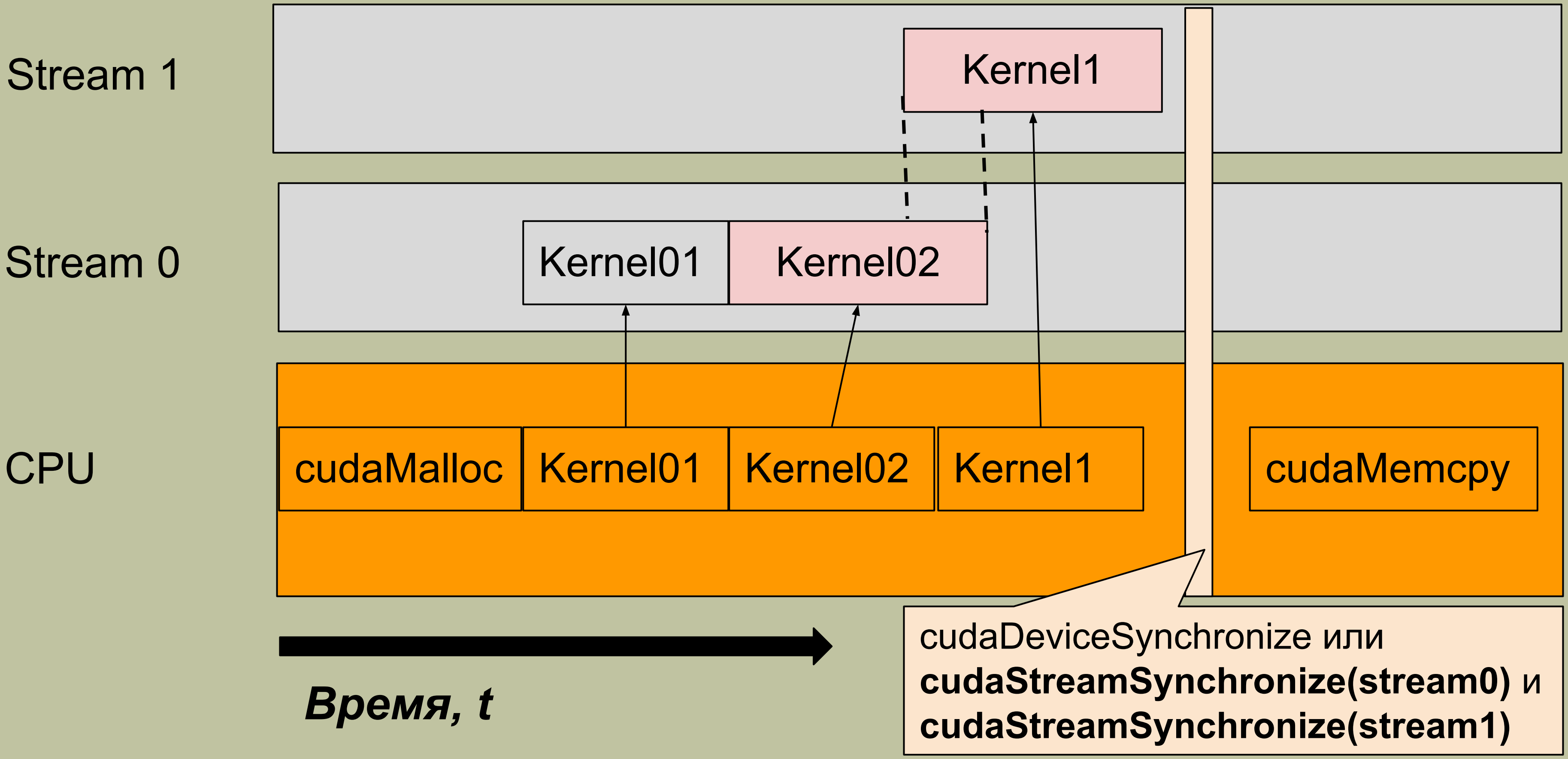
Лекция 12

Параллелизм по задачам.

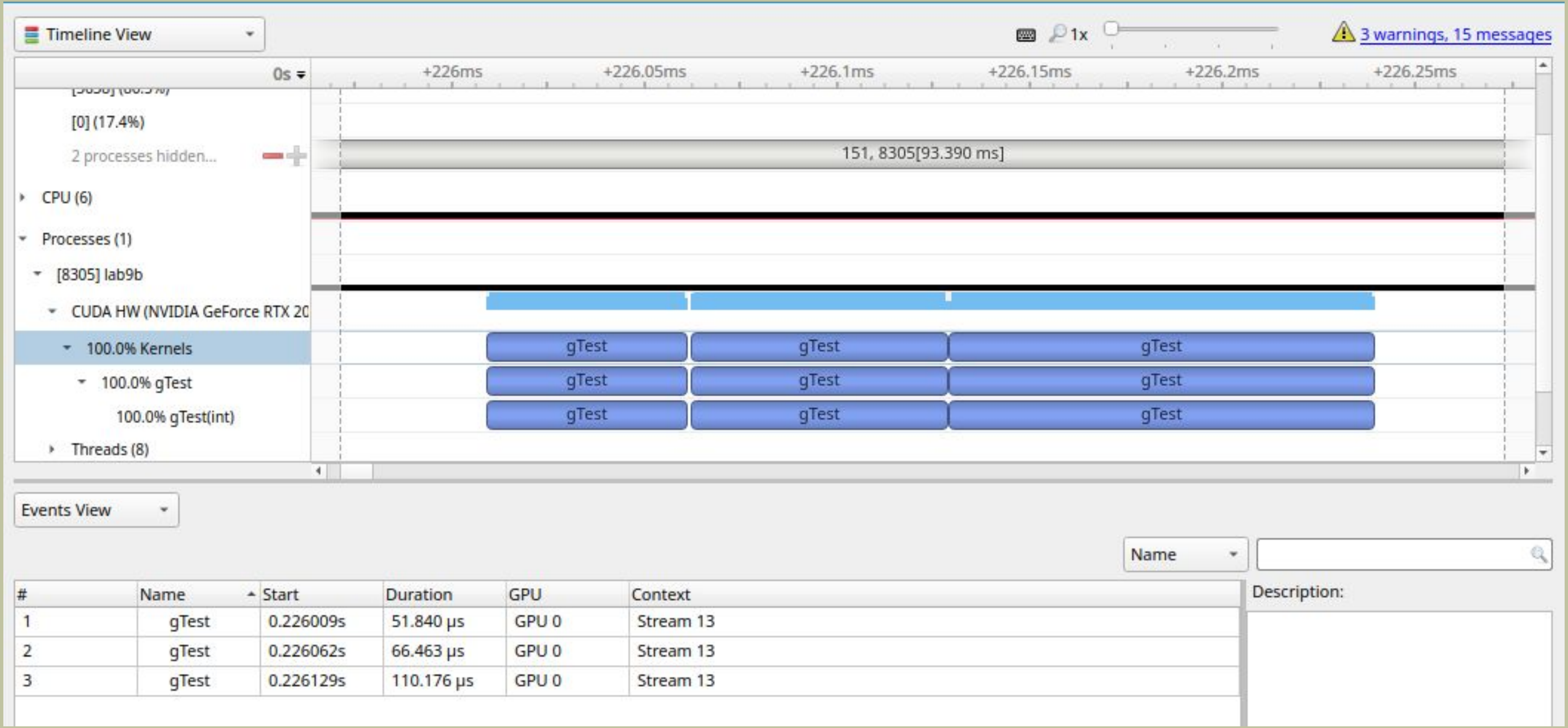
- Потоки *CUDA (CUDA Stream)*.
- Одновременное выполнение ядер.
- Одновременное копирование и выполнение ядра.
- Использование нескольких GPU.

Потоки CUDA (*CUDA Streams*)



cudaDeviceSynchronize или
cudaStreamSynchronize(stream0) и
cudaStreamSynchronize(stream1)

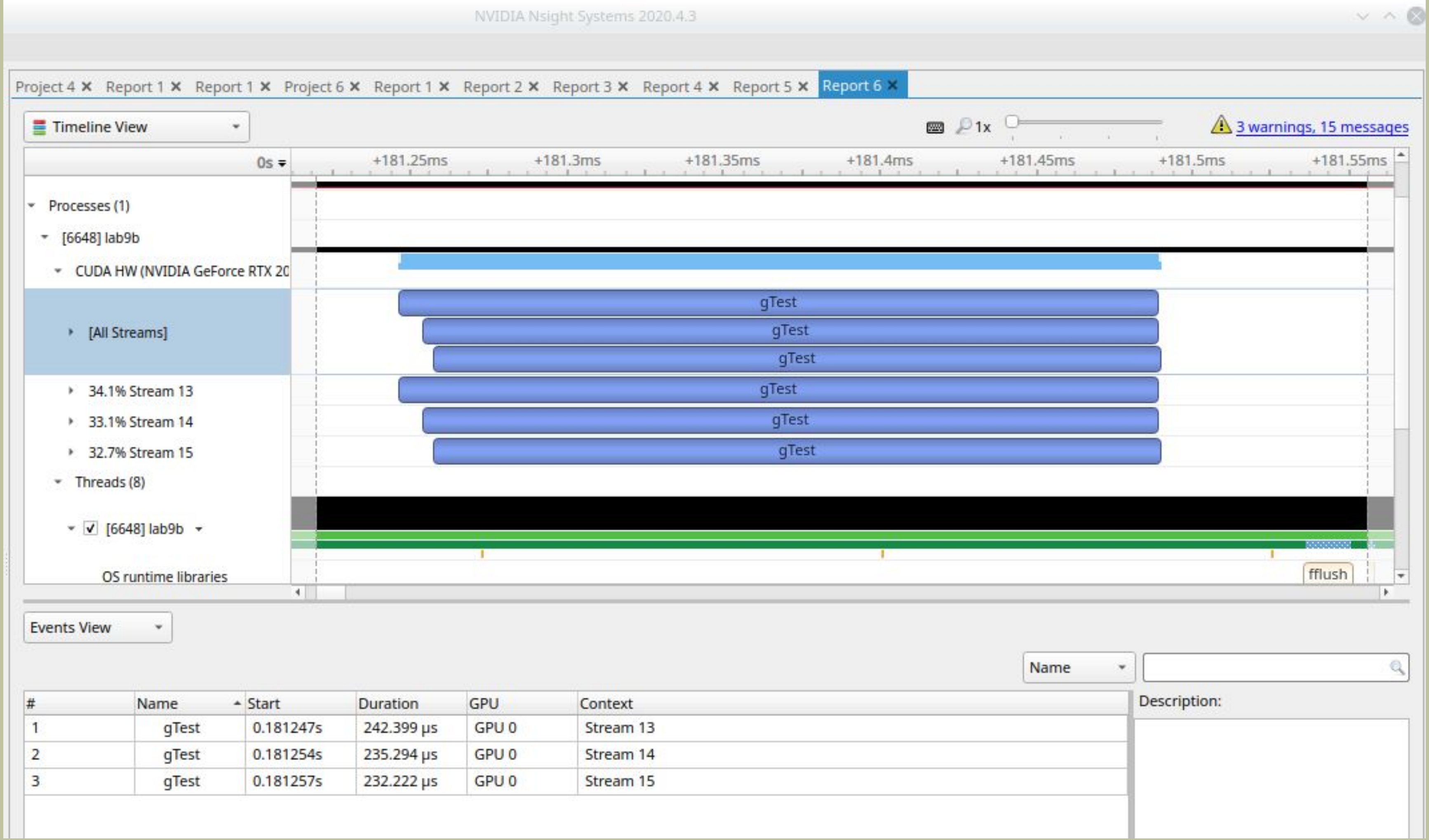
```
#include <stdio.h>
#include <malloc.h>

__global__ void gTest(int step){
  int n=threadIdx.x + blockIdx.x*blockDim.x;
  printf("kernel, thIdx: %d\t%d\n", step, n);
}
int main(){
  int NS=3;
  for (int i = 0; i < NS; i++)
    gTest<<< i+1, 32>>>(i);
  cudaDeviceSynchronize();

  return 0;
}
```



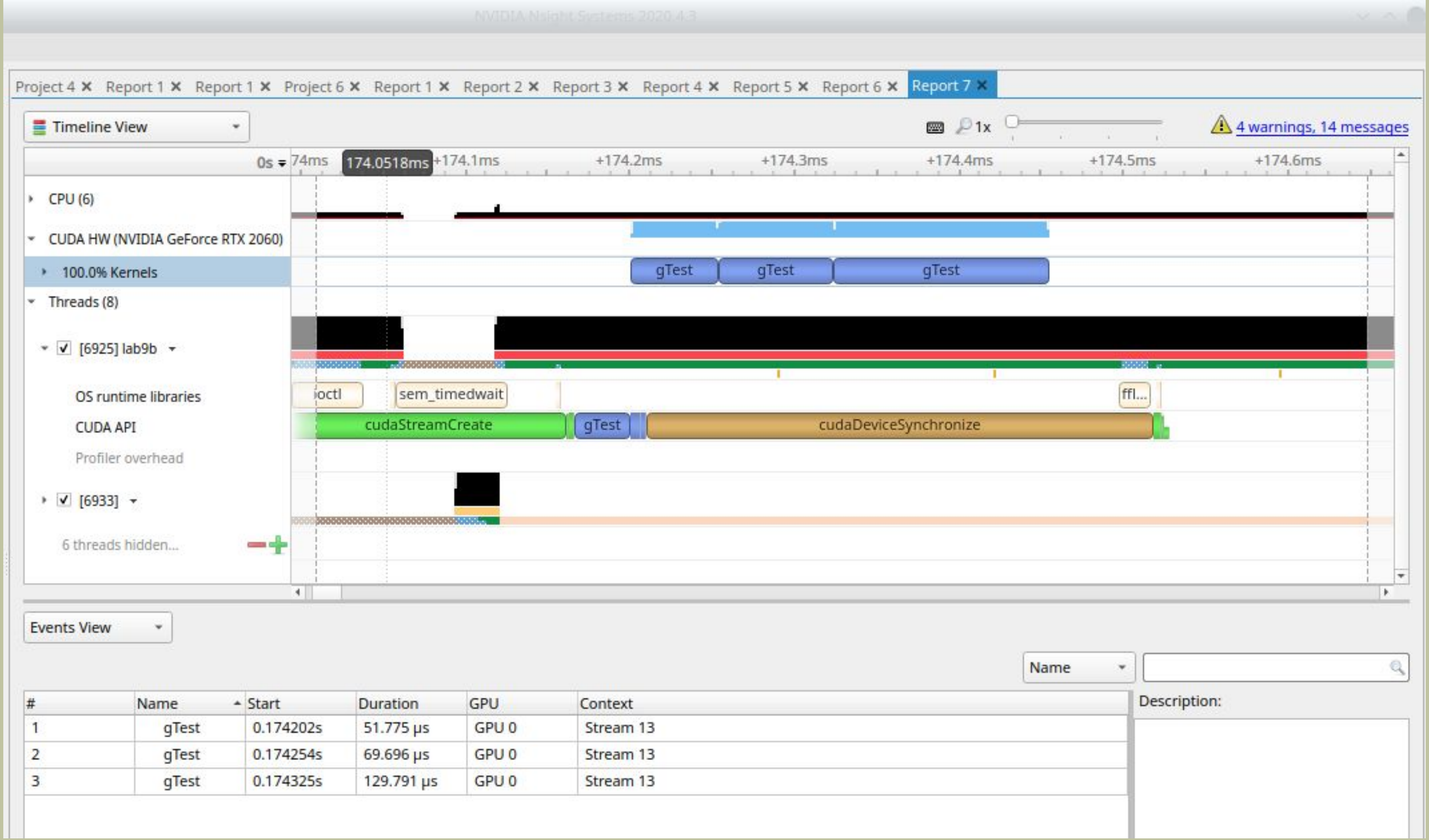
#	Name	Start	Duration	GPU	Context
1	gTest	0.226009s	51.840 μs	GPU 0	Stream 13
2	gTest	0.226062s	66.463 μs	GPU 0	Stream 13
3	gTest	0.226129s	110.176 μs	GPU 0	Stream 13

```
__global__ void gTest(int step){
  int n=threadIdx.x + blockIdx.x*blockDim.x;
  printf("kernel, thIdx: %d\t%d\n", step, n);
}

int main(){
  int NS = 3;
  cudaStream_t *streams;

  streams = (cudaStream_t*)calloc(NS, sizeof(cudaStream_t));

  for (int i = 0; i < NS; i++)
    cudaStreamCreate(&streams[i]);
```

```
 for (int i = 0; i < NS; i++)
    gTest<<< i+1, 32, 0, streams[i] >>>(i);

  cudaDeviceSynchronize();

  for (int i = 0; i < num_streams; i++)
    cudaStreamDestroy(streams[i]);

  free(streams);

  return 0;
}
```



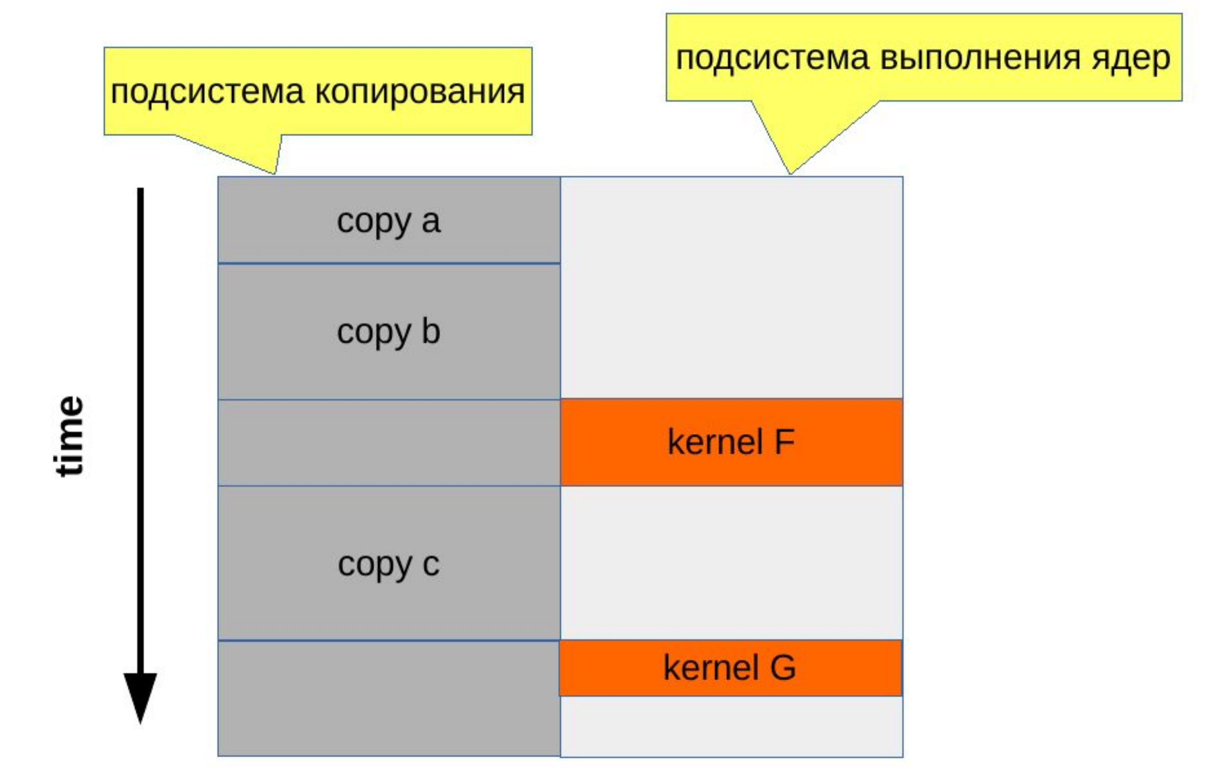
#	Name	Start	Duration	GPU	Context
1	gTest	0.181247s	242.399 μs	GPU 0	Stream 13
2	gTest	0.181254s	235.294 μs	GPU 0	Stream 14
3	gTest	0.181257s	232.222 μs	GPU 0	Stream 15



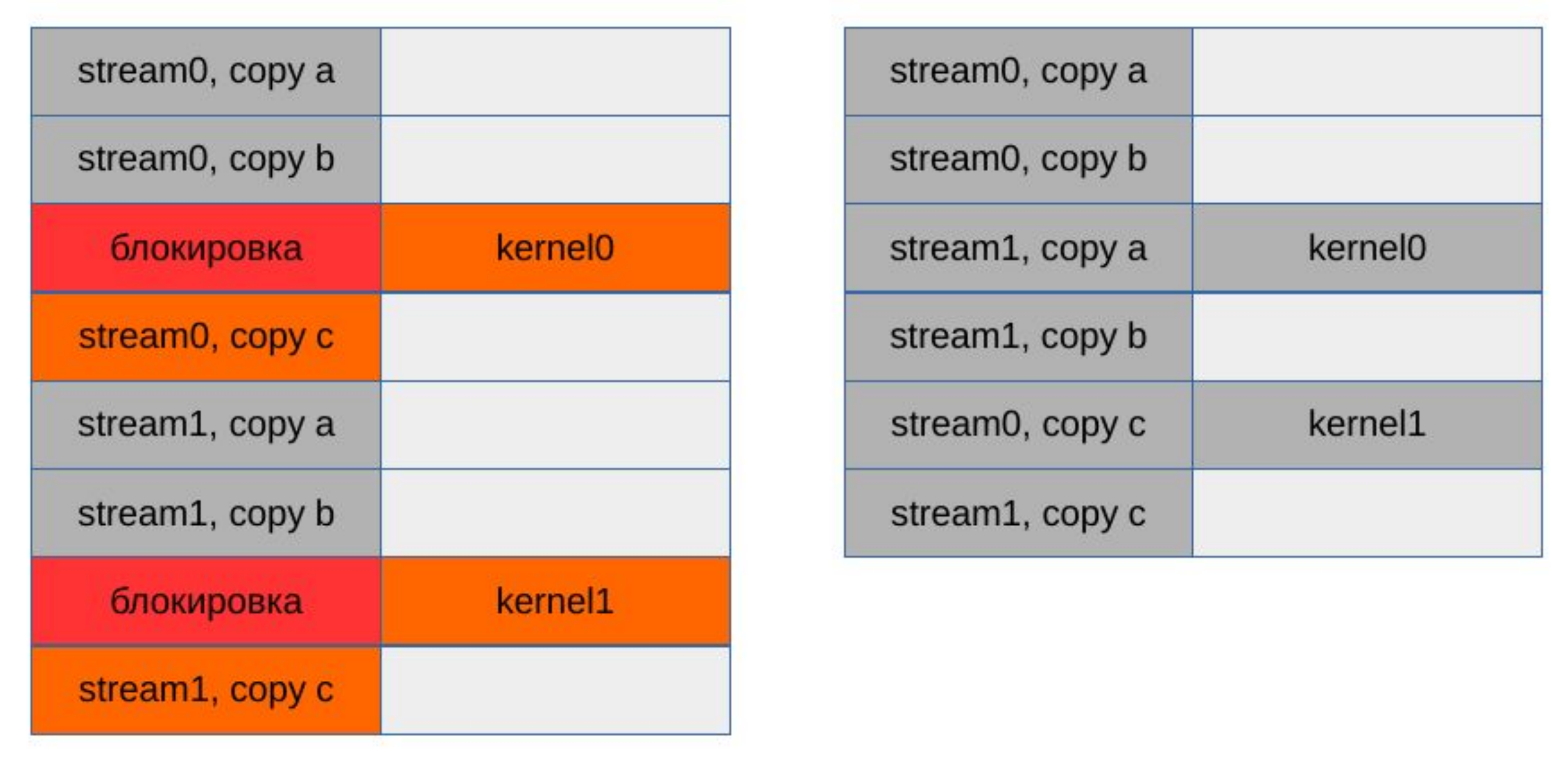
#	Name	Start	Duration	GPU	Context
1	gTest	0.174202s	51.775 μs	GPU 0	Stream 13
2	gTest	0.174254s	69.696 μs	GPU 0	Stream 13
3	gTest	0.174325s	129.791 μs	GPU 0	Stream 13

```
Device 0: "NVIDIA GeForce RTX 2060"
  CUDA Driver Version / Runtime Version          12.0 / 11.1
  CUDA Capability Major/Minor version number:    7.5
……………………………………………………………………………….
```

Concurrent copy and kernel execution: Yes with 3 copy engine(s)



Потоки CUDA и разрешение зависимостей при распараллеливании копирования и выполнения



Очередь копир.	*Очередь выполн.*
stream0, copy a	
stream0, copy b	
блокировка	kernel0
stream0, copy c	
stream1, copy a	
stream1, copy b	
блокировка	kernel1
stream1, copy c	

Очередь копир.	*Очередь выполн.*
stream0, copy a	
stream0, copy b	
stream1, copy a	kernel0
stream1, copy b	
stream0, copy c	kernel1
stream1, copy c	

```
#define N (1024*1024)
#define FULL_DATA_SIZE  (N*20)

__global__ void kernel(int* a, int* b, int* c){
  int idx=threadIdx.x+blockIdx.x*blockDim.x;
  if(idx<N){
    int idx1=(idx+1)%256;
    int idx2=(idx+2)%256;
    float as=(a[idx]+a[idx1]+a[idx2])/3.0f;
    float bs=(b[idx]+b[idx1]+b[idx2])/3.0f;
    c[idx]=(as+bs)/2;
  }
}
```

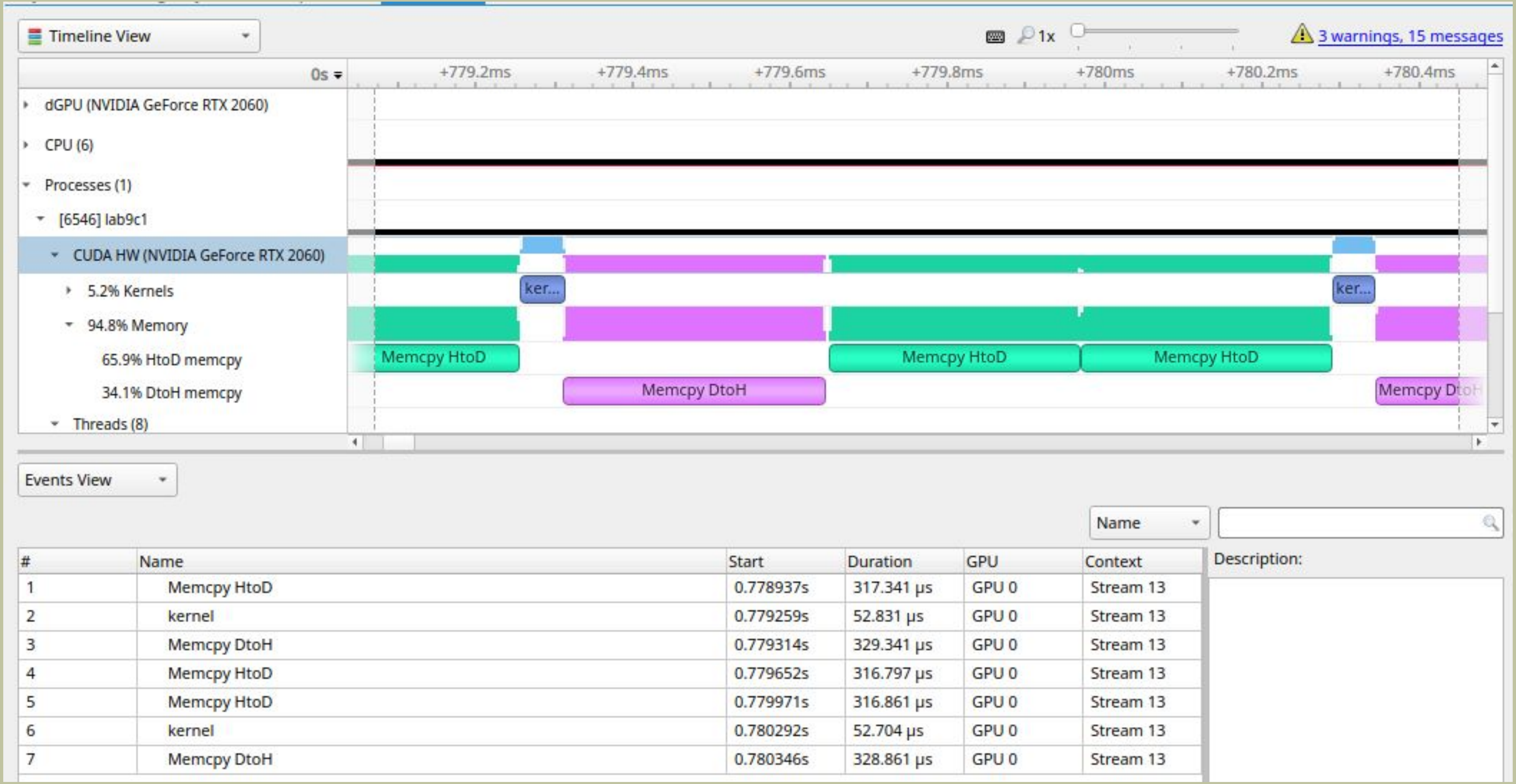
```
int main(){
  cudaDeviceProp prop;
  int whichDevice;

  cudaGetDevice(&whichDevice);

  cudaGetDeviceProperties(&prop, whichDevice);
  if(!prop.deviceOverlap){
    printf("Device does not support overlapping\n");
    return 0;
  }
```

```
int *host_a, *host_b, *host_c;
int *dev_a, *dev_b, *dev_c;

cudaMalloc( (void**)&dev_a, N*sizeof(int)) ;
cudaMalloc( (void**)&dev_b, N*sizeof(int)) ;
cudaMalloc( (void**)&dev_c, N*sizeof(int)) ;


cudaMallocHost( (void**)&host_a, FULL_DATA_SIZE*sizeof(int)) ;
cudaMallocHost( (void**)&host_b, FULL_DATA_SIZE*sizeof(int)) ;
cudaMallocHost( (void**)&host_c, FULL_DATA_SIZE*sizeof(int)) ;
```

Выделение прикрепленной памяти (pinned memory) на хосте.

```
for(int i=0; i<FULL_DATA_SIZE;i++){
  host_a[i]=rand();
  host_b[i]=rand();
}
cudaStream_t stream;
cudaStreamCreate(&stream);
for(int i=0; i<FULL_DATA_SIZE; i+=N){
    cudaMemcpyAsync(dev_a, host_a+i, N*sizeof(int),
                 cudaMemcpyHostToDevice, stream);
    cudaMemcpyAsync(dev_b, host_b+i, N*sizeof(int),
                 cudaMemcpyHostToDevice, stream);
    kernel<<<N/256, 256, 0, stream>>>(dev_a, dev_b, dev_c);
    cudaMemcpyAsync(host_c+i, dev_c, N*sizeof(int),
                 cudaMemcpyDeviceToHost, stream);
}
cudaStreamSynchronize( stream );
```

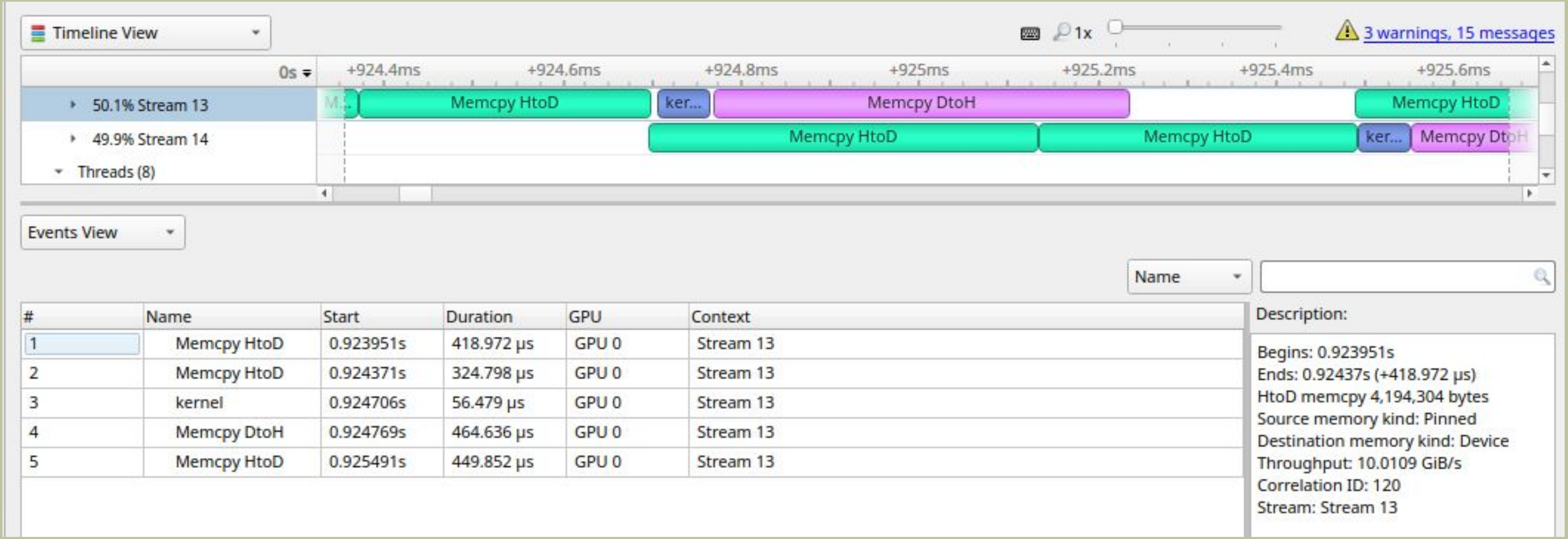
```
  cudaFreeHost(host_a);
  cudaFreeHost(host_b);
  cudaFreeHost(host_c);

  cudaFree(dev_a);
  cudaFree(dev_b);
  cudaFree(dev_c);

  cudaStreamDestroy(stream);

  return 0;
}
```



#	Name	Start	Duration	GPU	Context
1	Memcpy HtoD	0.778937s	317.341 µs	GPU 0	Stream 13
2	kernel	0.779259s	52.831 µs	GPU 0	Stream 13
3	Memcpy DtoH	0.779314s	329.341 µs	GPU 0	Stream 13
4	Memcpy HtoD	0.779652s	316.797 µs	GPU 0	Stream 13
5	Memcpy HtoD	0.779971s	316.861 µs	GPU 0	Stream 13
6	kernel	0.780292s	52.704 µs	GPU 0	Stream 13
7	Memcpy DtoH	0.780346s	328.861 µs	GPU 0	Stream 13

```
cudaStream_t stream0, stream1;
  cudaStreamCreate(&stream0);
  cudaStreamCreate(&stream1);
  for(int i=0; i<FULL_DATA_SIZE;...
  cudaStreamSynchronize( stream0) ;
  cudaStreamSynchronize( stream1);
…………………………………………
  cudaStreamDestroy(stream0);
  cudaStreamDestroy(stream1);
…………………………………………
```

```
for(int i=0; i<FULL_DATA_SIZE; i+=N*2){
    cudaMemcpyAsync(dev_a0, host_a+i, N*sizeof(int),
            cudaMemcpyHostToDevice, stream0);
    cudaMemcpyAsync(dev_a1, host_a+i+N, N*sizeof(int),
            cudaMemcpyHostToDevice, stream1);
    cudaMemcpyAsync(dev_b0, host_b+i, N*sizeof(int),
            cudaMemcpyHostToDevice, stream0);
    cudaMemcpyAsync(dev_b1, host_b+i+N, N*sizeof(int),
            cudaMemcpyHostToDevice, stream1);
    kernel<<<N/256, 256, 0, stream0>>>(dev_a0, dev_b0, dev_c0);
    kernel<<<N/256, 256, 0, stream1>>>(dev_a1, dev_b1, dev_c1);
    cudaMemcpyAsync(host_c+i, dev_c0, N*sizeof(int),
            cudaMemcpyDeviceToHost, stream0);
    cudaMemcpyAsync(host_c+i+N, dev_c1, N*sizeof(int),
            cudaMemcpyDeviceToHost, stream1);
}
```



#	Name	Start	Duration	GPU	Context
1	Memcpy HtoD	0.923951s	418.972 μs	GPU 0	Stream 13
2	Memcpy HtoD	0.924371s	324.798 μs	GPU 0	Stream 13
3	kernel	0.924706s	56.479 μs	GPU 0	Stream 13
4	Memcpy DtoH	0.924769s	464.636 μs	GPU 0	Stream 13
5	Memcpy HtoD	0.925491s	449.852 μs	GPU 0	Stream 13

Description:

Begins: 0.923951s
Ends: 0.92437s (+418.972 μs)
HtoD memcpy 4,194,304 bytes
Source memory kind: Pinned
Destination memory kind: Device
Throughput: 10.0109 GiB/s
Correlation ID: 120
Stream: Stream 13

Использование нескольких GPU

```
#include <stdio.h>
#define REAL float

__global__ void initFun(int* nf, int devnum){
  int n=threadIdx.x + blockIdx.x*blockDim.x;
  nf[n]*=10;
}
```

```
int main(int argc, char* argv[]){
  if(argc<4){
  fprintf(stderr, "USAGE: <prog_name>  <size_of_array> <num_of_devices>"
                          "<device_indices>\n");
  return -1;
 }
 int N=atoi(argv[1]);
```

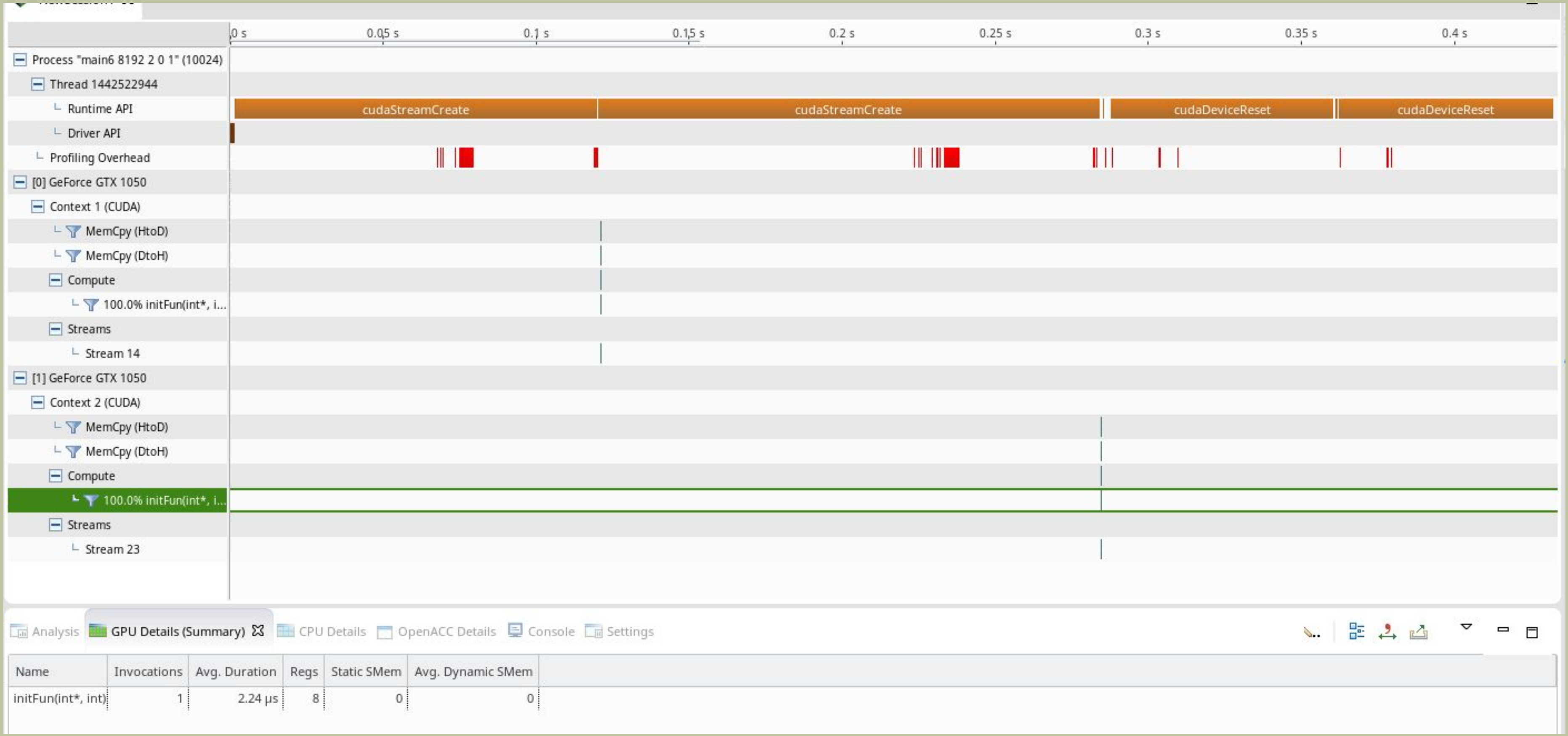
```
int* info_devs=(int*)calloc(argc-2, sizeof(int));
info_devs[0]=atoi(argv[2]);
for(int i=1;i<argc-2;i++){
  info_devs[i]=atoi(argv[i+2]);
}
fprintf(stderr,"num of devices: %d\n",info_devs[0]);
for(int i=1;i<argc-2;i++)
        fprintf(stderr,"i_d=%d\n",info_devs[i]);

int** nfd=(int**)calloc(info_devs[0], sizeof(int*));
int** nfh=(int**)calloc(info_devs[0], sizeof(int*));
```

```
 cudaStream_t* streams;
 streams=(cudaStream_t*)calloc(info_devs[0], sizeof(cudaStream_t));

for(int i=0;i<info_devs[0];i++){
   cudaSetDevice(info_devs[i+1]);
   cudaStreamCreate(&streams[i]);

   cudaMalloc((void**)&nfd[i], (N/info_devs[0])*sizeof(int));
   cudaMallocHost((void**)&nfh[i], (N/info_devs[0])*sizeof(int));

   for(int n=0;n<N/info_devs[0]; n++)
      nfh[i][n]=n+i*N/info_devs[0];
 cudaMemcpyAsync(nfd[i],nfh[i],
                (N/info_devs[0])*sizeof(int),
                cudaMemcpyHostToDevice, streams[i]);

 initFun<<<N/info_devs[0]/32, 32, 0, streams[i]>>>(nfd[i],i);

 cudaMemcpyAsync(nfh[i],nfd[i],
                (N/info_devs[0])*sizeof(int),
                cudaMemcpyDeviceToHost,streams[i]);
}
```

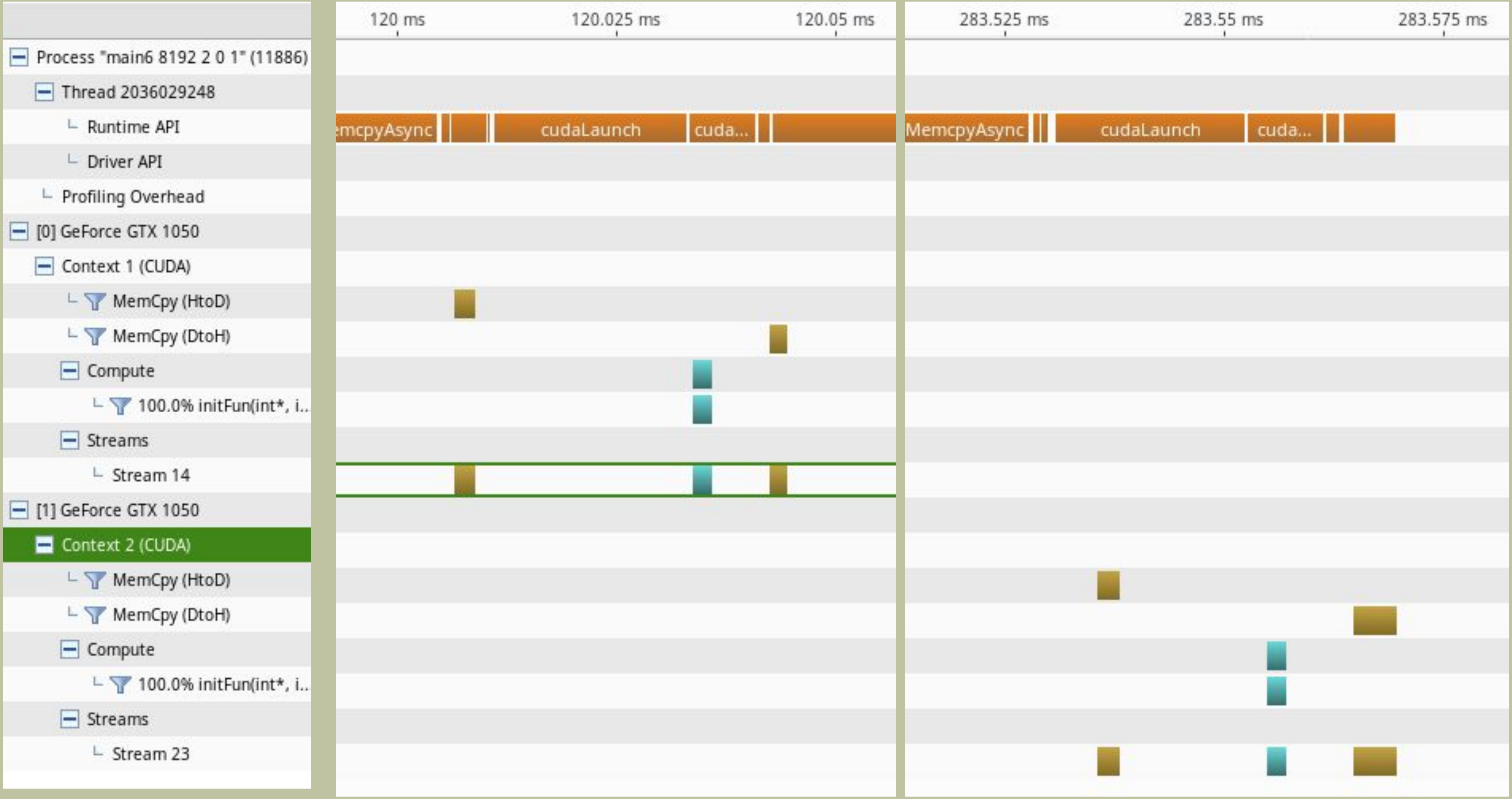
```
 for(int i=0;i<info_devs[0];i++){
    cudaSetDevice(info_devs[i+1]);
    cudaStreamSynchronize(streams[i]);

    for(int n=0;n<N/info_devs[0];n++)
      fprintf(stdout,"nfh[%d][%d]=%d\n",i,n, nfh[i][n]);

    cudaFree(nfd[i]);
    cudaFreeHost(nfh[i]);
    cudaStreamDestroy(streams[i]);
    cudaDeviceReset();
 }

 return 0;
}
```

```
> ./main6 1024 2 0 1 > tmp61
num of devices: 2
i_d=0
i_d=1
> vim tmp61
```

```
nfh[0][0]=0
nfh[0][1]=10
nfh[0][2]=20
nfh[0][3]=30
………………
nfh[0][510]=5100
nfh[0][511]=5110
nfh[1][0]=5120
nfh[1][1]=5130
……………………..
nfh[1][510]=10220
nfh[1][511]=10230
```



Name	Invocations	Avg. Duration	Regs	Static SMem	Avg. Dynamic SMem
initFun(int*, int)	1	2.24 μs	8	0	0


Process "main6 8192 2 0 1" (11886)
Thread 2036029248
Runtime API
Driver API
Profiling Overhead
[0] GeForce GTX 1050
Context 1 (CUDA)
MemCpy (HtoD)
MemCpy (DtoH)
Compute
100.0% initFun(int*, i...
Streams
Stream 14
[1] GeForce GTX 1050
Context 2 (CUDA)
MemCpy (HtoD)
MemCpy (DtoH)
Compute
100.0% initFun(int*, i...
Streams
Stream 23
120 ms
120.025 ms
120.05 ms
283.525 ms
283.55 ms
283.575 ms
emcpyAsync
cudaLaunch
cuda...
MemcpyAsync
cudaLaunch
cuda...