

Метод LBE для газодинамических течений Lattice Boltzman simulation flow

Э.Р. Прууэл, А. Л. Куперштох,
Д.И. Карпов, Д.А. Медведев.

Институт гидродинамики им. М. А. Лаврентьева

16 марта 2012 г.

Содержание

1	Массивы с автоматическим распределением данных на нескольких GPU	1
2	CUP препроцессор	4
3	Мультифизические процессы	5
3.1	Простое LBE	5
4	Приложение	6
4.1	Решение уравнения Лапласа с ручным распараллеливанием	6
4.2	Решения уравнения Лапласа с автоматическим распараллеливанием	8

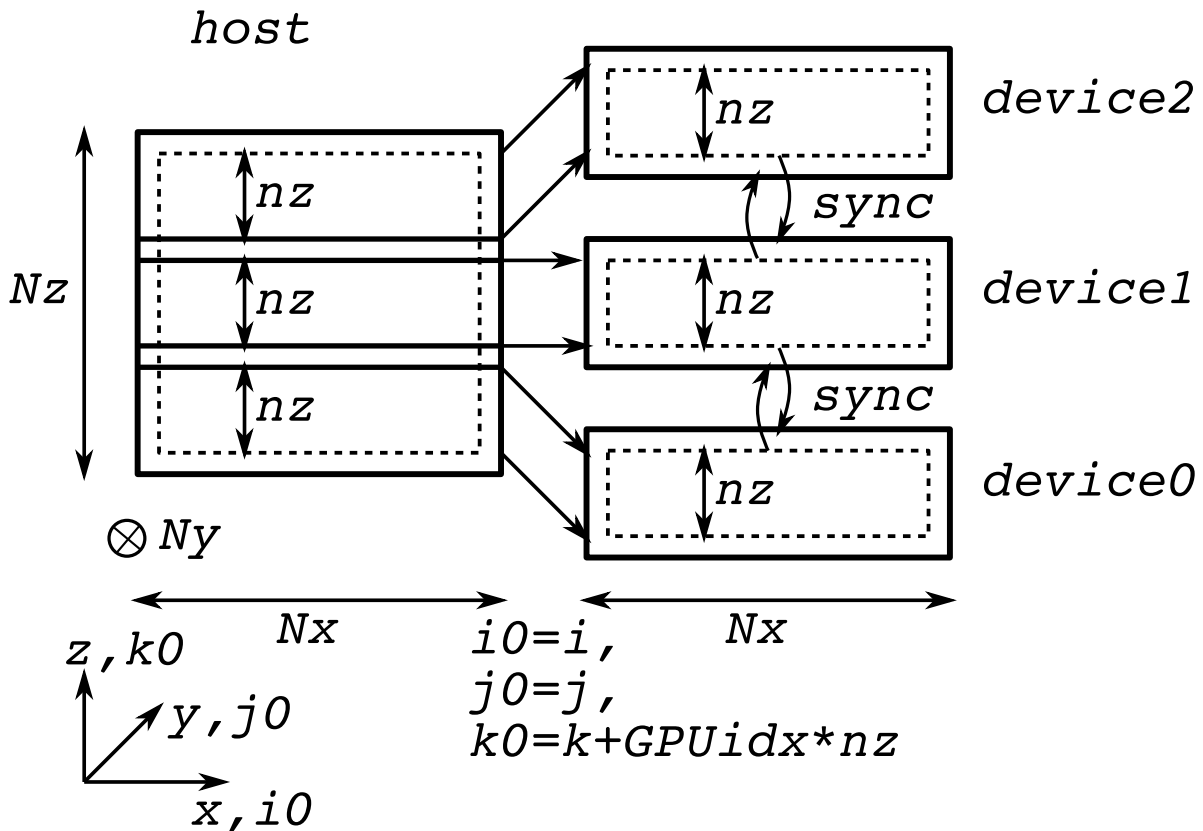


Рис. 1. Схема распределения массива данных (3d, 2d) на нескольких GPU.

1 Массивы с автоматическим распределением данных на нескольких GPU

Библиотека позволяет создавать 3d массивы с автоматическим распределением данных на нескольких графических устройствах (device) одного узла (host). Для этого используются два специализированных класса: `d_Array3d_base` – часть данных (слой по оси z) общего массива хранящаяся в памяти одного устройства; `d_Array3d` – набор слоев по оси z формирующих весь набор данных (рис. 1).

Первый класс непосредственно содержит блок данных в форме параллелепипеда с полным количеством узлов по осям x и y и часть данных по оси z . Самый низкий уровень иерархии представления данных, создается автоматически, непосредственно производит действия над данными при работе параллельных алгоритмов.

Второй класс предоставляет пользователю удобный интерфейс отвечающий за весь набор данных. Создается пользователем с необходимыми

размерами расчетной области. При работе алгоритмом автоматически параллельно запускает функции обработки для всех частей находящихся на разных устройствах.

Приведем небольшой пример параллельного алгоритма работающего на нескольких GPU.

```
int gpu_array[2]={0,1}; // массив с номерами устройств GPU (
    device)
MP_Parallel MP_parallel(gpu_array, 2);

__global__ void f_parallel(d_Array3d_base<double> a){
    const size_t i=blockIdx.x*blockDim.x+threadIdx.x;
    const size_t j=blockIdx.y*blockDim.y+threadIdx.y;
    const size_t k=blockIdx.z*blockDim.z+threadIdx.z;
    const size_t ind=k*a.Nx*a.Ny+j*a.Nx+i;
    a[ind]=1.0;
}

int main(){
    //Массив вещественных с распределением памяти на нескольких GPU
    d_Array3d<double> a(256, 256, 256); // Nx, Ny, Nz

    dim3 threads(MP_parallel.Nx, MP_parallel.Ny, MP_parallel.Nz);
    dim3 grid(a.Nx/threads.x, a.Ny/threads.y, ((a.Nz-2)/MP_parallel
        .GPU_pool.size()+2)/threads.z);
    cudaStream_t s[4]; // больше GPU/host не бывает :)
    for(size_t i=0; i<MP_parallel.GPU_pool.size(); i++){
        cudaSetDevice(MP_parallel.GPU_pool[i]);
        cudaStreamCreate(&s[i]);
        f_parallel<<<grid, threads, 0, s[i]>>>(split(i, a));
    }
    for(size_t i=0; i<MP_parallel.GPU_pool.size(); i++){
        cudaSetDevice(MP_parallel.GPU_pool[i]);
        cudaStreamDestroy(s[i]);
    }

    sync(a); // синхронизовать границы блоков
}
```

2 CUP препроцессор

CUP (CUDA parallel) – препроцессор генерирует код объявления функции и код параллельного вызова.

- CUP INSERT *prefix* – место вставки объявлений функций. *prefix* – префикс в именах создаваемых функций.
- CUP FORALL_3D *var1 expr1 var2 expr2 ... # code #* – параллельный цикл по всем элементам массива *expr1*.
- CUP FORALL_3D_IN *var1 expr1 var2 expr2 ... # code #* – параллельный цикл по всем внутренним элементам массива *expr1*.
- CUP FORALL_3D_IJK *var1 expr1 var2 expr2 ... # code #* – параллельный цикл по всем элементам массива *expr1*. Дополнительные доступные переменные: *i*, *j*, *k* – индексы ячейки по осям *x*, *y*, *z* соответственно.
- CUP FORALL_3DYZ *var1 expr1 var2 expr2 ... # code #* – параллельный цикл по ячейкам плоскости $i=0$ ($x = 0$).
- CUP FORALL_3DXZ *var1 expr1 var2 expr2 ... # code #* – параллельный цикл по ячейкам плоскости $j=0$ ($y = 0$).
- CUP FORALL_3DXY *var1 expr1 var2 expr2 ... # code #* – параллельный цикл по ячейкам плоскости $z=0$ ($z = 0$).

Все функции исполняют код *code*, внутри доступны переменные: *var1*, *var2*, ... , *ind* – индекс, ячейки массива, *gpuIdx* – номер обрабатываемого слоя по оси *z*.

Небольшой пример параллельного алгоритма работающего на нескольких GPU с использованием препроцессора CUP.

```
// массив с номерами устройств GPU (device)
int  gpu_array[2]={0,1};
MP_Parallel MP_parallel(gpu_array, 2);
```

```
// Обработывается препроцессором.
// В этом месте появятся определения функций
CUP INSERT cup_test
```

```
int main(){
    //Массив вещественных с распределением памяти на нескольких GPU
```

```

d_Array3d<double> a(256, 256, 256); // Nx, Ny, Nz

CUP FORALL_3D a a #
    a[ind]=1.0;
#
sync(a);
}

```

3 Мультифизические процессы

3.1 Простое LBE

```

class d_LB3d_base{
public:
    size_t Nx, Ny, Nz;
    Array3d_base<LBE_T> rho, ux, uy, uz;
    d_Array3d_base<LBE_T> f0, f1, f2, f3, f4, f5, f6, f7, f8, f9,
        f10, f1u, f2u, f3u, f4u, f1d, f2d, f3d, f4d;
    d_Array3d_base<LBE_T> ftmp;
    ...
}

class d_LB3d{
public:
    size_t Nx, Ny, Nz;
    d_Array3d<LBE_T> rho, ux, uy, uz;
    d_Array3d<LBE_T> f0, f1, f2, f3, f4, f5, f6, f7, f8, f9, f10,
        f1u, f2u, f3u, f4u, f1d, f2d, f3d, f4d;
    d_Array3d<LBE_T> ftmp;
    void free(){
        ...
    }
}

d_LB3d_base split(size_t i, d_LB3d & a)
void sync(d_LB3d & a)
__device__ void set_value(d_LB3d_base & lb, size_t ind, LBE_T rho,
LBE_T ux, LBE_T uy, LBE_T uz)
void setrhoul(d_LB3d & lb, LBE_T rho=1.0, LBE_T ux=0.0, LBE_T uy=0.0,
LBE_T uz=0.0)
void rhoul2f(d_LB3d & lb)
void transfer(d_LB3d & lb)
void periodicx(d_LB3d & lb)
void periodicy(d_LB3d & lb)

```



```

const size_t i=blockIdx.x*blockDim.x+threadIdx.x;
const size_t j=blockIdx.y*blockDim.y+threadIdx.y;
const size_t k=blockIdx.z*blockDim.z+threadIdx.z;
if (i>0 && i < a.Nx-1 && j>0 && j < a.Ny-1 && k>0 && k < a.Nz
-1){
    const size_t ind=k*a.Nx*a.Ny+j*a.Nx+i;
    tmp[ind]=m*(a[ind-1]+a[ind+1]+a[ind-Nx]+a[ind+Nx]+a[ind-Nxy
]+a[ind+Nxy]);
}
}

int main() {
    d_Array3d<T> a(Nx, Ny, Nz), tmp(Nx, Ny, Nz); // Массивы
    вещественных с распределением памяти на нескольких GPU

    {
        dim3 threads(MP_parallel.Nx, MP_parallel.Ny, MP_parallel.Nz)
        ;
        dim3 grid(a.Nx/threads.x, a.Ny/threads.y, ((a.Nz-2)/
MP_parallel.GPU_pool.size()+2)/threads.z);
        cudaStream_t s[4]; // больше GPU/host не бывает
        for(size_t i=0; i<MP_parallel.GPU_pool.size(); i++){
            cudaSetDevice(MP_parallel.GPU_pool[i]);
            cudaStreamCreate(&s[i]);
            laplas_init<<<<grid, threads, 0, s[i]>>>(i, split(i, a),
split(i, tmp));
        }
        for(size_t i=0; i<MP_parallel.GPU_pool.size(); i++){
            cudaSetDevice(MP_parallel.GPU_pool[i]);
            cudaStreamDestroy(s[i]);
        }
    }
    sync(a);

    for(int i=0; i<100; i++){
        dim3 threads(MP_parallel.Nx, MP_parallel.Ny, MP_parallel.Nz)
        ;
        dim3 grid(a.Nx/threads.x, a.Ny/threads.y, ((a.Nz-2)/
MP_parallel.GPU_pool.size()+2)/threads.z);
        cudaStream_t s[4];
        for(size_t i=0; i<MP_parallel.GPU_pool.size(); i++){
            cudaSetDevice(MP_parallel.GPU_pool[i]);
            cudaStreamCreate(&s[i]);
            laplas_base<<<<grid, threads, 0, s[i]>>>(split(i, a),
split(i, tmp));
        }
        for(size_t i=0; i<MP_parallel.GPU_pool.size(); i++){
            cudaSetDevice(MP_parallel.GPU_pool[i]);

```

```

        cudaStreamDestroy(s[i]);
    }
    sync(tmp);
    swap(a, tmp);
}

h_Array3d<T> c(Nx, Ny, Nz); // массив данных на хосте
cpy(c, a);
{
    ofstream ofs("1.dat");
    for (int k=0; k<c.nz(); k++){
        for (int i=0; i<c.ny(); i++)
            ofs << c(c.nx()/2, i, k) << '\t';

        ofs << "\n";
    }
}
return 0;
}

```

4.2 Решения уравнения Лапласа с автоматическим распараллеливанием

Пример решения уравнения Лапласа с автоматическим распараллеливанием.

```

// laplas3d_low_level/lapl3d.cup
#include <iostream>
#include <fstream>

#include <mp/mp.h>
#include <mp/rdtsc.h>
#include <mp/d_array3d.h>

const int Nx=256; // кратно 16
const int Ny=256; // кратно 16
const int Nz=256; // // Nz-2 кратно количеству GPU
const int Nxy=Nx*Ny;

using namespace std;

typedef double T;
#define m 0.166666666666666666666666666666666667

int a[2]={0,0}; // массив с номерами устройств GPU (device)
MP_Parallel MP_parallel(a, 2);

```



```

// Обрабатывается препроцессором. В этом месте появятся
// определения функций
CUP INSERT lapl3d

int main() {
    d_Array3d<T> a(Nx, Ny, Nz), tmp(Nx, Ny, Nz); // Массивы
        вещественных с распределением памяти на нескольких GPU

    // начальные и граничные условия
    CUP FORALL_3D_IJK a a tmp tmp #
        const size_t k0=k+gpuIdx*(a.Nz-2);
        if (i>a.Nx/3 && i < 2*a.Nx/3 && j>a.Ny/3 && j < 2*a.Ny/3 &&
            k0>Nz/3 && k0 < 2*Nz/3)
            tmp[ind]=a[ind]=k0;
        else tmp[ind]=a[ind]=0.0;
    #

    // итерации
    for(int i=0; i<100; i++){
        CUP FORALL_3D_IN a a tmp tmp # tmp[ind]=m*(a[ind-1]+a[ind
            +1]+a[ind-Nx]+a[ind+Nx]+a[ind-Nxy]+a[ind+Nxy]); #
        sync(tmp);
        swap(a, tmp);
    }

    h_Array3d<T> c(Nx, Ny, Nz);
    cpy(c, a);
    {
        ofstream ofs("l.dat");
        for (int k=0; k<c.nz(); k++){
            for (int i=0; i<c.ny(); i++)
                ofs << c(c.nx()/2, i, k) << '\t';
            ofs << "\n";
        }
    }

    return 0;
}

```