

What, You Mean GPUs Can Be Fast?!

Mentor: Brent Leback

CAPSL: José Monsalve, Jaime Arteaga, Stéphane Zuckerman

Previously, in “What do you mean, I don’t know how to GPU?”

- 2D Stencil code (memory bound case—data set doesn’t fit in caches)
- The speedup was $\sim 2.5\times$ over sequential runs in the best cases
 - The naïve OpenMP code yields a $\sim 3.5/4\times$ speedup (on 2-socket/32 threads and 4-socket/48 threads)
 - Our own data-driven runtime yields $\sim 6\times$ (same platforms), using fine-grain synchronization
- It was kindly suggested to us that we should maybe kinda sorta learn more about writing (good) GPU code:
 - Bad (non-unit) stride memory access pattern
 - One thread was doing $3\times$ more work than the others (was processing the halo all by itself).

The Code, Then and Now.

Before

```
__global__ void gpu_stencil2D_4pt(double * dst, double * src, int M, int N) {
    extern __shared__ double shared_mem[];
    double * shSrc = shared_mem;

    double north, south, east, west;

    int smColDim = HALO*2+blockDim.y*TILE_SIZE;
    int smRowDim = HALO*2+blockDim.x*TILE_SIZE;

    for (int i = 0; i < TILE_SIZE; i++) {
        for (int j = 0; j < TILE_SIZE; j++) {
            int globalIndex=HALO*M+blockIdx.x*blockDim.x*TILE_SIZE*M+threadIdx.x*TILE_SIZE*M+i*N+blockIdx.y*blockDim.y*TILE_SIZE+threadIdx.y*TILE_SIZE+j*HALO;
            int shMemIndex=HALO*smColDim+threadIdx.x*smColDim*TILE_SIZE+i*smColDim+HALO+threadIdx.y*TILE_SIZE+j;
            shSrc[shMemIndex]=src[globalIndex];
        }
    }

    if (threadIdx.x == 0 && threadIdx.y == 0) {
        int indexTopHalo, indexBottomHalo, indexLeftHalo, indexRightHalo;
        for (int i = 0; i < HALO; i++)
            for (int j = 0; j < smColDim; j++) {
                indexTopHalo = (blockIdx.x*blockDim.x*TILE_SIZE+i)*M + (blockIdx.y*blockDim.y*TILE_SIZE) + j;
                indexBottomHalo = (HALO + (blockIdx.x+1)*blockDim.x*TILE_SIZE)*M + (blockIdx.y*blockDim.y*TILE_SIZE)+j;
                shSrc[(i*smColDim+j) + src[indexTopHalo];
                shSrc[(HALO+blockDim.x*TILE_SIZE+i)*smColDim + j] = src[indexBottomHalo];
            }

        for (int i = 0; i < HALO; i++)
            for (int j = 0; j < smRowDim-HALO*2; j++) {
                indexLeftHalo = (HALO+blockIdx.x*blockDim.x*TILE_SIZE+j)*M + (blockIdx.y*blockDim.y*TILE_SIZE)+i;
                indexRightHalo = (HALO+blockIdx.x*blockDim.x*TILE_SIZE+j)*M + ((blockIdx.y+1)*blockDim.y*TILE_SIZE)+HALO+i;
                shSrc[(HALO+j)*smColDim+i] = src[indexLeftHalo];
                shSrc[(HALO+j+1)*smColDim-HALO+i] = src[indexRightHalo];
            }
    }
    __syncthreads();

    for (int i = 0; i < TILE_SIZE; i++)
        for (int j = 0; j < TILE_SIZE; j++) {
            int globalIndex=HALO*M+blockIdx.x*blockDim.x*TILE_SIZE*M+threadIdx.x*TILE_SIZE*M+i*N+blockIdx.y*blockDim.y*TILE_SIZE+threadIdx.y*TILE_SIZE+j*HALO;
            int shMemIndex=HALO*smColDim+threadIdx.x*smColDim*TILE_SIZE+i*smColDim+HALO+threadIdx.y*TILE_SIZE+j;

            north = shSrc[shMemIndex-smColDim];
            south = shSrc[shMemIndex+smColDim];
            east = shSrc[shMemIndex+1];
            west = shSrc[shMemIndex-1];
            dst[globalIndex] = ( north + south + east + west )/5.5;
        }
}
```

Now

```
__global__ void gpu_stencil2D_4pt_hack1(double * dst, double * src, int M, int N) {
    __shared__ double shared_mem[GRID_TILE_Y + HALO*2 ] [ GRID_TILE_X + HALO*2];
    //Cols * numRows/Tile * tileIndex
    int base_global_idx = ( N ) * ( GRID_TILE_Y * blockDim.y ) + GRID_TILE_X*blockIdx.x;

    for (int i = 0 ; i < GRID_TILE_Y+2*HALO ; i ++ )
        for (int j = threadIdx.x ; j < GRID_TILE_X+2*HALO ; j+=blockDim.x) {
            shared_mem [i] [j] = src[base_global_idx + i*N + j];
        }
    __syncthreads();

    for (int i = HALO ; i < GRID_TILE_Y+HALO ; i ++ )
        for (int j = threadIdx.x + HALO ; j < GRID_TILE_X+HALO ; j+=blockDim.x) {
            //left + right + top + bottom
            dst[base_global_idx + i*N + j] = (shared_mem[i-1] [j] + shared_mem[i+1] [j] + shared_mem[i] [j-1] + shared_mem[i] [j+1] )/5.5;
        }
    __syncthreads();
}
```

So, Er, We Finally Did It.

- Note: Yes, we could do better (in fact, our OpenACC code which is GPU-only does run even faster)
- But our objective is to evaluate the Host-GPU communication and overlapping strategies

0.11113 ==19330== Profiling application: ./StencilCUDA 1000 1000 30

==19330== Profiling result:

Time(%)	Time	Calls	Avg	Min	Max	Name
---------	------	-------	-----	-----	-----	------

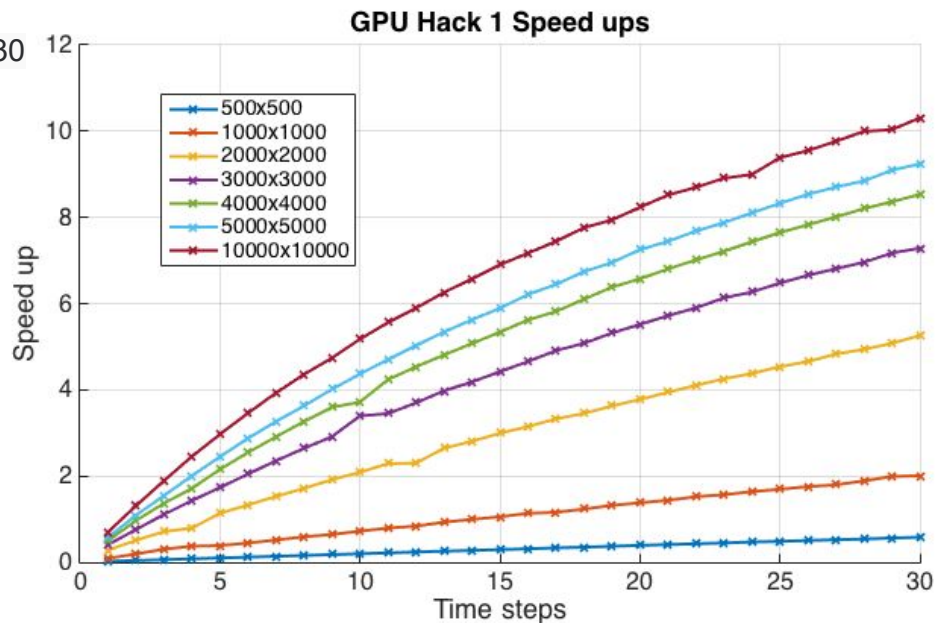
51.00%	206.18ms	600	343.62us	341.39us	346.38us	
---------------	----------	-----	----------	----------	----------	--

gpu_stencil2D_4pt_hack1

25.51%	103.12ms	40	2.5781ms	1.8760ms	3.0319ms	[CUDA memcpy DtoH]
--------	----------	----	----------	----------	----------	-----------------------

23.49%	94.947ms	40	2.3737ms	2.1776ms	5.3558ms	[CUDA memcpy HtoD]
--------	----------	----	----------	----------	----------	-----------------------

From 91% spent in the kernel



Next Steps

- Use the shared block memory as a “sliding window” to process more row elements in a single block
- **Start working on the Host \longleftrightarrow GPU communication schemes:**
 - Write an OpenMP+Cuda version (we expect ~90% of the computation to occur on the GPU)
 - Time permitting: Make use of streams to send/receive the shared rows
 - Time permitting: write a Codelet+Cuda version of the code