Параллельные методы решения гравитационной задачи *n* тел

1. Напишите параллельную программу решения задачи n тел с использованием OpenMP
2. Сравните время ее работы при разном числе потоков со временем работы последовательной программы (см. ниже)
3. Сравните время работы параллельной программы при разных способах балансировки нагрузки.

Постановка задачи и ее математическая модель

Задача. Дано начальное положение *N* точечных тел. Каждое тело характеризуется массой *m* (скаляр- ная 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.

Математическая модель задачи использует следующие формулы.

1. Закон всемирного тяготения

$F = Gm_1m_2/r^2$, где m_1 и m_2 – массы взаимодействующих тел, r – расстояние между ними, G – гравитационная постоянная, равная $6{,}67\times10^{-11}$ Н×м^2/кг^2, F – 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*).

2. Правило сложения сил

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

3. Второй закон Ньютона

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

Взаимодействие тел моделируется пошагово с помощью дискретных отрезков времени фиксированной длительности *dt*. На каждом шаге вычисляется сила, действующая на каждое тело в начальный момент времени (с применением формул 1 и 2), по найденной силе определяется ускорение *a* (формула 3).

Закон движения тела $S = S(t)$ определяется из дифференциального уравнения второго порядка

$$d^2S/dt^2 = a(t),$$

которое равносильно системе двух дифференциальных уравнений первого порядка:

$$dS/dt = V(t),\ dV/dt = a(t),$$

где $V(t)$ – скорость тела в момент t.

Обозначая $S_i = S(t_i)$ и $V_i = V(t_i)$, где $t_i = t_0 + i \times dt$, и используя простейший численный метод решения системы дифференциальных уравнений, при котором функция $a(x)$ заменяется постоянным значением $a_i = a(t_i)$ на каждом отрезке $[t_i,\ t_{i+1}]$, получаем следующие формулы для пересчета положения и скорости тела:

$$V_{i+1} = V_i + a_i \times dt,$$

$$S_{i+1} = S_i + V_i \times dt + a_i \times dt^2/2.$$

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 (x, y); каждая векторная величина также определяется двумя координатами: $F = (F_x, F_y)$. $a = (a_x. a_y)$. $V = (V_x, V_y)$.

В случае, когда взаимодействующие тела располагаются близко друг от друга, описанный простейший метод расчета оказывается неустойчивым в силу увеличения погрешностей вычисления. В данной ситуации можно использовать более точные методы интегрирования систем дифференциальных уравнений, а также переменный шаг по времени (мы не будем рассматривать подобные модификации).

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.

Непараллельный вариант программы

Вначале приведем менее эффективный вариант программы, в котором для каждого тела явным обра- зом вычисляются все действующие на него силы.

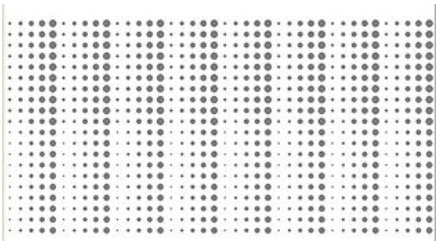
```
#include  <iostream>

 #include  <iomanip>

 #include  <cstdlib>

#include  <ctime>

 #include  <cmath>

#include  <omp.h>

using  namespace  std;

#define  gravity  10  // гравитационная постоянная

#define  dt  0.1   // шаг по времени

#define  N  800   // количество частиц

#define  fmax  1  // максимальное значение силы

 #define  Niter  100          // число итераций

struct  Particle

{

double  x,  y,  vx,  vy;

};

struct  Force

{

double  x,  y;

};

Particle  p[N]; Force  f[N]; double  m[N];
```

```
void Init()
{
for (int i = 0; i < N; i++)
{
p[i].x = 20 * (i / 20 - 20) + 10;
p[i].y = 20 * (i % 20 - 10) + 10;
p[i].vx = p[i].y / 15;
p[i].vy = -p[i].x / 50; m[i] = 100 + i % 100;
f[i].x = 0;
f[i].y = 0;
}
}
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;
}
}
void MoveParticlesAndFreeForces()
{
```

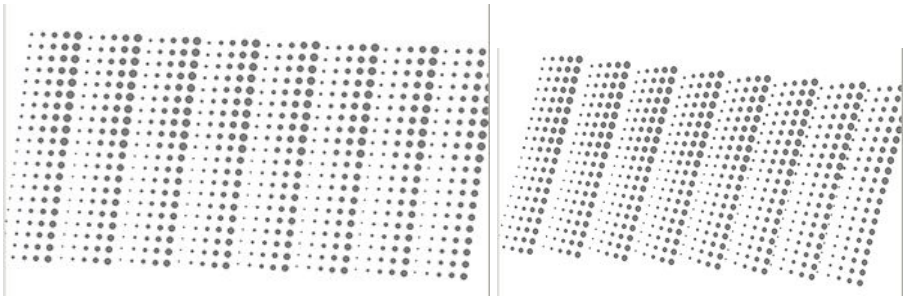
```
for (int i = 0; i < N; i++)
{
double dvx = f[i].x * dt / m[i], dvy = f[i].y * dt / m[i];
p[i].x += (p[i].vx + dvx / 2) * dt;
p[i].y += (p[i].vy + dvy / 2) * dt; p[i].vx += dvx;
p[i].vy += dvy;
f[i].x = 0;
f[i].y = 0;
}
}

void info(char* s, double time)
{
cout << setw(30) << left << s << "Time: " << fixed << setprecision(0) << setw(6) << 1000*time
<< setprecision(12) << "p0: " << setw(12) << p[0].x << ", " << setw(12) << p[0].y << endl;
}

void main()
{
Init();
double t = omp_get_wtime();
for (int i = 0; i < Niter; i++)
{
CalcForces1(); MoveParticlesAndFreeForces();
}
t = omp_get_wtime() - t; info("Non-Parallel (N*N)", t);
```

```
}
```

В качестве модельной системы используется набор из 800 точечных тел (частиц), расположенных в узлах прямоугольной сетки и вращающихся вокруг центра этой сетки.



Массы частиц изменяются в диапазоне от 100 до 199; на приведенных рисунках более массивные частицы представлены в виде кругов с большим радиусом. Ниже приведены изображения системы частиц после 20 и 40 итераций.



В качестве тестовых значений, которые в дальнейшем будут использоваться для проверки правильности различных модификаций исходной программы, выводятся значения координат начальной частицы после 100 итераций. Кроме того, выводится время расчета в миллисекундах; для этого используется функция omp_get_time().

При выполнении приведенной программы будет выведен следующий текст:

```
Non-Parallel (N*N)          Time: 8358 p0: -285.496803732846, 7.014089107234
```

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.

Очевидным способом ускорить вычисление сил является организация *одновременного* нахождения сил, действующих между парой тел.

Параллельный вариант программы

Для того, чтобы вычислять все силы параллельно, достаточно сделать параллельным цикл в функции CalcForces1Par(). Но если просто добавить туда #pragma omp parallel for, то возникнут гонки при использовании массива f.

Если предотвратить это за счет критической секции, то не будет получено никакого ускорения.

Для получения ускорения нужно добавить вспомогательные массивы “добавочных сил”, у каждого потока будет свой массив добавок. После вычисления всех добавок нужно будет добавить к каждому элементу массива f соответствующие добавки.

Напишите параллельный вариант программы решения гравитационной задачи с использованием массивов добавочных сил и OpenMP

Балансировка нагрузки

Наиболее оптимальная балансировка нагрузки достигается при распределении потоков по «обрат- ным полосам». В этом случае итерации распределяются по потокам в «зеркальном» порядке:

0, 1, 2, …, Nthr–1, Nthr–1, …, 2, 1, 0, 0, 1, 2, …, Nthr–1, Nthr–1, …, 2, 1, 0, … .

Другой вариант - использование schedule(dynamic, block)

Сделайте варианты параллельной программы с разными способами балансировки нагрузки и сравните время их работы