CUDA Implementation in MASS

Multi Agent Parallelization on the GPU
About Adam

- Pursuing second Bachelor's degree
  BA in Psychology from UBC in 2005

- Strong interest in hardware/embedded systems

- Found MASS interesting due to curiosity about scientific analysis. Particularly CFD.
Last Quarter

- Learning CUDA
- Learning Parallelization
- Learning Memory Management
- Beginning to understand MASS organization
CUDA

- Created and maintained by NVIDIA
- Extension of C++
- Contains all the same functionality of C++
- Adds ability to address GPU "devices"

- Devices run "kernels", exactly identical operations sent to GPU cores to compute simultaneously.
New Code Understanding

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

__device__ int get_global_index(void)
{ 
    return blockIdx.x * blockDim.x + threadIdx.x;
}

__device__ int get_constant(void)
{
    return 7;
}

__global__ void kernel1(int *array)
{ 
    int index = get_global_index();
    array[index] = get_constant();
}

__global__ void kernel2(int *array)
{ 
    int index = get_global_index();
    array[index] = get_global_index();
}

int main(void)
{ 
    int num_elements = 256;
    int num_bytes = num_elements * sizeof(int);
    int *device_array = 0;
    int *host_array = 0;

    // allocate memory
    host_array = (int*)malloc(num_bytes);
    cudaMalloc(void**)&device_array, num_bytes);
    int block_size = 128;
    int grid_size = num_elements / block_size;

    // launch kernel and inspect its results
    kernel1<<<grid_size, block_size>>>(device_array);
    cudaMemcpy(host_array, device_array, num_bytes, cudaMemcpyDeviceToHost);
    printf("kernel1 results: 
    for(int i = 0; i < num_elements; ++i)
{ 

__device__ kernels are functions that can be run within a core

__global__ kernels are functions for the entire device which are sent to every core

Device CAN access CPU memory, but slowly.

Thread has access to thread, block, device, and CPU memory.

May be an opportunity to avoid repeated CUDAmemcpy calls.
Hardware Layout

- GPU "Device"
  - GPU Core
- CPU "Host"
  - CPU Memory
  - Device Memory
  - Block Memory
  - Thread Memory
Rob's Work

- Functional version for running some specific programs
  - Range 2D, Wave 2D, Stripe 2D, Grid 2D

- Requires manual editing of MASS.cu due to direct incorporation of his implementations.

- Uses streams
Brief aside on Streams

Simultaneous memory copy and kernel execution.
Expanding Rob's Work

- Need to understand association between his version of MASS and the implementations of the various ___2D programs.

- Need to be able to generalize and his implementation.

- Make MASS static
Previous Challenges

"Can't divide problem directly to GPU cores"
- Thanks to __Device__ kernels, this is possible
- Problem of efficient division of problem across cores remains

"Simple cores may be incapable of effective load balancing"
- Still appears to be likely.
- Individual cores do not have this level of control from the device.
- Rob Jordan knew a lot more than I do.
- Extract Places creation functionality
- Extract CallAll functionality
- Make MASS.cu static
New Challenges

- CPU to GPU to Block to Thread (core)
  Need to pass instruction all the way down.
  Need to establish a place on a core.

- How to divide the data if...
  places outnumber cores
  cores significantly outnumber places