FDTD on the GPU

Jesse Lu

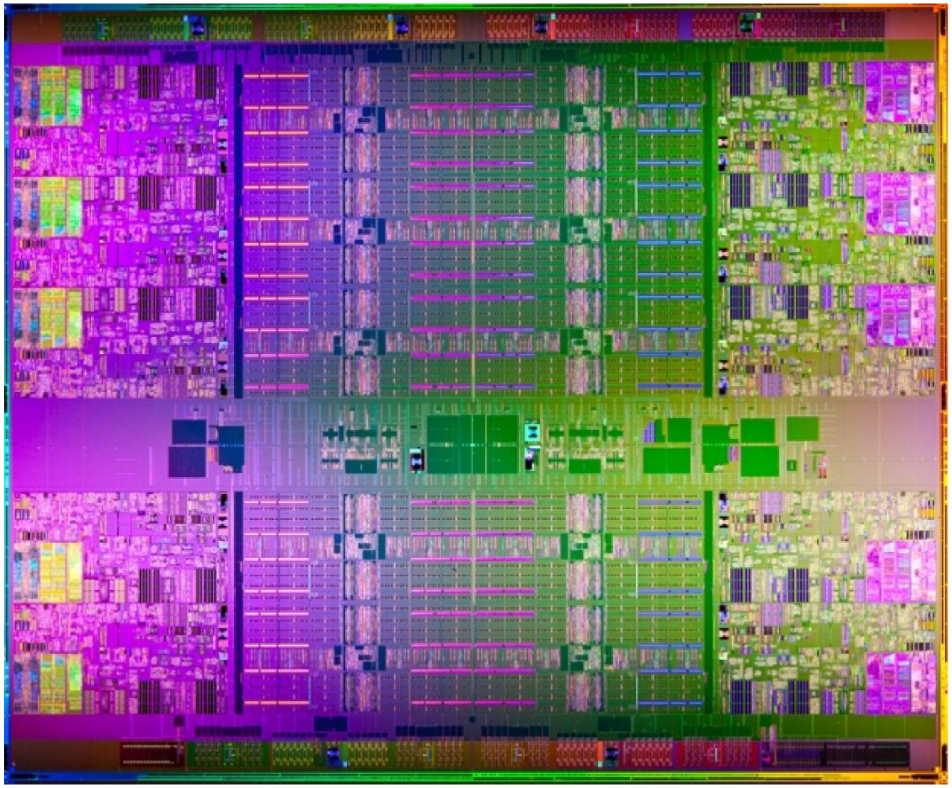
May 21, 2013

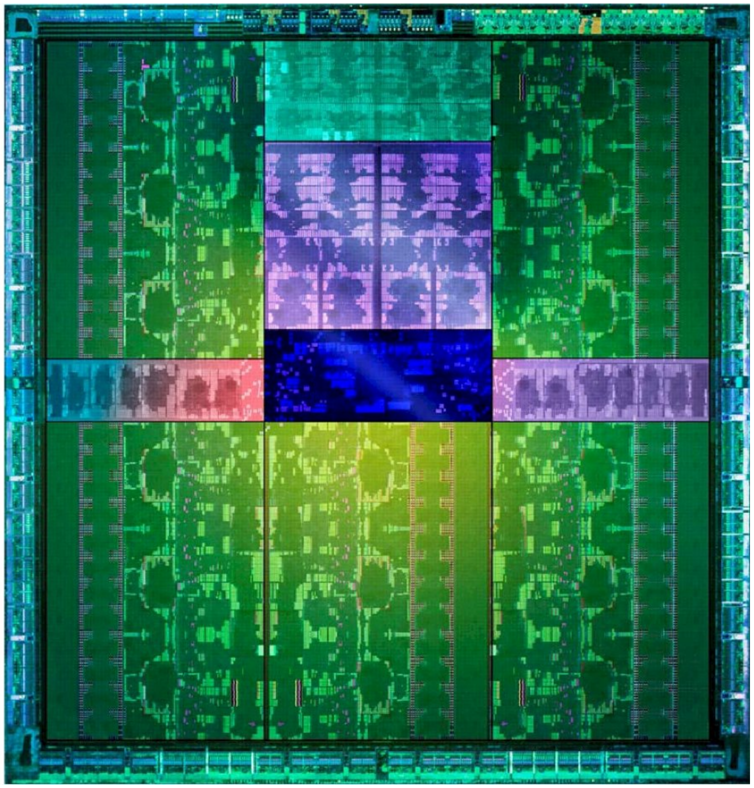
- The GPU today
- CPU vs GPU
- Programming the GPU
- Implementing FDTD on the GPU
- Optimizing FDTD on the GPU

The GPU today

- GPU (Graphical Processing Unit) once only for 3D gaming, now everywhere.
- Present in virtually all computing form factors
  - Raspberry Pi: 24 GFLOPS Broadcom GPU
  - Titan Supercomputer: 20 PetaFLOPS, 18K+ Nvidia GPUs
- Plays key role in numerous applications
  - graphics: games, video encoding/decoding, . . .
  - science: protein folding, FDTD simulations, . . .
  - misc.: bitcoin mining, password cracking, . . .

CPU vs. GPU





	CPU (Intel Xeon E7)	GPU (Nvidia Kepler)
Number of cores	10	2940
Cache size	30MB	$\sim$2MB
FLOPS (single-precision)	0.67 TFLOPS	3.7 TFLOPS
FLOPS (double-precision)	0.33 TFLOPS	1.2 TFLOPS
Memory bandwidth	102 GB/s	250 GB/s
Power draw	130 W	250 W

- Spec comparison needs to always be taken with a grain of salt (e.g. "theoretical maximums")

- Question: What kind of workloads are most suitable for the CPU? the GPU?

Programming the GPU

- Will focus on Nvidia GPUs and the CUDA programming language
- CUDA uses the SIMD (Single Instruction Multiple Data) model for parallelization
- Programming model mirrors GPU architecture

Computational	Memory	Software
core	registers	thread
multiprocessor	cache/shared memory	thread block
GPU	global memory	block grid

Implementing FDTD on the GPU

- We'll just look at a 1D E-field update:

$$E^i = c_0 E^i + c_1 (H^i - H^{i-1}) \tag{1}$$

- This is how we would normally do it on the CPU

```
for (int i=0 ; i < N ; i++) {
    E[i] = c0 * E[i] + c1 * (H[i] - H[i-1]);
}
```

- The simplest CUDA version of the update would be:

```
E_update<<<1, N>>>();
...

__global__ update_E () {
    i = threadId;
    E[i] = c0 * E[i] + c1 * (H[i] - H[i-1]);
}
```

- Each cell is updated by a single thread

- All threads grouped in a single thread block

- Launching multiple blocks will allow us to utilize multiple multiprocessors

```
E_update<<<N/16, 16>>>();
...

__global__ update_E () {
    i = blockId * 16 + threadId;
    E[i] = c0 * E[i] + c1 * (H[i] - H[i-1]);
}
```

Optimizing FDTD on the GPU

- Basic optimization question to answer: Are we limited by the memory-bandwidth or by computational power?
- Optimizing memory access patterns is often the key to faster GPU codes
- *Shared memory* resides at the multiprocessor level, and quickly accessed by all cores within the multiprocessor
- We can optimize memory access by eliminating redundant H-field loads

- Shared memory version of the update function

```
E_update<<<N/16, 16>>>();
...

__global__ update_E () {
    i = blockId * 16 + threadId;
    s_i = threadId;
    __shared__ float s_H[17];
    s_H[s_i+1] = H[i];
    if (threadId == 0) { s_H[0] = H[i-1] };
    __syncthreads();
    E[i] = c0 * E[i] + c1 * (s_H[s_i+1] - H[s_i]);
}
```