Санкт-Петербургский государственный университет
кафедра вычислительной физики

С.А.Немнюгин

Лабораторная работа 3.1
Оптимизация и распараллеливание вычислительной программы на примере моделирования системы взаимодействующих частиц методом молекулярной динамики

Методические материалы к курсу «Средства программирования для многопроцессорных вычислительных систем»

Intel Multicore Curriculum Initiative

Санкт-Петербург
2007

Метод молекулярной динамики

Метод молекулярной динамики является одним из основных методов моделирования динамики систем, состоящих из взаимодействующих между собой частиц. Он применяется в молекулярной физике, астрофизике и других науках. Интерпретация понятия «частица» зависит от предметной области. В молекулярной физике это могут быть атом или молекула, а в астрофизике «частицами» являются звезды или планеты, взаимодействующие с другими астрономическими объектами посредством гравитационного взаимодействия. Динамика систем таких частиц часто описывается уравнениями классической механики.

Будем рассматривать систему N частиц, динамика которых описывается уравнением второго закона Ньютона:

$$\mathbf{F}_i = m_i \mathbf{a}_i$$

Здесь $\mathbf{F}_i$ - равнодействующая всех сил, действующих на i-ю частицу, m_i - ее масса и $\mathbf{a}_i$ - ускорение, с которым частица двигается под действием силы. Это обыкновенное дифференциальное уравнение второго порядка, которое можно записать в виде:

$$\mathbf{F}_i = m_i \frac{d^2 \mathbf{r}_i}{dt^2}$$

где $\mathbf{r}_i$ - радиус-вектор i-й частицы.

Равнодействующая сил, является суммой парных взаимодействий i-й частицы со всеми остальными частицами:

$$\mathbf{F}_i \equiv \mathbf{F}(\mathbf{r}_i) = \sum_{i \neq j=1}^{N} \mathbf{F}(\mathbf{r}_i, \mathbf{r}_j)$$

или

$$\mathbf{F}_i = \sum_{j=1}^{i-1} \mathbf{F}(\mathbf{r}_i, \mathbf{r}_j) + \sum_{j=i+1}^{N} \mathbf{F}(\mathbf{r}_i, \mathbf{r}_j)$$

Сила взаимодействия связана с потенциалом взаимодействия. В расчетах молекулярных систем часто используют потенциал Леннарда-Джонса:

$$V(r) = 4\varepsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^{6} \right]$$

Здесь $r = |\mathbf{r}|$, а ε и σ - параметры потенциала. Первый определяет интенсивность взаимодействия, а второй – положение минимума (см. рис. 1).

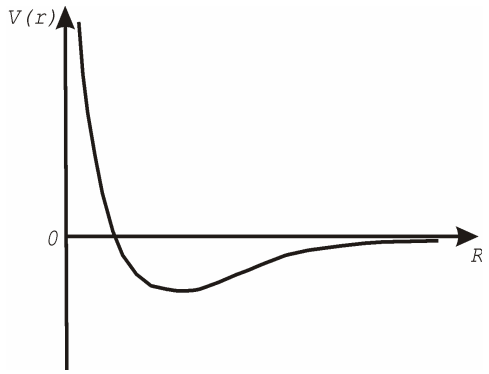

Рис. 1. Потенциал Леннарда-Джонса

Потенциал Леннарда-Джонса хорошо описывает взаимодействие между атомами Аргона. Для аргона $m \approx 6.69 \times 10^{-23} кг$, $\sigma \approx 3.405 \times 10^{-10} м$, $\varepsilon \approx 1.654 \times 10^{-21} Дж$. Будем считать, что массы всех частиц одинаковы, система единиц выбрана таким образом, что $m = 1$, $\varepsilon = 1$ и $\sigma = 1$. Тогда сила парного взаимодействия:

$$\mathbf{F}(\mathbf{r}_i, \mathbf{r}_j) = \frac{24\left(\mathbf{r}_i - \mathbf{r}_j\right)}{\left|\mathbf{r}_i - \mathbf{r}_j\right|^2}\left[\frac{2}{\left|\mathbf{r}_i - \mathbf{r}_j\right|^{12}} - \frac{1}{\left|\mathbf{r}_i - \mathbf{r}_j\right|^6}\right]$$

Интерес могут представлять траектории частиц или термодинамические характеристики системы, например, зависимость температуры от времени:

$$T(t) = \frac{1}{3Nk_B}\sum_{i=1}^{N}\left|\frac{d\mathbf{r}_i(t)}{dt}\right|^2$$

где $k_B \approx 1.38 \times 10^{-23} \frac{Дж}{К}$ - постоянная Больцмана.

Для вычисления траекторий в методе молекулярной динамики используются различные алгоритмы. Они основаны на замене непрерывного времени дискретным набором значений $t \to t^k = t^0 + k\Delta t$, $k = 1, 2, \ldots$. Одним из алгоритмов является *метод Верле*:

$$\mathbf{r}_i^{k+1} = 2\mathbf{r}_i^k - \mathbf{r}_i^{k-1} + \mathbf{F}(\mathbf{r}_i^k)\frac{\Delta t^2}{m_i}$$

Здесь $\mathbf{r}_i^k = \mathbf{r}_i(t^k)$. Наиболее трудоемким является вычисление силы, действующей на частицу.

Для оптимизации вычисления сил иногда вводится *радиус обрезания* – некоторый фиксированный (достаточно большой) радиус, такой, что пересчет сил, действующих между частицами, оказавшимися внутри этого радиуса, производится на каждом шаге. Пересчет сил, действующих со стороны более удаленных частиц, производится через несколько шагов. Такая стратегия позволяет уменьшить количество трудоемких операций расчета парных сил. Номера «ближайших» частиц могут храниться в специальном массиве. Такой массив называется *списком*. Обычно используются два массива целого типа. Каждый элемент первого массива (`list_nb`) содержит количество ближайших соседей соответствующей частицы:

```
k0 = 0
j0 = 0
do i = 1, n
 do j =1, n
 if(abs(x(i) - x(j)) < rcutoff) then
 j0 = j0 + 1
 list_nb(i) = j
 end if
 end do
 i_nb(i) = j0 - k0
end do
```

Во втором массиве содержатся номера соседей каждой частицы. Положения хранятся в элементах, начиная с `start(i)` и заканчивая `finish(i)`:

```
start(1) = 1
finish(1) = i_nb(1)
do i = 2, n
 start(i) = finish(i - 1) + 1
 finish(i) = finish(i - 1) + i_nb(i)
end do
```

Лабораторная работа

В заданиях лабораторной работы 3.1 предлагается выполнить оптимизацию вычислительной программы. Первый этап оптимизации – сокращение числа операций в последовательном коде на основе использования физических законов. Затем выполняется распараллеливание программы. Выбор инструмента распараллеливания должен быть аргументирован.

Данная работа не предполагает выполнения всех заданий. Учащемуся может быть предложен набор из нескольких заданий, обязательно включающий номера 1, 2 и 3, а также одно или несколько заданий, связанных с распараллеливанием.

По результатам составляется отчет, содержащий решения с обоснованием выбранного метода, результаты измерения быстродействия программы, а также физические результаты и выводы.

Выполнение данной работы рассчитано на несколько занятий (2-6 академических часов). Имеется полный исходный текст программы на языке Fortran 90.

Задания для практической работы

Задание 1

Получить у преподавателя файл с исходным текстом программы и ознакомиться с реализацией метода молекулярной динамики.

Задание 2

Откомпилировать программу, выполнить моделирование для системы с параметрами, заданными преподавателем. Определить процессорное время, потраченное на выполнение моделирования.

Задание 3

Выполнить оптимизацию последовательного кода программы на основе «физических соображений». Определить процессорное время, потраченное на выполнение моделирования. Сравнить с результатом, полученным в задании 2.

Задание 4

Проанализировать последовательный код и выявить участки потенциального параллелизма. Выполнить распараллеливание с помощью OpenMP (если это возможно). Определить процессорное время, потраченное на выполнение моделирования. Сравнить с результатом, полученным в задании 3.

Задание 5

Проанализировать последовательный код, выявить участки потенциального параллелизма и предложить способ распараллеливания в модели передачи сообщений. Выполнить распараллеливание с помощью MPI, используя блокирующие двухточечные операции обмена (если это возможно). Определить процессорное время, потраченное на выполнение моделирования. Сравнить с результатами, полученными в заданиях 3 и 4.

Задание 6

Проанализировать возможность использования других видов обмена (двухточечные неблокирующие, коллективные). Если предполагается выигрыш в производительности, модифицировать вариант программы с использованием MPI. Определить процессорное время, потраченное на выполнение моделирования. Сравнить с результатами, полученными в заданиях 3 , 4 и 5.

Задание 7

Выполнить оптимизацию последовательного кода, полученного при выполнении задания 3, используя введение радиуса обрезания и метод генерации списков частиц. Определить процессорное время, потраченное на выполнение моделирования. Сравнить с результатом, полученным в задании 3.

Задание 8

Провести анализ зависимостей при генерации списков частиц. Выполнить распараллеливание, выбрав наиболее подходящий инструмент. Определить процессорное время, потраченное на выполнение моделирования. Сравнить с результатом, полученным в задании 7.

Задание 9

Выполнить распараллеливание вычисления сил, действующих на частицы, используя метод списков. Определить процессорное время, потраченное на выполнение моделирования. Сравнить с результатом, полученным в заданиях 4 или 5.

Задание 10

Исследовать степень сбалансированности загрузки процессоров (ядер при использовании многоядерной архитектуры). Если сбалансированность неудовлетворительная, предложить способы ее улучшения.

Задание 11

Исследовать масштабируемость программы. Если масштабируемость неудовлетворительная, предложить способы ее улучшения.

Задание 12

На основании результатов, полученных при выполнении заданий данной лабораторной работы, написать отчет, в котором содержатся выводы об эффективности различных способов оптимизации исходного последовательного кода и трудоемкости реализации этих способов на практике.

Пример 1

Далее приводится листинг программы молекулярной динамики.

```
program mol_dyn

implicit real(8) (a-h, o-z)
integer, parameter :: ndim = 1000
!
! Массивы координат, проекций скоростей и ускорений частиц
!
real(8), dimension(1:ndim) :: x, y, vx, vy, ax, ay
!
! Задание начальной конфигурации частиц
!
call start(x, y, vx, vy, N, Sx, Sy, dt, dt2, nsnap, ntime)
!
! Вычисление ускорений частиц
!
call accel(x, y, ax, ay, N, Sx, Sy, zpe)

zpe =0.0d0
!
! Энергия выводится через nsnap шагов
!
do isnap =  1, nsnap
!
! Внутренний цикл - по шагам по времени
!
 do itime = 1, ntime
!
! Выполняется сдвиг частиц
!
  call move(x, y, vx, vy, ax, ay, N, Sx, Sy, dt, dt2, zke, zpe)

 end do
!
! Вывод значений энергии: кинетической, потенциальной и полной
!
 call output(zke, zpe, Sx, Sy, dt, N, ntime)

end do

end program
!
!=================================================
!
subroutine start(x, y, vx, VY, N, Sx, Sy, dt, dt2, nsnap, ntime)

implicit real(8) (a-h, o-z)
integer, parameter :: ndim = 1000
real(8), dimension(1:ndim) :: x, y, vx, vy, ax, ay
```

```
!
! Число частиц
!
n = 200
!
! Размер области моделирования
!
sx = 100.0d0
sy = 100.0d0
!
! Шаг по времени
!
dt = 1.0d-11
dt2 = dt**2
!
! Максимальное значение скорости
!
vmax = 10.0d0
!
! Число строк в таблице вывода и количество шагов по времени
между ними
!
nsnap = 200
ntime = 500
!
! Формируется начальная хаотическая конфигурация
!
do i = 1, N
 call rndm(ranx)
 x(i) = sx * ranx
 call rndm(rany)
 y(i) = sy * rany
 call rndm(ranx)
 call rndm(rany)
 vx(i) = vmax * (2 * ranx - 1)
 vy(i) = vmax * (2 * rany - 1)
end do

do i = 1, N
 vxcum = vxcum + vx(i)
 vycum = vycum + vy(i)
end do

vxcum = vxcum / N
vycum = vycum / N

do i = 1, N
 vx(i) = vx(i) - vxcum
 vy(i) = vy(i) - vycum
end do

end subroutine
!
```

```
!=====================================================
!
subroutine move(x, y, vx, vy, ax, ay, N, Sx, Sy, dt, dt2, zke,
zpe)

implicit real(8) (a-h, o-z)
integer, parameter :: ndim = 1000
real(8), dimension(1:ndim) :: x, y, vx, vy, ax, ay

do i = 1, N
 xnew = x(i) + vx(i) * dt + 0.5d0 * ax(i) * dt2
 ynew = y(i) + vy(i) * dt + 0.5d0 * ay(i) * dt2
!
! Учет периодических граничных условий
!
 call cellp(xnew, ynew, vx(i), vy(i), sx, sy)
 x(i) = xnew
 y(i) = ynew
 vx(i) = vx(i) + 0.5d0 * ax(i) * dt
 vy(i) = vy(i) + 0.5d0 * ay(i) * dt
end do

call accel(x, y, ax, ay, N, Sx, Sy, zpe)

do i = 1, n
 vx(i) = vx(i) + 0.5d0 * dt * ax(i)
 vy(i) = vy(i) + 0.5d0 * dt * ay(i)
 zke = zke + vx(i)**2 + vy(i)**2
end do

end subroutine
!
!=====================================================
!
subroutine accel(x, Y, ax, ay, N, sx, sy, zpe)

implicit real(8) (a-h, o-z)
integer, parameter :: ndim = 1000
real(8), dimension(1:ndim) :: x, y, ax, ay

do i = 1, n
 ax(i) = 0.0d0
 ay(i) = 0.0d0
end do

do i = 1, n
 do j = 1, n
  if(i /= j) then
  dx = x(i) - x(j)
  dY = y(i) - y(j)

  if(dabs(dx) > 0.5 * sx) dx = dx -  sign(sx, dx)
  if(dabs(dy) > 0.5 * sy) dy = dy -  sign(sy, dy)
```

```
!
! Вычисление силы, действующей на j-ю частицу
!
  r = dsqrt(dx**2 + dy**2)
  ri = 1.0d0 / r
  ri6 =  ri**6
  g = 24.0d0 * ri6 * ri * (2.0d0 * ri6 -  1)
  force = 0.5d0 * g * ri
  pot = 4.0d0 * ri6 * (ri6**2 - 1.0d0)

  ax(i) = ax(i) +  force * dx
  ay(i) = ay(i) +  force * dy
  ax(j) = ax(j) -  force * dx
  ay(j) = ay(j) -  force * dy
  zpe = zpe + pot
  end if

  end do
end do

end subroutine
!
!=====================================================
!
subroutine cellp(xnew, ynew, vx, vy, Sx, Sy)

implicit real(8) (a-h, o-z)
if(xnew < 0)  xnew = xnew + sx
if(xnew > sx) xnew = xnew - sx
if(ynew < 0)  ynew = ynew + sy
if(ynew > sy) ynew = ynew - sy

end subroutine
!
!=====================================================
!
subroutine output(zke, zpe, sx, sy, dt, n, ntime)

implicit real(8) (a-h, o-z)
data iff/0/

if(iff == 0) then
iff = 1
write(6,*) '      ke            pe            tot'
end if

zke = 0.5d0 * zke / ntime
zpe = zpe / ntime
tot = zke + zpe

write(6, "(6(1x, e13.6))") zke, zpe, tot

zke = 0.0d0
```

```
zpe = 0.0d0

end subroutine
!
! Генератор псевдослучайных чисел
!
subroutine rndm(ran)

implicit real*8(a-h,o-z)
data a31/2147483647.d0/, mul/16807.d0/, x/268435451.d0/
t = dmod(x * mul, a31)
ra = t
x = t
ran = ra / a31

end subroutine
```

Приложение. Рекомендации по выполнению работы

Задание 1

Исходный текст последовательного варианта программы снабжен комментариями.

Задание 3

Применение третьего закона Ньютона:

$$\mathbf{F}(\mathbf{r}_i, \mathbf{r}_j) = -\mathbf{F}(\mathbf{r}_j, \mathbf{r}_i)$$

позволяет сократить объем вычислений в цикле примерно в 2 раза.
Следующий прием основан на введении «радиуса обрезания» R_{cut} и разбиении всех частиц на две группы по отношению к данной:

1. частицы, для которых $\left|\mathbf{r}_i - \mathbf{r}_j\right| \le R_{cut}$;
2. частицы, для которых $\left|\mathbf{r}_i - \mathbf{r}_j\right| > R_{cut}$.

R_{cut} выбирается таким образом, чтобы частицы из первой группы давали основной вклад во взаимодействие. Выбор обычно выполняется эмпирически, то есть, на основе пробных расчетов.

При вычислении силы сумма делится на две подсуммы по группам частиц:

$$\mathbf{F}_i = \sum_{|\mathbf{r}_i - \mathbf{r}_j| \le R_{cut}} \mathbf{F}(\mathbf{r}_i, \mathbf{r}_j) + \sum_{|\mathbf{r}_i - \mathbf{r}_j| > R_{cut}} \mathbf{F}(\mathbf{r}_i, \mathbf{r}_j)$$

Если вклад от удаленных частиц пересчитывать реже, чем от частиц, для которых $\left|\mathbf{r}_i - \mathbf{r}_j\right| \le R_{cut}$, трудоемкость вычисления силы уменьшится.