Задача

Разработать программу для задачи вычисления определенного интеграла с использованием метода прямоугольников.
Функция, для которой находится интеграл, – exp(x)
Интервал вычисления – [0,1]

Программа с комментариями

```
#include <iostream>
#include <cmath>
#include <omp.h>

double f(double x) {
   // функция, которую мы интегрируем
   return exp(x);
}

double integral(double a, double b, int n, int num_threads) {
   double h = (b - a) / n;  // шаг
   double sum = 0.0;
   omp_set_num_threads(num_threads); // устанавливаем количество потоков
   #pragma omp parallel for reduction(+:sum)
   for (int i = 0; i < n; ++i) {
      double x = a + (i + 0.5) * h;
      sum += f(x);
   }
   return sum * h;
}

double integral2(double a, double b, int n, int num_threads) {
   double h = (b - a) / n;  // шаг
   double sum = 0.0;
   double *sums;
   sums = new double[num_threads];  // суммы значений функции для каждого потока
   #pragma omp parallel
   {
      int tid = omp_get_thread_num();  // идентификатор потока
      for (int i = tid; i < n; i += num_threads) {
         double x = a + (i + 0.5) * h;
         sums[tid] += f(x);
      }
   }
   for (int i = 0; i < num_threads; ++i) {
      sum += sums[i];
   }
   delete []sums;
```

```
    return sum * h;
}


double integralSeq(double a, double b, int n) {
    double h = (b - a) / n;  // шаг
    double sum = 0.0;

    for (int i = 0; i < n; ++i) {
        double x = a + (i + 0.5) * h;
        sum += f(x);
    }
    return sum * h;
}

int main() {
    double a = 0.0;  // начальная точка
    double b = 1.0;  // конечная точка
    int n = 10000000;  // количество разбиений
    int num_threads = 8; // задаем количество потоков

    double start_time = omp_get_wtime(); // сохраняем текущее время

    double result = integral(a, b, n, num_threads);
    //double result = integralSeq(a, b, n);

    double end_time = omp_get_wtime(); // сохраняем текущее время
    std::cout << "Значение n = " << n << std::endl;
    //std::cout << "Количество потоков: " << num_threads << std::endl;
    std::cout << "Результат: " << result << std::endl;
    std::cout << "Время выполнения: " << end_time - start_time << " секунд" << std::endl;

    return 0;
}
```

Результаты полученные при разных N

N = 100000

```
Значение n = 100000
Количество потоков: 4
Результат: 1.71828
Время выполнения: 0.000853062 секунд
```

```
Значение n = 100000
Количество потоков: 8
Результат: 1.71828
Время выполнения: 0.00104904 секунд
```

```
Значение n = 100000
Количество потоков: 2
Результат: 1.71828
Время выполнения: 0.00148296 секунд
```

```
Значение n = 100000
Количество потоков: 6
Результат: 1.71828
Время выполнения: 0.00091815 секунд
```

```
Значение n = 100000
Количество потоков: 5
Результат: 1.71828
Время выполнения: 0.000701189 секунд
```

Sequentially

```
Значение n = 100000
Результат: 1.71828
Время выполнения: 0.00289011 секунд
Program ended with exit code: 0
```

N = 1000000

```
Значение n = 1000000
Количество потоков: 2
Результат: 1.71828
Время выполнения: 0.0133541 секунд
```

```
Значение n = 1000000
Количество потоков: 4
Результат: 1.71828
Время выполнения: 0.0106671 секунд
```

```
Значение n = 1000000
Количество потоков: 6
Результат: 1.71828
Время выполнения: 0.00493908 секунд
```

```
Значение n = 1000000
Количество потоков: 8
Результат: 1.71828
Время выполнения: 0.00358391 секунд
```

```
Значение n = 1000000
Количество потоков: 10
Результат: 1.71828
Время выполнения: 0.00651002 секунд
```

```
Значение n = 1000000
Количество потоков: 9
Результат: 1.71828
Время выполнения: 0.00601912 секунд
```

Sequentially

```
Значение n = 1000000
Результат: 1.71828
Время выполнения: 0.0274282 секунд
Program ended with exit code: 0
```

N = 10000000

```
Значение n = 10000000
Количество потоков: 2
Результат: 1.71828
Время выполнения: 0.129014 секунд
```

```
Значение n = 10000000
Количество потоков: 4
Результат: 1.71828
Время выполнения: 0.066288 секунд
```

```
Значение n = 10000000
Количество потоков: 6
Результат: 1.71828
Время выполнения: 0.0438309 секунд
```

```
Значение n = 10000000
Количество потоков: 8
Результат: 1.71828
Время выполнения: 0.037549 секунд
```

```
Значение n = 10000000
Количество потоков: 10
Результат: 1.71828
Время выполнения: 0.0385549 секунд
```

```
Значение n = 10000000
Количество потоков: 9
Результат: 1.71828
Время выполнения: 0.0398161 секунд
```

Sequentially

```
Значение n = 10000000
Результат: 1.71828
Время выполнения: 0.28343 секунд
Program ended with exit code: 0
```

Вывод:
Быстродействие программы при использовании OpenMP может зависеть от двух факторов: числа потоков и числа выполняемых операций.

Число потоков определяет, сколько нитей будет выполнять вычисления параллельно. Если число потоков меньше числа доступных ядер процессора, то некоторые ядра будут простаивать и параллельная обработка будет неэффективной. С другой стороны, если число потоков больше числа доступных ядер, то многопоточность будет эффективна только в том случае, если задача имеет достаточно много операций, чтобы занять все ядра.

Число выполняемых операций также влияет на эффективность параллельной обработки. Если задача имеет мало операций, то выигрыш от параллельной обработки может быть незначительным или даже нулевым, из-за накладных расходов на создание и управление потоками.

Таким образом, эффективность параллельной обработки с использованием OpenMP зависит от баланса между числом потоков и числом выполняемых операций. Важно проводить эксперименты и тестирование с разными значениями числа потоков, чтобы определить оптимальную конфигурацию для каждой конкретной задачи.

Полученные результаты показывают, что при 100.000 итераций, лучше использовать 5 потоков, при 1.000.000 итераций – 8 потоков, для 10.000.000 итераций результаты для 8,9,10 потоков были приблизительны равны с погрешностью в 0.001. Использование последовательного кода проигрывает во всех случаях.