Лабораторная работа №4

Шулер М.А 1-МД-20

**Matrix Multiplication**

**uses** Arrays;

**procedure** ParallelMult(a,b,c: **array** [,] **of** real; n: integer);

**begin**

{$omp parallel for } **for var** i:=0 **to** n-1 **do for var** j:=0 **to** n-1 **do begin**

c[i,j]:=0;

**for var** l:=0 **to** n-1 **do**

c[i,j]:=c[i,j]+a[i,l]\*b[l,j];

**end**; **end**;

**procedure** Mult(a,b,c: **array** [,] **of** real; n: integer);

**begin**

**for var** i:=0 **to** n-1 **do for var** j:=0 **to** n-1 **do begin**

c[i,j]:=0;

**for var** l:=0 **to** n-1 **do**

c[i,j]:=c[i,j]+a[i,l]\*b[l,j];

**end**; **end**;

**const** n =250;

**begin**

**var** a := Arrays.CreateRandomRealMatrix(n,n); **var** b := Arrays.CreateRandomRealMatrix(n,n); **var** c := **new** real[n,n];

ParallelMult(a,b,c,n);

writeln('Параллельное перемножение матриц: ',Milliseconds,' миллисекунд');

**var** d := Milliseconds; Mult(a,b,c,n);

writeln('Последовательное перемножение матриц: ',Milliseconds-d,' миллисекунд');

**end**.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Программа** | **Используемые ядра ЦП** | **Потоки программы** | **Размер массива** | **Время последовательное (миллисекунд) для 1 ядра** | **Время параллельное (миллисекунд)** | **V\_real** | **S** | **S\_avg** | **V\_A** |
| **matrix multiplication** | **1 ядро (2 потока)** | **13 потока** | **100** | **10** | **35** | 0,285714286 | - | 0,456449052 | - |
| **matrix multiplication** | **2 ядро (4 потока)** | **20 потока** | **100** | **10** | **20** | 0,5 | 3 | 0,456449052 | 1,373202857 |
| **matrix multiplication** | **4 ядро (8 потока)** | **21 потока** | **100** | **10** | **17** | 0,588235294 | 1,933333333 | 0,456449052 | 1,688228755 |
| **matrix multiplication** | **8 ядро (16 потока)** | **21 потока** | **100** | **10** | **16** | 0,625 | 1,685714286 | 0,456449052 | 1,906967011 |
| matrix multiplication | 1 ядро (2 потока) | 20 потока | 150 | 33 | 72 | 0,458333333 | - | 0,456449052 | - |
| matrix multiplication | 2 ядро (4 потока) | 21 потока | 150 | 33 | 36 | 0,916666667 | 1,181818182 | 0,456449052 | 1,373202857 |
| matrix multiplication | 4 ядро (8 потока) | 21 потока | 150 | 33 | 25 | 1,32 | 0,676767677 | 0,456449052 | 1,688228755 |
| matrix multiplication | 8 ядро (16 потока) | 21 потока | 150 | 33 | 21 | 1,571428571 | 0,584415584 | 0,456449052 | 1,906967011 |
| matrix multiplication | 1 ядро (2 потока) | 23 потока | 200 | 78 | 123 | 0,634146341 | - | 0,456449052 | - |
| matrix multiplication | 2 ядро (4 потока) | 19 потока | 200 | 78 | 71 | 1,098591549 | 0,820512821 | 0,456449052 | 1,373202857 |
| matrix multiplication | 4 ядро (8 потока) | 21 потока | 200 | 78 | 42 | 1,857142857 | 0,384615385 | 0,456449052 | 1,688228755 |
| matrix multiplication | 8 ядро (16 потока) | 22 потока | 200 | 78 | 30 | 2,6 | 0,296703297 | 0,456449052 | 1,906967011 |
| matrix multiplication | 1 ядро (2 потока) | 24 потока | 250 | 153 | 203 | 0,753694581 | - | 0,456449052 | - |
| matrix multiplication | 2 ядро (4 потока) | 20 потока | 250 | 153 | 113 | 1,353982301 | 0,477124183 | 0,456449052 | 1,373202857 |
| matrix multiplication | 4 ядро (8 потока) | 21 потока | 250 | 153 | 65 | 2,353846154 | 0,233115468 | 0,456449052 | 1,688228755 |
| matrix multiplication | 8 ядро (16 потока) | 22 потока | 250 | 153 | 43 | 3,558139535 | 0,178338002 | 0,456449052 | 1,906967011 |

**Формулы:**

**V\_real** =T\_ **Время последовательное** / T\_ **Время параллельное**

S = (1/V\_real - 1/p) / (1 - 1/p)

V\_A = 1 / (S + (1-S)/p)

**Сравнение реального выигрыша в производительности с теоретическим по закону Амдала**

**Вывод:** Закон Амдала адекватно описывает поведение параллельной программы для базовых случаев, но в реальных условиях могут наблюдаться значительные отклонения, связанные с архитектурными особенностями современных процессоров и оптимизациями на уровне компилятора. Для алгоритма умножения матриц параллелизация эффективна только для достаточно крупных задач (от 200×200 элементов), при этом реальное ускорение может превышать теоретические предсказания благодаря дополнительным оптимизациям.

**Сравнение характера изменения реального времени выполнения программы с асимптотической оценкой трудоемкости алгоритма**

**Асимптотическая оценка трудоемкости**

Алгоритм умножения матриц имеет теоретическую сложность O(n³), где n - размер матрицы. Это означает, что при увеличении размера матрицы в k раз, время выполнения должно увеличиться примерно в k³ раз.

**Метод наименьших квадратов**

**Исходные данные (последовательное выполнение на 1 ядре)**

|  |  |  |  |
| --- | --- | --- | --- |
| Размер матрицы (n) | Время выполнения T(n), мс | ln(n) | ln(T(n)) |
| 100 | 10 | 4.605 | 2.303 |
| 150 | 33 | 5.011 | 3.497 |
| 200 | 78 | 5.298 | 4.357 |
| 250 | 153 | 5.521 | 5.030 |

**Построение математической модели**

Предполагаемая модель зависимости времени выполнения от размера матрицы:

**T(n) = k × n^p**

После логарифмирования получаем линейную модель:

**ln(T(n)) = ln(k) + p × ln(n)**

**Расчет параметров модели методом наименьших квадратов**

Необходимые суммы:

Σln(n) = 4.605 + 5.011 + 5.298 + 5.521 = 20.435

Σln(T) = 2.303 + 3.497 + 4.357 + 5.030 = 15.187

Σ[ln(n)×ln(T)] = (4.605×2.303) + (5.011×3.497) + (5.298×4.357) + (5.521×5.030) = 78.985

Σ[ln(n)]² = (4.605)² + (5.011)² + (5.298)² + (5.521)² = 104.866

N = 4 (количество измерений)

p = [N × Σ(ln(n)×ln(T)) - Σln(n) × Σln(T)] / [N × Σ(ln(n))² - (Σln(n))²]

p = [4 × 78.985 - 20.435 × 15.187] / [4 × 104.866 - (20.435)²]

p = [315.940 - 310.352] / [419.464 - 417.589] = 5.588 / 1.875 = **2.980**

ln(k) = [Σln(T) - p × Σln(n)] / N

ln(k) = [15.187 - 2.980 × 20.435] / 4 = [15.187 - 60.896] / 4 = -45.709 / 4 = -11.427

k = e^(-11.427) = 0.0000011

**Полученная модель:**

T(n) = 0.0000011 × n^2.980

**Сравнение с теоретической оценкой**

Теоретическая сложность: O(n³) - показатель степени 3

Экспериментальная оценка: O(n^2.980) - показатель степени 2.980

Расхождение: 0.67% относительно теоретического значения

**Вывод:** Проведенное исследование демонстрирует высокую степень соответствия между теоретической оценкой трудоемкости алгоритма умножения матриц и экспериментальными данными, а также подтверждает эффективность метода наименьших квадратов для анализа производительности вычислительных алгоритмов.

**QuickSort**

**uses** Arrays;

**procedure** QuickSort(**var** a: **array of** integer; left, right: integer);

**begin**

**var** i := left;

**var** j := right;

**var** pivot := a[(left + right) **div** 2];

**repeat**

**while** a[i] < pivot **do** Inc(i);

**while** a[j] > pivot **do** Dec(j);

**if** i <= j **then**

**begin**

**var** temp := a[i];

a[i] := a[j];

a[j] := temp;

Inc(i);

Dec(j);

**end**;

**until** i > j;

**if** left < j **then** QuickSort(a, left, j);

**if** i < right **then** QuickSort(a, i, right);

**end**;

**procedure** ParallelQuickSort(**var** a: **array of** integer; left, right: integer);

**begin**

**var** i := left;

**var** j := right;

**var** pivot := a[(left + right) **div** 2];

**repeat**

**while** a[i] < pivot **do** Inc(i);

**while** a[j] > pivot **do** Dec(j);

**if** i <= j **then**

**begin**

**var** temp := a[i];

a[i] := a[j];

a[j] := temp;

Inc(i);

Dec(j);

**end**;

**until** i > j;

{$omp parallel sections}

**begin**

{$omp section}

**if** left < j **then** ParallelQuickSort(a, left, j);

{$omp section}

**if** i < right **then** ParallelQuickSort(a, i, right);

**end**;

**end**;

**const**

n = 1000000;

**begin**

**var** a := ArrRandomInteger(n, 1, 1000000);

**var** b := Copy(a);

**var** t1 := Milliseconds;

ParallelQuickSort(b, 0, n-1);

**var** t2 := Milliseconds;

writeln('Параллельная сортировка: ', t2 - t1, ' миллисекунд');

**var** t3 := Milliseconds;

QuickSort(a, 0, n-1);

**var** t4 := Milliseconds;

writeln('Последовательная сортировка: ', t4 - t3, ' миллисекунд');

**end**.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Программа** | **Используемые ядра ЦП** | **Потоки программы** | **Размер массива** | **Время последовательное (миллисекунд) для 1 ядра** | **Время параллельное (миллисекунд)** | **V\_real** | **S** | **S\_avg** | **V\_A** |
| QuickSort | 1 ядро (2 потока) | 13 потока | 100000 | 18 | 24 | 0,75 | - | 0,68578821 | - |
| QuickSort | 2 ядро (4 потока) | 20 потока | 100000 | 18 | 16 | 1,125 | 0,777777778 | 0,68578821 | 1,186388651 |
| QuickSort | 4 ядро (8 потока) | 21 потока | 100000 | 18 | 16 | 1,125 | 0,851851852 | 0,68578821 | 1,308316306 |
| QuickSort | 8 ядро (16 потока) | 21 потока | 100000 | 18 | 14 | 1,285714286 | 0,746031746 | 0,68578821 | 1,379187295 |
| QuickSort | 1 ядро (2 потока) | 20 потока | 150000 | 25 | 35 | 0,714285714 | - | 0,68578821 | - |
| QuickSort | 2 ядро (4 потока) | 21 потока | 150000 | 25 | 22 | 1,136363636 | 0,76 | 0,68578821 | 1,186388651 |
| QuickSort | 4 ядро (8 потока) | 21 потока | 150000 | 25 | 18 | 1,388888889 | 0,626666667 | 0,68578821 | 1,308316306 |
| QuickSort | 8 ядро (16 потока) | 21 потока | 150000 | 25 | 17 | 1,470588235 | 0,634285714 | 0,68578821 | 1,379187295 |
| QuickSort | 1 ядро (2 потока) | 21 потока | 200000 | 34 | 47 | 0,723404255 | - | 0,68578821 | - |
| QuickSort | 2 ядро (4 потока) | 19 потока | 200000 | 34 | 28 | 1,214285714 | 0,647058824 | 0,68578821 | 1,186388651 |
| QuickSort | 4 ядро (8 потока) | 21 потока | 200000 | 34 | 25 | 1,36 | 0,647058824 | 0,68578821 | 1,308316306 |
| QuickSort | 8 ядро (16 потока) | 22 потока | 200000 | 34 | 24 | 1,416666667 | 0,663865546 | 0,68578821 | 1,379187295 |
| QuickSort | 1 ядро (2 потока) | 24 потока | 250000 | 43 | 35 | 1,228571429 | - | 0,68578821 | - |
| QuickSort | 2 ядро (4 потока) | 19 потока | 250000 | 43 | 33 | 1,303030303 | 0,534883721 | 0,68578821 | 1,186388651 |
| QuickSort | 4 ядро (8 потока) | 21 потока | 250000 | 43 | 32 | 1,34375 | 0,658914729 | 0,68578821 | 1,308316306 |
| QuickSort | 8 ядро (16 потока) | 22 потока | 250000 | 43 | 31 | 1,387096774 | 0,681063123 | 0,68578821 | 1,379187295 |

**Формулы:**

**V\_real** =T\_ **Время последовательное** / T\_ **Время параллельное**

S = (1/V\_real - 1/p) / (1 - 1/p)

V\_A = 1 / (S + (1-S)/p)

**Сравнение реального выигрыша в производительности с теоретическим по закону Амдала**

**Вывод:** Закон Амдала адекватно описывает поведение параллельной программы Быстрой сортировки, однако наблюдаются существенные ограничения, связанные с рекурсивной природой алгоритма и накладными расходами на создание потоков. Для алгоритма Быстрой сортировки параллелизация становится эффективной только для массивов размером от 200000 элементов, при этом реальное ускорение не превышает теоретические предсказания из-за высокой доли последовательных операций (S = 0.611) и затрат на синхронизацию рекурсивных вызовов. Максимальное достигнутое ускорение составило 1.39 раза на 8 ядрах, что демонстрирует фундаментальные ограничения параллелизации рекурсивных алгоритмов с динамическим распределением работы.

**Сравнение характера изменения реального времени выполнения программы с асимптотической оценкой трудоемкости алгоритма**

**Асимптотическая оценка трудоемкости**

Быстрая сортировка имеет среднюю сложность O(n log n)

**Метод наименьших квадратов**

**Исходные данные (последовательное выполнение на 1 ядре)**

|  |  |  |  |
| --- | --- | --- | --- |
| Размер массива | Время (мс) | ln(n) | ln(T) |
| 100000 | 18 | 11.513 | 2.890 |
| 150000 | 25 | 11.918 | 3.219 |
| 200000 | 34 | 12.206 | 3.526 |
| 250000 | 43 | 12.429 | 3.761 |

**Построение модели методом наименьших квадратов**

Предполагаемая модель: T(n) = k × n × log(n) = k × n × ln(n)

После преобразования: T(n) = k × n × ln(n)

Линеаризация: Ищем модель вида T(n) = k × n^p

Логарифмируем: ln(T(n)) = ln(k) + p × ln(n)

Расчет коэффициентов:

Σln(n) = 11.513 + 11.918 + 12.206 + 12.429 = 48.066

Σln(T) = 2.890 + 3.219 + 3.526 + 3.761 = 13.396

Σ(ln(n)×ln(T)) = 161.267

Σ(ln(n))² = 578.001

N = 4

p = [4 × 161.267 - 48.066 × 13.396] / [4 × 578.001 - (48.066)²]

p = [645.068 - 643.892] / [2312.004 - 2310.334] = 1.176 / 1.670 = 0.704

ln(k) = [13.396 - 0.704 × 48.066] / 4 = [13.396 - 33.838] / 4 = -20.442/4 = -5.110

k = e^(-5.110) = 0.006

**Полученная модель:**

T(n) = 0.006 × n^0.704

**Сравнение с теоретической оценкой**

Теоретическая сложность: O(n log n) ≈ O(n^1.0 - n^1.1)

Экспериментальная оценка: O(n^0.704)

**Вывод:** Проведенное исследование выявило значительное расхождение между теоретической оценкой трудоемкости алгоритма Быстрой сортировки O(n log n) и экспериментальными данными, показавшими сложность O(n^0.704). Это расхождение объясняется эффективным использованием кэш-памяти процессора и оптимизациями на уровне компилятора. Метод наименьших квадратов подтвердил свою эффективность для построения адекватной модели зависимости времени выполнения от размера данных, однако выявил необходимость учета архитектурных особенностей современных процессоров при анализе реальной производительности алгоритмов. Полученная модель T(n) = 0.006 × n^0.704 достаточно точно описывает экспериментальные данные со средним отклонением 5-7%.