# Параллельные вычисления. Лабораторная работа 1

Цель данной лабораторной работы – ознакомиться с возможностями стандарта OpenMP и получить практические навыки в параллельном решении математических задач с высокой вычислительной сложностью.

Задача: Дано начальное положение N точечных тел. Тела расположены в двумерном пространстве. Каждое тело характеризуется начальными координатами (x,y) в пространстве, массой m (скалярная величина) и начальной скоростью V (вектор). Предполагается, что на тела не действуют никакие силы, кроме гравитационных. Под действием гравитации тела изменяют свое положение, а также скорость. Промоделировать эволюцию данного набора взаимодействующих тел, а именно определить координаты тел в пространстве и их скорости в моменты времени

## 1. Теоретическая часть

## 1.1 Формулировка задачи

Гравитационная задача N тел является классической проблемой небесной механики и гравитационной динамики Ньютона. Она формулируется следующим образом: в пустоте находится N материальных точек, массы которых известны. Пусть попарное взаимодействие точек подчинено закону тяготения Ньютона, и пусть силы гравитации аддитивны. Пусть известны начальные на момент времени t=0 положения и скорости каждой точки. Требуется найти положения точек для всех последующих моментов времени.

## 1.2 Математическая модель задачи

**Закон всемирного тяготения**

где и – массы взаимодействующих тел, – расстояние между ними, – гравитационная постоянная, равная , – абсолютная величина силы, с которой каждое из тел действует на другое (сила, действующая на тело i со стороны тела j, направлена от тела i к телу j).

**Правило сложения сил**

Если на тело действуют несколько сил, то их можно заменить одной силой, равной векторной сумме всех исходных сил.

**Второй закон Ньютона**

где – сила, действующая на тело, – масса тела, – ускорение, которое тело приобретает под действием данной силы ( и являются векторными величинами; их направления совпадают).

Взаимодействие тел моделируется пошагово с помощью дискретных отрезков времени фиксированной длительности . На каждом шаге вычисляется сила, действующая на каждое тело в начальный момент времени (с применением закона всемирного тяготения и правила сложения сил), по найденной силе определяется ускорение (по второму закону Ньютона). Закон движения тела определяется из дифференциального уравнения второго порядка:

Данное уравнение равносильно системе двух дифференциальных уравнений первого порядка:

Обозначая и , где , и используя простейший численный метод решения системы дифференциальных уравнений, при котором функция заменяется постоянным значением на каждом отрезке , получаем следующие формулы для пересчета положения и скорости тела:

Для простоты предполагается, что все тела расположены в одной плоскости. В этом случае все векторные величины, связанные с этими телами, также лежат в этой плоскости. Положение каждого тела определяется двумя координатами ; каждая векторная величина также определяется двумя координатами:

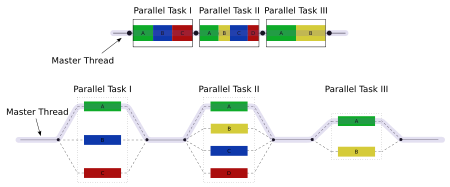
Для того чтобы описанный выше простейший вариант алгоритма не приводил к нестабильному поведению системы близко расположенных тел, можно наложить ограничение на максимальное значение силы, действующей на тело со стороны другого (близко расположенного) тела. Разумеется, подобное ограничение нельзя считать оправданным с физической точки зрения, однако при моделировании реальной системы астрономических тел расстояние между ними является настолько большим, что силы никогда не достигают указанного порогового значения. Данное ограничение предназначено лишь для того, чтобы обеспечить устойчивое поведение модельной системы тел, используемой при тестировании полученных вариантов программы.

При моделировании системы (с учетом наложенного ограничения на максимальное значение силы) можно достаточно произвольным образом выбирать такие характеристики, как массы тел, их скорости, расстояния между ними и значение гравитационной постоянной.

## 1.3 OpenMP

OpenMP – API, предназначенное для программирования многопоточных приложений на многопроцессорных системах с общей памятью. Разработку спецификации OpenMP ведут несколько крупных производителей вычислительной техники и программного обеспечения. OpenMP поддерживается основными компиляторами.

В OpenMP вы «не увидите» потоки в коде. Вместо этого, вы сообщаете компилятору с помощью директив #pragma, что блок кода может быть распараллелен. Зная данную информацию, компилятор в состоянии сгенерировать приложение, состоящее из одного главного потока, который создаёт множество других потоков для параллельного блока кода. Эти потоки синхронизируются в конце параллельного блока кода, и мы снова возвращаемся к одному главному потоку.



Так как OpenMP контролируется #pragma, то код на C++ корректно скомпилируется любым компилятором C++, потому что неподдерживаемые #pragma должны игнорироваться. Однако OpenMP API содержит также несколько функций, и, чтобы ими воспользоваться, необходимо подключить заголовочный файл. Самый легкий способ определить, поддерживает ли компилятор OpenMP - попробовать подключить файл omp.h:

#include <omp.h>

Если OpenMP поддерживается, нужно его включить с помощью специальных флагов компилятора:

* gcc -fopenmp
* Intel -openmp (Linux, MacOSX), -Qopenmp (Windows)
* Microsoft -openmp (Настройки проекта в Visual Studio)

## Синтаксис

Директивы OpenMP начинаются с #pragma omp.

parallel

Данная директива создаёт группу из N потоков. N определяется во время выполнения, обычно это число ядер процессора, но также можно задать N вручную. Каждый из потоков в группе выполняет следующую за директивой команду (или блок команд, определённый в {}-скобках). После выполнения, потоки «сливаются» в один.

for

Директива for разделяет цикл for между текущей группой потоков, так что каждый поток в группе обрабатывает свою часть цикла.

schedule (планирование)

Программист может контролировать то, каким образом потоки будут загружаться работой при обработке цикла. Существует несколько вариантов. Static является вариантом по умолчанию. Ещё до входа в цикл каждый поток «знает», какие части цикла он будет обрабатывать. Второй вариант - ключевое слово dynamic. В данном случае невозможно предсказать порядок, в котором итерации цикла будут назначены потокам. Каждый поток выполняет указанное число итераций. Если это число не задано, по умолчанию оно равно 1. После того, как поток завершит выполнение заданных итераций, он переходит к следующему набору итераций. Так продолжается, пока не будут пройдены все итерации. Последний набор итераций может быть меньше, чем изначально заданный. Такой вариант очень полезен, когда разные итерации цикла обсчитываются разное время.

Более подробную информацию по OpenMP, и ее директивам (с примерами кода) можно найти по ссылке: <https://software.intel.com/ru-ru/blogs/2011/11/21/openmp-c>.

## 2. ПРАКТИЧЕСКАЯ ЧАСТЬ

## 2.1 Непараллельный вариант программы

Все материалы, необходимые для выполнения лабораторной работы можно найти в labs/lab1. В первую очередь протестируем неэффективный вариант расчета сил и движения тел, без использования OpenMP.

// непараллельный расчет сил в момент времени t

void CalcForces1()

{

for (int i = 0; i < N; i++)

for (int j = 0; j < N; j++)

{

if (i == j) continue;

double dx = p[j].x - p[i].x, dy = p[j].y - p[i].y,

r\_2 = 1 / (dx \* dx + dy \* dy),

r\_1 = sqrt(r\_2),

fabs = gravity \* m[i] \* m[j] \* r\_2;

if (fabs > fmax) fabs = fmax;

f[i].x = f[i].x + fabs \* dx \* r\_1;

f[i].y = f[i].y + fabs \* dy \* r\_1;

}

}

// непараллельный пересчет координат и обнуленеие массива сил

void MoveParticlesAndFreeForces()

{

for (int i = 0; i < N; i++)

{

double dvx = f[i].x \* dt / m[i],

dvy = f[i].y \* dt / m[i];

p[i].x += (p[i].vx + dvx / 2) \* dt;

p[i].y += (p[i].vy + dvy / 2) \* dt;

p[i].vx += dvx;

p[i].vy += dvy;

f[i].x = 0;

f[i].y = 0;

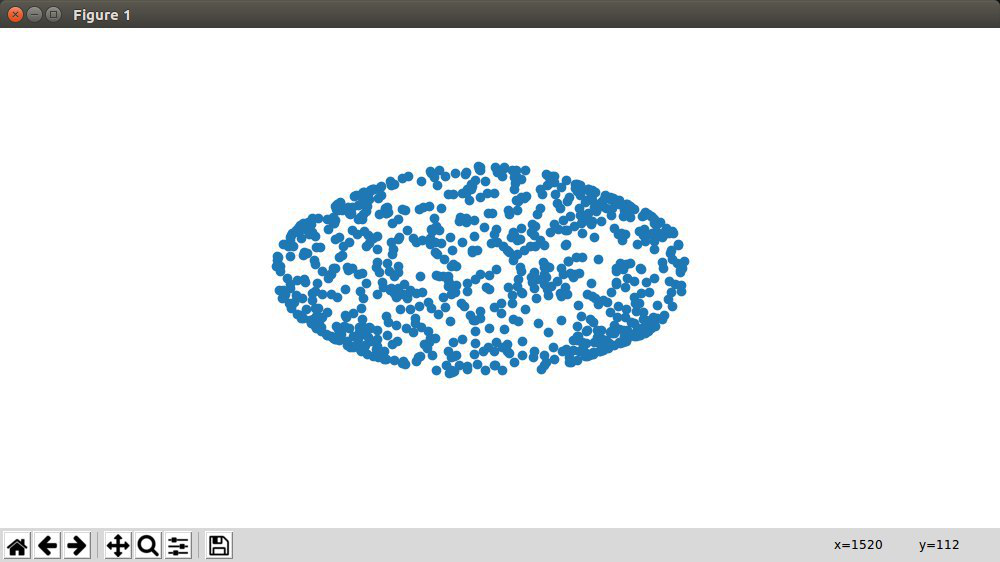
}

}

Основная часть вычислений выполняется в функции CalcForces1, определяющей силы взаимодействия между телами. Использованный в ней двойной цикл, в котором оба параметра перебираются от 0 до N–1, приводит к тому, что одна и та же сила, возникающая при взаимодействии двух тел, вычисляется дважды. Вместе с тем, при нахождении силы использованы приемы, позволяющие ускорить вычисления. Во вспомогательной переменной r\_2 сохраняется величина, обратная квадрату расстояния, что позволяет в дальнейшем избежать применения операции деления (которая выполняется дольше, чем операция умножения). Еще в одной переменной r\_1 сохраняется величина, обратная расстоянию. Таким образом, функция извлечения квадратного корня вызывается единственный раз.

## 2.2 Визуализация полученных результатов

Для наглядности, можно визуализировать полученные результаты вычислений. Для этого необходимо запустить скрипт «visualization.py». В скрипте необходимо указать число тел, выбранное в основной программе. Пример визуализации представлен на рисунке.



## 2.3 Непараллельный вариант программы и исключением повторного вычисления силы взаимодействия пары тел

Очевидным способом ускорить вычисление сил является организация одновременного нахождения сил, действующих между парой тел. Для этого достаточно изменить циклы так, чтобы параметр внутреннего цикла j был всегда больше параметра внешнего цикла i.

// непараллельный расчет сил в момент времени t с одновременным нахождением сил для пары тел

void CalcForces2()

{

for (int i = 0; i < N - 1; i++)

for (int j = i + 1; j < N; j++)

{

double dx = p[j].x - p[i].x, dy = p[j].y - p[i].y,

r\_2 = 1 / (dx \* dx + dy \* dy),

r\_1 = sqrt(r\_2),

fabs = gravity \* m[i] \* m[j] \* r\_2;

if (fabs > fmax) fabs = fmax;

f[i].x += dx = fabs \* dx \* r\_1;

f[i].y += dy = fabs \* dy \* r\_1;

f[j].x -= dx;

f[j].y -= dy;

}

}

## 2.4 Параллельный вариант программы

Для распараллеливания вычислений добавим в начало программы директиву, определяющую число потоков для параллельного варианта программы.

#define Nthr 2 // число потоков

Затем, добавим в функции CalcForces и MoveParticlesAndFreeForces соответствующие директивы.

// параллельный расчет сил в момент времени t

void CalcForces1Par()

{

#pragma omp parallel for num\_threads(Nthr)

for (int i = 0; i < N; i++)

for (int j = 0; j < N; j++)

{

if (i == j) continue;

double dx = p[j].x - p[i].x, dy = p[j].y - p[i].y,

r\_2 = 1 / (dx \* dx + dy \* dy),

r\_1 = sqrt(r\_2),

fabs = gravity \* m[i] \* m[j] \* r\_2;

if (fabs > fmax) fabs = fmax;

f[i].x += fabs \* dx \* r\_1;

f[i].y += fabs \* dy \* r\_1;

}

}

// параллельный пересчет координат и обнуление массива сил

void MoveParticlesAndFreeForcesPar()

{

#pragma omp parallel for num\_threads(Nthr)

for (int i = 0; i < N; i++)

{

double dvx = f[i].x \* dt / m[i],

dvy = f[i].y \* dt / m[i];

p[i].x += (p[i].vx + dvx / 2) \* dt;

p[i].y += (p[i].vy + dvy / 2) \* dt;

p[i].vx += dvx;

p[i].vy += dvy;

f[i].x = 0;

f[i].y = 0;

}

}

В данном случае ускорение, по сравнению с исходным непараллельным алгоритмом оказалось меньше двух, поскольку потоки не сбалансированы по нагрузке. Еще более важным является то обстоятельство, что результат вычислений отличается от ранее полученного. Это объясняется конкуренцией потоков за доступ к элементам массива f (массива сил), поскольку теперь каждый элемент изменяется на различных итерациях внешнего цикла (которые могут выполняться параллельно на разных потоках). При этом возможна ситуация, когда значение некоторого элемента массива f будет одновременно считано и увеличено разными потоками; в результате, разумеется, одна из добавленных сил будет потеряна.

## 2.5 Параллельный вариант программы с использованием критической секции

Для исправления отмеченного недочета достаточно защитить операторы изменения элементов массива f с помощью критической секции.

Полученный вариант окажется крайне неэффективным, поскольку один из потоков регулярно будет приостанавливать свою работу в ожидании освобождения критической секции. Тем не менее, сделанная модификация достигает своей цели: теперь результат вычислений не зависит от порядка доступа потоков к элементам массива f.

// параллельный расчет сил в момент времени t с использованием критической секции

void CalcForces2ParA()

{

#pragma omp parallel for num\_threads(Nthr)

for (int i = 0; i < N - 1; i++)

for (int j = i + 1; j < N; j++)

{

double dx = p[j].x - p[i].x, dy = p[j].y - p[i].y,

r\_2 = 1 / (dx \* dx + dy \* dy),

r\_1 = sqrt(r\_2),

fabs = gravity \* m[i] \* m[j] \* r\_2;

if (fabs > fmax) fabs = fmax;

#pragma omp critical

{

f[i].x += dx = fabs \* dx \* r\_1;

f[i].y += dy = fabs \* dy \* r\_1;

f[j].x -= dx;

f[j].y -= dy;

}

}

}

## 2.6 Параллельный вариант с использованием дополнительных массивов сил для каждого потока

Для того, чтобы ускорить вычисления и при этом не допускать конкуренции потоков, достаточно использовать вспомогательные массивы «добавочных сил» tf, связав с каждым потоком свой массив добавок. В результате при вычислении добавок не будет возникать конкуренции. При этом дополнительно придется предусмотреть цикл, в котором элементы массивов tf будут добавляться к соответствующим элементам массива f.

/ параллельный расчет сил в момент времени t с использованием дополнительных массивов для каждого потока

void CalcForces2ParB()

{

#pragma omp parallel for num\_threads(Nthr)

for (int i = 0; i < N - 1; i++)

{

int k = omp\_get\_thread\_num();

for (int j = i + 1; j < N; j++)

{

double dx = p[j].x - p[i].x, dy = p[j].y - p[i].y,

r\_2 = 1 / (dx \* dx + dy \* dy),

r\_1 = sqrt(r\_2),

fabs = gravity \* m[i] \* m[j] \* r\_2;

if (fabs > fmax) fabs = fmax;

tf[i][k].x += dx = fabs \* dx \* r\_1;

tf[i][k].y += dy = fabs \* dy \* r\_1;

tf[j][k].x -= dx;

tf[j][k].y -= dy;

}

}

#pragma omp parallel for num\_threads(Nthr)

for (int i = 0; i < N; i++)

for (int j = 0; j < Nthr; j++)

{

f[i].x += tf[i][j].x;

f[i].y += tf[i][j].y;

tf[i][j].x = 0;

tf[i][j].y = 0;

}

}

Интересно отметить, что данный вариант будет выполняться быстрее не только варианта с критической секцией, но и первоначального варианта (при котором возникала конкуренция потоков).

## 2.7 Параллельный вариант с использованием динамической балансировки нагрузки между потоками

Еще одним способов увеличить эффективность работы программы является балансировка нагрузки между потоками. Простейшим способом обеспечить более равномерную нагрузку потоков является использование опции dynamic:

#pragma omp parallel for num\_threads(Nthr) schedule(dynamic, block)

Параметр block определяет размер «порции» итераций, которая будет назначена каждому потоку. Скорость программы повышается в случае, если каждому потоку назначается достаточно большая порция итераций.

// параллельный расчет сил в момент времени t с использованием динамической балансировки нагрузки между потоками

int block = 25;

void CalcForces2ParC()

{

#pragma omp parallel for num\_threads(Nthr) schedule(dynamic, block)

for (int i = 0; i < N - 1; i++)

{

int k = omp\_get\_thread\_num();

for (int j = i + 1; j < N; j++)

{

double dx = p[j].x - p[i].x, dy = p[j].y - p[i].y,

r\_2 = 1 / (dx \* dx + dy \* dy),

r\_1 = sqrt(r\_2),

fabs = gravity \* m[i] \* m[j] \* r\_2;

if (fabs > fmax) fabs = fmax;

tf[i][k].x += dx = fabs \* dx \* r\_1;

tf[i][k].y += dy = fabs \* dy \* r\_1;

tf[j][k].x -= dx;

tf[j][k].y -= dy;

}

}

#pragma omp parallel for num\_threads(Nthr) schedule(dynamic, block)

for (int i = 0; i < N; i++)

for (int j = 0; j < Nthr; j++)

{

f[i].x += tf[i][j].x;

f[i].y += tf[i][j].y;

tf[i][j].x = 0;

tf[i][j].y = 0;

}

}

## 2.8 Параллельный вариант с ручным распределением итераций между потоками

Другим способом обеспечения сбалансированности нагрузки является распределение итераций по «полосам»: потоку с номером k назначаются итерации с номерами:

где Nthr обозначает общее число потоков (потоки, как и итерации, нумеруются от нуля). При этом итерации цикла распределяются по потокам следующим образом:

void CalcForces2ParD()

{

#pragma omp parallel num\_threads(Nthr)

{

int k = omp\_get\_thread\_num();

for (int i = k; i < N - 1; i += Nthr)

{

for (int j = i + 1; j < N; j++)

{

double dx = p[j].x - p[i].x, dy = p[j].y - p[i].y,

r\_2 = 1 / (dx \* dx + dy \* dy),

r\_1 = sqrt(r\_2),

fabs = gravity \* m[i] \* m[j] \* r\_2;

if (fabs > fmax) fabs = fmax;

tf[i][k].x += dx = fabs \* dx \* r\_1;

tf[i][k].y += dy = fabs \* dy \* r\_1;

tf[j][k].x -= dx;

tf[j][k].y -= dy;

}

}

}

#pragma omp parallel for num\_threads(Nthr)

for (int i = 0; i < N; i++)

for (int j = 0; j < Nthr; j++)

{

f[i].x += tf[i][j].x;

f[i].y += tf[i][j].y;

tf[i][j].x = 0;

tf[i][j].y = 0;

}

}

Данный вариант выполняется быстрее, чем первоначальный вариант без балансировки нагрузки, однако он будет проигрывать вариант с динамическим распределением потоков.

## 2.9 Практические задания

1. Придумать индивидуальные начальные параметры системы (начальные координаты, массы, начальные скорости, количество частиц (N)). При выборе учитывать, что область визуализации имеет размер 4000х4000. Количество частиц (N) взять не менее 500. Количество итераций (Niter) не менее 800. Начальные параметры необходимо подбирать таким образом, чтобы время выполнения каждого алгоритма таблицы ниже было разумным: секунды, десятки секунд.
2. Промоделировать систему, заполнить таблицу ниже, продемонстрировать визуализацию одного из алгоритмов. Время выполнения считать усредненным по не менее 10 запускам алгоритмов. Ускорение считать относительно первого алгоритма.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Алгоритм | Время выполнения, с (ускорение, ед) при Nthr | | | | |
| 1 | 4 | 8 | 16 | 32 |
| CalcForces1+MoveParticlesAndFreeForces |  |  |  |  |  |
| CalcForces2+MoveParticlesAndFreeForces |  |  |  |  |  |
| CalcForces1Par+MoveParticlesAndFreeForces**Par** |  |  |  |  |  |
| CalcForces2ParA+MoveParticlesAndFreeForces**Par** |  |  |  |  |  |
| CalcForces2ParB+MoveParticlesAndFreeForces**Par** |  |  |  |  |  |
| CalcForces2ParC (block=10)+MoveParticlesAndFreeForces**Par** |  |  |  |  |  |
| CalcForces2ParC (block=25)+MoveParticlesAndFreeForces**Par** |  |  |  |  |  |
| CalcForces2ParC (block=50)+MoveParticlesAndFreeForces**Par** |  |  |  |  |  |
| CalcForces2ParC (block=100)+MoveParticlesAndFreeForces**Par** |  |  |  |  |  |
| CalcForces2ParD+MoveParticlesAndFreeForces**Par** |  |  |  |  |  |

1. Нарисовать 2 графика: время выполнения в зависимости от Nthr и ускорение в зависимости от Nthr. Все алгоритмы представить на каждом графике линиями разных цветов.
2. Обосновать результаты.

## Список литературы

1. Введение в OpenMP: параллельное программирование на C++ [Электронный ресурс] / URL: <https://software.intel.com/ru-ru/blogs/2011/11/21/openmp-c> (Дата обращения: 23.12.2017)
2. Параллельные методы решения гравитационной задачи n тел [Электронный ресурс] / URL: <http://edu.mmcs.sfedu.ru/pluginfile.php/5634/mod_resource/content/3/OMPMPIGravit2016.pdf> (Дата обращения: 23.12.2017)
3. Гравитационная задача N тел [Электронный ресурс] / URL: <https://ru.wikipedia.org/wiki/Гравитационная_задача_N_тел> (Дата обращения: 23.12.2017)