Кафедра Параллельных вычислительных технологий

Лабораторная работа №1
по дисциплине «Архитектура ЭВМ и ВС»

Цель работы

Понять, как влияет оптимизация исходного кода на скорость выполнения программы, исследовать методы оптимизации.

План работы

1. Написать на Си или Си++ программу умножения двух квадратных матриц.

2. Проверить правильность работы программы на нескольких тестовых наборах входных данных.

3. Измерить время работы подпрограммы умножения матриц для матриц различных размеров. Минимальный размер матриц выбирается такой, при котором время счета приблизительно равно 0.1-0.5 сек. Максимальный размер матриц выбирается так, чтобы время счета было приблизительно равно 30-50 сек.

4. Повторить п. 3 для двух-трех различных уровней оптимизации компилятора. По п.п. 3-4 построить графики зависимости времени счета от размера матриц.

5. Модифицировать программу из п. 1 следующим образом. Матрицу, обход которой при умножении происходит по столбцам, необходимо транспонировать до умножения. Т.е. при умножении элементы обеих матриц будут считываться последовательно, по строкам. Для модифицированной программы выполнить п.п. 2-4.

6. Оформить отчет о проделанной работе.

Исходный код

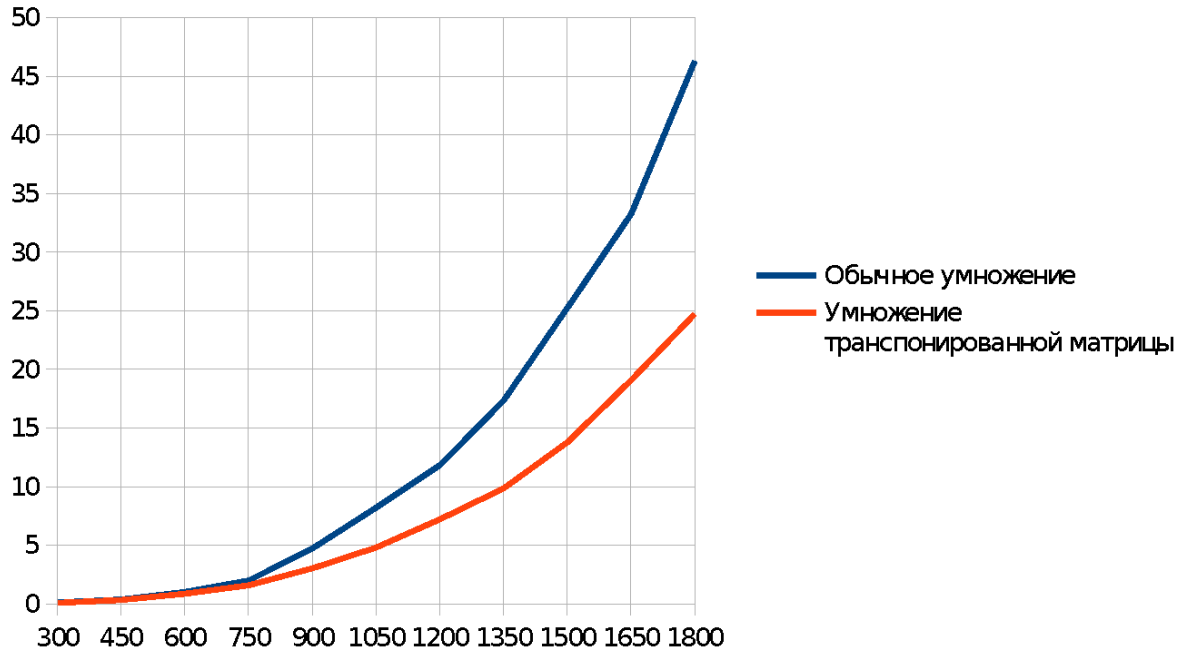
```
#include <iostream>
#include <ctime>
#include <stdlib.h>

void first_multiply(int n);
void second_multiply(int n);
void transponation(int n);

const int M_SIZE = 10000;

int matrix1[M_SIZE][M_SIZE];
int matrix2[M_SIZE][M_SIZE];
int matrix3[M_SIZE][M_SIZE];

int main()
{
  struct timespec cl_start, cl_end;
  long int runtime_sec;
  long int runtime_ms;

  int n = 0;
  std::cin >> n;

  for(int row = 0; row < n; row++)
    for(int column = 0; column < n; column++)
      {
        matrix1[row][column] = rand() % 100;
        matrix2[row][column] = rand() % 100;
      }

  // Первое перемножение
  clock_gettime(CLOCK_REALTIME, &cl_start);
  first_multiply(n);
  clock_gettime(CLOCK_REALTIME, &cl_end);

  runtime_sec = cl_end.tv_sec - cl_start.tv_sec;
  runtime_ms = (cl_end.tv_nsec - cl_start.tv_nsec) / 1000000;
  if (runtime_ms < 0)
    {
      runtime_sec--;
      runtime_ms += 1000;
    }
  printf("Time 1: %ld,%0.3ld sec.\n", runtime_sec, runtime_ms);

  // Второе перемножение
  transponation(n);
  clock_gettime(CLOCK_REALTIME, &cl_start);
  second_multiply(n);
  clock_gettime(CLOCK_REALTIME, &cl_end);

  runtime_sec = cl_end.tv_sec - cl_start.tv_sec;
```

```
  runtime_ms = (cl_end.tv_nsec - cl_start.tv_nsec) / 1000000;
  if (runtime_ms < 0)
    {
      runtime_sec--;
      runtime_ms += 1000;
    }
  printf("Time 2: %ld,%0.3ld sec.\n", runtime_sec, runtime_ms);

  printf("Success\n");
  return 0;
}


void first_multiply(int n)
{
  for(int row = 0; row < n; row++)
    for(int col = 0; column < n; column++)
      for(int cnt = 0; cnt < n; cnt++)
        matrix3[row][column] += matrix1[row][cnt] * matrix2[cnt][column];
}


void second_multiply(int n)
{
  for(int row = 0; row < n; row++)
    for(int column = 0; column < n; column++)
      for(int cnt = 0; cnt < n; cnt++)
        matrix3[row][column] += matrix1[row][cnt] * matrix2[column][cnt];
}


void transponation(int n)
{
  int temp;

  for(int cnt = 0; cnt < n; cnt++)
    for(int row = cnt+1, column = cnt+1; row < n; row++, column++)
      {
        temp = matrix2[row][cnt];
        matrix2[row][cnt] = matrix2[cnt][column];
        matrix2[cnt][column] = temp;
      }
}
```

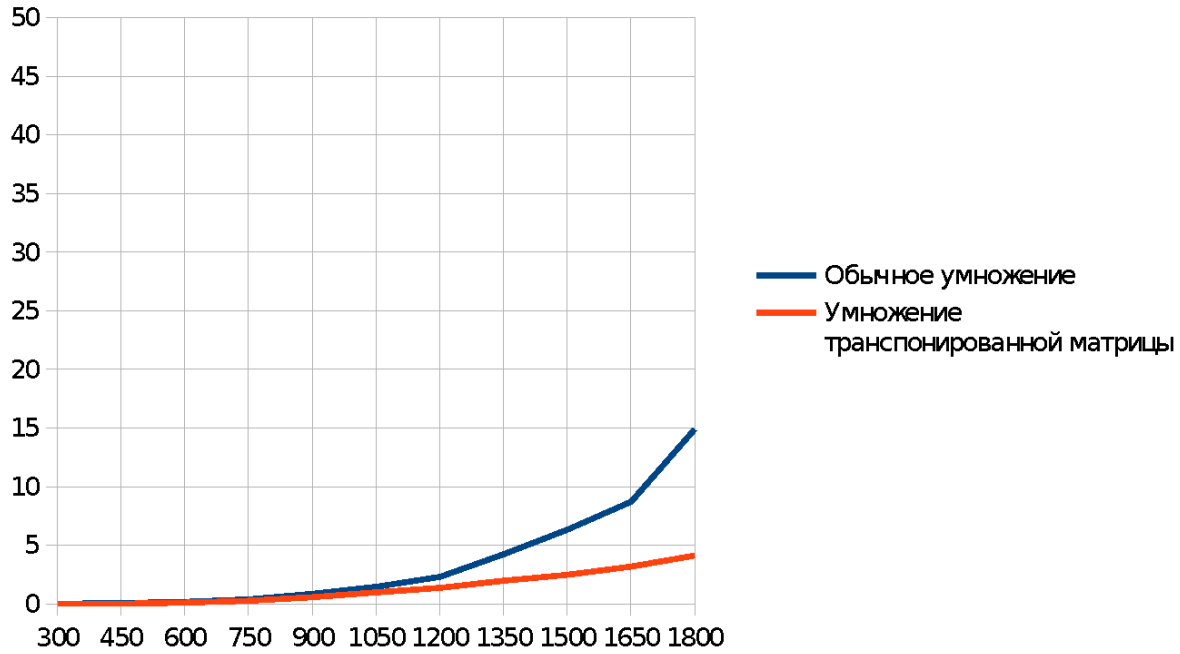
Графики зависимости времени счёта от размеров матриц при разном уровне оптимизации:

Ось *X* - *размерность матриц;*
Ось *Y* - *время, сек.*

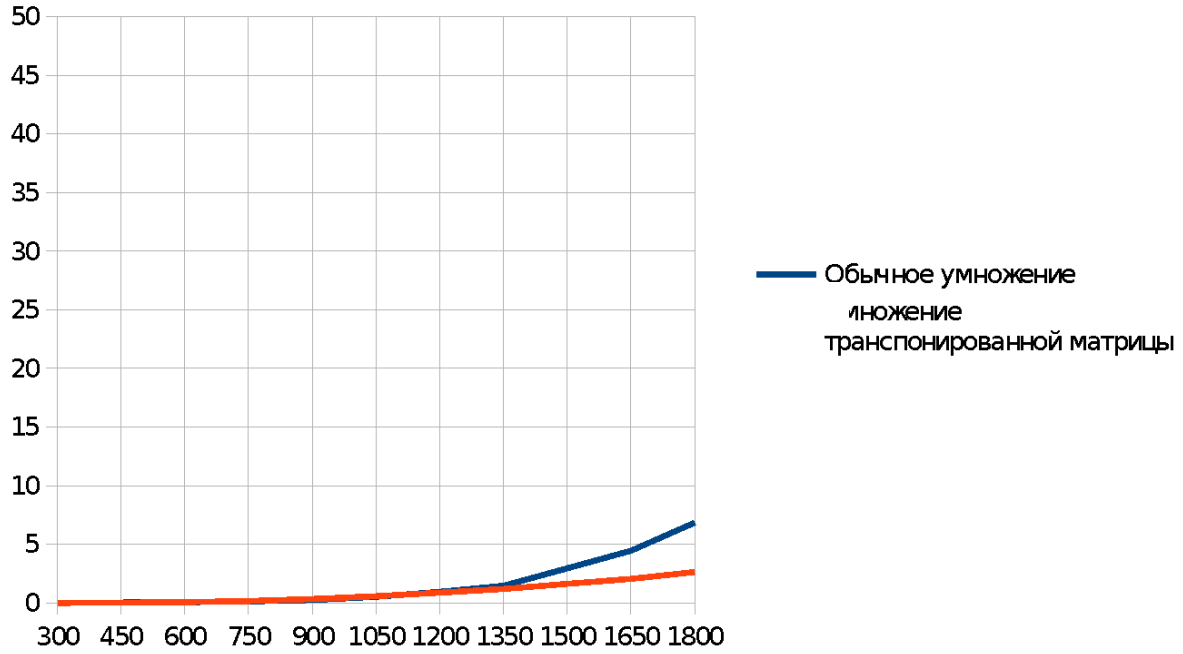
Компиляция без оптимизации



Компиляция с простой оптимизацией



Компиляция с высокоуровневой оптимизацией



Вывод

Вне зависимости от уровня оптимизации, процесс обычного умножения по времени более затратный, чем умножение заранее транспонированной матрицы. Однако это не так для маленьких размерностей матрицы при высокоуровневой оптимизации (умножение заранее транспонированной матрицы затратнее).

Важно понимать, как работает оптимизация. От этого зависит эффективность результата. Нельзя сказать, что оптимизированный алгоритм всегда идеален. В некоторых ситуациях использование для умножения заранее транспонированной матрицы дает более медленный результат (хотя выигрывает в перспективе).
Это даёт понимание того, что в разных ситуациях следует использовать разные алгоритмы.