CUDA Implementation of the Lattice Boltzmann Method
CSE 633 Parallel Algorithms

Andrew Leach
University at Buffalo
2 Dec 2010
Motivation

- The Lattice Boltzmann Method (LBM) solves the Navier-Stokes equation accurately and efficiently.
The Lattice Boltzmann Method (LBM) solves the Navier-Stokes equation accurately and efficiently. Uniformity makes it easy to parallelize.
Motivation

- The Lattice Boltzmann Method (LBM) solves the Navier-Stokes equation accurately and efficiently.
- Uniformity makes it easy to parallelize.
- High volume of simple calculations make it ideal for GPGPU computing.
Each lattice point has an associated mass density
Each lattice point has an associated mass density
This mass density is projected in 9 directions
At each time step, each neighbor passes mass density
Collision occurs with the accepted mass densities
Collision occurs with the accepted mass densities

Equilibrium condition is solved
Collision occurs with the accepted mass densities
Equilibrium condition is solved
New projected mass densities are assigned
Bounceback is implemented at solid boundaries
Bounceback is implemented at solid boundaries
The inlet has predetermined mass density
LBM Boundary Conditions

- Bounceback is implemented at solid boundaries
- The inlet has predetermined mass density
- The outlet accepts outward flow
Data initialized as an array on host
- Data initialized as an array on host
- Pitch stores the width of a row in memory, determined by CudaMallocPitch()
Data initialized as an array on host

- Pitch stores the width of a row in memory, determined by CudaMallocPitch()
- Memory is allocated on the device linear memory with CudaMallocArray()
Data initialized as an array on host
- Pitch stores the width of a row in memory, determined by CudaMallocPitch()
- Memory is allocated on the device linear memory with CudaMallocArray()
- Array copied from host to device with CudaMemcpy2D()
The stream step requires a lot of data retrieval
The stream step requires a lot of data retrieval.

Texture memory has fast retrieval but limited space.
The stream step requires a lot of data retrieval
Texture memory has fast retrieval but limited space
Use cudaBindTextureToArray() to copy data as a texture
A kernel is launched on a grid of blocks.
A kernel is launched on a grid of blocks.

Each block consists of threads which will independently run the kernel (SIMD).
A kernel is launched on a grid of blocks.
Each block consists of threads which will independently run the kernel (SIMD).
What follows is the Kernel for the stream() method. This example utilizes a lock-step texture look up.
//Define stream kernel//
__global__ void stream_kernel(int pitch, float *f1, float *f2,
    float *f3, float *f4, float *f5, float *f6, float *f7, float *f8){
    int i, j, id;
    i = blockDim.x * blockIdx.x + threadIdx.x;
    j = blockDim.y * blockIdx.y + threadIdx.y;
    id = i + j * (pitch/sizeof(float));

    //Gather adjacent f's for the stream step//
    f1[id] = tex2D(f1_tex, (float) i-1, (float) j);
    f2[id] = tex2D(f2_tex, (float) i, (float) j-1);
    f3[id] = tex2D(f3_tex, (float) i+1, (float) j);
    f4[id] = tex2D(f4_tex, (float) i, (float) j+1);
    f5[id] = tex2D(f5_tex, (float) i-1, (float) j+1);
    f6[id] = tex2D(f6_tex, (float) i+1, (float) j-1);
    f7[id] = tex2D(f7_tex, (float) i+1, (float) j+1);
    f8[id] = tex2D(f8_tex, (float) i-1, (float) j+1);
}

The following slides contain graphs comparing run times for the LBM on a laptop with 1.3 GHZ processor running sequential C code and a single Tesla GPU running parallel code in CUDA. The change in performance based on block size is also explored.
Sequential vs Parallel

A. Leach (University at Buffalo)

CUDA LBM

Nov 2010 14 / 16
Thanks to Dr. Graham Pullan from Cambridge University for letting me use and modify his code.