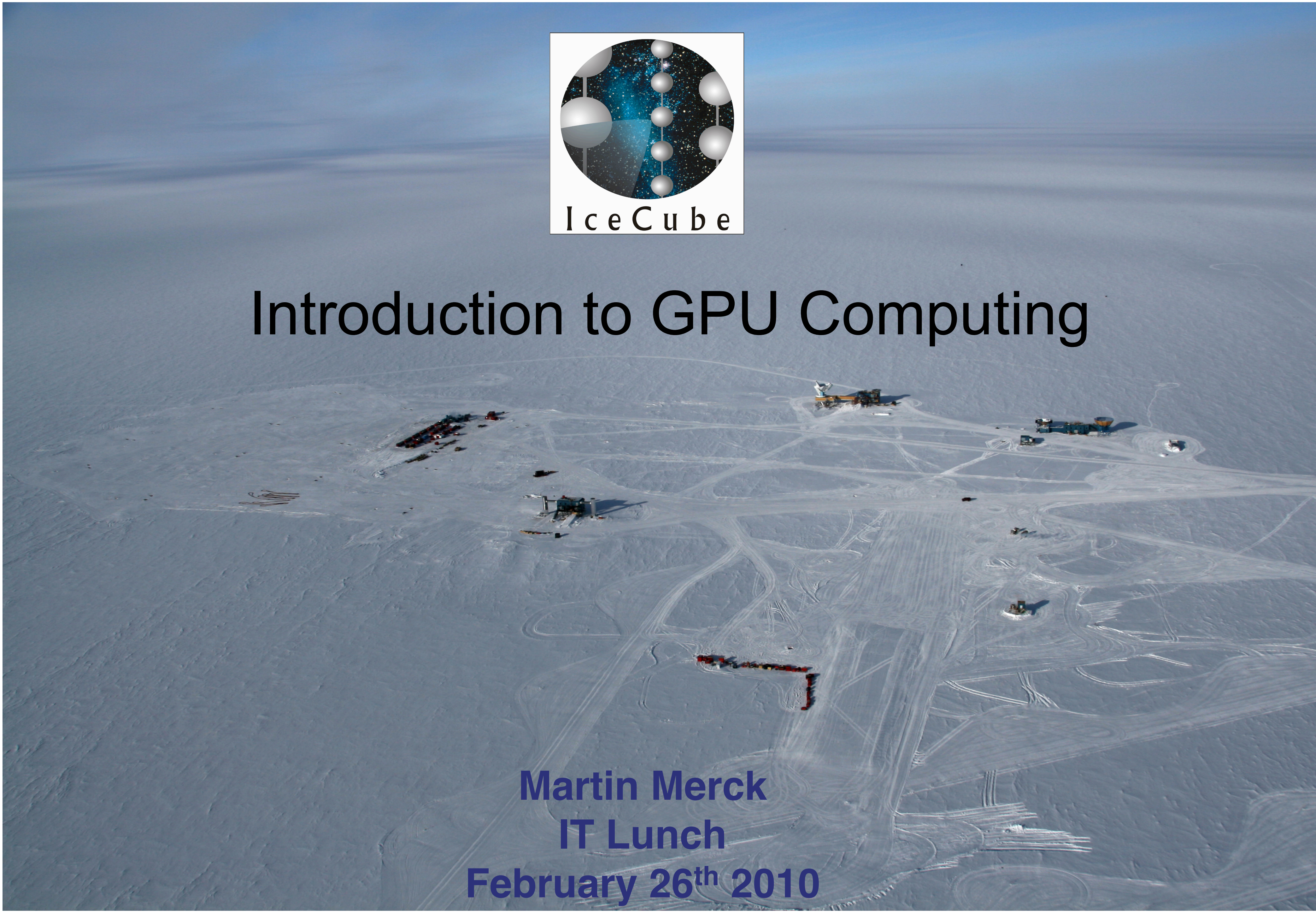


Introduction to GPU Computing

Martin Merck
IT Lunch
February 26th 2010

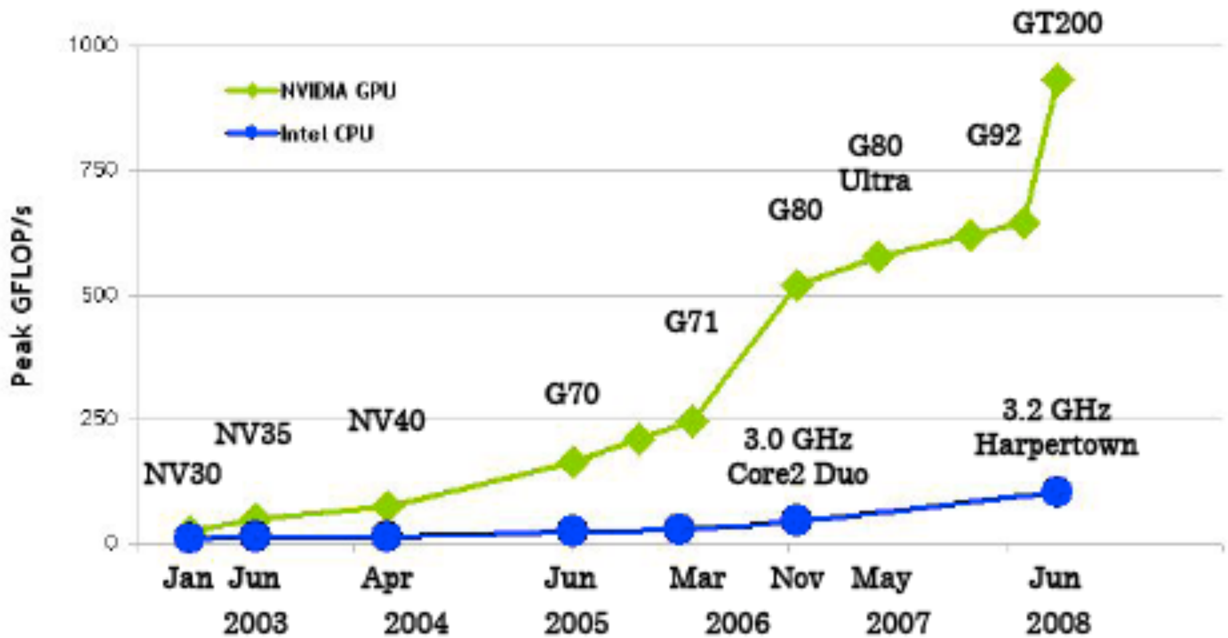
What is GPU computing

IT Lunch
February 26th
2010

- 3D Graphic cards need to perform lots of matrix operations to render images
  - To achieve the high throughput massively parallel computing is implemented on graphic cards
  - Basically old vector processing at a much bigger scale
  - CHEAP !!! (Gaming hardware is driving the development and has a mass market)

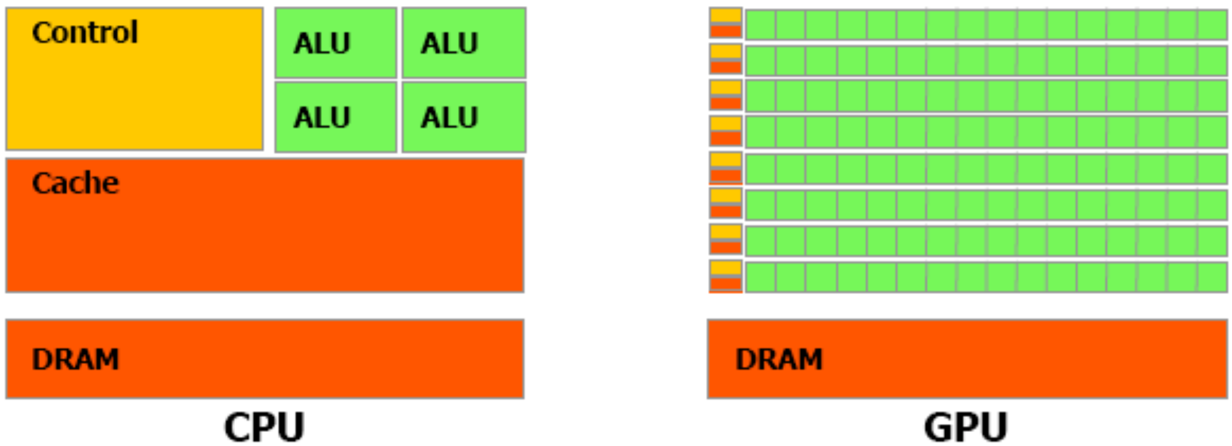
Computing power evalution

IT Lunch
February 26th
2010



Difference between a classical CPU and the GPU architecture

IT Lunch
February 26th
2010

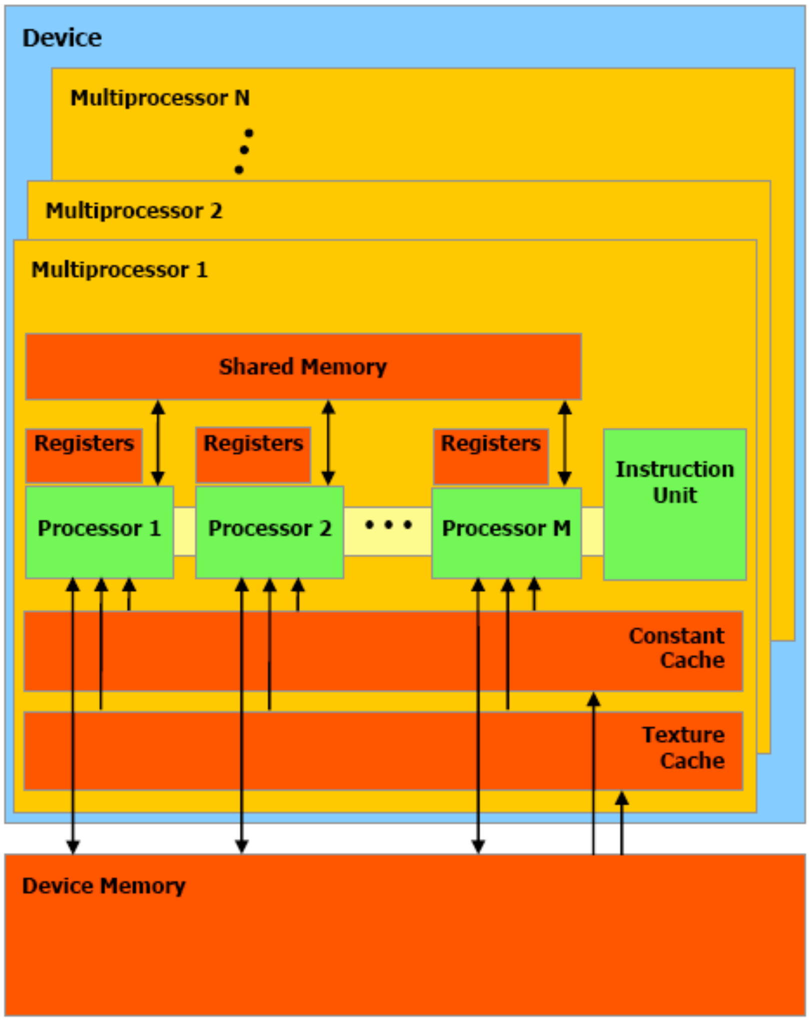


- CPU
  - Independent cores
  - Lots of cache
  - Prefetching and branch prediction
  - Heavy weight threading
- GPU
  - Single instruction multiple thread (SIMT)
  - Little cache, lots of registers
  - Overhead free threading
  - No real branching

IceCube

Detailed architecture of the NVidia Graphics cards

IT Lunch
February 26th
2010



- Basic building block is a Streaming Multiprocessor (SM) (30 on GTX 295)
  - Executes the same program on all processors
  - Branching handled by executing all branches and turning of single processors
  - Each multiprocessor consists of 8 Scalar Processor (SP), 2 special function units and shared (fast) memory
  - Runs 32 threads concurrently (called warp)

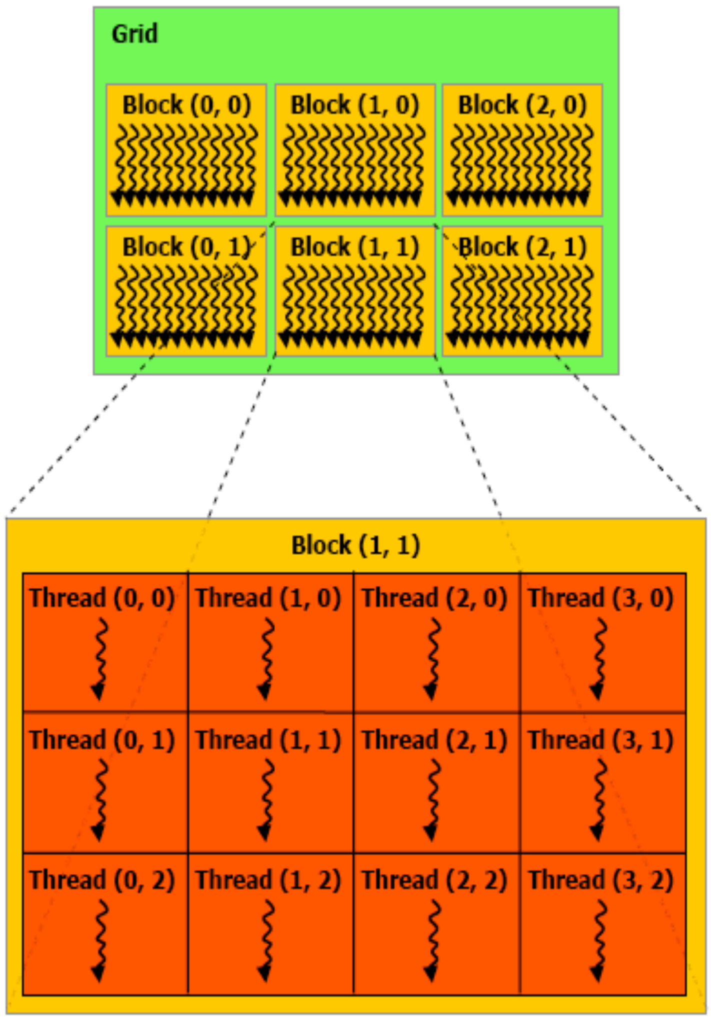
The CUDA software architecture

IT Lunch
February 26th
2010

- Basic programming unit is a "Kernel"
  - Represents a function which is executed on a CUDA device un a huge number of parallel threads. (~1000 - 10000)
  - Individual threads are combined into "blocks". A block can have up to 3 dimensions to map easily to vectors, matrices and fields
  - Each block is executed on one multiprocessor
  - Blocks are group at a higher level into one or two dimensional "Grids". This allows to execute the same kernel on several of the multiprocessors

Software

IT Lunch
February 26th
2010



Each thread is the same code.
To select different data to process they use global variables for the blockIdx/ gridDim and threadIdx/ blockDim. These are 3 dimensional indices.

IceCube

The CUDA software architecture

IT Lunch
February 26th
2010

- Basic programming unit is a “Kernel”
  - Represents a function which is executed on a CUDA device un a huge number of parallel threads. (~1000 - 10000)
  - Individual threads are combined into “blocks”. A block can have up to 3 dimensions to map easily to vectors, matrices and fields
  - Each block is executed on one multiprocessor
  - Blocks are group at a higher level into one or two dimensional “Grids”. This allows to execute the same kernel on several of the multiprocessors

A simple code example Matrix Multiplication 1

IT Lunch
February 26th
2010

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

// Matrix multiplication kernel – per thread code
__global__ void MatrixMulKernel(float* A, float* B, float* C, int Width)
{
    float Celement= 0;
    for (int k = 0; k < Width; ++k) {
       float Aelement = A[ threadIdx.y * Width + k ];
       float Belement = B[ k * Width + threadIdx.x ];
       Celement += Aelement * Belement;
    }

    B[ threadIdx.y * Width+threadIdx.x ] = Celement;
}
```

A simple code example
Matrix Multiplication 1

IT Lunch
February 26th
2010

```
int main(void)
{
   float *a_h, *b_h;                                  // pointers to host memory
   float *a_d, *b_d, *c_d;                   // pointer to device memory
   int N = 10*10;
   size_t size = N*sizeof(float);
   // allocate arrays on host
   a_h = (float *) malloc(size);
   b_h = (float *) malloc(size);
   // Initialize arrays

  cudaMalloc((void **) &a_d, size); // allocate array on device
  cudaMalloc((void **) &b_d, size); // allocate array on device
  cudaMalloc((void **) &c_d, size); // allocate array on device
  cudaMemcpy(a_d, a_h, sizeof(float)*N, cudaMemcpyHostToDevice); // copy data from h2d
  cudaMemcpy(b_d, b_h, sizeof(float)*N, cudaMemcpyHostToDevice)

   MatrixMulKernel<<< 1, N*N>>> (a_d, b_d, c_d, 10);  // do calculation on device
   cudaMemcpy(c_h, a_d, sizeof(float)*N, cudaMemcpyDeviceToHost);
   free(a_h); free(b_h);
   cudaFree(a_d); cudaFree(b_d); cudaFree(c_d);
}
```

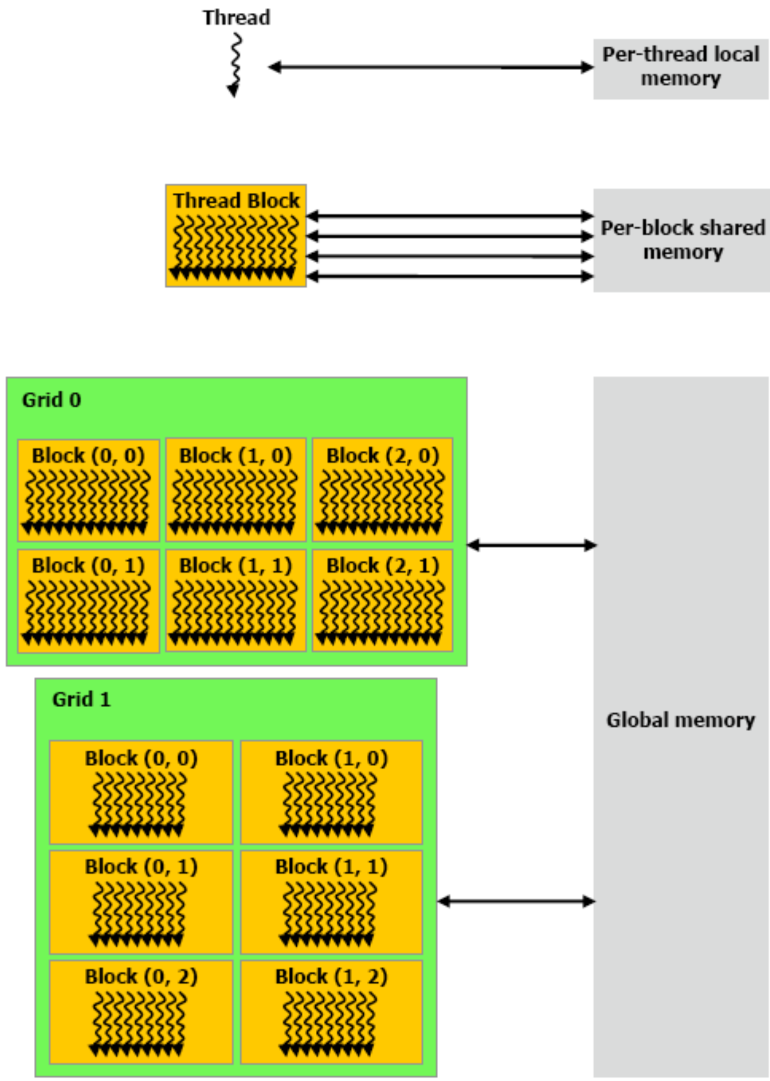
nvcc
The CUDA C-Compiler

IT Lunch
February 26th
2010

- Code is C-like
  - No C library functions can be used, but cuda library replicates all math functions and basic malloc, memcpy etc.
  - No recursions
  - No variable list parameters
  - No static variables
  - Just basic operators and flow control
- Compiler separates device and host code and compiles device code.
- Does all register allocation etc.

Memory models in CUDA

IT Lunch
February 26th
2010



- Each thread has a set of registers and local variables.
- A thread block can share fast memory between all threads. Threads in a block can be synchronized.
- Threads in different blocks and grids can only share global device memory.
- The host process can only access the global memory area

Streams
Parallel execution of kernels

IT Lunch
February 26th
2010

- CUDA host functions are all asynchronous.
- Streams can be used to group host/memory IO and kernel execution into parallel flows.
- Overlaps IO and kernel execution to use device while other IO is being performed.

IceCube

Conclusions

IT Lunch
February 26th
2010

- Easy to learn (basics)
- Cheap to build test/play systems
- Works great for highly parallel compute intensive applications
(Image processing, video en-/decoding, neural nets, SVM)
- Algorithms need lots of optimization to hardware limitations
- Lots of applications already available or having optimized plugins
(Mathematica, MatLab, Adobe Flash, NueralNet tools, etc.)

Showcase

IceCube

IT Lunch
February 26th
2010

More info

- http://www.nvidia.com/object/cuda_home_new.html
- http://developer.nvidia.com/object/gpucomputing.html
- http://www.drdobbs.com/architect/207200659
- http://courses.ece.illinois.edu/ece498/al/Syllabus.html