

Лекция 9

Гравитационная задача n-тел (n-body)

Курносов Михаил Георгиевич

E-mail: mkurnosov@gmail.com

WWW: www.mkurnosov.net

Курс «Параллельные вычислительные технологии»

Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)

Весенний семестр, 2020

Гравитационная задача N тел (N-body)

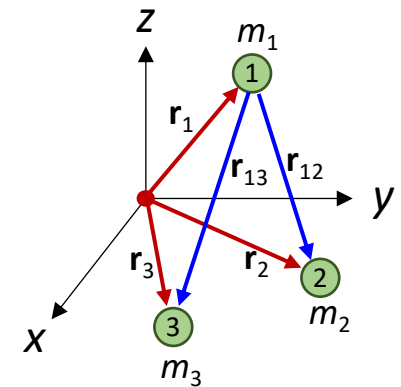
- **Гравитационная задача N тел (n-body simulation)** – задача небесной механики и гравитационной динамики Ньютона
- **Имеется** N материальных точек с заданными массами m_i . Попарное взаимодействие точек подчинено закону тяготения Ньютона, а силы гравитации аддитивны. Известны начальные на момент времени $t = 0$ положения и скорости каждой точки.
- **Требуется** найти положения точек для всех последующих моментов времени.

$$\frac{d\mathbf{r}_i}{dt} = \mathbf{v}_i, \quad i = 1, 2, \dots, N,$$

$$m_i \frac{d\mathbf{v}_i}{dt} = \mathbf{F}_i, \quad i = 1, 2, \dots, N,$$

Сила гравитации между телами i и j

$$F = \frac{Gm_i m_j}{r^2}$$



- G – гравитационная постоянная, r – расстояние между телами,
- На данный момент в общем виде задача N тел для $N > 3$ может быть решена только численно

Численное решение задачи N тел (N-body)

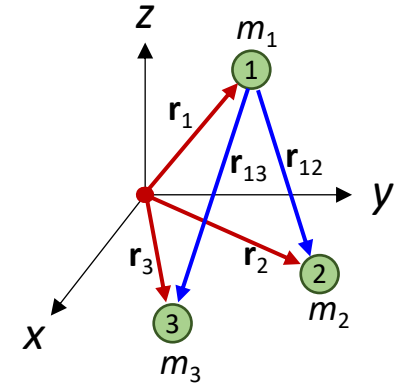
- Прямое моделирование задачи N-тел – $O(N^2)$

Цикл по времени

1. Вычисление сил, действующих на каждое тело – $O(N^2)$
2. Перемещение тел – $O(N)$

- Приближенные методы

- Метод Барнса-Хата (Treecode, Barnes–Hut simulation) – $O(N \log N)$
- Fast multipole methods – $O(N)$
- Particle mesh methods
- P3M and PM-tree methods
- Mean field methods



Прямое моделирование задачи N-тел – $O(N^2)$

Цикл по модельному времени: пока $t < t_{end}$

1. Вычисление сил, действующих на каждое тело

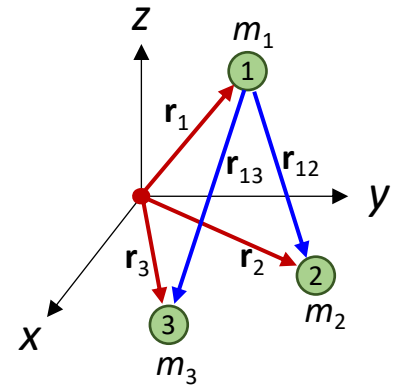
- Направление силы, действующей на тело i со стороны тела j , указывает от i в сторону j (вектор \mathbf{r}_{ij})
- Длина вектора \mathbf{r}_{ij} – расстояние между точками $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$
- Сила, действующая на тело i , – сумма сил воздействия всех тел на него

$$\mathbf{F}_i = \sum_{j \neq i}^N \frac{G m_i m_j \mathbf{r}_{ij}}{\left(\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \right)^2}$$

2. Перемещение тел

- Изменение скорости за интервал времени Δt : $d\mathbf{v}_i = \frac{\mathbf{F}_i}{m_i} \Delta t$
- Изменение положения тела за время Δt (схема leapfrog): $d\mathbf{r}_i = \left(\mathbf{v}_i + \frac{d\mathbf{v}_i}{2} \right) \Delta t$

3. Переход к следующему шагу по времени $t = t + \Delta t$



Последовательная программа

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

int main(int argc, char *argv[])
{
    double tttotal, tinit = 0, tforces = 0, tmove = 0;
    tttotal = wtime();
    int n = (argc > 1) ? atoi(argv[1]) : 10;
    char *filename = (argc > 2) ? argv[2] : NULL;

    tinit = -wtime();
    struct particle *p = malloc(sizeof(*p) * n);           // Положение частицы
    struct particle *f = malloc(sizeof(*f) * n);           // Сила, действующая на каждую частицу
    struct particle *v = malloc(sizeof(*v) * n);           // Скорость частицы
    float *m = malloc(sizeof(*m) * n);                    // Масса частицы
    for (int i = 0; i < n; i++) {
        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;
        f[i].x = f[i].y = f[i].z = 0;
    }
    tinit += wtime();
}
```

Последовательная программа

```
double dt = 1e-5;
for (double t = 0; t <= 1; t += dt) {    // Цикл по времени (модельному)
    tforces -= wtime();
    calculate_forces(p, f, m, n);        // Вычисление сил –  $O(N^2)$ 
    tforces += wtime();

    tmove -= wtime();
    move_particles(p, f, v, m, n, dt);    // Перемещение тел  $O(N)$ 
    tmove += wtime();
}
ttotal = wtime() - ttotal;

printf("# NBody (n=%d)\n", n);
printf("# Elapsed time (sec): ttotal %.6f, tinit %.6f, tforces %.6f, tmove %.6f\n",
       ttotal, tinit, tforces, tmove);
```

Последовательная программа

```
if (filename) {  
    FILE *fout = fopen(filename, "w");  
    if (!fout) {  
        fprintf(stderr, "Can't save file\n");  
        exit(EXIT_FAILURE);  
    }  
    for (int i = 0; i < n; i++) {  
        fprintf(fout, "%15f %15f %15f\n", p[i].x, p[i].y, p[i].z);  
    }  
    fclose(fout);  
}  
  
free(m);  
free(v);  
free(f);  
free(p);  
return 0;  
}
```

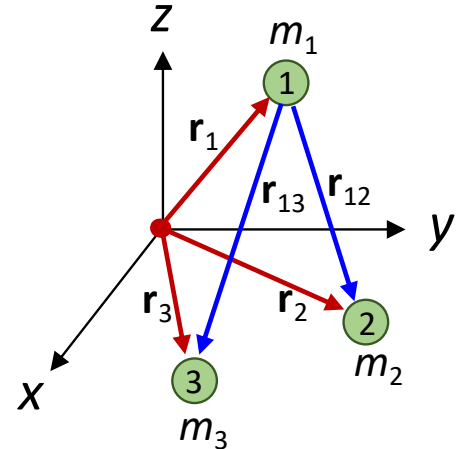
Последовательная программа

```
const float G = 6.67e-11;
```

```
void calculate_forces(struct particle *p, struct particle *f, float *m, int n)
{
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            float dist = sqrtf(powf(p[i].x - p[j].x, 2) + powf(p[i].y - p[j].y, 2) +
                                powf(p[i].z - p[j].z, 2));

            float mag = (G * m[i] * m[j]) / powf(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;
        }
    }
}
```



$$\mathbf{F}_i = \sum_{j \neq i}^N \frac{G m_i m_j \mathbf{r}_{ij}}{\left(\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \right)^2}$$

Последовательная программа

```
void move_particles(struct particle *p, struct particle *f, struct particle *v, float *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;
    }
}
```

$$d\mathbf{v}_i = \frac{\mathbf{F}_i}{m_i} \Delta t$$

$$d\mathbf{r}_i = \left(\mathbf{v}_i + \frac{d\mathbf{v}_i}{2} \right) \Delta t$$

Профилирование последовательной программы (Linux perf)

```
$ perf record ./nbody 100
# NBody (n=100)
# Elapsed time (sec): ttotat 9.696869, tinit 0.000043, tfoces 9.559545, tmove 0.130254

$ perf report
```

```
Samples: 38K of event 'cycles:u', Event count (approx.): 25946061841
Overhead Command Shared Object Symbol
 98.50% nbody nbody [.] calculate_forces
  1.33% nbody nbody [.] move_particles
  0.07% nbody [vdso] [.] __vdso_gettimeofday
  0.05% nbody [vdso] [.] 0x00000000000000949
  0.04% nbody nbody [.] main
  0.02% nbody [kernel.kallsyms] [k] __irqentry_text_start
  0.00% nbody [vdso] [.] 0x00000000000000947
  0.00% nbody libc-2.24.so [.] _dl_addr
  0.00% nbody nbody [.] gettimeofday@plt
  0.00% nbody ld-2.24.so [.] dl_main
  0.00% nbody [kernel.kallsyms] [k] page_fault
  0.00% nbody ld-2.24.so [.] _start
```

Профилирование последовательной программы (GNU gprof)

```
$ gcc ... nbody.c -o nbody -pg
$ ./nbody 100
# NBody (n=100)
# Elapsed time (sec): ttotat 9.584010, tinit 0.000045, tfoces 9.449003, tmove 0.128511

$ gprof ./nbody
```

Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self Ts/call	total Ts/call	name
98.99	9.40	9.40				calculate_forces
0.95	9.49	0.09				move_particle

Параллельная версия 1

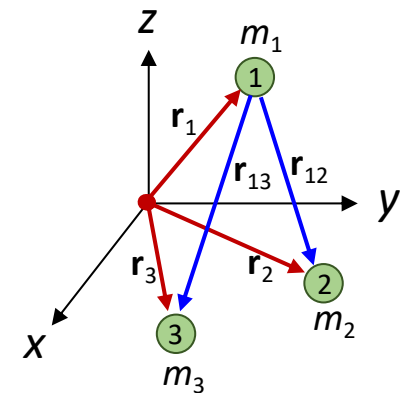
```
void calculate_forces(struct particle *p, struct particle *f, float *m, int n)
{
    #pragma omp parallel for
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            float dist = sqrtf(powf(p[i].x - p[j].x, 2) +
                                powf(p[i].y - p[j].y, 2) +
                                powf(p[i].z - p[j].z, 2));

            float mag = (G * m[i] * m[j]) / powf(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;
        }
    }
}
```

Распределим по P потокам вычисление сил для всех N тел

Разделяемые переменные есть?
Синхронизация требуется?



Параллельная версия 1

```
void calculate_forces(struct particle *p, struct particle *f, float *m, int n)
{
    #pragma omp parallel for
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            float dist = sqrtf(powf(p[i].x - p[j].x, 2) +
                                powf(p[i].y - p[j].y, 2) +
                                powf(p[i].z - p[j].z, 2));

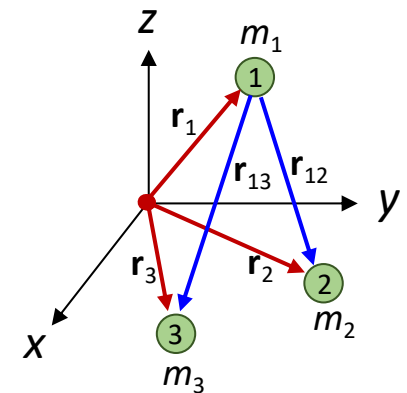
            float mag = (G * m[i] * m[j]) / powf(dist, 2);
            struct particle dir = {
                .x = p[j].x - p[i].x,
                .y = p[j].y - p[i].y,
                .z = p[j].z - p[i].z
            };
            f[i].x += mag * dir.x / dist;
            f[i].y += mag * dir.y / dist;
            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;
        }
    }
}
```

Распределим по P потокам вычисление сил для всех N тел

Разделяемые переменные есть?
Синхронизация требуется?

Необходимо атомарно обновлять $f[i], f[j]$



Параллельная версия 1: проблемы

```
void calculate_forces(struct particle *p, struct particle *f, float *m, int n)
{
    #pragma omp parallel for
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            float dist = sqrtf(powf(p[i].x - p[j].x, 2) +
                                powf(p[i].y - p[j].y, 2) +
                                powf(p[i].z - p[j].z, 2));

            float mag = (G * m[i] * m[j]) / powf(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
            };
            #pragma omp critical
            {
                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;
            }
        }
    }
}
```

Потоки загружены неравномерно

i = 0: (0,0), (0, 1), (0, 2), (0, 3), (0, 4),
i = 1: (1, 2), (1, 3), (1, 4),
i = 2: (2, 3), (2, 4)
i = 3: (3, 4)

Потокобезопасное обновление сил

1. Использовать одну критическую секцию
#pragma omp critical – за нее конкурируют все потоки
2. Использовать атомарную операцию для обновления каждой компоненты вектора сил
3. Использовать отдельную блокировку для каждой частицы (массив) – снижает конкуренцию потоков за блокировку
4. Реализовать алгоритм без использования блокировок

Параллельная версия 2: atomic

```
double dt = 1e-5;
#pragma omp parallel      // Параллельный регион активируется один раз
{
    for (double t = 0; t <= 1; t += dt) {
        calculate_forces(p, f, m, n);

        #pragma omp barrier    // Ожидание завершения расчетов f[i]
        move_particles(p, f, v, m, n, dt);

        #pragma omp barrier    // Ожидание завершения обновления p[i], f[i]
    }
}
ttotal = wtime() - ttotal;
printf("# NBody (n=%d)\n", n);
printf("# Elapsed time (sec): ttotal %.6f, tinit %.6f, tforces %.6f, tmove %.6f\n",
        ttotal, tinit, tforces, tmove);
```

Параллельная версия 2: atomic

```
void calculate_forces(struct particle *p, struct particle *f, float *m, int n)
{
    #pragma omp for schedule(dynamic, 4) nowait           // Циклическое распределение итераций
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            float dist = sqrtf(powf(p[i].x - p[j].x, 2) + powf(p[i].y - p[j].y, 2) + powf(p[i].z - p[j].z, 2));
            float mag = (G * m[i] * m[j]) / powf(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
            };
            #pragma omp atomic
            f[i].x += mag * dir.x / dist;
            #pragma omp atomic
            f[i].y += mag * dir.y / dist;
            #pragma omp atomic
            f[i].z += mag * dir.z / dist;
            #pragma omp atomic
            f[j].x -= mag * dir.x / dist;
            #pragma omp atomic
            f[j].y -= mag * dir.y / dist;
            #pragma omp atomic
            f[j].z -= mag * dir.z / dist;
        }
    }
}
```


Параллельная версия 2: atomic

```
void move_particles(struct particle *p, struct particle *f, struct particle *v,
                  float *m, int n, double dt)
{
    #pragma omp for nowait
    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;
    }
}
```

Параллельная версия 3: N блокировок

```
omp_lock_t *locks;    // Массив из N блокировок (мьютексов)

void calculate_forces(struct particle *p, struct particle *f, float *m, int n)
{
    #pragma omp for schedule(dynamic, 4) nowait
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            float dist = sqrtf(powf(p[i].x - p[j].x, 2) + powf(p[i].y - p[j].y, 2) + powf(p[i].z - p[j].z, 2));
            float mag = (G * m[i] * m[j]) / powf(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
            };
            omp_set_lock(&locks[i]);
            f[i].x += mag * dir.x / dist;
            f[i].y += mag * dir.y / dist;
            f[i].z += mag * dir.z / dist;
            omp_unset_lock(&locks[i]);
            omp_set_lock(&locks[j]);
            f[j].x -= mag * dir.x / dist;
            f[j].y -= mag * dir.y / dist;
            f[j].z -= mag * dir.z / dist;
            omp_unset_lock(&locks[j]);
        }
    }
}
```

Параллельная версия 3: N блокировок

```
int main(int argc, char *argv[])
{
    // ...
    locks = malloc(sizeof(omp_lock_t) * n);
    for (int i = 0; i < n; i++)
        omp_init_lock(&locks[i]);

    double dt = 1e-5;
    #pragma omp parallel
    {
        for (double t = 0; t <= 1; t += dt) {
            calculate_forces(p, f, m, n);
            #pragma omp barrier
            move_particles(p, f, v, m, n, dt);
            #pragma omp barrier
        }
    }
    // ...

    free(locks);
    // ...
}
```

Параллельная версия 4: дополнительные вычисления вместо блокировок

```
void calculate_forces(struct particle *p, struct particle *f, float *m, int n)
{
    #pragma omp for schedule(dynamic, 4) nowait
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i == j)
                continue;

            float dist = sqrtf(powf(p[i].x - p[j].x, 2) +
                                powf(p[i].y - p[j].y, 2) +
                                powf(p[i].z - p[j].z, 2));
            float mag = (G * m[i] * m[j]) / powf(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;
        }
    }
}
```

Поток, отвечающий за тело i , полностью вычисляет силы действия со стороны всех тел (не используется симметричность сил между делами, вычисления дублируются)

Параллельная версия 5:

дополнительная память вместо вычислений

```
void calculate_forces(struct particle *p, struct particle *f[], float *m, int n)
{
    int tid = omp_get_thread_num();
    int nthreads = omp_get_num_threads();
    for (int i = 0; i < n; i++) {
        f[tid][i].x = 0;
        f[tid][i].y = 0;
        f[tid][i].z = 0;
    }
    #pragma omp for schedule(dynamic, 8)
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            float dist = sqrtf(powf(p[i].x - p[j].x, 2) + powf(p[i].y - p[j].y, 2) + powf(p[i].z - p[j].z, 2));
            float mag = (G * m[i] * m[j]) / powf(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[tid][i].x += mag * dir.x / dist;
            f[tid][i].y += mag * dir.y / dist;
            f[tid][i].z += mag * dir.z / dist;
            f[tid][j].x -= mag * dir.x / dist;
            f[tid][j].y -= mag * dir.y / dist;
            f[tid][j].z -= mag * dir.z / dist;
        }
    }
}
```

- Вместо вектора сила используется матрица сил
- Число строк в матрице равно количеству потоков
- Дублируется память, а не вычисления

Параллельная версия 5: дополнительная память вместо вычислений

```
#pragma omp single    // Суммарная сила будет храниться в первой строке – f[0][i]
{
    for (int i = 0; i < n; i++) {
        for (int tid = 1; tid < nthreads; tid++) {
            f[0][i].x += f[tid][i].x;
            f[0][i].y += f[tid][i].y;
            f[0][i].z += f[tid][i].z;
        }
    }
}

} // calculate_forces
```

Параллельная версия 5:

дополнительная память вместо вычислений

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

Параллельная версия 5: дополнительная память вместо вычислений

```
int main(int argc, char *argv[])
{
    // ...

    struct particle *f[omp_get_max_threads()];
    for (int i = 0; i < omp_get_max_threads(); i++)
        f[i] = malloc(sizeof(struct particle) * n);

    // ...
    return 0;
}
```


Варианты балансировки загрузки

```
void calculate_forces(struct particle *p, struct particle *f, float *m, int n)
{
    #pragma omp for schedule(dynamic, 4) nowait
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            // Обработка пары (i, j)
        }
    }
}
```



Треугольное пространство итераций



```
void calculate_forces(struct particle *p, struct particle *f, float *m, int n)
{
    #pragma omp for collapse(2) schedule(dynamic, 1024) nowait
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            // Обработка пары (i, j)
        }
    }
}
```

Варианты балансировки загрузки

```
void calculate_forces(struct particle *p, struct particle *f, float *m, int n)
{
    #pragma omp for schedule(dynamic, 4) nowait
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            // Обработка пары (i, j)
        }
    }
}
```

Треугольное пространство итераций



Fusing a triangular loop (слияние циклов)

```
void calculate_forces(struct particle *p, struct particle *f, float *m, int n)
{
    #pragma omp for schedule(dynamic, 1024) nowait
    for (int w = 0; w < n * (n - 1) / 2; w++) {
        int i = (-1 + sqrt(1.0 + 8.0 * w)) / 2;
        int j = w - i * (i + 1) / 2;

        // Обработка пары (i, j)
    }
}
```