Решение задачи *N*-тел на GPU средствами NVIDIA и визуализация на основе OpenGL

ЛАБОРАТОРНАЯ РАБОТА

http://www.kimrt.ru
Онлайн-курс
Суперкомпьютерные технологии в задачах моделирования

Рассмотрим задачу движения *N*-тел за счёт гравитационного взаимодействия. Полагаем, что все тела – точки с одинаковой массой. Полную силу, действующую на *i*-е тело, можно описать формулой

$$\mathbf{F}_i = \sum_{j=1}^{N} \frac{C}{\left|\mathbf{r}_i - \mathbf{r}_j\right|^3}\left(\mathbf{r}_i - \mathbf{r}_j\right), \qquad (1)$$

где C – это некоторая константа, связанная с массой и гравитационной постоянной, для простоты полагаем $C = 1$ [1, 2]. Здесь $\mathbf{r}$ – вектор координат тела, индекс i обозначает текущее тело и j – все остальные. Согласно этой формуле воздействие тела самого на себя исключается автоматически, так как в этом случае происходит умножение на $r_i - r_j = 0$ (при $j = i$). Для этой задачи можно записать систему уравнений [2]

$$\begin{cases} \dfrac{\partial \mathbf{v}_i}{\partial t} = \mathbf{F}_i, \\ \dfrac{\partial \mathbf{r}_i}{\partial t} = \mathbf{v}_i. \end{cases}$$

Здесь $\mathbf{v}_i$ – скорость i-го тела и t – время. Используя разностную схему Эйлера, запишем эту систему в дискретном виде

$$\begin{cases} \dfrac{\mathbf{v}_i^n - \mathbf{v}_i^o}{\Delta t} = \mathbf{F}_i^o, \\ \dfrac{\mathbf{r}_i^n - \mathbf{r}_i^o}{\Delta t} = \mathbf{v}_i^n. \end{cases} \qquad (2)$$

Временной шаг Δt – заданная величина, $\Delta t \to 0$. Верхние индексы n и o обозначают новый и старый временные слой, соответственно. Численный метод решения этой задачи можно записать следующим образом.

1. Случайным образом генерируем начальные положения и скорости для всех тел: $\mathbf{r}_i^o$ и $\mathbf{v}_i^o$ для $i = 0, 1, \ldots N - 1$.
2. Запускаем цикл по временным шагам.
    a. Вычисляем $\mathbf{F}_i^o$ из (1) для каждого тела.

b. Находим $\mathbf{v}_i^n$ из первой формулы в (2).
c. Определяем $\mathbf{r}_i^n$ из второй формулы в (2).
d. Копируем значения из $\mathbf{r}_i^n$ в $\mathbf{r}_i^o$ и из $\mathbf{v}_i^n$ в $\mathbf{v}_i^o$.
e. При необходимости с определённой периодичностью записываем промежуточные данные в файл. Если счётчик временных шагов не достиг требуемого значения, то увеличиваем значение счётчика на единицу и повторяем цикл ещё раз (переход к пункту 2a) для расчёта следующего временного слоя, иначе прерываем цикл.

3. Освобождаем память и завершаем программу.

В листинге 1 приведён пример возможной реализации алгоритма программы для расчёта с использованием графического процессора. Информацию по установке и настройке CUDA см. в [3, 4].

Листинг 1. Алгоритм расчёта движения тел

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

#define EPS 0.00000001f
#define BLOCK_SIZE 256

const int N = 16 * 1024;

//подобрать подходящие значения этих параметров вам предлагается
самостоятельно
int NumOfTimeSteps = 10001;
int periodSize = 1000;
float TimeStepSize = 0.01f;

__global__ void integrateBodies(float4* newPos, float4* newVel,\
float4* oldPos, float4* oldVel, float dt)
{
     int index = blockIdx.x * blockDim.x + threadIdx.x;
     float4 pos = oldPos[index];
     float4 f = make_float4(0.0f, 0.0f, 0.0f, 0.0f);

     for (int i = 0; i < N; i++)
     {
          float4 pi = oldPos[i];
          float4 r;
          // вектор от текущей точки к pi
          r.x = pi.x - pos.x;
          r.y = pi.y - pos.y;
          r.z = pi.z - pos.z;
          // используем EPS, чтобы не было деления на ноль
```

```
            float invDist = 1.0f / sqrtf(r.x * r.x + r.y * r.y + r.z\
* r.z + EPS);
            float s = invDist * invDist * invDist;
            // добавляем к сумме всех сил силу, вызванную i-м телом
            f.x += r.x * s;
            f.y += r.y * s;
            f.z += r.z * s;
        }

        float4 vel = oldVel[index];
        // корректируем скорость и положение тела
        vel.x += f.x * dt;
        vel.y += f.y * dt;
        vel.z += f.z * dt;
        pos.x += vel.x * dt;
        pos.y += vel.y * dt;
        pos.z += vel.z * dt;

        //Подумайте, как можно здесь реализовать замкнутую область с
периодическими граничными условиями?

        newPos[index] = pos;
        newVel[index] = vel;
}

void randomInit(float4* a, int n)
{
        for (int i = 0; i < n; i++)
        {
            a[i].x = rand() / (float)RAND_MAX - 0.5f;
            a[i].y = rand() / (float)RAND_MAX - 0.5f;
            a[i].z = rand() / (float)RAND_MAX - 0.5f;
        }
}

int main(int argc, char* argv[])
{
        float4* p = new float4[N];
        float4* v = new float4[N];
        float4* pDev[2] = { NULL, NULL };
        float4* vDev[2] = { NULL, NULL };
        cudaEvent_t start, stop;
        float gpuTime = 0.0f;
        int index = 0;

        randomInit(p, N);
        randomInit(v, N);

        cudaEventCreate(&start);
        cudaEventCreate(&stop);
```

```
	cudaEventRecord(start, 0);

	cudaMalloc((void**)&pDev[0], N * sizeof(float4));
	cudaMalloc((void**)&vDev[0], N * sizeof(float4));
	cudaMalloc((void**)&pDev[1], N * sizeof(float4));
	cudaMalloc((void**)&vDev[1], N * sizeof(float4));

	//копируем начальные данные на видеокарту
	cudaMemcpy(pDev[0], p, N * sizeof(float4),\
cudaMemcpyHostToDevice);
	cudaMemcpy(vDev[0], v, N * sizeof(float4),\
cudaMemcpyHostToDevice);

	//здесь для index используется поразрядное исключающее или ^,
	//то есть index на каждой итерации инвертируется,
	//чтобы чередовать предыдущий и последующий временной слой
	for (int TimeStepCount = 0; TimeStepCount < NumOfTimeSteps;\
TimeStepCount++, index ^= 1)
	{
		integrateBodies << <dim3(N / BLOCK_SIZE), dim3(BLOCK_SIZE)
>> > (pDev[index ^ 1], vDev[index ^ 1], \
			pDev[index], vDev[index], TimeStepSize);
		if (TimeStepCount % periodSize == 0)
		{
			FILE* pFile;
			char fName[] = "data_";
			char str[10];
			sprintf(str, "%d", TimeStepCount);
			strcat(fName, str);
			strcat(fName, ".txt");
			pFile = fopen(fName, "w");

			cudaMemcpy(p, pDev[index ^ 1], N * sizeof(float4),\
cudaMemcpyDeviceToHost);
			cudaMemcpy(v, vDev[index ^ 1], N * sizeof(float4),\
cudaMemcpyDeviceToHost);

//Такой способ записи данных в файлы очень медленный и является
"бутылочным горлышком".
//Подумайте, как это можно реализовать более эффективно?
			for (int i = 0; i < N; i++)
			{
				fprintf(pFile, "%.2f\t", p[i].x);
				fprintf(pFile, "%.2f\t", p[i].y);
				fprintf(pFile, "%.2f\n", p[i].z);
			}

			fclose(pFile);
		}

	}
```

```
    cudaFree(pDev[0]);
    cudaFree(vDev[0]);
    cudaFree(pDev[1]);
    cudaFree(vDev[1]);

    cudaEventRecord(stop, 0);
    cudaEventSynchronize(stop);
    cudaEventElapsedTime(&gpuTime, start, stop);

    printf("Elapsed time: %.2f\n", gpuTime);
    delete p;
    delete v;

    return 0;
}
```

Задание

Оптимизируйте код программы, к примеру, запись данных можно реализовать более эффективно, чтобы сохранение результатов не являлось «бутылочным горлышком». Это позволит значительно увеличить скорость расчёта. Реализуйте периодические граничные условия для области в виде шара или куба. Визуализируйте динамику тел средствами библиотеки OpenGL [5]. Поэкспериментируйте с параметрами задачи. Подумайте, какое малое значение временно шага следует взять для расчётов и почему. Реализуйте последовательную версию программы и сравните время расчёта для двух реализаций. Как зависит это время от *N* для последовательной и параллельной программы?

http://www.kimrt.ru/index/course_stm/0-24

Проект реализуется победителем Конкурса на предоставление грантов преподавателям магистратуры 2020/2021 благотворительной программы «Стипендиальная программа Владимира Потанина» Благотворительного фонда Владимира Потанина.

Источники

1. Боресков А.В., Харламов А.А. Основы работы с технологией CUDA. – М.: ДМК Пресс, 2010. – 232 с.: ил. ISBN 978-5-94074-578-5 https://books.google.ru/books?id=rLjNCQAAQBAJ&pg=PA56&lpg=PA56&dq=integrateBodies+randomInit&source=bl&ots=9elR7Uh9Vh&sig=ACfU3U07NMIcPTq5dPSpc5QoEVV5thUCjw&hl=ru&sa=X&pli=1#v=onepage&q=integrateBodies%20randomInit&f=false

2. Колегов К.С. Лекция 2. Методы моделирования: Часть 2. Онлайн-курс «Суперкомпьютерные технологии в задачах моделирования». 2022. URL: http://www.kimrt.ru/_ld/1/142_2.Simulation_on.pdf (дата обращения: 30.03.2023 г.).
3. Золотарев П.А., Колегов К.С. Установка CUDA toolkit: самостоятельная работа. Онлайн-курс «Суперкомпьютерные технологии в задачах моделирования». 2022. URL: http://www.kimrt.ru/courses/stm/self_work/Cuda_Toolkit.pdf (дата обращения: 30.03.2023 г.).
4. Золотарев П.А., Колегов К.С. Установка CUDA под операционной системой Linux Debian 11: самостоятельная работа. Онлайн-курс «Суперкомпьютерные технологии в задачах моделирования». 2022. URL: http://www.kimrt.ru/courses/stm/self_work/cudaInDebian.pdf (дата обращения: 30.03.2023 г.).
5. Золотарев П.А., Колегов К.С. Визуализация моделирования средствами языка C++ и библиотеки OpenGl на примере клеточного автомата «Игра жизнь». Онлайн-курс «Суперкомпьютерные технологии в задачах моделирования». 2022. URL: http://www.kimrt.ru/courses/stm/self_work/Game_of_Life_visualization_via_OpenGL.pdf (дата обращения: 30.03.2023 г.).