Tutorial:
Parallel programming technologies on hybrid architectures

HybriLIT Team

Laboratory of Information Technologies

Joint Institute for Nuclear Research

The Helmholtz International Summer School

"**Lattice QCD, Hadron Structure and Hadronic Matter**"

5, September 2014

Tutorial's materials

https://indico-hybrilit.jinr.ru

Parallel programming technologies on hybrid architectures II

Friday, 5 September 2014 from **08:00** to **18:00** (Europe/Moscow)
at **LIT JINR (407)**
Dubna, Russia

Description In-depth study of algorithms' parallelization for computations on hybrid architectures:
study of OpenMP, MPI technologies;
optimization of CUDA applications;
study of OpenCL technologies;
study of MPI+CUDA hybrid technologies;
research on comparative analysis of the efficiency of using GPU, multi-core CPU and Intel Xeon Phi coprocessors.

Friday, 5 September 2014

12:10 - 12:25	Introduction: how to work on cluster "HybriLIT" *15'* Speakers: Evgeny Aleksandrov, Mikhail Matveev, Martin Vala, Dmitry Belyakov
12:25 - 13:10	Lecture and practical training: Parallel Programming with CUDA *45'* Speakers: Oksana Streltsova, Maxim Zuev
13:10 - 15:00	Break
15:00 - 16:00	Lecture and practical training: OpenCL parallel programming technology *1h0'* Speaker: Alexander Ayriyan
16:00 - 16:30	Koffee break
16:30 - 17:30	Practical training on the parallel programming technologies: Comparative analysis of the efficiency of the GPU, multi-core CPU, and Intel Xeon Phi coprocessor approaches *1h0'* Speakers: Maxim Zuev, Alexander Ayriyan, Oksana Streltsova, Tatiana Sapozhnikova

Parallel Programming with CUDA

Streltsova O., Zuev M.

Laboratory of Information Technologies

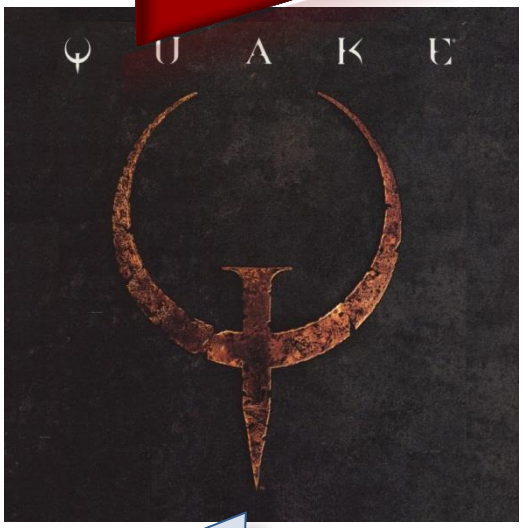
Joint Institute for Nuclear Research

The Helmholtz International Summer School

“**Lattice QCD, Hadron Structure and Hadronic Matter**”

5, September 2014

From 3D graphics to General-purpose graphics processing units (GPGPU)



Quake 1
(1996)
http://www.idsoftware.com/en-gb

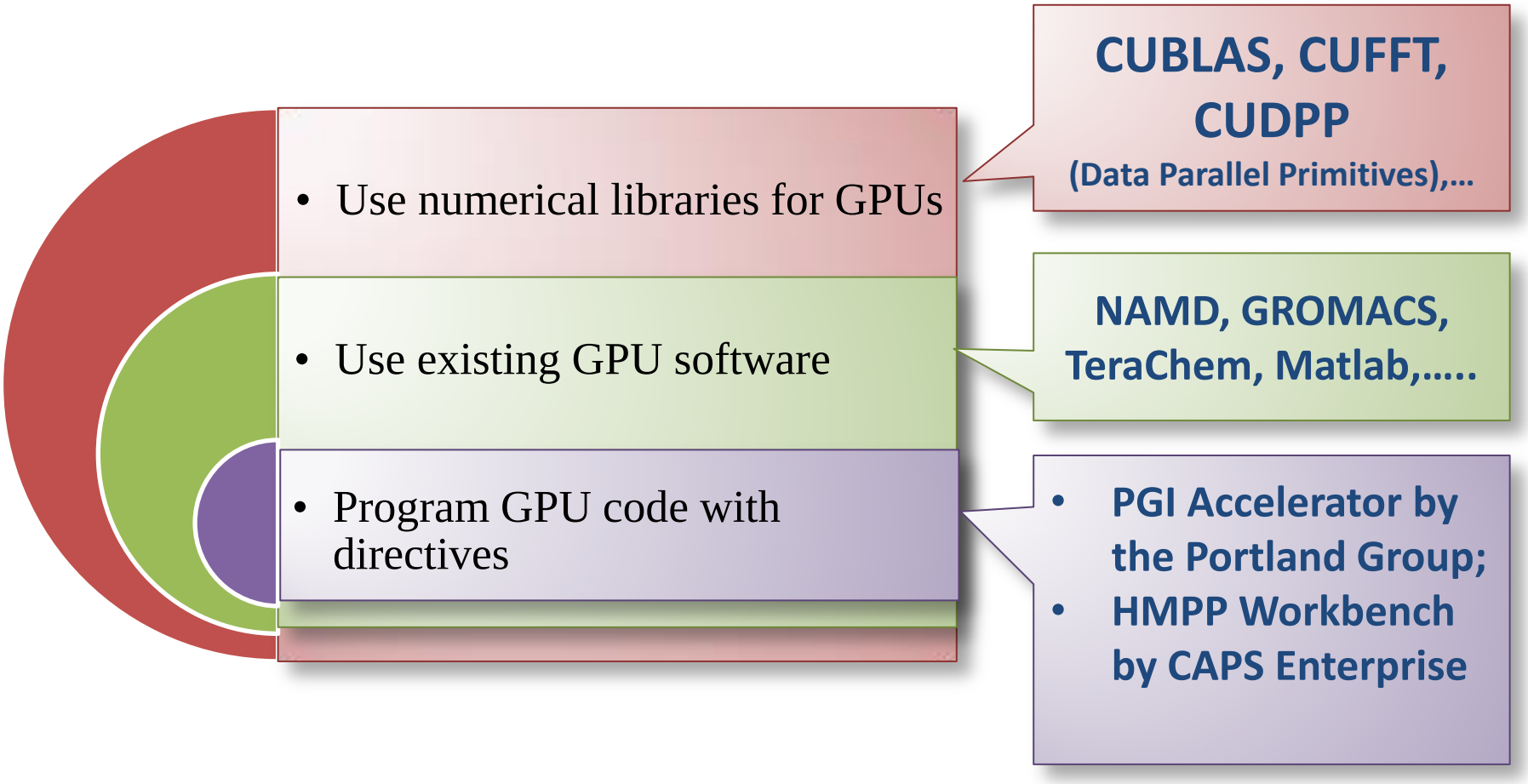
NVIDIA GeForce 8 (G80)

CUDA
NVIDIA unveiled
in 2006

Tesla K40 (2013)

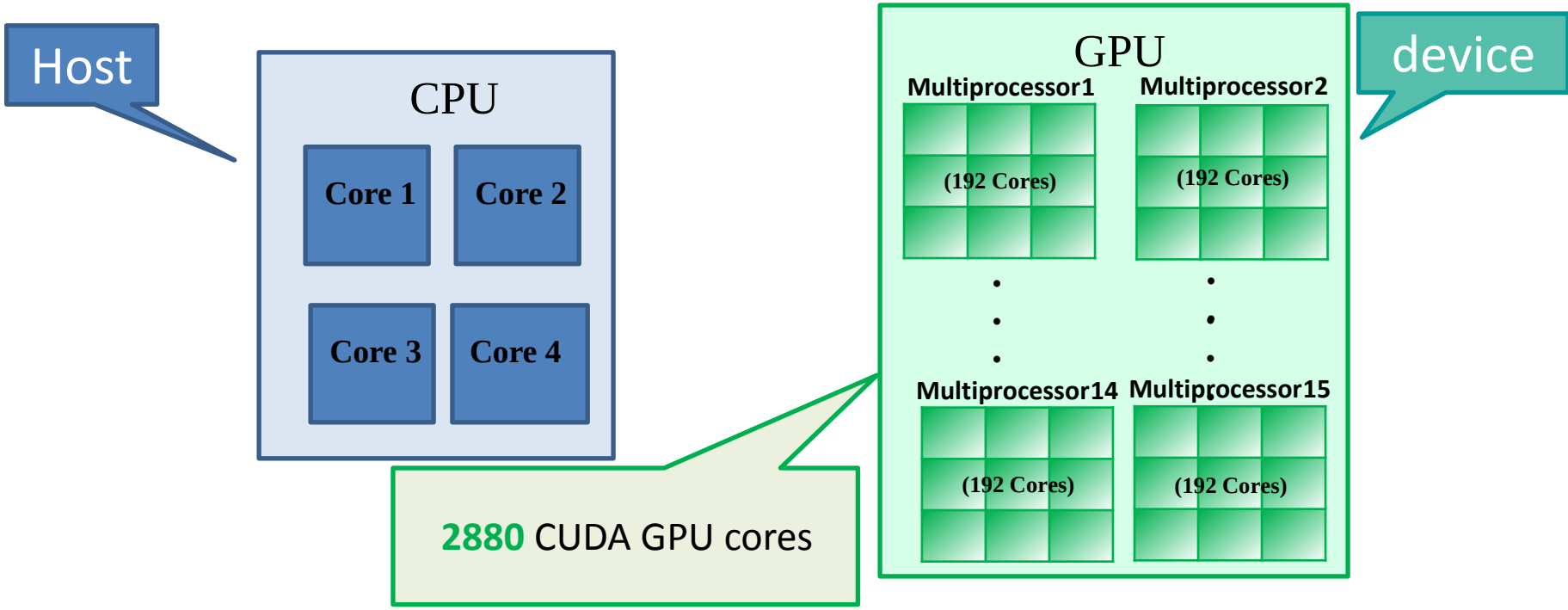
http://www.nvidia.ru/object/cuda-parallel-computing-ru.html

Effective and quick development parallel applications



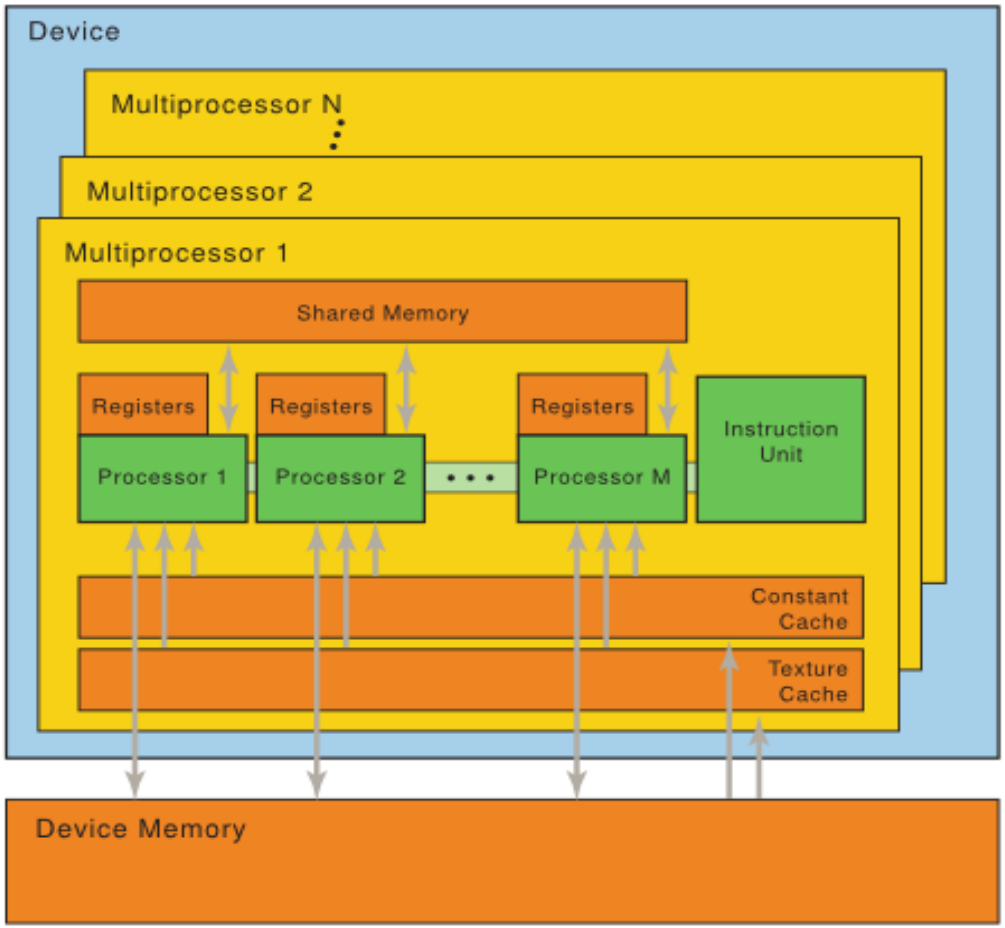
CUDA (Compute Unified Device Architecture) programming model, CUDA C

CPU / GPU Architecture



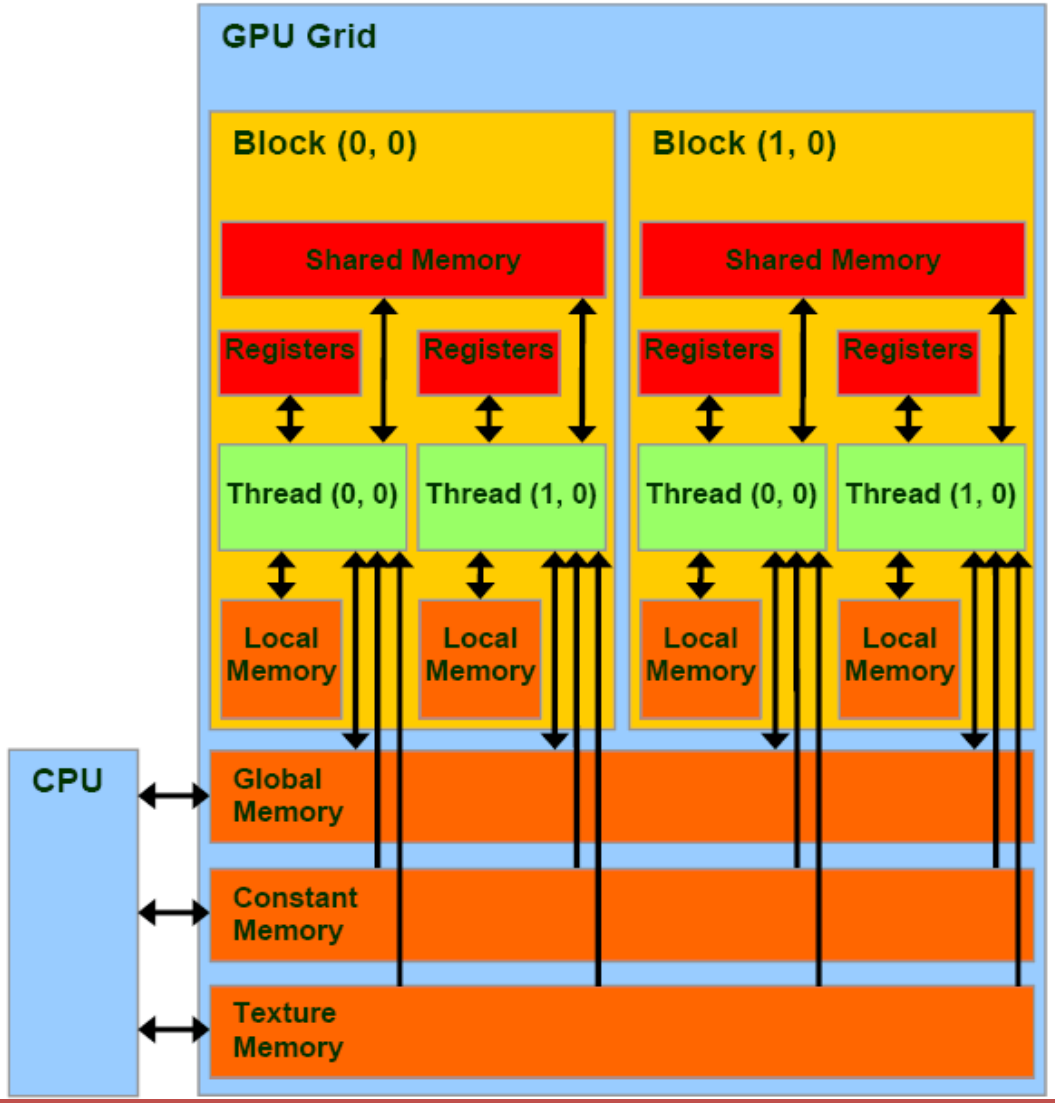
Source:
http://blog.goldenhelix.com/?p=374

CUDA (Compute Unified Device Architecture) programming model



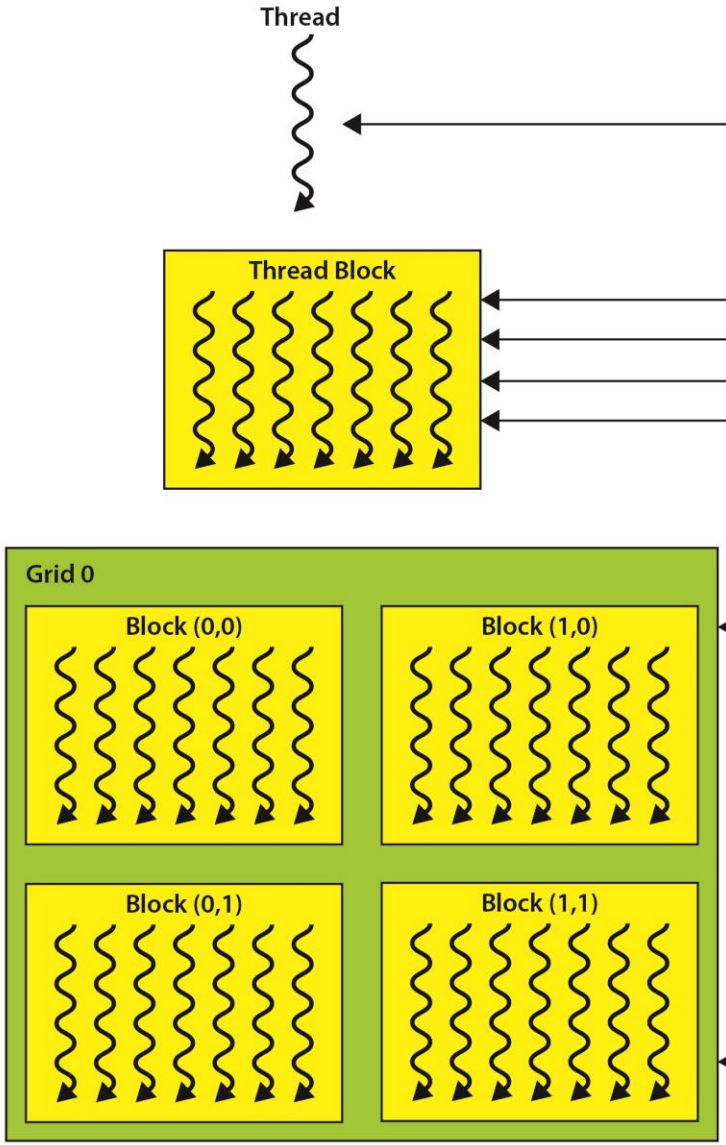
Source: docs.nvidia.com/cuda/parallel-thread-execution/graphics/hardware-model.png

CUDA (Compute Unified Device Architecture) programming model



Source: http://www.realworldtech.com/includes/images/articles/g100-2.gif

Device Memory Hierarchy



Registers are fast, off-chip local memory has high latency

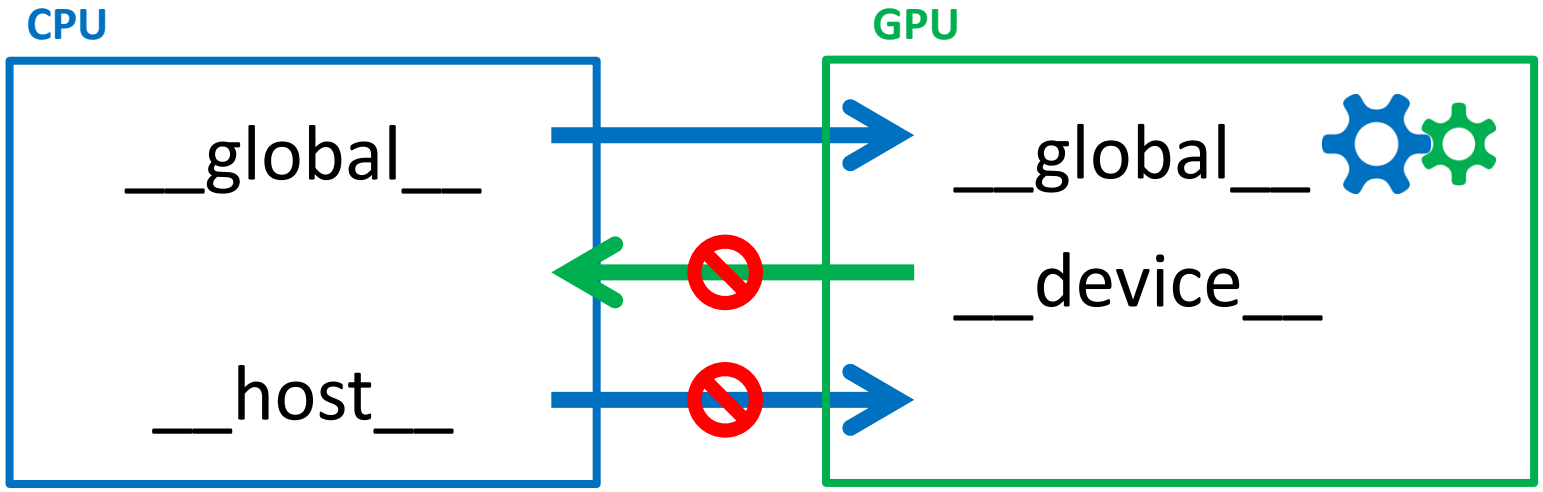
Tens of kb per block, on-chip, very fast

Size up to 12 Gb, high latency

Random access very expensive!
Coalesced access much more efficient

CUDA C Programming Guide (February 2014)

Function Type Qualifiers

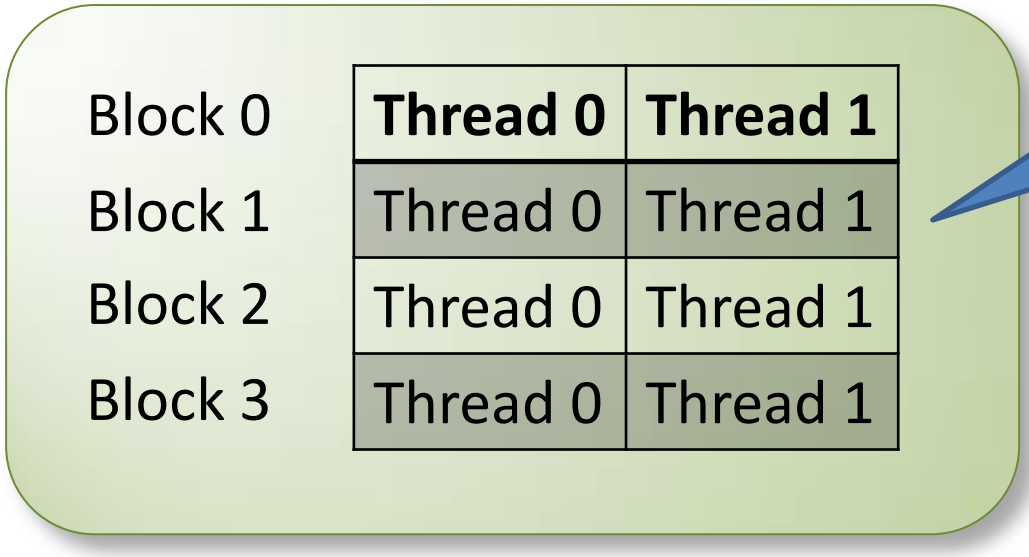



```
__global__ void kernel ( void ){
}

int main{
    …
    kernel <<< gridDim, blockDim >>> ( args );
    …
}
```

Language extensions:
Kernel execution directive

dim3 `gridDim` – dimension of grid of blocks,
dim3 `blockDim` – dimension of blocks

Threads and blocks




```
int tid = threadIdx.x + blockIdx.x * blockDim.x
```

Scheme program on CUDA C/C++ and C/C++

CUDA	C / C++
1. Memory allocation	
cudaMalloc (void ** devPtr, size_t size);	void * malloc (size_t size);
2. Copy variables	
cudaMemcpy (void * dst, const void * src, size_t count, enum cudaMemcpyKind kind);	void * memcpy (void * destination, const void * source, size_t num);
copy: host ⟶ device, device ⟶ host, host ⟷ host, device ⟷ device	---
3. Function call	
kernel <<< gridDim, blockDim >>> (args);	double * Function (args);
4. Copy results to host	
cudaMemcpy (void * dst, const void * src, size_t count, device ⟶ host);	---

Compilation

Compilation tools are a part of CUDA SDK

- NVIDIA CUDA Compiler Driver NVCC
  - Full information http://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc/#axzz37LQKVSFi

```
nvcc -arch=compute_35 test_CUDA_deviceInfo.cu -o test_CUDA –o deviceInfo
```

Device-info: example

Location:

hydra.jinr.ru/eos/hybrilit.jinr.ru/scratch/Tutorial_Hscool/CUDA/

Environment Modules:

-bash-4.1$ module add cuda-6.0-x86_64

Compilation:

-bash-4.1$ nvcc -gencode=arch=compute_35,code=sm_35
test_CUDA_deviceInfo.cu -o test1

Submit a job script for later execution:

--bash-4.1$ sbatch script_cuda

NVIDIA Tesla K40 “Atlas” GPU Accelerator

- **Supports CUDA and OpenCL**
- **Specifications**
  - **2880** CUDA GPU cores
  - **Performance**
    4.29 TFLOP single-precision;
    1.43 T FLOP double-precision
  - 12GB **of global memory**
  - Memory bandwidth up to 288 GB/s

Timing a CUDA application using events

```
1.  //----------------------------  Event Management--------------------//
2.  cudaEvent_t start, stop;
3.  cudaEventCreate(&start);
4.  cudaEventCreate(&stop);
5.  cudaEventRecord(start);     //   cudaEventRecord(start, 0);
6.  …. Some operation on CPU and GPU……
7.  cudaEventRecord(stop);
8.  float time0 = 0.0;
9.   cudaEventSynchronize(stop);
10.  cudaEventElapsedTime(&time0, start, stop);
11.   printf("GPU compute time_load data(msec): %.5f\n", time0);
```

Error Handling

Run-Time Errors:

every CUDA call (except kernel launches) return an error code of type

cudaError_t

Getting the last error:

cudaError_t cudaGetLastError (void)

```
1.  //------------------------------ Getting the last error  --------------------//
2.  cudaError_t error = cudaGetLastError();
3.  if(error != cudaSuccess){
4.     printf("CUDA error, CUFFT:  %s\n", CudaGetErrorString(error));
5.     exit(-1);
6.       }
```

Error Handling: uses macro substitution

```
1.    #define CUDA_CALL(x) do { cudaError_t err = x; if (( err ) != cudaSuccess ) { \
2.       printf ("Error \"%s\" at %s :%d \n" , cudaGetErrorString(err), \
3.       __FILE__ , __LINE__ ) ; return -1;\
4.     }} while (0);
```

that works with any CUDA call that returns an error code.

Example:

```
CUDA_CALL(cudaMalloc((void**) dA, n*n*sizeof(float)));
```

- GPU kernel calls return before completing, cannot immediately use cudaGetLastError to get errors:
  - First use **cudaDeviceSynchronize()** as a synchronization point;
  - Then call **cudaGetLastError**.

Error Handling: Example

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

#define NX 64

#define CUDA_CALL(x) do { cudaError_t err = x; if (( err ) != cudaSuccess ){ \
printf ("Error \"%s\" at %s :%d \n" , cudaGetErrorString(err), \
__FILE__ , __LINE__ ) ; return -1;\
}} while (0);

int main() {
  printf(" ================ Test Error Handling ================ \n" );
  printf("Array size  NX = %d  \n", NX);
  // Input data: allocate memory on CPU
  double* A_input;
  A_input = new double[NX*NX];

  for (int ix= 0; ix< NX*NX; ix++){
    A_input[ix]= (double)ix;
  }
```

...

Error Handling: Example (continue)

```
// Allocate memory on device
double* dev_A1;
double* dev_A2;
CUDA_CALL(cudaMalloc((void**)&dev_A1, NX*NX*sizeof(double)) );
CUDA_CALL(cudaMalloc((void**)&dev_A2, NX*NX*sizeof(double)) );
// Getting the last error
cudaError_t error = cudaGetLastError();
if(error != cudaSuccess){
  printf("CUDA error :  %s\n", cudaGetErrorString(error));
  exit(-1);
}
// Memory volume:
size_t avail, total;
cudaMemGetInfo( &avail, &total );  // in bytes
size_t used = total - avail;
printf("GPU memory  total: %lf  ( Mb) , Used memory:  %lf   ( Mb) \n",
          total/(pow(2,20)), used/(pow(2,20) ) );
```

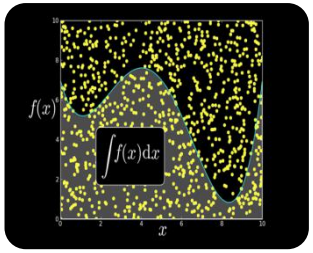
...

Error Handling: Example (end)

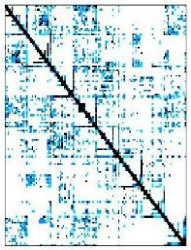
```
  // Copy input data from Host to Device
  cudaMemcpy(dev_A1 , A_input, NX*sizeof(double),
  cudaMemcpyHostToDevice);
  cudaDeviceSynchronize();

  // Free memory on CPU and GPU
  delete[] A_input;
  cudaFree(dev_A1);
  cudaFree(dev_A2);
  return 0;
}
```

Some GPU-accelerated Libraries

NVIDIA cuBLAS



NVIDIA cuRAND



NVIDIA cuSPARSE

NVIDIA NPP

GPU VSIPL

Vector Signal Image Processing

CULA tools

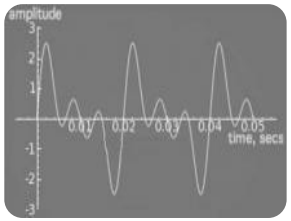
GPU Accelerated Linear Algebra

MAGMA

ICL UT

Matrix Algebra on GPU and Multicore

open source initiative



NVIDIA cuFFT

ROGUE WAVE SOFTWARE

IMSL Library

ArrayFire Matrix Computations

CUSP

Sparse Linear Algebra

Thrust

C++ STL Features for CUDA

Source: https://developer.nvidia.com/cuda-education. (Will Ramey ,NVIDIA Corporation)

cuBLAS library: cublas<t>scal()

The cuBLAS library is an implementation of BLAS (Basic Linear Algebra Subprograms) on top of the NVIDIA®CUDA runtime.

Site:

http://docs.nvidia.com/cuda/cublas/

- cublasStatus_t cublas**S**scal(cublasHandle_t handle, int n,
  const float *alpha, float *x, int incx)
- cublasStatus_t cublas**D**scal(cublasHandle_t handle, int n,
  const double *alpha, double *x, int incx)
- cublasStatus_t cublas**C**scal(cublasHandle_t handle, int n,
  const cuComplex *alpha, cuComplex *x, int incx)
- cublasStatus_t cublas**Cs**scal(cublasHandle_t handle, int n,
  const float *alpha, cuComplex *x, int incx)
- cublasStatus_t cublas**Z**scal(cublasHandle_t handle, int n,
  const cuDoubleComplex *alpha,
  cuDoubleComplex *x, int incx)
- cublasStatus_t cublas**Zd**scal(cublasHandle_t handle, int n,
  const double *alpha, cuDoubleComplex *x, int incx)

cuBLAS library: example

Location:

hydra.jinr.ru/eos/hybrilit.jinr.ru/scratch/Tutorial_HSchool

Module:

-bash-4.1$ module add cuda-6.0-x86_64

Compilation:

-bash-4.1$ nvcc -gencode=arch=compute_35,code=sm_35 -lcublas cuda_blas.cu -o test1

Submit a job script for later execution:

--bash-4.1$ sbatch script_cuda

cuBLAS library: example

```
script_CUDA
#!/bin/sh
#SBATCH -p gpu
 srun test1
```

Submit a job script for later execution:

--bash-4.1$ sbatch script_cuda