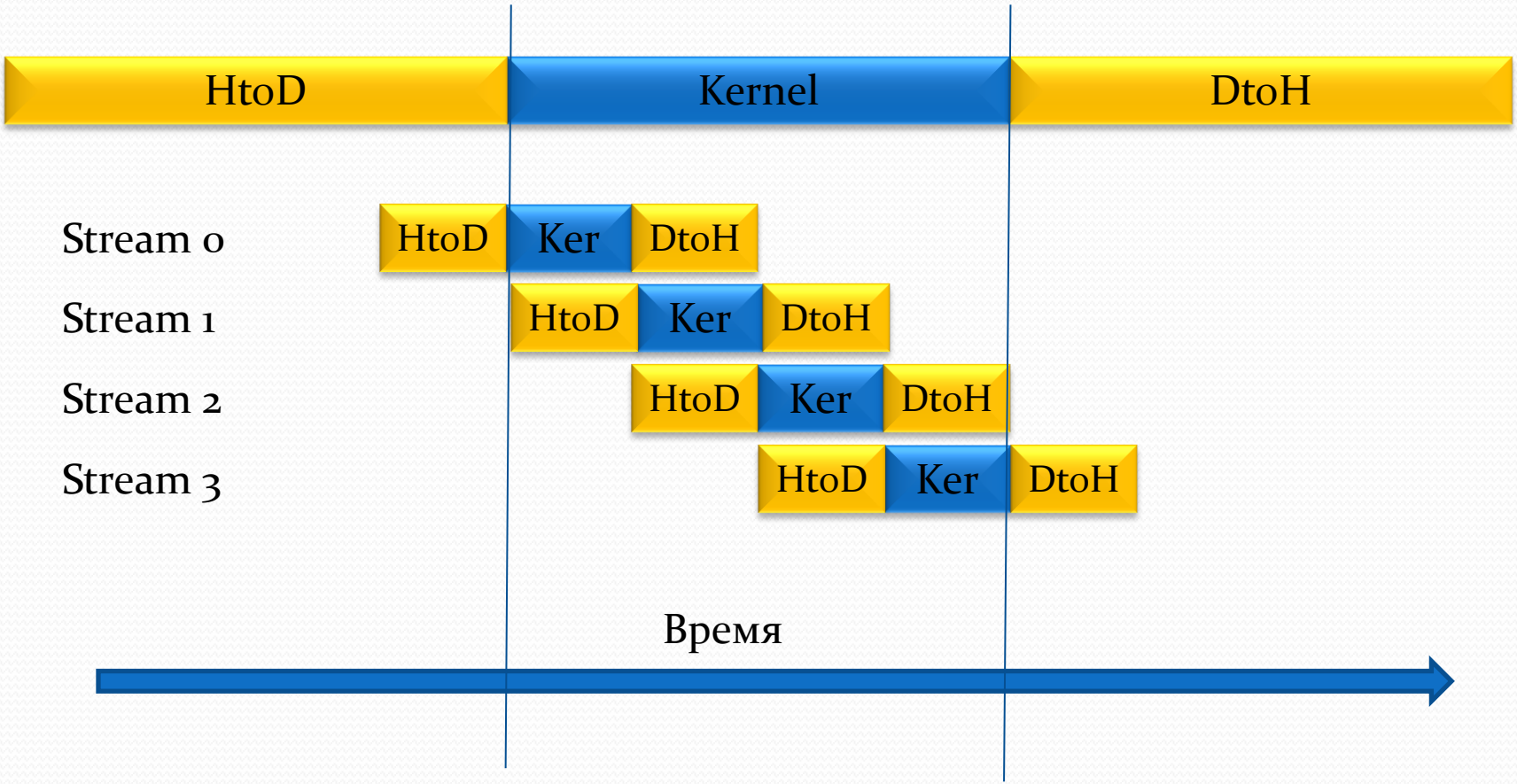
Идеальный случай

- Выполнения копирований сопоставимы по времени с выполнением ядра

- Разобьем задачу на подзадачи
  - Разделим грид на части и запустим то же ядро на подгридах – результат не изменится
  - Подзадаче нужна только часть данных для старта
  - Запустим подзадачи в разных потоках

Идеальный случай

- Запустим подзадачи в разных потоках



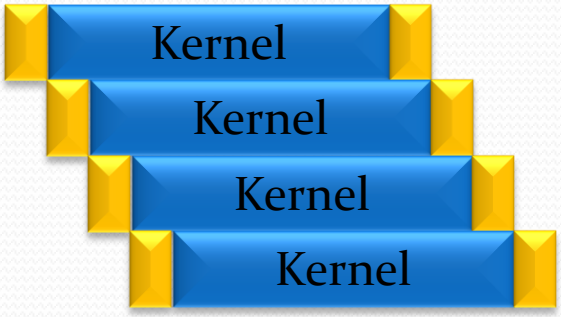
Идеальный случай

- Выполнение первой подзадачи начнется сразу после копирования нужной ей части данных
- Выполнение ядер будет происходить параллельно с копированиями
- Параллельное копирование в обе стороны, если аппаратура позволяет

Макимальное ускорение: $\dfrac{3}{1+\dfrac{2}{\text{число потоков}}}$

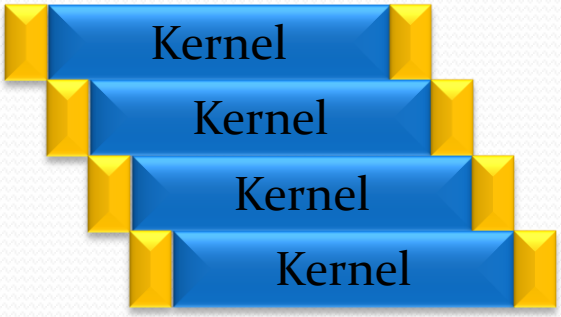
Небольшие копирования

Получится ли?



Небольшие копирования

Получится ли?



Нет!

Суммарное время выполнения ядра на подгридах никак не сделать меньше времени выполнения на целом гриде

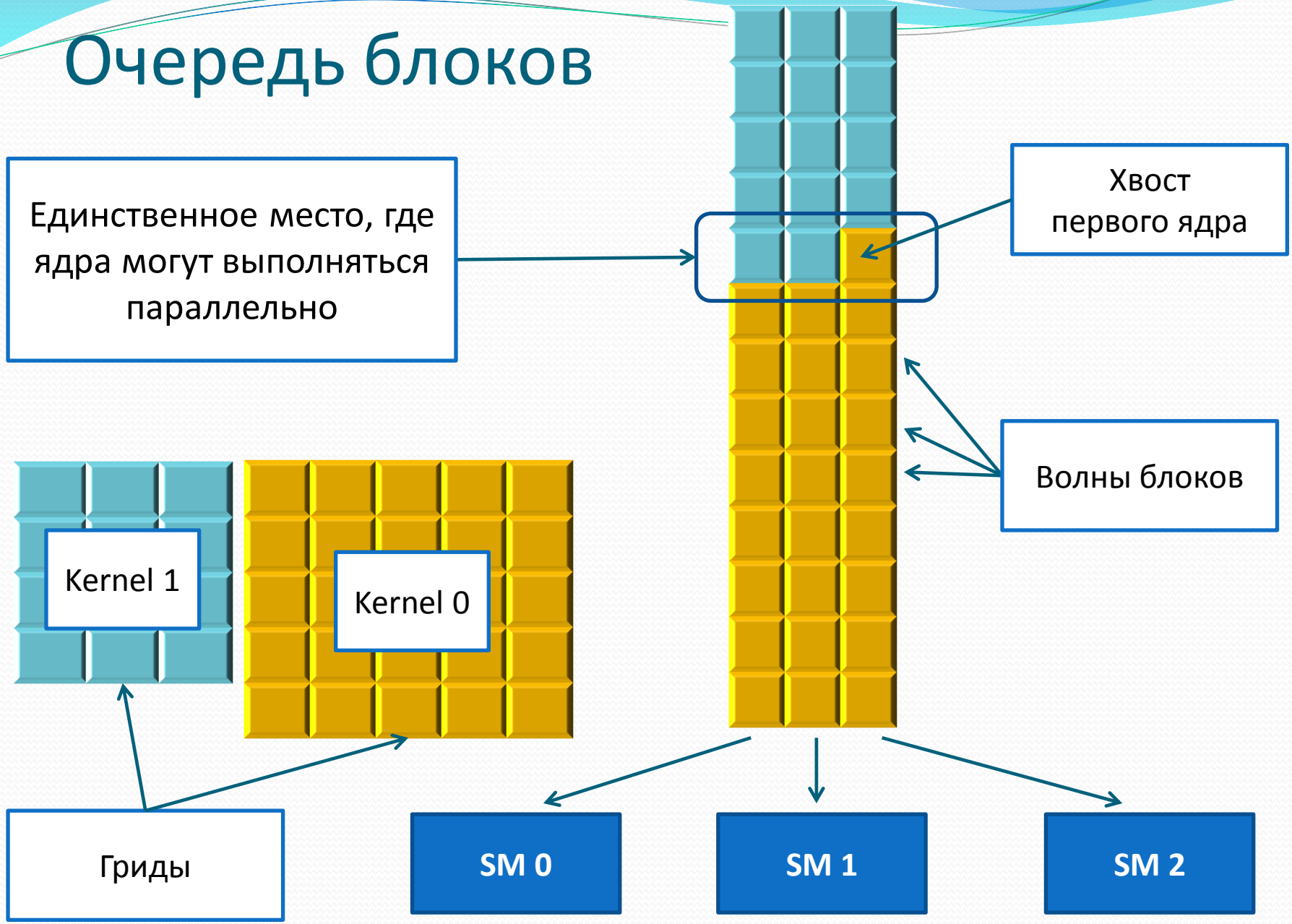
Параллельное выполнение ядер

- Ресурсы GPU ограничены – до 16 SM, до 1536 нитей на одном SM, до 8-ми блоков на SM

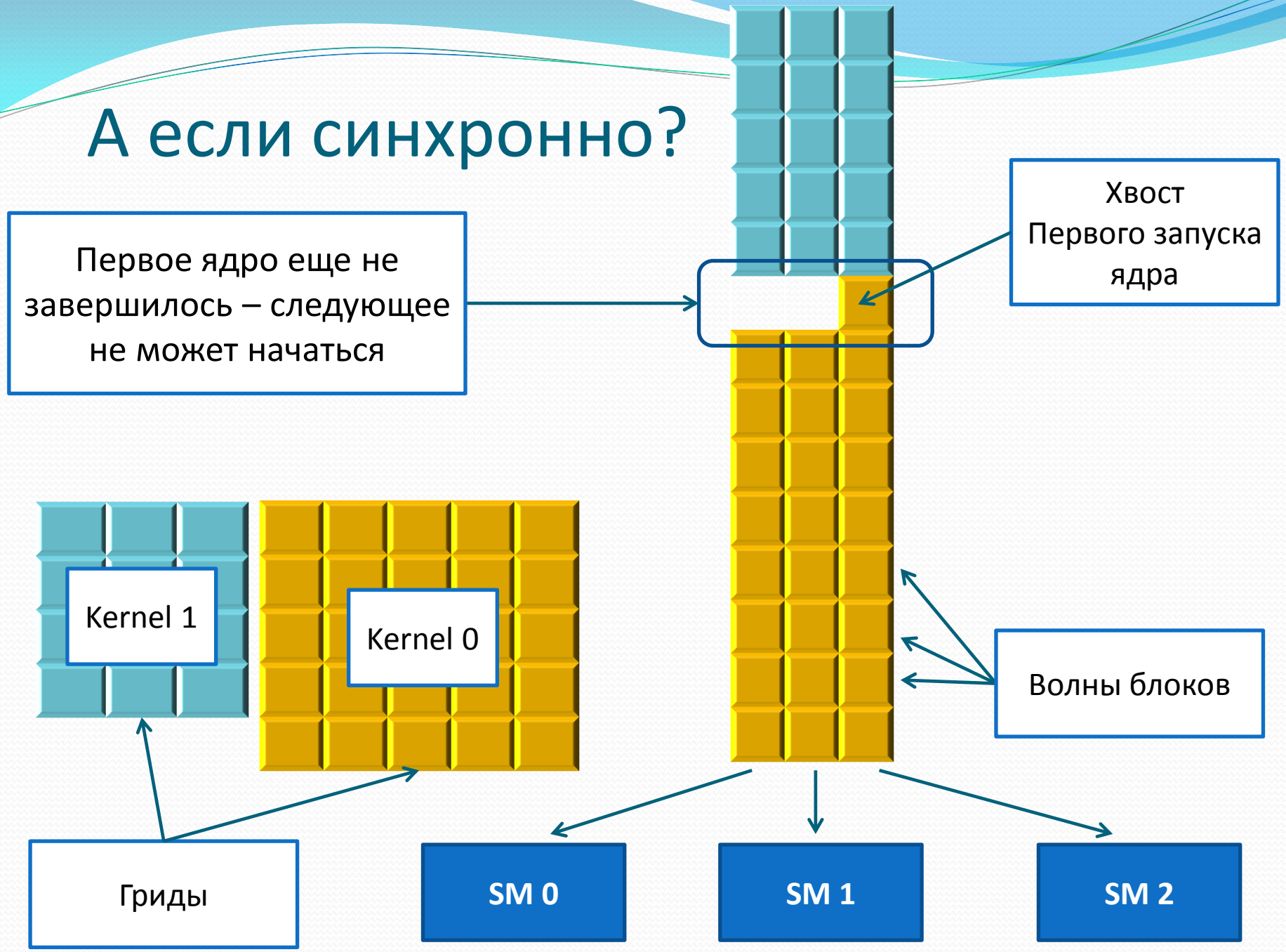
- Ядра запускаются на гридах из миллионов нитей = тысячи блоков
  - Блоки всех гридов попадают в одну общую очередь и выполняются по мере освобождения мультипроцессоров

Мультипроцессоры – «bottle neck» этой очереди

Очередь блоков

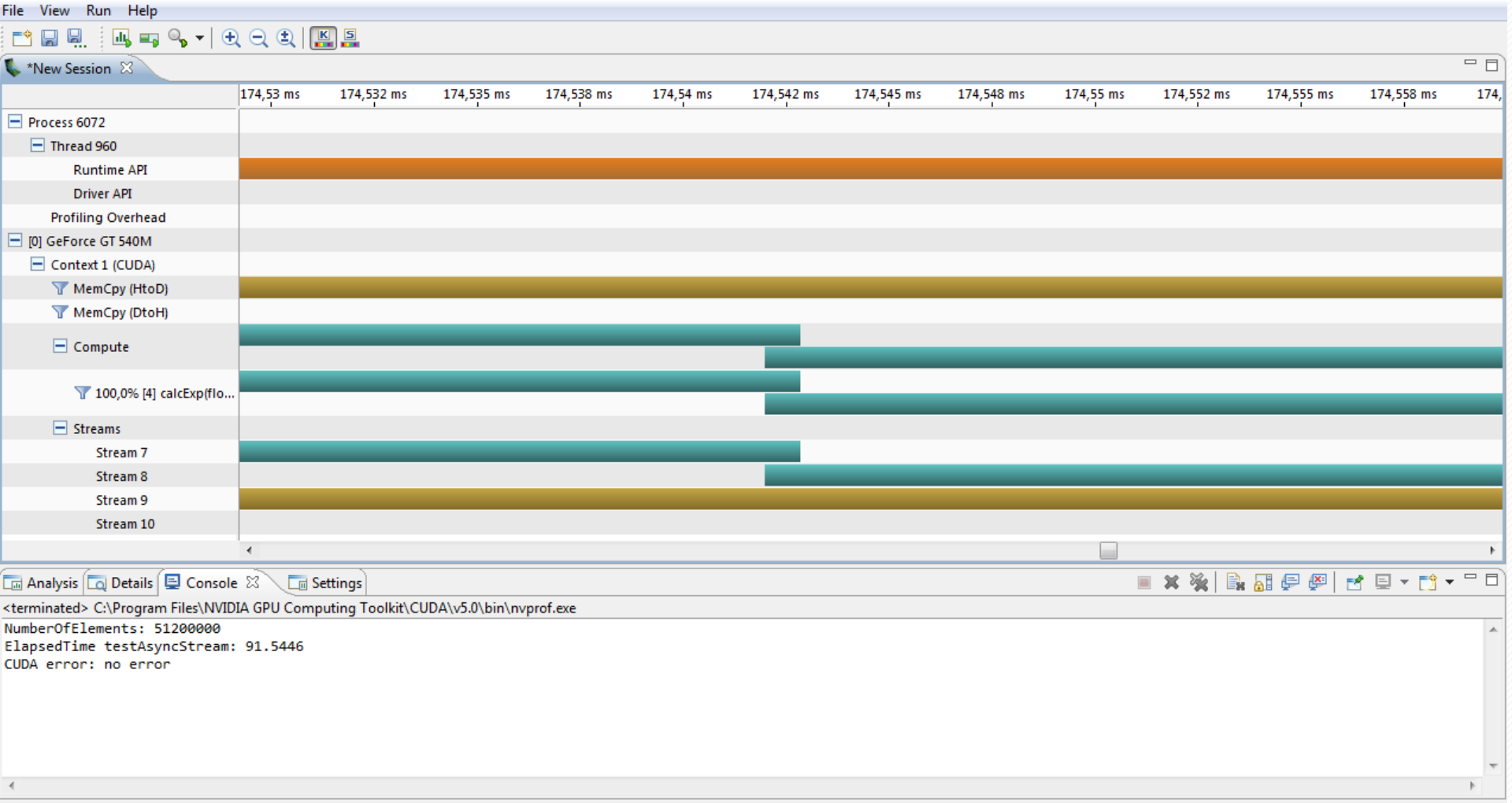


А если синхронно?



Пример в NVVP

- При большом приближении видно хвосты



Вывод

- При запуске двух ядер в разных потоках они могут выполняться параллельно только на границах
  - Когда последняя волна блоков (хвост) первого из запущенных ядер не полностью загружает устройство

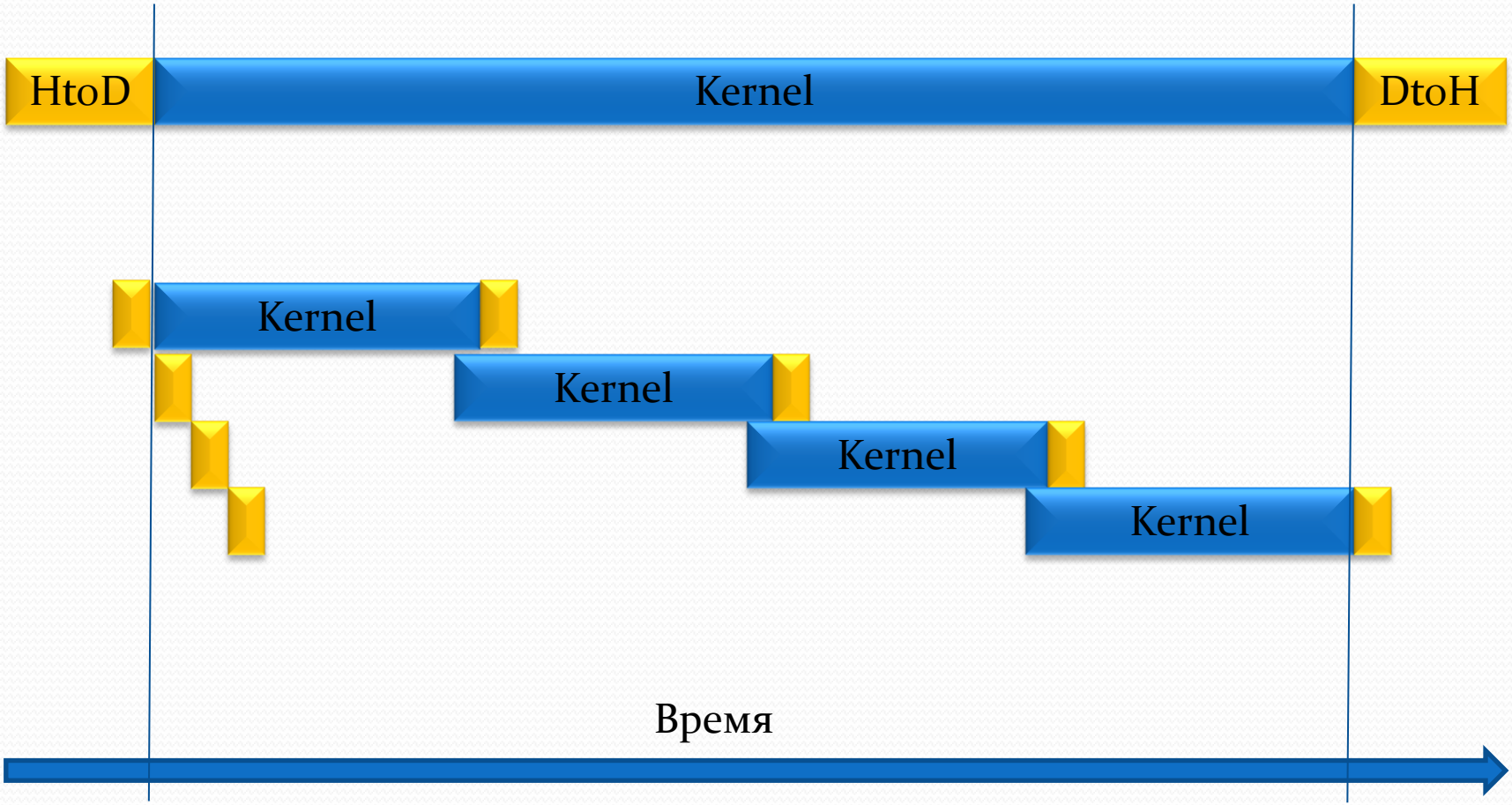
- Суммарное время выполнения одинаковых по сложности ядер, запущенных в разных потоках:

$$\frac{\text{суммарное число блоков} * \text{время выполнения блока}}{\text{число мультипроцессоров} * \left(\frac{1536}{\text{размер блока}} * occupancy\right)}$$

Вывод

- Суммарное время отдельных выполнений ядра на подгридах в разных потоках равно времени выполнения на целом гриде
  - Суммарное число блоков не меняется
- Суммарное время отдельных выполнений ядра на подгридах в одном потоке >= времени выполнения на целом гриде
  - Из-за простоя на границах

Небольшие копирования



Вывод

- При разбиении задачи на подзадачи выигрыш можем получить только за счет
  - Параллельного выполнения ядер и копирований
  - Параллельного выполнения копирований

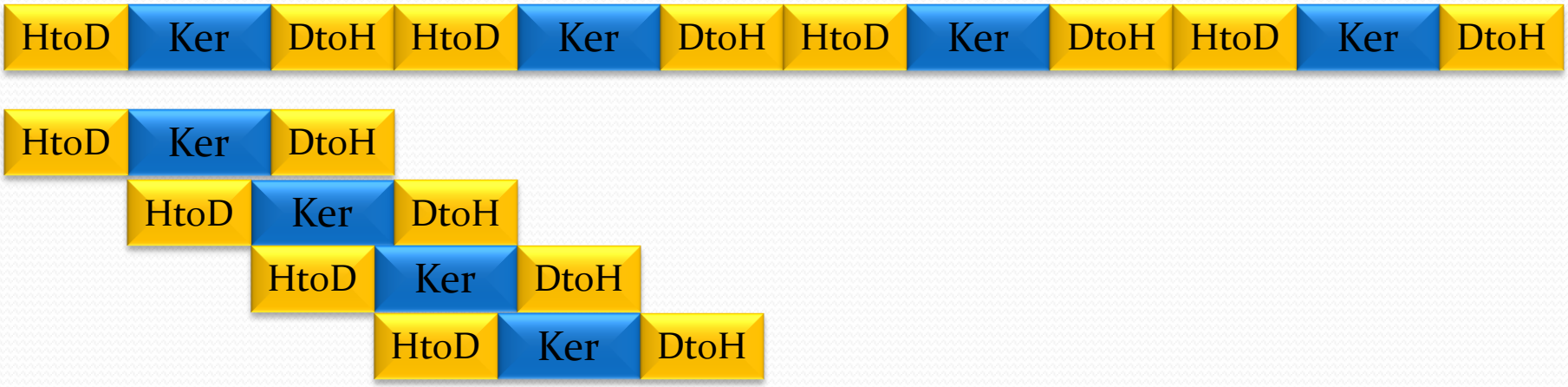
$\Rightarrow$При небольших копированиях

не имеет смысла возиться с потоками для подзадач

Множество задач

- При распределении по потокам результат зависит от
  - Соотношения копирования / ссчет
  - Размера гридов

Ядра запускаются на больших гридах



- Гриды большие => хвосты запусков ядер занимают малую долю в общем число блоков => доля параллельного выполнения невелика
  - Выигрываем в основном за счет параллельных копирований

Мало копирований – мало ускорения!

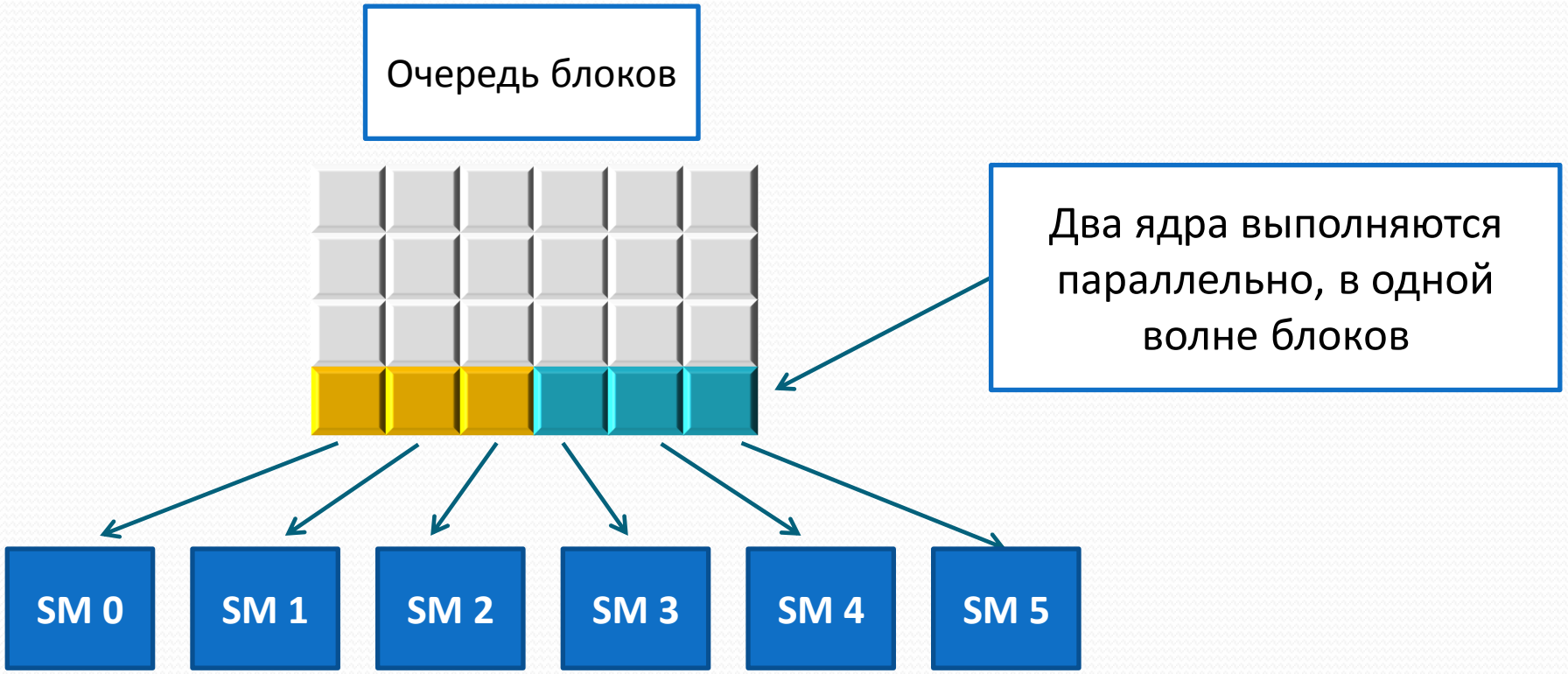
Ядра запускаются на малых гридах

- Пусть ядра запускаются на гридах такого размера, что ресурсов хватает для размещения всех блоков нескольких гридов
  - Например двух
- Пусть ядра примерно одинаковой сложности

Тогда два ядра, запущенные таких гридах, будут выполняться параллельно!

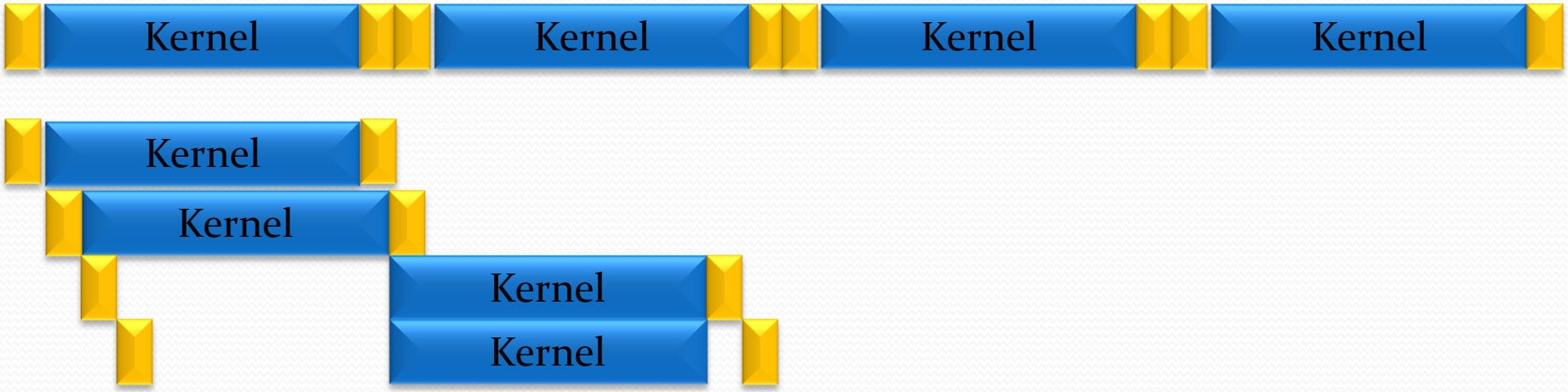
Ядра запускаются на малых гридах

- Ядра запускаются на трех блоках по 1024 нити, устройство состоит из 6 SM



Ядра запускаются на малых гридах

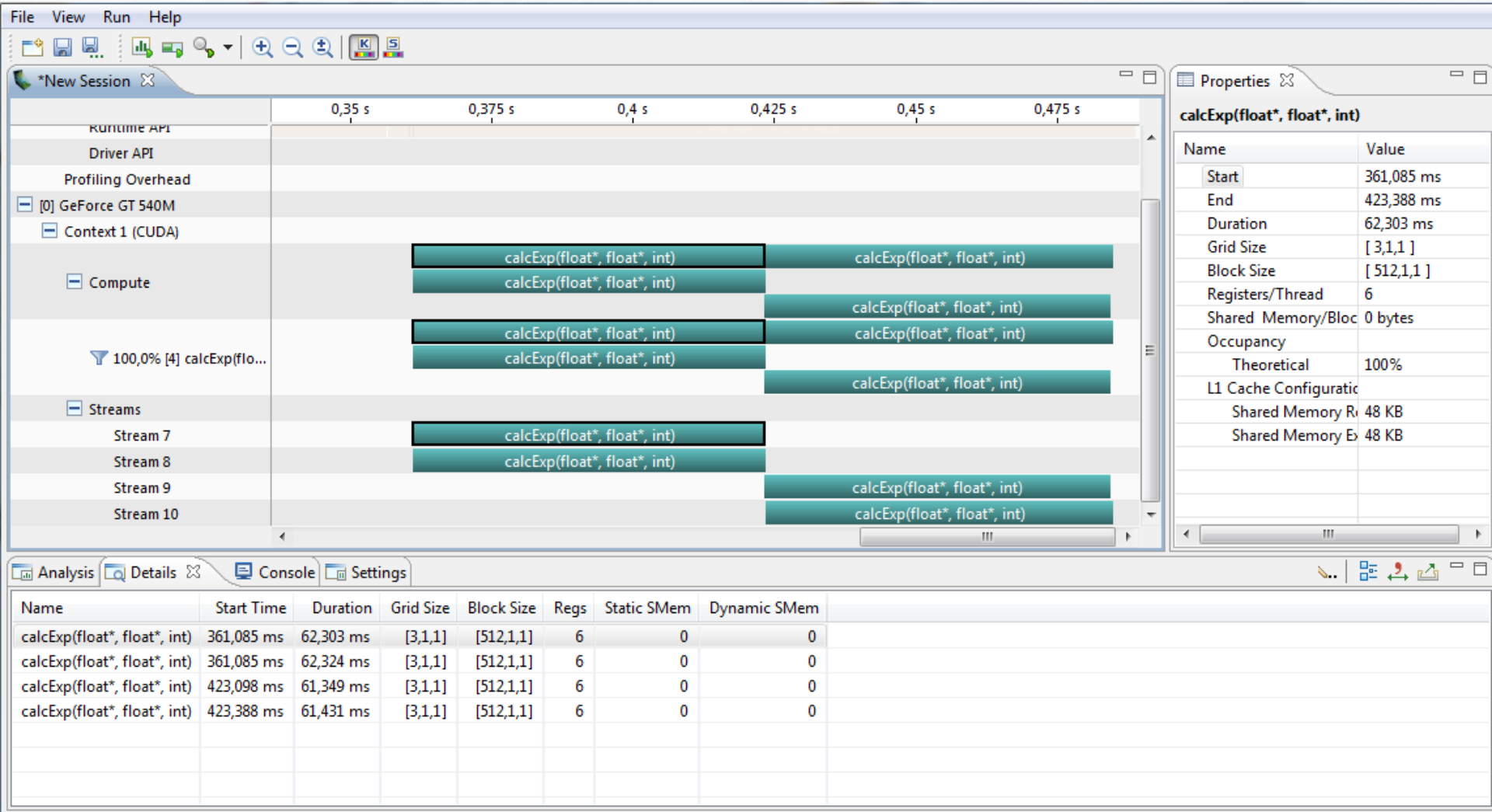
- Ядра запускаются на трех блоках по 1024 нити, устройство состоит из 6 SM



Можем получить ускорение даже при небольших копированиях!

Пример в NVVP

- Небольшие гриды



Дополнительные проблемы

- Устройство не поддерживает параллельное копирование в обе стороны
- В рассматриваемой архитектуре одна аппаратная очередь блоков

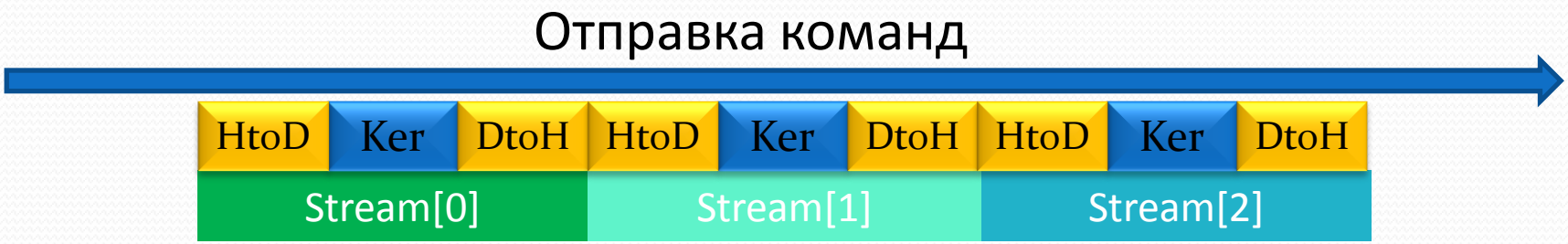
Важен порядок отправки команд!

Нет параллельных копирований

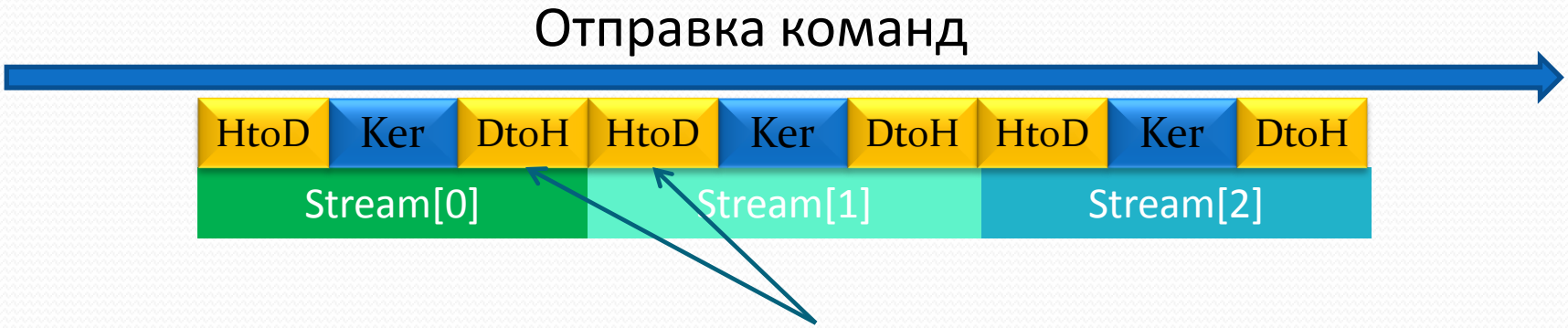
- Отправка команды копирования блокирует старт выполнения всех копирований в другую сторону, отправляемых после неё в любые потоки
- Ускорение только за счет параллельного копирования и выполнения ядер

Модельный пример

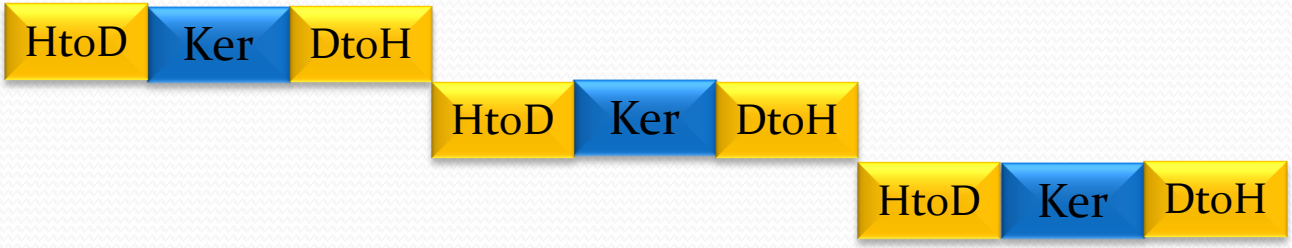
```
for (int i = 0; i < 3; ++i) {
    cudaMemcpyAsync(inputDevPtr + i * size, hostPtr + i *
    size, size, cudaMemcpyHostToDevice, stream[i]);
    MyKernel <<<100, 512, 0, stream[i]>>> (outputDevPtr
        + i * size, inputDevPtr + i * size, size);
    cudaMemcpyAsync(hostPtr + i * size, outputDevPtr + i
     * size, size, cudaMemcpyDeviceToHost, stream[i]);
}
```



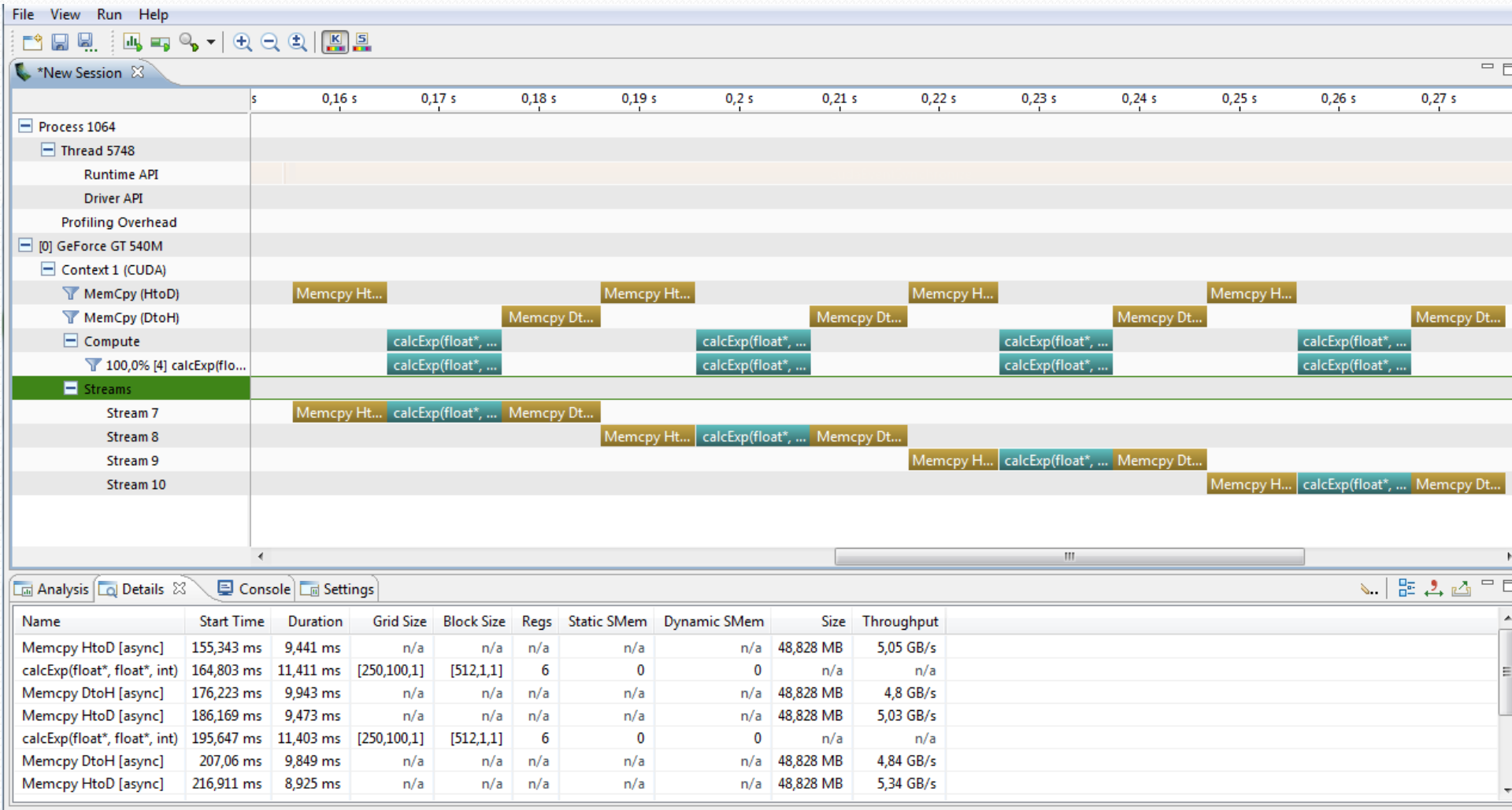
Нет параллельных копирований



Не будут выполняться параллельно

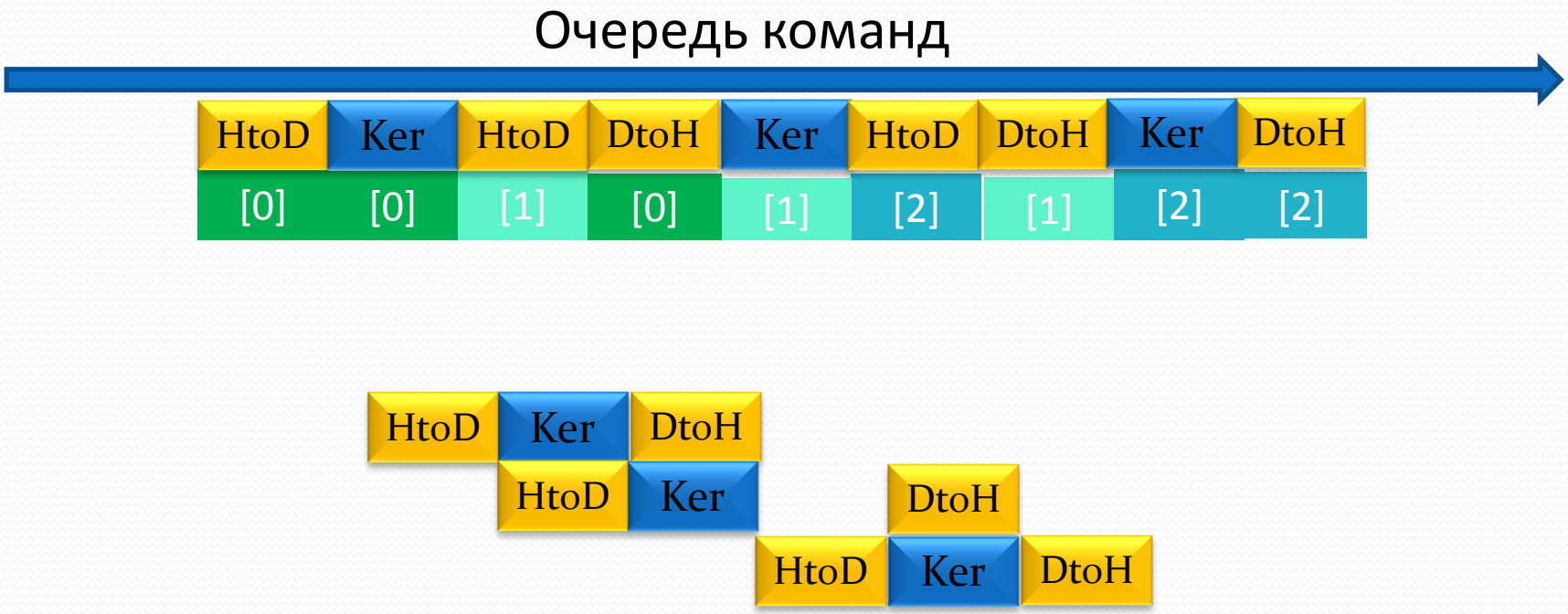


Пример в NVVP

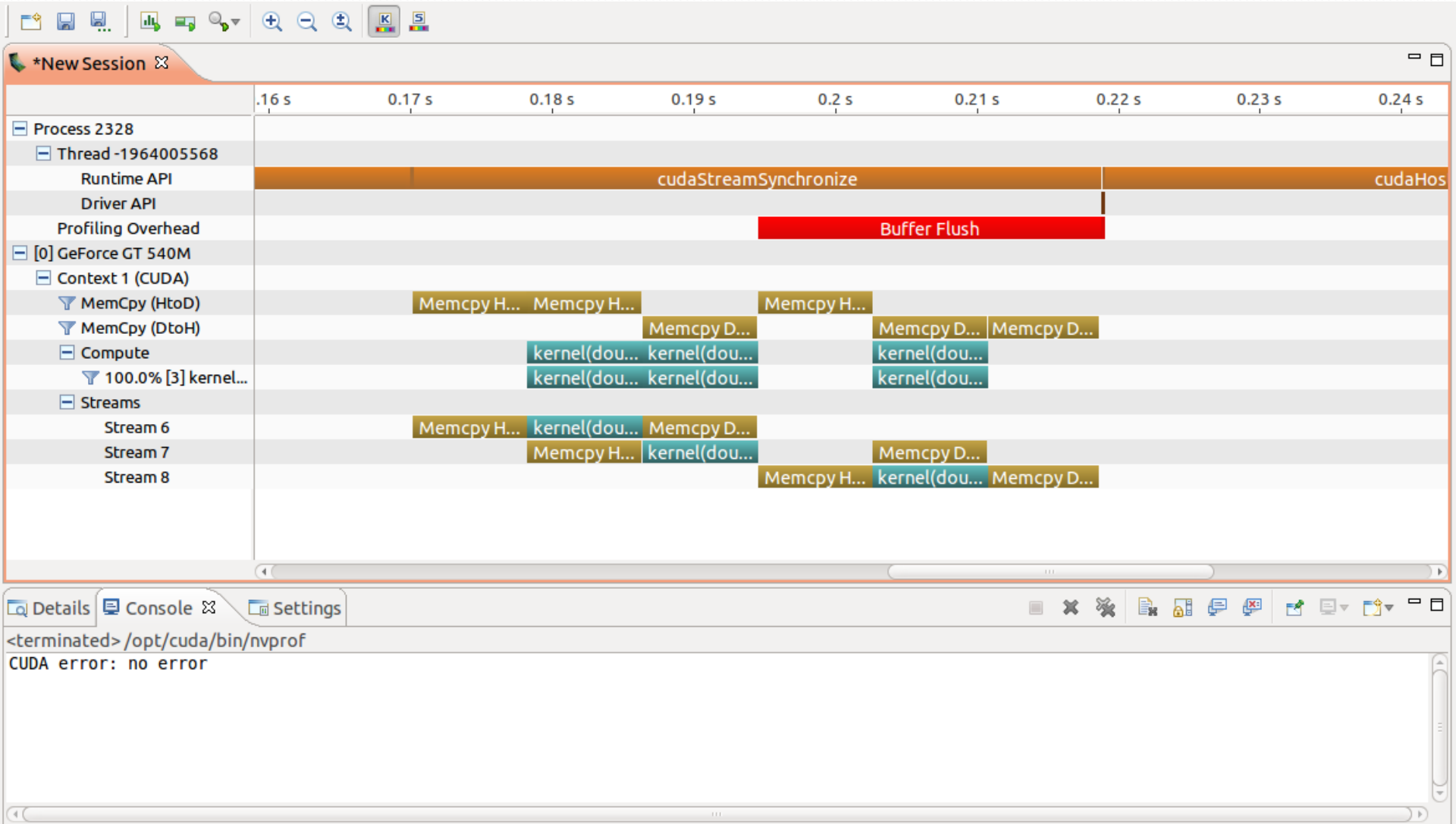


Name	Start Time	Duration	Grid Size	Block Size	Regs	Static SMem	Dynamic SMem	Size	Throughput
Memcpy HtoD [async]	155,343 ms	9,441 ms	n/a	n/a	n/a	n/a	n/a	48,828 MB	5,05 GB/s
calcExp(float*, float*, int)	164,803 ms	11,411 ms	[250,100,1]	[512,1,1]	6	0	0	n/a	n/a
Memcpy DtoH [async]	176,223 ms	9,943 ms	n/a	n/a	n/a	n/a	n/a	48,828 MB	4,8 GB/s
Memcpy HtoD [async]	186,169 ms	9,473 ms	n/a	n/a	n/a	n/a	n/a	48,828 MB	5,03 GB/s
calcExp(float*, float*, int)	195,647 ms	11,403 ms	[250,100,1]	[512,1,1]	6	0	0	n/a	n/a
Memcpy DtoH [async]	207,06 ms	9,849 ms	n/a	n/a	n/a	n/a	n/a	48,828 MB	4,84 GB/s
Memcpy HtoD [async]	216,911 ms	8,925 ms	n/a	n/a	n/a	n/a	n/a	48,828 MB	5,34 GB/s

Если поменять порядок запусков



Пример в NVVP



Единственная аппаратная очередь

- Если в поток отправлен запуск ядра, то все последующие запуски команд в тот же поток должны начать выполняться только после его полного завершения

- В устройствах с Compute Capability <= 3.0 одна аппаратная очередь блоков, в которую попадают блоки всех запусков ядер во всех потоках

  - Можно быть уверенным, что какой-либо запуск ядра полностью отработал, только когда в очереди больше нет блоков!

Единственная аппаратная очередь

Если в поток отправлен запуск ядра, то все последующие запуски команд в тот же поток должны начать выполняться только после его полного завершения

+

Можно быть уверенным, что какой-либо запуск ядра полностью отработал, только когда когда в очереди больше не блоков

=

Неявная синхронизация перед стартом выполнения зависимой от запуска ядра команды:

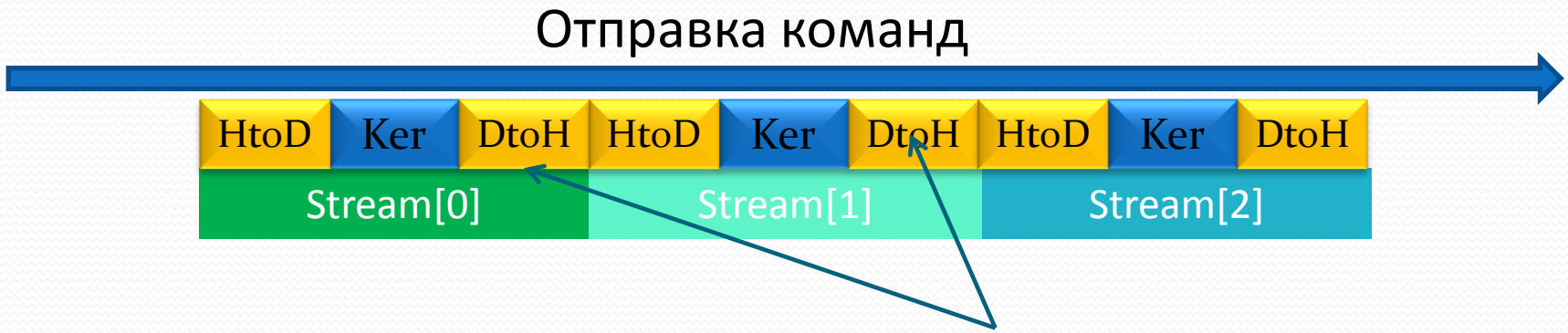
- Начало выполнения команды откладывается до момента, когда в очереди не останется блоков
- Добавление новых блоков в очередь приостанавливается, пока команда не начнет выполняться

Единственная аппаратная очередь

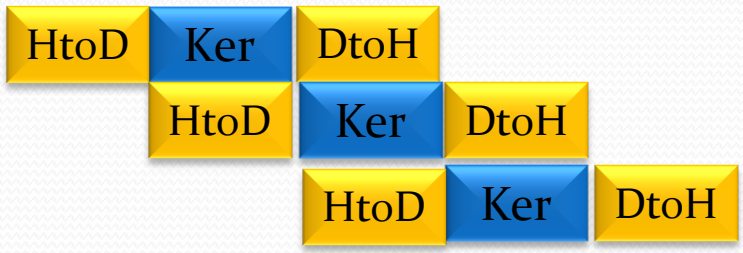
- Начало выполнения зависимой от запуска ядра команды откладывается до момента, когда в очереди не останется блоков
  - Зависимая от запуска ядра команда может параллельно выполняться только с последней волной запуска ядра в другом потоке

- Добавление новых блоков в очередь приостанавливается, пока зависимая от запуска ядра команда не начнет выполняться
  - Запуски всех ядер во всех потоках приостанавливаются до момента, когда полностью отработает запуск-зависимость.

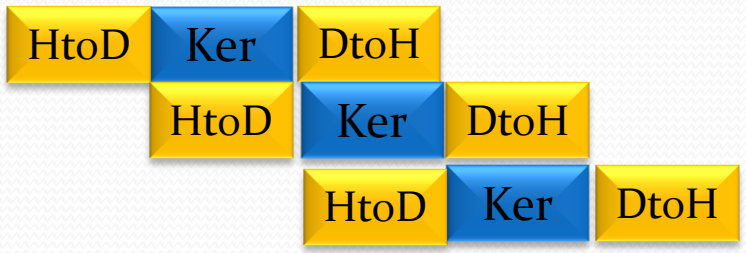
Единственная аппаратная очередь



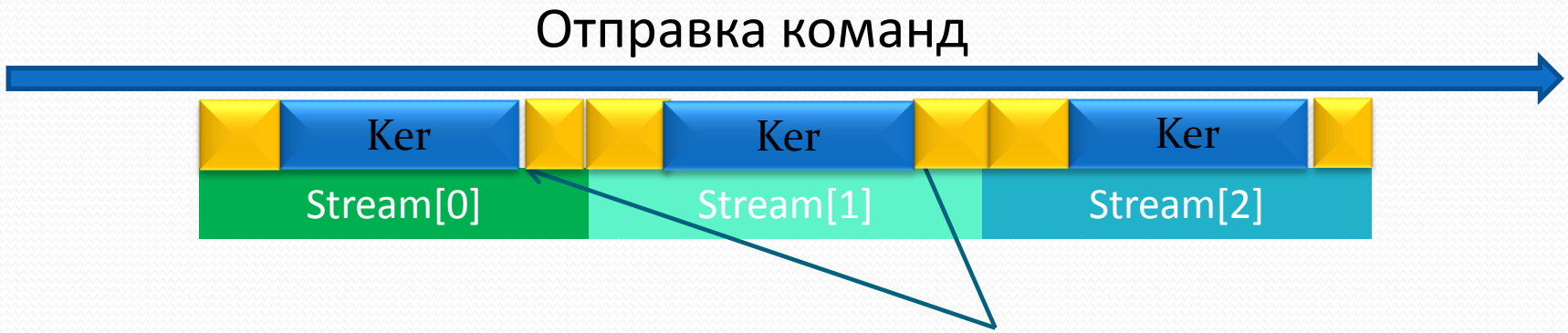
Ожидание



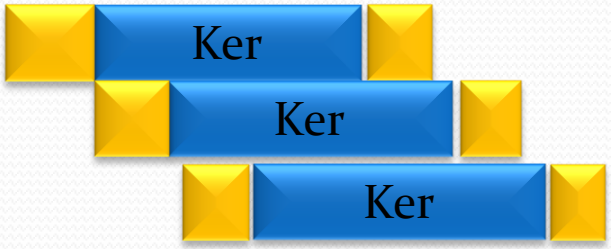
Реальность



Единственная аппаратная очередь

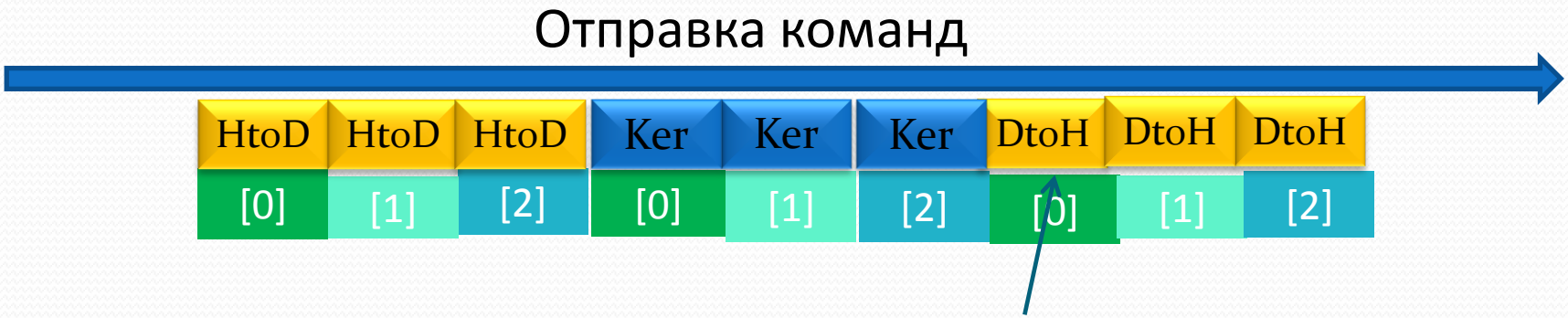


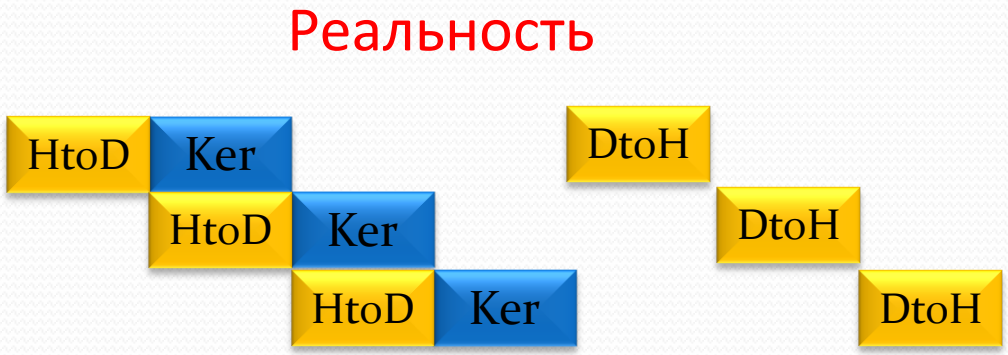
Ожидание



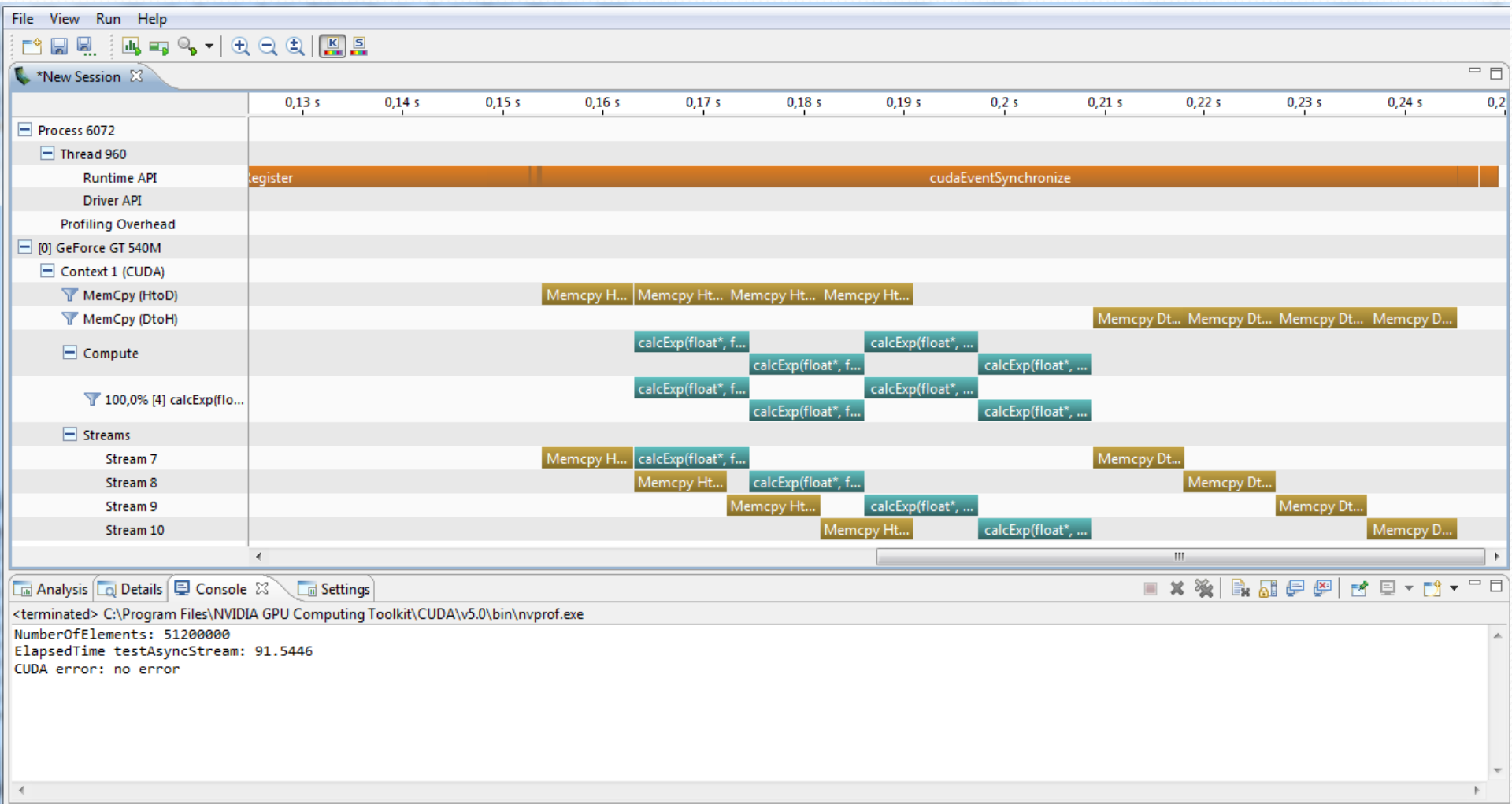
Реальность

Единственная аппаратная очередь





Пример в NVVP



Вывод

- Мало копирований с запусками ядер на больших гридах – не стоит пытаться ускорить программу за счет использования потоков

- Аккуратно отправлять команды на GPU

The end