Hello CUDA World !

Hello.cpp

```
#include <stdlib.h>
#include <stdio.h>

int main( int argc, char** argv)
{
   unsigned int mem_size=1024*1024*sizeof(float);
   float* h_idata = (float*) malloc( mem_size);
   load( “in.tga”, h_idata);

   for ( int i=0; i<1024 * 1024; i++)
        h_idata[ i ] = h_idata [ i ] * 0.5f;

   save( “out.tga”, h_idata);
   return 1;
}
```

Hello.cu

```
#include <stdlib.h>
#include <stdio.h>

int main( int argc, char** argv)
{
        return 1;
}
```

```
#include <stdlib.h>
#include <stdio.h>
#include <cutil.h>

int main( int argc, char** argv)
{
  CUT_DEVICE_INIT();

  CUT_EXIT(argc, argv);
}
```

```
#include <stdlib.h>
#include <stdio.h>
#include <cutil.h>

int main(int argc, char** argv)
{
    CUT_DEVICE_INIT();
    unsigned int mem_size=1024*1024*sizeof(float);
// host (CPU) input data
    float* h_idata = (float*) malloc( mem_size);

// host (CPU) output data
    float* h_odata = (float*) malloc( mem_size);

    free( h_idata);
    free( h_odata);

    CUT_EXIT(argc, argv);
}
```

```
...
int main(int argc, char** argv)
{
    CUT_DEVICE_INIT();
    unsigned int mem_size=1024*1024*sizeof(float);
    float* h_idata = (float*) malloc(mem_size);
    float* h_odata = (float*) malloc(mem_size);

    float* d_idata;        // device (GPU) input data
    cudaMalloc( (void**) &d_idata, mem_size);

    float* d_odata;        // device (GPU) output data
    cudaMalloc( (void**) &d_odata, mem_size);

    cudaFree(d_idata);
    cudaFree(d_odata);
    free( h_idata);
    free( h_odata);

    CUT_EXIT(argc, argv);
}
```

```
__global__ void kernel( float* g_idata, float* g_odata)
{

}

int main(int argc, char** argv)
{
    CUT_DEVICE_INIT();
    unsigned int mem_size=1024*1024*sizeof(float);
    float* h_idata = (float*) malloc(mem_size);
    float* h_odata = (float*) malloc(mem_size);

    float* d_idata;         // device (GPU) input data
    cudaMalloc( (void**) &d_idata, mem_size);
    float* d_odata;         // device (GPU) output data
    cudaMalloc( (void**) &d_odata, mem_size);

    cudaFree(d_idata);
    cudaFree(d_odata);
    free( h_idata);
    free( h_odata);

    CUT_EXIT(argc, argv);
}
```

```
...

__global__ void kernel( float* g_idata, float* g_odata)
{
    const unsigned int tid = threadIdx.x + blockDim.x *
        ( threadIdx.y + blockDim.y * threadIdx.z );

    const unsigned int bsize =
        blockDim.x * blockDim.y * blockDim.z;

    const unsigned int bid = ( blockIdx.x + gridDim.x *
        ( blockIdx.y + gridDim.y * blockIdx.z ) ) * bsize;
}

int main(int argc, char** argv)
{
    CUT_DEVICE_INIT();
...
```

```
...
__global__ void kernel( float* g_idata, float* g_odata)
{
    const unsigned int tid = threadIdx.x + blockDim.x *
        ( threadIdx.y + blockDim.y * threadIdx.z);

    const unsigned int bsize =
        blockDim.x * blockDim.y * blockDim.z;

    const unsigned int bid = ( blockIdx.x +
        gridDim.x * ( blockIdx.y + gridDim.y * blockIdx.z)) * bsize;

    // 1. reading from global input memory
    // 2. computation
    // 3. writing to global output memory
    g_odata[ bid + tid ] = g_idata[ bid + tid ] * 0.5f;
}

int main(int argc, char** argv)
{
    CUT_DEVICE_INIT();
...
```

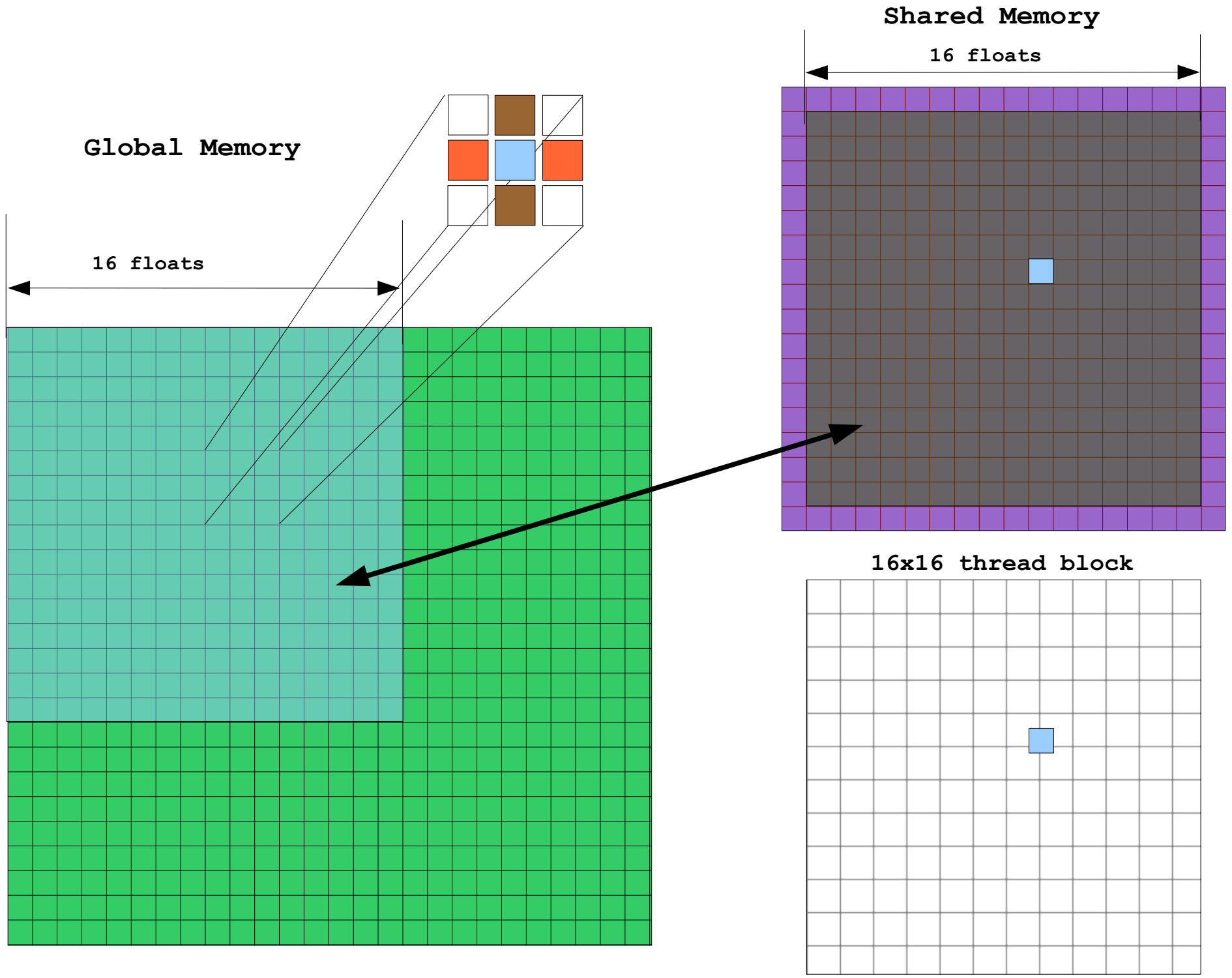
```
int main(int argc, char** argv)
{
    CUT_DEVICE_INIT();
    unsigned int mem_size=1024*1024*sizeof(float);
    float* h_idata = (float*) malloc(mem_size);
    float* h_odata = (float*) malloc(mem_size);

    float* d_idata;       // device (GPU) input data
    cudaMalloc( (void**) &d_idata, mem_size);
    float* d_odata;       // device (GPU) output data
    cudaMalloc( (void**) &d_odata, mem_size);

    // fill source host data (h_idata) by CPU

    // copy source data from device to host
    cudaMemcpy( d_idata, h_idata,
        mem_size, cudaMemcpyHostToDevice );

    cudaFree(d_idata);
    cudaFree(d_odata);
    free( h_idata);
    free( h_odata);

    CUT_EXIT(argc, argv);
}
```

```
int main(int argc, char** argv)
{
    CUT_DEVICE_INIT();
    unsigned int mem_size=1024*1024*sizeof(float);
    float* h_idata = (float*) malloc(mem_size);
    float* h_odata = (float*) malloc(mem_size);

    float* d_idata;     // device (GPU) input data
    cudaMalloc( (void**) &d_idata, mem_size);
    float* d_odata;     // device (GPU) output data
    cudaMalloc( (void**) &d_odata, mem_size);
    // fill source host data (h_idata) by CPU

    cudaMemcpy( d_idata, h_idata, mem_size,cudaMemcpyHostToDevice);

    dim3  threads( 16, 16, 1);      // threads per block
    dim3  grid( 64, 64, 1);         // blocks in grid

    // execute the kernel with defined configuration
    kernel<<< grid, threads >>>( d_idata, d_odata);
    // kernel execution is asynchronous, so CPU is free

    cudaFree(d_idata);
    cudaFree(d_odata);
    free( h_idata);
    free( h_odata);

    CUT_EXIT(argc, argv);
}
```

```
int main(int argc, char** argv)
{
    CUT_DEVICE_INIT();
    unsigned int mem_size=1024*1024*sizeof(float);
    float* h_idata = (float*) malloc(mem_size);
    float* h_odata = (float*) malloc(mem_size);

    float* d_idata;        // device (GPU) input data
    cudaMalloc( (void**) &d_idata, mem_size);
    float* d_odata;        // device (GPU) output data
    cudaMalloc( (void**) &d_odata, mem_size);

    // fill source host data (h_idata) by CPU
    cudaMemcpy( d_idata, h_idata, mem_size,cudaMemcpyHostToDevice);

    dim3  threads( 16, 16, 1);          // threads per block
    dim3  grid( 64, 64, 1);             // blocks in grid

    kernel<<< grid, threads >>>( d_idata, d_odata);

    // wait until device finishes all preceeding requested tasks
    cudaThreadSynchronize();
    // copy result from device to host
    cudaMemcpy( h_odata, d_odata,
        mem_size, cudaMemcpyDeviceToHost );

    cudaFree(d_idata);
    cudaFree(d_odata);
    free( h_idata);
    free( h_odata);

    CUT_EXIT(argc, argv);
}
```

```
int main(int argc, char** argv)
{
    CUT_DEVICE_INIT();
    unsigned int mem_size=1024*1024*sizeof(float);
    float* h_idata = (float*) malloc(mem_size);
    float* h_odata = (float*) malloc(mem_size);

    float* d_idata;         // device (GPU) input data
    cudaMalloc( (void**) &d_idata, mem_size);
    float* d_odata;         // device (GPU) output data
    cudaMalloc( (void**) &d_odata, mem_size);

    load( "in.tga", h_idata);        // fill source host data (h_idata) by CPU
    cudaMemcpy( d_idata, h_idata, mem_size,cudaMemcpyHostToDevice);

    dim3  threads( 16, 16, 1);       // threads per block
    dim3  grid( 64, 64, 1);          // blocks in grid

    kernel<<< grid, threads >>>( d_idata, d_odata);
    cudaThreadSynchronize();

    cudaMemcpy( h_odata, d_odata, mem_size,cudaMemcpyDeviceToHost);
    save( "out.tga", h_odata);

    cudaFree(d_idata);
    cudaFree(d_odata);
    free( h_idata);
    free( h_odata);

    CUT_EXIT(argc, argv);
}
```

```
unsigned char head[18+768];
unsigned char image[1024*1024];

void load( const char* s, float* o_data )
{
    FILE *f = fopen( s, "rb" );
    fread( head, 18+768, 1, f );
    fread( image, 1024*1024, 1, f );
    fclose( f );
    for ( int i=0; i<1024*1024; i++ )
        o_data[ i ] = (float) image [ i ];
}

void save( const char* s, float* i_data )
{
    FILE *f = fopen( s, "wb" );
    fwrite( head, 18+768, 1, f);
    for ( int i=0; i<1024*1024; i++ )
        image[ i ] = (unsigned char) i_data[ i ];
    fwrite( image, 1024*1024, 1, f );
    fclose( f );
}
```


Shared Memory
16 floats
Global Memory
16 floats
16x16 thread block

```
__global__ void kernel( float* g_idata, float* g_odata)
{
    __shared__ float s_data[25*25];

    const unsigned int sdid = (threadIdx.y+1) * (blockDim.x+2) +
                                    threadIdx.x + 1;

    const unsigned int gdid=threadIdx.x + blockDim.x * (blockIdx.x +
        gridDim.x * (threadIdx.y + blockIdx.y * blockDim.y));

    s_data[sdid]=g_idata[gdid];

    __syncthreads();

    float x=((s_data[sdid+1] – s_data[sdid-1]) * 0.5f) + 128.0f;

    float y=((s_data[sdid +(blockDim.x+2)] -
                s_data[sdid -(blockDim.x+2)]) * 0.5f) + 128.0f;

    g_odata[gdid]=(y+x)*0.5;
}
```

CUDA 1.1 beta

- Asynchronous Memory copy
  - Streams and Asynchronous memcpy functions
    - CPU/GPU concurrency and compute/host<->device memcpy concurrency
- Events
  - enable high-precision timing and CPU/GPU synchronization

Async example

```
cudaEvent_t start, stop;

cudaEventCreate(&start);
cudaEventCreate(&stop);

cudaEventRecord(start, 0);     // 0 = all streams

cudaMemcpyAsync(d_a, a, nbytes, cudaMemcpyHostToDevice, 0);
kernel<<<blocks, threads, 0, 0>>>(d_a, value);     // 0 = all streams
cudaMemcpyAsync(a, d_a, nbytes, cudaMemcpyDeviceToHost, 0);

cudaEventRecord(stop, 0);     // 0 = all streams

while( cudaEventQuery(stop) == cudaErrorNotReady )
{
        // do some CPU work
}

unsigned int gpu_time;
cudaEventElapsedTime(&gpu_time, start, stop);
```

References

- Nvidia CUDA SDK version 1.0 & 1.1
  - Linux
    - http://developer.download.nvidia.com/compute/cuda/1_0/linux/sdk/NVIDIA_CUDA_SDK_1.0.run
  - Windows XP
    - http://developer.download.nvidia.com/compute/cuda/1_0/windows/sdk/NVIDIA_CUDA_SDK_1.0.exe
- Nvidia CUDA Programming Guide 1.0 & 1.1
  - http://developer.download.nvidia.com/compute/cuda/1_0/NVIDIA_CUDA_Programming_Guide_1.0.pdf