ОБ ОДНОЙ ПРОГРАММЕ МОДЕЛИРОВАНИЯ МОЛЕКУЛЯРНОЙ ДИНАМИКИ ГАЗА С ЭЛЕМЕНТАМИ РАСПАРАЛЛЕЛИВАНИЯ АЛГОРИТМА

В. Мирный, М. Фрэнер
Бранденбургский технический университет, Котбус, Германия
e-mail: myrnyvo@tu-cottbus.de

This article is devoted to the modelling of Newtonian molecular dynamics of gas. Modern numerical methods for solution of the problem are considered. On the basis of these methods the software package on C++ with usage of parallel computations is developed.

Введение

Задача моделирования молекулярной динамики в последнее время набрала большую популярность, так как позволяет, моделируя движение молекул на микроуровне, изучать физические макрохарактеристики определенных веществ. В ньютоновской модели молекулярной динамики газа движение молекул описывается системой обычных дифференциальных уравнений, которая разрешима аналитически только для двух молекул [1]. А интерес представляет рассмотрение систем более чем 1000 молекул. Поэтому необходимо применение современных и наиболее подходящих для этой задачи численных методов, таких, как многошаговые методы предиктор-корректор, представление Гира, а также симплектический метод для интегрирования гамильтоновых систем. Так как вычислительная сложность такой задачи велика, целесообразно для повышения эффективности разложить программу расчетов на несколько параллельных вычислений, т. е. несколько параллельно работающих независимых программ, способных синхронизироваться и обмениваться данными. Такой программный пакет может быть реализован на языке C++ в любой UNIX-подобной операционной системе.

1. Постановка задачи моделирования молекулярной динамики

В ньютоновской модели динамики газа исследуется движение сферической молекулы i под действием некоторой силы $\mathbf{F}_i$, созданной другими такими же молекулами. Движение и сила $\mathbf{F}_i$ подчиняются законам Ньютона:

$$\mathbf{F}_i = m\ddot{\mathbf{r}}_i, \quad \mathbf{f}_{ij} = -\mathbf{f}_{ji}, \tag{1}$$

где $\mathbf{f}_{ij}$ — сила, с которой молекула i действует на молекулу j.

Мы рассматриваем систему из N молекул в замкнутой прямоугольной области, в которой сохраняется общая энергия E. Определим энергию E как гамильтониан $\mathcal{H}$, зависящий от координат и скорости частиц:

$$E = \mathcal{H}(\mathbf{r}^N, \mathbf{p}^N) = E_K + \mathcal{U} = \text{const}, \tag{2}$$

где импульс и кинетическая энергия соответственно равны:

$$\mathbf{p}_i = m\dot{\mathbf{r}}_i, \quad E_K = \frac{1}{2m}\sum_i \mathbf{p}_i^2. \tag{3}$$

В такой системе выполняются первое и второе уравнения движения Гамильтона [1]:

$$\frac{\partial \mathcal{H}}{\partial \mathbf{p}_i} = \frac{\mathbf{p}_i}{m} = \dot{\mathbf{r}}_i, \quad \frac{\partial \mathcal{H}}{\partial \mathbf{r}_i} = -\dot{\mathbf{p}}_i. \tag{4}$$

Отсюда, используя второй закон Ньютона (1), получим выражение для силы:

$$\frac{\partial \mathcal{H}}{\partial \mathbf{r}_i} = -m\ddot{\mathbf{r}}_i \Longrightarrow \mathbf{F}_i = -\frac{\partial \mathcal{H}}{\partial \mathbf{r}_i} = -\frac{\partial \mathcal{U}}{\partial \mathbf{r}_i}. \tag{5}$$

Межмолекулярный потенциал $\mathcal{U}$ вычисляется как сумма потенциалов между каждыми двумя частицами. Учитывая третий закон Ньютона, среди пар молекул (i, j) следует перебрать только те, для которых $i < j$, отсюда

$$\mathcal{U}(\mathbf{r}^N) = \sum\sum_{i<j} u(r_{ij}), \quad \mathbf{r}^N = (r_1, r_2, \ldots, r_N)^T, \quad r_{ij} = |\mathbf{r}_i - \mathbf{r}_j|. \tag{6}$$

В качестве потенциала между парами частиц $u(r_{ij})$ можно использовать известный потенциал Леннарда — Джонса. При сближении частиц он асимптотически возрастает, а при удалении стремится к нулю. Поэтому часто радиус взаимодействия молекул ограничивают константой r_c, тогда потенциал Леннарда — Джонса принимает вид

$$u(r) = \begin{cases} 4\epsilon\left[\left(\dfrac{\sigma}{r}\right)^{12} - \left(\dfrac{\sigma}{r}\right)^{6}\right], & r \le r_c, \\ 0, & r > r_c, \end{cases} \tag{7}$$

где ϵ — энергия одной молекулы; σ — ее диаметр. Обычно r_c берется равным 2.5σ, тогда $u = -0.0163\epsilon$ и $F = -0.039\epsilon/\sigma$, т. е. взаимодействие частиц на таком удалении уже достаточно мало. Подставляя потенциал (7) в (5), получим силу взаимодействия двух частиц на расстоянии r_{ij}:

$$f_{ij}(r_{ij}) = -\frac{du(r_{ij})}{dr_{ij}} = 48\epsilon\left[\left(\frac{\sigma}{r_{ij}}\right)^{13} - 0.5\left(\frac{\sigma}{r_{ij}}\right)^{7}\right]. \tag{8}$$

Таким образом построенная система из N частиц описывается системой из $3N$ обыкновенных дифференциальных уравнений второго порядка (трехмерный случай) или системой

из $6N$ дифференциальных уравнений первого порядка:

$$\begin{cases} \dots \\ \dot{r}_{ix} = v_{ix}, \\ \dot{r}_{iy} = v_{iy}, \quad i = 1, \dots, N, \\ \dot{r}_{iz} = v_{iz}, \\ \dots \\ \dot{v}_{ix} = 48 \sum\limits_{\substack{j=1 \\ j \neq i}}^{N} \left(r_{ij}^{-14} - \dfrac{1}{2} r_{ij}^{-8} \right) r_{ijx}, \\ \dot{v}_{iy} = 48 \sum\limits_{\substack{j=1 \\ j \neq i}}^{N} \left(r_{ij}^{-14} - \dfrac{1}{2} r_{ij}^{-8} \right) r_{ijy}, \quad i = 1, \dots, N, \\ \dot{v}_{iz} = 48 \sum\limits_{\substack{j=1 \\ j \neq i}}^{N} \left(r_{ij}^{-14} - \dfrac{1}{2} r_{ij}^{-8} \right) r_{ijz}, \\ \dots \end{cases} \tag{9}$$

где

$$r_{ijx} = r_{ix} - r_{jx}, \quad r_{ijy} = r_{iy} - r_{jy}, \quad r_{ijz} = r_{iz} - r_{jz}, \quad r_{ij} = \sqrt{r_{ijx}^2 + r_{ijy}^2 + r_{ijz}^2}; \tag{10}$$

все физические величины рассматриваются без констант m, ϵ и σ.

Для того чтобы с помощью конечного числа молекул описать бесконечное пространство, применяются *периодические краевые условия*. Они заключаются в том, что молекула, вылетающая за границу рассматриваемой области, влетает в нее с другой стороны так же, как она влетела бы в соседнюю область (рис. 1). При этом поле действия частицы, находящейся у одной границы области, распространяется аналогичным образом на частицы, находящиеся у другой границы, т. е. следует рассматривать не только сами молекулы, но и их образы, находящиеся за границами области. Ясно, что для однозначности взаимодействия частиц необходимо выполнение условия $L > 2r_c$, т. е. молекула взаимодействует только с одним образом или самой молекулой. Потенциал (6) теперь следует вычислять с учетом перебора образов для каждой молекулы:

$$\mathcal{U} = \sum_{\alpha} \sum \sum_{i<j} u(\mathbf{r}_{ij}^{(0,0)} - \alpha L). \tag{11}$$

Для решения системы (9) необходимо определить *начальные условия*. Предположим, что в начальный момент времени молекулы расположены в серединах ячеек, получающихся при разбивке рассматриваемой прямоугольной области с помощью равномерной сетки. Начальные скорости вычисляются с помощью равномерно распределенных на отрезке $[-1; 1]$ случайных величин ξ_x, ξ_y и ξ_z следующим образом:

$$v_{ix}(0) = \frac{\xi_x}{\xi}, \quad v_{iy}(0) = \frac{\xi_y}{\xi}, \quad v_{iz}(0) = \frac{\xi_z}{\xi}, \tag{12}$$

где $\xi = \sqrt{\xi_x^2 + \xi_y^2 + \xi_z^2}$. Значения скоростей далее могут быть масштабированы с учетом начальной температуры или кинетической энергии:

$$v_{ix} = v_{ix} \sqrt{\frac{E_k^D}{E_k^A}} \quad \text{или} \quad v_{ix} = v_{ix} \sqrt{\frac{T_D}{T_A}}, \tag{13}$$

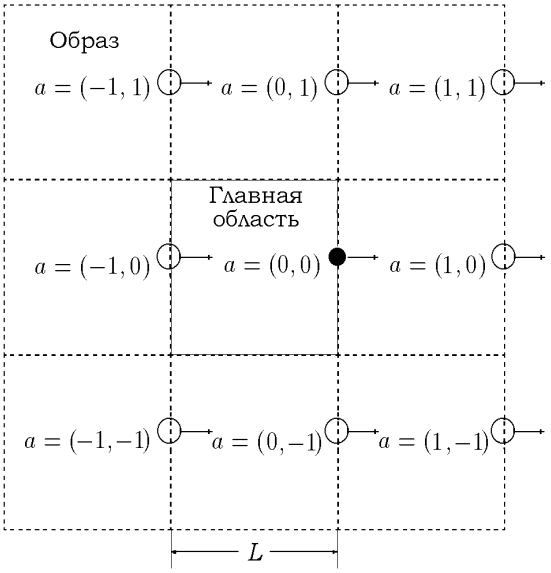

Рис. 1. Периодические краевые условия (двумерный случай).

где $T = \dfrac{2}{d}E_k$; E_k^A — заданное начальное значение кинетической энергии; E_k^D — значение кинетической энергии, вычисленное на основе начальных скоростей (3); d — размерность рассматриваемой задачи. Начальные условия для координат и скоростей могут быть также взяты из предыдущего моделирования.

Описанная модель позволяет применить методы для уменьшения вычислительной сложности потенциала. Одним из таких методов является "Список соседей" (Neighbor list), предложенный Верлетом [1, 2]. Суть его заключается в том, чтобы не перебирать на каждом шаге интегрирования всевозможные пары молекул, вычисляя между ними расстояния, а сохранять в списке те, которые находятся на удалении не более r_L. Список должен обновляться каждые M шагов интегрирования ($5 \le M \le 30$) с учетом того, что молекулы за это время не проходят расстояние больше $\Delta r_L = r_L - r_c$. Константа Δr_L обычно берется равной 0.3σ. Такой метод немного снижает точность, но позволяет выиграть до 20 % времени вычислений. Различные источники предлагают разные подходы к организации этого списка. Можно хранить номера только взаимодействующих между собой молекул, но тогда заранее не известен размер требуемой памяти и он выбирается из примерного числа взаимодействующих одновременно молекул [1]. Другой подход предлагает хранить информацию о каждой паре [2], тогда потребуется $\dfrac{N(N-1)}{2}$ битов, т. е. $\dfrac{N(N-1)}{16}$ байт памяти. Возможна модификация этого подхода. В массиве $\dfrac{N(N-1)}{2}$ байт хранит номер, определяющий, с каким из образов либо самой молекулой происходит взаимодействие. Это освобождает от перебора образов на каждом шаге и дает выигрыш примерно в 5 % вычислительного времени, хотя и требует дополнительных затрат памяти.

Итак, решение системы (9) позволяет определить точные траектории движения частиц и физические макровеличины состояния (энергию, температуру, давление, коэффициент диффузии, вязкость и др.) [1, 2].

2. Численное решение

Основная трудность при компьютерном моделировании рассматриваемой задачи заключается в том, что функция в правой части системы (9) имеет сложность $O(N^2)$. Ее вычисление занимает основную долю времени при реализации алгоритма численного интегрирования. Этим в большей степени обусловлен выбор численных методов. В таких условиях наиболее эффективны многошаговые методы. Самыми известными из них являются методы Адамса — Башфорта (А – Б) и Адамса — Моултона (А – М) или соответственно экстраполяционный и интерполяционный методы Адамса, а также их комбинация — методы *предиктор-корректор*.

Введем следующие обозначения. Пусть y_n — искомое приближенное решение на шаге n, h — шаг интегрирования. В табл. 1 представлена общая схема методов предиктор-корректор для системы дифференциальных уравнений как первого, так и второго порядков. Последний называется также методом Штермера. На первом шаге методов (Prediction) вычисляется начальное приближение. Затем находится значение функции (Evaluation), на основе которого далее производится одно или несколько уточнений (Correction) решения на данном шаге. Между шагами P – E и C – E может выполняться модифицирование (Modification) за счет оценки локальной погрешности на шаге интегрирования [3]. Таким образом, существуют следующие схемы методов предиктор-корректор: PE(CE)^m, P(EC)^m и PME(CME)^m. При расчетах по второй схеме экономится компьютерное время на одном вычислении функции, но при этом немного теряется точность.

Таблица 1

	Система первого порядка	Система второго порядка
P	$P_{n+1} = y_n + h\sum\limits_{i=1}^{k} \beta_i^* y'_{n+1-i}$	$P_{n+1} = 2y_n - y_{n-1} + h\sum\limits_{i=1}^{k} \beta_i^* y'_{n+1-i}$
M	$P_{n+1}^{(m)} = P_{n+1} - \dfrac{A_P}{(A_P - A_C)}(P_{n+1} - C_{n+1})$	
E	$y'_{n+1} = f(t_{n+1}, P_{n+1}^{(m)})$	
C	$C_{n+1} = y_n + h\sum\limits_{i=0}^{k} \beta_i y'_{n+1-i}$	$C_{n+1} = 2y_n - y_{n-1} + h\sum\limits_{i=0}^{k} \beta_i y'_{n+1-i}$
M	$C_{n+1}^{(m)} = C_{n+1} - \dfrac{A_C}{(A_P - A_C)}(P_{n+1} - C_{n+1})$	
E	$y'_{n+1} = f(t_{n+1}, C_{n+1}^{(m)}), \quad y_{n+1} = C_{n+1}^{(m)}$	

Примечание. Локальная погрешность порядка q:

$$T_{n+1}^{(P)} = A_P h^{q+1} y_n^{(q+1)} + O(h^{q+2}),$$

$$T_{n+1}^{(C)} = A_C h^{q+1} y_n^{(q+1)} + O(h^{q+2});$$

β_i и β_i^*, A_P и A_C — константы методов [3, 4], приведенные в табл. 2 и 3.

Таблица 2

Метод	α_1^*	α_2^*	β_1^*	β_2^*	β_3^*	β_4^*	β_5^*	β_6^*	A_P
Euler	1	0	1	0					$\frac{h^2}{2}y^{[5]}$
А–Б, $k=2$	1	0	$\frac{3}{2}$	$-\frac{1}{2}$	0				$\frac{5h^3}{12}y^{[3]}$
А–Б, $k=3$	1	0	$\frac{23}{12}$	$-\frac{16}{12}$	$\frac{5}{12}$	0			$\frac{9h^4}{24}y^{[6]}$
А–Б, $k=4$	1	0	$\frac{55}{24}$	$-\frac{59}{24}$	$\frac{37}{24}$	$-\frac{9}{24}$	0		$\frac{251h^5}{720}y^{[7]}$
А–Б, $k=5$	1	0	$\frac{1901}{720}$	$-\frac{2774}{720}$	$\frac{2616}{720}$	$-\frac{1274}{720}$	$\frac{251}{720}$	0	$\frac{475h^6}{1440}y^{[1]}$
А–Б, $k=6$	1	0	$\frac{4277}{1440}$	$-\frac{7923}{1440}$	$\frac{9982}{1440}$	$-\frac{7298}{1440}$	$\frac{2877}{1440}$	$-\frac{475}{1440}$	$\frac{19087h^7}{60480}y^{[8]}$

Таблица 3

Метод	α_1	α_2	β_0	β_1	β_2	β_3	β_4	β_5	A_C
А–М, $k=1$	1	0	$\frac{1}{2}$	$\frac{1}{2}$	0				$-\frac{h^3}{12}y^{[3]}$
А–М, $k=2$	1	0	$\frac{5}{12}$	$\frac{8}{12}$	$-\frac{1}{12}$	0			$-\frac{h^4}{24}y^{[6]}$
А–М, $k=3$	1	0	$\frac{9}{24}$	$\frac{19}{24}$	$-\frac{5}{24}$	$\frac{1}{24}$	0		$-\frac{19h^5}{720}y^{[7]}$
А–М, $k=4$	1	0	$\frac{251}{720}$	$\frac{646}{720}$	$-\frac{-264}{720}$	$\frac{106}{720}$	$-\frac{19}{720}$	0	$-\frac{27h^6}{1440}y^{[1]}$
А–М, $k=5$	1	0	$\frac{475}{1440}$	$\frac{1427}{1440}$	$-\frac{798}{1440}$	$\frac{482}{1440}$	$-\frac{173}{1440}$	$\frac{27}{1440}$	$-\frac{863h^7}{60480}y^{[8]}$

Иногда удобней на каждом шаге метода вместо предыдущих решений $y(t_n)$, $y(t_{n-1})$, ..., $y(t_{n-k+1})$ использовать значения производных $y(t_n)$ до k-го порядка включительно. Такую идею предложил в 1962 г. Нордсик, а вектор, составленный из этих значений, получил название Нордсик-вектора:

$$Z_n := \left[y(t_n), hy'(t_n), \frac{h}{2}y''(t_n), \ ..., \ \frac{h^k}{k!}y^{(k)}(t_n) \right].$$

Разработанный позже на его основе метод называется представлением Гира и имеет следующую схему:

P: $Z_n^{[0]} = \Pi Z_{n-1}^{[\mu]}$,

C: $Z_n^{[\nu+1]} = Z_n^{[\nu]} + Gh(f_n^{P[\nu]} - f_n^{[\nu]})$, $\nu = 1, 2, \ ..., \ \mu - 1$,

где f_n^P — значение первой производной из второго компонента Нордсик-вектора; f_n — значение функции из приближения P; Π — треугольная матрица Паскаля, G — вектор констант, полученных из экстраполяционного и интерполяционного методов Адамса [6, 9];

$$\Pi = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 0 & 1 & 2 & 3 & \cdots & \binom{k}{1} \\ 0 & 0 & 1 & 3 & \cdots & \binom{k}{2} \\ 0 & 0 & 0 & 1 & \cdots & \binom{k}{3} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 \end{bmatrix}, \quad G = \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_n \end{pmatrix},$$

элементы вектора G принимают значения, приведенные в табл. 4.

Таблица 4

Значения элементов вектора G для системы										
первого порядка							второго порядка			
k	1	2	3	4	5	6	k	2	3	4
g_1	$\frac{1}{2}$	$\frac{5}{12}$	$\frac{3}{8}$	$\frac{251}{720}$	$\frac{95}{288}$	$\frac{19087}{2}$	g_1	$\frac{1}{6}$	$\frac{19}{120}$	$\frac{3}{20}$
g_2	1	1	1	1	1	1	g_2	$\frac{5}{6}$	$\frac{3}{4}$	$\frac{251}{360}$
g_3		$\frac{1}{2}$	$\frac{3}{4}$	$\frac{11}{12}$	$\frac{25}{24}$	$\frac{137}{120}$	g_3	1	1	1
g_4			$\frac{1}{6}$	$\frac{1}{3}$	$\frac{35}{72}$	$\frac{5}{8}$	g_4	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{11}{18}$
g_5				$\frac{1}{24}$	$\frac{5}{48}$	$\frac{17}{96}$	g_5		$\frac{1}{12}$	$\frac{1}{6}$
g_6					$\frac{1}{120}$	$\frac{1}{40}$	g_6			$\frac{1}{60}$
g_7						$\frac{1}{720}$				

Следует заметить, что в представлении Гира изменение временного шага h осуществляется простым умножением Нордсик-вектора на диагональную матрицу

$$Z_{\theta_h} = \mathrm{diag}(1, \theta, \theta^2, \ldots, \theta^k) Z_h, \quad \theta = \frac{h_{\mathrm{new}}}{h}.$$

При этом изменение порядка метода ведет к сложным дополнительным вычислениям. В обычной же записи многошаговых методов (см. табл. 1), т. е. через ньютоновский интерполяционный многочлен, наоборот, изменение временного шага h требует пересчета нескольких предыдущих решений, а изменение порядка метода элементарно. Таким образом, методы предиктор-корректор и представление Гира, имея одинаковую точность, применяются в зависимости от того, какие параметры метода будут изменяться: шаг интегрирования или порядок метода.

Для любого типа многошаговых методов необходимо вычисление первых k шагов решения каким-либо одношаговым методом. Может быть применен обычный метод Рунге—Кутты, но более удобным для этого является *симплектический метод* [7]. Это самый новый из рассматриваемых здесь методов. Он предложен в конце 80-х годов для интегрирования гамильтоновых систем и рекомендуется для моделирования молекулярной динамики. При решении данной задачи этот метод может быть применен самостоятельно либо

для нахождения начальных шагов при использовании других многошаговых методов. Его суть заключается в следующем. Пусть рассматриваемый гамильтониан представим в виде

$$\mathcal{H} = A(\mathbf{p}) + V(\mathbf{x}),$$

где в нашем случае $A(\mathbf{p})$ — кинетическая энергия (3), а $V(\mathbf{x})$ — потенциальная энергия взаимодействия молекул. Тогда численная схема симплектических методов выглядит так:

$$\mathbf{p}_{t,i+1} = \mathbf{p}_{t,i} - c_i h V_{\mathbf{x}}(\mathbf{x}_{t,i}),$$
$$\mathbf{x}_{t,i+1} = \mathbf{x}_{t,i} - d_i h A_{\mathbf{p}}(\mathbf{p}_{t,i+1}),$$

где t — текущее время и $i = 1, ..., p$ (p — порядок симплектического метода). В нашей задаче производные функций A и V принимают вид

$$A_{\mathbf{p}}(\mathbf{p}) = \frac{1}{m}\sum_{i=1}^{N}\mathbf{p}_i, \quad V_{\mathbf{x}}(\mathbf{x}) = \frac{d\mathcal{U}}{d\mathbf{r}} = -\mathbf{F}(\mathbf{r}).$$

Ниже приведены коэффициенты методов c_i и d_i до четвертого порядка включительно [7]:

$p=2$	$c_1 = 0,$	$d_1 = \frac{1}{2},$		
	$c_2 = 1,$	$d_2 = \frac{1}{2}.$		
$p=3$	$c_1 = \frac{7}{24},$	$c_2 = \frac{3}{4},$	$c_3 = -\frac{1}{24},$	
	$d_1 = \frac{2}{3},$	$d_2 = -\frac{2}{3},$	$d_3 = 1.$	
$p=4$	$x = \frac{2^{1/3} + 2^{-1/3} - 1}{6} = 0.1756...$			
	$c_1 = 0,$	$c_2 = 2x+1,$	$c_3 = -4x-1,$	$c_4 = 2x+1,$
	$d_1 = x + \frac{1}{2},$	$d_2 = -x,$	$d_3 = -x,$	$d_4 = x + \frac{1}{2}$

3. Разработка программного пакета

Реализация численных методов, а также вычисление физических макрохарактеристик достаточно просты и могут быть выполнены с помощью любого высокоуровневого языка программирования. Программа должна получать входные параметры, считывая их из текстового файла или командной строки, и сохранять вычисленные результаты в файлах на диске. Организовать такой ввод/вывод не представляет сложности. Главной же задачей является достижение максимальной производительности программы, чтобы моделировать систему как можно большего числа молекул и затрачивать на это минимальное вычислительное время. Для этого в первую очередь применяются обычные средства оптимизации программ: максимально снижается вычислительная сложность в часто повторяющихся блоках, сложные структуры данных и большие иерархии классов заменяются более простыми, копирование (присваивание) больших структур данных и массивов заменяется по

возможности изменением указателей на эти структуры и др. Когда все подобные приемы применены, остаются все-таки еще некоторые возможности для повышения производительности программы. Одной из них является использование параллельных вычислений, хотя реальный результат от их внедрения становится виден только на компьютерах с несколькими процессорами. Программы, использующие параллельные алгоритмы, хорошо работают и имеют ряд преимуществ и на однопроцессорных компьютерах с многозадачными операционными системами. Об этих преимуществах и разработке таких программ на примере моделирования молекулярной динамики подробно рассказывается далее.

3.1. Концепция разделения на параллельные процессы

Вычислительная сложность решаемой задачи велика, поэтому полезно весь расчет разбить на параллельные процессы с целью увеличить эффективность вычислений на компьютерах с более чем одним процессором либо в однопроцессорной системе избавиться от вычисления незатребованных пользователем результатов моделирования. Произведем разбивку задачи на параллельные процессы следующим образом (рис. 2). Пусть главная программа, назовем ее "сервер-процесс", решает систему (9) одним из выбранных пользователем численных методов. Такой сервер-процесс получает на вход начальные условия в виде списка параметров (температура, плотность и др.) и выдает векторы-решения, еще не пригодные для восприятия пользователем. Пусть несколько других программ, назовем их клиент-процессами, могут считывать эти векторы-решения, вычислять из них определенные физические величины и выдавать в необходимой для пользователя форме. Для данной задачи определим три клиент-процесса:

1. Программа, отображающая движение молекул в трехмерной кубической области.

2. Программа, отображающая двумерные графики следующих физических величин: общей энергии, кинетической энергии, потенциальной энергии, температуры, средней скорости, функции плотности распределения модуля скорости (распределение Максвелла).

3. Программа, выдающая численные результаты тех же физических величин в текстовом виде, а также полную информацию проведенных расчетов, т. е. векторы скорости и координат молекул, хотя они и мало пригодны для восприятия.

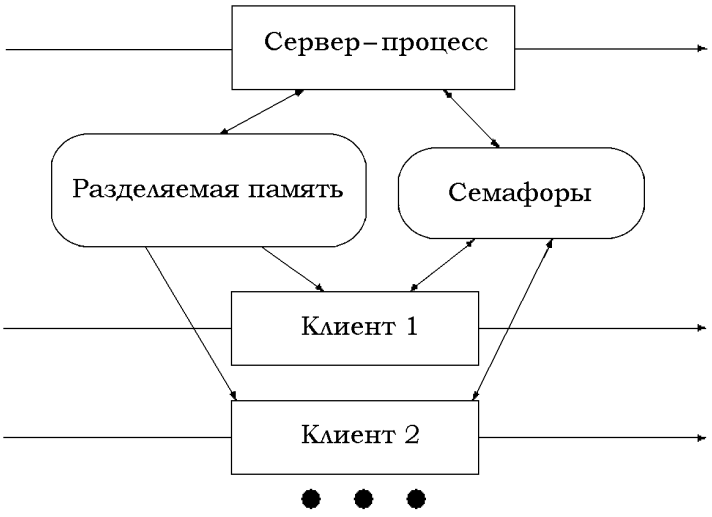

Рис. 2. Взаимодействие параллельных процессов.

Возникает вопрос, как эти процессы должны взаимодействовать, чтобы не потерять выигрыш в быстродействии, к которому мы стремимся? Взаимодействие параллельных процессов включает в себя два аспекта: *обмен данными* и *синхронизацию*. Операционная система UNIX предлагает для этого следующие возможности: механизм сообщений, разделяемую память, сигналы и семафоры [5, 10]. Наиболее быстрый и одновременно подходящий для нашей задачи вариант — это обмен данными через *разделяемую память* и синхронизация с помощью *семафоров*. Существуют два типа разделяемой памяти: общее адресное пространство в оперативной памяти и такое же пространство, но с отображением на диск в файл. Данные из такого файла можно просмотреть позже с помощью тех же клиент-процессов. Так как оба типа разделяемой памяти имеют одинаковое быстродействие, целесообразно воспользоваться ее вторым типом для сохранения вычисленных траекторий движения молекул и последующего их изучения. Этот вариант позволяет также считывать сохраненные данные теми же системными вызовами, что и при параллельной работе. Это означает, что клиент-программы могут как работать в параллельном режиме, так и отображать сохраненные данные с помощью одного и того же программного кода.

Семафоры были введены в 1968 г. голландским программистом Эдсгером В. Дейкстра (Edsger W. Dijkstra) в качестве средства синхронизации. Впоследствии теорию семафоров продолжили С. С. Патил (1971) и Л. Прессер (1975). Семафоры в UNIX-системе представляют собой целочисленные переменные, хранящиеся в области ядра UNIX. Они обладают одним важным свойством — не могут принимать отрицательные значения. Если процесс попытается изменить значение семафора так, чтобы оно стало отрицательным, эта операция и сам процесс будут заблокированы до тех пор, пока другой процесс не увеличит значание семафора до величины, достаточной для успешного выполнения операции заблокированного процесса (т. е. чтобы значение семафора после операции уменьшения было положительным либо равным нулю). Если также процесс проверит значение семафора на равенство нулю, когда на самом деле это не так, то ядро заблокирует вызывающий процесс. Эти принципы лежат в основе синхронизации процессов с помощью семафоров.

Определим теперь протокол взаимодействия параллельных процессов. Пусть выделенная разделяемая область памяти может вмещать ровно M векторов-решений, т. е. имеет размер $6NM \cdot 8$ байт (одна переменная типа DOUBLE занимает 8 байт). Тогда сервер-процесс может заполнять эту область памяти векторами-решениями, но количеством не более M. На M-м векторе сервер-процесс останавливается и ждет, пока хотя бы один клиент-процесс начнет считывать векторы-решения. После чего сервер-процесс продолжает циклически записывать новые решения поверх старых, начиная с начала разделяемой памяти. С другой стороны, если клиент-процесс считывает векторы-решения быстрее, чем сервер-процесс вычисляет новые, то клиент-процесс должен дождаться окончания расчета, чтобы не получить некорректных данных. Такой механизм нестрогой синхронизации позволяет повысить эффективность программы, а также удобство работы пользователя.

Другой аспект синхронизации — это несколько параллельно работающих клиент-процессов. Они должны быть строго синхронизированы, т. е. должны считывать одновременно один и тот же вектор-решение, чтобы представлять пользователю результаты, приходящиеся на один и тот же шаг интегрирования. При этом все клиент-процессы, работающие параллельно, должны синхронизироваться, как описано выше, с сервер-процессом. Кроме того, чтобы добиться максимальной эффективности от использования пакета параллельных программ, необходимо реализовать их так, чтобы один и тот же клиент-процесс мог быть запущен несколько раз параллельно. Например, клиент-процесс №2 запущен три раза с разными опциями для отображения соответственно общей, кинетической и потен-

циальной энергий, а также запущен клиент-процесс №3 для отображения тех же величин в текстовом виде.

3.2. Реализация системными средствами

Описанную модель программы нужно реализовать с помощью системных функций, описанных в библиотеках C++ *sys\sem.h* и *sys\mman.h* для семафоров и разделяемой памяти соответственно. Механизм семафоров работает с помощью следующих трех функций:

1. `int semget(key_t key, int num_sem, int flag)`. Эта функция открывает набор из *num_ sem* семафоров, определенный положительным ключом *key*, или подключает процесс к существующему набору семафоров, если набор с таким ключом уже существует. Для нового набора семафоров флаг *flag* должен представлять собой результат побитового сложения константы IPC_CREATE и числовых значений прав доступа к новому набору. Для подключения к существующему набору *flag* должен равняться нулю. Вызов возвращает неотрицательный целочисленный идентификатор, который затем следует использовать в остальных двух функциях работы с семафорами.

2. `int semop(int semid, struct sembuf* opPtr, int len)` изменяет значение одного или нескольких семафоров в наборе с идентификатором *semid*. Список семафоров с соответствующими им операциями задается указателем *opPtr* на массив из *len* структур следующего вида:

```
struct sembuf {
    unsigned short  sem_num;    /* индекс семафора в наборе */
    short           sem_op;     /* операция над семафором */
    short           sem_flg;    /* флаг(и) операции */
}
```

3. `int semctl(int semid, int num, int cmd, union semun arg)`. С помощью этой функции можно запрашивать и изменять управляющие параметры набора семафоров, указанного аргументом *semid*, а также удалять семафор. Аргумент *cmd* задает операцию, которая будет выполнена над семафором под номером *num* в наборе. Структура *arg* используется для задания или выборки управляющих параметров одного или нескольких семафоров в соответствии с аргументом *cmd*. Подробнее см. [10].

Функционирование семафоров становится значительно сложней и имеет больше возможностей за счет функции `semop()`, которая может изменять одним вызовом целую группу (вектор) семафоров.

Разделяемая память (РП) с отображением на диск обслуживается следующими тремя функциями:

— `caddr_t mmap(caddr_t addr, int size, int prot, int flags, int fd, off_t pos)` отображает файл, указанный аргументом *fd*, в виртуальное адресное пространство процесса, начиная с адреса *addr*. Если *addr* равен нулю, ядро UNIX само назначает виртуальный адрес для отображения. Аргумент *pos* задает начальную позицию в файле, а *prot* определяет права доступа к отображенной памяти.

— `int munmap(caddr_t addr, int size)` отсоединяет отображенную область памяти от виртуального адресного пространства процесса. Освобождаемая область начинается с виртуального адреса *addr* и имеет размер *size*.

— `int msync(caddr_t addr, int size, int flags)` синхронизирует данные, находящиеся в отображенной области памяти, с соответствующими данными файла, находящегося на диске.

Разделяемую область памяти заданного пользователем размера, как и семафоры, создает сервер-процесс и уничтожает их по окончании своей работы. Клиент-процессы при этом либо подключаются к ним, если сервер-процесс в это время работает, либо считывают сохраненные результаты из определенного пользователем файла. Наиболее сложными остаются вопросы, сколько семафоров необходимо и как они должны изменяться для правильного взаимодействия программ в параллельном режиме. Определим следующие семь семафоров:

№	Имя	Описание	Начальное значение
1	SEM_RD	Разрешение на чтение текущего вектора-решения из РП	0
2	SEM_WR	Разрешение на запись нового вектора-решения в РП	Кол-во векторов-решений в РП
3	SEM_CR	Номер читаемого вектора-решения	0
4	SEM_BR	Число заполнений РП	0
5	SEM_NP	Количество клиент-процессов, подключенных в данный момент к РП	0
6	SEM_NR	Количество клиент-процессов, которые уже прочитали текущий вектор-решение	0
7	SEM_WT	Вспомогательный семафор, с помощью которого клиент-процессы могут ожидать друг друга во время чтения одного вектора-решения	1

Сервер-процесс синхронизируется с клиент-процессами с помощью первых двух семафоров (SEM_RD и SEM_WR). Имея начальными значениями 0 и допустимое количество векторов-решений в разделяемой памяти, они являются своего рода ограничителями “свободы” процессов в пределах разделяемой памяти. Сервер-процесс, записывая в память новый вектор-решение, уменьшает SEM_WR на 1, а затем после записи увеличивает на 1 SEM_RD. Клиент-процесс, наоборот, сперва уменьшает на 1 SEM_RD, получая разрешение на чтение, а затем увеличивает SEM_WR на 1. Таким образом клиент-процесс не может считать вектор-решение, пока тот не записан сервер-процессом, а сервер-процесс в свою очередь не может записать вектор поверх другого, если он еще не был прочитан клиент-процессом. В такой синхронизации с сервер-процессом должен участвовать только один клиент-процесс. Он выделяется из множества всех параллельно работающих клиент-процессов с помощью семафоров SEM_NR, SEM_NP, SEM_CR и SEM_WT. Каждый вновь подключившийся клиент-процесс увеличивает SEM_NP на 1 и при завершении своей работы отнимает 1. На определенном шаге каждый клиент-процесс, прочитывая вектор-решение, прибавляет 1 к SEM_NR, а затем проверяет равенство SEM_NR и SEM_NP. Если оно выполняется, то этот процесс и есть тот один, который осуществляет синхронизацию с сервер-процессом. Семафор SEM_CR позволяет процессу проверить, находится ли он при чтении того же вектора-решения, что и все, и предотвратить преждевременный переход далее за счет семафора SEM_WT.

Рассмотрим алгоритмы синхронизации.

Сервер-процесс	Клиент-процесс
1. SEM_WR-1. Получение разрешения на запись нового вектора-решения	1. Если номер считываемого вектора-решения больше SEM_CR, то проверка SEM_NR на равенство нулю (ожидание SEM_NR = 0). 2. Если SEM_NR = 0, то синхронизация с сервер-процессом: SEM_NR + 1, SEM_RD −1; SEM_WT −1, иначе ожидание разрешения на чтение: проверка SEM_WT = 0, SEM_NR + 1
2. Запись нового вектора-решения в РП	3. Чтение текущего вектора-решения из РП и переход на следующий 4. Если SEM_NR = SEM_NP, то синхронизация с сервер-процессом
3. SEM_RD+1	Если номер вектора-решения равен нулю, т. е. начато новое заполнение РП, то SEM_BR + 1 SEM_WT = 1, SEM_NR = 0, SEM_CR = номер вектора-решения, SEM_WR + 1

Для параллельно работающих программ особым моментом является их завершение, так как даже в случае прерывания пользователем программа должна установить верные значения семафоров, чтобы остальные параллельные программы продолжали нормально работать либо корректно завершились. В нашей задаче сервер-процесс, завершаясь, дает знать об этом всем клиент-процессам, которые сразу же все прерываются, так как сервер-процесс при этом уничтожает все средства синхронизации. Клиент-процесс при завершении должен лишь отключиться от РП и уменьшить значение семафора SEM_NP на 1.

Для взаимодействия параллельных процессов при их завершении и одновременно для защиты от непредусмотренного прерывания пользователем или системой служат *сигналы*. В операционной системе UNIX определен стандартный набор сигналов, каждый из которых посылается процессу в определенной ситуации работы программы или системы в целом [5, 10]. Механизм сигналов позволяет определить в программе функцию-обработчик, которая будет вызвана при получении программой определенного сигнала. Этот сигнал может быть послан как системой, так и другим процессом. После получения сигнала программа прерывается, выполняется предопределенная функция-обработчик, а затем программа продолжает свою работу со следующей строчки после той, на которой она была прервана.

Другой аспект применения механизма сигналов — это создание так называемых таймеров. Если программа, такая, как, например, сервер-процесс, работает достаточно длительное время, то через определенные промежутки времени хотелось бы знать, на каком шаге численного решения она сейчас находится. Писать вывод такого сообщения в определенной строчке программы не совсем правильно, так как при разном количестве молекул каждый шаг численного интегрирования выполняется разное время. В этом случае удобно применить таймер. После его создания ядро UNIX посылает процессу сигнал SIGALRM через равные промежутки времени, которые программа может обрабатывать с помощью определенной функции и выводить необходимые сообщения. Такой таймер встроен в сервер-

процесс и выдает сообщения о том, сколько шагов интегрирования выполнено и за какое процессорное время. Следует заметить, что процессорное время в UNIX-системах часто отличается от обычного. Оно представляет собой время, затраченное на вычисление, если бы процессор выполнял на 100 % только его. Если же одним процессором выполняется одновременно два процесса и более, то процессорное время для одного из них становится значительно меньше реального.

3.3. Вычисление сложной функции двумя параллельными процессами

Так как вычисление функции системы (9) занимает бóльшую часть времени работы программы, то ее оптимизация способна дать наибольший выигрыш в быстродействии. Рассмотрим на ее примере, как одна функция может вычисляться двумя параллельными процессами с целью уменьшить вычислительное время. До сих пор мы считали, что программы вызываются пользователем независимо друг от друга. Теперь же вспомогательную программу для расчета функции будет вызывать сервер-процесс. Для создания одним процессом нового процесса служат системные вызовы fork() и exec() из библиотек *sys/types.h* и *unistd.h* соответственно [10]. В этом случае вызывающая программа называется родительским процессом, а вызванные им — порожденными процессами. Родительский процесс после вызова fork() получает целочисленный уникальный идентификатор (PID) порожденного процесса. Идентификатор процесса может быть использован для получения сведений о процессе или посылки ему сигналов. Порожденный процесс, в свою очередь, после вызова exec() отсоединяется от родительского и становится полностью независимым. Взаимодействие между параллельно работающими родительским и порожденным процессами осуществляется с помощью тех же средств, что описаны в п. 3.1. Мы же продолжим применять из них наилучшую пару — семафоры и разделяемую память.

Функция системы (9) вычисляет силы, действующие на каждую молекулу при заданных координатах и скоростях. Для этого должны быть перебраны всевозможные упорядоченные (т. е. $i < j$) пары (i, j) молекул. Если они находятся на расстоянии меньше r_c (7), то взаимодействуют с некоторой силой f_{ij} (8), которая добавляется к результирующей силе F_i, действующей на i-ю молекулу, и вычитается по третьему закону Ньютона (1) из силы F_j, действующей на j-ю молекулу. Такой перебор пар молекул с накоплением результирующих сил может быть разбит на две (и более) части для вычисления с помощью двух (и более) параллельных процессов. Например, один перебирает только четные, а второй только нечетные значения i при всех значениях $j > i$. При этом, конечно, массивы, содержащие исходные данные, выходные значения F_i и взаимодействующие пары, должны находиться в доступной обоим процессам (разделяемой) памяти. Вообще говоря, такое разбиение стало возможным, так как каждый шаг перебора не зависит от предыдущего шага, а только от исходных данных.

Сформулируем теперь, на каких условиях должна основываться синхронизация двух таких параллельных процессов. Вспомогательная программа, запущенная сервер-процессом, должна начинать выполнять перебор только одновременно с ним, чтобы сервер-процесс успел записать в РП корректные исходные данные, а сервер-процесс должен дождаться завершения перебора вспомогательным процессом, чтобы затем добавить его результаты из РП к своим. Таким образом, вспомогательный процесс считает в РП, а сервер-процесс — в своей локальной, а потом суммирует результаты. Наиболее эффективной может показаться одновременная работа процессов в РП, но тесты показали, что при

одновременной записи двумя процессами в РП по одному адресу одно из записываемых значений теряется.

Описанный механизм синхронизации возможно осуществить с помощью всего одного семафора. Введем новый семафор SEM_FC с начальным значением, равным 1. Тогда алгоритмы синхронизации будут выглядеть следующим образом.

Сервер-процесс	SEM_FC	Вспомогательный процесс
1. SEM_FC-1. Обнуление SEM_FC и разблокирование вспомогательного процесса	1	1. Проверка SEM_FC=0. Ожидание равенства нулю
2. Цикл перебора и вычислений f_{ij}	0	2. Цикл перебора и вычислений f_{ij}
3. SEM_FC-1. Ожидание, пока SEM_FC примет положительное значение	1	3. SEM_FC+2. После этого сервер-процесс может продолжать свою работу

Тестирование алгоритма с использованием такого вспомогательного процесса на двухпроцессорном компьютере показало, что время работы всей программы моделирования уменьшилось примерно на 25 % по сравнению со случаем, когда силы взаимодействия молекул вычислялись только одним процессом.

3.4. Особенности реализации и отладки параллельных алгоритмов

Разработка программ, способных работать параллельно, в значительной степени отличается от обычного программирования. Кроме обычной логики последовательного выполнения алгоритма здесь должно присутствовать понимание того, что второй параллельно работающий алгоритм полностью независим от первого и скорость выполнения одного по отношению к другому часто трудно предсказуема. Поэтому взаимодействие между программами может происходить только в точках синхронизации, когда четко известно, на каком шаге каждая из них находится и какие данные в себе несет. При этом все же стоит внимательно оценивать быстродействие программ, чтобы средствами синхронизации не заставлять программы простаивать длительное время. Организация взаимодействия ставит программы в тесную зависимость друг от друга, поэтому нужно обращать особое внимание на корректность передаваемых между программами данных, а также на избежание зависания одной из них и правильное завершение. При завершении программ должны быть обязательно уничтожены средства взаимодействия, так как иначе они остаются в области ядра UNIX и возможно помешают повторному запуску. Это, как правило, выполняет одна главная программа (сервер-процесс). Ярким примером нестандартной логики при разработке параллельных алгоритмов является, например, написание клиент-процесса, несколько экземпляров которого могут работать параллельно и взаимодействовать между собой.

Отладка таких программ также имеет свои особенности. Как правило, две программы, созданные для параллельной работы, не могут работать в одиночку без дополнительно специально написанного кода, так как останавливаются при уменьшении семафоров и ожидают их увеличения. Поэтому и отлаживаться они должны вместе. Часто оказывается, что при реальной работе программы ведут себя иначе, чем было задумано и как было при пошаговой отладке. Это случается из-за неправильной предварительной оценки скорости работы программ. Для правильной синхронизации необходимо продумать все варианты,

т. е. когда одна опережает другую или же они работают с одинаковой скоростью. Отсюда следует, что в большинстве случаев приходится отказаться от пошаговой отладки, а выполнять программы в реальном режиме, вставляя в интересующих местах вывод на экран каких-либо промежуточных или контрольных значений.

4. Результаты моделирования

Рассмотрим, какие результаты и исследования можно проводить с помощью описанного выше пакета программ. Основная программа включает в себя реализации восьми численных методов и имеет набор входных параметров для начала моделирования. Самые важные из них — это тип численного метода, шаг интегрирования, начальная температура, плотность, количество молекул и свойства одной молекулы m, ϵ, σ. Изменяя эти параметры и наблюдая за изменением результатов, можно получить полезные выводы.

Рассмотрим два эксперимента: изменение начальной температуры и интегрирование с различным временным шагом. В этих экспериментах физические свойства газа соответствуют аргону. Использован один и тот же численный метод — предиктор-корректор третьего порядка со схемой $\mathrm{P(EC)^2}$, который является оптимальным по точности и скорости работы. Из графиков энергии (рис. 3, *а*, 4) следует, что энергия системы сохраняется. Этот факт доказывает также устойчивость используемых численных методов. В первом эксперименте выполнены 6000 шагов интегрирования с шагом $h = 0.001$ для температур $T = 0.1$, $T = 1.0$ и $T = 7.0$ и 30 000 шагов интегрирования с шагом $h = 0.0002$ для $T = 50.0$ (табл. 5), так как скорость движения молекул в этом случае высока и решение требует большей точности. В этом эксперименте для бóльшей начальной температуры энергия устанавливается на бóльшем уровне, имеет более сильные колебания и равновесное состояние наступает быстрее из-за бóльшей скорости молекул. Для наибольшей начальной температуры значение энергии положительно (табл. 5). Это означает, что кинетическая энергия в этом случае превысила потенциальную.

Таблица 5

	$T = 0.1$	$T = 1.0$	$T = 7.0$	$T = 50.0$
h	0.001	0.001	0.001	0.0002
Количество шагов	6000	6000	6000	30000
Сохраняемый шаг	20	20	20	100
Время расчета, с	487.570	476.950	491.170	2372.510
		Полная энергия		
1	−4071.90257	−4053.90138	−3889.40000	2070.31647
50	−4293.18912	−3936.04583	−2015.87305	6527.63746
100	−4595.70529	−3953.51194	−2104.81649	6937.57784
150	−4745.31513	−3960.29729	−1971.99984	6145.94986
200	−4791.60437	−3922.43551	−2056.46154	5640.58216
250	−4818.11753	−3957.67546	−1955.91419	6498.22277
300	−4892.74061	−3910.97458	−1975.07618	7049.06991
Среднее значение	−4613.37439	−3932.19556	−2058.54450	6431.26775

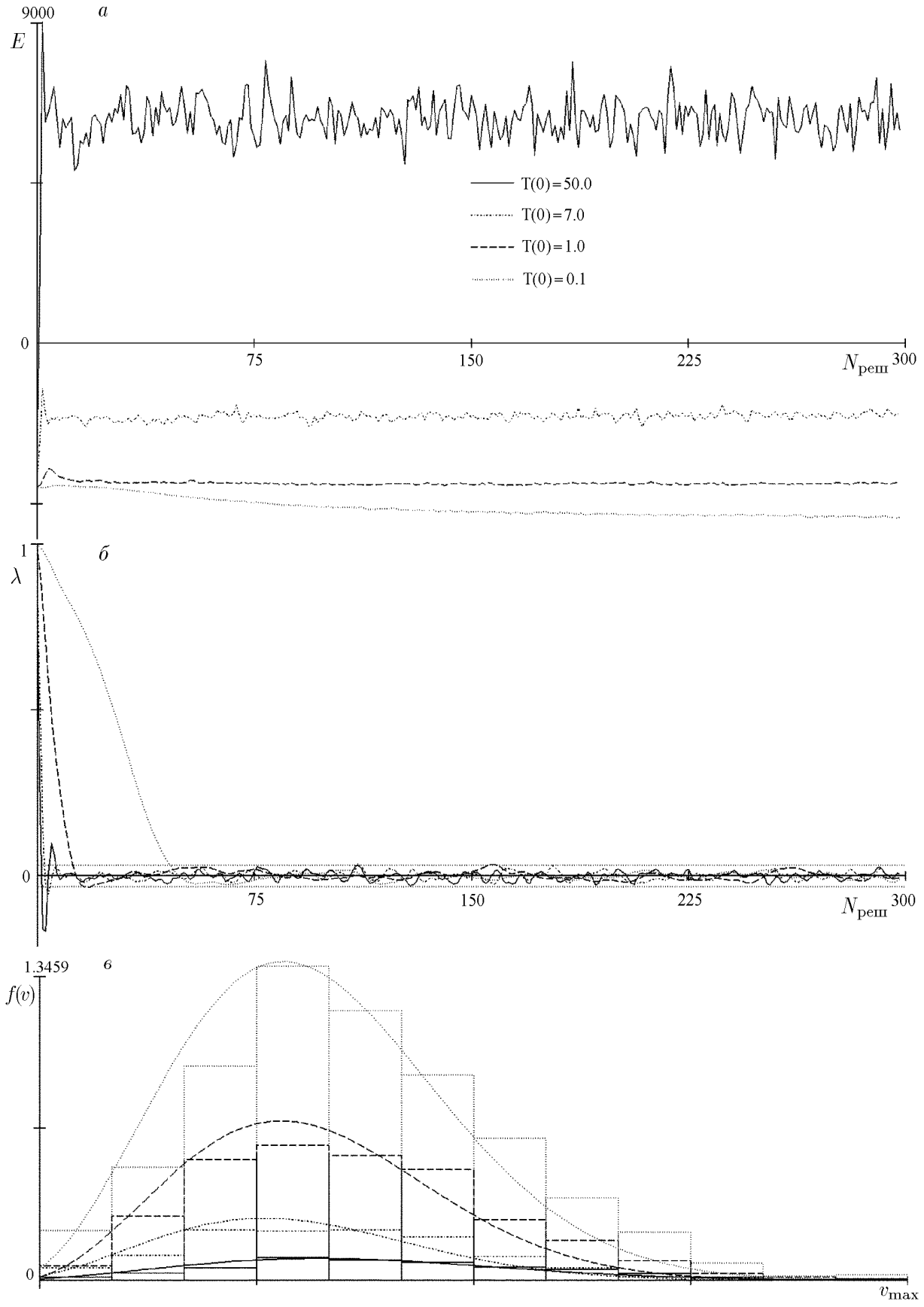

Рис. 3. Эксперимент с различной начальной температурой: полная энергия (*а*); установление и сохранение равновесного состояния (*б*); функция плотности распределения модуля скорости молекул (распределение Максвелла) (*в*); $N_{\text{реш}}$ — шаг решения.

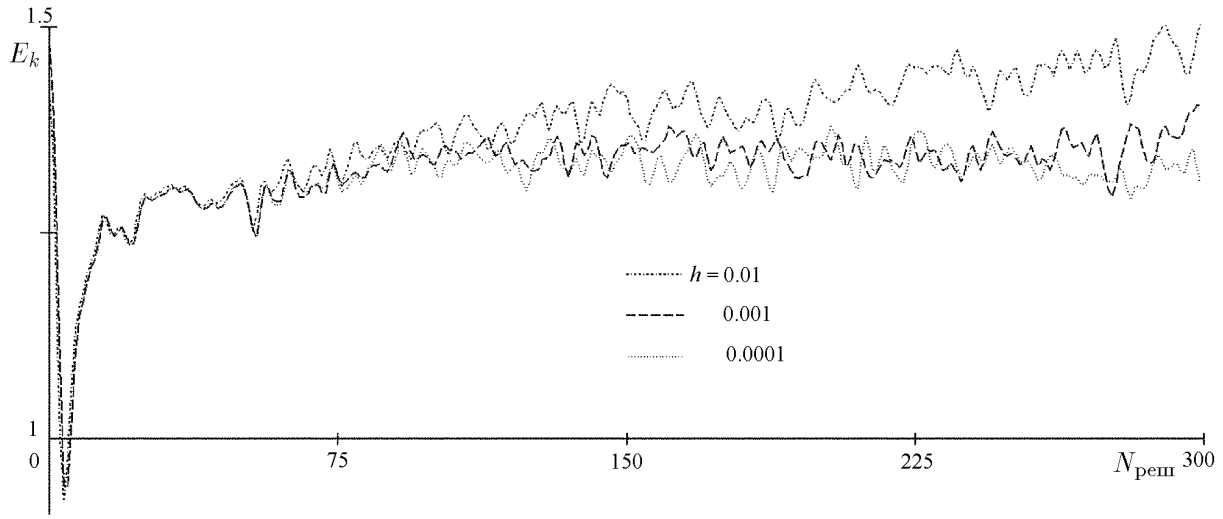

Рис. 4. Кинетическая энергия, рассчитанная при разных параметрах интегрирования.

Особое место при моделировании молекулярной динамики занимает наступление равновесного состояния в системе. До этого момента все физические характеристики системы имеют большие колебания и не являются достоверными результатами моделирования. Поэтому очень важно проследить момент наступления равновесного состояния, после которого можно на основе полученных результатов делать какие-то выводы. Для наблюдения перехода от начального состояния к равновесному Верлет ввел параметр упорядочивания молекул λ, который определяется соотношениями

$$\lambda = \frac{1}{3}[\lambda_x + \lambda_y + \lambda_z], \quad \lambda_x = \frac{1}{N}\sum_{i=1}^{N} \cos\left(\frac{4\pi x_i}{a}\right),$$

где a — длина одной ячейки равномерной сетки, в которой находятся молекулы в начальный момент времени, а значения λ_y и λ_z вычисляются аналогично λ_x по соответствующей оси координат. В начальный момент времени $\lambda = 1$, так как все координаты молекул делятся нацело на $\dfrac{a}{2}$. Когда начальная равномерная сетка полностью разрушена, координаты молекул являются случайными числами, поэтому график параметра λ колеблется около нуля в интервале $\pm\dfrac{\sqrt{N}}{N}$ (см. рис. 3, *б*). На примере такого графика можно убедиться, что момент перехода в равновесное состояние для более высокой начальной температуры наступает быстрее и колебания не превышают теоретически определенного интервала. После перехода в равновесное состояние модули скоростей молекул подчиняются распределению Максвелла [2, 8]:

$$f(v) = \frac{4v^2}{\sqrt{\pi}}\left(\frac{m}{2kT}\right)^{3/2} \exp\left(-\frac{mv^2}{2kT}\right).$$

Разработанная программа также позволяет проследить установление этого распределения вместе с установлением равновесного состояния. Для каждого сохраняемого решения на одном графике строятся гистограмма распределения модулей скоростей молекул, а также теоретическая кривая с помощью вычисленного значения температуры T. На рис. 3, *в* представлено распределение модулей скоростей молекул для различных начальных температур на 1000-м шаге интегрирования, которое действительно соответствует распределению Максвелла с различными параметрами.

Во втором эксперименте для значений параметра интегрирования $h = 0.01$, $h = 0.001$ и $h = 0.0001$ выполнено соответственно 600, 6000 и 60 000 шагов. Из них сохранены каждые 2-й, 20-й и 200-й шаги, поэтому они представляют собой результаты в один и тот же момент времени и их можно сравнить. Наиболее показательным оказался график кинетической энергии (см. рис. 4), из которого ясно, что для наибольшего шага $h = 0.01$ кинетическая энергия растет, следовательно, такое значение нельзя использовать для интегрирования при таких физических параметрах.

Различные комбинации входных параметров позволяют также проводить множество других экспериментов. Наиболее интересными из них являются подбор параметров и сравнение численных методов при одинаковых физических свойствах газа, изменение плотности газа и количества молекул, а также контролирование изменения вычислительного времени в зависимости от этих параметров.

Заключение

Разработан программный пакет, реализующий современные численные методы для моделирования процессов молекулярной динамики газа. Скорость вычислений увеличена за счет разделения всех проводимых расчетов на параллельные вычисления. Для достижения наилучшего результата на практике исследовано быстродействие отдельных частей алгоритма. Все изложенные здесь и включенные в программный пакет общие идеи распараллеливания алгоритмов реализованы и протестированы на безошибочную работу на одно- и двухпроцессорном компьютере в операционных системах RedHat Linux 6.2 и 7.0, они могут быть использованы при моделировании других физических процессов и численном решении как систем, так и отдельных дифференциальных уравнений.

Список литературы

[1] Haile J. M. Molecular Dynamics Simulation. Elementary methods. N. Y.: John Wiley & Sons, 1992.

[2] Rapaport. The Art of Molecular Dynamics Simulation. Cambridge: Cambridge Univ. Press, 1995.

[3] Dormand J. R. Numerical Methods of Differential Equations. Boca Raton: CRC Press, 1996.

[4] Lapidus L., Seinfeld J. H. Numerical Solution of Ordinary Differential Equations. N. Y.: Acad. Press, 1971.

[5] Davignon B. UNIX C-Programmierung. München: Te-wi Verlag GmbH, 1992.

[6] Deuflhard P., Bornemann F. Numerische Mathematik. P. II. Integration gewöhnlicher Differentialgleichungen. Berlin: Walter de Gruyter, 1994.

[7] Forest E. Ruth R. D. Fourth-order symplectic integration // Phys. D. 1990. Vol. 43. P. 105–117.

[8] Huang K. Statistical Mechanics. N. Y.: John Wiley & Sons, 1987.

[9] Lambert J. D. Numerical Methods for Ordinary Differential Systems. The Initial Value Problems. N. Y.: John Wiley & Sons, 1997.

[10] Чан Т. Системное программирование на C++ для UNIX. Киев: BHV, 1999.

Поступила в редакцию 14 февраля 2001 г.