

Tuned and Wildly Asynchronous Stencil Kernels for Hybrid CPU/GPU Systems

Mickey Ju

May 12, 2011

Jacobi's method for the 2D Poisson Equation

Mickey Ju

Discrete Approximation

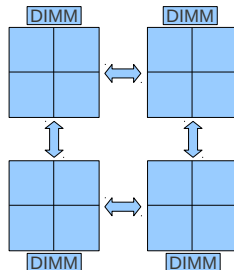
```
for  $t \leftarrow 1$  to  $T$  do
  for  $i \leftarrow 1$  to  $n$  do
    for  $j \leftarrow 1$  to  $n$  do
       $U_{i,j}^{t+1} \leftarrow \frac{1}{4} \cdot \left( U_{i+1,j}^t + U_{i-1,j}^t + U_{i,j+1}^t + U_{i,j-1}^t + h^2 \cdot F_{i,j} \right)$ 
    end for
  end for
end for
```

CPU Baselines

Mickey Ju

- Vectorization, verified by inspecting the assembly code
- Bind threads to cores using the Linux affinity scheduling routines
- Allocate the per-thread data blocks in the memory of the closest processor socket
- How to get the ghost-cells from other processors?

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15



Naive GPU Implementation

Mickey Ju

- Used only global device memory
- Suffers from non-coalesced memory accesses? (Padding to solve it?)

Copy the grid from host to the device

for $t \leftarrow 1$ **to** T **do**

 Inovke GPU kernel

 (Implicit) Synchronize host and device

 Logically swap active grid

end for

Copy results from device to host

GPU Implementation Using Shared Memory

Mickey Ju

- Reduces the accesses to global memory
- Stores only one grid in global memory
- Applies loop-unrolling
- Questions

Hybrid Implementation

Mickey Ju

- Expects the overlapping to help
- Needs data-transfer overhead to be low
- Same idea can be applied to multi-GPU implementation

```
assign rows 1 to  $s$  to CPUs  
assign rows  $s + 1$  to  $n$  to the GPUs  
for  $t \leftarrow 1$  to  $T$  do  
    GPU part and CPU part  
    Exchange data  
end for
```

Async0

Mickey Ju

- Transfers the dependent elements back to global memory
- Loads the ghost cells from global memory
- Can work totally in shared memory
- Drops the implicit synchronizations between kernel calls
- Introduces more synchronizations between threads

```
for  $v \leftarrow 1$  to  $\alpha/2$  do
  Compute 1 iteration in  $B_1$ , writing to  $B_2$ 
  __syncthreads()
  Write penultimate fringe from  $B_2$  to global memory
  __syncthreads()
  Fetch fringe elements from device memory to  $B_2$ 
  __syncthreads()
  Compute 1 iteration in  $B_2$ , writing to  $B_1$ 
  __syncthreads()
  Write penultimate fringe from  $B_1$  to global memory
  __syncthreads()
  Fetch fringe elements from device memory to  $B_1$ 
  __syncthreads()
end for
Compute 1 iteration in  $B_1$ 
```

Async1

Mickey Ju

- Removes some synchronizations between threads

```
for  $v \leftarrow 1$  to  $\alpha/2$  do
  Compute 1 iteration in  $B_1$ , writing to  $B_2$ 
  Write penultimate fringe from  $B_2$  to global memory
  Fetch fringe elements from device memory to  $B_2$ 
  __syncthreads()
  Compute 1 iteration in  $B_2$ , writing to  $B_1$ 
  Write penultimate fringe from  $B_1$  to global memory
  Fetch fringe elements from device memory to  $B_1$ 
  __syncthreads()
end for
Compute 1 iteration in  $B_1$ 
```

Async2

Mickey Ju

- Does not transfer elements on border back and forth

```
for  $v \leftarrow 1$  to  $\alpha/2$  do
  Compute 1 iteration in  $B_1$ , writing to  $B_2$ 
  __syncthreads()
  Compute 1 iteration in  $B_2$ , writing to  $B_1$ 
  __syncthreads()
end for
Compute 1 iteration in  $B_1$ 
```

Async3

Mickey Ju

- Eliminates all synchronizations between threads
- Uses only one array in kernel

for $v \leftarrow 1$ **to** α **do**

 Compute 1 iteration in B , writing to B

 Write penultimate fringe from B to global memory

 Fetch fringe elements from device memory to B

end for

Compute 1 iteration in B