Enabling GPU Computing

- GPU Computing Arch
- CUDA
  - Targeted platform for GPU Computing

Diagram:
- Thread Number
- Thread Program
- Memory
- Parallel Data Cache
- Constants
- Registers
GPU Computing

- Processors execute computing threads
- Thread Execution Manager issues threads
- 128 Thread Processors
- Parallel Data Cache accelerates processing

Thread Processor Group

- 128, 1.35 GHz processors
- 16KB Parallel Data Cache per group
- Scalar architecture
- IEEE 754 Precision
Scaling the Architecture

- Same program
- Scalable performance

GPU Computing Model

- Dedicated computing mode
- Thread programs use ‘C’
- On-chip shared memory
- General load/store
GPU Computing Model

- **Thread ID**
- **Thread Program**
  - Written in 'C'
- **Global Memory**
- **Parallel Data Cache**
  - Dedicated on-chip memory
  - Shared between threads for inter-thread communication
  - Explicitly managed – software managed cache
  - As Fast Registers

Computing Evolution

- **CPU**
  - Single thread out of cache
- **GPGPU**
  - Multiple passes through video memory

CUDA GPU Computing

- **Parallel Data Cache**
- **Shared Data**
- **Parallel execution through cache**
**Programming Model:**

**A Massively Multi-threaded Processor**

Move data-parallel application portions to the GPU

Differences between GPU and CPU threads

- Lightweight threads
- GPU supports 1000’s of threads

**Programming Model:**

**A Highly Multi-threaded Coprocessor**

- The GPU is viewed as a compute device that:
  - Is a coprocessor to the CPU or host
  - Has its own DRAM (device memory)
  - Runs many threads in parallel

- Data-parallel portions of an application execute on the device as kernels which run many cooperative threads in parallel

- Differences between GPU and CPU threads
  - GPU threads are extremely lightweight
    - Very little creation overhead
  - GPU needs 1000s of threads for full efficiency
    - Multi-core CPU needs only a few
C on the GPU

A simple, explicit programming language solution
Extend only where necessary

```c
__global__ void KernelFunc(...);
__device__ int GlobalVar;
__shared__ int SharedVar;

KernelFunc<<< 500, 128 >>>(...);
```

Execution Model

Multiple levels of parallelism

- Thread block
  - Up to 512 threads per block
  - Communicate through shared memory
  - Threads guaranteed to be resident
  - threadIdx, blockIdx
  - __syncthreads()

- Grid of thread blocks
  - f<<<nblocks, nthreads>>>(a,b,c)
C-Code Example to Add Arrays

CPU C program

```c
void add_matrix_cpu(float *a, float *b, float *c, int N)
{
    int i, j, index;
    for (i=0;i<N;i++)
    for (j=0;j<N;j++)
    {
        index =i+j*N;
        c[index]=a[index]+b[index];
    }
}
void main()
{
    ....
    add_matrix(a,b,c,N);
}
```

CUDA C program

```c
__global__ void add_matrix_gpu(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 index =i+j*N;
    c[index]=a[index]+b[index];
}
void main()
{
    ....
    add_matrix_gpu<<<dimGrid,dimBlock>>>(a,b,c,N);
}
```

Example Algorithm - Fluids

Goal: Calculate PRESSURE in a fluid

Pressure = Sum of neighboring pressures

\[ P'_n = P_1 + P_2 + P_3 + P_4 \]

So the pressure for each particle is...

Pressure_1 = P_1 + P_2 + P_3 + P_4
Pressure_2 = P_3 + P_4 + P_5 + P_6
Pressure_3 = P_5 + P_6 + P_7 + P_8
Pressure_4 = P_7 + P_8 + P_9 + P_10

Pressure depends on neighbors

Parallel Data Cache
Divergence in Parallel Computing

- Removing divergence pain from parallel programming

- SIMD Pain
  - User required to SIMD-ify
  - User suffers when computation goes divergent

- GPUs: Decouple execution width from programming model
  - Threads can diverge freely
  - Inefficiency only when granularity exceeds native machine width
  - Hardware managed
  - Managing divergence becomes performance optimization
  - Scalable

Runtime Component: Memory Management

- Explicit GPU memory allocation
- Returns **pointers** to GPU memory
- Device memory allocation
  - `cudaMalloc()`, `cudaFree()`
- Memory copy from host to device, device to host, device to device
  - `cudaMemcpy()`, `cudaMemcopy2D()`, ...
- OpenGL & DirectX interoperability
  - `cudaGLMapBufferObject()`
CUDA Software Development Kit

CUDA Optimized Libraries: math.h, 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

Compiling CUDA

C/C++ CUDA Application

NVCC

CPU Code

PTX Code

Target

PTX to Target Translator

GPU

Target code

Virtual
**Standard C Compiler**

```
float4 me = gw[gtid];
me.x += me.y * me.z;
```

- **EDG**
  - Separates GPU and CPU code
- **Open64**
  - Generates GPU PTX assembly
- **Parallel Thread eXecution (PTX)**
  - Virtual Machine and ISA
  - Programming model
  - Execution resources and state

---

**Virtual to Target ISA Translation**

```
ld.global.v4.f32  {$f1,$f3,$f5,$f7}, [0x9+0];
mad.f32           $f1, $f5, $f3, $f1;
```

- **Parallel Thread eXecution (PTX)**
  - Virtual Machine and ISA
  - Distribution format for applications
  - Install-time translation
  - “fat binary” caches target-specific versions
- **Target-specific optimization**
  - ISA differences
  - Resource allocation
  - Performance
CUBLAS Library

- Self-contained BLAS library
  - Application needs no direct interaction with CUDA driver
- Currently only a subset of BLAS core functions
  - Single/Real Routines, BLAS1 Complex, CGEMM
- Simple to use:
  - Create matrix and vector objects in GPU memory
  - Fill them with data
  - Call sequence of CUBLAS functions
  - Upload results back from GPU to host
- Column-major storage and 1-based indexing
  - For maximum compatibility with existing Fortran apps

CUFFT Library

- Efficient of FFT on CUDA
- Features
  - 1D, 2D, and 3D FFTs of complex and real-valued signal data
  - Batch execution for multiple 1D transforms in parallel
  - Transform sizes (for 1D) in the range [2, 16M]
  - Transform sizes (for 2D and 3D) in the range [2, 16384]
CUDA Stable Fluids Demo

**CUDA port of:**

---

**Single Precision Floating Point**

<table>
<thead>
<tr>
<th></th>
<th>8-Series GPU</th>
<th>SSE</th>
<th>IBM Altivec</th>
<th>Cell SPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>IEEE 754</td>
<td>IEEE 754</td>
<td>IEEE 754</td>
<td>IEEE 754</td>
</tr>
<tr>
<td>Rounding modes for</td>
<td>Round to nearest and round to zero</td>
<td>All 4 IEEE, round to nearest, zero, inf, -inf</td>
<td>Round to nearest only</td>
<td>Round to zero/truncate only</td>
</tr>
<tr>
<td>FADD and FMUL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Denormal handling</td>
<td>Flush to zero</td>
<td>Supported, 1000's of cycles</td>
<td>Supported, 1000's of cycles</td>
<td>Flush to zero</td>
</tr>
<tr>
<td>NaN support</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Overflow and Infinity support</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No infinity, only clamps to max norm</td>
</tr>
<tr>
<td>Flags</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Some</td>
</tr>
<tr>
<td>Square root</td>
<td>Software only</td>
<td>Hardware</td>
<td>Software only</td>
<td>Software only</td>
</tr>
<tr>
<td>Division</td>
<td>Software only</td>
<td>Hardware</td>
<td>Software only</td>
<td>Software only</td>
</tr>
<tr>
<td>Reciprocal estimate accuracy</td>
<td>24 bit</td>
<td>12 bit</td>
<td>12 bit</td>
<td>12 bit</td>
</tr>
<tr>
<td>Reciprocal sqrt estimate accuracy</td>
<td>23 bit</td>
<td>12 bit</td>
<td>12 bit</td>
<td>12 bit</td>
</tr>
<tr>
<td>log2(x) and 2^x estimates accuracy</td>
<td>23 bit</td>
<td>No</td>
<td>12 bit</td>
<td>No</td>
</tr>
</tbody>
</table>
### GPU Computing Roadmap

<table>
<thead>
<tr>
<th>Processors</th>
<th>2006</th>
<th>2007</th>
<th>2008</th>
</tr>
</thead>
<tbody>
<tr>
<td>G80</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>November</td>
<td>Q4</td>
<td>Q1</td>
<td>Q2</td>
</tr>
<tr>
<td>• First Compute</td>
<td>Q3</td>
<td>Q4</td>
<td></td>
</tr>
<tr>
<td>Beta</td>
<td>1.0</td>
<td>1.1</td>
<td></td>
</tr>
<tr>
<td>February</td>
<td>June</td>
<td>Nov</td>
<td></td>
</tr>
<tr>
<td>• Wide release</td>
<td>• Production release</td>
<td>• WinXP64</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Async</td>
</tr>
</tbody>
</table>

#### More Info

http://www.nvidia.com/cuda