

NVIDIA

GPU Parallel Computing Architecture and CUDA Programming Model

John Nickolls

Hot Chips 2007: NVIDIA GPU Parallel Computing Architecture

© NVIDIA Corporation 2007

Outline



- Why GPU Computing?
- GPU Computing Architecture
- Multithreading and Thread Arrays
- Data Parallel Problem Decomposition
- Parallel Memory Sharing
- Transparent Scalability
- CUDA Programming Model
- CUDA: C on the GPU
- CUDA Example
- Applications
- Summary

Parallel Computing on a GPU



- NVIDIA GPU Computing Architecture is a scalable parallel computing platform
- In laptops, desktops, workstations, servers
- 8-series GPUs deliver 50 to 200 GFLOPS on compiled parallel C applications
- GPU parallel performance pulled by the insatiable demands of PC game market
- GPU parallelism is doubling every year
- Programming model scales transparently
- Programmable in C with CUDA tools
- Multithreaded SPMD model uses application data parallelism and thread parallelism



GeForce 8800




Tesla D870

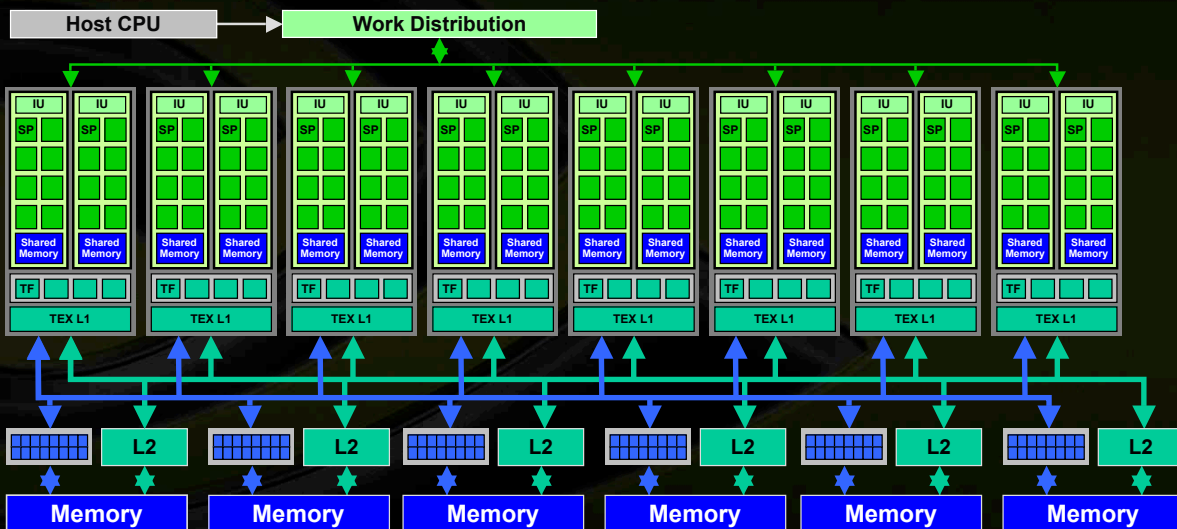


Tesla S870

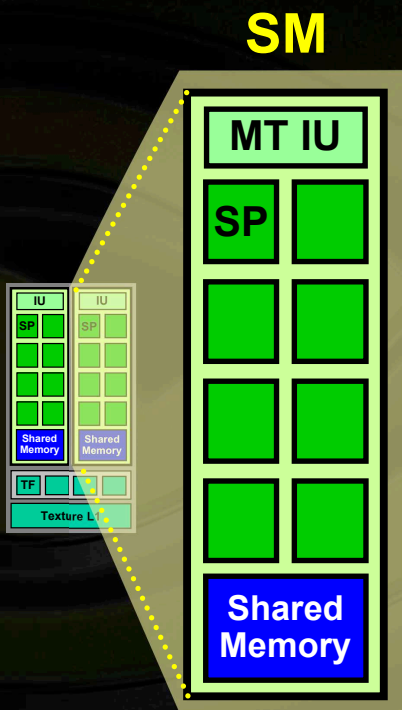
NVIDIA 8-Series GPU Computing



- Massively multithreaded parallel computing platform
- 12,288 concurrent threads, hardware managed
- 128  Thread Processor cores at 1.35 GHz == 518 GFLOPS peak
- GPU Computing features enable C on Graphics Processing Unit

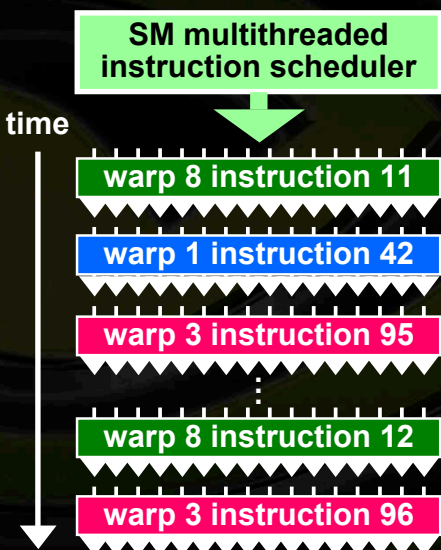


SM Multithreaded Multiprocessor



- SM has 8 SP Thread Processors
 - 32 GFLOPS peak at 1.35 GHz
 - IEEE 754 32-bit floating point
 - 32-bit integer
- Scalar ISA
 - Memory load/store
 - Texture fetch
 - Branch, call, return
 - Barrier synchronization instruction
- Multithreaded Instruction Unit
 - 768 Threads, hardware multithreaded
 - 24 SIMD warps of 32 threads
 - Independent MIMD thread execution
 - Hardware thread scheduling
- 16KB Shared Memory
 - Concurrent threads share data
 - Low latency load/store

SM SIMD Multithreaded Execution



- Weaving: the original parallel thread technology is about 10,000 years old
- **Warp**: the set of 32 parallel threads that execute a SIMD instruction
- SM hardware implements zero-overhead warp and thread scheduling
- Each SM executes up to 768 concurrent threads, as 24 SIMD warps of 32 threads
- Threads can execute independently
- SIMD warp diverges and converges when threads branch independently
- Best efficiency and performance when threads of a warp execute together
- SIMD across threads (not just data) gives easy single-thread scalar programming with SIMD efficiency

Programmer Partitions Problem with Data-Parallel Decomposition

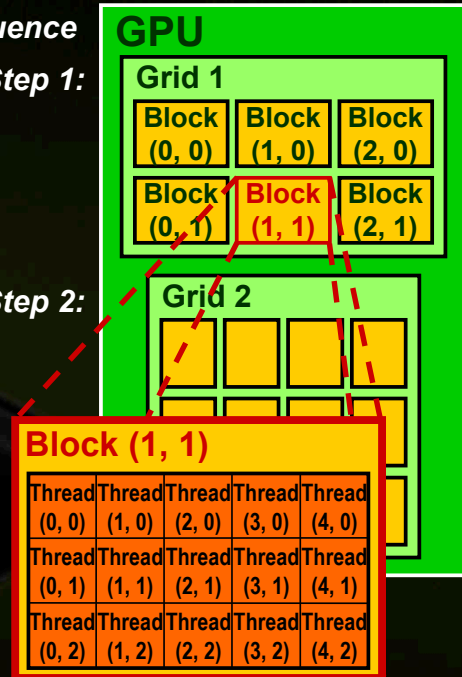


- CUDA Programmer partitions problem into Grids, one Grid per sequential problem step
- Programmer partitions Grid into result Blocks computed independently in parallel
- GPU thread array computes result Block
- Programmer partitions Block into elements computed cooperatively in parallel
- GPU thread computes result element

Sequence

Step 1:

Step 2:



Cooperative Thread Array CTA Implements CUDA Thread Block

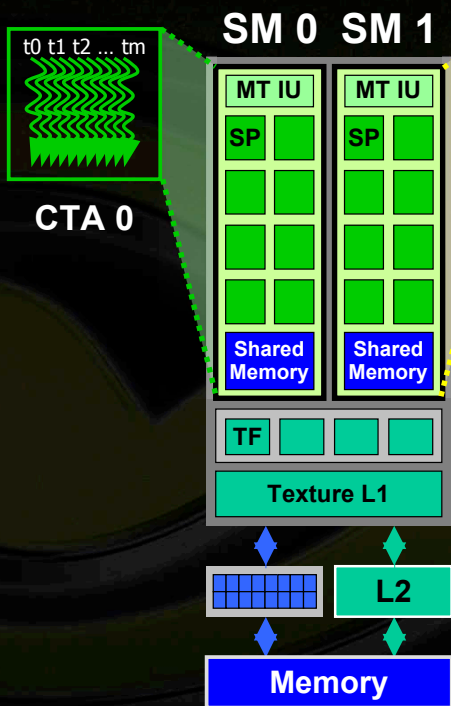


- A **CTA** is an array of concurrent threads that cooperate to compute a result
- A CUDA **thread block** is a CTA
- Programmer declares CTA:
 - CTA size 1 to 512 concurrent threads
 - CTA shape 1D, 2D, or 3D
 - CTA dimensions in threads
- CTA threads execute thread program
- CTA threads have thread id numbers
- CTA threads share data and synchronize
- Thread program uses thread id to select work and address shared data

CTA
CUDA Thread Block



SM Multiprocessor Executes CTAs



- CTA threads run concurrently
- SM assigns thread id #s
- SM manages thread execution
- CTA threads share data & results
 - In Memory and Shared Memory
 - Synchronize at barrier instruction
- Per-CTA Shared Memory
 - Keeps data close to processor
 - Minimize trips to global Memory
- CTA threads access global Memory
 - 76 GB/sec GDDR DRAM

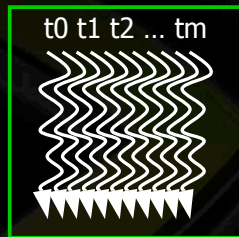
Data Parallel Levels



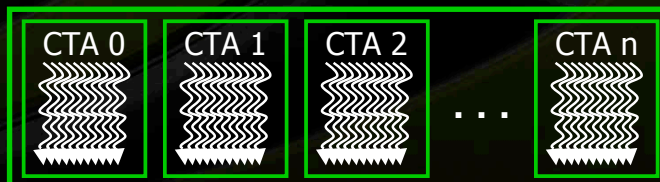
Thread



CTA

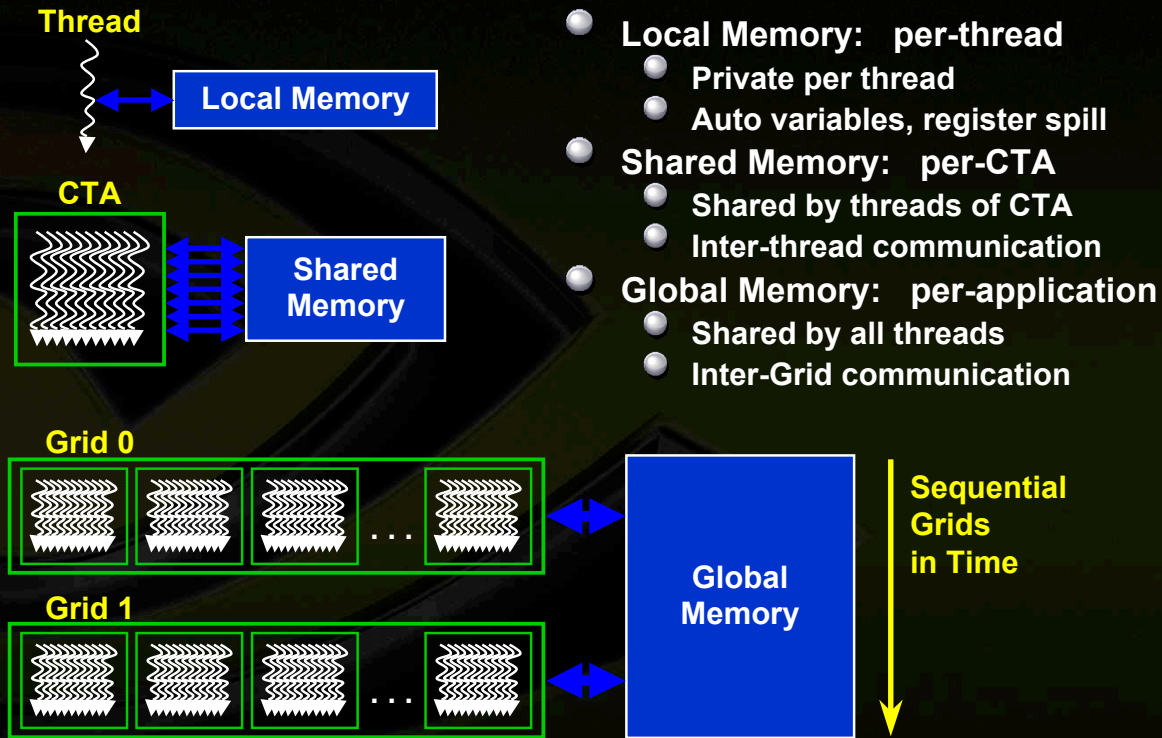


Grid



- Thread
 - Computes result elements
 - Thread id number
- CTA – Cooperative Thread Array
 - Computes result Block
 - 1 to 512 threads per CTA
 - CTA (Block) id number
- Grid of CTAs
 - Computes many result Blocks
 - 1 to many CTAs per Grid
- Sequential Grids
 - Compute sequential problem steps

Parallel Memory Sharing



11

Hot Chips 2007: NVIDIA GPU Parallel Computing Architecture

© NVIDIA Corporation 2007

How to Scale GPU Computing?



- GPU parallelism varies widely
 - Ranges from 8 cores to many 100s of cores
 - Ranges from 100 to many 1000s of threads
 - GPU parallelism doubles yearly
- Graphics performance scales with GPU parallelism
 - Data parallel mapping of pixels to threads
 - Unlimited demand for parallel pixel shader threads and cores

Challenge:

- Scale **Computing** performance with GPU parallelism
 - Program must be insensitive to the number of cores
 - Write one program for any number of SM cores
 - Program runs on any size GPU without recompiling

12

Hot Chips 2007: NVIDIA GPU Parallel Computing Architecture

© NVIDIA Corporation 2007

Transparent Scalability



- Programmer uses multi-level data parallel decomposition
 - Decomposes problem into sequential steps (**Grids**)
 - Decomposes Grid into computing parallel Blocks (**CTAs**)
 - Decomposes Block into computing parallel elements (**threads**)
- GPU hardware distributes CTA work to available SM cores
 - GPU balances CTA work load across any number of SM cores
 - SM core executes CTA program that computes Block
- CTA program computes a Block independently of others
 - Enables parallel computing of Blocks of a Grid
 - No communication among Blocks of same Grid
 - Scales one program across any number of parallel SM cores
- Programmer writes one program for all GPU sizes
- Program does not know how many cores it uses
- Program executes on GPU with any number of cores

13

Hot Chips 2007: NVIDIA GPU Parallel Computing Architecture

© NVIDIA Corporation 2007

CUDA Programming Model: Parallel Multithreaded Kernels



- Execute data-parallel portions of application on GPU as **kernels** which run in parallel on many cooperative threads
- Integrated CPU + GPU application C program
 - Partition problem into a sequence of kernels
 - Kernel C code executes on GPU
 - Serial C code executes on CPU
 - Kernels execute as blocks of parallel threads
- View GPU as a computing device that:
 - Acts as a coprocessor to the CPU host
 - Has its own memory
 - Runs many lightweight threads in parallel



14

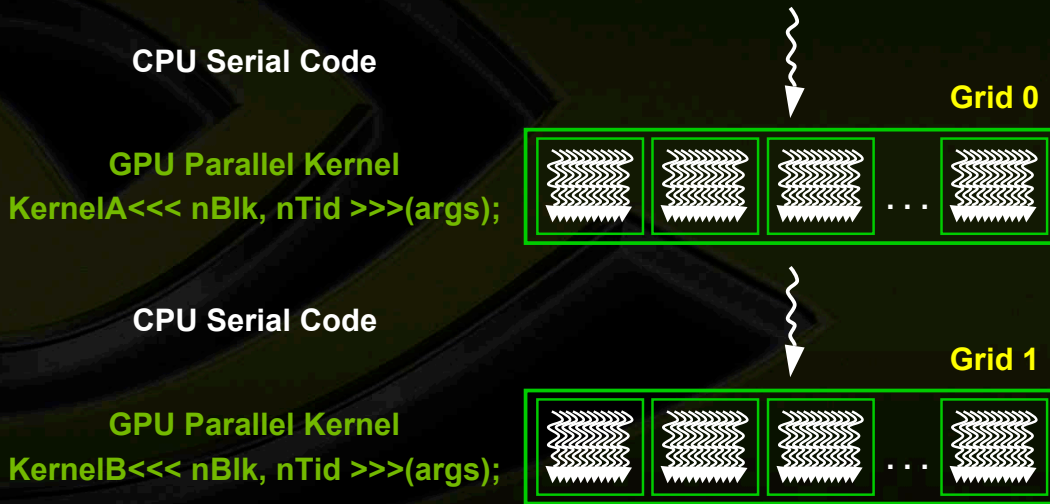
Hot Chips 2007: NVIDIA GPU Parallel Computing Architecture

© NVIDIA Corporation 2007

Single-Program Multiple-Data (SPMD)



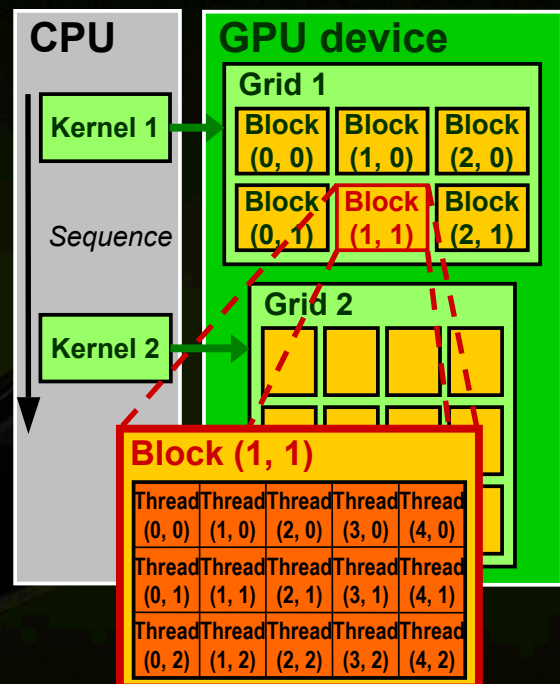
- CUDA integrated CPU + GPU application C program
 - Serial C code executes on CPU
 - Parallel Kernel C code executes on GPU thread blocks



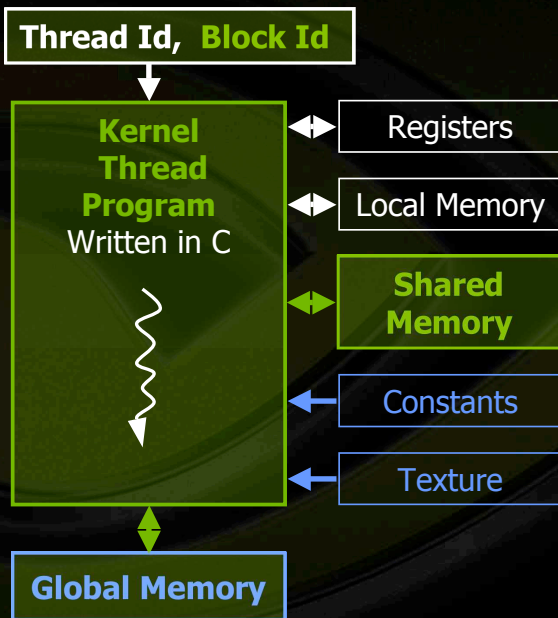
CUDA Programming Model: Grids, Blocks, and Threads



- Execute a sequence of **kernels** on GPU computing device
- A kernel executes as a **Grid** of thread blocks
- A **thread block** is an array of threads that can cooperate
- Threads within the same block synchronize and share data in **Shared Memory**
- Execute thread blocks as CTAs on multithreaded multiprocessor SM cores



CUDA Programming Model: Thread Memory Spaces



- Each kernel thread can read:
 - Thread Id per thread
 - Block Id per block
 - Constants per grid
 - Texture per grid

- Each thread can read and write:
 - Registers per thread
 - Local memory per thread
 - Shared memory per block
 - Global memory per grid

- Host CPU can read and write:
 - Constants per grid
 - Texture per grid
 - Global memory per grid

CUDA: C on the GPU



- Single-Program Multiple-Data (SPMD) programming model
 - C program for a thread of a thread block in a grid
 - Extend C only where necessary
 - Simple, explicit language mapping to parallel threads
- Declare C kernel functions and variables on GPU:


```
__global__ void KernelFunc(...);
__device__ int GlobalVar;
__shared__ int SharedVar;
```
- Call kernel function as Grid of 500 blocks of 128 threads:


```
KernelFunc<<< 500, 128 >>>(args ...);
```
- Explicit GPU memory allocation, CPU-GPU memory transfers


```
cudaMalloc( ), cudaFree( )
cudaMemcpy( ), cudaMemcpy2D( ), ...
```

CUDA C Example: Add Arrays



C program

```
void addMatrix
(float *a, float *b, float *c, int N)
{
    int i, j, idx;
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            idx = i + j*N;
            c[idx] = a[idx] + b[idx];
        }
    }
}

void main()
{
    .....
    addMatrix(a, b, c, N);
}
```

CUDA C program

```
__global__ void addMatrixG
(float *a, float *b, float *c, int N)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    int j = blockIdx.y*blockDim.y + threadIdx.y;
    int idx = i + j*N;
    if (i < N && j < N)
        c[idx] = a[idx] + b[idx];
}

void main()
{
    dim3 dimBlock (blocksize, blocksize);
    dim3 dimGrid (N/dimBlock.x, N/dimBlock.y);
    addMatrixG<<<dimGrid, dimBlock>>>(a, b, c, N);
}
```

CUDA Software Development Kit



CUDA Optimized Libraries:
FFT, BLAS, ...

Integrated CPU + GPU
C Source Code

NVIDIA C Compiler

NVIDIA Assembly
for Computing (PTX)

CPU Host Code

CUDA
Driver

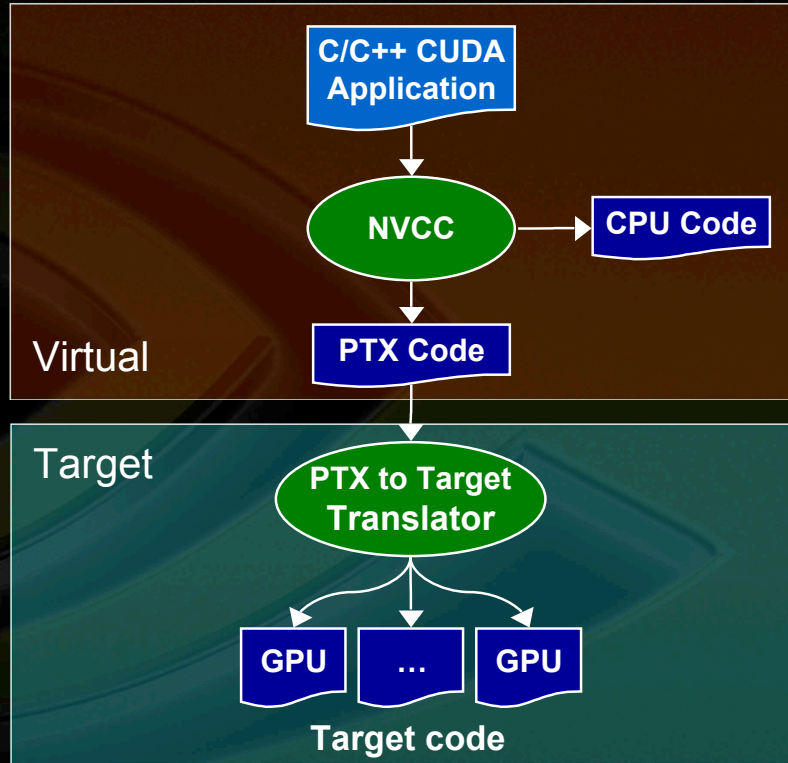
Debugger
Profiler

Standard C Compiler

GPU

CPU

Compiling CUDA Programs



GPU Computing Application Areas



A grid of application areas for GPU computing, each with a representative image and text:

- Computational Geoscience:** Image of a 3D geological model.
- Computational Chemistry:** Image of a molecular structure.
- Computational Medicine:** Image of a human torso with a red mesh overlay.
- Computational Modeling:** Image of a blue and white mechanical part.
- Computational Science:** Image of a complex scientific visualization.
- Computational Biology:** Image of a blue and white molecular structure.
- Computational Finance:** Image of a line graph showing stock market data.
- Image Processing:** Image of a woman's face with a grid overlay.

Summary



- **NVIDIA GPU Computing Architecture**
 - Computing mode enables parallel C on GPUs
 - Massively multithreaded – 1000s of threads
 - Executes parallel threads and thread arrays
 - Threads cooperate via Shared and Global memory
 - Scales to any number of parallel processor cores
 - Now on: Tesla C870, D870, S870, GeForce 8800/8600/8500, and Quadro FX 5600/4600
- **CUDA Programming model**
 - C program for GPU threads
 - Scales transparently to GPU parallelism
 - Compiler, tools, libraries, and driver for GPU Computing
 - Supports Linux and Windows

<http://www.nvidia.com/Tesla>
<http://developer.nvidia.com/CUDA>