Scalable Parallel Programming with CUDA

John Nickolls, Ian Buck, Michael Garland and Kevin Skadron

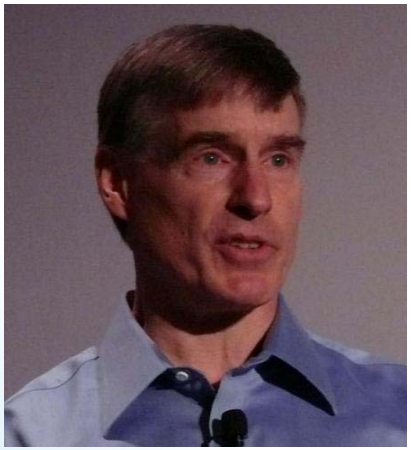
Presentation by Christian Hansen

Article Published in ACM Queue, March 2008

- About the Authors
- Hardware Platform
- What is CUDA?
- Programming in CUDA
- GPGPU/MC Programming Approaches
- Conclusion

Outline

About the Authors

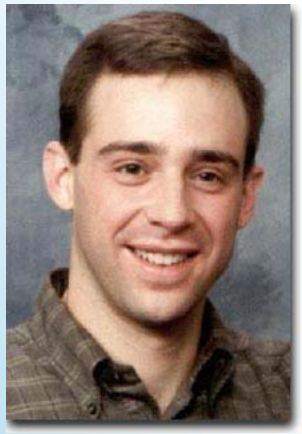


- John Nickolls
  - Director of Architecture at NVIDIA
  - MS and PhD degrees in electrical engineering from Stanford University
  - Previously at Broadcom and Sun Microsystems

- Ian Buck
  - GPU-Compute Software Manager at NVIDIA
  - PhD in computer science from Stanford
  - Has previously worked on Brook

About the Authors

- Michael Garland
  - Research scientist at NVIDIA
  - PhD in computer science at Carnegie Mellon University



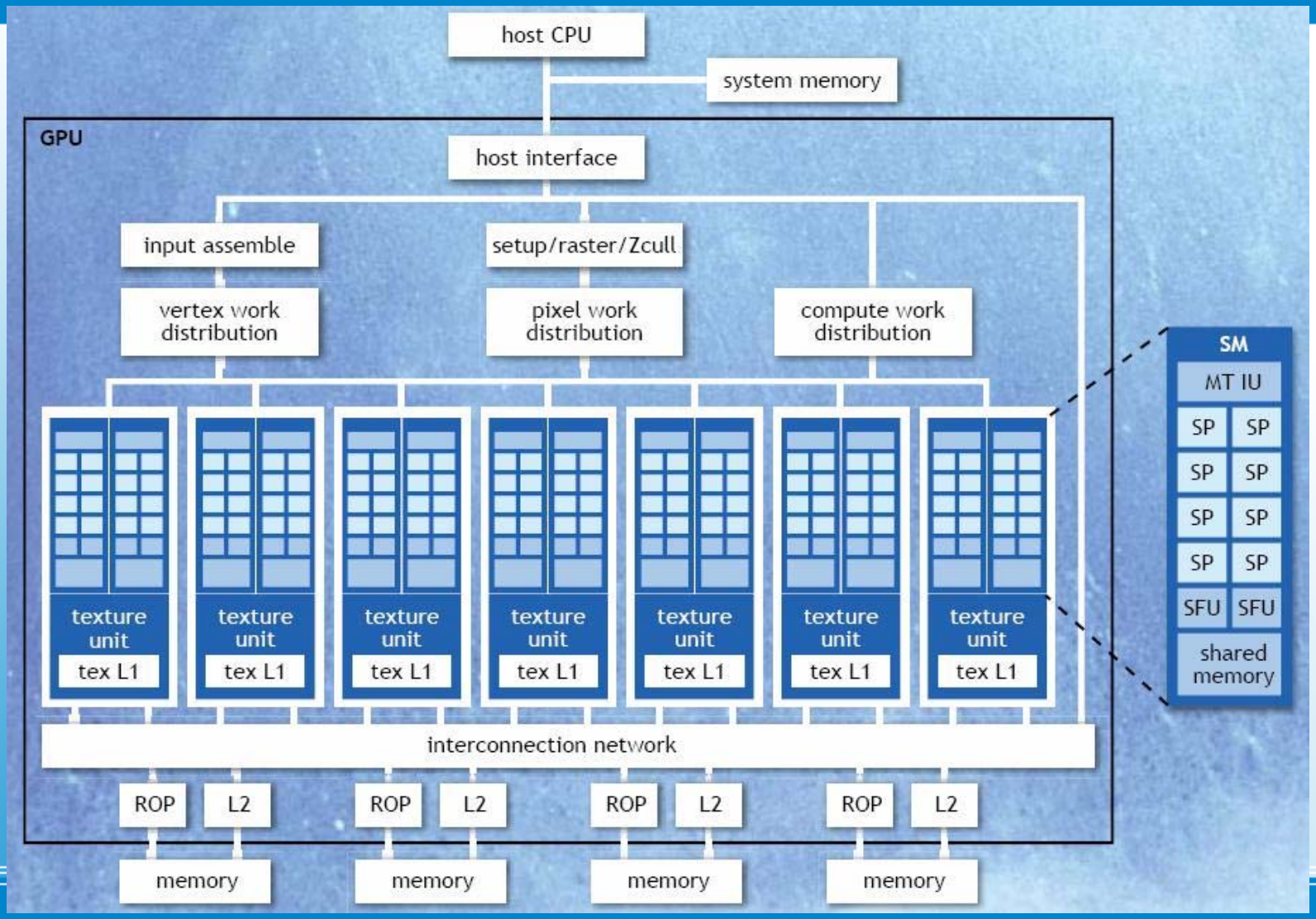
- Kevin Skadron
  - Associate Professor of Computer Science at the University of Virginia, but currently at sabbatical at NVIDIA Research
  - PhD in Computer Science from Princeton University

- GPU vs. CPU
  - GPU: Few instructions but very, very fast execution

    Uses very fast GDDR3 RAM
  - CPU: Lots of instructions, but slower execution

    Uses slower DDR2 or DDR3 RAM (but has direct access to more memory than GPUs)

Hardware Platform



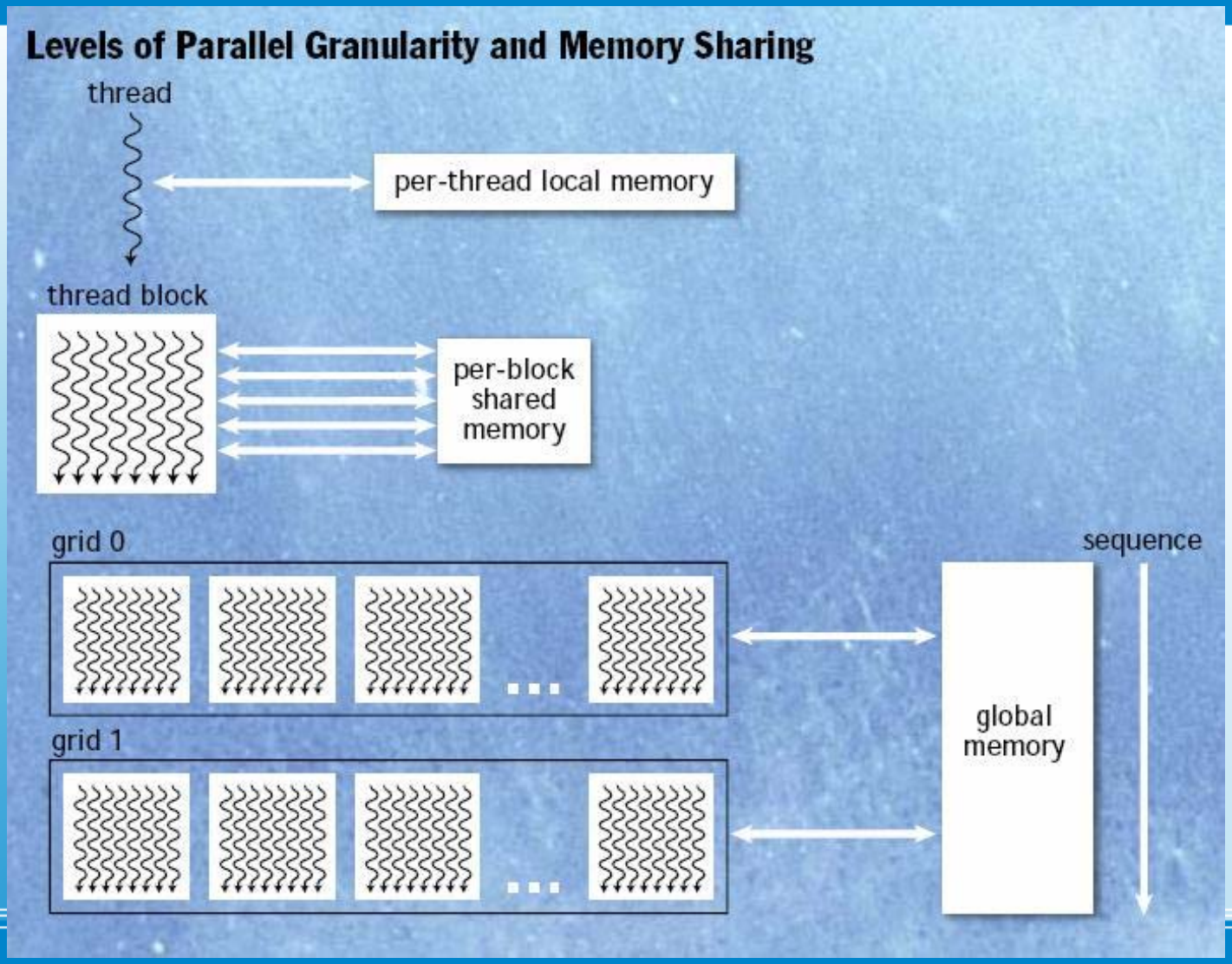
Hardware Platform

What is CUDA?

- CUDA is a minimal extension to C and C++ (like CILK, but not quite as easy)
- A serial program calls parallel *kernels* that may be a function or a full program
- Function type qualifiers
  - __device__, __global__, __host__
- Value type qualifiers
  - __device__, __constant__, __shared__

- *Kernels* execute over a set of parallel *threads*
- *Threads* are organized in a hierarchy of *grids* of thread *blocks*
- *Blocks* can have up to 3 dimensions and contain up to 512 *threads*
  - Threads in blocks can communicate
- *Grids* can also have up to 3 dimensions and $65{,}536^2$ *blocks*
  - No communication between blocks

What is CUDA?


Levels of Parallel Granularity and Memory Sharing
thread
per-thread local memory
thread block
per-block shared memory
grid 0
grid 1
sequence
global memory

What is CUDA

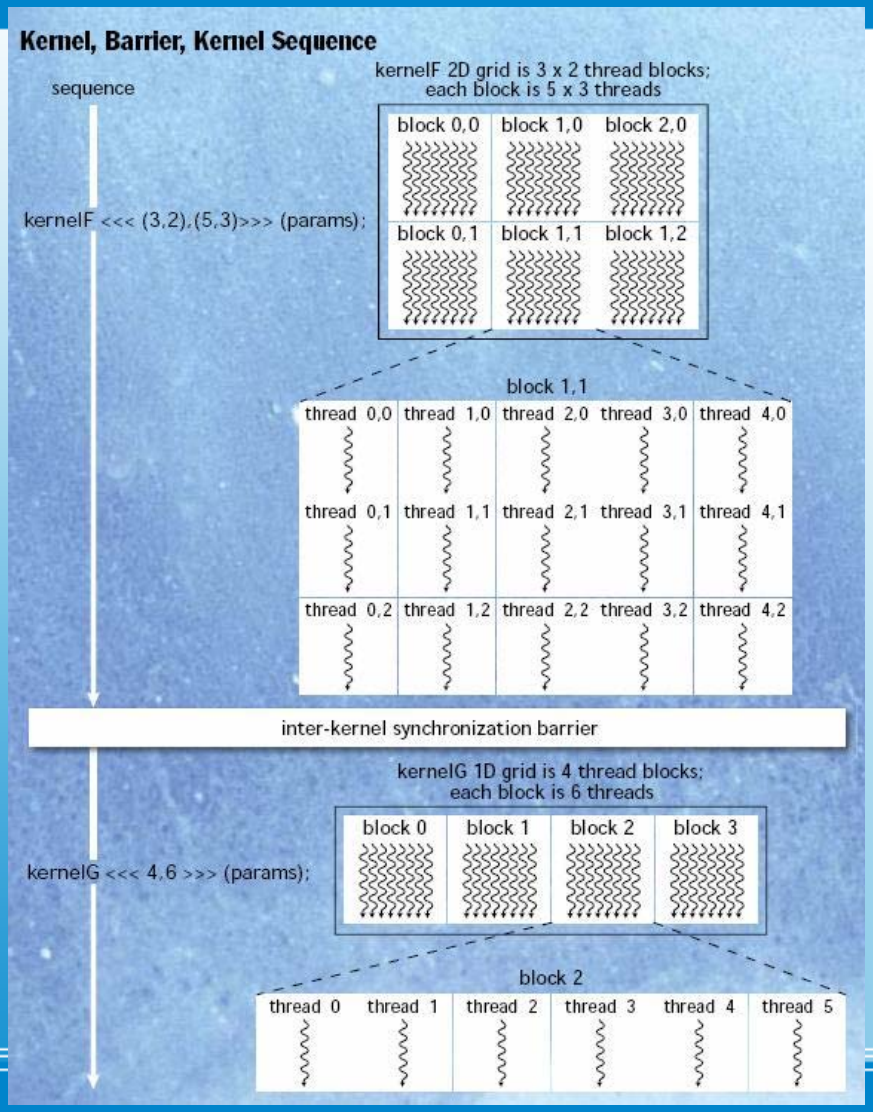
- Computing y <- ax + y with a Serial Loop

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

// Invoke serial SAXPY kernel
saxpy_serial(n, 2.0, x, y);
```

- Computing y <- ax + y in parallel using CUDA

```
__global__
void saxpy_parallel(int n, float
    alpha, float *x, float *y)
{
    int i = blockIdx.x*blockDim.x +
        threadIdx.x;
    if( i<n ) y[i] = alpha*x[i] + y[i];
}

// Invoke parallel SAXPY kernel
(256 threads per block)
int nblocks = (n + 255) / 256;
saxpy_parallel<<<nblocks,
    256>>>(n, 2.0, x, y);
```

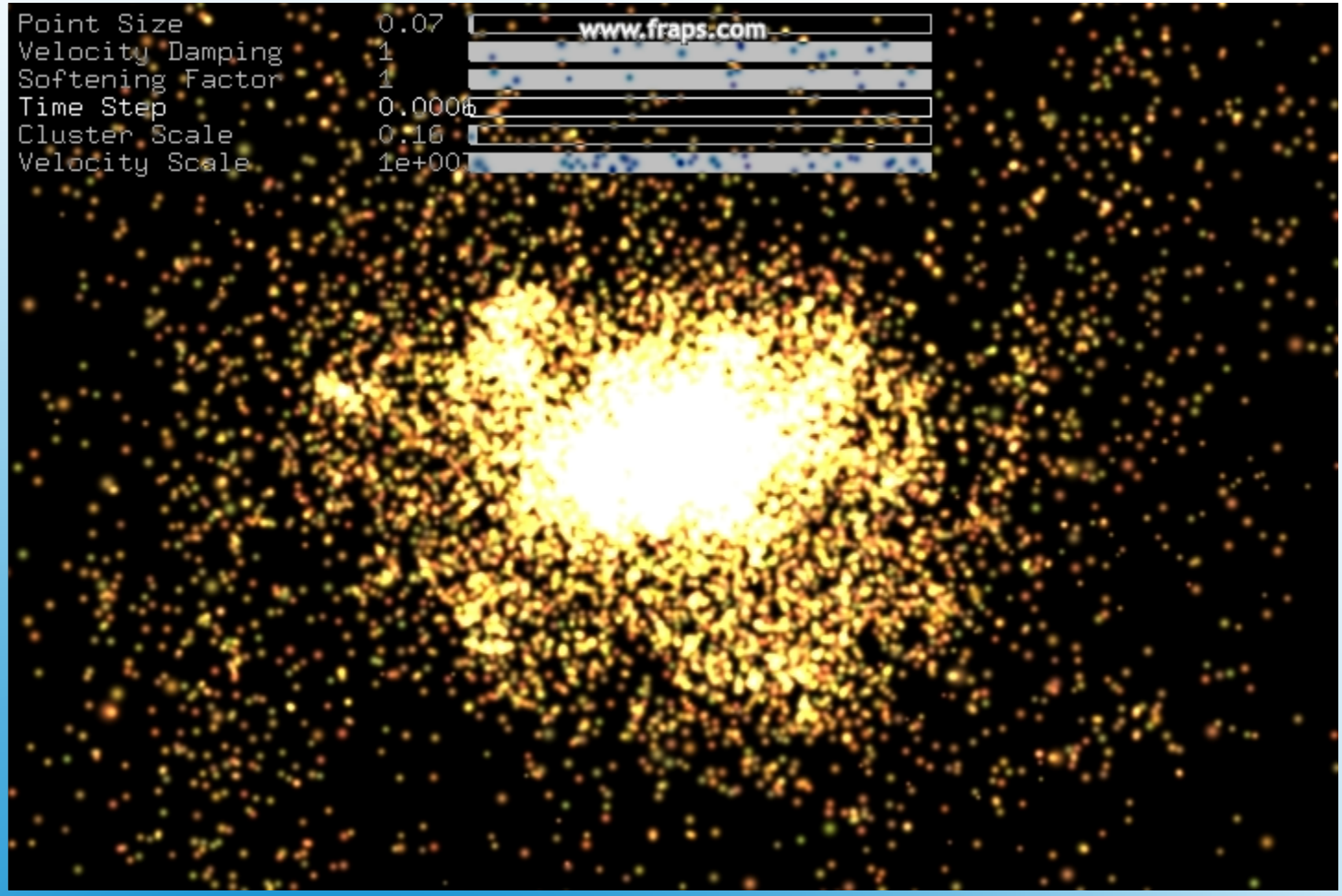
Programming in CUDA



Programming in CUDA

Other Applications

- Lots of different examples on nvidia.com
  - Examples are image analysis (e.g. facial recognition), MRI mapping, ray tracing, neural networks, and molecular dynamics simulation
  - Speed-ups from 1.3x (numerical weather prediction) to 250x (graphic-card cluster for astrophysics simulations)



N-Body Simulation

- OpenCL
- CTM
- RapidMind

GPGPU/MC Approaches

- Extremely high (and cheap) processing power
  - 8800GTS: 640 GFLOP/s
  - Core2Duo 2.66GHz: 17 GFLOP/s
  - Core2Quad 3GHz (3,500kr): 43 GFLOP/s
  - 2 x 8800GT(2,000kr): 1 TFLOP/s
  - 8600GTM: 30 GFLOP/s

Conclusion

- Is GPGPU taking over multi-core CPUs?
  - No (not yet, anyway)
- GPGPU programming has some problems
  - Only applicable to large applications (or so it seems)
  - When is it worth it to do it on the GPU?
  - Possible problems with optimization
  - Most programmers not used to working with GPUs
- Many rumors in the press on unified CPU and GPU in the future, but nothing confirmed yet.

Conclusion

- Nice article, well written
- Gives good insight into what CUDA is, but the hardware description is lacking
- Good sales speech, does not mention possible problems with CUDA

Presenters Opinion

- Thank you

All Done