CSCI 596: HW 6

M. Oneeb H. Khan

October 25, 2021

1 Pair-Distribution Computation with CUDA

1.1 Source Code of *pdf.cu*

The text below is the code for the modifications made to the *pdf.cu* file from the previously named *pdf0.c*.

```
/*------------------------------------------------------------------------
Program pdf0.c computes a pair distribution function for n atoms
given the 3D coordinates of the atoms.
------------------------------------------------------------------------*/
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <stdlib.h>
#include <cuda.h>

#define NHBIN 2000 // Histogram size

float al[3]; // Simulation box lengths
int n;       // Number of atoms
float *r;  // Atomic position array
FILE *fp;

/*
    ADDED
*/
__constant__ float DALTH[3];
__constant__ int DN;
__constant__ float DDRH;


// float SignR(float v,float x) {if (x > 0) return v; else return -v;}
/*
    ADDED
*/
__device__ float d_SignR(float v, float x)
{
  if (x > 0)
    return v;
  else
    return -v;
}

/*
    ADDED
```

```
*/
__global__ void gpu_histogram_kernel(float *r, float *nhis)
{
  int i, j, a, ih;
  float rij, dr;

  int iBlockBegin = (DN / gridDim.x) * blockIdx.x;
  int iBlockEnd = (DN / gridDim.x) * (blockIdx.x + 1);
  if (blockIdx.x == gridDim.x - 1)
    iBlockEnd = DN;
  int jBlockBegin = (DN / gridDim.y) * blockIdx.y;
  int jBlockEnd = (DN / gridDim.y) * (blockIdx.y + 1);
  if (blockIdx.y == gridDim.y - 1)
    jBlockEnd = DN;
  for (i = iBlockBegin + threadIdx.x; i < iBlockEnd; i += blockDim.x)
  {
    for (j = jBlockBegin + threadIdx.y; j < jBlockEnd; j += blockDim.y)
    {
      if (i < j)
      {
        // Process (i,j) atom pair
        rij = 0.0;

        for (a = 0; a < 3; a++)
        {
          dr = r[3 * i + a] - r[3 * j + a];
          /* Periodic boundary condition */
          dr = dr - d_SignR(DALTH[a], dr - DALTH[a]) - d_SignR(DALTH[a], dr + DALTH[a
    ]);
          rij += dr * dr;
        }
        rij = sqrt(rij); /* Pair distance */
        ih = rij / DDRH;

        // nhis[ih] += 1.0;
        atomicAdd(&nhis[ih], 1.0); //In order to avoid race condition

      } // end if i<j
    }   // end for j
  }     // end for i
}

/*------------------------------------------------------------------------------*/
void histogram()
{
  /*
  PREVIOUS CODE NOT SHOWN
  */

  cudaMalloc((void**)&dev_r,sizeof(float)*3*n);
  cudaMalloc((void**)&dev_nhis,sizeof(float)*NHBIN);

  cudaMemcpy(dev_r,r,3*n*sizeof(float),cudaMemcpyHostToDevice);
  cudaMemset(dev_nhis,0.0,NHBIN*sizeof(float));

  cudaMemcpyToSymbol(DALTH, alth,sizeof(float)*3,0,cudaMemcpyHostToDevice);
  cudaMemcpyToSymbol(DN,&n,sizeof(int),0,cudaMemcpyHostToDevice);
  cudaMemcpyToSymbol(DDRH,&drh,sizeof(float),0,cudaMemcpyHostToDevice);

  // Compute dev_nhis on GPU: dev_r[] -> dev_nhis[]
```

```
  dim3 numBlocks(8,8,1);
  dim3 threads_per_block(16,16,1);
  gpu_histogram_kernel<<<numBlocks,threads_per_block>>>(dev_r,dev_nhis);

  cudaMemcpy(nhis,dev_nhis,NHBIN*sizeof(float),cudaMemcpyDeviceToHost);
  cudaFree(dev_r);
  cudaFree(dev_nhis);

  density = n / (al[0] * al[1] * al[2]);
  /* Print out the histogram */
  fp = fopen("pdf.d", "w");
  for (ih = 0; ih < NHBIN; ih++)
  {
    gr = nhis[ih] / (2 * M_PI * pow((ih + 0.5) * drh, 2) * drh * density * n);
    fprintf(fp, "%e %e\n", (ih + 0.5) * drh, gr);
  }
  fclose(fp);
  free(nhis);
}

/*---------------------------------------------------------------------*/
int main()
{
  /*---------------------------------------------------------------------*/
  int i;
  float cpu1, cpu2;

  /* Read the atomic position data */
  fp = fopen("pos.d", "r");
  fscanf(fp, "%f %f %f", &(al[0]), &(al[1]), &(al[2]));
  fscanf(fp, "%d", &n);
  r = (float *)malloc(sizeof(float) * 3 * n);
  for (i = 0; i < n; i++)
    fscanf(fp, "%f %f %f", &(r[3 * i]), &(r[3 * i + 1]), &(r[3 * i + 2]));
  fclose(fp);

  /* Compute the histogram */
  cpu1 = ((float)clock()) / CLOCKS_PER_SEC;
  histogram();
  cpu2 = ((float)clock()) / CLOCKS_PER_SEC;
  printf("Execution time (s) = %le\n", cpu2 - cpu1);

  free(r);
  return 0;
}
```

1.2 Plot of Pair Distribution Function

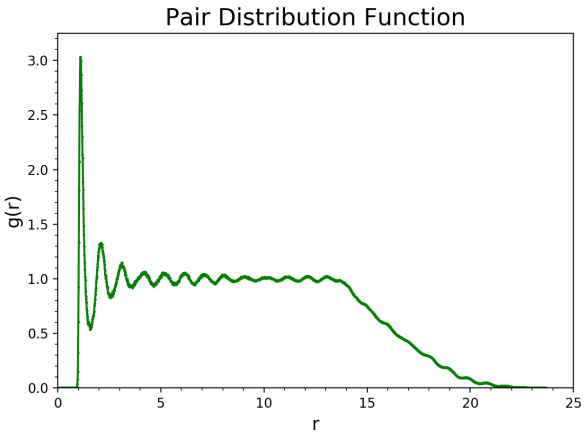

Figure 1: Plot of Pair Distribution Function

2 MPI+OpenMP+CUDA Computation of π

2.1 Source Code of *pi3.cu*

The text below is the code for the modifications made to the *pi3.cu* file from the previously named *hypi_setdevice.cu*.

```
// Hybrid MPI+OpenMP+CUDA computation of Pi
#include <stdio.h>
#include <mpi.h>
#include <omp.h> // NEW
#include <cuda.h>

#define NBIN  10000000  // Number of bins
#define NUM_DEVICE 2 // # of GPU devices = # of OpenMP threads // NEW
#define NUM_BLOCK  13  // Number of thread blocks
#define NUM_THREAD 192  // Number of threads per block

// Kernel that executes on the CUDA device
__global__ void cal_pi(float *sum,int nbin,float step,float offset,int nthreads,int
    nblocks) {
  int i;
  float x;
  int idx = blockIdx.x*blockDim.x+threadIdx.x;  // Sequential thread index across the
     blocks
  for (i=idx; i<nbin; i+=nthreads*nblocks) {  // Interleaved bin assignment to
    threads
    x = offset+(i+0.5)*step;
    sum[idx] += 4.0/(1.0+x*x);
  }
}

int main(int argc,char **argv) {
```

```
  /*
  PREVIOUS CODE NOTE SHOWN
  */

  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&myid);  // My MPI rank
  MPI_Comm_size(MPI_COMM_WORLD,&nproc);  // Number of MPI processes
  // nbin = NBIN/nproc;  // Number of bins per MPI process
  // step = 1.0/(float)(nbin*nproc);  // Step size with redefined number of bins
  // offset = myid*step*nbin;  // Quadrature-point offset
  //NEW
  omp_set_num_threads(NUM_DEVICE); // One OpenMP thread per GPU device
  nbin = NBIN/(nproc*NUM_DEVICE); // # of bins per OpenMP thread
  step = 1.0/(float)(nbin*nproc*NUM_DEVICE);

  #pragma omp parallel private(offset, sumHost, sumDev, tid, dev_used) reduction(+:pi
    )
  {
    int mpid = omp_get_thread_num();
    offset = (NUM_DEVICE*myid+mpid)*step*nbin; // Quadrature-point offset
    cudaSetDevice(mpid%2);

    // cudaSetDevice(myid%2);
    size_t size = NUM_BLOCK*NUM_THREAD*sizeof(float);  //Array memory size
    sumHost = (float *)malloc(size);  //  Allocate array on host
    cudaMalloc((void **) &sumDev,size);  // Allocate array on device
    cudaMemset(sumDev,0,size);  // Reset array in device to 0
    // Calculate on device (call CUDA kernel)
    cal_pi <<<dimGrid,dimBlock>>> (sumDev,nbin,step,offset,NUM_THREAD,NUM_BLOCK);
    // Retrieve result from device and store it in host array
    cudaMemcpy(sumHost,sumDev,size,cudaMemcpyDeviceToHost);
    // Reduction over CUDA threads
    for(tid=0; tid<NUM_THREAD*NUM_BLOCK; tid++)
      pi += sumHost[tid];
    pi *= step;
    // CUDA cleanup
    free(sumHost);
    cudaFree(sumDev);
    cudaGetDevice(&dev_used);
    printf("myid = %d; mpid = %d: device used = %d; partial pi = %f\n", myid, mpid,
    dev_used, pi);
    // printf("myid = %d: device used = %d; partial pi = %f\n",myid,dev_used,pi);
  } // end omp parallel

  // Reduction over MPI processes
  MPI_Allreduce(&pi,&pig,1,MPI_FLOAT,MPI_SUM,MPI_COMM_WORLD);
  if (myid==0) printf("PI = %f\n",pig);

  MPI_Finalize();
  return 0;
}
```

2.2 Output of *pi3.sl*

```
========================================
SLURM_JOB_ID = 6305698
SLURM_JOB_NODELIST = e07-[07-08]
TMPDIR = /tmp/SLURM_6305698
========================================
myid = 1; mpid = 1: device used = 1; partial pi = 0.567582
myid = 1; mpid = 0: device used = 0; partial pi = 0.719409
myid = 0; mpid = 1: device used = 1; partial pi = 0.874671
myid = 0; mpid = 0: device used = 0; partial pi = 0.979926
PI = 3.141588
```

Figure 2: Printout of pi3.sl