```
// #include <iostream>
#include <stdio.h>
// #include <vector>
#include <cmath>
#include <cassert>
#include <cutil.h>
#include <omp.h>
#include "cuda_pointer.h"

#define NTHREAD 64 // 64, 96, 128 or 192
#define NJBLOCK 16 // 8800GTS/512 has 16
#define NIBLOCK 16 // 16 or 32
#define NIMAX (NTHREAD * NIBLOCK) // 1024

#define NBMAX 128 // NNB per block

template <class T>
struct myvector{
        int num;
        T *val;
        myvector(){
                num = 0;
                val = NULL;
        }
        ~myvector(){
                delete [] val;
        }
        void clear(){
                num = 0;
        }
        void reserve(size_t count){
                val = new T[count];
        }
        void free(){
                delete [] val;
        }
        void push_back(const T &t){
                val[num++] = t;
        }
        size_t size(){
                return num;
        }
        T &operator[](int i){
                return val[i];
        }
};

#define PROFILE
#ifdef PROFILE
#include <sys/time.h>
static double get_wtime(){
        struct timeval tv;
        gettimeofday(&tv, NULL);
        return tv.tv_sec + 1.e-6 * tv.tv_usec;
}
#else
static double get_wtime(){
        return 0.0;
}
#endif

static double time_send, time_grav;
static long long numInter;

struct Jparticle{
        float3 pos;
        float  mass;
        float3 vel;
        float  pad;
```

```
        Jparticle() {}
        Jparticle(double mj, double xj[3], double vj[3]){
                pos.x = xj[0];
                pos.y = xj[1];
                pos.z = xj[2];
                mass  = mj;
                vel.x = vj[0];
                vel.y = vj[1];
                vel.z = vj[2];
        }
};
struct Iparticle{
        float3 pos;
        float  h2;
        float3 vel;
        float  pad;
        Iparticle() {}
        Iparticle(double h2i, double xi[3], double vi[3]){
                pos.x = xi[0];
                pos.y = xi[1];
                pos.z = xi[2];
                h2    = h2i;
                vel.x = vi[0];
                vel.y = vi[1];
                vel.z = vi[2];
        }
};
struct Force{
        float3 acc;
        float  pot;
        float3 jrk;
        int    nnb;          //  8 words
        unsigned short  neib[NBMAX]; // 24 words
        __device__  Force(){
                acc.x = acc.y = acc.z = 0.f;
                jrk.x = jrk.y = jrk.z = 0.f;
                pot = 0.f;
                nnb = 0;
        }
};

__device__ float rsqrtfNR(float x){
        float y = rsqrtf(x);
        return (-0.5f * y) * (x*y*y - 3.0f);
}

#if 0
struct force1{
        float dx, dy, dz;
        float dvx, dvy, dvz;
        float r2;
        float rv;
        // __device__ force1(){}
        __device__ void calc(
                        const Iparticle &ip,
                        const Jparticle &jp){
                dx = jp.pos.x - ip.pos.x;
                dy = jp.pos.y - ip.pos.y;
                dz = jp.pos.z - ip.pos.z;
                dvx = jp.vel.x - ip.vel.x;
                dvy = jp.vel.y - ip.vel.y;
                dvz = jp.vel.z - ip.vel.z;
                r2 = dx*dx + dy*dy + dz*dz;
                rv = dx*dvx + dy*dvy + dz*dvz;
        }
};
struct force2{
        float rinv1;
        // __device__ force2(){}
        __device__ void calc(
                        const int j,
                        const Iparticle &ip,
                        const force1 &f1,
                        Force &fo){
                rinv1 = rsqrtf(f1.r2);
                if(f1.r2 < ip.h2){
                        fo.neib[fo.nnb++ % NBMAX] = j;
                        rinv1 = 0.f;
                }
        }
};
struct force3{
        float rinv1, rinv2, rinv3;
        float rv;
        // __device__ force3(){}
        __device__ void calc(
                        const Jparticle &jp,
                        const force1 &f1,
                        const force2 &f2,
                        Force &fo){
                rinv1 = f2.rinv1;
                rinv2 = rinv1 * rinv1;
                rinv1 *= jp.mass;
                rinv3 = rinv1 * rinv2;
                rv = f1.rv * -3.f * rinv2;

                fo.pot += rinv1;
                fo.acc.x += rinv3 * f1.dx;
                fo.acc.y += rinv3 * f1.dy;
                fo.acc.z += rinv3 * f1.dz;
                fo.jrk.x += rinv3 * (f1.dvx + rv * f1.dx);
                fo.jrk.y += rinv3 * (f1.dvy + rv * f1.dy);
                fo.jrk.z += rinv3 * (f1.dvz + rv * f1.dz);
        }
};
#endif

__device__ void h4_kernel(
                const int j,
                const Iparticle &ip,
                const Jparticle &jp,
                Force &fo){
        float dx = jp.pos.x - ip.pos.x;
        float dy = jp.pos.y - ip.pos.y;
        float dz = jp.pos.z - ip.pos.z;
        float dvx = jp.vel.x - ip.vel.x;
        float dvy = jp.vel.y - ip.vel.y;
        float dvz = jp.vel.z - ip.vel.z;

        float r2 = dx*dx + dy*dy + dz*dz;
        float rv = dx*dvx + dy*dvy + dz*dvz;
        float rinv1 = rsqrtf(r2);
        if(r2 < ip.h2){
                // fo.neib[fo.nnb++ % NBMAX] = j;
                fo.neib[fo.nnb & (NBMAX-1)] = (unsigned)j;
                fo.nnb++;
                rinv1 = 0.f;
        }
        float rinv2 = rinv1 * rinv1;
        float mrinv1 = jp.mass * rinv1;
        float mrinv3 = mrinv1 * rinv2;
        rv *= -3.f * rinv2;

#ifdef POTENTIAL
        fo.pot += mrinv1;
#endif
        fo.acc.x += mrinv3 * dx;
        fo.acc.y += mrinv3 * dy;
```

```
        fo.acc.z += mrinv3 * dz;
        // fo.acc.z += 1.0;
        fo.jrk.x += mrinv3 * (dvx + rv * dx);
        fo.jrk.y += mrinv3 * (dvy + rv * dy);
        fo.jrk.z += mrinv3 * (dvz + rv * dz);
}
__global__ void h4_gravity(
                int nbody,
                Iparticle ipbuf[],
                Jparticle jpbuf[],
                Force fobuf[][NJBLOCK]){
        int ibid = blockIdx.x;
        int jbid = blockIdx.y;
        int tid = threadIdx.x;
        int iaddr = tid + NTHREAD * ibid;
        int jstart = (nbody * (jbid  )) / NJBLOCK;
        int jend   = (nbody * (jbid+1)) / NJBLOCK;

        Iparticle ip = ipbuf[iaddr];
        Force fo;
        for(int j=jstart; j<jend; j+=NTHREAD){
                __shared__ Jparticle jpshare[NTHREAD];
                __syncthreads();
#if 0
                jpshare[tid] = jpbuf[j+tid];
#else
                float4 *src = (float4 *)&jpbuf[j];
                float4 *dst = (float4 *)jpshare;
                dst[        tid] = src[        tid];
                dst[NTHREAD+tid] = src[NTHREAD+tid];
#endif
                __syncthreads();

                if(jend-j < NTHREAD){
                        for(int jj=0; jj<jend-j; jj++){
                                Jparticle jp = jpshare[jj];
                                h4_kernel(j+jj, ip, jp, fo);
                        }
                }else{
#pragma unroll
                        for(int jj=0; jj<NTHREAD; jj++){
                                Jparticle jp = jpshare[jj];
                                h4_kernel(j+jj, ip, jp, fo);
                        }
                }
        }
        fobuf[iaddr][jbid] = fo;
}

#if 0
static Jparticle *jp_host, *jp_dev;
static Iparticle *ip_host, *ip_dev;
static Force (*fo_host)[NJBLOCK], (*fo_dev)[NJBLOCK];
#else
static cudaPointer <Jparticle> jpbuf;
static cudaPointer <Iparticle> ipbuf;
static cudaPointer <Force[NJBLOCK]> fobuf;
#endif
#define MAX_CPU 1
static myvector<int> nblist[MAX_CPU];
static int nbody, nbodymax;
// static int *nblist;

void GPUNB_open(int nbmax){
        time_send = time_grav = 0.0;
        numInter = 0;
    // CUT_DEVICE_INIT();
        // size_t jpsize = nbmax * sizeof(Jparticle);
        // size_t ipsize = NIMAX * sizeof(Iparticle);
        // size_t fosize = NIBLOCK * NJBLOCK * NTHREAD * sizeof(Force);
        // cudaMallocHost((void **)&jp_host, jpsize);
        // jpsize += NTHREAD * sizeof(Jparticle);
        // cudaMalloc    ((void **)&jp_dev , jpsize);
        // cudaMallocHost((void **)&ip_host, ipsize);
        // cudaMalloc    ((void **)&ip_dev , ipsize);
        // cudaMallocHost((void **)&fo_host, fosize);
        // cudaMalloc    ((void **)&fo_dev , fosize);
        jpbuf.allocate(nbmax + NTHREAD);
        ipbuf.allocate(NIMAX);
        fobuf.allocate(NIMAX);
        nbodymax = nbmax;
#pragma omp parallel
        {
/* int tid = omp_get_thread_num();   */
                int tid = 0;
                nblist[tid].reserve(nbmax);
        }
}
void GPUNB_close(){
        // cudaFreeHost(jp_host);
        // cudaFree    (jp_dev);
        // cudaFreeHost(ip_host);
        // cudaFree    (ip_dev);
        // cudaFreeHost(fo_host);
        // cudaFree    (fo_dev);
        jpbuf.free();
        ipbuf.free();
        fobuf.free();
        nbodymax = 0;

#ifdef PROFILE
#if 0
        std::cerr << "***********************" << std::endl;
        std::cerr << "time send : " << time_send << " sec " << std::endl;
        std::cerr << "time grav : " << time_grav << " sec " << std::endl;
        std::cerr << 60.e-9 * numInter / time_grav << " Gflops (gravity part onl
y)" << std::endl;
        std::cerr << "***********************" << std::endl;
#else
        fprintf(stderr, "***********************\n");
        fprintf(stderr, "time send : %f sec\n", time_send);
        fprintf(stderr, "time grav : %f sec\n", time_grav);
        fprintf(stderr, "%f Gflops (gravity part only)\n", 60.e-9 * numInter / t
ime_grav);
        fprintf(stderr, "***********************\n");
#endif
#endif
}
void GPUNB_send(
                int nj,
                double mj[],
                double xj[][3],
                double vj[][3]){
        time_send -= get_wtime();
        nbody = nj;
        assert(nbody <= nbodymax);
        for(int j=0; j<nj; j++){
                // jp_host[j] = Jparticle(mj[j], xj[j], vj[j]);
                jpbuf[j] = Jparticle(mj[j], xj[j], vj[j]);
        }
        // size_t jpsize = nj * sizeof(Jparticle);
        // cudaMemcpy(jp_dev, jp_host, jpsize, cudaMemcpyHostToDevice);
        jpbuf.htod(nj);
        time_send += get_wtime();
}
void GPUNB_regf(
                int ni,
                double h2[],
```

```
                double xi[][3],
                double vi[][3],
                double acc[][3],
                double jrk[][3],
                double pot[],
                int lmax,
                int nbmax,
                int *listbase){
        time_grav -= get_wtime();
        numInter += ni * nbody;
        assert(0 < ni && ni <= NIMAX);

/*        printf(" ni lm %d %d %d \t %e %e %e\n",ni, lmax, nbmax, h2[0], xi[0][0
], vi[0][0]);*/

        for(int i=0; i<ni; i++){
                // ip_host[i] = Iparticle(h2[i], xi[i], vi[i]);
                ipbuf[i] = Iparticle(h2[i], xi[i], vi[i]);
        }
        // set i-particles
        // size_t ipsize = ni * sizeof(Iparticle);
        // cudaMemcpy(ip_dev, ip_host, ipsize, cudaMemcpyHostToDevice);
        ipbuf.htod(ni);

        // gravity kernel
        int niblock = 1 + (ni-1) / NTHREAD;
        dim3 grid(niblock, NJBLOCK, 1);
        dim3 threads(NTHREAD, 1, 1);
#if 0
        int sharedMemSize = NTHREAD * sizeof(Jparticle);
        h4_gravity <<< grid, threads, sharedMemSize >>>
                (nbody, ip_dev, jp_dev, fo_dev);
#else
        // h4_gravity <<< grid, threads >>>
        //      (nbody, ip_dev, jp_dev, fo_dev);
        h4_gravity <<< grid, threads >>>
                (nbody, ipbuf, jpbuf, fobuf);
#endif

        // recieve force
        // size_t fosize = ni * NJBLOCK * sizeof(Force);
        // cudaMemcpy(fo_host, fo_dev, fosize, cudaMemcpyDeviceToHost);
        fobuf.dtoh(ni);

        // reduction phase
#pragma omp parallel for
        for(int i=0; i<ni; i++){
/* int tid = omp_get_thread_num();   */
                int tid = 0;
                double ax=0, ay=0, az=0;
                double jx=0, jy=0, jz=0;
#ifdef POTENTIAL
                double poti=0;
#endif
                for(int jb=0; jb<NJBLOCK; jb++){
                        // Force &fo = fo_host[i][jb];
                        Force &fo = fobuf[i][jb];
                        ax += fo.acc.x;
                        ay += fo.acc.y;
                        az += fo.acc.z;
                        jx += fo.jrk.x;
                        jy += fo.jrk.y;
                        jz += fo.jrk.z;
#ifdef POTENTIAL
                        poti += fo.pot;
#endif
                }
                acc[i][0] = ax;
                acc[i][1] = ay;
                acc[i][2] = az;
                jrk[i][0] = jx;
                jrk[i][1] = jy;
                jrk[i][2] = jz;
                // fprintf(stderr, "%f %f %f %f %f %f\n", ax, ay, az, jx, jy, jz
);
                // exit(0);
#ifdef POTENTIAL
                pot[i] = poti;
#endif
                bool overflow = false;
                nblist[tid].clear();
                for(int jb=0; jb<NJBLOCK; jb++){
                        // Force &fo = fo_host[i][jb];
                        Force &fo = fobuf[i][jb];
                        int jstart = (nbody * jb) / NJBLOCK;
                        if(fo.nnb <= NBMAX){
                                for(int k=0; k<fo.nnb; k++){
                                        int nb = fo.neib[k];
/*                       printf(" 1 %d %d %d %d %d %d %d %d\n",tid,k,nb, jb, jst
art,nbody,fo.nnb,fo.neib[k]); */
                                        while(nb < jstart) nb += (1<<16);
/*                       printf(" 2 %d %d %d %d\n",nb, jb, jstart,nbody); */
                                        nblist[tid].push_back(nb);
/*                       printf(" 3 %d %d %d\n",nb, jb, jstart,nbody); */
                                        // nblist.push_back(fo.neib[k]);
                                }
                        }else{
                                overflow = true;
                        }
                }
                int *nnbp = listbase + lmax * i;
                int *nblistp = nnbp + 1;
                int nnb = nblist[tid].size();
                if(nnb > nbmax) overflow = true;
                // assert(!overflow);
                if(overflow){
                        *nnbp = -1;
                }else{
                        *nnbp = nnb;
                        for(int k=0; k<nnb; k++){
                                nblistp[k] = nblist[tid][k];
                        }
                }
        }
#if 0
        if(ni > 0){
                FILE *fp = fopen("Force.gpu", "w");
                assert(fp);
                for(int i=0; i<ni; i++){
                        int nnb =  listbase[i*lmax];
                        fprintf(fp, "%d %9.2e %9.2e %9.2e %9.2e %9.2e %9.2e %d\n
",
                                        i, acc[i][0], acc[i][1], acc[i][2],
                                           jrk[i][0], jrk[i][1], jrk[i][2], nnb)
;
                }
                fprintf(fp, "\n");
                fclose(fp);
                exit(1);
        }
#endif
        time_grav += get_wtime();
}

extern "C" {
        void gpunb_open_(int *nbmax){
                GPUNB_open(*nbmax);
        }
```

```
        void gpunb_close_(){
                GPUNB_close();
        }
        void gpunb_send_(
                        int *nj,
                        double mj[],
                        double xj[][3],
                        double vj[][3]){
                GPUNB_send(*nj, mj, xj, vj);
        }
        void gpunb_regf_(
                        int *ni,
                        double h2[],
                        double xi[][3],
                        double vi[][3],
                        double acc[][3],
                        double jrk[][3],
                        double pot[],
                        int *lmax,
                        int *nbmax,
                        int *list){ // list[][lmax]
                GPUNB_regf(*ni, h2, xi, vi, acc, jrk, pot, *lmax, *nbmax, list);
        }
}
```