HMW05 Report

Jacob Engel

April 25, 2019

1 Code Verification

1.1 GPU OpenCL

N	iters	jtime ($\mu secs$)	rtime ($\mu secs$)
32	2722	3.41	3.70
48	5727	1.95	2.12
64	9692	1.16	1.13
128	34538	0.37	0.52

All calculations were calculated with 32 threads per block. The parallelized kernels become more efficient as N increased. At some point, it will plateau.

1.2 GPU CUDA

N	iters	jtime ($\mu secs$)	rtime ($\mu secs$)
32	21	3.50	4.86
48	18	3.45	5.44
64	15	3.59	5.67
128	7	3.48	5.93

Suprisingly, the number of iterations decreased as N increased. The time for the jacobian didn't change very much, except for the reductions. It makes sense that the reduction took longer, becauase the vectors that needed to be reduced were bigger.

1.3 CPU OpenCL

N	iters	jtime ($\mu secs$)	rtime ($\mu secs$)
32	2722	0.48	3.80
48	5727	0.98	6.867
64	9692	0.44	3.93
128	34538	0.41	3.00

The number of iterations are the same as OpenCL run on the GPU, which makes sense. The times are also unexpected, as the jacobian kernel seems to have taken less time on the CPU. The time recorded for n = 48 is also more than twice as much as the others, which may be because 48 is not divisible by 16.

1.4 Seriel

N	iters	time ($\mu secs$)
32	529	14.94
48	1121	33
64	1914	60
128	6964	262

The time was expectedly much longer than any code written to be run on GPUs, whether they were or not.

The makefile is designed to run on a cpu, not NOTS, and there is no command line input for the size and tolerance.

2 Kernels

2.1 Jacobi Calculation

```
__kernel void jacobi(int N,
                     __global float * u,
                     __global float *f,
                     __global float *unew){

  const int i = get_local_id(0) +
  get_local_size(0)*get_group_id(0) + 1;

  const int j = get_local_id(1) +
  get_local_size(1)*get_group_id(1) + 1;
```

```
  if (i < N+1 && j < N+1){
    const int Np = (N+2);
    const int id = i + j*(N+2);
    const float ru = -u[id-Np]-u[id+Np]-u[id-1]-u[id+1];
    const float newu = .25 * (f[id] - ru);
    unew[id] = newu;
  }
}
```

2.2 Reduction

```
__kernel void reduce(int N2, __global  float * u,
                                        __global float * unew,
                                        __global float *res){

        __local volatile float shared_vec[BDIM];
       const int tid = get_local_id(0);
       const int i = tid +
              get_group_id(0)*(get_local_size(0));
        shared_vec[tid] = 0;
  if (i < N2){
    const float unew1 = unew[i];
    const float diff1 = unew1 - u[i];
    shared_vec[tid] = diff1*diff1;
    u[i] = unew1;
    }
  barrier(CLK_LOCAL_MEM_FENCE);
    for (unsigned int s = 1; s<get_local_size(0); s*=2){
    int index = 2*s*tid;
    if (index <get_local_size(0)){
      shared_vec[index] +=shared_vec[index+s];
    }
    barrier(CLK_LOCAL_MEM_FENCE);
  }

  if (tid == 0){
    res[get_group_id(0)] = shared_vec[0];

  }
}
```