CAAM520 Computational Science

Homework 4

AO CAI
S01255664
Earth Science Department
Rice University
April 5, 2019

Contents

1 Introduction

The Goal is to solve the matrix system $Au = b$ resulting from an $(N+2) \times (N+2)$ 2D finite difference method for Laplace's equation $-\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = f(x, y)$ on $[-1, 1]^2$. At each point (x_i, y_i), derivatives are approximated by:

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2}, \frac{\partial^2 u}{\partial y^2} \approx \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2}$$

where $h = 2/(N+1)$. Assuming zero boundary conditions $u(x, y) = 0$ reduce this to an $N \times N$ system for the interior nodes.

2 CUDA parallelism for Jacobi method

To implement the parallel computing for the Poisson's equaion, I write the CUDA code with the main funciton to allocate arrays and move data onto the GPU. In the code, one kernel is used to perform the Jacobi iteraion and a second kernel perform the objective function calculation using a reduction.

I first determined the size of blocks and number of threads per block. In my first CUDA application, I used the global memory instead of shared memory implementation. Each thread will compute the update of the corresponding $u[ii]$ by using the values from surrounding nodes, extracting from global memory, this method is less computational efficient comparing with the second implementation using shared memory.

To make a easy transit from double percision to single percision. I used the trick taught in the class. I defined the dfloat as float in all of my applications. The reduction kernel is a tree-based approach that makes the reduction within each thread block.

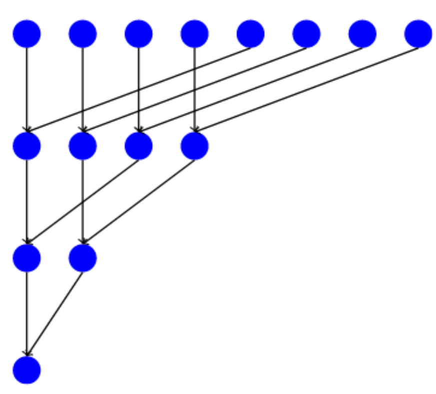


Figure 1: Tree-based GPU Reduction, cited from Yang et al., Geophysics, 2015

```
/*************************************************************************
        > File Name: cuda_Jacobi.cu
        > Author: Ao Cai
        > Mail: aocai166@gmail.com
        > Created Time: April 04 2019 09:48:38 AM CST
 ************************************************************************/

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

#define p_N 100
#define p_Nthreads 256
#define dfloat float  // switch between double/single precision
#define MAX(a,b) ((a)>(b)?(a):(b))

__global__ void compute_xk(int N, dfloat *u, dfloat *b, dfloat *res)
/* Computing the new xk and send it to u */
{
        const int ii = blockIdx.x*blockDim.x + threadIdx.x;
        const int ix = blockIdx.x;
        const int iz = threadIdx.x;

        dfloat newu = 0.0;
        if(ix > 0 && ix < N){
                if(iz > 0 && iz < N){

                        dfloat invD=0.25, Ru, tmp;
                        Ru = 0.0;
                        Ru -= (iz-1>=0)?u[ii-N]:0.0;
                        Ru -= (iz+1<N)?u[ii+N]:0.0;
                        Ru -= (ix-1>=0)?u[ii-1]:0.0;
                        Ru -= (ix+1<N)?u[ii+1]:0.0;

                        tmp = b[ii] - Ru;
                        newu = invD*tmp;
                        tmp = tmp - 4.0*u[ii];
                        res[ii] = tmp*tmp;
                }
        }
```

```
        __syncthreads();

        if(ix > 0 && ix < N){
                if(iz > 0 && iz < N){
                        u[ii] = newu;
                }
        }
}

__global__ void reduce1(int N, float *x, float *xout){

  __shared__ float s_x[p_Nthreads];

  const int tid = threadIdx.x;
  const int i = blockIdx.x*blockDim.x + tid;

  // load smem
  s_x[tid] = 0;
  if (i < N){
    s_x[tid] = x[i];
  }
  __syncthreads();

  for (unsigned int s = 1; s < blockDim.x; s *= 2){
    int index = 2*s*tid;
    if (index < blockDim.x){
      s_x[index] += s_x[index+s]; // bank conflicts
    }
    __syncthreads();
  }

  if (tid==0){
    xout[blockIdx.x] = s_x[0];
  }
}

int main(void){
        int N = p_N;

        int ii, ix, iz;
        dfloat h, tmp, tmpx, tmpz, obj=1.0, tol; // objective function & mode
        dfloat *u, *b, *res; // A is the differential matrix and b is the sou
```

```
    printf("N=%d, Thread-block size: %d\n",N,p_Nthreads);
    u = (dfloat*) calloc(N*N, sizeof(dfloat));
    b = (dfloat*) calloc(N*N, sizeof(dfloat));
    res = (dfloat*) calloc(N*N, sizeof(dfloat));

//  for(ii=0; ii<N*N; ii++){
//          u[ii] = 1.0;
//  }

    h = 2.0/(N+1.0);
    tmp = h*h;
    tol = 1e-6;

    for(iz = 0; iz < N; iz++){
            for (ix = 0; ix < N; ix++){
                    ii = ix + iz*N;

                    tmpx = (ix+1.0)*h-1.0;
                    tmpz = (iz+1.0)*h-1.0;
                    b[ii] = tmp*sin(M_PI*tmpx)*sin(M_PI*tmpz);
            }
    }

    // Allocate CUDA memory
    dfloat *c_u, *c_b, *c_res, *c_out;
    cudaMalloc(&c_u, N*N*sizeof(dfloat));
    cudaMalloc(&c_b, N*N*sizeof(dfloat));
    cudaMalloc(&c_res, N*N*sizeof(dfloat));

    // Copy host memory over to GPU
    cudaMemcpy(c_u, u, N*N*sizeof(dfloat), cudaMemcpyHostToDevice);
    cudaMemcpy(c_b, b, N*N*sizeof(dfloat), cudaMemcpyHostToDevice);
    cudaMemcpy(c_res, res, N*N*sizeof(dfloat), cudaMemcpyHostToDevice);

    // Initialization
    int Nthreads = N;
    int Nblocks = N;
    dim3 threadsPerBlock(Nthreads,1,1);
    dim3 blocks(Nblocks,1,1);

    int Nthreads_reduce = p_Nthreads;
```

```
int Nblocks_reduce = (N*N+Nthreads_reduce-1)/Nthreads_reduce;
dim3 threadsPerBlock_reduce(Nthreads_reduce,1,1);
dim3 blocks_reduce(Nblocks_reduce,1,1);

dfloat *out = (dfloat*)malloc(Nblocks_reduce*sizeof(dfloat));
cudaMalloc(&c_out, Nblocks_reduce*sizeof(dfloat));

int iter = 0;

while(obj > tol*tol & iter < 1){
        obj = 0.0;

        compute_xk<<< blocks, threadsPerBlock >>> (N, c_u, c_b, c_res

        reduce1<<< blocks_reduce, threadsPerBlock_reduce >>> (N*N, c_

        cudaMemcpy(out, c_out, Nblocks_reduce*sizeof(dfloat), cudaMem

        for (ii=0; ii< Nblocks_reduce; ii++){
                obj += out[ii];
        }
        if(!(iter%1000)){
                printf("Iter: %d, error = %lg\n", iter, sqrt(obj));
        }

        iter++;
}
printf("%s\n",cudaGetErrorString(cudaGetLastError()));

if(N==2)printf("The numerical solution u1=%f u2=%f u3=%f u4=%f\n",u[0

// check result
dfloat err=0.0;
for(int ii=0; ii<N*N; ii++){
        err = MAX(err,fabs(u[ii]-b[ii]/(h*h*2.0*M_PI*M_PI)));
}
printf("Final Iteration: %d, obj=%lg\n", iter, sqrt(obj));
printf("Max error: %lg\n", err);

// free memory on both CPU and GPU
cudaFree(c_u);
cudaFree(c_b);
```

```
        cudaFree(c_res);
        cudaFree(c_out);
        free(u);
        free(b);
        free(res);
        free(out);
}
```

In the second implementation. I improved the performance of my code by using a shared memory implementation. When the update of the interior nodes of one row of a block is computed, the values of $u[ii]$ in its two neighboring rows are stored, that include the $u[ii-N]$ and $u[ii+N]$. This will improve the efficiency of the data I/O when computing the updates of u.

```
/**********************************************************************
        > File Name: cuda_Jacobi_shared.cu
        > Author: Ao Cai
        > Mail: aocai166@gmail.com
        > Created Time: April 04 2019 09:48:38 AM CST
 *********************************************************************/

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

#define p_N 100
#define p_Nthreads 256
#define dfloat float  // switch between double/single precision
#define MAX(a,b) ((a)>(b)?(a):(b))

__global__ void compute_xk(int N, dfloat *u, dfloat *b, dfloat *res)
/* Computing the new xk and send it to u */
{
        __shared__ dfloat s_x[3*p_Nthreads];
        __shared__ dfloat s_b[p_Nthreads];

        const int ii = blockIdx.x*blockDim.x + threadIdx.x;
        const int ix = blockIdx.x;
        const int iz = threadIdx.x;

        dfloat newu = 0.0;

        if(ix>0 && ix<N){
```

```
    s_x[iz] = u[ii-N];
    s_x[iz+N] = u[ii];
    s_x[iz+2*N] = u[ii+N];
    s_b[iz] = b[ii];
  }

  __syncthreads();

  if(ix > 0 && ix < N){
    if(iz > 0 && iz < N){

      dfloat invD=0.25, Ru, tmp;
      Ru = 0.0;
      Ru -= (iz-1>=0)?s_x[iz]:0.0;
      Ru -= (iz+1<N)?s_x[iz+2*N]:0.0;
      Ru -= (ix-1>=0)?s_x[iz+N-1]:0.0;
      Ru -= (ix+1<N)?s_x[iz+N+1]:0.0;

      tmp = s_b[iz] - Ru;
      newu = invD*tmp;
      tmp = tmp - 4.0*u[ii];
      res[ii] = tmp*tmp;
    }
  }

  __syncthreads();

  if(ix > 0 && ix < N){
    if(iz > 0 && iz < N){
      u[ii] = newu;
    }
  }
}

__global__ void reduce1(int N, float *x, float *xout){

 __shared__ float s_x[p_Nthreads];

 const int tid = threadIdx.x;
 const int i = blockIdx.x*blockDim.x + tid;

 // load smem
```

```
  s_x[tid] = 0;
  if (i < N){
    s_x[tid] = x[i];
  }
  __syncthreads();

  for (unsigned int s = 1; s < blockDim.x; s *= 2){
    int index = 2*s*tid;
    if (index < blockDim.x){
      s_x[index] += s_x[index+s]; // bank conflicts
    }
    __syncthreads();
  }

  if (tid==0){
    xout[blockIdx.x] = s_x[0];
  }
}

int main(void){
        //int N = atoi(argv[1]);
        int N = p_N;

        int ii, ix, iz;
        dfloat h, tmp, tmpx, tmpz, obj=1.0, tol; // objective function & mode
        dfloat *u, *b, *res; // A is the differential matrix and b is the sou

        printf("N=%d, thread-block size: %d\n",N, p_Nthreads);
        u = (dfloat*) calloc(N*N, sizeof(dfloat));
        b = (dfloat*) calloc(N*N, sizeof(dfloat));
        res = (dfloat*) calloc(N*N, sizeof(dfloat));

//      for(ii=0; ii<N*N; ii++){
//              u[ii] = 1.0;
//      }

        h = 2.0/(N+1.0);
        tmp = h*h;
        tol = 1e-6;

        for(iz = 0; iz < N; iz++){
                for (ix = 0; ix < N; ix++){
```

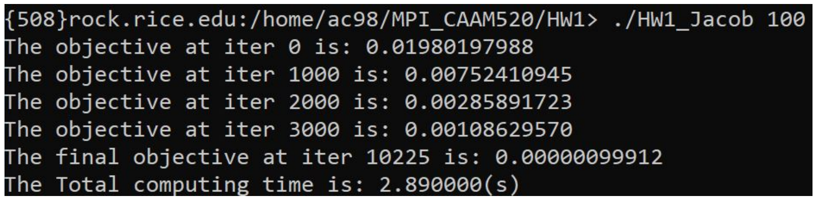
```
            ii = ix + iz*N;

            tmpx = (ix+1.0)*h-1.0;
            tmpz = (iz+1.0)*h-1.0;
            b[ii] = tmp*sin(M_PI*tmpx)*sin(M_PI*tmpz);
        }
    }

    // Allocate CUDA memory
    dfloat *c_u, *c_b, *c_res, *c_out;
    cudaMalloc(&c_u, N*N*sizeof(dfloat));
    cudaMalloc(&c_b, N*N*sizeof(dfloat));
    cudaMalloc(&c_res, N*N*sizeof(dfloat));

    // Copy host memory over to GPU
    cudaMemcpy(c_u, u, N*N*sizeof(dfloat), cudaMemcpyHostToDevice);
    cudaMemcpy(c_b, b, N*N*sizeof(dfloat), cudaMemcpyHostToDevice);
    cudaMemcpy(c_res, res, N*N*sizeof(dfloat), cudaMemcpyHostToDevice);

    // Initialization
    int Nthreads = N;
    int Nblocks = N;
    dim3 threadsPerBlock(Nthreads,1,1);
    dim3 blocks(Nblocks,1,1);

    int Nthreads_reduce = p_Nthreads;
    int Nblocks_reduce = (N*N+Nthreads_reduce-1)/Nthreads_reduce;
    dim3 threadsPerBlock_reduce(Nthreads_reduce,1,1);
    dim3 blocks_reduce(Nblocks_reduce,1,1);

    dfloat *out = (dfloat*)malloc(Nblocks_reduce*sizeof(dfloat));
    cudaMalloc(&c_out, Nblocks_reduce*sizeof(dfloat));

    int iter = 0;

    while(obj > tol*tol){
        obj = 0.0;

        compute_xk<<< blocks, threadsPerBlock >>> (N, c_u, c_b, c_res

        reduce1<<< blocks_reduce, threadsPerBlock_reduce >>> (N*N, c_
```

```
            cudaMemcpy(out, c_out, Nblocks_reduce*sizeof(dfloat), cudaMem

            for (ii=0; ii< Nblocks_reduce; ii++){
                    obj += out[ii];
            }
            if(!(iter%1000)){
                    printf("Iter: %d, error = %lg\n", iter, sqrt(obj));
            }

            iter++;
    }
    printf("%s\n",cudaGetErrorString(cudaGetLastError()));

    if(N==2)printf("The numerical solution u1=%f u2=%f u3=%f u4=%f\n",u[0

    // check result
    dfloat err=0.0;
    for(int ii=0; ii<N*N; ii++){
            err = MAX(err,fabs(u[ii]-b[ii]/(h*h*2.0*M_PI*M_PI)));
    }
    printf("Final Iteration: %d, obj= %lg\n", iter, sqrt(obj));
    printf("Max error: %lg\n", err);

    // free memory on both CPU and GPU
    cudaFree(c_u);
    cudaFree(c_b);
    cudaFree(c_res);
    cudaFree(c_out);
    free(u);
    free(b);
    free(res);
    free(out);
}
```

3 Weighted Jacobi method: Numerical results

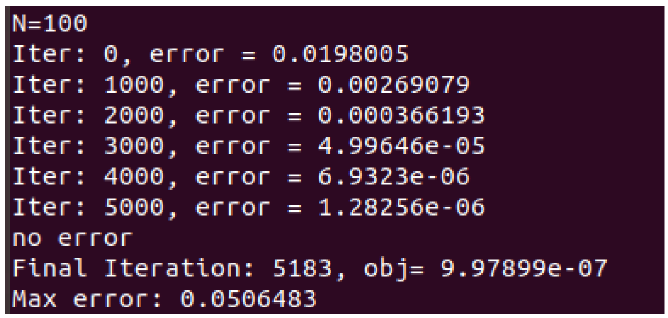
In this section, I will show the numerical result by runing the parallelized weighted Jacobi method. The objective function is the L2-norm of $b - Au$

First is the verification of my code's correctness. For $N = 100$, the reported errors by my serial code and the cuda code is shown below:



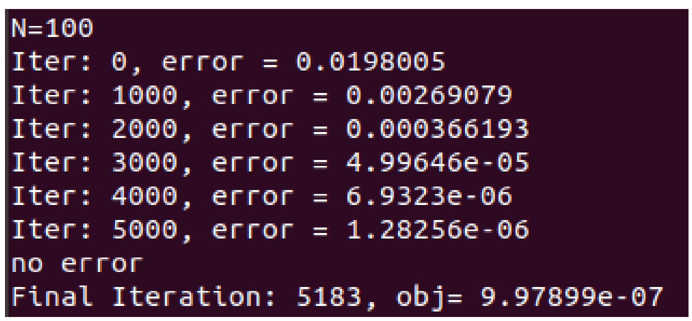
```
{508}rock.rice.edu:/home/ac98/MPI_CAAM520/HW1> ./HW1_Jacob 100
The objective at iter 0 is: 0.01980197988
The objective at iter 1000 is: 0.00752410945
The objective at iter 2000 is: 0.00285891723
The objective at iter 3000 is: 0.00108629570
The final objective at iter 10225 is: 0.00000099912
The Total computing time is: 2.890000(s)
```

Figure 2: Reported error by Serial Code



```
N=100
Iter: 0, error = 0.0198005
Iter: 1000, error = 0.00269079
Iter: 2000, error = 0.000366193
Iter: 3000, error = 4.99646e-05
Iter: 4000, error = 6.9323e-06
Iter: 5000, error = 1.28256e-06
no error
Final Iteration: 5183, obj= 9.97899e-07
Max error: 0.0506483
```

Figure 3: Reported error by CUDA global memory Implementation



```
N=100
Iter: 0, error = 0.0198005
Iter: 1000, error = 0.00269079
Iter: 2000, error = 0.000366193
Iter: 3000, error = 4.99646e-05
Iter: 4000, error = 6.9323e-06
Iter: 5000, error = 1.28256e-06
no error
Final Iteration: 5183, obj= 9.97899e-07
```

Figure 4: Reported error by CUDA shared memory Implementation

Comparing the serial code results with the CUDA implementation results, we know that the implementation is correct, as the initial error is nearly the same (0.01980197988 to 0.019805), the small difference is due to the difference percision, as we are using double percision in the serial code and single percision in the CUDA code. From this, we verify the correct implementation of our code.

```
N=100, Thread-block size: 128
==11005== NVPROF is profiling process 11005, command: ./cuda_Jacobi1
==11005== Warning: Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
Iter: 0, error = 0.0198005
no error
Final Iteration: 1, obj= 0.0198005
Max error: 0.0506483
==11005== Profiling application: ./cuda_Jacobi1
==11005== Profiling result:
==11005== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "Tesla K80 (0)"
        Kernel: compute_xk(int, float*, float*, float*)
          1                      dram_read_throughput             Device Memory Read Throughput  66.528MB/s  66.528MB/s  66.528MB/s
        Kernel: reduce1(int, float*, float*)
          1                      dram_read_throughput             Device Memory Read Throughput  65.619MB/s  65.619MB/s  65.619MB/s
```

Figure 5: Weighted Jacobi method on N=100, thread-block 128, reading speed

```
N=100, Thread-block size: 128
==8218== NVPROF is profiling process 8218, command: ./cuda_Jacobi1
==8218== Warning: Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
Iter: 0, error = 0.0198005
no error
Final Iteration: 1, obj= 0.0198005
Max error: 0.0506483
==8218== Profiling application: ./cuda_Jacobi1
==8218== Profiling result:
==8218== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "Tesla K80 (0)"
        Kernel: compute_xk(int, float*, float*, float*)
          1                     dram_write_throughput            Device Memory Write Throughput  13.296GB/s  13.296GB/s  13.296GB/s
        Kernel: reduce1(int, float*, float*)
          1                     dram_write_throughput            Device Memory Write Throughput  105.84MB/s  105.84MB/s  105.84MB/s
```

Figure 6: Weighted Jacobi method on N=100, thread-block 128, write speed

```
N=100, Thread-block size: 128
==10949== NVPROF is profiling process 10949, command: ./cuda_Jacobi1
Iter: 0, error = 0.0198005
no error
Final Iteration: 1, obj= 0.0198005
Max error: 0.0506483
==10949== Profiling application: ./cuda_Jacobi1
==10949== Profiling result:
==10949== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "Tesla K80 (0)"
        Kernel: compute_xk(int, float*, float*, float*)
          1                             flop_count_sp  Floating Point Operations(Single Precisi       49005       49005       49005
        Kernel: reduce1(int, float*, float*)
          1                             flop_count_sp  Floating Point Operations(Single Precisi       10033       10033       10033
```

Figure 7: Weighted Jacobi method on N=100, thread-block 128, flop counts

```
N=500, Thread-block size: 128
==10866== NVPROF is profiling process 10866, command: ./cuda_Jacobi1
==10866== Warning: Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
Iter: 0, error = 0.00399203
no error
Final Iteration: 1, obj= 0.00399203
Max error: 0.0506601
==10866== Profiling application: ./cuda_Jacobi1
==10866== Profiling result:
==10866== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "Tesla K80 (0)"
        Kernel: compute_xk(int, float*, float*, float*)
          1                      dram_read_throughput             Device Memory Read Throughput  59.126GB/s  59.126GB/s  59.126GB/s
        Kernel: reduce1(int, float*, float*)
          1                      dram_read_throughput             Device Memory Read Throughput  23.111GB/s  23.111GB/s  23.111GB/s
```

Figure 8: Weighted Jacobi method on N=500, thread-block 128, reading speed

```
N=500, Thread-block size: 128
==7861== NVPROF is profiling process 7861, command: ./cuda_Jacobi1
==7861== Warning: Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
Iter: 0, error = 0.0039922
no error
Final Iteration: 1, obj= 0.0039922
Max error: 0.0506601
==7861== Profiling application: ./cuda_Jacobi1
==7861== Profiling result:
==7861== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "Tesla K80 (0)"
        Kernel: compute_xk(int, float*, float*, float*)
          1                      dram_write_throughput            Device Memory Write Throughput  54.421GB/s  54.421GB/s  54.421GB/s
        Kernel: reduce1(int, float*, float*)
          1                      dram_write_throughput            Device Memory Write Throughput  211.61MB/s  211.61MB/s  211.61MB/s
```

Figure 9: Weighted Jacobi method on N=500, thread-block 128, write speed

```
N=500, Thread-block size: 128
==10798== NVPROF is profiling process 10798, command: ./cuda_Jacobi1
Iter: 0, error = 0.00399201
no error
Final Iteration: 1, obj= 0.00399201
Max error: 0.0506601
==10798== Profiling application: ./cuda_Jacobi1
==10798== Profiling result:
==10798== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "Tesla K80 (0)"
        Kernel: compute_xk(int, float*, float*, float*)
          1                             flop_count_sp  Floating Point Operations(Single Precisi     1245005     1245005     1245005
        Kernel: reduce1(int, float*, float*)
          1                             flop_count_sp  Floating Point Operations(Single Precisi      248158      248158      248158
```

Figure 10: Weighted Jacobi method on N=500, thread-block 128, flop counts

```
N=1000, Thread-block size: 128
==10357== NVPROF is profiling process 10357, command: ./cuda_Jacobi1
==10357== Warning: Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
Iter: 0, error = 0.0019985
no error
Final Iteration: 1, obj= 0.0019985
Max error: 0.0506605
==10357== Profiling application: ./cuda_Jacobi1
==10357== Profiling result:
==10357== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "Tesla K80 (0)"
        Kernel: compute_xk(int, float*, float*, float*)
          1                       dram_read_throughput             Device Memory Read Throughput  64.048GB/s  64.048GB/s  64.048GB/s
        Kernel: reduce1(int, float*, float*)
          1                       dram_read_throughput             Device Memory Read Throughput  26.224GB/s  26.224GB/s  26.224GB/s
```

Figure 11: Weighted Jacobi method on N=1000, thread-block 128, reading speed

```
N=1000, Thread-block size: 128
==7768== NVPROF is profiling process 7768, command: ./cuda_Jacobi1
==7768== Warning: Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
Iter: 0, error = 0.001998
no error
Final Iteration: 1, obj= 0.001998
Max error: 0.0506605
==7768== Profiling application: ./cuda_Jacobi1
==7768== Profiling result:
==7768== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "Tesla K80 (0)"
        Kernel: compute_xk(int, float*, float*, float*)
          1                      dram_write_throughput            Device Memory Write Throughput  58.909GB/s  58.909GB/s  58.909GB/s
        Kernel: reduce1(int, float*, float*)
          1                      dram_write_throughput            Device Memory Write Throughput  237.02MB/s  237.02MB/s  237.02MB/s
```

Figure 12: Weighted Jacobi method on N=1000, thread-block 128, write speed

```
N=1000, Thread-block size: 128
==10421== NVPROF is profiling process 10421, command: ./cuda_Jacobi1
Iter: 0, error = 0.001998
no error
Final Iteration: 1, obj= 0.001998
Max error: 0.0506605
==10421== Profiling application: ./cuda_Jacobi1
==10421== Profiling result:
==10421== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "Tesla K80 (0)"
        Kernel: compute_xk(int, float*, float*, float*)
          1                             flop_count_sp  Floating Point Operations(Single Precisi     4990005     4990005     4990005
        Kernel: reduce1(int, float*, float*)
          1                             flop_count_sp  Floating Point Operations(Single Precisi      992251      992251      992251
```

Figure 13: Weighted Jacobi method on N=1000, thread-block 128, flop counts

```
Iter: 0, error = 0.0198005
no error
Final Iteration: 1, obj= 0.0198005
Max error: 0.0506483
==9982== Profiling application: ./cuda_Jacobi1
==9982== Profiling result:
==9982== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "Tesla K80 (0)"
        Kernel: compute_xk(int, float*, float*, float*)
          1                      dram_read_throughput             Device Memory Read Throughput  127.57MB/s  127.57MB/s  127.57MB/s
        Kernel: reduce1(int, float*, float*)
          1                      dram_read_throughput             Device Memory Read Throughput  59.869MB/s  59.869MB/s  59.869MB/s
```

Figure 14: Weighted Jacobi method on N=100, thread-block 256, reading speed

```
N=100, Thread-block size: 256
==13230== NVPROF is profiling process 13230, command: ./cuda_Jacobi1
==13230== Warning: Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
Iter: 0, error = 0.0198005
no error
Final Iteration: 1, obj= 0.0198005
Max error: 0.0506483
==13230== Profiling application: ./cuda_Jacobi1
==13230== Profiling result:
==13230== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "Tesla K80 (0)"
        Kernel: compute_xk(int, float*, float*, float*)
          1                     dram_write_throughput            Device Memory Write Throughput  13.178GB/s  13.178GB/s  13.178GB/s
        Kernel: reduce1(int, float*, float*)
          1                     dram_write_throughput            Device Memory Write Throughput  56.732MB/s  56.732MB/s  56.732MB/s
```

Figure 15: Weighted Jacobi method on N=100, thread-block 256, write speed

```
N=100, Thread-block size: 256
==9918== NVPROF is profiling process 9918, command: ./cuda_Jacobi1
Iter: 0, error = 0.0198005
no error
Final Iteration: 1, obj= 0.0198005
Max error: 0.0506483
==9918== Profiling application: ./cuda_Jacobi1
==9918== Profiling result:
==9918== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "Tesla K80 (0)"
        Kernel: compute_xk(int, float*, float*, float*)
          1                             flop_count_sp  Floating Point Operations(Single Precisi       49005       49005       49005
        Kernel: reduce1(int, float*, float*)
          1                             flop_count_sp  Floating Point Operations(Single Precisi       10200       10200       10200
```

Figure 16: Weighted Jacobi method on N=100, thread-block 256, flop counts

```
N=500, Thread-block size: 256
==10059== NVPROF is profiling process 10059, command: ./cuda_Jacobi1
==10059== Warning: Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
Iter: 0, error = 0.00399202
no error
Final Iteration: 1, obj= 0.00399202
Max error: 0.0506601
==10059== Profiling application: ./cuda_Jacobi1
==10059== Profiling result:
==10059== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "Tesla K80 (0)"
        Kernel: compute_xk(int, float*, float*, float*)
          1                    dram_read_throughput             Device Memory Read Throughput  58.982GB/s  58.982GB/s  58.982GB/s
        Kernel: reduce1(int, float*, float*)
          1                    dram_read_throughput             Device Memory Read Throughput  18.972GB/s  18.972GB/s  18.972GB/s
```

Figure 17: Weighted Jacobi method on N=500, thread-block 256, reading speed

```
N=500, Thread-block size: 256
==6789== NVPROF is profiling process 6789, command: ./cuda_Jacobi1
==6789== Warning: Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
Iter: 0, error = 0.00399201
no error
Final Iteration: 1, obj= 0.00399201
Max error: 0.0506601
==6789== Profiling application: ./cuda_Jacobi1
==6789== Profiling result:
==6789== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "Tesla K80 (0)"
        Kernel: compute_xk(int, float*, float*, float*)
          1                   dram_write_throughput            Device Memory Write Throughput  54.564GB/s  54.564GB/s  54.564GB/s
        Kernel: reduce1(int, float*, float*)
          1                   dram_write_throughput            Device Memory Write Throughput  102.64MB/s  102.64MB/s  102.64MB/s
```

Figure 18: Weighted Jacobi method on N=500, thread-block 256, write speed

```
N=500, Thread-block size: 256
==10124== NVPROF is profiling process 10124, command: ./cuda_Jacobi1
Iter: 0, error = 0.00399201
no error
Final Iteration: 1, obj= 0.00399201
Max error: 0.0506601
==10124== Profiling application: ./cuda_Jacobi1
==10124== Profiling result:
==10124== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "Tesla K80 (0)"
        Kernel: compute_xk(int, float*, float*, float*)
          1                         flop_count_sp  Floating Point Operations(Single Precisi     1245005     1245005     1245005
        Kernel: reduce1(int, float*, float*)
          1                         flop_count_sp  Floating Point Operations(Single Precisi      249135      249135      249135
```

Figure 19: Weighted Jacobi method on N=500, thread-block 256, flop counts

```
N=1000, Thread-block size: 256
==10274== NVPROF is profiling process 10274, command: ./cuda_Jacobi1
==10274== Warning: Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
Iter: 0, error = 0.00199801
no error
Final Iteration: 1, obj= 0.00199801
Max error: 0.0506605
==10274== Profiling application: ./cuda_Jacobi1
==10274== Profiling result:
==10274== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "Tesla K80 (0)"
        Kernel: compute_xk(int, float*, float*, float*)
          1                    dram_read_throughput             Device Memory Read Throughput  64.107GB/s  64.107GB/s  64.107GB/s
        Kernel: reduce1(int, float*, float*)
          1                    dram_read_throughput             Device Memory Read Throughput  21.116GB/s  21.116GB/s  21.116GB/s
```

Figure 20: Weighted Jacobi method on N=1000, thread-block 256, reading speed

```
N=1000, Thread-block size: 256
==7617== NVPROF is profiling process 7617, command: ./cuda_Jacobi1
==7617== Warning: Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
Iter: 0, error = 0.001998
no error
Final Iteration: 1, obj= 0.001998
Max error: 0.0506605
==7617== Profiling application: ./cuda_Jacobi1
==7617== Profiling result:
==7617== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "Tesla K80 (0)"
        Kernel: compute_xk(int, float*, float*, float*)
          1                      dram_write_throughput           Device Memory Write Throughput  58.057GB/s  58.057GB/s  58.057GB/s
        Kernel: reduce1(int, float*, float*)
          1                      dram_write_throughput           Device Memory Write Throughput  116.91MB/s  116.91MB/s  116.91MB/s
```

Figure 21: Weighted Jacobi method on N=1000, thread-block 256, write speed

```
N=1000, Thread-block size: 256
==10194== NVPROF is profiling process 10194, command: ./cuda_Jacobi1
Iter: 0, error = 0.001998
no error
Final Iteration: 1, obj= 0.001998
Max error: 0.0506605
==10194== Profiling application: ./cuda_Jacobi1
==10194== Profiling result:
==10194== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "Tesla K80 (0)"
        Kernel: compute_xk(int, float*, float*, float*)
          1                             flop_count_sp  Floating Point Operations(Single Precisi     4990005     4990005     4990005
        Kernel: reduce1(int, float*, float*)
          1                             flop_count_sp  Floating Point Operations(Single Precisi      996285      996285      996285
```

Figure 22: Weighted Jacobi method on N=1000, thread-block 256, flop counts

```
N=100, Thread-block size: 1024
==9481== NVPROF is profiling process 9481, command: ./cuda_Jacobi1
==9481== Warning: Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
Iter: 0, error = 0.0198005
no error
Final Iteration: 1, obj= 0.0198005
Max error: 0.0506483
==9481== Profiling application: ./cuda_Jacobi1
==9481== Profiling result:
==9481== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "Tesla K80 (0)"
        Kernel: compute_xk(int, float*, float*, float*)
          1                       dram_read_throughput            Device Memory Read Throughput  66.528MB/s  66.528MB/s  66.528MB/s
        Kernel: reduce1(int, float*, float*)
          1                       dram_read_throughput            Device Memory Read Throughput  18.766MB/s  18.766MB/s  18.766MB/s
```

Figure 23: Weighted Jacobi method on N=100, thread-block 1024, reading speed

```
N=100, Thread-block size: 1024
==8427== NVPROF is profiling process 8427, command: ./cuda_Jacobi1
==8427== Warning: Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
Iter: 0, error = 0.0198005
no error
Final Iteration: 1, obj= 0.0198005
Max error: 0.0506483
==8427== Profiling application: ./cuda_Jacobi1
==8427== Profiling result:
==8427== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "Tesla K80 (0)"
        Kernel: compute_xk(int, float*, float*, float*)
          1                      dram_write_throughput           Device Memory Write Throughput  13.279GB/s  13.279GB/s  13.279GB/s
        Kernel: reduce1(int, float*, float*)
          1                      dram_write_throughput           Device Memory Write Throughput  23.474MB/s  23.474MB/s  23.474MB/s
```

Figure 24: Weighted Jacobi method on N=100, thread-block 1024, write speed

```
N=100, Thread-block size: 1024
==9561== NVPROF is profiling process 9561, command: ./cuda_Jacobi1
Iter: 0, error = 0.0198005
no error
Final Iteration: 1, obj= 0.0198005
Max error: 0.0506483
==9561== Profiling application: ./cuda_Jacobi1
==9561== Profiling result:
==9561== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "Tesla K80 (0)"
        Kernel: compute_xk(int, float*, float*, float*)
          1                             flop_count_sp  Floating Point Operations(Single Precisi       49005       49005       49005
        Kernel: reduce1(int, float*, float*)
          1                             flop_count_sp  Floating Point Operations(Single Precisi       10230       10230       10230
```

Figure 25: Weighted Jacobi method on N=100, thread-block 1024, flop counts

```
N=500, Thread-block size: 1024
==9386== NVPROF is profiling process 9386, command: ./cuda_Jacobi1
==9386== Warning: Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
Iter: 0, error = 0.00399201
no error
Final Iteration: 1, obj= 0.00399201
Max error: 0.0506601
==9386== Profiling application: ./cuda_Jacobi1
==9386== Profiling result:
==9386== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "Tesla K80 (0)"
        Kernel: compute_xk(int, float*, float*, float*)
          1                      dram_read_throughput             Device Memory Read Throughput  59.136GB/s  59.136GB/s  59.136GB/s
        Kernel: reduce1(int, float*, float*)
          1                      dram_read_throughput             Device Memory Read Throughput  11.504GB/s  11.504GB/s  11.504GB/s
```

Figure 26: Weighted Jacobi method on N=500, thread-block 1024, reading speed

```
N=500, Thread-block size: 1024
==8651== NVPROF is profiling process 8651, command: ./cuda_Jacobi1
==8651== Warning: Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
Iter: 0, error = 0.00399201
no error
Final Iteration: 1, obj= 0.00399201
Max error: 0.0506601
==8651== Profiling application: ./cuda_Jacobi1
==8651== Profiling result:
==8651== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "Tesla K80 (0)"
        Kernel: compute_xk(int, float*, float*, float*)
          1                     dram_write_throughput            Device Memory Write Throughput  54.359GB/s  54.359GB/s  54.359GB/s
        Kernel: reduce1(int, float*, float*)
          1                     dram_write_throughput            Device Memory Write Throughput  18.160MB/s  18.160MB/s  18.160MB/s
```

Figure 27: Weighted Jacobi method on N=500, thread-block 1024, write speed

```
N=500, Thread-block size: 1024
==9307== NVPROF is profiling process 9307, command: ./cuda_Jacobi1
Iter: 0, error = 0.00399201
no error
Final Iteration: 1, obj= 0.00399201
Max error: 0.0506601
==9307== Profiling application: ./cuda_Jacobi1
==9307== Profiling result:
==9307== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "Tesla K80 (0)"
        Kernel: compute_xk(int, float*, float*, float*)
          1                             flop_count_sp  Floating Point Operations(Single Precisi     1245005     1245005     1245005
        Kernel: reduce1(int, float*, float*)
          1                             flop_count_sp  Floating Point Operations(Single Precisi      250635      250635      250635
```

Figure 28: Weighted Jacobi method on N=500, thread-block 1024, flop counts

```
N=1000, Thread-block size: 1024
==9125== NVPROF is profiling process 9125, command: ./cuda_Jacobi1
==9125== Warning: Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
Iter: 0, error = 0.001998
no error
Final Iteration: 1, obj= 0.001998
Max error: 0.0506605
==9125== Profiling application: ./cuda_Jacobi1
==9125== Profiling result:
==9125== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "Tesla K80 (0)"
        Kernel: compute_xk(int, float*, float*, float*)
          1                        dram_read_throughput            Device Memory Read Throughput  63.906GB/s  63.906GB/s  63.906GB/s
        Kernel: reduce1(int, float*, float*)
          1                        dram_read_throughput            Device Memory Read Throughput  12.524GB/s  12.524GB/s  12.524GB/s
```

Figure 29: Weighted Jacobi method on N=1000, thread-block 1024, reading speed

```
N=1000, Thread-block size: 1024
==8764== NVPROF is profiling process 8764, command: ./cuda_Jacobi1
==8764== Warning: Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
Iter: 0, error = 0.001998
no error
Final Iteration: 1, obj= 0.001998
Max error: 0.0506605
==8764== Profiling application: ./cuda_Jacobi1
==8764== Profiling result:
==8764== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "Tesla K80 (0)"
        Kernel: compute_xk(int, float*, float*, float*)
          1                       dram_write_throughput           Device Memory Write Throughput  58.869GB/s  58.869GB/s  58.869GB/s
        Kernel: reduce1(int, float*, float*)
          1                       dram_write_throughput           Device Memory Write Throughput  20.784MB/s  20.784MB/s  20.784MB/s
```

Figure 30: Weighted Jacobi method on N=1000, thread-block 1024, write speed

```
Iter: 0, error = 0.001998
no error
Final Iteration: 1, obj= 0.001998
Max error: 0.0506605
==9204== Profiling application: ./cuda_Jacobi1
==9204== Profiling result:
==9204== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "Tesla K80 (0)"
        Kernel: compute_xk(int, float*, float*, float*)
          1                           flop_count_sp  Floating Point Operations(Single Precisi     4990005     4990005     4990005
        Kernel: reduce1(int, float*, float*)
          1                           flop_count_sp  Floating Point Operations(Single Precisi      999471      999471      999471
```

Figure 31: Weighted Jacobi method on N=1000, thread-block 1024, flop counts

In sum, using parallelism helps improving the performance of linear solver, such as weighted Jacobi method. I find out that the reading and writing speed with generally increase with larger number of the problem size N and the top reading and writing speed are reached with N=1000 and p Nthreads = 1024, which means the largest thread-block size. However, the increase of the I/O is limited from $N = 256$ to $N = 1024$, as we are approaching the roofline of the computation resources.

In the numerical computation, I am using the K80 GPU on the nots server. From the website, it has 480 GB/s aggregate memory bandwidth and 8.73 teraflops single-precision performance. The roofline plots are provided in the following:

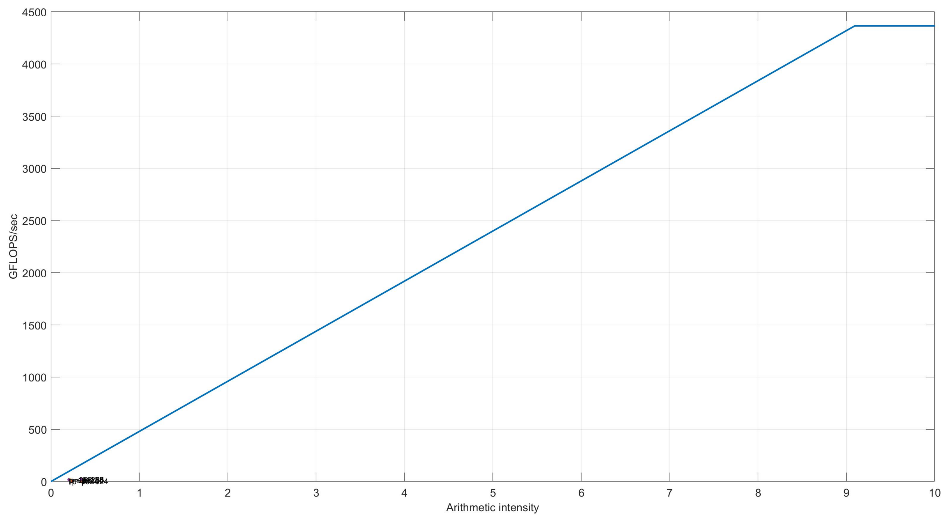


Figure 32: Weighted Jacobi method on N=1000, with different thread-block Roofline

From the roofline plot, the code is still away from the optimum. The reason is that the roofline is the peak performance that the GPU could have, which is usually not available in the numerical examples. Besides, I didn't make the plot for the share memory improved results, so currently I still not close to the roofline.