Московский Авиационный Институт
(Национальный Исследовательский Университет)

Курсовой проект по курсу
«Программирование графических процессоров»

Численное моделирование гравитационной задачи N-тел на GPU с использованием технологии CUDA

Студент:
Колесников Е. В.

Преподаватели:
Крашенинников К. Г.
Морозов А. Ю.

26 июня 2016 г.

Содержание

1 Введение

1.1 Аннотация

В данной работе описываются и сравниваются между собой три различные реализации численного моделирования гравитационной задачи N-тел на GPU с использованием технологии CUDA:

1. Реализация All-pairs algorithm на CPU,
2. Реализация All-pairs algorithm на GPU,
3. Barnes-Hut simulation на GPU.

1.2 Описание задачи

Численное моделирование задачи N-тел аппроксимирует эволюцию системы тел, в которой все тела непрерывно взаимодействуют друг с другом. Самым известным примером, который рассматривается в данной работе, является гравитационная задача N-тел, возникающая в небесной механике, где "тела" являются либо галактиками, либо звездами, между которыми возникает гравитационное взаимодействие. Однако задача N-тел встречается во многих других науках:

- При изучении сворачивания белков численное решение задачи N-тел используется для вычисления электростатических и Ван дер Ваальсовских сил,
- Задача N-тел является математической моделью таких физических процессов, как течение турбулентной жидкости или освещение.

1.3 Постановка задачи

Пусть даны N-тел с начальными положениями $\mathbf{x}_i$ и скоростями $\mathbf{v}_i$ для $1 \leq i \leq N$. Сила $\mathbf{f}_{ij}$, возникающая в результате гравитационного взаимодействия тел i и j, задается формулой:

$$\mathbf{f}_{ij} = G\frac{m_i m_j}{||\mathbf{r}_{ij}||^3}\mathbf{r}_{ij},$$

где $m_i,\ m_j$ – массы тел i и j соответсвенно; $\mathbf{r}_{ij} = \mathbf{x}_j - \mathbf{x}_i$ – вектор, соединяющий тела i и j, направленный в сторону тела j; G – гравитационная постоянная.

Результирующее взаимодействие $\mathbf{F}_i$ на тело i, в результате взаимодействий на него других $N-1$ тел, получается суммированием всех взаимодействий:

$$\mathbf{F}_i = \sum_{\substack{1 \leq j \leq N \\ j \neq i}} \mathbf{f}_{ij} = Gm_i \sum_{\substack{1 \leq j \leq N \\ j \neq i}} \frac{m_j}{||\mathbf{r}_{ij}||^3}\mathbf{r}_{ij}.$$

По мере того, как тела притягиваются, сита между ними безгранично возрастает, что является нежелательным поведением при численном интегрирования. В астрофизических симуляциях обычно не учитывают столкновения частиц. Такое поведение тел соответсвует движению галактик, которые могут проходить друг сквозь друга.

Для того, чтобы исключить такое нежелательное поведение, вводят смягчающий коэффициент $\varepsilon^2 > 0$, в результате чего формулу гравитационного взаимодействия записывают в виде:

$$\mathbf{F}_i = Gm_i \sum_{1 \leq j \leq N} \frac{m_j}{\left(||\mathbf{r}_{ij}||^2 + \varepsilon^2\right)^{\frac{3}{2}}}\mathbf{r}_{ij}.$$

Следует отметить, что при такой записи нет необходимости в условии $j \neq i$, так как при $\varepsilon^2 > 0 : \ \mathbf{f}_{ii} = 0$. Смягчающий коэффициент моделирует взаимодействие между двумя массами, которые ведут себя как сферические галактики. Также его присутствие ограничивает максимальное значение гравитационного взаимодействия, что крайне необходимо для численного интегрирования задачи N-тел.

Для того, чтобы найти положения $\mathbf{x}_i$ и скорости $\mathbf{v}_i$ системы во времени, необходимо знать ускорения $\mathbf{a}_i = \dfrac{\mathbf{F}_i}{m_i}$, поэтому вычисление ускорений имеет вид:

$$\mathbf{a}_i = G \sum_{1 \leq j \leq N} \frac{m_j}{\left(||\mathbf{r}_{ij}||^2 + \varepsilon^2\right)^{\frac{3}{2}}} \mathbf{r}_{ij}.$$

Для решения получающихся систем дифференциальных уравнений вида

$$\begin{cases} \dfrac{d\mathbf{x}_i}{dt} = \mathbf{v}_i \\ \dfrac{d\mathbf{v}_i}{dt} = G \displaystyle\sum_{1 \leq j \leq N} \frac{m_j}{\left(||\mathbf{r}_{ij}||^2 + \varepsilon^2\right)^{\frac{3}{2}}} \mathbf{r}_{ij} \end{cases} \tag{1.3.1}$$

используется метод Leapfrog-Verlet, потому что он применим для решения поставленной задачи из-за своей вычислительной эффективности: он имеет очень хорошую точность вычислений относительно своей вычислительной сложности. (Выбор метода интегрирования в задаче N-тел обычно зависит от природы исследуемой системы).

1.4 Метод интегрирования системы ОДУ Leapfrog-Verlet

Данный метод применим к задаче моделирования гравитационного взаимодействия, т.к. ускорение тела зависит только от расположения других гравитационных тел и не зависит от их скоростей.

Аппроксимация первого уравнения системы (1.3.1) методом Эйлера:

$$\mathbf{x}_i^{(1)} = \mathbf{x}_i^{(0)} + h\mathbf{v}_i^{(0)},$$

где под h подразумевают временной шаг. Аппроксимация будет более точной если заменить $\mathbf{v}$ значением в середине рассматриваемого интервала:

$$\mathbf{x}_i^{(1)} = \mathbf{x}_i^{(0)} + h\mathbf{v}_i^{(\frac{1}{2})}.$$

Предположим, что мы знаем значение $\mathbf{v}_i^{(\frac{1}{2})}$. В таком случае мы сразу получаем аналогичное правило для вычисления второго уравнения системы (1.3.1):

$$\mathbf{v}_i^{(\frac{3}{2})} = \mathbf{v}_i^{(\frac{1}{2})} + hG \sum_{1 \leq j \leq N} \frac{m_j}{\left(||\mathbf{r}_{ij}^{(1)}||^2 + \varepsilon^2\right)^{\frac{3}{2}}} \mathbf{r}_{ij}^{(1)},$$

т.к. нам на этот момент уже известно значение $\mathbf{r}_{ij}^{(1)}$. Т.о. задав начальные значения с помощью $\mathbf{x}_i^{(0)}$, $\mathbf{v}_i^{(\frac{1}{2})}$, можно производить вычисления по схеме, указанной на (Рис.1).

Рассмотрим точность метода. Значение разности $|\mathbf{x}_i^{(1)} - \mathbf{x}_i^{(0)}|$ имеет порядок $O(h)$, а т.к. для midpoint approximation ошибка $O(h^2)$ пропадает, то ошибка на интервале имеет порядок $O(h^3)$. Для того, чтобы проинтегрировать систему по конечному промежутку времени

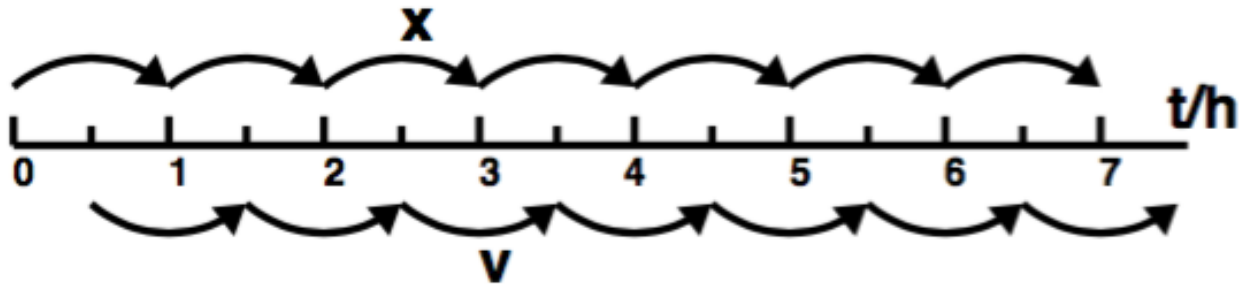


Рис. 1: Схема вычислений методом Leapfrog

T, необходимо весь промежуток разбить на $\frac{T}{h}$ интервалов и проинтегрировать по каждому отдельно. В таком случае общая ошибка интегрирования будет порядка $O(h^3\frac{1}{h}) = O(h^2)$. Т.о. алгоритм Leapfrog является алгоритмом второго порядка точности, как, например, алгоритм RK2 и он лучше, чем метод Эйлера, который имеет лишь первый порядок точности.

Самый простой метод вычисления начальной скорости, не понижающий аппроксимацию систему, есть ни что иное как метод Эйлера с уменьшенным в два раза временным шагом:

$$\mathbf{v}_i^{(\frac{1}{2})} = \mathbf{v}_i^{(0)} + \frac{h}{2}G \sum_{1 \le j \le N} \frac{m_j}{\left(||\mathbf{r}_{ij}^{(0)}||^2 + \varepsilon^2\right)^{\frac{3}{2}}}\mathbf{r}_{ij}^{(0)}$$

Таким образом алгоритм решения дифференциального уравнения может быть записан в виде следующей системы:

$$\begin{cases} \mathbf{x}_i^{(n+1)} = \mathbf{x}_i^{(n)} + h\mathbf{v}_i^{(n+\frac{1}{2})} \\ \mathbf{v}_i^{(n+\frac{3}{2})} = \mathbf{v}_i^{(n+\frac{1}{2})} + hG \sum\limits_{1 \le j \le N} \dfrac{m_j}{\left(||\mathbf{r}_{ij}^{(n+1)}||^2 + \varepsilon^2\right)^{\frac{3}{2}}}\mathbf{r}_{ij}^{(n+1)} \end{cases}$$

В гравитационной задаче N тел сила взаимодействия каждого тела со всеми остальными телами системы меняется при малейшем изменении положения системы, поэтому на каждой итерации интегрирования системы происходит вычисление силы взаимодействия.

Перед первой итерацией, когда известны начальные координаты тел и начальные скорости, происходит вычисление $\mathbf{v}_i^{(\frac{1}{2})}$, после чего достаточно хранить значения $\mathbf{x}_i^{(n)}$ и $\mathbf{v}_i^{(n+\frac{1}{2})}$ для достижения точности вычислений $O(h^2)$.

У метода интегрирования Leapfrog есть два основных достоинства:

1. Время-обратимость метода. У данного метода существует способность после n шагов в прямом направлении начать интегрирование в обратном направлении и прийти в ту же самую точку, откуда и начинались вычисления.

2. Симплектическая природа метода. Leapfrog метод сохраняет (немного измененную) полную энергию динамической системы. Этот факт исключительно полезен для задач небесной механики, т.к. такие методы, как RK4 не сохраняют полную энергию системы и тем самым позволяют параметрам системы значительно меняться со временем.

1.5 Рассматриваемые алгоритмы

All-pairs algorithm для задачи N-тел является алгоритмом полного перебора, который вычисляет все попарные взаимодействия между N-телами. Данный метод является относительно простым, однако он никогда не используется при моделировании больших систем из-за его вычислительной сложности $O(N^2)$.

Вышеуказанный подход используется как составная часть более сложных и быстрых алгоритмов для вычисления сил между соседними телами. Такие алгоритмы как Barnes-Hut simulation или Fast multipole method основаны на более грубой аппроксимации взаимодействий с далеко отстоящими телами.

1.6 Используемое программное и аппаратное обеспечение

```
> The device: GeForce GT 750M
> The major and minor revision numbers defining the device's compute capability: 3.0
> The total amount of global memory available on the device in bytes: 2147024896
> The maximum amount of shared memory available to a thread block in bytes: 49152
> The maximum number of 32-bit registers available to a thread block: 65536
> The warp size in threads: 32
> The maximum number of threads per block: (1024, 1024, 64)
> The maximum size of each dimension of a grid: (2147483647, 65535, 65535)
> The total amount of constant memory available on the device in bytes: 65536
> The number of multiprocessors on the device: 2
```

2 All-pairs algorithm

2.1 Описание алгоритма и стратегия распараллеливания

Рассматриваемый алгоритм основан на вычислении всех попарных взаимодействий $\mathbf{f}_{ij},\ (i,j) \in \{1,...,N\} \times \{1,...N\}$, образующих матрицу взаимодействий $\mathbf{f}_{mn}$ размеров $N \times N$.

В рассматриваемой гравитационной задаче существует взаимосвязь между взаимодействиями, называемая третьим законом Ньютона: $\mathbf{f}_{ij} = -\mathbf{f}_{ji}$. Это соотношение может быть использовано для того, чтобы уменьшить количество вычислений в два раза, однако такой подход плохо повлияет на эффективность параллельных вычислений (особенно в задачах с маленьким значением N), поэтому в данном алгоритме данное соотношение не учитывается.

Результирующая сила $\mathbf{F}_i$, действующая на i-ое тело получается в результате суммирования всех взаимодействий i-ой строки матрицы взаимодействий $\mathbf{f}_{mn}$. Каждое вхождение матрицы может быть вычислено независимо, поэтому задача имеет сложность параллельных вычислений $O(N^2)$, однако такой подход требует $O(N^2)$ и будет ограничен по времени пропускной способностью памяти. В следствие этого следует отказаться от полностью параллельной схемы вычислений и ввести некую последовательную составляющую в вычислениях для достижения пика производительности арифметико-логического устройства и понизить требуемую пропускную способность памяти.

Для создания параллельного алгоритма необходимо ввести понятие подпространства – квадратная сетка попарных взаимодействий, состоящая из p строк и p столбцов. Все p^2 взаимодействий могут быть получены из характеристик $2p$ тел, p из которых могут быть использованы в дальнейшем. Характеристики этих p тел могут быть сохранены в разделяемой памяти или в регистрах. Результатом взаимодействий тел рассматриваемого подпространства является p обновленных векторов-ускорений.

Для того, чтобы достигнуть оптимального использования памяти элементы строки подпространства вычисляются последовательно, обновляя вектор-ускорений, в то время

как p потоков параллельно работают с p строками. На (Рис.2) показана стратегия распараллеливания и входные-выходные данные, получающиеся в результате работы программы.

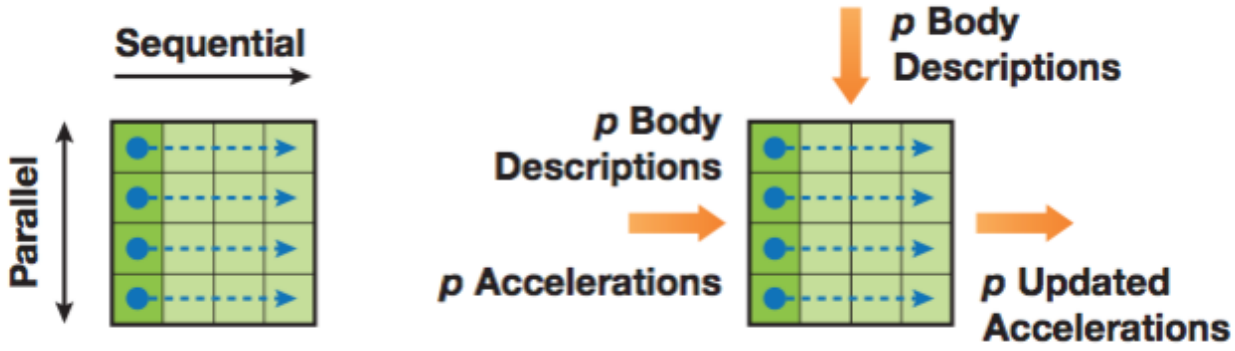


Рис. 2: Стратегия распараллеливания

2.2 Взаимодействие двух тел

Взаимодействие двух тел реализуется с помощью последовательных вычислений. В исходном коде, приведенном в Листинге 2.2.1, вычисляется сила взаимодействия между телами i и j, а также обновляется вектор-ускорение тела i, в результате действия на него тела j. В приведенном исходном коде всего 20 операций с плавающий точкой типа float, включая операции суммирования, произведения, взятие корня и деления.

Листинг 2.2.1: Взаимодействие двух тел

```
__device__ float3
bodyBodyInteraction(float4 bi, float4 bj, float3 ai) {
    float3 r;
    // r_ij [3 FLOPS]
    r.x = bj.x - bi.x; r.y = bj.y - bi.y; r.z = bj.z - bi.z;
    // distSqr = dot(r_ij, r_ij) + EPS^2 [6 FLOPS]
    float distSqr = r.x * r.x + r.y * r.y + r.z * r.z + _EPS2_;
    // invDistCube =1/distSqr^(3/2) [4 FLOPS (2 mul, 1 sqrt, 1 inv)]
    float distSixth = distSqr * distSqr * distSqr;
    float invDistCube = 1.0f/sqrtf(distSixth_eps);
    // s = m_j * invDistCube [1 FLOP]
    float s = bj.w * _GAMMA_ * invDistCube;
    //a_i= a_i+s*r_ij[6FLOPS]
    ai.x += r.x * s; ai.y += r.y * s; ai.z += r.z * s;
    return ai;
}
```

Использование float4 позволяет хранить как координаты точек, так и их массы (в поле w). Также использование вышеприведенного типа данных вместо float3 позволяет видеокарте использовать coalesced memory access к массивам, хранящимся в глобальной памяти, что приводит к более эффективной передаче данных.

2.3 Вычисления внутри подпространств

Подпространство вычисляется с помощью p потоков, выполняющих одну и ту же последовательность операций над разными данными. Каждый поток обновляет ускорение одного тела в результате действия на него p других тел. p характеристик тел из GPU памяти записываются в разделяемую память. Каждый поток в блоке вычисляет p взаимодействий. Результатом вычисления подпространства является p обновленных ускорений.

Исходный код работы с подпространствами приведен в Листинге 2.3.1. Входной аргумент myPosition содержит местоположение тела, соответствующего текущему потоку, в то время, как массив shPosition – множество тел для которых надо вычислить взаимодействие с текущим телом myPosition. Все p потоков внутри одного подпространства работают параллельно и каждый поток вычисляет взаимодействие своего текущего тела со всеми телами из shPosition. Результатом работы каждого потока будет обновленное ускорение текущего тела после учета его взаимодействия со всеми телами массива shPosition.

Листинг 2.3.1: Вычисления внутри подпространства

```
__device__ float3
tileComputation(float4 myPosition, float3 accel)
{
    extern __shared__ float4 shPosition[];
    for (int i = 0; i < blockDim.x; ++i)
        accel = bodyBodyInteraction(myPosition, shPosition[i], accel);
    return accel;
}
```

Архитектура GPU поддерживает одновременное чтение несколькими потоками одного адреса в разделяемой памяти, поэтому отсутствуют конфликты чтения разделяемой памяти (shared-memory-bank conflicts) при вычислении взаимодействий между телами.

2.4 Вычисление нескольких подпространств одним CUDA блоком

Каждый CUDA блок состоит из p независимых потоков, которые последовательно производят вычисления над несколькими подпространствами. Размер подпространств подбирается таким образом, чтобы сбалансировать параллелизм и повторное использование данных. Степень параллелизма (количество строк – p) должен быть достаточно большим для того, чтобы несколько параллельно работающих варпов (warps) смогло “скрыть” задержки на вычисление взаимодействий между телами. Количество повторных использований данных растет с увеличением числа колонок, что также влияет на количество передач данных из глобальной памяти GPU в разделяемую память. Размер подпространства также определяет то, сколько требуется регистровой и разделяемой памяти. В данной реализации мы используем квадратные подпространства размеров p на p. Перед тем как произвести вычисления внутри подпространства каждый поток копирует параметры одного тела в разделяемую память, после чего происходит синхронизация.

На (Рис.3) показан последовательный процесс вычисления p параллельными потоками нескольких подпространств. Жирные линии ограничивают подпространства, тем самым показывая в какие моменты происходит обновление данных в разделяемой памяти и после – барьерная синхронизация. Каждому CUDA блоку соответствует $\frac{N}{p}$ подпространств, над которыми производят вычисления p параллельных потоков для вычисления сил, действующих телами подпространств на текущие p тел. Каждый поток вычисляет все N взаимодействий между своим текущим телом и всеми остальными телами.

Исходный код для вычисления сил, которые действуют со стороны N тел на текущие p тел приведен в Листинге 2.4.1. Этот код является CUDA ядром, который вызывается с CPU.

Аргументами функции calculate_forces являются указатели на глобальную память GPU, содержащую местоположение и ускорение тел. Первым делом происходит преобразование к нужным типам памяти, чтобы система могла рассматривать их как массивы данных. Итерирование по подпространствам требует использования двух барьерных синхронизаций. Первая синхронизация необходима для того, чтобы удостовериться, что разделяемая память полностью загружена перед тем как начать вычислять взаимодействия, вторая необходимо для того, чтобы убедиться, что все потоки отработали и можно

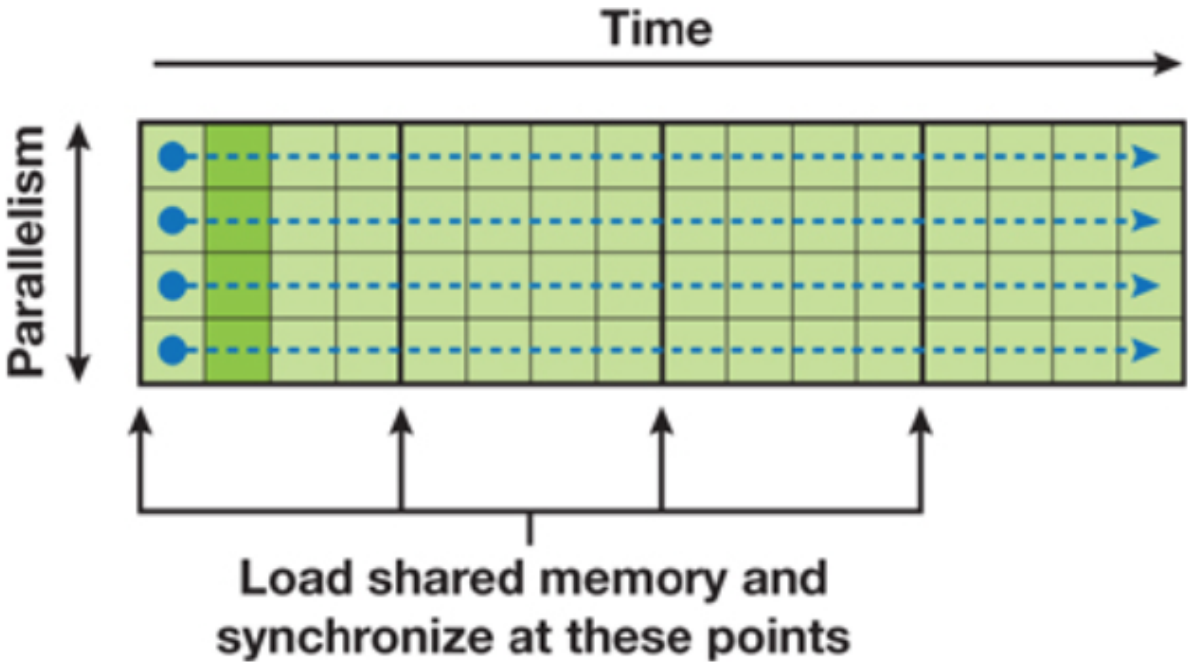


Рис. 3: Вычисление нескольких подпространств одним CUDA блоком

загружать в разделяемую память элементы последующего подпространства.

Листинг 2.4.1: Вычисления всех гравитационных ускорений для p тел

```
__global__ void
calculateForces(void* devX, void* devA, size_t N)
{
    extern __shared__ float4 shPosition[];
    int gtid = blockIdx.x * blockDim.x + threadIdx.x;
    if(gtid >= N)
        return;
    float4* globalX = (float4*)devX;
    float3* globalA = (float3*)devA;
    float3 acc = {0.0f, 0.0f, 0.0f};
    float4 myPosition = globalX[gtid];
    for (int i = 0, tile = 0; i < N; i += blockDim.x, tile++) {
        int idx = tile * blockDim.x + threadIdx.x;
        shPosition[threadIdx.x] = globalX[idx];
        __syncthreads();
        acc = tileComputation(myPosition, acc);
        __syncthreads();
    }
    // Save the result in global memory for the integration step.
    globalA[gtid] = acc;
}
```

2.5 Определение сетки CUDA блоков

CUDA ядро в Листинге 2.4.1 вычисляет ускорение p тел системы, вызванное взаимодействием этих тел со всеми N телами системы. Для того, чтобы вычислить ускорение всех N тел необходимо запустить 1D CUDA сетку из $\frac{N}{p}$ CUDA блоков. В результате получим N независимых потоков, каждый из которых вычисляет N взаимодействий.

Вычисление всей матрицы взаимодействий может быть визуализировано как показано на (Рис.4). Ось Y показывает параллелизм системы: 1D сетка, состоящая из $\frac{N}{p}$ независимых CUDA блоков, каждый из которых состоит из p независимых потоков. Ось X показывает количество последовательных шагов, которое требуется для вычисления всех

N взаимодействий с текущим потоком. Массив данных, состоящий из p элементов, который расположен в разделяемой памяти каждого блока перезаписывается после каждых p итераций.

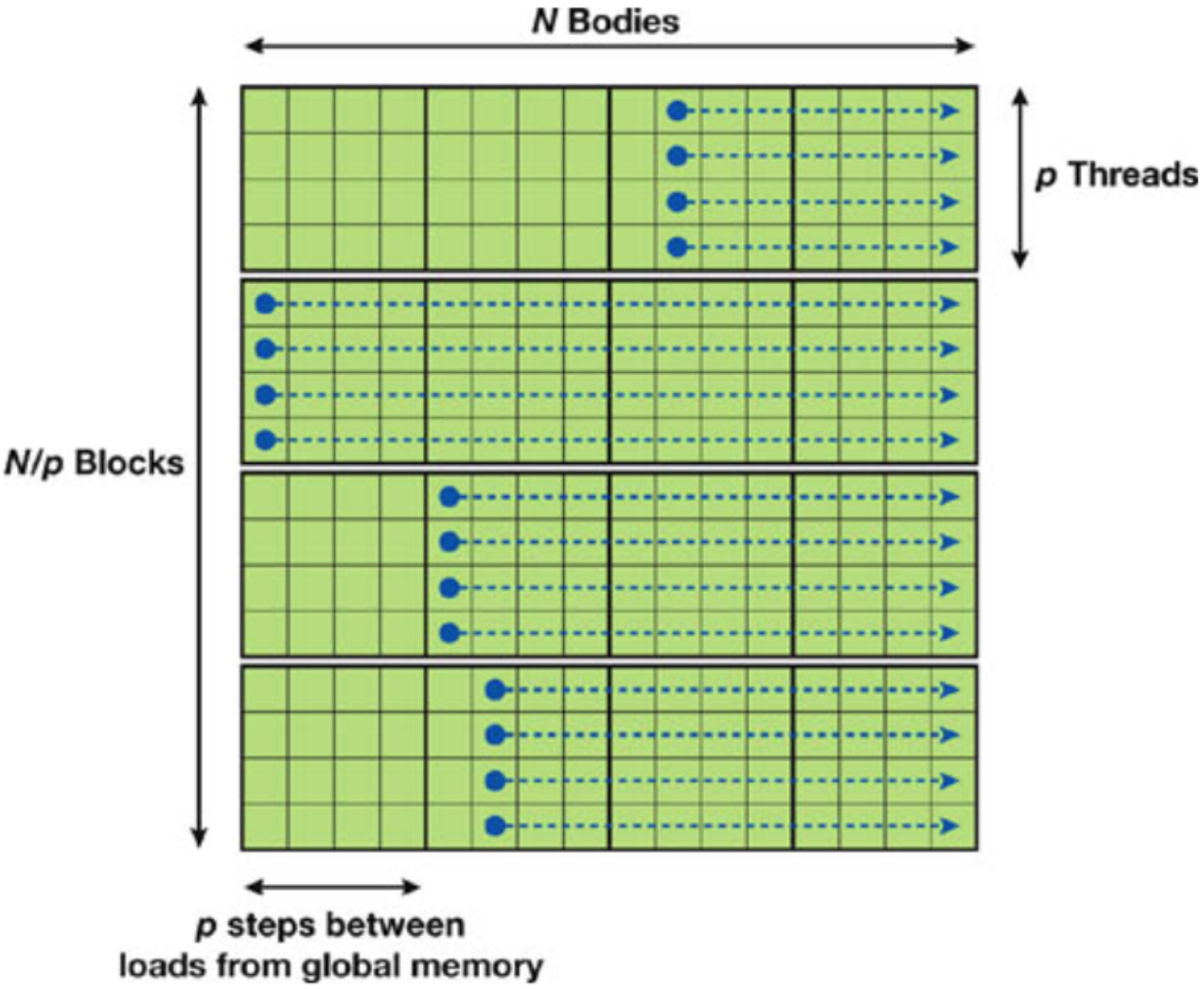
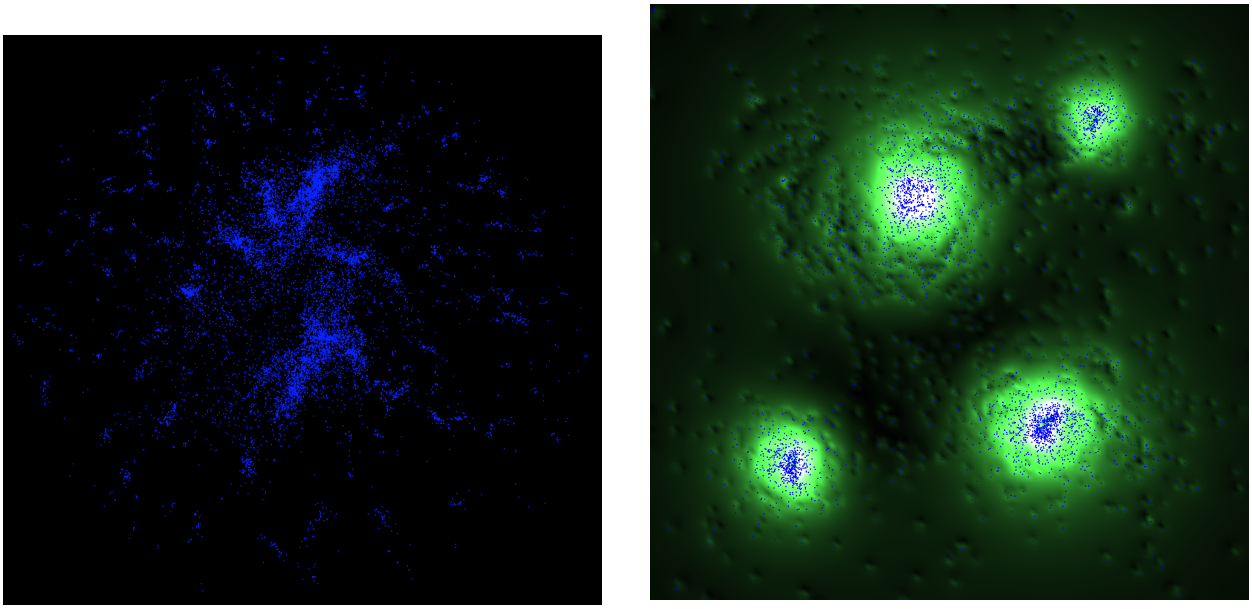


Рис. 4: CUDA сетка, вычисляющая все N^2 взаимодействий

2.6 Результаты работы программы

Результат работы программы можно видеть на (Рис. 5). При симуляции взаимодействия N-тел без вычисления потенциального поля можно симулировать движение 11264 тел в реальном времени без видимой "заторможенности", однако при вычислении потенциального поля возможность вычислений в реальном времени сильно падает из-за большой дополнительной нагрузки, накладываемой из-за вычисления потенциальной энергии во многих точках простратсва. Поэтому при вычислении потенциального поля происходит симуляции взаимодействия лишь 4096 тел.

Основная сложность при численном моделировании поведения N-тел заключается в том, что тяжело подобрать масштаб единиц измерения, который бы подходил для интересующего поведения: если выбрать слишком маленькое расстояние или большую массу, симуляция показывает бесконтрольное увеличение скоростей и симуляция визуально кажется неверной; при выборе же слишком большого расстояния или слишком маленькой



(a) Симуляция системы из 11264 тел

(b) Симуляция системы из 4096 тел с вычислением потенциального поля

Рис. 5: Симуляция задачи N-тел

массы получающееся движение также противоречит представляемой теоретической траектории движения.

К сожалению смоделировать движение галактики так и не получилось, видимо она имеет определенную структуру, которую на глаз подобрать очень сложно.

3 Barnes–Hut algorithm

3.1 Идея метода

All-pairs algorithm – алгоритм просчета всех взаимодействий между всеми парами тел. Сложность такого алгоритма $O(N^2)$, что является ограничением для моделирования взаимодействия большого набора тел (например моделирование поведения галактики).

Существует несколько типов оптимизации алгоритма All-pairs algorithm, одним из которых является иерархический подход к оптимизации. Идея, заложенная в иерархическую оптимизацию, состоит в том, чтобы взаимодействие рассматриваемого тела с далеко отстоящими от него объектами аппроксимировать одним взаимодействием с центром масс этих далеко отстоящих объектов.

Основная идея алгоритма Barnes–Hut algorithm заключается в том, чтобы рекурсивно делить пространство на подпространства до того момента, пока не выполнится какой-то критерий, например пока в каждом подпространстве не будет не больше k тел (в случае не такого большого количества исследуемых тел принято брать $k = 1$). В случае если два тела находятся достаточно близко друг к другу, используется All-pairs algorithm для вычисления взаимодействия между ними, однако если тела расположены далеко друг от друга, то используется специальная аппроксимация, сокращающая вычислительную сложность.

3.2 Описание метода

Алгоритм Barnes–Hut algorithm широко используется при симуляции взаимодействия галактик. Он иерархическим образом разбивает пространство, в котором содержатся все исследуемые объекты, на подпространства, называемые узлами, и вычисляет необходимую информацию об объектах, находящихся в каждом узле, что позволяет быстро аппроксимировать силы (например гравитационные, электрические или магнитные), с которыми N тел действуют друг на друга.

Иерархическое разбиение пространства записывается в октодерево (Рис. 6), которое является трехмерным аналогом бинарного дерева.

Barnes–Hut algorithm уменьшает вычислительную сложность с $O(n^2)$ до $O(n \log n)$, что позволяет решать намного более сложные вычислительные задачи.

Алгоритм Barnes–Hut algorithm тяжело эффиктивно реализовать с помощью CUDA, из-за следующих причин:

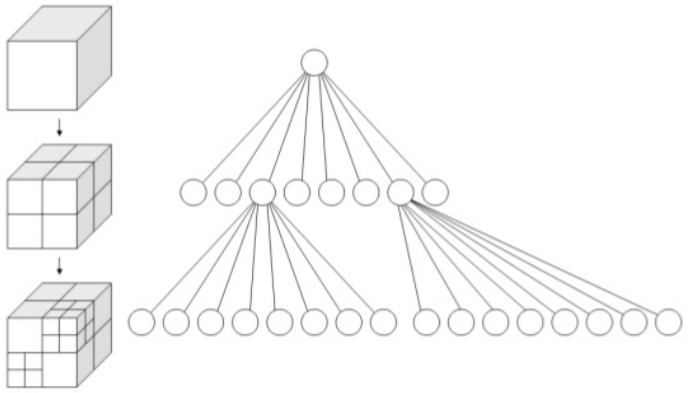


Рис. 6: Разбиение пространства в виде октодерева

1. алгоритм постоянно добавляет и изменяет элементы нерегулярной деревоподобной структуры данных, следствием чего является рассинхронизация потоков и потеря параллелизма;

2. алгоритм выполняет много pointer-chasing memory операций, что приводит к медленному uncoalesced доступу к мамяти;

3. алгоритм определяется рекурсивным образом, в то время как динамический параллелизм доступен в CUDA только на видеокартах с архитектурой SM ≥ 3.5.

3.3 Реализация метода

Листинг 3.3.1: Barnes-Hut algorithm

```
(0) Read input data and transfer to GPU
for each timestep do {
    (1) Compute bounding box around all
        bodies
    (2) Build hierarchical decomposition by
        inserting each body into octree
    (3) Summarize body information in each
        internal octree node
    (4) Approximately sort the bodies by
        spatial distance
    (5) Compute forces acting on each body
        with help of octree
    (6) Update body positions and velocities
}
(7) Transfer result to CPU and output
```

На Листинге 3.3.1 показаны высокоуровневые шаги, которые делает Barnes-Hut algorithm. Шаги с первого по шестой, являющиеся телом временного цикла, выполняются на GPU. Каждый из этих шагов реализован как отдельное CUDA ядро. Такая структура программы выбрана в связи с тем, что необходимо иметь глобальную барьерную синхронизацию потоков между шагами. Плюсом такого подхода также является тот факт, что для каждого ядра можно указывать необходимое ему количество блоков и нитей.

Четвертый шаг не обязателен, т.к. его задача лишь в ускорении работы программы.

На (Рис. 7), (Рис. 8), (Рис. 9) проиллюстрированы операции, которые выполняют ядра (1), (2), (3) и (5).

Ядро (1) вычисляет размеры пространства, в котором находятся все тела; это пространство становится корнем октодерева (ячейка, содержащая все рассматриваемые тела). Результат работы первого ядра можно видеть на (Рис. 7).

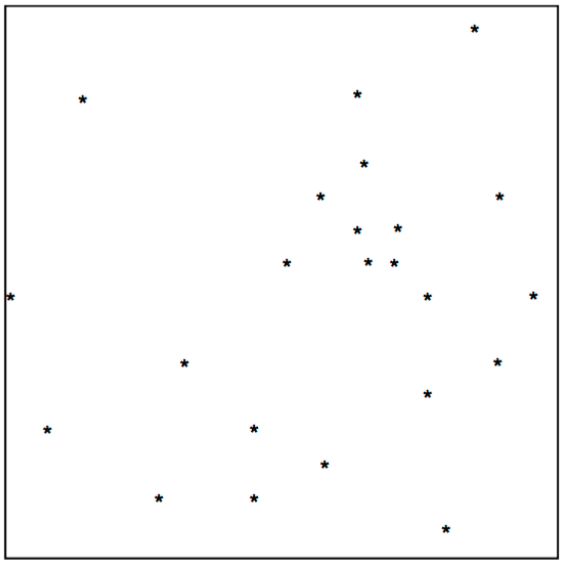


Рис. 7: Bounding box

Ядро (2) иерархически разделяет эту ячейку до тех пор, пока каждое тело не окажется в своей подпространстве. Это достигается путем добавления каждого тела в октодерево. Результат работы первого ядра можно видеть на (Рис. 8).

Ядро (3) вычисляет для каждой ячейки центр масс и суммарную массу для всех тел, находящихся внутри рассматриваемого подпространства (на (Рис. 9a) показаны два вычисленных центра масс для заштрихованных ячеек).

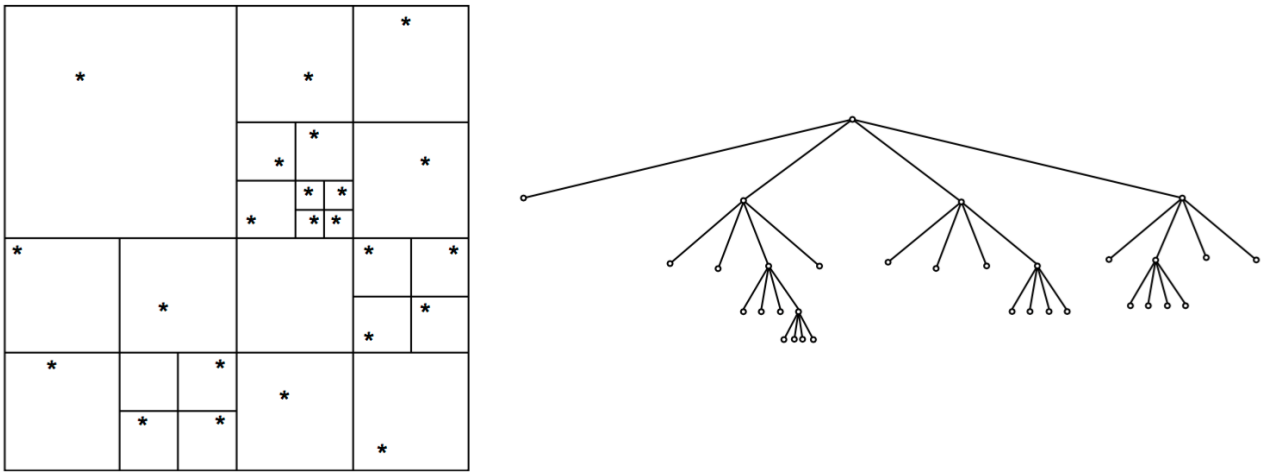


Рис. 8: Иерархическая структура пространства

Ядро (4) сортирует тела таким образом, чтобы рядом стоящие в пространстве тела располагались в октодереве также располагались друг с другом.

Ядро (5) вычисляет силы, действующие на каждое тело. Начиная с корня октодерева, оно проверяет находится ли центр масс каждой ячейки достаточно далеко от исследуемого тела (исследуемое тело находится в точке из которой исходят стрелки на (Рис. 9b)). Если центр масс находится недостаточно далеко (сплошная стрелка), то алгоритм рекурсивно спускается дочерние ячейки для более точных вычислений (три пунктирные стрелки). Если же центр масс находится достаточно далеко (одна пунктирная стрелка), то вычисляется лишь одна сила – сила взаимодействия исследуемого тела и центра масс ячейки, и спуск к дочерним ячейкам не производится, тем самым сильно сокращая общее число вычислений.

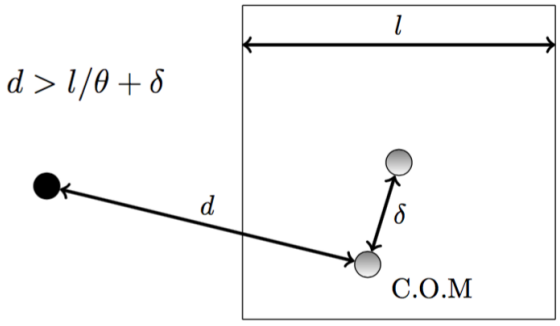
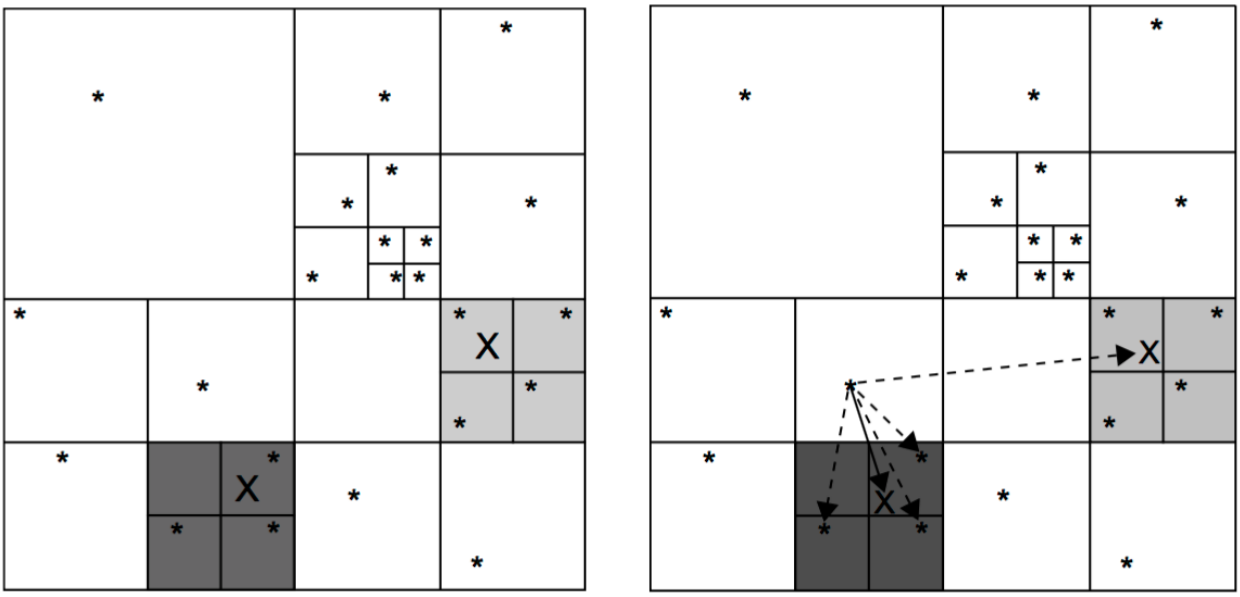


Рис. 10: Критерий



(a) Вычисление центра масс

(b) Вычисление сил с помощью октодерева

Рис. 9: Вычисление взаимодействий между телами

Критерий, по которому определяется находится ли центр масс ячейки достаточно далеко от исследуемого тела, имеет пороговую структуру, которая определяется следующим образом: предположим, что $\mathbf{r}_0$ – радиус вектор до исследуемого объекта, в то время как $\mathbf{r}_C$ – радиус вектор до центра масс ячейки, l – размер подпространства, в котором расположен центр масс, δ – расстояние между центром масс и геометрическим центром подпространства, как показано на (Рис. 10); в таком случае если

$$d = ||\mathbf{r}_0 - \mathbf{r}_C|| > \frac{l}{\theta} + \delta,$$

при наперед заданном параметре θ, то считается, что центр масс находится достаточно далеко, т.о. набор сил можно аппроксимировать одной силой. Если же данное условие не выполняется, то необходимо рекурсивно спускаться в дочерние ячейки и повторять проверку критерия. В симуляции поведения небесных тел принято брать параметр $\theta = 0.5$.

Ядро (6) обновляет координаты и скорости исследуемых тел. В конкретном случае используется Leapfrog algorithm для численного решения системы дифференциальных уравнений.

4 Вывод

Задача N-тел – очень вычислительно сложная задача, т.к. необходимо на каждой итерации вычислять попарные взаимодействия между всеми телами. Существуют некоторые оптимизации алгоритма, однако все они основаны на аппроксимации точного решения и в большинстве своем являются очень непростыми в реализации алгоритмами.

Список литературы

[1] Hubert Nguyen. 2007. Gpu Gems 3 (First ed.). Addison-Wesley Professional.

[2] Peter Young. 2014. The leapfrog method and other “symplectic” algorithms for integrating Newton’s laws of motion.

[3] Martin Burtscher, Keshav Pingali. 2011. An Efficient CUDA Implementatioon of the Tree-Based Barnes Hut n-Body Algorithm. Elsevier Inc.

[4] Gunther Lukat, Robi Banerjee. 2015. A GPU accelerated Barnes-Hut Tree Code for FLASH4. Elsevier Inc.