Explicit Finite Difference Simulations with Graphics Processing Units

Igor Terentyev

TRIP Annual Meeting

02/20/2009

Floating Point Accelerators

Cell, ClearSpeed, FPGA, GPU, ...

- Different hardware architecture
- Special programming languages/techniques

GPUs (video cards):

- High performance (+ nice FLOP/$ and FLOP/W ratios):
  - 1 TFLOPS
  - 100 GB/s
  - 4 GB RAM
  - $1500
  - 160 W
- Architecture similar to GP CPU:
  - Manycore
  - Multithreaded (highly parallel)
- Programming similar to GP CPU:
  - Minimal C/C++ extensions

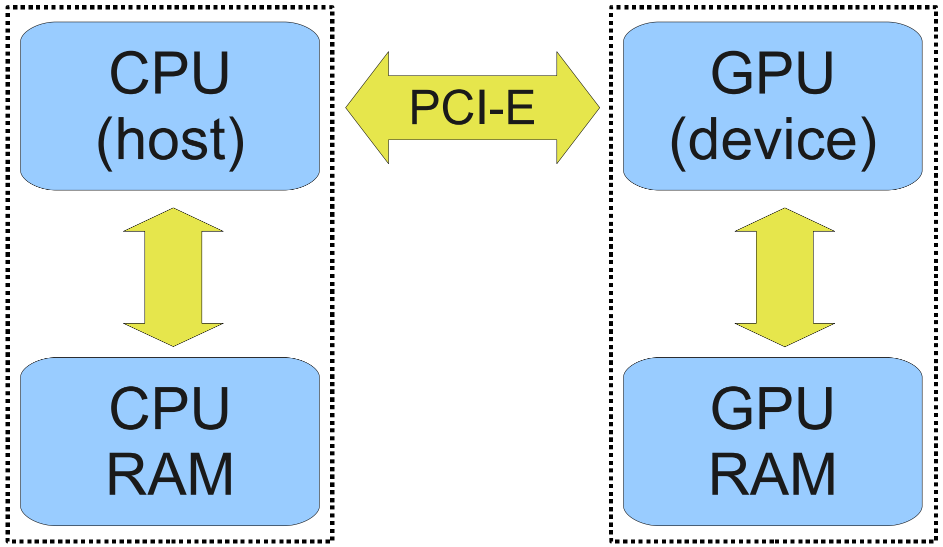
CUDA

Scalable parallel programming model and C language

NVIDIA, 2007

- Minimal C/C++ extension and compiler (device kernel functions)
- Single Instruction Multiple Threads (SIMT) approach
- Based on TESLA architecture

WORKFLOW



- GPU kernels can access device memory only
- CPU can `memcpy between` host and device memory

- Host: load data into Device memory
- Host: invoke device kernel (returns `void`)
- Host: wait for kernel to finish (do smth.)
- Host: unload results from Device memory

SIMPLE PROGRAM

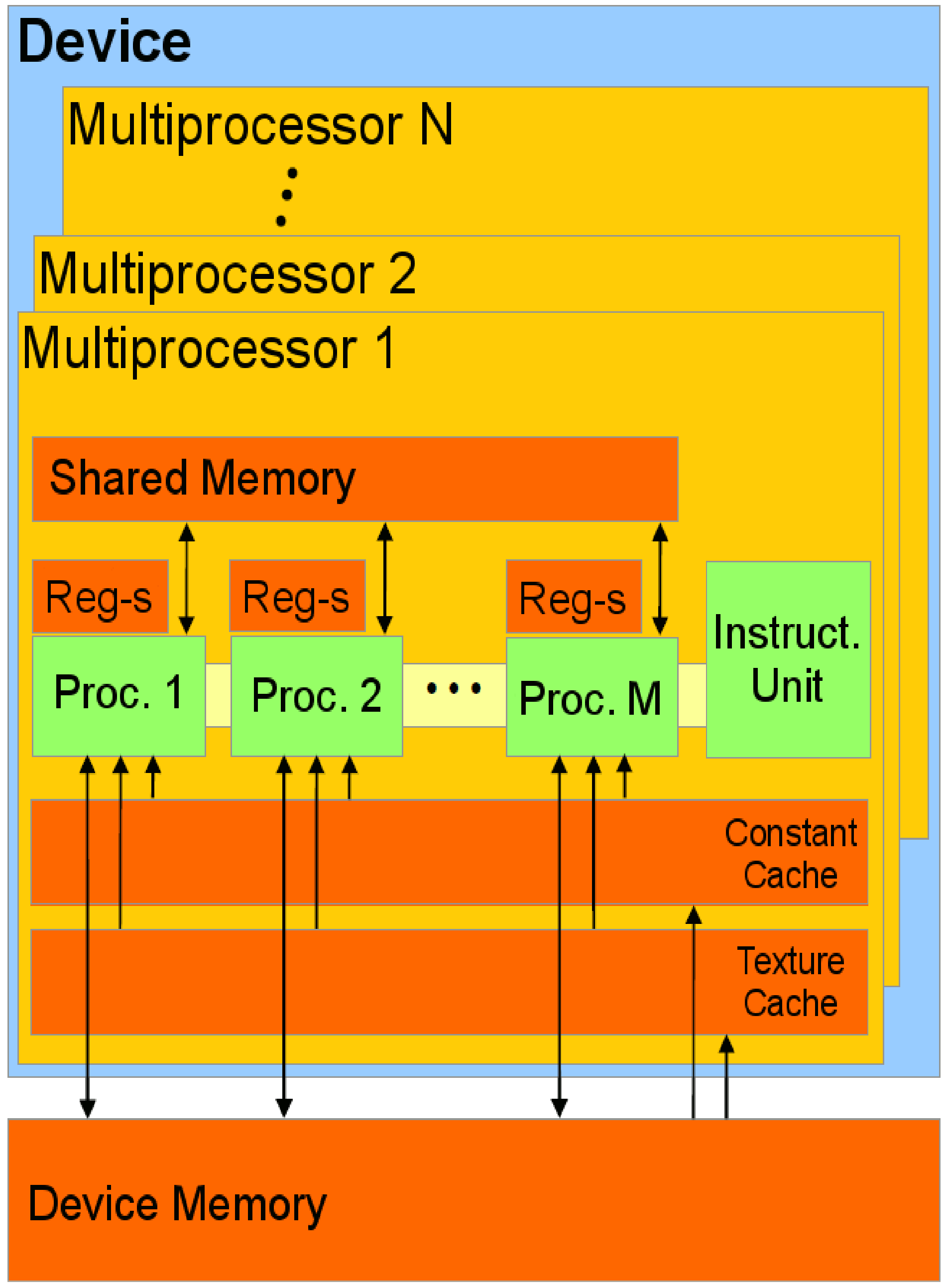
Serial:

```
void saxpy_s(int n, float alpha, float *x, float *y)
{
    for(int i = 0; i < n; ++i)
        y[i] = alpha * x[i] + y[i];
}
...
saxpy_s(4096, 0.5, x, y);
```

Parallel:

```
__device__
void saxpy_p(int n, float alpha, float *x, float *y)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    y[i] = alpha * x[i] + y[i];
}
...
saxpy_p<<<4096/256, 256>>>(4096, 0.5, x, y);
```

TESLA ARCHITECTURE



NVIDIA Programming Guide

Scalable array of SMs
(Streaming Multiprocessors)

- Common device memory
- Local resources (reg-s, etc.)
- Cannot communicate

- Kernel grid: each block on a separate SM
- Block distribution between SMs and order of execution undefined

CUDA Programming Specifics

Easy to start:

- Familiar architecture model
- Familiar programming language

Possible to create efficient code:

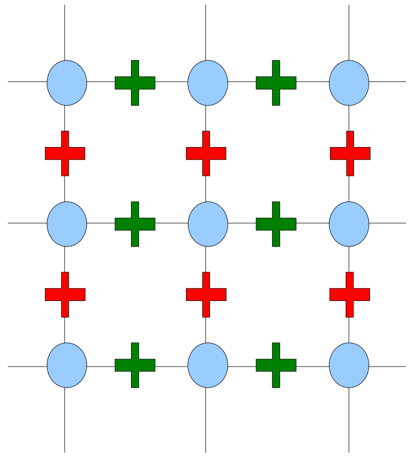
- High peak performance
- High memory bandwidth
- Hardware interpolation, etc.

Not so easy to create efficient code:

- Best utilization of resources
- Memory access patterns
- Hard to predict expected performance

FINITE DIFFERENCE SIMULATIONS

- Acoustic wave equation
- Pressure-velocity formulation
- 2D, staggered grid
- 2nd order in time, 4th order in space

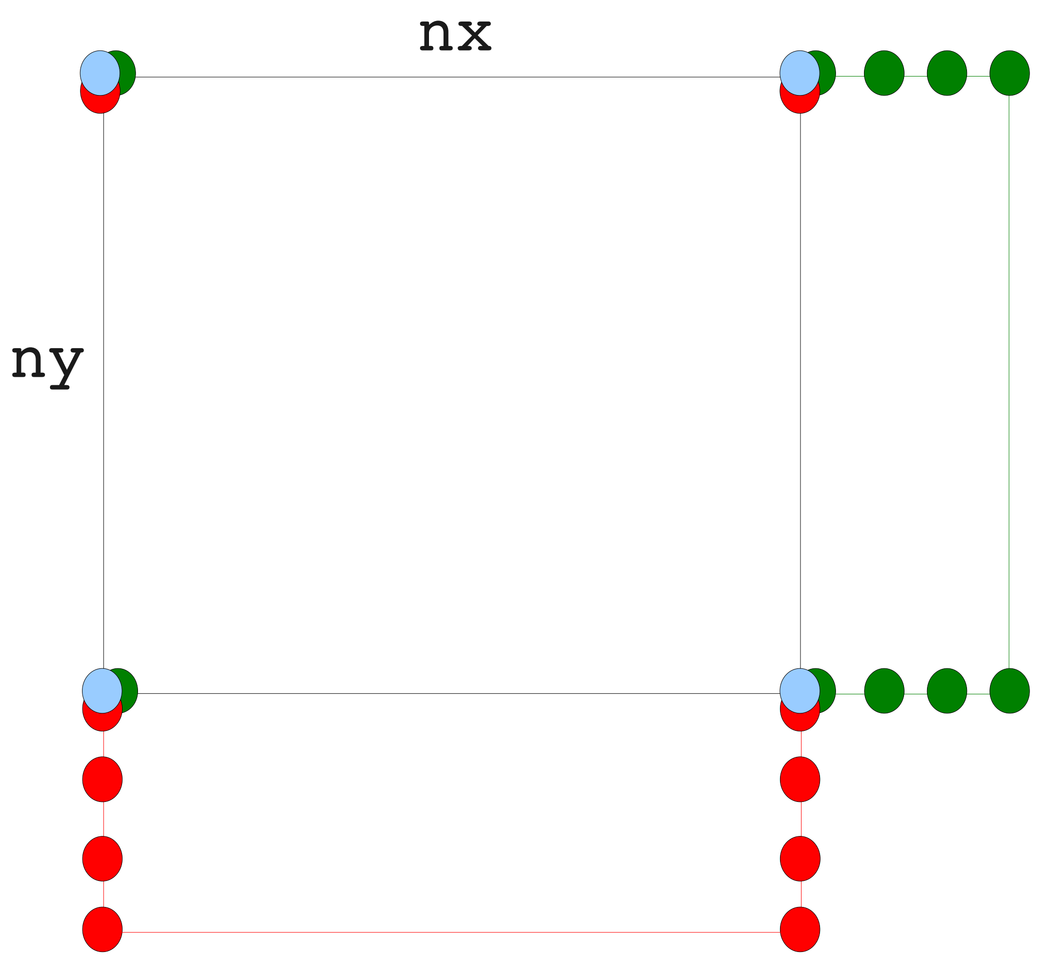


```
for i = 0; i < ny
    for j = 0; j < nx

        p[i,j] += b[i,j] *
          (
           LX * ( u[i,j+3]-u[i,j] + C*(u[i,j+1]-u[i,j+2]) )
           +
           LY * ( v[i+3,j]-v[i,j] + C*(v[i+1,j]-v[i+2,j]) )
          );

    end
end
```

ARRAY ORGANIZATION



```
P:   ny    x nx
U:   ny    x nx+3
V:   ny+3  x nx
```

PERFORMANCE ESTIMATE

Performance:

- `P` is read and written once and `B`, `U`, and `V` are read once
- 5 memory access ops, 4 bytes each
- 100 GB/sec / 20 = **$5*10^9$ grid points / sec**
- 13 FLOP / grid point = **65 GFLOPS**

Best result: **45 GFLOPS**

Kernels:

- No need to cache `P`, `B`
- Can use texture memory for `B`, `U`, and `V`
- Tried shared memory only

KERNELS

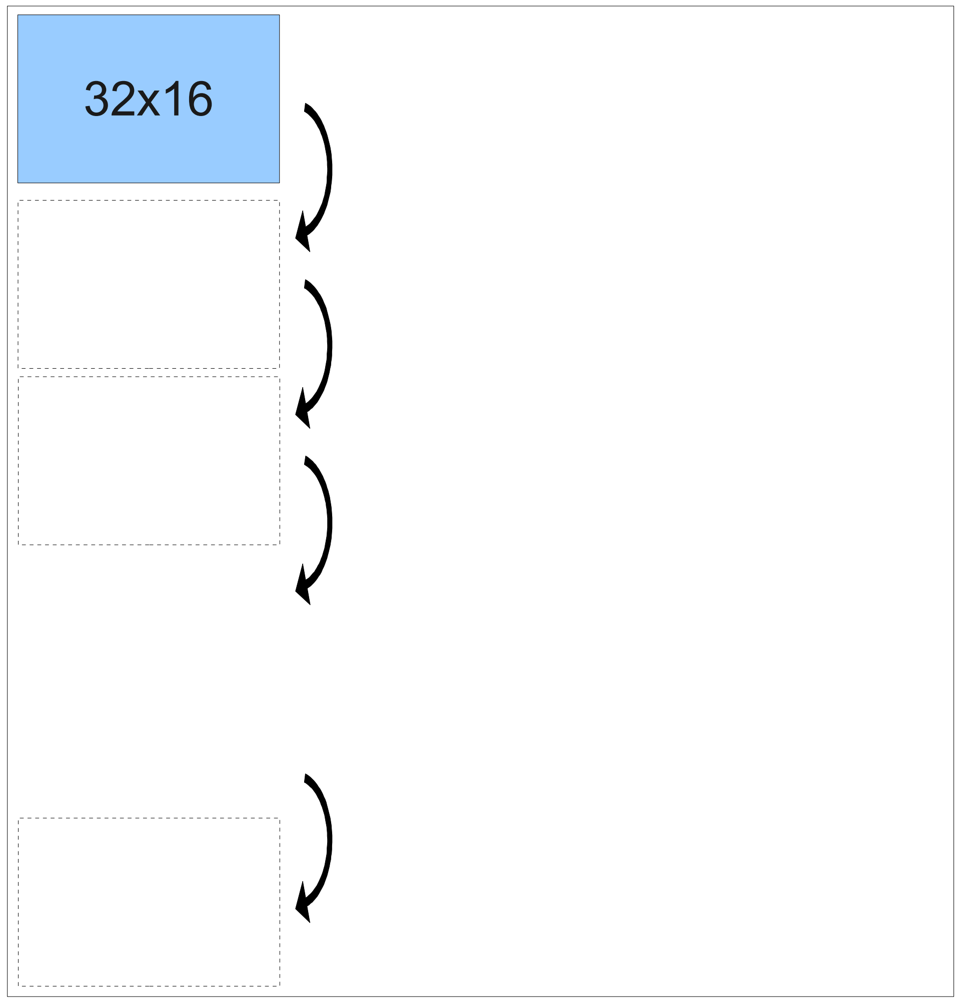
- Thread per outer loop entry
  - each thread performs inner loop
  - `ny` threads
- Thread per inner loop entry
  - each thread performs outer loop
  - `nx` threads

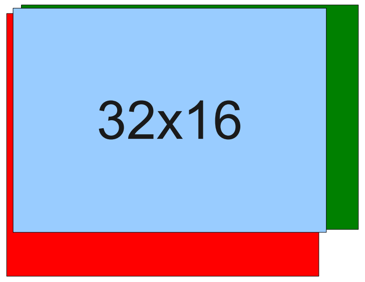
Results (4096 x 4096 grid size, TESLA C1060):

- 0.2 GFLOPS
- 25 GFLOPS

BLOCK KERNEL

Block kernel: 32 x 16 threads (512)
uses shared memory for velocities





BLOCK KERNEL CONT.

```
int tX = threadIdx.x;
int tY = threadIdx.y;
for i = 0; i < 512 / 16
    for j = 0; j < 512 / 32

        int ia = i*16+tY;     // Shifted index.
        int ja = j*32+tX;     // Shifted index.
        //--------------------------------------------------------------
        // Load u, v sub-block into shared memory su, sv.
        su[tY,tX] = u[ia,ja];                    // Main block.
        sv[tY,tX] = v[ia,ja];                    // Main block.

        if ( tx < 3 ) su[tY,tX+32] = u[ia,ja+32]; // Extra strip along y.
        if ( tx < 3 ) sv[tY+16,tX] = v[ia+16,ja]; // Extra strip along x.
        //--------------------------------------------------------------
        // Compute pressure.
        p[ia,ja] += b[ia,ja] *
        (
               LX * ( su[tY,tX+3]-su[tY,tX] + C*(su[tY,tX+1]-su[tY,tX+2]) )
               +
               LY * ( sv[tY+3,tX]-sv[tY,tX] + C*(sv[tY+1,tX]-sv[tY+2,tX]) )
        );
    end
end
```

RESULTS - TIMINGS

GPUs:

- NVIDIA GeForce GTX 280 GPU
- NVIDIA Tesla C1060

Data: 4000 x 4000 – 8000 x 8000

Performance:

- 35 – 50 GFLOPS for 2D 2-4 scheme
- 55 – 80 GFLOPS for 2D 2-8 scheme
- 90 – 115 GFLOPS for 2D 2-16 scheme

2-4 Scheme, GFLOPS

GPU	BK	IBK	BBK
TESLA	32.9	37.3	39.8
GTX	45.6	49.8	55.4

CONCLUSIONS

- Very appealing GPU FD kernels' performance
- Would like to have GPU kernels in IWAVE
  - work estimate: semester of 1 student's time

THANKS:

- Prof. Vivek Sarkar (Rice, CS Dep-t)
- Prof. Tim Warburton (Rice, CAAM Dep-t)