Многопроцессорные системы (продолжение). Графические ускорители.

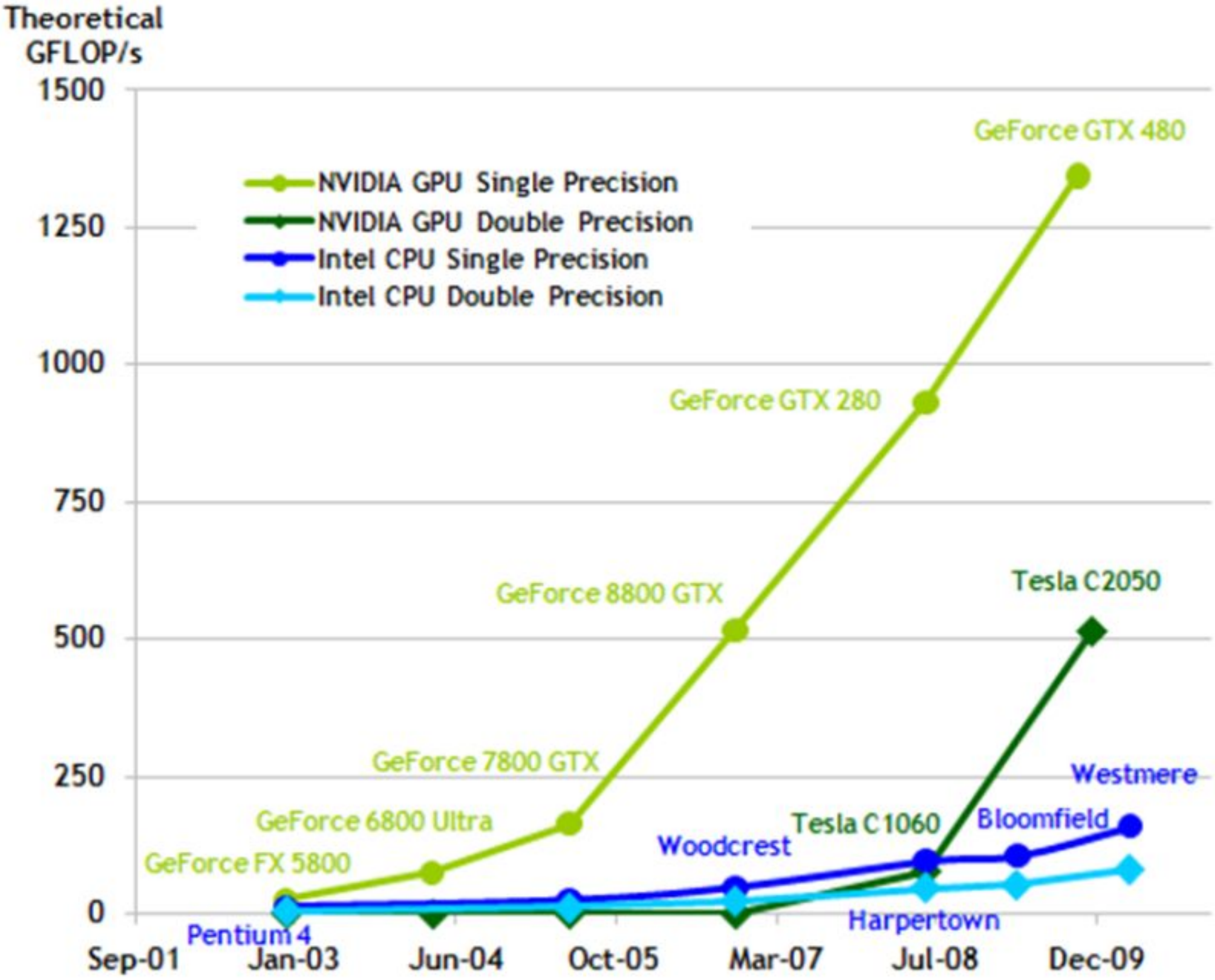
- Использование графических ускорителей для прикладных вычислений (**GPGPU**).
- Физическая организация **GPU**.
- Архитектура **CUDA** (*Compute Unified Device Architecture*).
- Интерфейс программирования **CUDA C**.
- Установка среды разработки и исполнения.

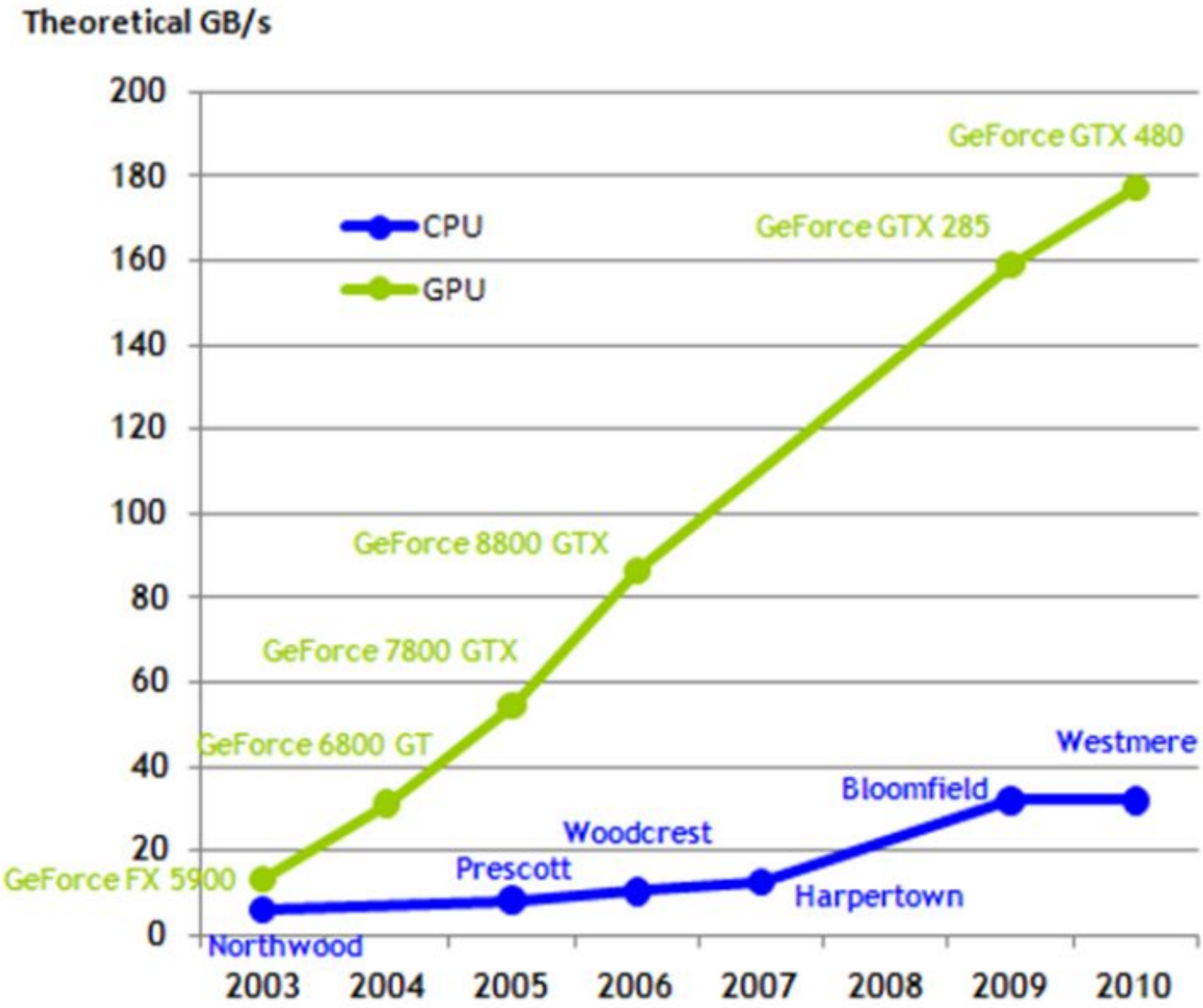
http://www.nvidia.ru/object/cuda_home_new_ru.html

http://developer.nvidia.com/cuda-toolkit-32-downloads

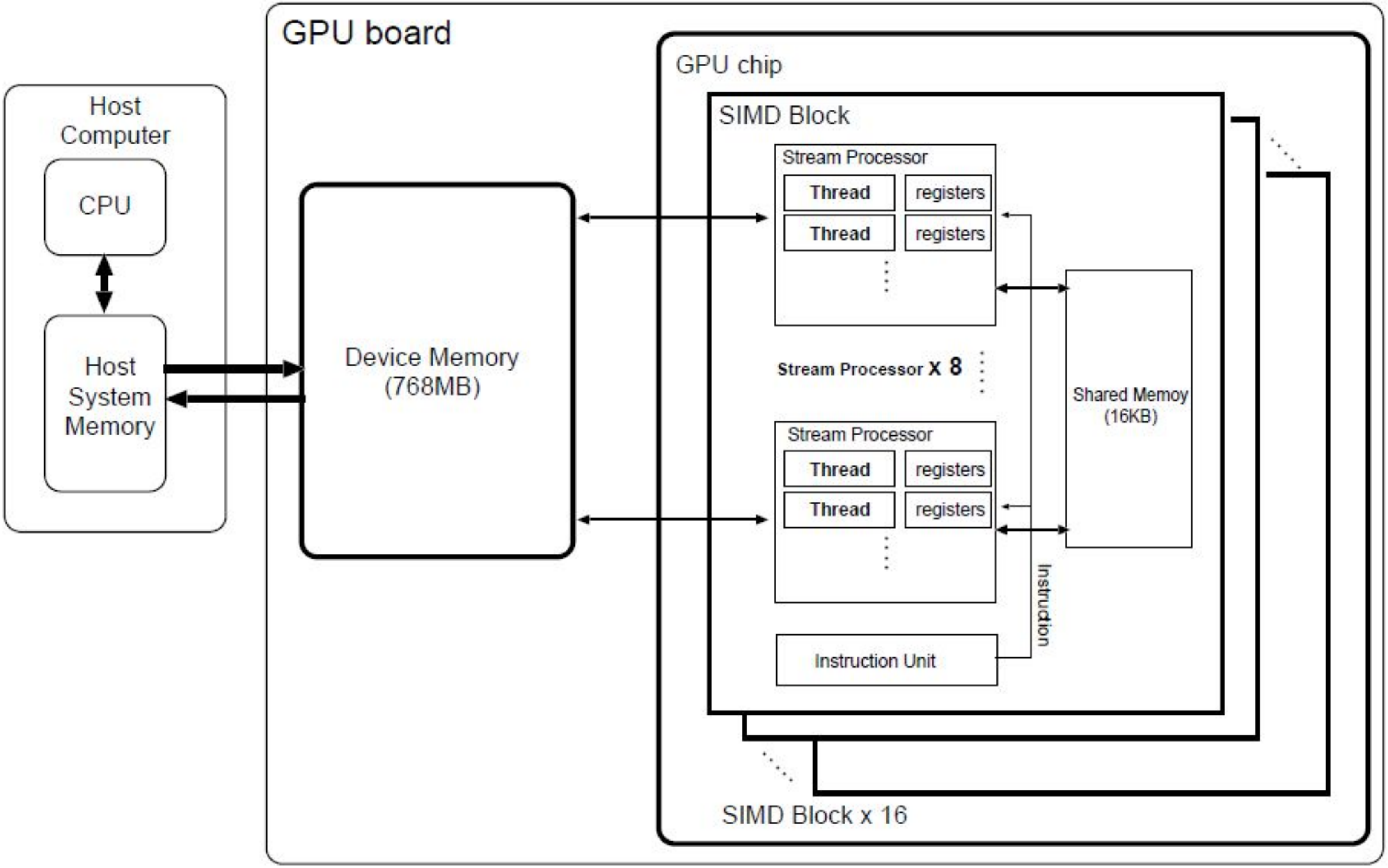
CUDA_C_Programming_Guide.pdf

Боресков А.В., Харламов А.А. Основы работы с технологией CUDA, Москва: ДМК, 2010

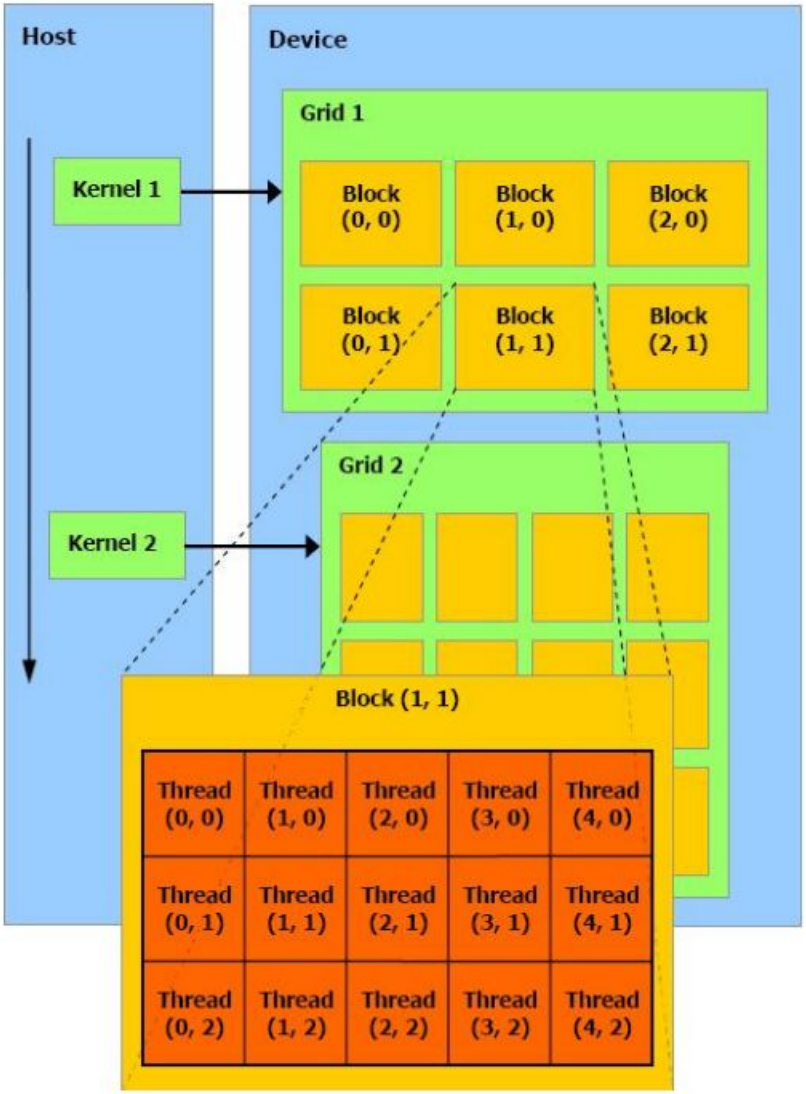

Theoretical GFLOP/s
1500
1250
1000
750
500
250
0
Sep-01
Jan-03
Jun-04
Oct-05
Mar-07
Jul-08
Dec-09
NVIDIA GPU Single Precision
NVIDIA GPU Double Precision
Intel CPU Single Precision
Intel CPU Double Precision
GeForce GTX 480
GeForce GTX 280
GeForce 8800 GTX
GeForce 7800 GTX
GeForce 6800 Ultra
GeForce FX 5800
Tesla C2050
Tesla C1060
Westmere
Bloomfield
Woodcrest
Harpertown
Pentium 4

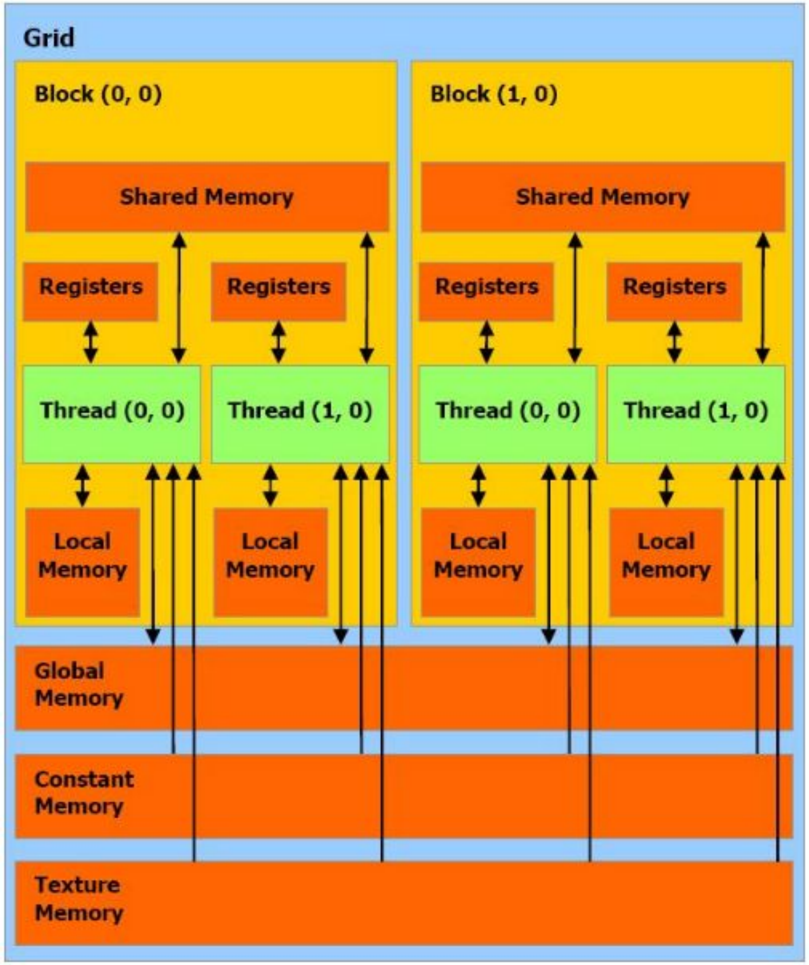

Theoretical GB/s
200
180
160
140
120
100
80
60
40
20
0
CPU
GPU
GeForce GTX 480
GeForce GTX 285
GeForce 8800 GTX
GeForce 7800 GTX
GeForce 6800 GT
GeForce FX 5900
Westmere
Bloomfield
Woodcrest
Prescott
Harpertown
Northwood
2003
2004
2005
2006
2007
2008
2009
2010

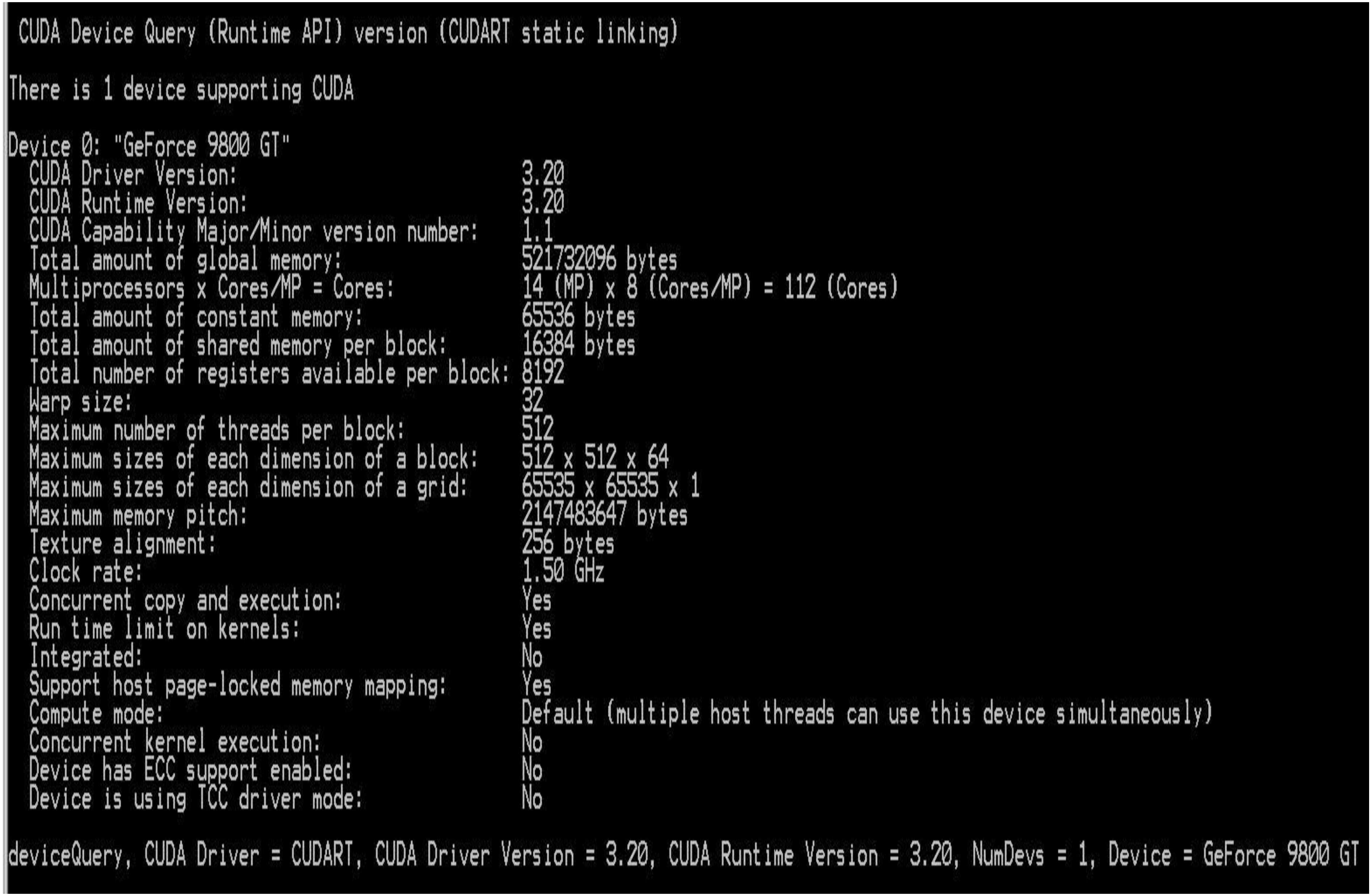
Физическое представление GPU



Логическое представление GPU (**архитектура CUDA**)





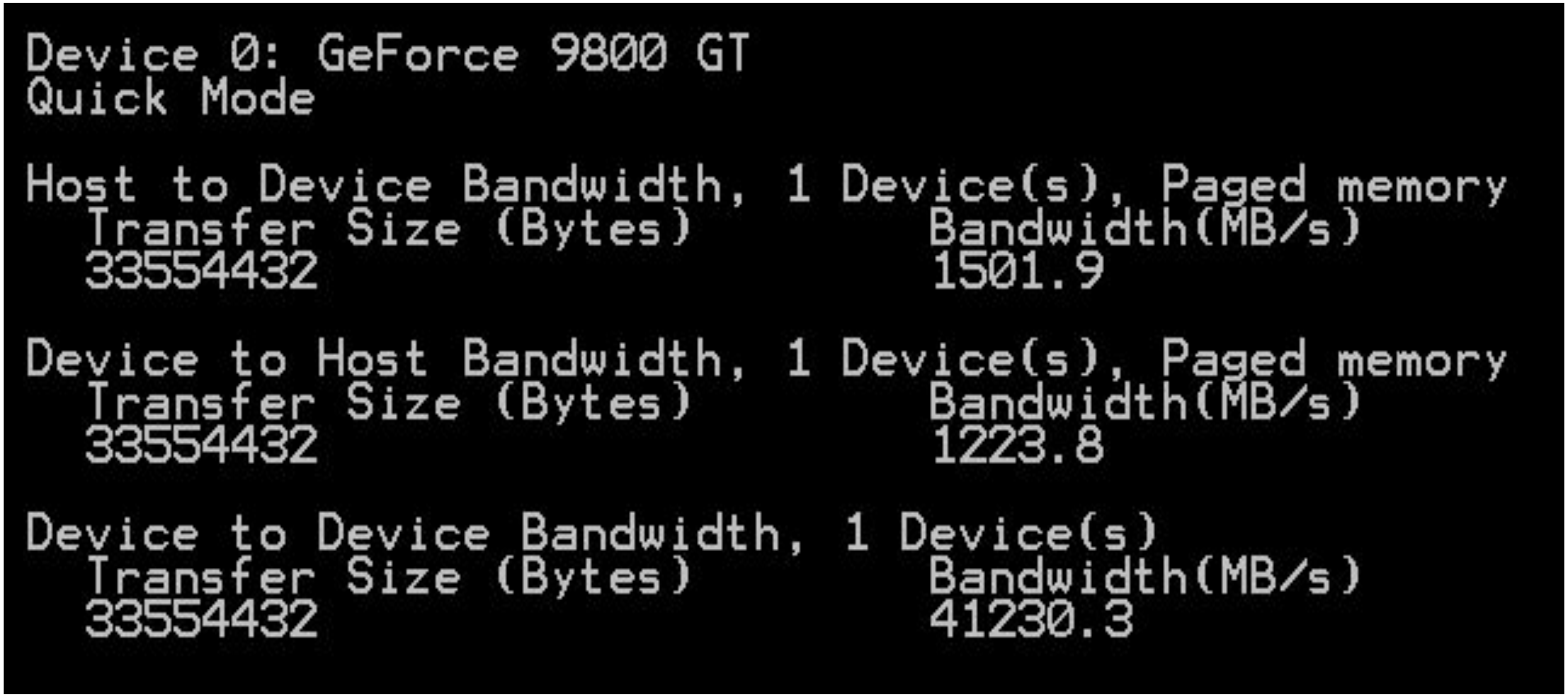


```
 CUDA Device Query (Runtime API) version (CUDART static linking)

There is 1 device supporting CUDA

Device 0: "GeForce 9800 GT"
  CUDA Driver Version:                           3.20
  CUDA Runtime Version:                          3.20
  CUDA Capability Major/Minor version number:    1.1
  Total amount of global memory:                 521732096 bytes
  Multiprocessors x Cores/MP = Cores:            14 (MP) x 8 (Cores/MP) = 112 (Cores)
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       16384 bytes
  Total number of registers available per block: 8192
  Warp size:                                     32
  Maximum number of threads per block:           512
  Maximum sizes of each dimension of a block:    512 x 512 x 64
  Maximum sizes of each dimension of a grid:     65535 x 65535 x 1
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             256 bytes
  Clock rate:                                    1.50 GHz
  Concurrent copy and execution:                 Yes
  Run time limit on kernels:                     Yes
  Integrated:                                    No
  Support host page-locked memory mapping:       Yes
  Compute mode:                                  Default (multiple host threads can use this device simultaneously)
  Concurrent kernel execution:                   No
  Device has ECC support enabled:                No
  Device is using TCC driver mode:               No

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 3.20, CUDA Runtime Version = 3.20, NumDevs = 1, Device = GeForce 9800 GT
```



```
Device 0: GeForce 9800 GT
Quick Mode

Host to Device Bandwidth, 1 Device(s), Paged memory
  Transfer Size (Bytes)          Bandwidth(MB/s)
  33554432                       1501.9

Device to Host Bandwidth, 1 Device(s), Paged memory
  Transfer Size (Bytes)          Bandwidth(MB/s)
  33554432                       1223.8

Device to Device Bandwidth, 1 Device(s)
  Transfer Size (Bytes)          Bandwidth(MB/s)
  33554432                       41230.3
```

```
#include <stdio.h>
#include <malloc.h>
float serial(float* f, long N);
__global__ void summator(float* f,float* s, long N);
float parallel(float* f,long N, int num_of_blocks, int
                              threads_per_block);
int main(int argc, char* argv[]){
 long N;
 int i;
 float* fun;
 int     num_of_blocks, threads_per_block;
 if(argc<4) {                          printf("USAGE: test1
<array_size> <num_of_blocks>
<threads_per_block>\n"); return -1;  }
```

```
 N=atoi(argv[1]);
 num_of_blocks=atoi(argv[2]);
 threads_per_block=atoi(argv[3]);
 fun=(float*)malloc(N*sizeof(float));

 for(i=0;i<N;i++)
     fun[i]=((i+0.5F)*(1.0/N))*((i+0.5F)*(1.0/N));

 printf("Serial calculation is over! Result=%g\n",
                              serial(fun,N));
 printf("Parallel calculation is over! Result=%g\n",
parallel(fun,N,num_of_blocks, threads_per_block));
                                          return 0;
}
```

```
float serial(float* f, long N){
    int i;
    double s=0.0;

    for(i=0;i<N; i++)
            s+=f[i];

    return s/(float)N;
}
```

```
float parallel(float* f,long N, int num_of_blocks,
      int threads_per_block){
  float* f_dev;
  float* s_dev;
  float* s_host;
  float s=0.0;
  int i;

  cudaMalloc((void **) &f_dev, N*sizeof(float) );
  cudaMemcpy(f_dev, f, N*sizeof(float),
        cudaMemcpyHostToDevice);
```

```
s_host=(float*)malloc(
num_of_blocks*threads_per_block*sizeof(float))cudaM
alloc((void **) &s_dev,
num_of_blocks*threads_per_block*sizeof(float));

  for(i=0;i<num_of_blocks*threads_per_block;i++)
                                  s_host[i]=0.0;
  cudaMemcpy(s_dev, s_host,
num_of_blocks*threads_per_block*sizeof(float),
cudaMemcpyHostToDevice);
```

```
summator<<<num_of_blocks,threads_per_block>>>(f
_dev, s_dev,N);
  cudaThreadSynchronize();

  cudaMemcpy(s_host, s_dev,
num_of_blocks*threads_per_block*sizeof(float) ,
cudaMemcpyDeviceToHost);

  for(i=0;i<num_of_blocks*threads_per_block;i++)
                                              s+=s_host[i];
  return s/(float)N;
}
```

```
__global__ void summator(float* f,float* s, long N){
    int tId = blockIdx.x*blockDim.x+threadIdx.x;
    int num_of_threads=blockDim.x*gridDim.x;
    int portion=N/num_of_threads;
    int i;

    for(i=tId*portion;i<(tId+1)*portion;i++)
        s[tId]+=((i+0.5F)*(1.0/N))*((i+0.5F)*(1.0/N));
}
```

```
 > nvcc test1.cu   -o test1
```

```
 > test1 327680000  1024 32
Serial calculation is over! Result=0.333333
Parallel calculation is over! Result=0.333333
```