CSCI596 Assignment 6—Hybrid MPI+OpenMP+CUDA Programming

Youzhi Qu

CSCI-596 assignment4

Prof. Aiichiro Nakano

**Part 1:**

**pdf1.cu code:**

/\*----------------------------------------------------------------------

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>

#define NHBIN 2000 // Histogram size

float al[3]; // Simulation box lengths

int n; // Number of atoms

float \*r; // Atomic position array

FILE \*fp;

\_\_constant\_\_ float DALTH[3];

\_\_constant\_\_ int DN;

\_\_constant\_\_ float DDRH;

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

\_\_device\_\_ float d\_SignR(float v,float x) {if (x > 0) return v; else return -v;}

\_\_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 = min((DN/gridDim.x)\*(blockIdx.x+1),DN);

int jBlockBegin = (DN/gridDim.y)\*blockIdx.y;

int jBlockEnd = min((DN/gridDim.y)\*(blockIdx.y+1),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);

} // end if i<j

} // end for j

} // end for i

}

/\*--------------------------------------------------------------------\*/

void histogram() {

/\*----------------------------------------------------------------------

Constructs a histogram NHIS for atomic-pair distribution.

----------------------------------------------------------------------\*/

float alth[3];

float\* nhis; // Histogram array

float rhmax,drh,density,gr;

int a,ih;

float\* dev\_r; // Atomic positions

float\* dev\_nhis; // Histogram

/\* Half the simulation box size \*/

for (a=0; a<3; a++) alth[a] = 0.5\*al[a];

/\* Max. pair distance RHMAX & histogram bin size DRH \*/

rhmax = sqrt(alth[0]\*alth[0]+alth[1]\*alth[1]+alth[2]\*alth[2]);

drh = rhmax/NHBIN; // Histogram bin size

nhis = (float\*)malloc(sizeof(float)\*NHBIN);

//for (ih=0; ih<NHBIN; ih++) nhis[ih] = 0.0; // Reset the histogram

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);

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;

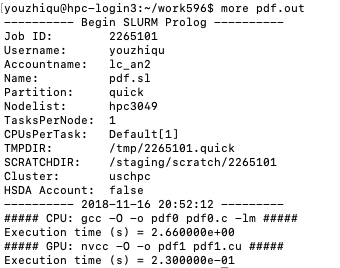
printf("Execution time (s) = %le\n",cpu2-cpu1);

free(r);

return 0;

}

**Output:**



**2:**

**Part 2:**

**pi3.cu code:**

// Hybrid MPI+OpenMP+CUDA computation of Pi

#include <stdio.h>

#include <mpi.h>

#include <omp.h>

#include <cuda.h>

#define NBIN 10000000 // Number of bins

#define NUM\_DEVICE 2 // # of GPU devices = # of OpenMP threads

#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) {

int myid,nproc,nbin,tid;

int mpid;

float step,offset,pi=0.0,pig;

dim3 dimGrid(NUM\_BLOCK,1,1); // Grid dimensions (only use 1D)

dim3 dimBlock(NUM\_THREAD,1,1); // Block dimensions (only use 1D)

float \*sumHost,\*sumDev; // Pointers to host & device arrays

int dev\_used;

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

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(mpid,offset,SumHost,sumDev,tid,dev\_used) reduction(+:pi)

{

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: device used = %d; partial pi = %f\n",myid,dev\_used,pi);

printf("myid = %d; mpid = %d: device used = %d; partial pi = %f\n", myid, mpid, 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. output:**

