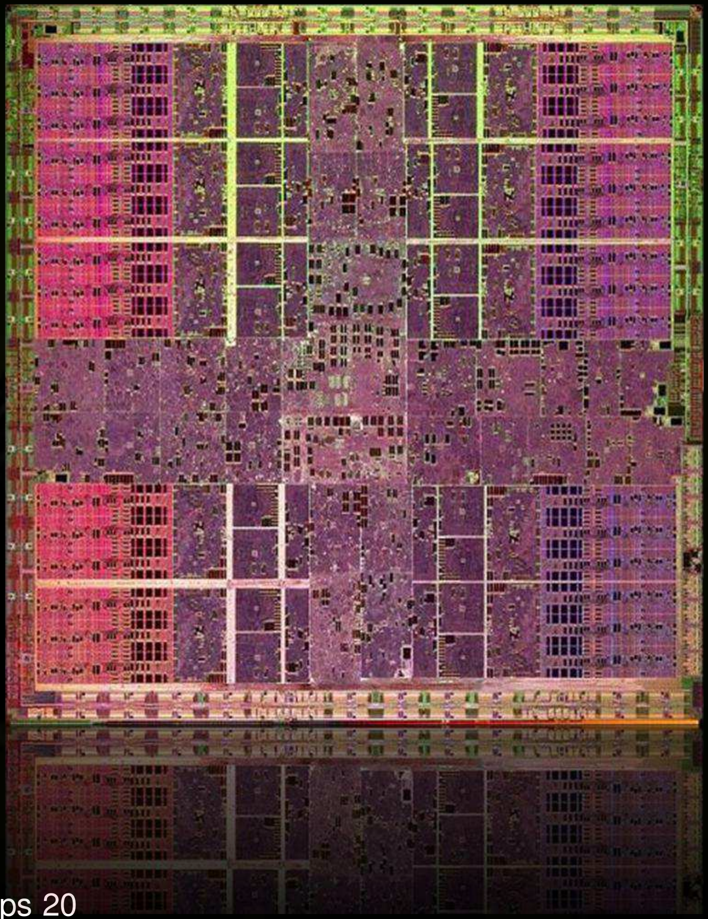
NVIDIA GTX200: TeraFLOPS Visual Computing

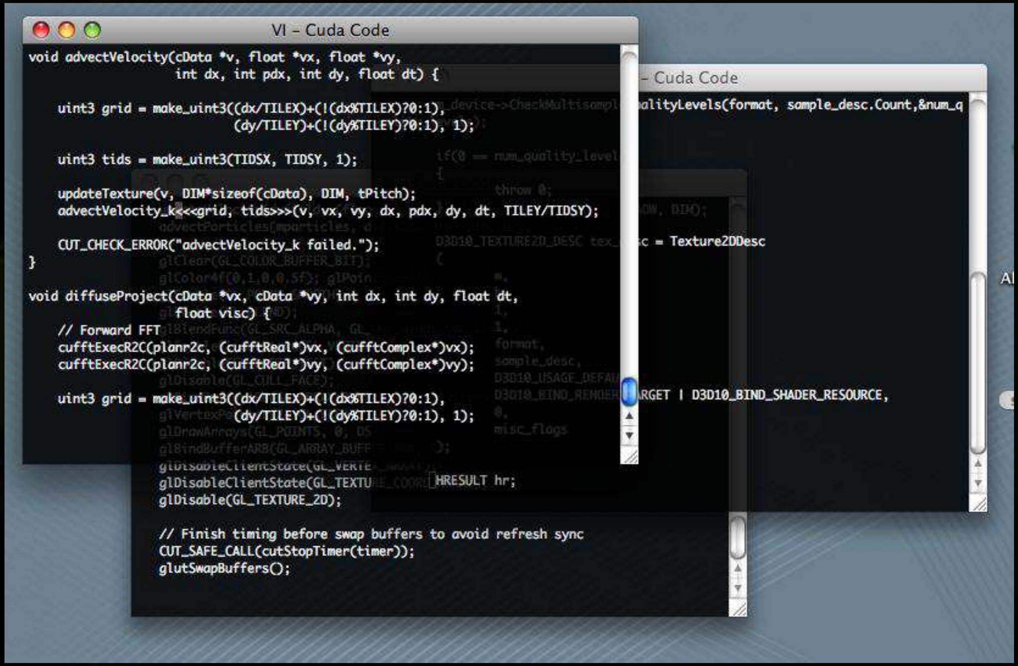
August 26, 2008
John Tynefield

Outline

- Execution Model
- Architecture
- Demo



Execution Model



```
void advectVelocity(cData *v, float *vx, float *vy,
                    int dx, int pdx, int dy, float dt) {

    uint3 grid = make_uint3((dx/TILEX)+(!(dx%TILEX)?0:1),
                            (dy/TILEY)+(!(dy%TILEY)?0:1), 1);

    uint3 tids = make_uint3(TIDSX, TIDSY, 1);

    updateTexture(v, DIM*sizeof(cData), DIM, tPitch);
    advectVelocity_k<<<grid, tids>>>(v, vx, vy, dx, pdx, dy, dt, TILEY/TIDSY);

    CUT_CHECK_ERROR("advectVelocity_k failed.");
}

void diffuseProject(cData *vx, cData *vy, int dx, int dy, float dt,
                    float visc) {
    // Forward FFT
    cufftExecR2C(planr2c, (cufftReal*)vx, (cufftComplex*)vx);
    cufftExecR2C(planr2c, (cufftReal*)vy, (cufftComplex*)vy);

    uint3 grid = make_uint3((dx/TILEX)+(!(dx%TILEX)?0:1),
                            (dy/TILEY)+(!(dy%TILEY)?0:1), 1);
```

```
    glDisableClientState(GL_TEXTU
    glDisable(GL_TEXTURE_2D);

    // Finish timing before swap buffers to avoid refresh sync
    CUT_SAFE_CALL(cutStopTimer(timer));
    glutSwapBuffers();
```

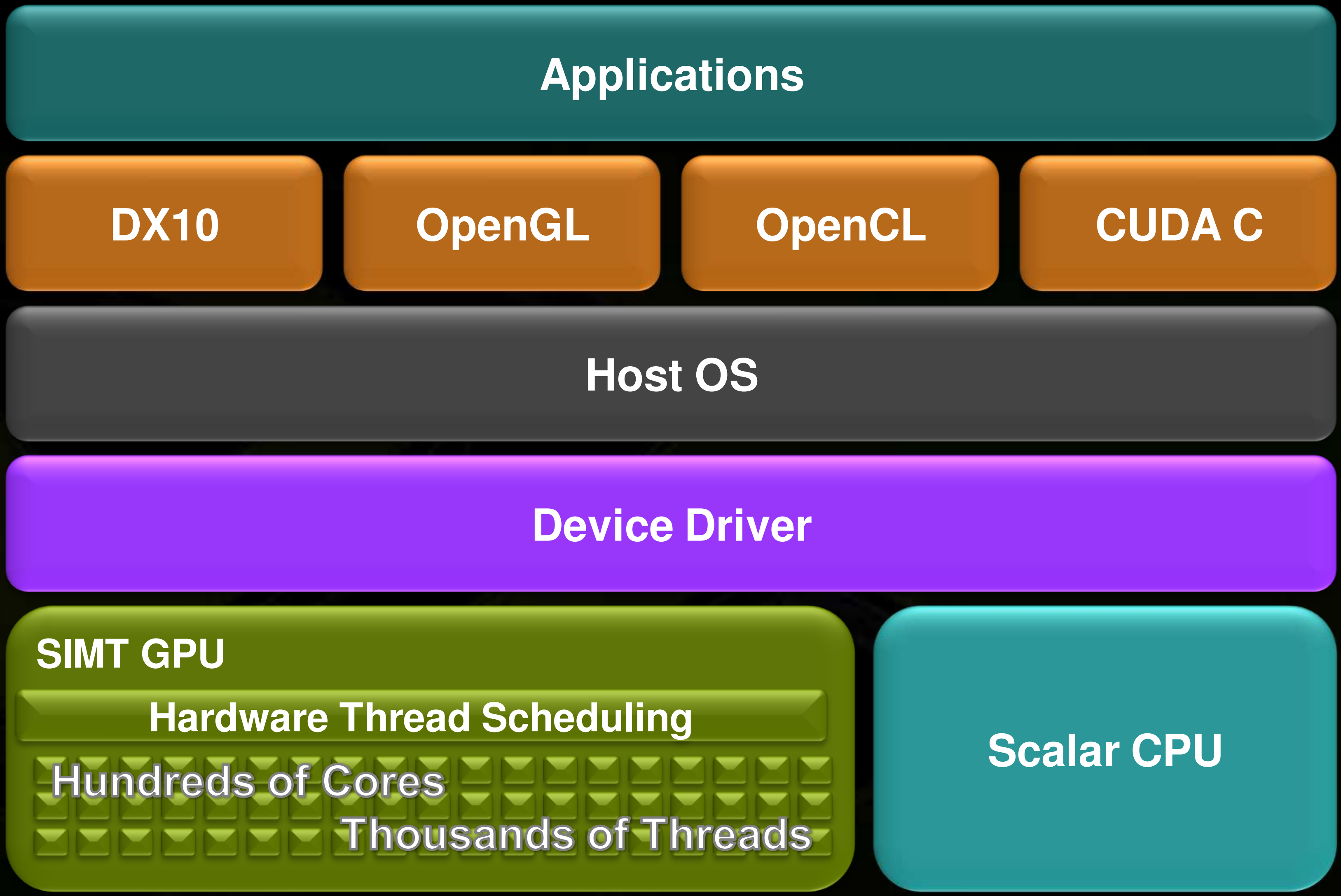
```
alityLevels(format, sample_desc.Count,&num_q

c = Texture2DDesc

RGET | D3D10_BIND_SHADER_RESOURCE,

HRESULT hr;
```

Software Architecture



Applications

DX10

OpenGL

OpenCL

CUDA C

Host OS

Device Driver

SIMT GPU

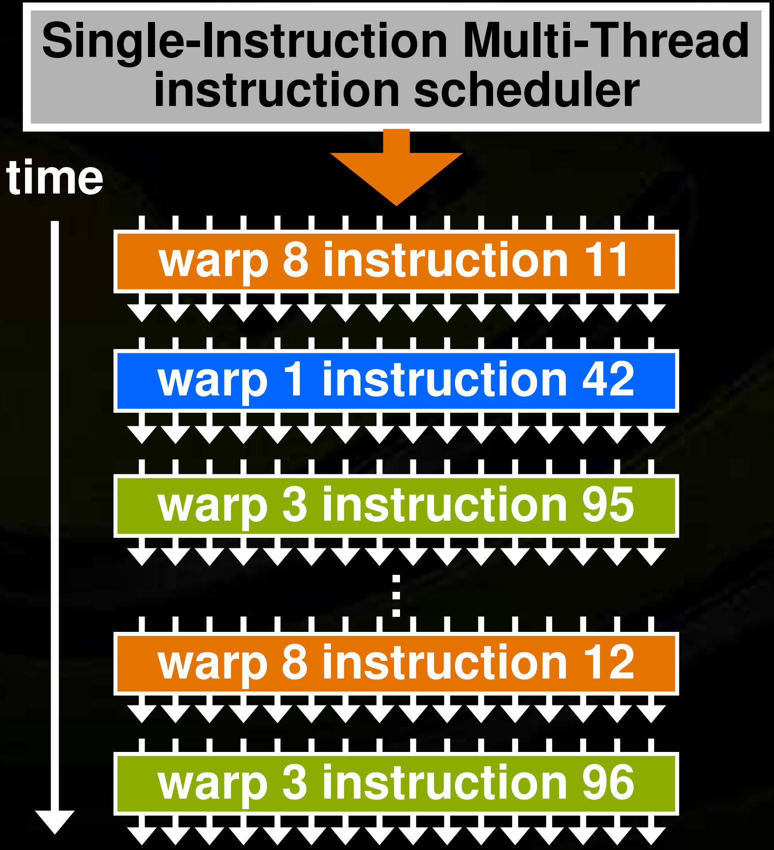
Hardware Thread Scheduling

Hundreds of Cores

Thousands of Threads

Scalar CPU

SIMT Multithreaded Execution



Single-Instruction Multi-Thread instruction scheduler

time

warp 8 instruction 11

warp 1 instruction 42

warp 3 instruction 95

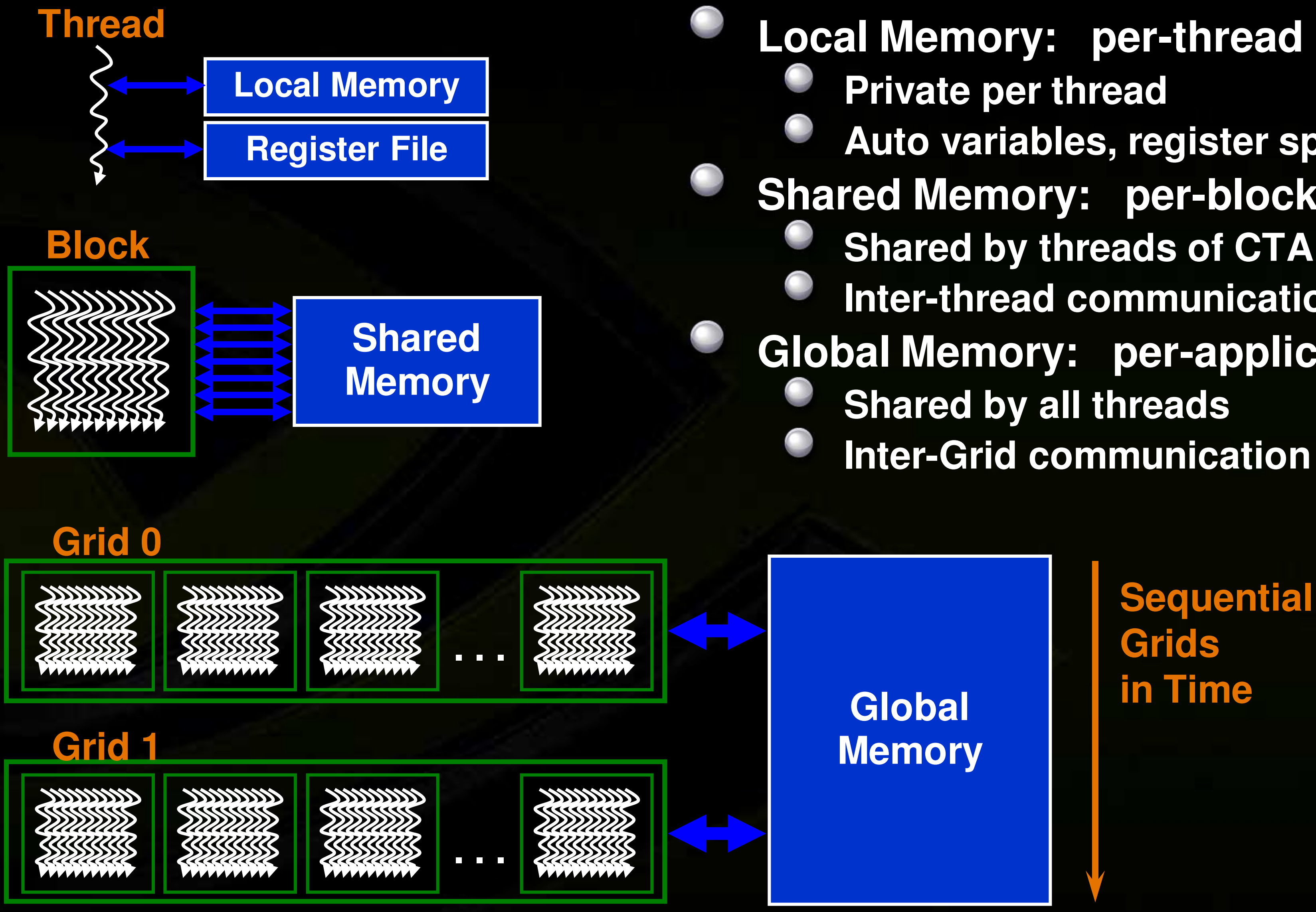
⋮

warp 8 instruction 12

warp 3 instruction 96

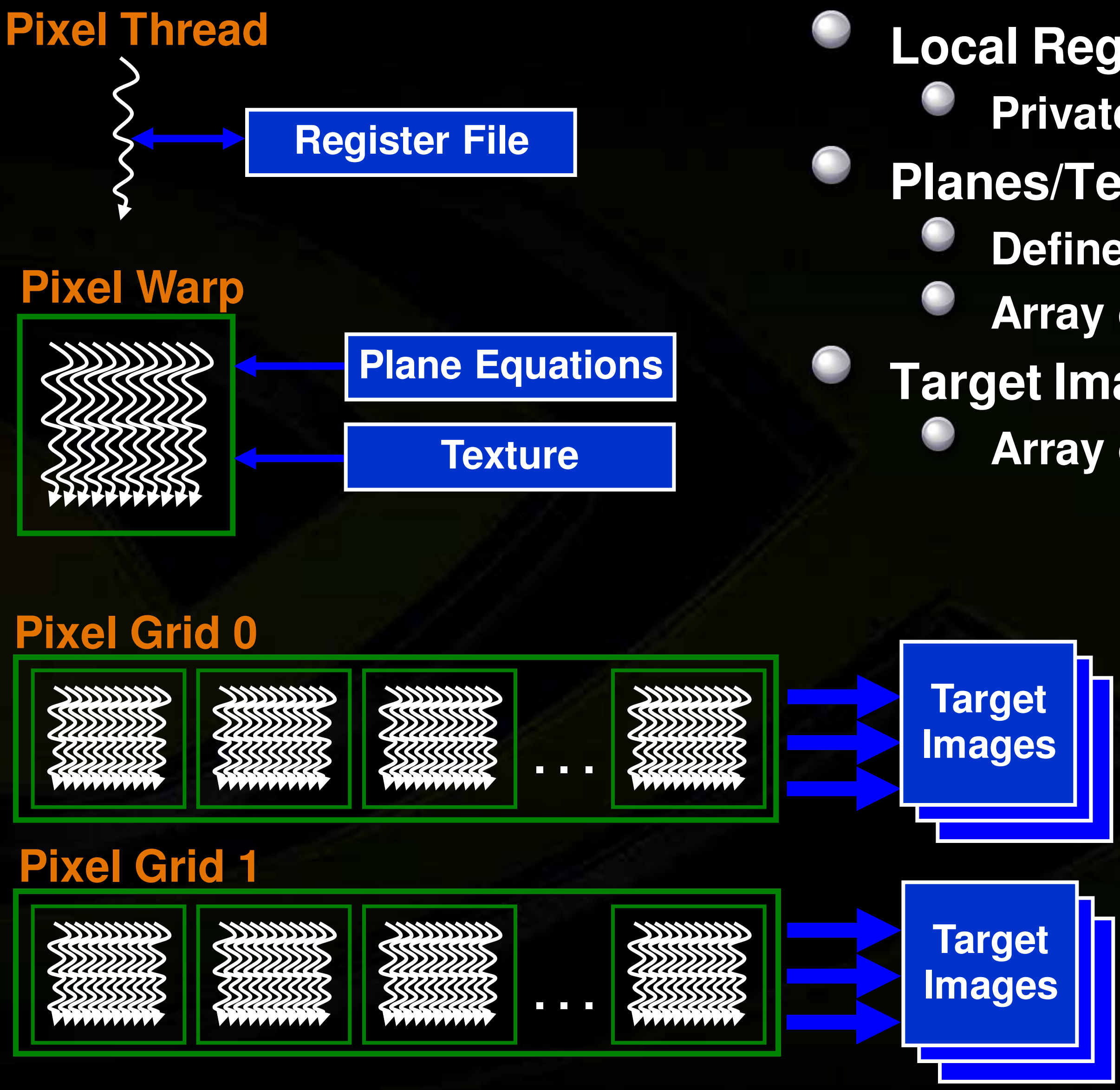
- SIMT: Single-Instruction Multi-Thread applies instruction to independent threads
- SIMT provides easy single-thread scalar programming with SIMD efficiency
- Warp: the set of 32 parallel threads that execute a SIMT instruction
- Hardware implements zero-overhead warp and thread scheduling
- Each thread processor manages state for up to 128 threads
- SIMT threads execute independently
- SIMT warp diverges and converges when threads branch independently
- Best efficiency and performance when threads of a warp execute together

Execution Model - CUDA



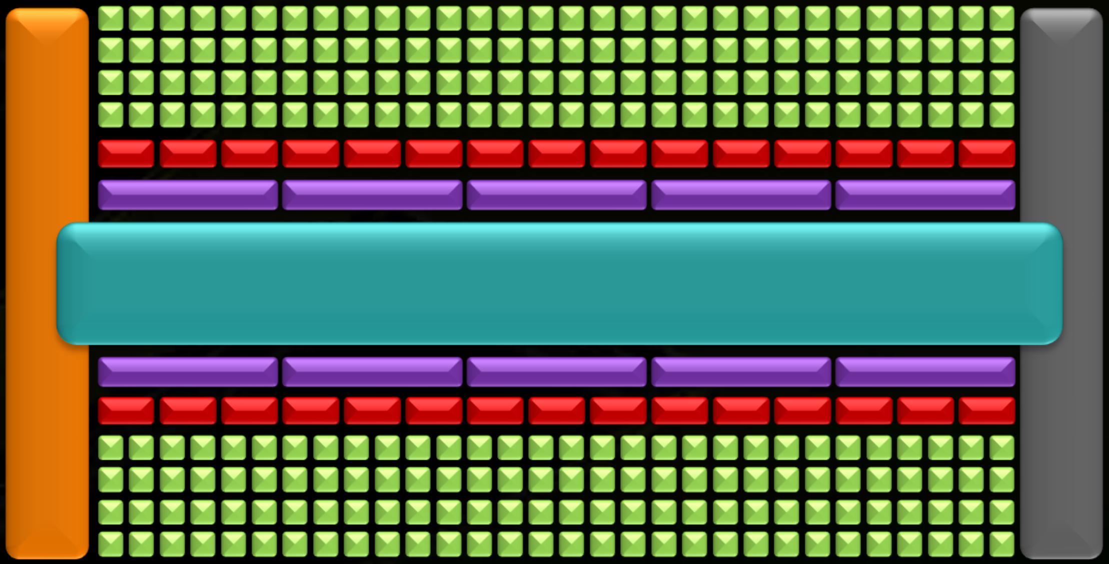
- Local Memory: per-thread
  - Private per thread
  - Auto variables, register spill
- Shared Memory: per-block
  - Shared by threads of CTA
  - Inter-thread communication
- Global Memory: per-application
  - Shared by all threads
  - Inter-Grid communication

Execution Model - Graphics



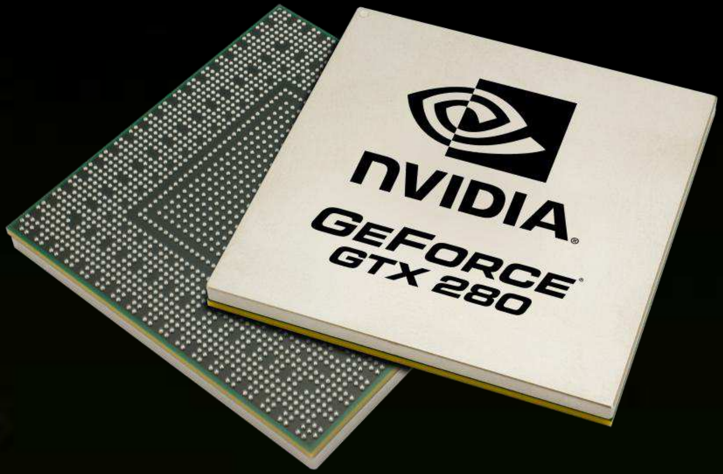
- Local Registers: per Pixel
  - Private per pixel
- Planes/Textures: per Warp
  - Define surface-space inputs
  - Array of 1D/2D/3D data arrays
- Target Images: per Grid
  - Array of 2D surfaces

Architecture



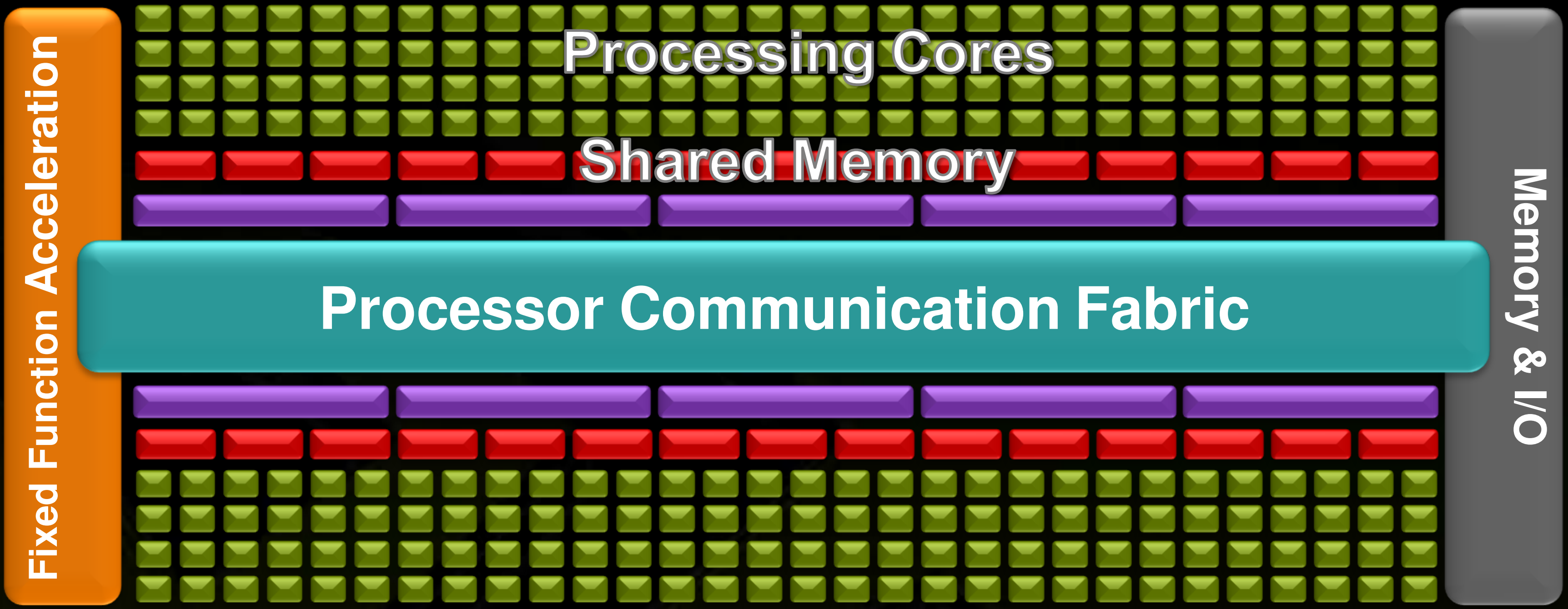
Implementation

- TSMC 65nm
- 1.4 Billion Transistors
- 24.3 x 24.0 mm die
- 2236 Ball BGA



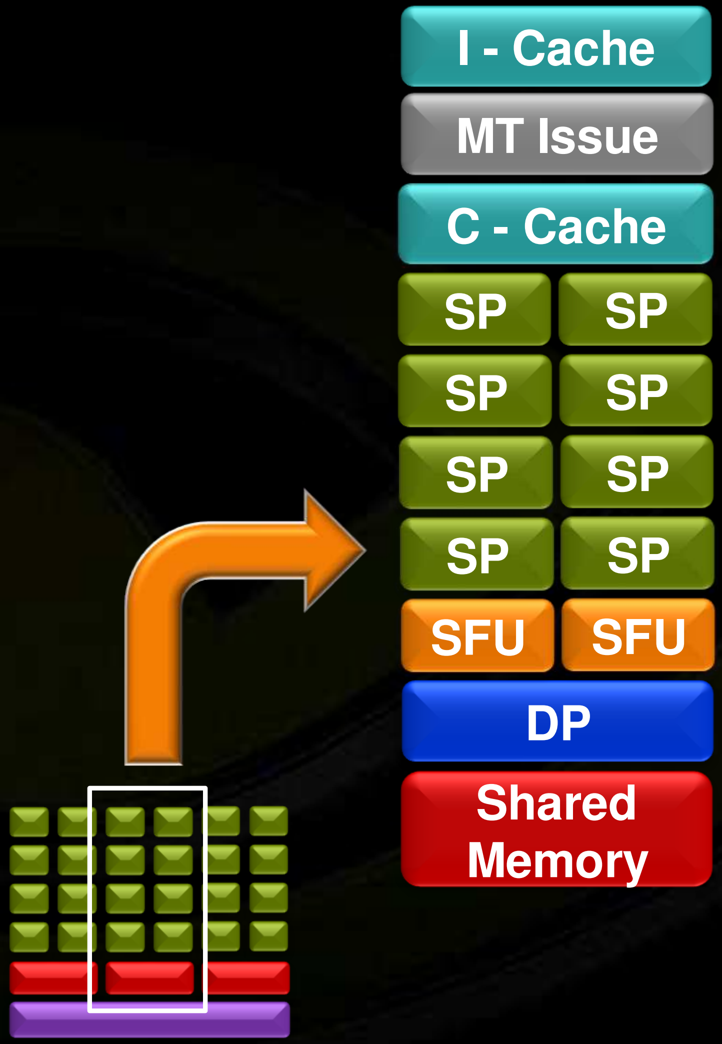
- 1 TeraFLOPS SP / 84 GigaFLOPS DP
  - 1.4 GHz Processor Clock
- 140 GB/s
  - 1.1 GHz Memory Clock
- Up to 4GB on-board Memory

GTX200 Unified Visual Computing



- Tesla Unified Graphics and Computing Architecture
- Scales parallel performance 2X beyond G80
- 240 🟩 Thread Processor cores, 30K threads
- Double precision 64-bit IEEE 754 floating point

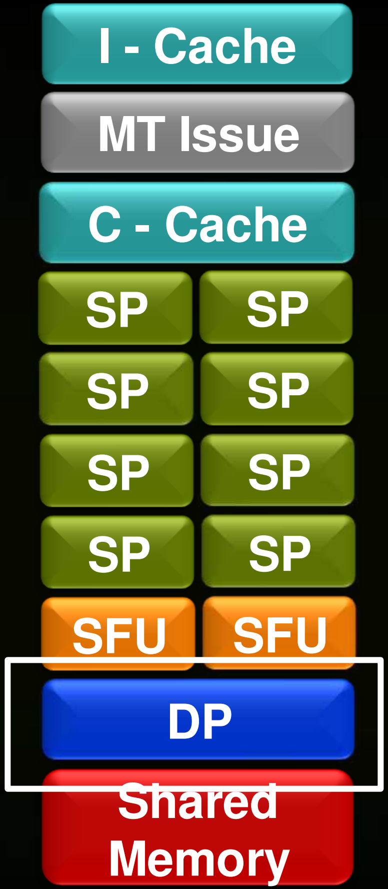
SM Multithreaded Multiprocessor



- 8 SP Thread Processors
  - IEEE 754 32-bit floating point
  - 32-bit and 64-bit integer
  - 2K 32-bit registers per SP
  - Variable 4-128 registers / thread
- 2 SFU Special Function Units
- 1 DP Double Precision Unit
  - IEEE 754R 64-bit floating point
  - Fused multiply-add
- Scalar register-based ISA
- Multithreaded Instruction Unit
  - 1024 threads, hardware multithreaded
  - 32 SIMT warps of 32 threads
  - Independent thread execution
  - Mixed concurrent thread types
- 16KB Shared Memory
  - Concurrent threads share data
  - Low latency load/store

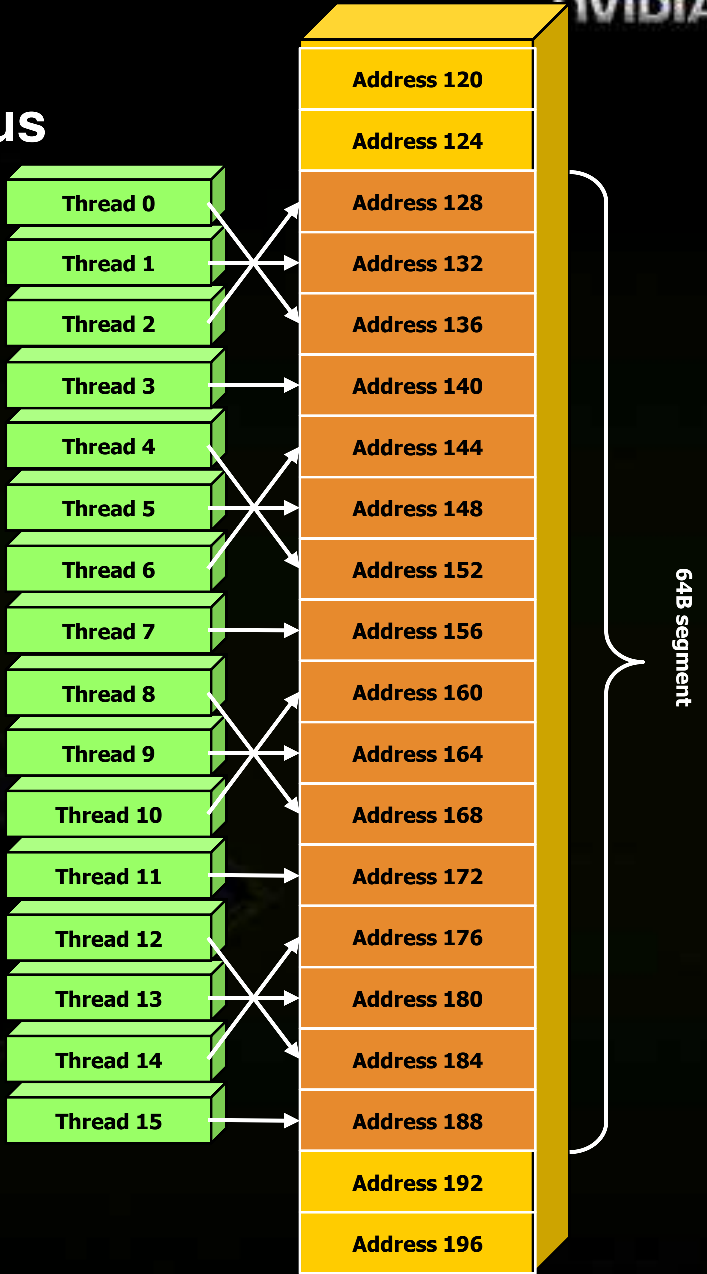
Double Precision Fused Multiply-Add

- Revised IEEE 754R standard specifies FMA
- FMA has lower latency than FMAD
- FMA improves accuracy of many computations
    - Dot products
    - Matrix multiplication
- Enables fast mixed-precision convergence algorithms
- Enables higher performance algorithms for extended-precision arithmetic
- Enables efficient implementation of exactly-rounded division and square root

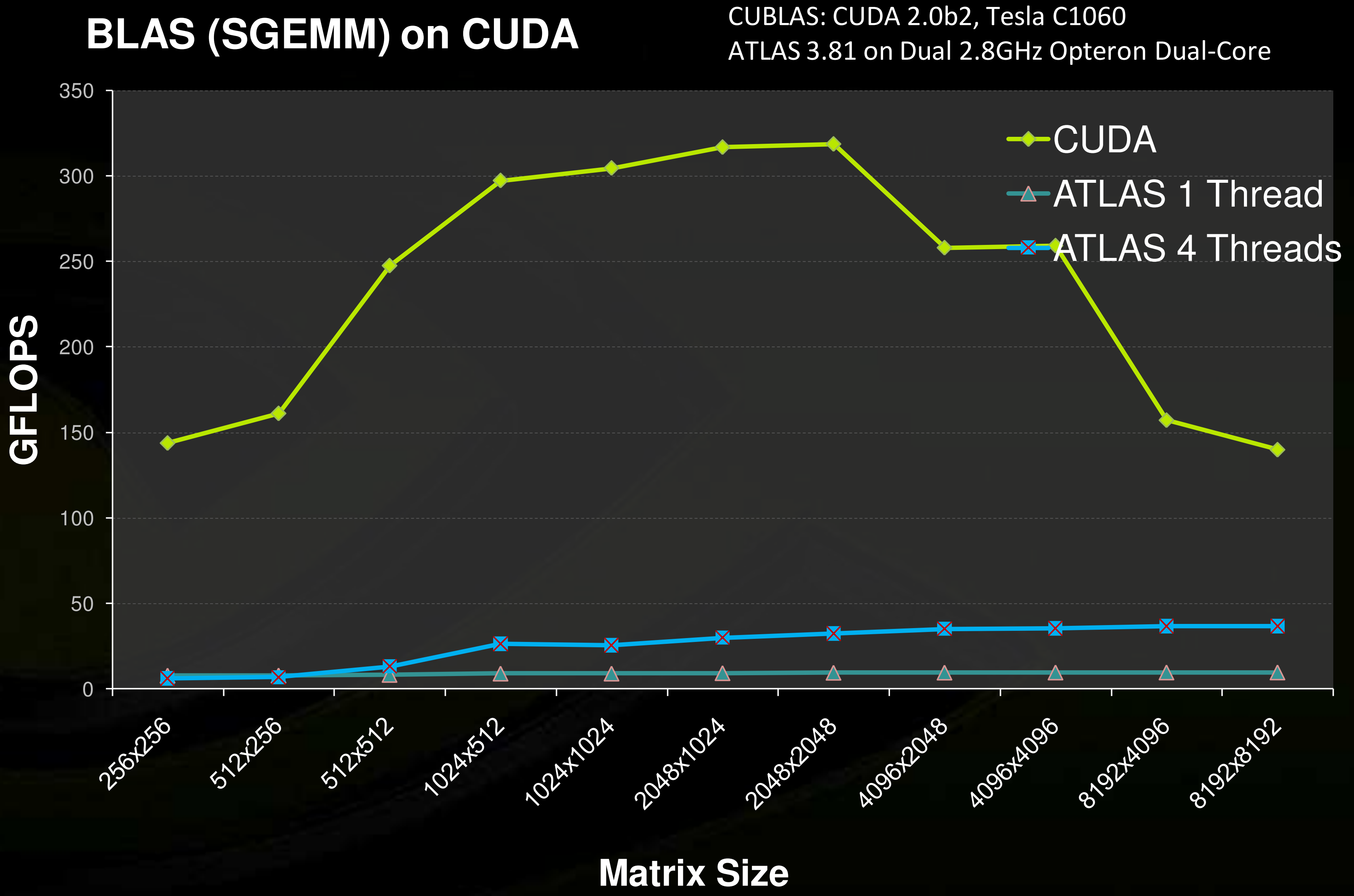


Coalesced Memory I/O

- When 16 threads access a contiguous region of device memory
- 16 data elements loaded in one instruction
  - int, float: 64 bytes (fastest)
  - int2, float2: 128 bytes
  - int4, float4: 256 bytes (2 transactions)
- Regions aligned to multiple of size
- Coalescing scales gracefully for partially contiguous accesses



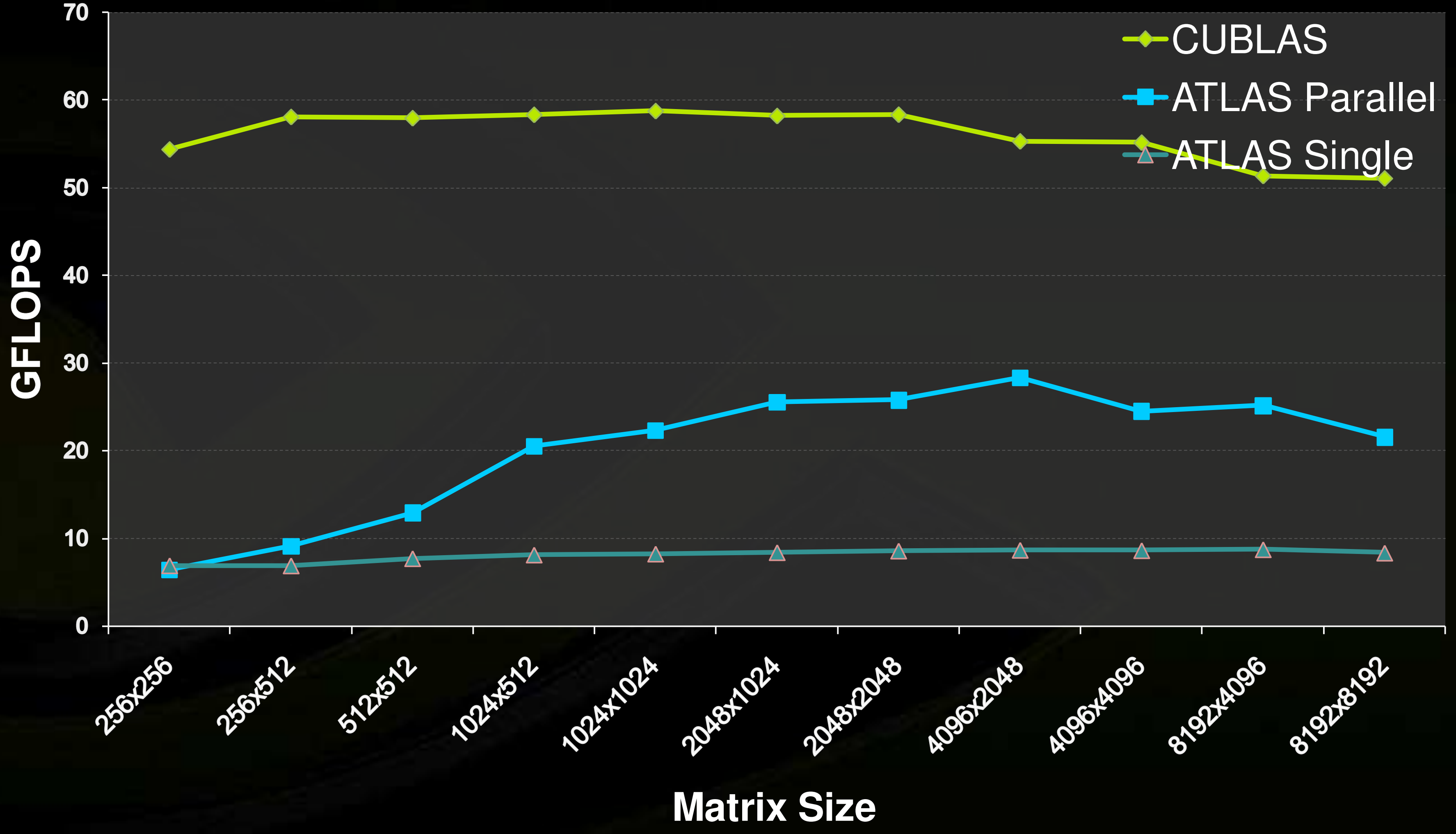
Performance: Single Precision BLAS



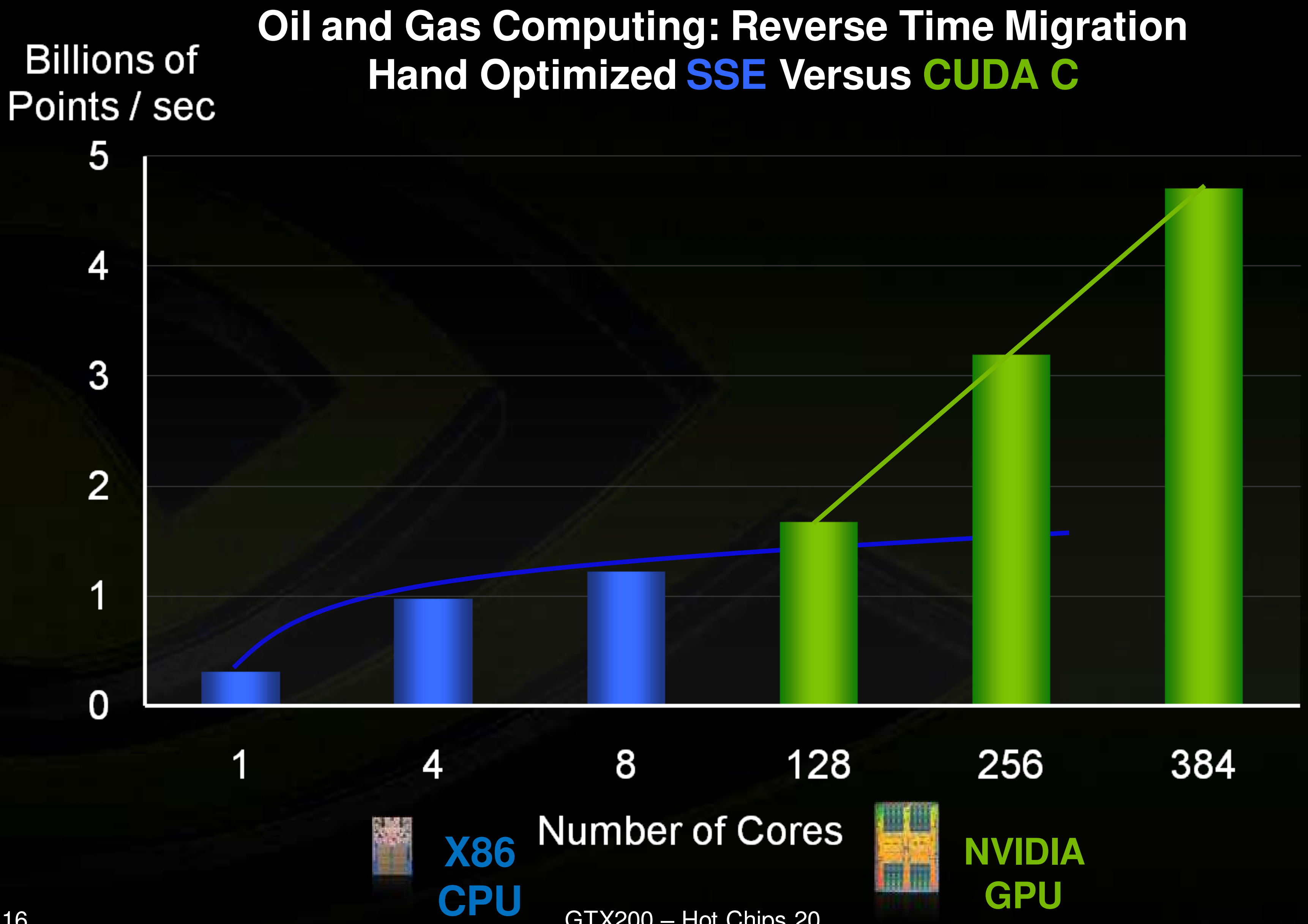
Performance: Double Precision BLAS

BLAS (DGEMM) on CUDA

CUBLAS CUDA 2.0b2 on Tesla C1060
ATLAS 3.81 on Intel Xeon E5440 Quad-core, 2.83 GHz



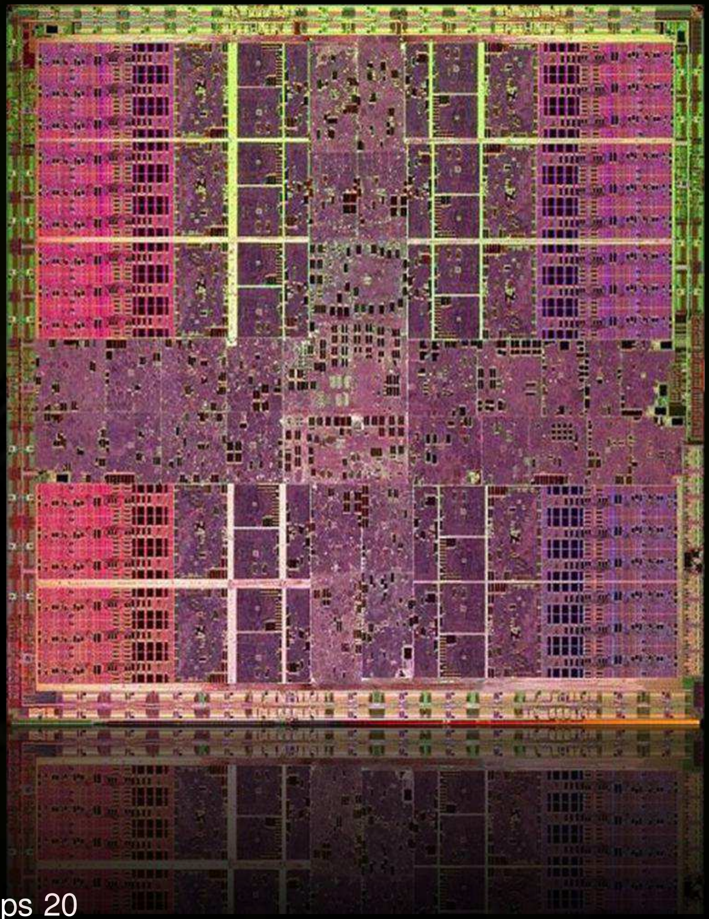
Performance: Scaling



Summary

- Rebalanced architecture to workload trends
- Scaled from 128 to 240 processors
- Hardware manages thousands of threads
  - Zero software overhead
  - Hides huge latencies
  - High achieved utilization
- Natively Scalar
  - No swizzling or vectorization overhead
  - Coalescing for high bandwidth memory I/O
- Software architecture allows 2X scaling on customer C code with no modification

Demo



More Information

Erik Lindholm, et. al., “NVIDIA Tesla: A Unified Graphics and Computing Architecture,” *IEEE Micro*, March-April 2008.

http://www.nvidia.com/cuda