Введение в параллельное программирование и высокопроизводительные вычисления

С.А.Немнюгин

Различные аспекты создания эффективных приложений

(конспект)

Санкт-Петербург
2008

Различные аспекты создания эффективных приложений

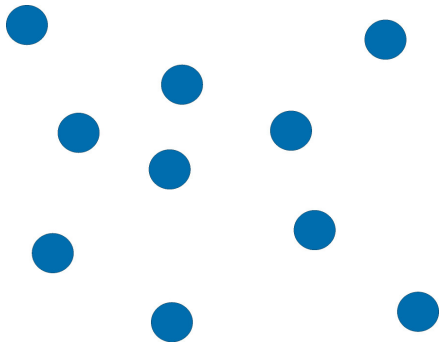
- Модель
- Численный метод
- Алгоритм
- Реализация (выбор языка, использование оптимизированных библиотек, применение приемов написания оптимального кода)
- Оптимизация на этапе компиляции
- Параллельное программирование
- «Тонкая» настройка параллельной программы

Модель

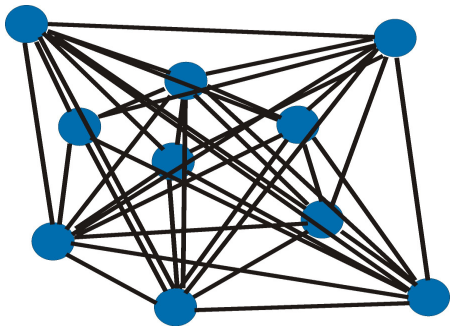
Пример

Моделирование многочастичной системы методом молекулярной динамики

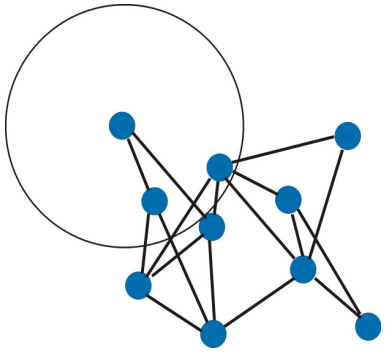
Система частиц



Частицы взаимодействуют:



Введение обрезания взаимодействия приводит к уменьшению трудоемкости:



Численный метод

Пример

Решение СЛАУ с разреженной матрицей.
Метод Гаусса или метод прогонки?

Алгоритм

Пример

Умножение матрицы на матрицу

- Прямой метод. Трудоемкость $O(n3)$
- Метод Штрассена. Трудоемкость $O(n2.81)$
- Метод Копперсмита-Винограда. Трудоемкость $O(n2.38)$

Реализация

выбор языка программирования

Пример

C, C++, Fortran, Java, Pascal, Ruby…?

Реализация

использование оптимизированных библиотек

Пример

Матричные вычисления:

- **BLAS** (Basic Linear Algebra Subroutines) Levels 1-3;
- **LAPACK** (Linear Algebra Package) и ее предшественница LINPACK, ScaLAPACK;
- **Visual Numerics Inc. ® IMSL** (International Mathematics and Statistics Library) ;
- **Intel ® MKL** (Math Kernel Library).

BLAS

1) **уровень 1**: основные операции с векторами;
2) **уровень 2**: основные матрично-векторные операции;
3) **уровень 3**: основные матрично-матричные операции.

Библиотека оптимизируется с учетом архитектуры.

Intel ® MKL

Состав библиотеки:

- **BLAS** (3 уровня + расширение – уровень 1 для разреженных векторов)
- **LAPACK** – вычислительная алгебра, в том числе решение спектральных задач
- **DFT** (дискретное преобразование Фурье) – в том числе многомерное. Многопоточная реализация
- **Vector Mathematical Library** – математические функции
- **Vector Statistical Library** – набор векторизованных генераторов случайных чисел

Оптимизирована для архитектуры Intel ®.
Имеет смысл использовать только для решения больших задач.

Для решения «маленьких» задач можно использовать IPP (Intel ® Performance Primitives)

Intel ® Integrated Performance Primitives

Библиотека готовых компонентов для разработки мультимедийных приложений для вычислительных платформ Intel. Включает в себя модули для обработки сигналов и выполнения векторных и матричных операций, функции сжатия и распаковки речи и статических/динамических изображений, а также средства шифрования и обработки аудиоданных и текстовых строк.
Библиотека Intel Integrated Performance Primitives (Intel IPP) представляет собой универсальные низкоуровневые API-интерфейсы.
Intel IPP обеспечивает прозрачное использование расширенных возможностей процессоров Intel, таких, как технология MMX, наборы команд Streaming SIMD Extensions и Streaming SIMD Extensions 2, а также возможностей микроархитектуры Intel XScale. Библиотека Intel IPP оптимизирована для работы с самыми разными микропроцессорами компании Intel, включая Intel Pentium 4, Intel Itanium 2, Intel Xeon, а также процессорами для карманных устройств Intel PCA с архитектурой XScale.
Библиотека Intel IPP поддерживает 32- и 64-битные операционные системы Windows и Linux, включая встраиваемые версии, такие как Windows Mobile.

LAPACK

Библиотеки подпрограмм LAPACK и ScaLAPACK поддерживает работу с:
- заполненными прямоугольными или квадратными матрицами;
- ленточными матрицами (узкими);
- заполненными матрицами, хранящимися на внешнем носителе.

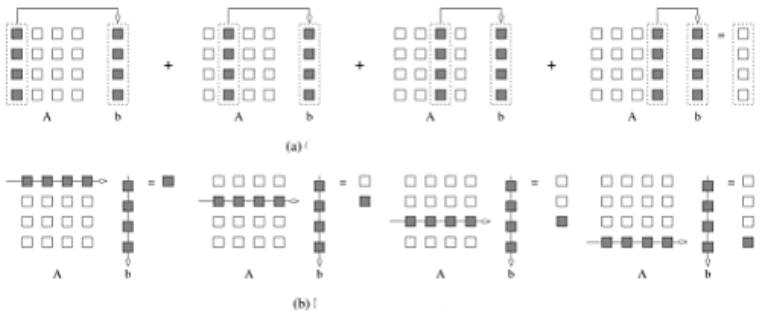
Visual Numerics ® IMSL

Состав:
- математические подпрограммы;
- статистика;
- специальные функции.

Реализация алгоритма

Применение приемов написания оптимального кода

Пример



Время выполнения различается. Причиной является разное количество «промахов кэша».

Алгоритм

Умножение матриц
«Классический алгоритм»

$$C = AB$$

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Трудоемкость $O(n^3)$

Метод Штрассена

Решаемая задача – вычисление произведения матриц A × B при больших A и B. Предполагаем, что обе матрицы – квадратные, размера n × n.
Предположим, что n – степень двойки (в противном случае матрицу можно дополнить строками и столбцами, содержащими единицу на главной диагонали и нули на остальных позициях). Тогда можно воспользоваться следующим рекурсивным алгоритмом:

$$A \times B = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & g \\ f & h \end{pmatrix} = \begin{pmatrix} ae+bf & ag+bh \\ ce+df & cg+dh \end{pmatrix} = \begin{pmatrix} r & s \\ t & u \end{pmatrix}$$

Эффективность (число умножений) $T(n)$ этого алгоритма можно вычислить: $T(1) = 1$; $T(n) = 8T(n/2)$, откуда получаем $T(n) = n^3 = n^{\log 8}$.

Формулы:
$P1 = A1 \times B1$; $P2 = A2 \times B2$; $P3 = A3 \times B3$; $P4 = A4 \times B4$;
$P5 = A5 \times B5$; $P6 = A6 \times B6$; $P7 = A7 \times B7$; $P8 = A8 \times B8$;
$r = P1 + P2$; $s = P3 + P4$; $t = P5 + P6$; $u = P7 + P8$;
Всего 8 матричных умножений и 4 матричных сложения.
Алгоритм Штрассена позволяет уменьшить асимптотическую оценку до $T(n) = n^{\log 7}$ с помощью уменьшения количества необходимых промежуточных матриц Pi.

i	Ai	Bi	$Pi = Ai \times Bi$
1	A	$g - h$	$ag - ah$
2	$a + b$	h	$ah + bh$
3	$c + d$	e	$ce + de$
4	D	$f - e$	$df - de$
5	$a + d$	$e + h$	$ae + ah + de + dh$
6	$b - d$	$f + h$	$bf + bh - df - dh$
7	$a - c$	$e + g$	$ae + ag - ce - cg$

$r = P4 - P2 + P5 + P6 = ae + bf$
$s = P1 + P2 = ag + bh$
$t = P3 + P4 = ce + df$
$u = P1 + P5 - P3 - P7 = cg + dh$

Итого 7 матричных умножений и 18 сложений

На практике применять алгоритм Штрассена следует для перемножения больших плотных (не разреженных) матриц размерности не меньше 64. Для матриц небольшого размера лучше использовать классический алгоритм умножения.

Алгоритм Копперсмита-Винограда

Трудоемкость $O(n^{2.38})$. Особенности данного алгоритма – повышенная трудоемкость программирования, константа больше, чем в алгоритме Штрассена.

Оптимизация при компиляции

Не существует универсальной технологии оптимизации, одинаково пригодной для всех приложений.
Общая оптимизация – достаточно надежные и простые технологии оптимизации, дающие эффект для большинства программ. Замена встроенных функций их кодом, простая оптимизация циклов
Использование агрессивных методов оптимизации часто позволяет выявить скрытые ошибки кода.
Возможно снижение точности при оптимизации.
Оптимизация, специфичная для процессора – ухудшает переносимость программы
Межпроцедурная оптимизация (IPO) – с учетом подпрограмм, подставляемые функции и т. д.
Оптимизация по профилю выполнения (PGO) – используется промежуточный «инструментальный» файл

Пример
Ключи оптимизации (по времени выполнения и размеру) gcc 4.3.0

- **-O0** отключить оптимизацию;
- **-O1**, **-O2**, **-O3** ключи оптимизации в порядке возрастания эффективности;
- **-Os** оптимизация по размеру исполняемого кода.

Выравнивание по границе слова:

- при выравнивании выборка занимает 1 цикл
- без выравнивания 2 цикла

Пример

Компилятор Интел, избранные ключи оптимизации:

- **-O0** оптимизация отключена;
- **-O1**, **-O2**, **-O3** общая оптимизация;
- **-ip**, **-ipo** межпроцедурная оптимизация;
- **-mcpu**, **-march** оптимизация, специфическая для процессора;
- **-prof-use** оптимизация на основе профилирования.

Распараллеливание программы

- Разработка параллельного алгоритма.
- Выбор инструмента распараллеливания.
- Отладка и верификация параллельной программы.

Тонкая настройка параллельной программы

Выбор оптимальных значений параметров процедур и/или директив распараллеливания (например, способ распределения итераций между потоками или величина «порции» в директиве OpenMP распараллеливания цикла).