Лекция 9
Параллельное решение задачи *N*-тел

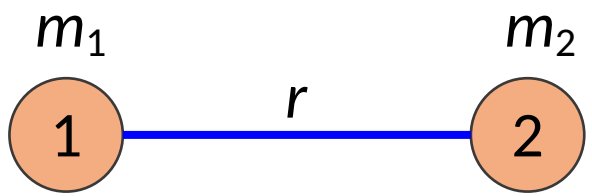
Курносов Михаил Георгиевич

E-mail: mkurnosov@gmail.com
WWW: www.mkurnosov.net

Курс «Параллельные вычислительные технологии»
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)
Осенний семестр

CC-BY

Задача *n*-тел (*n*-body)



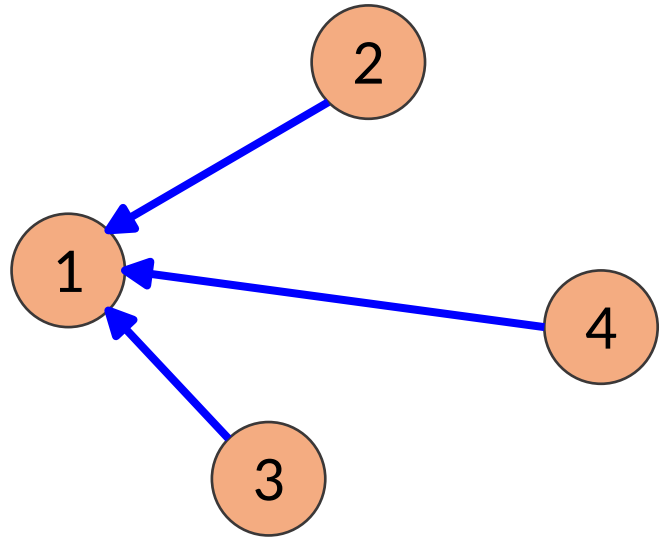
Сила гравитации между двумя телами

$$F = \frac{Gm_1m_2}{r^2}$$

Задача N-тел (nbody)

- Сила, действующая на тело 1, равна сумме сила действующих на тело 1 со стороны тел 2, 3 и 4

$$f_1 = f_{21} + f_{31} + f_{41}$$



- Гравитационные силы, действующие на тела, вызывают их ускорение и перемещение
- Интегрирование уравнений движения (схема leapfrog):

$$a_i = F_i / m_i$$

- Второй закон Ньютона

$$dv_i = a_i dt$$

$$dp_i = v_i dt + \frac{a_i}{2} dt^2 = \left(v_i + \frac{dv_i}{2} \right) dt$$

- Равноускоренное движение

[*] Эндрюс Г. *Основы многопоточного, параллельного и распределенного программирования*. - М.: Вильямс, 2003.

Задача N-тел (nbody)

```
enum { NPARTICLES = 1000 };
const double G = 6.67e-11;

struct particle {
    float x, y, z;
};

int main(int argc, char *argv[])
{
    double ttotal = wtime();
    int n = NPARTICLES;

    struct particle *p = xmalloc(sizeof(*p) * n); // Положение тела i
    struct particle *f = xmalloc(sizeof(*f) * n); // Сила, действующая на тело i
    struct particle *v = xmalloc(sizeof(*v) * n); // Скорость тела i
    double *m = xmalloc(sizeof(*m) * n);          // Масса тела i

    for (int i = 0; i < n; i++) {                 // Инициализация системы тел
        p[i].x = rand() / (float)RAND_MAX - 0.5;
        p[i].y = rand() / (float)RAND_MAX - 0.5;
        p[i].z = rand() / (float)RAND_MAX - 0.5;
        v[i].x = rand() / (float)RAND_MAX - 0.5;
        v[i].y = rand() / (float)RAND_MAX - 0.5;
        v[i].z = rand() / (float)RAND_MAX - 0.5;
        m[i] = rand() / (float)RAND_MAX * 10 + 0.01;
    }
```

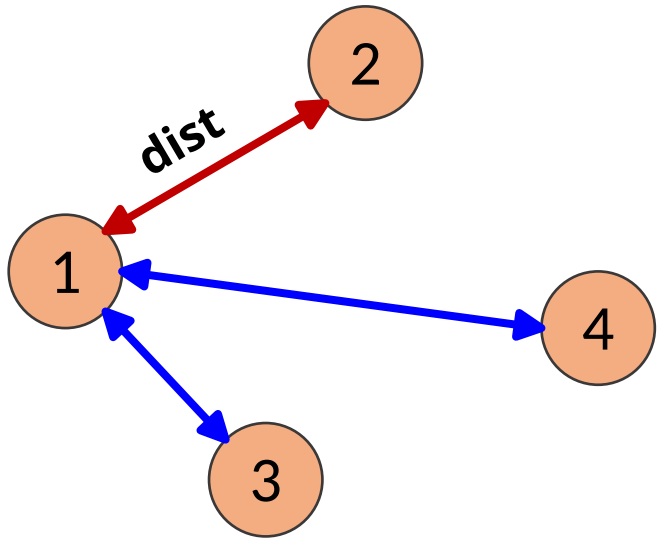
Задача N-тел (nbody)

```
    // Моделирование динамики положения тел – цикл по времени
    double dt = 1e-5;
    for (double t = 0; t <= 1; t += dt) {
        calculate_forces(p, f, m, n);          // расчет сил, действующих на тела
        move_particles(p, f, v, m, n, dt);     // перемещение тел
    }
    ttotal = wtime() - ttotal;

    printf("Elapsed time (sec): %.6f\n", ttotal);
    free(m);
    free(v);
    free(f);
    free(p);
    return 0;
}
```

Расчет сил, действующих на тела

```
void calculate_forces(struct particle *p, struct particle *f, double *m, int n)
{
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            double dist = sqrt(pow(p[i].x - p[j].x, 2) + pow(p[i].y - p[j].y, 2) +
                               pow(p[i].z - p[j].z, 2));
            double mag = (G * m[i] * m[j]) / pow(dist, 2);
            struct particle dir = {
                .x = p[j].x - p[i].x,
                .y = p[j].y - p[i].y,
                .z = p[j].z - p[i].z
            };
            f[i].x += mag * dir.x / dist;
            f[i].y += mag * dir.y / dist;
            f[i].z += mag * dir.z / dist;
            f[j].x -= mag * dir.x / dist;
            f[j].y -= mag * dir.y / dist;
            f[j].z -= mag * dir.z / dist;
        }
    }
}
```



Перемещение тел

```
void move_particles(struct particle *p, struct particle *f, struct particle *v, double *m,
                    int n, double dt)
{
    for (int i = 0; i < n; i++) {
        struct particle dv = {
            .x = f[i].x / m[i] * dt,
            .y = f[i].y / m[i] * dt,
            .z = f[i].z / m[i] * dt,
        };
        struct particle dp = {
            .x = (v[i].x + dv.x / 2) * dt,
            .y = (v[i].y + dv.y / 2) * dt,
            .z = (v[i].z + dv.z / 2) * dt,
        };
        v[i].x += dv.x;
        v[i].y += dv.y;
        v[i].z += dv.z;

        p[i].x += dp.x;
        p[i].y += dp.y;
        p[i].z += dp.z;

        f[i].x = f[i].y = f[i].z = 0;
    }
}
```

$$a_i = F_i / m_i$$

$$dv_i = a_i dt$$

$$dp_i = v_i dt + \frac{a_i}{2} dt^2 = \left(v_i + \frac{dv_i}{2} \right) dt$$

Варианты распараллеливания алгоритма

- Шаблон «производитель-потребитель»
- Программа пульсации
- Программа с конвейером
- …

Шаблон «производитель-потребитель»

- Имеется P процессов
- Множество тел разбивается на *P* блоков, в каждом из которых *n* / *P* тел
- Образуются пары (*i*, *j*) для всех возможных комбинация блоков
- Число пар блоков

$$\frac{P\,(P-1)}{2}$$

- Каждая пара блоков тел – это элементарная задача
- **Производитель** (главный процесс, producer) – раздает элементарные задачи рабочим процессам
- **Потребители** (рабочие процессы, consumer) – запрашивают у производителя задачи и выполняют вычисление сил между телами блоков *i* и *j*
- После вычисления сил, процессы обмениваются силами и перемещают тела

Производитель

```
chan getTask(int worker), task[1:PR](int block1, block2);
chan bodies[1:PR](int worker; point pos[*], vel[*]);
chan forces[1:PR](point force[*]);

process Manager {
  декларации и инициализация локальных переменных;
  for [time = start to finish by DT] {
    инициализация портфеля задач;
    for [i = 1 to numTasks+PR ] {
      receive getTask(worker);
      выбрать следующую задачу; если портфель пуст, сигнализировать (0, 0);
      send task[worker] (block1, block2);
    }
  }
}
```

[*] Эндрюс Г. *Основы многопоточного, параллельного и распределенного программирования.* – М.: Вильямс, 2003

Потребитель

```
process Worker[w = 1 to PR] {
  point p[1:n], v[1:n], f[1:n]; # положения, скорости
  double m[1:n]; # силы и массы тел
  декларации остальных локальных переменных, например tf[1:n], tp[1:n], tv[1:n];
  инициализация всех локальных переменных;
  for [time = start to finish by DT] {
    while (true) {
      send getTask(w); receive task[w](block1, block2);
      if (block1 == 0) break; # портфель пуст
      вычислить силы между телами block1 и block2;
    }
    for [i = 1 to PR st i != w]  # обмен силами
      send forces[i](f[*]);
    for [i = 1 to PR st i != w]
      receive forces[w](tf[*]);
      добавить значения tf к значениям в f;
    }
    обновить p и v в своем блоке тел;
    for [i = 1 to PR st i !=w] # обмен телами
      send bodies[i](w, p[*], v[*]);
    for [i = 1 to PR st i !=w]
      receive bodies[w](worker, tp[*], tv[*]);
      переместить тела процесса worker из tp и tv в p и v;
    }
    реинициализировать f нулями;
  }
}
```

Молекулярная динамика (Molecular dynamic)

- Рапапорт Д.К. **Искусство молекулярной динамики**. - РХД НИЦ. - 2012. - 630 с.
- LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) — пакет для классической молекулярной динамики (MPI, OpenMP)
- GROMACS, AMBER, CP2K, NWChem