МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Вычислительная техника»

Дисциплина «Высокопроизводительные вычисления»

**Лабораторная работа №3.**

**Исследование многопоточных реализаций алгоритма численного интегрирования в среде одной ПЭВМ**

Выполнил:

студент группы ИВТАПбд-31

Кондратьев П.С.

Проверил:

Негода В. Н.

Ульяновск, 2019

Оглавление

[Варианты заданий 3](#_Toc2686031)

[Введение 3](#_Toc2686032)

[Цель работы: 5](#_Toc2686033)

[Интегрирование по методу прямоугольников 6](#_Toc2686034)

[Программная реализация 9](#_Toc2686035)

[Параллельная версия. Распараллеливание базового алгоритма 11](#_Toc2686036)

[Примечание 14](#_Toc2686037)

[Литература 15](#_Toc2686038)

[Приложение 1. 16](#_Toc2686039)

[Приложение 2. 22](#_Toc2686040)

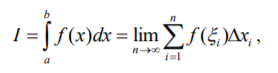
[Приложение 3. 23](#_Toc2686041)

# Варианты заданий

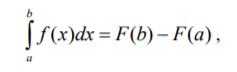
|  |  |
| --- | --- |
| № | Функция |
| 11 | *ch x* |

# Введение

Задача вычисления определенного интеграла I для некоторой заданной на отрезке [a, b] функции f(x) является классической задачей математического анализа. Известно, что для функций, имеющих на [a, b] конечное число точек разрыва первого рода, такое значение существует, единственно и может быть формально получено по определению как



В математическом анализе обосновывается аналитический способ нахождения значения интеграла с помощью знаменитой формулы НьютонаЛейбница:



где F(x) – некоторая первообразная для данной функции f(x). К сожалению, применение этого весьма привлекательного подхода к вычислению I наталкивается на несколько серьезных препятствий.

Во-первых, для многих элементарных функций f(x) не существует первообразной среди элементарных функций: например, отсутствуют первообразные для функций

*ch x*

Во-вторых, даже если первообразная F(x) для заданной функции f(x) найдена, то вычисление двух ее значений F(a) и F(b) может оказаться более трудоемким, чем вычисление существенно большего количества значений f(x).

И наконец, для многих реальных приложений определенного интеграла характерна дискретность задания подынтегральной функции, что делает указанный аналитический подход не применимым в принципе.

Сказанное предопределяет необходимость использования приближенных формул для вычисления определенного интеграла на основе значений подынтегральной функции f(x). Такие специальные приближенные формулы называют квадратурными формулами или формулами численного интегрирования. Происхождение термина можно связать с геометрическим смыслом определенного интеграла: вычисление I при f(x) ≥ 0 равносильно построению квадрата, равновеликого криволинейной трапеции с основанием [a, b] и «крышей» f(x).

Простейшую квадратурную формулу – формулу прямоугольников – можно вывести непосредственно из определения интеграла, зафиксировав некоторое n, выбрав равномерную систему точек ξi = (xi + xi−1 )/ 2.

В данной лабораторной работе мы ограничимся использованием формулы прямоугольников. В работе будут рассмотрены: различные подходы к распараллеливанию метода прямоугольников; идеи по алгоритмической оптимизации, приводящие к уменьшению времени вычислений; методы параллельной отладки.

# Цель работы:

Цель данной работы – изучение принципов написания высокопроизводительных реализаций алгоритмов с использованием современных компилирующих и отладочных средств.

Данная цель предполагает решение следующих основных задач:

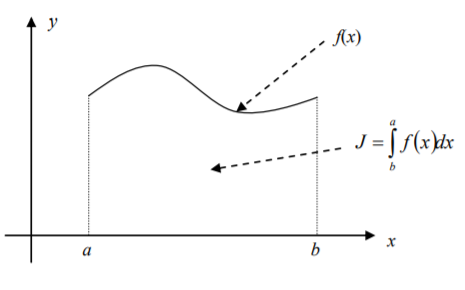
1. Изучение общей схемы алгоритма численного расчета определенного интеграла.
2. Написание нескольких программных реализаций численного расчета определенного интеграла и сравнение их производительности.
3. Демонстрация использования механизма написания параллельных программ – OpenMP.

# Интегрирование по методу прямоугольников

На практике, довольно часто приходится подсчитывать значение определенного интеграла. Например, в задачах физики, если известно распределение плотности тока, общий ток в сети можно узнать, вычислив определенный интеграл. В теории вероятности один из способов задать случайную величину – задать плотность распределения вероятности. Для подсчета вероятности, на практике, также вычисляют определенный интеграл.

Методов вычисления интеграла достаточно много. Распространенным способом вычисления определенного интеграла, является применение квадратурных формул, таких как методы прямоугольников или трапеций.

Пусть на отрезке [a, b] задана непрерывная неотрицательная функция y = f (x). В этом случае значение определенного интеграла от f(x) на отрезке [a, b] совпадает с площадью фигуры, ограниченной графиком функции, осью Ox и прямыми x = a, x = b (см. рис. 1).



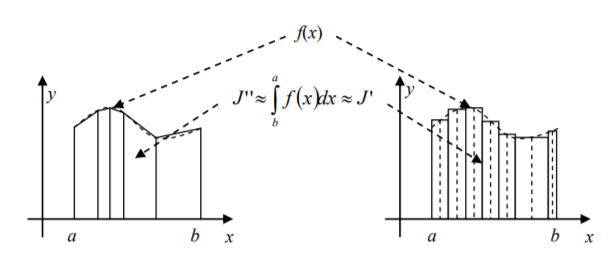
**Рис**. **1.** Геометрический смысл определенного интеграла

При решении прикладных задач приходится сталкиваться с достаточно

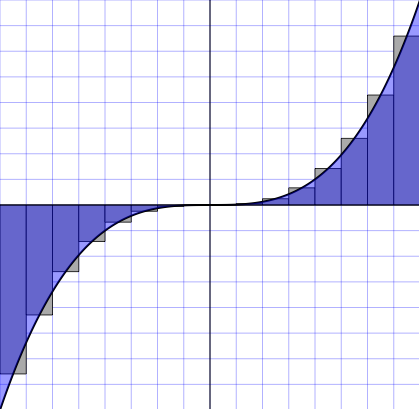
сложными функциями, которые не интегрируются аналитически. В этом случае значение интеграла вычисляется приближенно. Если функция близка к константе, то значение интеграла можно заменить площадью прямоугольника. Если функция близка к линейной, то интеграл может быть подсчитан через площадь трапеции. Погрешность вычислений квадратурных формул метода прямоугольников может быть вычислена по следующей формуле:



Для повышения точности вычисления интеграла отрезок интегрирования разбивается на смежные непересекающиеся отрезки. На каждом отрезке, значение интеграла также может быть заменено на площадь прямоугольника или трапеции. На рис. 2 изображен пример численного вычисления интеграла методами трапеций и прямоугольников.



**Рис. 2.** Численное вычисление интеграла методами трапеций и прямоугольников



**Рис. 3.** Метод средних прямоугольников

Если отрезок [a, b] является элементарным и не подвергается дальнейшему разбиению, значение интеграла можно найти по

1. Формуле левых прямоугольников:
2. Формуле правых прямоугольников:
3. Формуле прямоугольников (средних):

**Составные формулы для равномерных сеток**

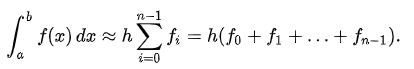
Равномерную сетку можно описать следующим набором формул:



где h — шаг сетки.

Для равномерных сеток формулы прямоугольников можно записать в виде следующих формул Котеса:

1. Составная формула левых прямоугольников:



1. Составная формула правых прямоугольников:



1. Составная формула средних прямоугольников если нет возможности изменять точки в которых вычисляется значение функции выглядит так



# Программная реализация

Программную реализацию начнем с написания функции, выполняющей расчет приближенного значения интеграла для учебного примера.

Далее создадим заготовку функции main(). Функция должна содержать код, несколько раз запускающий тестируемую реализацию алгоритма, и вычисляющий времени ее работы.

int main()

{

setlocale(LC\_CTYPE, "rus");

int flow, i;

double time;

// speedup

for (flow = 1; flow <= 8; flow++)

{

for (i = 100; i <= SIZE\_DARR; i \*= 10)

{

cout << endl << "Поток " << flow << ": " << i << " элементов" << endl << "Время (с) \tРезультат \t\n";

time = experiment(i, flow);

cout.width(10);

cout.setf(ios::right);

cout << time << "\t" << res << endl;

}

}

system("pause");

return 0;

}

Далее необходимо реализовать функцию experiment(). Данная функция должна возвращать значение подсчитанного интеграла. Результатом выполнения функции должно быть время работы программной реализации алгоритма интегрирования. Внутри функции должны быть заданы параметры экспериментов. Также функция должна содержать код, вызывающий реализацию алгоритма интегрирования и замеряющий время ее работы. Отметим, что здесь и во всех остальных функциях мы используем вещественные числа с двойной точностью, то есть тип double. Код, который необходимо написать, представлен ниже.

double experiment(int i, int flow)

{

double a = 1.0; // левая граница интегрирования

double b = 8.0; // правая граница интегрирования

double tmin = 100000000.0;

for(int k = 1; k <= 100000000 / i; k++)

{

stime = omp\_get\_wtime();

res = CalcIntegral(a, b, i, flow);

ftime = omp\_get\_wtime();

if (tmin > ftime - stime)

tmin = ftime - stime;

}

return tmin;

}

Следующим шагом напишем функцию CalcIntegral(). Данная функция через параметр должна возвращать значение подсчитанного интеграла.

double CalcIntegral(double a, double b, int n)

{

int i;

double sum, h;

sum = 0.0;

// n - количество отрезков интегрирования

h = (b - a) / n; //Шаг сетки

for (i = 0; i < n; i++) {

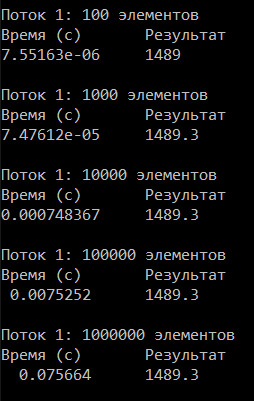
sum += InFunction(a + h \* (i + 0.5)); //Вычисляем в средней точке и добавляем в сумму

}

sum \*= h;

return sum;

}



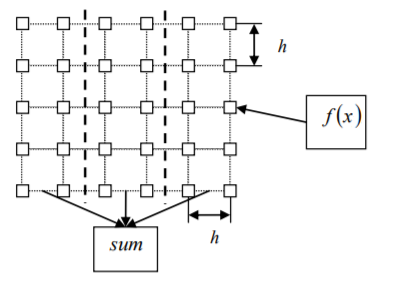
**Рис. 4.** Результаты базовой версии кода

# Параллельная версия. Распараллеливание базового алгоритма

Следующим шагом предлагается распараллелить базовую реализацию алгоритма.

В системах с общей памятью основной способ распараллеливания – использование многопоточности. Существуют различные реализации механизмов работы с потоками. Наиболее известные – p\_thread под операционными система семейства Linux и Windows Threads под ОС Windows соответственно. К сожалению, применение потоков в явном виде часто является непростой задачей. Для упрощения написания параллельных программ, использующих потоки, часто применяют технологию OpenMP.

Используя средства OpenMP, реализуем первую параллельную версию подсчета интеграла с разделением сетки интегрирования по столбцам (рис. 5).



**Рис. 5.** Схема алгоритма с разделением данных по столбцам

Согласно схеме разбиения данных, представленной на рис. 5, необходимо распараллеливать внешний цикл функции CalcIntegral(). Применим директиву OpenMP omp parallel for, позволяющую создать потоки и распределить итерации цикла между ними.

double CalcIntegral(double a, double b, int n, int flow)

{

int i;

double sum, h;

sum = 0.0;

// n - количество отрезков интегрирования

h = (b - a) / n; //Шаг сетки

omp\_set\_dynamic(0);

#pragma omp parallel for private(i) reduction(+:sum) num\_threads(flow)

for (i = 0; i < n; i++) {

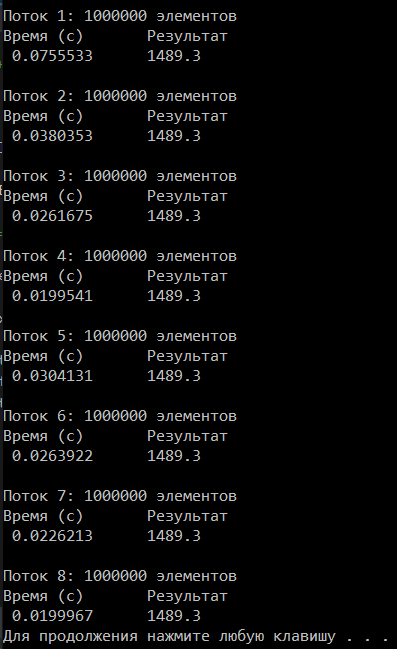
sum += InFunction(a + h \* (i + 0.5)); //Вычисляем в средней точке и добавляем в сумму

}

sum \*= h;

return sum;

}



**Рис. 6.** Результаты параллельной версии

Так же была указана модель использования для потоков, которая может состоять из двух моделей исполнения: динамическая, когда количество используемых потоков в программе может варьироваться от одной области параллельного выполнения к другой, и статическая, когда количество потоков фиксировано.

Модель исполнения контролируется или через переменную окружения OPM\_DYNAMIC или с помощью вызова функции omp\_set\_dynamic().

В дополнение к выше перечисленному, добелено условие private, которое указывает на то, что каждый поток должен иметь свою копию переменной на всем протяжении своего исполнения и reduction, которое гарантирует безопасное выполнение операций редукции, например, вычисление глобальной суммы.

Все запуски тестируемой программы проводилась в режиме Debug, при запуске в Release, возникает усложняющий фактор. Дело в том, что сборка в конфигурации Release предполагает оптимизацию кода, часто связанную с существенными изменениями, вносимыми в процессе компиляции в объектный код. В силу большей сложности параллельного кода, компилятор может в нем оптимизировать далеко не все и не так, как в коде последовательном.

# Примечание

Параметры машины, в среде которой проводились измерения, приведены в **Приложение 1**.

Базовые сведения о механизм написания параллельных программ, приведены в **Приложение 2**.

Сведенье о результатах замеров времени вычисления при варьировании числа потоков (от 1 до 8), числа ядер и гранулярности задачи (от 100 вычислений подынтегральной функции до 1000000), приведены в **Приложение 3**.

# Литература

1. Ильин В.А., Позняк Э.Г. Основы математического анализа. Часть I. – М.: Физматлит, 2005. – 648 с.
2. Бахвалов Н.С., Жидков Н.П., Кобельков Г.М. Численные методы. –Бином. Лаборатория знаний, 2008. – 640 c.
3. Параллельное программирование на OpenMP <http://ccfit.nsu.ru> /arom/data/openmp.pdf (Дата обращения 04.03.19)
4. Метод прямоугольников [https://ru.wikipedia.org/wiki/Метод\_пря -моугольников](https://ru.wikipedia.org/wiki/Метод_пря%20-моугольников) (Дата обращения 04.03.19)

# Приложение 1.

Сводная таблица характеристик, тестируемых ЭВМ;

|  |  |
| --- | --- |
| Наименование характеристик | Компьютер 1 |
| Тип процессора | DualCore Intel Core i5-7200U |
| Исходная частота | 2500 МГц |
| Максимальная частота | 3100 MHz |
| Архитектура процессора | Kaby Lake-U |
| Cash-памяти | Кэш L1 — 2x32+2x32 Кб per core  Кэш L2 — 256x2 Кб per core (On-Die, ECC, Full-Speed)  Кэш L3 — 3 Мб (On-Die, ECC, Full-Speed) |
| Тип сокета | BGA1356 |
| Напряжение питания процессора | 0.6 V |
| Мощность процессора | 15 W |
| Количество ядер/потоков | 2/4 |
| Производитель | Asus |
| Материнская плата | Asus VivoBook X556UQK |
| Наличие и тип слотов PCI | PCI  PCI Express 2.0 |
| Тип видеокарты (встроенная, дополнительная) | Intel HD Graphics 620  NVIDIA GeForce 940MX |
| Тип шины | Встроено  PCI-устройство 8086-5916 / 1043-1490 (Rev 02) |
| Объем памяти видеокарты | 1 Гб  2 Гб |
| Физическая память ЭВМ | HGST HTS721010A9E630 (1 ТБ, 7200 RPM, SATA-III) |
| Оперативная память (объем, тип и скорость) | 8Гб SO-DIMM DDR4 15-15-15-35 4-50-17-8 2T  DDR3 SO-DIMM DDR4  2x 1ГГц (2.13ГГц) |
| Драйвер видеокарты | 23.20.16.4905 |
| Интерфейсы ввода-вывода | USB 2.0, USB 3.0, USB type C, HDMI, microSD, Audio 3.5, LAN, VGA |
| ОС | Windows 10 |

Вторым тестируемым компьютером стал СУПЕРКОМПЬЮТЕР «ГОВОРУН» учебно-тестовый полигон «HybriLIT». (url: <http://hlit.jinr.ru>)

Гетерогенная платформа «HybriLIT» является частью Многофункционального информационно-вычислительного комплекса (МИВК), Лаборатории информационных технологий ОИЯИ, г. Дубна. Гетерогенная платформа состоит из Суперкомпьютера «ГОВОРУН» и учебно-тестового полигона “HybriLIT”.

Суперкомпьютер «ГОВОРУН» представляет собой двухкомпонентную систему: CPU-компоненту, базирующуюся на новейших архитектурах Intel (процессоров Intel Xeon Phi и Intel Skylake) и GPU-компоненты, базирующейся на узлах NVIDIA DGX-1 Volta.

Учебно-тестовый полигон имеет гетерогенную структуру вычислительных узлов и позволяет разрабатывать параллельные приложения для проведения расчетов на различных вычислительных архитектурах, таких как многоядерные процессоры, сопроцессоры Intel Xeon Phi и линейки графических процессоров NVIDIA (Testla K20, K40, K80), а также проводить учебные курсы по технологиях параллельного программирования, позволяющим студентам осваивать работу на новейших вычислительных архитектурах.

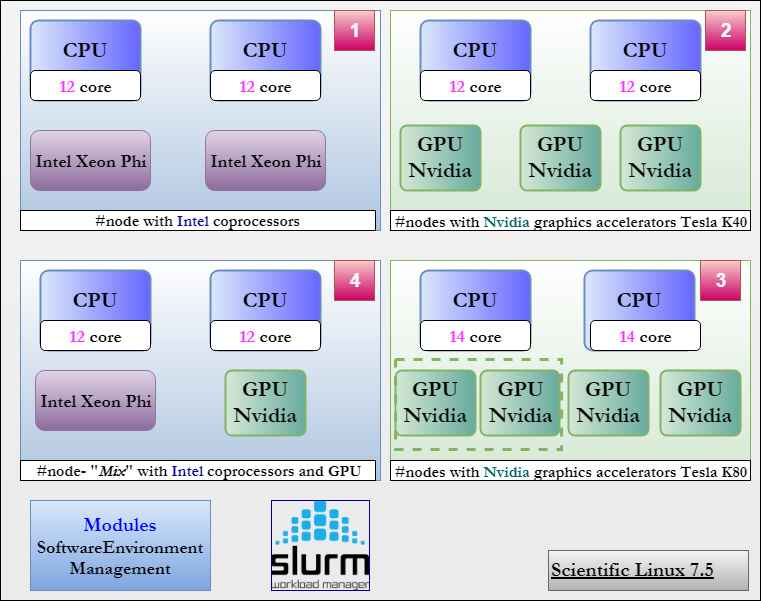
Программно-аппаратная среда платформы HybriLIT

Гетерогенная вычислительная платформа HybriLIT состоит из Учебно-тестового полигона и суперкомпьютера «ГОВОРУН», работающих в единой программно-информационной среде.

Учебно-тестовый полигон содержит вычислительные узлы с многоядерными процессорами Intel, графическими процессорами Nvidia и сопроцессорами Intel Xeon Phi (Подробные характеристики в разделе «Аппаратное обеспечение«).

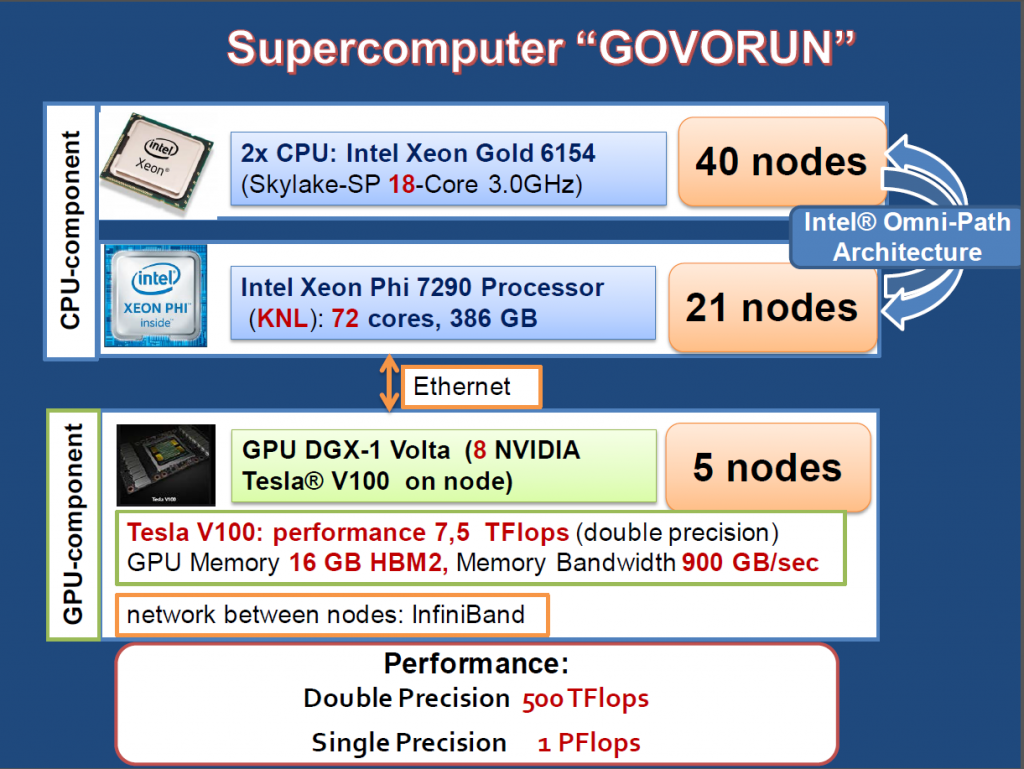
Типы основных вычислительных узлов:

* Узлы с многоядерными CPU и сопроцессорами Intel Xeon Phi
* Узлы с многоядерными CPU и 3 графическими ускорителями Nvidia Tesla K40
* Узлы с многоядерными CPU и 2 (4) графическими ускорителями Nvidia Tesla K80
* Mix-узел с многоядерными CPU и сопроцессорами Intel Xeon Phi и графическим ускорителем Nvidia Tesla K20



Гетерогенный кластер находится под управлением операционной системы Scientific Linux 7.5 , с планировщиком задач SLURM и установленным программным обеспечением: компиляторами и пакетами для разработки, отладки и профилировки параллельных приложений, а также пакетом Modules.

Суперкомпьютер является естественным развитием гетерогенной платформы HybriLIT и приведет к существенному увеличению производительности как CPU, так и GPU-компонент платформы. Модернизированный вычислительный кластер позволит проводить ресурсоемкие, массивно-параллельные расчеты в решеточной квантовой хромодинамике для исследования свойств адронной материи при высокой плотности энергии и барионного заряда и в присутствии сверхсильных электромагнитных полей, качественно повысит оперативность моделирования динамики столкновений релятивистских тяжелых ионов, откроет новые возможности для исследования свойств сильно-коррелированных систем в области физики новых материалов, а также позволит разрабатывать и адаптировать программное обеспечение для мега-проекта NICA на новые вычислительные архитектуры от основных лидеров рынка HPC – Intel и NVIDIA, создать программно-аппаратную среду на базе HPC и готовить IT-специалистов по всем необходимым направлениям.



**Результат работы суперкомпьютера «ГОВОРУН»**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Потоки** | **100** | **1000** | **10000** | **100000** | **1000000** |
| **1** | 6,25E-06 | 5,84E-05 | 0,00058239 | 0,00580604 | 0,0581315 |
| **2** | 4,94E-06 | 3,11E-05 | 0,0002939 | 0,00290743 | 0,0290957 |
| **3** | 4,22E-06 | 2,21E-05 | 0,0002043 | 0,0020032 | 0,0200292 |
| **4** | 3,73E-06 | 2,03E-05 | 0,00015411 | 0,00150453 | 0,0152997 |
| **5** | 3,66E-06 | 1,49E-05 | 0,0001263 | 0,00124091 | 0,0124297 |
| **6** | 3,93E-06 | 1,33E-05 | 0,00010829 | 0,0010372 | 0,0106354 |
| **7** | 4,01E-06 | 1,20E-05 | 9,46E-05 | 0,00091703 | 0,00917921 |
| **8** | 3,92E-06 | 1,12E-05 | 8,53E-05 | 0,00080544 | 0,00825126 |

**Рис. 7.** Результаты замеров времени (c) вычисления при варьировании числа потоков (от 1 до 8), числа ядер и гранулярности задачи на

**Рис. 8.** Графики коэффициентов ускорения при варьировании числа потоков

# Приложение 2.

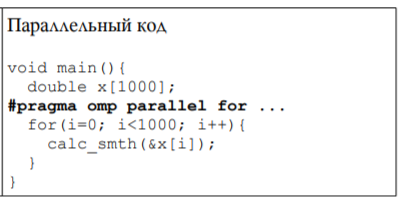
OpenMP – механизм написания параллельных программ для систем с общей памятью. Состоит из набора директив компилятора и библиотечных функций. Позволяет достаточно легко создавать многопоточные приложения на С/С++, Fortran. Поддерживается производителями аппаратуры (Intel, HP, SGI, Sun, IBM), разработчиками компиляторов (Intel, Microsoft, KAI, PGI, PSR, APR, Absoft).

**Программная модель OpenMP**

Основной поток порождает дочерние потоки по мере необходимости.

Программирование путем вставки директив компилятора в ключевые места исходного кода программы.

Компилятор интерпретирует эти директивы и вставляет в соответствующие места программы библиотечные вызовы для расспараллеливания участков кода.



Директива #pragma omp parallel for указывает на то, что данный цикл следует разделить по итерациям между потоками.

Количество потоков можно контролировать из программы, или через среду выполнения программы- переменную окружения OMP\_NUM\_THREADS.

**Синтаксис**

В основном конструкции OpenMP – это директивы компилятора

Для С/С++ директивы имеют следующий вид:

#pragma omp конструкция [условие [условие]...]

# Приложение 3.

Время рассчитывалась с помощью функцией для работы с системным таймером OpenMP. Для разработки эффективных параллельных программ важно знать время выполнения того или иного фрагмента программ или всей программы в целом. Поскольку на каждой вычислительной системе функции таймеров сильно отличаются друг от друга, то OpenMP предлагает свои платформенно-независимые функции.

OMP\_GET\_WTIME - функция для работы с системным таймером.

**C/C++:**

#include <omp.h>

double omp\_get\_wtime(void)

Функция возвращает в виде переменной двойной точности разрешение таймера в секундах.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Потоки** | **100** | **1000** | **10000** | **100000** | **1000000** |
| **1** | 7,55E-06 | 7,48E-05 | 0,000749877 | 0,00754483 | 0,0757198 |
| **2** | 4,15E-06 | 3,81E-05 | 0,000375694 | 0,00376864 | 0,0378473 |
| **3** | 3,02E-06 | 2,68E-05 | 0,000262797 | 0,0026291 | 0,0262502 |
| **4** | 2,64E-06 | 2,04E-05 | 0,000197853 | 0,00197362 | 0,0199658 |
| **5** | 4,91E-06 | 3,32E-05 | 0,000317168 | 0,00312713 | 0,0315262 |
| **6** | 5,29E-06 | 2,87E-05 | 0,00026544 | 0,00263212 | 0,0263918 |
| **7** | 4,91E-06 | 2,53E-05 | 0,000228437 | 0,00226247 | 0,0228301 |
| **8** | 4,53E-06 | 2,23E-05 | 0,000199741 | 0,00197664 | 0,0199763 |

**Рис. 9.** Результаты замеров времени (c) вычисления при варьировании числа потоков (от 1 до 8), числа ядер и гранулярности задачи

**Рис. 10.** Графики коэффициентов ускорения при варьировании числа потоков