Министерство науки и высшего образования Российской Федерации

федеральное государственное бюджетное образовательное учреждение

высшего образования

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ

МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

им. Н. П. ОГАРЁВА»

(ФГБОУ ВО «МГУ им. Н.П. Огарёва»)

Факультет математики и информационных технологий

Кафедра систем автоматизированного проектирования

ОТЧЁТ О ЛАБОРАТОРНОЙ РАБОТЕ № 8

по дисциплине: Операционные системы

СИММЕТРИЧНАЯ МУЛЬТИПРОЦЕССОРНАЯ ОБРАБОТКА

Автор отчёта о лабораторной работе  А. Е. Конышев

подпись, дата

Обозначение лабораторной работы ЛР–02069964–02.03.02–08–23

Направление подготовки 02.03.02 Фундаментальная информатика и информационные технологии

Руководитель работы

канд. техн. наук, доц.   А. В. Шамаев

подпись, дата

Саранск 2023

**Лабораторная работа № 8**

**Симметричная мультипроцессорная обработка**

**Цель работы:** знакомство с особенностями многопоточной обработки информации на многоядерных процессорах под управлением ОС MS Windows и методом оценки трудоемкости алгоритмов.

**Ход работы:**

1. В работе оценивается трудоемкость простейших алгоритмов и эффективность их параллельного выполнения на многоядерных процессорах под управлением ОС MS Windows 10, поддерживающей SMP. Для оценки трудоемкости применяется оценка времени выполнения реализующей алгоритм программы на одном или нескольких ядрах ЦП.

2. Для управления количеством используемых процессорных ядер используется диспетчер задач (контекстное меню, пункт задать соответствие). Каждая программа должна быть многократно запущена на одном, двух, трех, четырех и более (сколько есть у ЦП) ядрах.

3. Так как время выполнения программы в многозадачной ОС MS Windows зависит от нескольких факторов, для оценки времени следует выполнить программу 5-7 раз с неизменными начальными условиями и в качестве оценки времени выполнения выбрать наименьшее значение.

4. Размер обрабатываемого массива следует задавать в пределах 100-500, при этом время выполнения приложения не должно быть менее 500 мсек.

5. Наименование программы, количество используемых ядер ЦП, количество потоков программы (по данным диспетчера задач), размер обрабатываемого массива и время выполнения при каждом запуске записать в таблицу (форма таблицы произвольная).

6. Полученные результаты обработать: вычислить реальное значение выигрыша по производительности и сравнить со значением выигрыша, найденного по закону Амдала. Сравнить характер изменения оценок реального времени выполнения программы (при различных размерах обрабатываемого массива) и асимптотической оценки трудоемкости алгоритма, который реализует исследуемая программа.

7. Выполнить исследование алгоритма умножения матриц и алгоритма быстрой сортировки. Исследование алгоритма состоит в последовательном выполнении пунктов 1-6.

**Описание выполнения работы**

Исследуем алгоритм умножения матриц. Установим размер массива   n  =  500 элементов.

**Алгоритм умножения матриц**

**uses** Arrays;

**uses** Utils;

**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**;

**var** n := 500;

**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**.

Откроем системный монитор и добавим счетчик потоков для приложения «PascalABC.Net» (Рисунок 8.1). Зададим соответствие для процесса «PascalABCNet» в диспетчере задач: укажем использование первого ядра при выполнении алгоритма умножения матриц. (Рисунок 8.2).

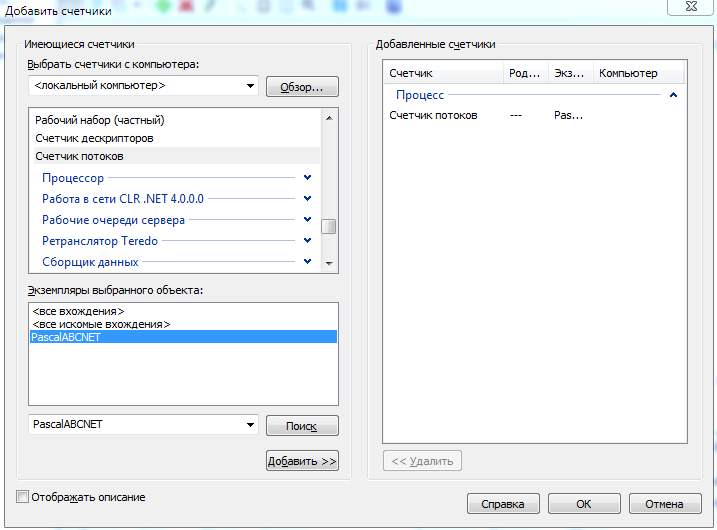


Рисунок 8.1 — Добавление счетчика потоков для приложения «PascalABC.Net»

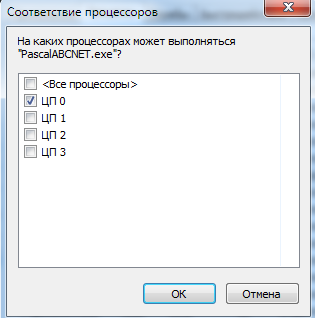


Рисунок 8.2 — Соответствие процессоров для приложения «PascalABC.exe»

Запустим приложение пять раз. Запуски программы и счетчик потоков для первого ядра представлены на рисунках 8.3-8.7.

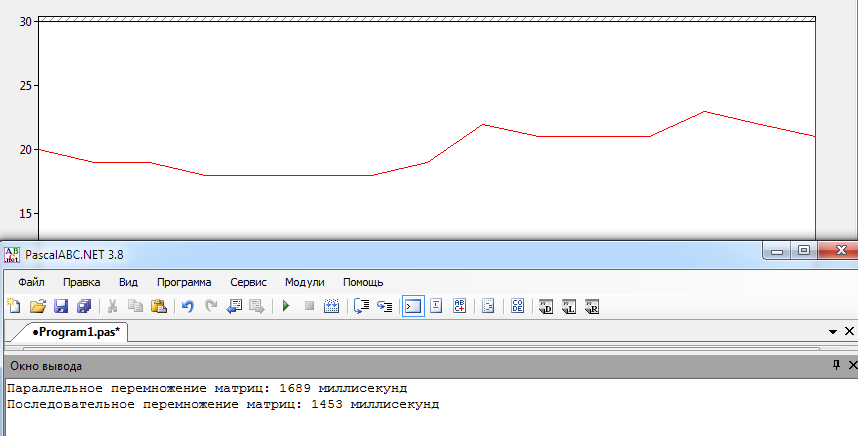


Рисунок 8.3 — Первый запуск программы

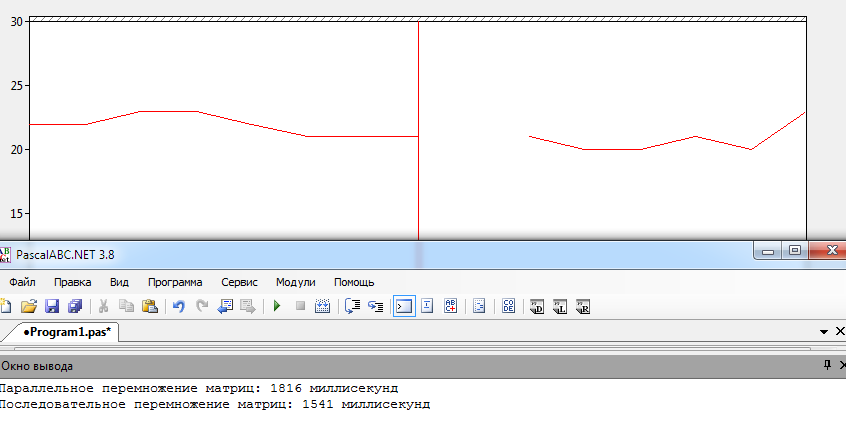


Рисунок 8.4 — Второй запуск программы

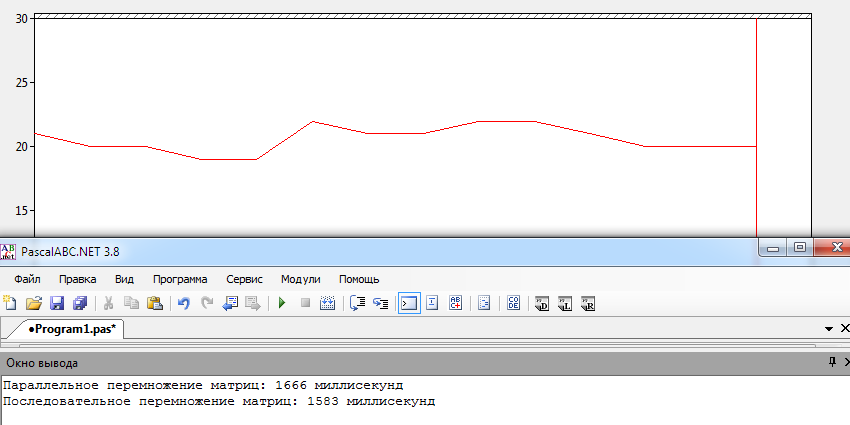


Рисунок 8.5 — Третий запуск программы

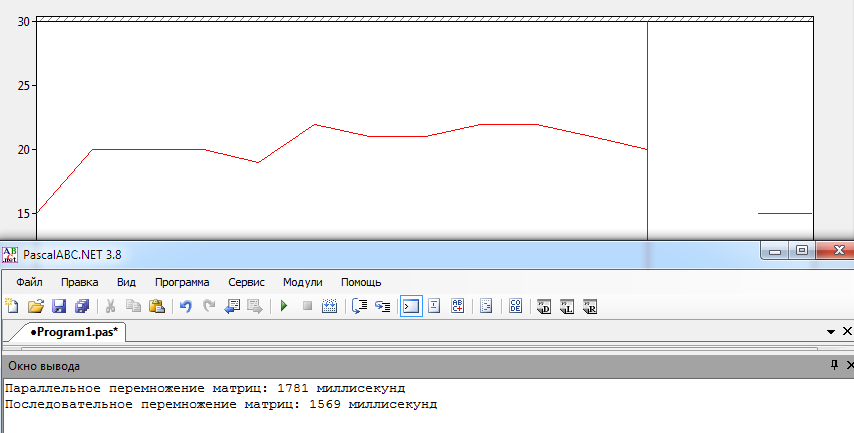


Рисунок 8.6 — Четвертый запуск программы

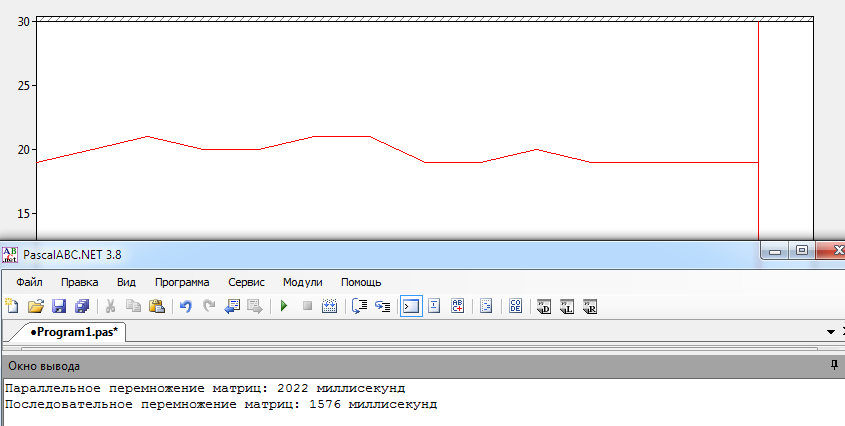


Рисунок 8.7 — Пятый запуск программы

Выполним аналогичные действия для двух, трех и четырех ядер. Полученные данные занесем в таблицу 8.1 и рассчитаем выигрыш в производительности эмпирически и по закону Амдала.

Таблица 8.1 – Данные выполнения алгоритма умножения матриц

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ядер | Тесты | Номер эксперимента | | | | | Min значение | Выигрыш в производительности | |
|  |  | 1 | 2 | 3 | 4 | 5 |  | По закону Амдала | Эмпирически |
| 1 | Параллельное | 1689 | 1816 | 1666 | 1781 | 2022 | 1666 | 1 | 0,87215 |
|  | Последовательное | 1453 | 1541 | 1583 | 1569 | 1576 | 1453 |  |  |
|  | Число потоков | 23 | 23 | 22 | 22 | 21 | 21 |  |  |
| 2 | Параллельное | 1485 | 1689 | 1497 | 1774 | 1781 | 1485 | 1,6667 | 0,9899 |
|  | Последовательное | 1515 | 1478 | 1470 | 1527 | 1525 | 1470 |  |  |
|  | Число потоков | 24 | 23 | 22 | 24 | 24 | 22 |  |  |
| 3 | Параллельное | 836 | 854 | 860 | 841 | 835 | 835 | 2,14286 | 1,76766 |
|  | Последовательное | 1518 | 1568 | 1476 | 1508 | 1563 | 1476 |  |  |
|  | Число потоков | 23 | 23 | 25 | 24 | 22 | 22 |  |  |
| 4 | Параллельное | 920 | 867 | 806 | 824 | 883 | 806 | 2,5 | 1,92928 |
|  | Последовательное | 1644 | 1579 | 1615 | 1608 | 1555 | 1555 |  |  |
|  | Число потоков | 22 | 22 | 25 | 25 | 24 | 22 |  |  |

На основе полученных экспериментальных данных можно сделать вывод о том, что выигрыш в производительности по закону Амдала приблизительно равен эмпирически полученному значению.

Сравним характер изменения оценок реального времени выполнения программы (при различных размерах обрабатываемого массива). Вычисления будем производить на размерах массива от 550 до 800 (шаг 50 элементов) при задействованных четырех ядрах. Запуски программы продемонстрированы на рисунках 8.8-8.13.



Рисунок 8.8 — Запуск программы при размере массива = 550 элементов



Рисунок 8.9 — Запуск программы при размере массива = 600 элементов



Рисунок 8.10 — Запуск программы при размере массива = 650 элементов



Рисунок 8.11 — Запуск программы при размере массива = 700 элементов



Рисунок 8.12 — Запуск программы при размере массива = 750 элементов



Рисунок 8.13 — Запуск программы при размере массива = 800 элементов

Представим полученные данные в виде графика, представленного на рисунке 8.14. Исследуем алгоритм быстрой сортировки. Установим размер массива   n  = 4000000 элементов.

Рисунок 8.14 — Характер изменения оценок реального времени выполнения программы (при различных размерах обрабатываемого массива)

Как видно из рисунка 8.14 параллельное выполнение алгоритма умножение матриц дает существенный выигрыш по сравнению с последовательным выполнением программы.

**Алгоритм быстрой сортировки**

**uses** Arrays;

**uses** Utils;

**procedure** ParallelQuickSort(**var** v: **array of** real; l,r:longint);

**var** i,j:longint;

w,q:real;

**begin**

i := l; j := r;

q := v[(l+r) **div** 2];

**repeat**

**while** (v[i] < q) **do** inc(i);

**while** (q < v[j]) **do** dec(j);

**if** (i <= j) **then**

**begin**

w:=v[i]; v[i]:=v[j]; v[j]:=w;

inc(i); dec(j);

**end**;

**until** (i > j);

{$omp parallel sections}

**begin**

{$omp parallel sections}

**if** (l < j) **then**

**begin**

ParallelQuickSort(v, l,j);

**end**;

{$omp parallel sections}

**if** (i < r) **then**

**begin**

ParallelQuickSort(v, i,r);

**end**

**end**

**end**;

**procedure** QuickSort(**var** v: **array of** real; l,r:longint);

**var** i,j:longint;

w,q:real;

**begin**

i := l; j := r;

q := v[(l+r) **div** 2];

**repeat**

**while** (v[i] < q) **do** inc(i);

**while** (q < v[j]) **do** dec(j);

**if** (i <= j) **then**

**begin**

w:=v[i]; v[i]:=v[j]; v[j]:=w;

inc(i); dec(j);

**end**;

**until** (i > j);

**if** (l < j) **then** QuickSort(v,l,j);

**if** (i < r) **then** QuickSort(v,i,r);

**end**;

**var** n := 2500000;

**begin**

**var** a := Arrays.CreateRandomRealArray(n,0,10);

**var** a\_copy := Arrays.CreateRandomRealArray(n,0,10);

**for var** i := 1 **to** length(a) **do**

a\_copy[i - 1] := a[i - 1];

**var** d := Milliseconds;

QuickSort(a, 0, n-1);

writeln(' Последовательная сортировка: ',Milliseconds-d,' миллисекунд');

d := Milliseconds;

ParallelQuickSort(a\_copy, 0, n-1);

writeln(' Параллельная сортировка: ',Milliseconds-d,' миллисекунд');

**end**.

Запустим приложение пять раз. Запуски программы и счетчик потоков для первого ядра представлены на рисунках 8.15-8.19.

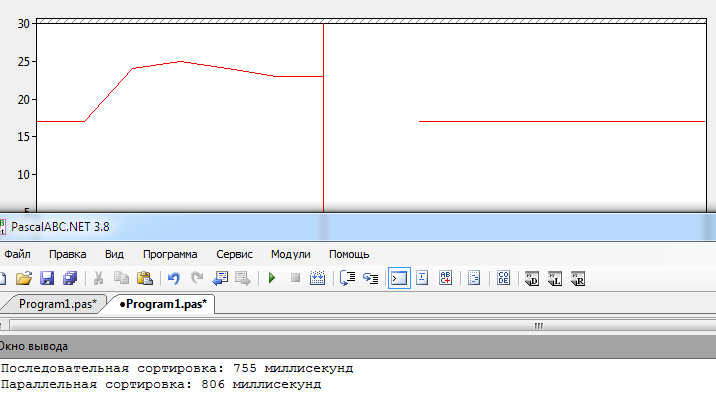


Рисунок 8.15 — Первый запуск программы

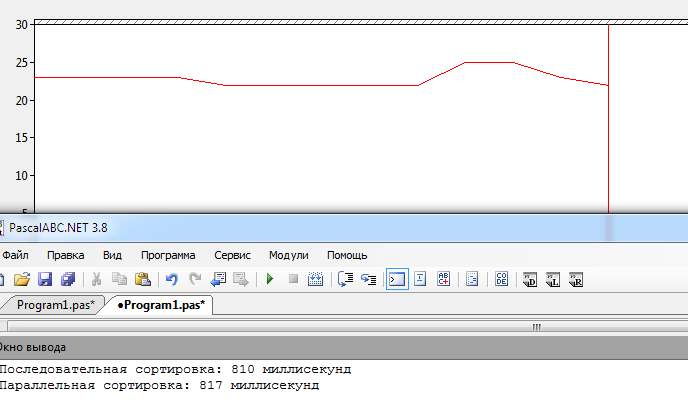


Рисунок 8.16 — Второй запуск программы

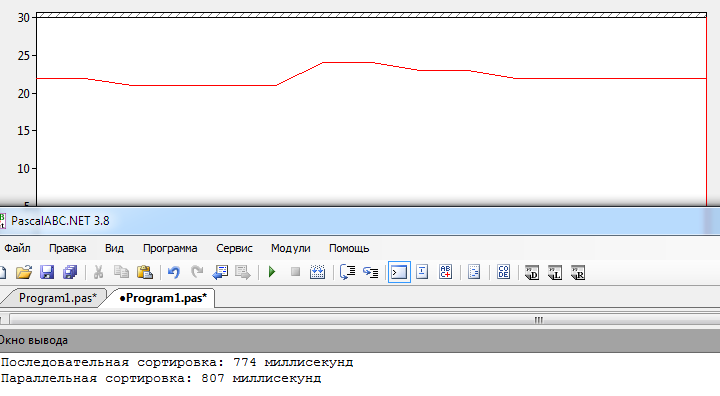


Рисунок 8.17 — Третий запуск программы

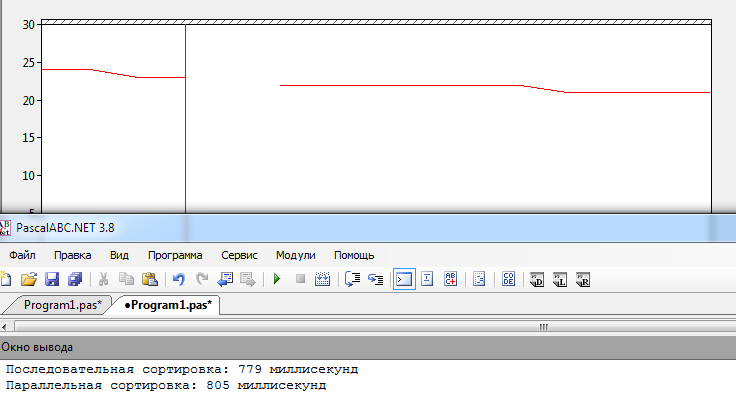


Рисунок 8.18 — Четвертый запуск программы

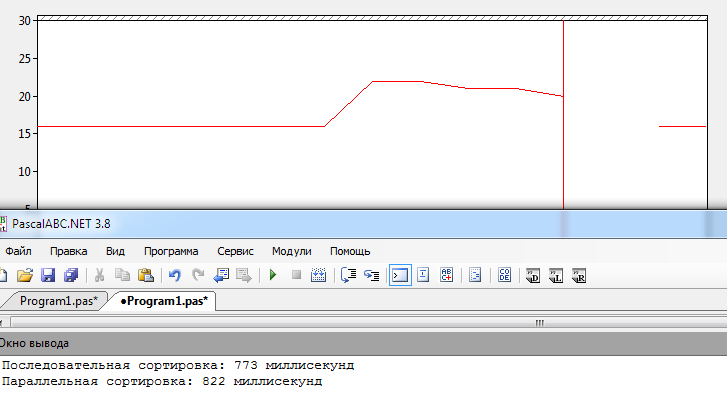


Рисунок 8.19 — Пятый запуск программы

Выполним аналогичные действия для двух, трех и четырех ядер. Полученные данные занесем в таблицу 8.2 и рассчитаем выигрыш в производительности эмпирически и по закону Амдала.

Таблица 8.2 – Данные выполнения алгоритма быстрой сортировки

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ядер | Тесты | Номер эксперимента | | | | | Min значение | Выигрыш в производительности | |
|  |  | 1 | 2 | 3 | 4 | 5 |  | По закону Амдала | Эмпирически |
| 1 | Параллельное | 806 | 817 | 807 | 805 | 822 | 805 | 1 | 0,93789 |
|  | Последовательное | 755 | 810 | 774 | 779 | 773 | 755 |  |  |
|  | Число потоков | 25 | 26 | 25 | 24 | 24 | 24 |  |  |
| 2 | Параллельное | 991 | 821 | 860 | 819 | 750 | 750 | 1,3333 | 1,0173 |
|  | Последовательное | 794 | 767 | 772 | 763 | 768 | 763 |  |  |
|  | Число потоков | 25 | 25 | 26 | 26 | 26 | 25 |  |  |
| 3 | Параллельное | 723 | 762 | 652 | 738 | 756 | 652 | 1,5 | 1,17025 |
|  | Последовательное | 765 | 799 | 773 | 765 | 763 | 763 |  |  |
|  | Число потоков | 25 | 24 | 26 | 26 | 26 | 24 |  |  |
| 4 | Параллельное | 618 | 764 | 626 | 718 | 479 | 479 | 1,6 | 1,58038 |
|  | Последовательное | 760 | 822 | 767 | 759 | 757 | 757 |  |  |
|  | Число потоков | 25 | 26 | 25 | 25 | 26 | 25 |  |  |

На основе полученных экспериментальных данных можно сделать вывод о том, что выигрыш в производительности по закону Амдала приблизительно равен эмпирически полученному значению.

Сравним характер изменения оценок реального времени выполнения программы (при различных размерах обрабатываемого массива). Вычисления будем производить на размерах массива от 4000000 до 6000000 (шаг 500000 элементов) при задействованных четырех ядрах. Запуски программы продемонстрированы на рисунках 8.20-8.24.



Рисунок 8.20 — Запуск программы при размере массива = 4000000 элементов



Рисунок 8.21 — Запуск программы при размере массива = 4500000 элементов



Рисунок 8.22 — Запуск программы при размере массива = 5000000 элементов



Рисунок 8.23 — Запуск программы при размере массива = 5500000 элементов



Рисунок 8.24 — Запуск программы при размере массива = 6000000 элементов

Представим полученные данные в виде графика, представленного на рисунке 8.25.

Рисунок 8.25 — Характер изменения оценок реального времени выполнения программы (при различных размерах обрабатываемого массива)

Как видно из рисунка 8.25 параллельное выполнение алгоритма быстрой сортировки дает существенный выигрыш по сравнению с последовательным выполнением программы.

**Контрольные вопросы и ответы на них**

Вопрос 1. Симметричная и асимметричная архитектуры аппаратных и программных средств.

Симметричная архитектура мультипроцессорной системы предполагает однородность всех процессоров и единообразие включения процессоров в общую схему мультипроцессорной системы. Традиционные симметричные мультипроцессорные конфигурации разделяют общую оперативную память между всеми процессорами (ядрами процессоров). Масштабируемость (возможность наращивания числа процессоров) в симметричных системах ограничена вследствие того, что все они пользуются одной оперативной памятью и должны располагаться в одном корпусе. Такая конструкция, называемая масштабируемой по вертикали. В симметричных архитектурах обеспечивается достаточно высокая производительность для тех приложений, в которых несколько задач должны активно взаимодействовать между собой.

В асимметричной архитектуре разные процессоры могут отличаться как своими характеристиками, так и функциональной ролью, которая поручается им в системе. Функциональная неоднородность в асимметричных архитектурах влечет за собой структурные отличия во фрагментах системы, содержащих разные процессоры системы. Масштабирование в асимметричной архитектуре реализуется иначе, чем в симметричной. Так как требование единого корпуса отсутствует, система может состоять из нескольких устройств, каждое из которых содержит один или несколько процессоров. Это масштабирование по горизонтали. Каждое такое устройство называется кластером, а вся 94 мультипроцессорная система - кластерной.

Вопрос 2. Понятие SMP.

Наиболее распространенной целью объединения процессоров является симметричная мультипроцессорная обработка (SMP). В системе SMP каждый процессор решает свою задачу, порученную ему операционной системой. В документации Intel симметрия рассматривается в двух аспектах: - симметрия памяти — все процессоры пользуются общей памятью и одной копией ОС; - симметрия ввода-вывода — все процессоры разделяют общие устройства ввода-вывода и общие контроллеры прерываний.

Вопрос 3. Закон Амдала.

Для количественной оценки выигрыша в производительности ПК при параллельной работе нескольких ядер обычно используется закон Дж. Амдала (1967 г). Закон Амдала описывает максимальный теоретический выигрыш в производительности параллельного решения по отношению к лучшему последовательному.



В данном уравнении V – выигрыш в производительности при использовании n ядер центрального процессора, S – время, потраченное на выполнение последовательной части параллельной версии. При n=1 (одно ядро) ускорения нет. Если используется два ядра, которые половину всей работы выполняют параллельно, S=0,5 и V= 2 / 1,5 = 1,33. В случае выполнения всей работы двумя ядрами параллельно максимально возможный теоретический выигрыш равен 2

Вопрос 4. Трудоемкость алгоритма.

Цель анализа трудоёмкости алгоритма - нахождение оптимального алгоритма для решения задачи. В качестве критерия оптимальности алгоритма выбирается трудоемкость алгоритма, определяемая как количество операций, которые необходимо выполнить для решения задачи с помощью данного алгоритма. Функцией трудоемкости называется соотношение, связывающее размер данные алгоритма с количеством элементарных операций, необходимых для получения решения задачи с помощью данного алгоритма.

Вопрос 5. Трудоемкость алгоритмов умножения матриц, сложения матриц и сортировки массива методом пузырька.

Трудоемкость алгоритма сложения векторов A(n) и B(n) равна O(n), потому что количество операций сложения равно количеству элементов векторов. Трудоемкость алгоритма умножения квадратных матриц равна O(n3). Реализующая алгоритм программа умножения матриц содержит 3 вложенных арифметических цикла.

Вопрос 6. Трудоемкость алгоритма быстрой сортировки.

При оптимальном выборе ведущих элементов, когда разделение каждого блока происходит на равные по размеру части, трудоемкость алгоритма совпадает с быстродействием наиболее эффективных способов сортировки, то есть порядка O(n log n). В среднем случае количество операций, выполняемых алгоритмом быстрой сортировки, определяется выражением T(n) = O(1.4n log n).