Федеральное государственное автономное образовательное учреждение профессионального образования   
«Нижегородский государственный университет им. Н.И.Лобачевского»

Институт информационных технологий математики и механики

Отчет по лабораторной работе №1

«Вычисление многомерных интегралов с использованием многошаговой схемы (метод прямоугольников)»

**Выполнил:**

Студент ИИТММ

группы 381506-3

Бунтова К.

**Проверил:**

Кандидат технических наук,

доцент каф. МОСТ, ИИТММ

Сысоев А.В.

Нижний Новгород

2018

**Содержание**

Содержание…………………………………………………………………………………..2

1)Постановка задачи…………………………………………………………………………3

2)Метод решения………………………………………………………………………….…4

3)Схема распараллеливания…………………………………………………………….…..6

5)Подтверждение корректности………………………………………………………….…7

6)Результаты экспериментов по оценке масштабируемости……………………………..8

Литература……………………………………………………………………………………9

**1) Постановка задачи**

В этом проекте необходимо создать несколько программ:

1. Solver – основа проекта. Вычисляет многомерный интеграл с использованием многошаговой системы, данные задающие параметры для вычисления (подынтегральная функция, концы отрезков интегрирования) должны считываться из бинарного файла. Вычисление должно быть реализовано последовательно, с использованием методов распараллеливания OpenMP и с использованием методов распараллеливания TBB. Полученный результат (число) и время выполнения реализации записываются в бинарный файл.
2. Generator – создает бинарный файл содержащий данные (параметры вычисляемого интеграла в Solver), создаваемые рандомно и записываемые в бинарный файл.
3. Checker – считывает данные из бинарного файла результатов (полученного из Solver) и исходного файла с параметрами интеграла (полученного из Generator), проверяет их корректность (сравнивает результаты последовательной и параллельной версий относительно вычисляемой в программе погрешности по данным исходного файла) и записывает вердикт (результаты корректны результаты некорректны, ошибка входных данных). Также вычисляется ускорение параллельной версии относительно последовательной.

Метод прямоугольников (одномерный случай):

(отрезок [a,b] разбивается на n меньших отрезков [xi , xi+1), для метода средних прямоугольников (который планируется использовать) xi – середины этих отрезков)

Метод прямоугольников (многомерный случай):

**2) Метод решения**

Возможна была рекурсивная реализация, однако, из-за необходимости передачи в параметрах рекурсии большого количества данных, стек быстро переполняется, что не дает возможности программе работать на больших данных (для компьютера, на котором проводились тесты, не выше двойного интеграла), хотя и является более быстрой реализацией. В связи с этим была необходимость переписать рекурсию в виде циклов for.

По формуле:

можно заметить, что нам нужно найти сумму значений функции при всех возможных сочетаниях значений переменных, то есть всего итераций будет

nkol-1 , где n – число отрезков, на которые разбиваются отрезки пределов интегрирования (для упрощения все отрезки разбиваются на одинаковое количество подотрезков), kol – количество извлекаемых интегралов (или кол-во переменных в функции).

Для получения всевозможных сочетаний значений переменных будет использоваться следующий алгоритм:

//с[]-массив в котором перебираем середины отрезков

//a[]-массив начал отрезков интегрирования

//b[]-массив концов отрезков интегрирования

for (int i = 0; i < pow(n, kol - 1); i++)

{

for (int j = 0; j < n; j++)

{

c[kol - 1] += h[kol - 1]; //перебираются варианты последней переменной

}

for (int j = kol-1; j >= 0; j--)

{

if (c[j] > b[j]) //

{ //увеличиваем предыдущую переменную,

c[j - 1] += h[j - 1]; //если следующая для нее превысила свой

c[j] = a[j] + h[j] / 2; //максимум, а превысившую максимум меняем

} //на минимум

} //

}

Если в первый цикл по j добавить суммирование значений функции от вектора с, а так же после цикла i добавить умножение на произведение всех длин подотрезков(для каждого интеграла один множитель) то его результатом будет подсчитанное значение определенного многомерного интеграла.

В Generator будет задаваться количество интегралов и будут создаваться рандомно массивы (размером соответствующем кол-ву интегралов), которые потом можно будет расшифровать:

1. Func[] – значения в диапазоне 1..12, каждому числу соответствующая функция при чтении: 1 – ln, 2 – cos, 3 – sin, 4 – exp, 5 – x2, 6 – x3, 7 – x4, 8 – x5, 9 – x6,

10 – , 11 – , 12 –

1. Zn[] – значения в диапазоне 1..3, при чтении: 1 – «+», 2 – «-», 3 – «\*», 4 – «/»
2. a[] – нижние границы интегрирования
3. b[] – нижние границы интегрирования

Для интерпретации этих данных в основной части проекта будет использована функция

double func(double\* x, int \*c, int \*z, int k) //x[i]-значение переменной, с[i]-номер математической функции, z[i]-знак, с которой математическая функция вошла в подынтегральную функцию.

В Checker погрешность будет высчитываться по следующей формуле:

**3) Схема распараллеливания**

Сложность распараллеливания заключается в реализации изменения вектора с. Наиболее оптимальным вариантом было из суммы

вынести внешнюю с изменением 0-ой переменной, а внутреннюю часть оставить реализованной как прежде только для меньшего числа переменных, т.е. количество итераций внутреннего цикла nkol-2 , где n – число отрезков, на которые разбиваются отрезки пределов интегрирования, kol – количество извлекаемых интегралов (или кол-во переменных в функции).

Для этого заведем отдельную функцию iter с параметрами как у solver + num – параметр определяющий номер итерации внешнего цикла и соответственно какому значению с[0] нужно обращаться. Также изменением относительно solver будет дополнительная проверка на количество интегралов: если kol=1, то

if (kol == 1)

{

c[0] = a[0] + h[0] / 2 + num \* h[0];

double m = func(c, f, z, kol);

res += m;

return res;

}

Т.к. при этом внутренний цикл будет с 0 итераций (в него не будет захода), то вычисление возвращаемого результата можно обеспечить таким образом.

Внешний цикл, который будет подвергнут распараллеливанию при это будет выглядеть подобным образом:

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

{

double \*c\_pr = new double[kol];

for (int i = 0; i < kol; i++)

{

c\_pr[i] = c[i];

}

res += iter(a, b, h, c\_pr, f, z, n, kol, i);

delete[]c\_pr;

}

Копирование массива с для каждой итерации в массив c\_pr обязательно из-за необходимости обнуления этого массива после выполнения функции iter.

OpenMP:

#pragma omp parallel for reduction(+:res) for(. . .){. . .}

reduction объединяет суммирование res со всех потоков, чтобы получить сумму по всем итерациям.

TBB:

parallel\_for(blocked\_range<int>(0, n, 1), [&](blocked\_range<int> r) {

for (int i = r.begin(); i < r.end(); i++)

{

double \*c\_pr = new double[kol];

for (int i = 0; i < kol; i++)

{

c\_pr[i] = c[i];

}

res += iter(a, b, h, c\_pr, f, z, n, kol, i);

delete[]c\_pr;

}}, simple\_partitioner());

**4) Подтверждение корректности**

Для подтверждения корректности последовательной версии будут вручную посчитаны несколько определенных многомерных интегралов, а затем результаты, полученные при ручном подсчете и при работе программы, будут сравнены. В методе реализовано деление на 50 отрезков.

Результат программы: 2.9523940

Результат программы: 1.7182532

Точность результатов можно увеличить за счет увеличения количества отрезков, однако, это даст значительный прирост затрат на память и на время.

Корректность работы параллельных версий проверяется в Checker.

**5) Результаты экспериментов по оценке масштабируемости**

Сравнение времени последовательного выполнения программы, параллельного с использованием технологий OpenMP и параллельного с использованием технологий TBB проводилось на пятимерных рандомных определенных интеграллах.

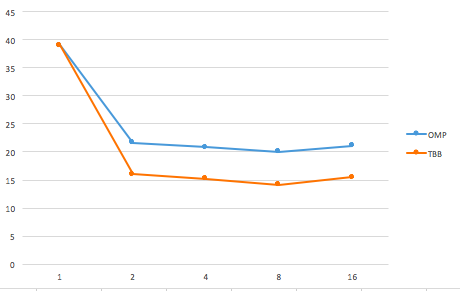


Рис. 1 График зависимости времени выполнения от количества потоков

(1 поток – последовательная реализация)

|  |  |
| --- | --- |
| TBB | OMP |
| 2 | 2,43 | 1,79 |
| 4 | 2,56 | 1,86 |
| 8 | 2,76 | 1,94 |
| 16 | 2,5 | 1,85 |

Таблица 1 Полученное ускорение на разных

количествах потоков

**Литература**

* Официальный сайт OpenMP – [www.openmp.org]
* Официальный сайт Intel® Threading Building Blocks. – [http://www.intel.com/software/products/tbb/]
* <http://www.cleverstudents.ru/integral/method_of_rectangles.html>
* <https://slemeshevsky.github.io/python-num-pde/term1/build/html/_computing-integrals/double-triple.html>