САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Архитектура ЭВМ

Отчет

по домашней работе №2

«Сравнение наивной и кэш-эффективной реализации транспонирования матрицы»

Выполнил(а): Бородин Ярослав Алексеевич

студ. гр. М3137

Санкт-Петербург

2020

Цель работы: Сравнить время работы транспонирования матрицы двух реализаций на различных размерах входных данных.

Теоретическая часть

В вычислительной технике Cache-oblivious алгоритм - это алгоритм, предназначенный для использования преимуществ кэша процессора, без использования размера кэша, как определенного параметра. Оптимальный cache-oblivious алгоритм – это cache-oblivious алгоритм, который использует в асимптотическом смысле кэш оптимально, то есть игнорируя постоянные факторы.

Транспозиция матриц является фундаментальной операцией в линейной алгебре и быстрых преобразованиях Фурье и имеет прикладное значение в численном анализе, обработке изображений и графике.

Наивной реализацией транспонирования квадратной матрицы N × N в C++ может быть:

```
void naive() {
	for (int i = 0; i < MAX_N; i++) {
		for (int j = i + 1; j < MAX_M; j++) {
			a[j][i] = b[i][j];
			a[i][j] = b[j][i];
		}
	}
}
```

Асимптотика такого алгоритма – O(nm)

В наивной реализации алгоритма количество кэш промахов будет равно Θ(n*m). Нужно уменьшить количество кеш промахов для ускорения программы. В связи с этим можно использовать другой алгоритм, основывающийся на идее «Разделяй и властвуй». В такой реализации количество кеш промахов будет равно $O(\frac{nm}{B})$, где B - размер кеша в элементах, а асимптотика останется неизменным O(nm).

```
void transpose(int x, int delx, int y, int dely) {
	if ((delx == 1) && (dely == 1)) {
		b[y][x] = a[x][y];
		return;
	}
	if (delx >= dely) {
		int xmid = delx / 2;
		transpose(x, xmid, y, dely);
		transpose(x + xmid, delx - xmid, y, dely);
		return;
	}
	int ymid = dely / 2;
	transpose(x, delx, y, ymid);
	transpose(x, delx, y + ymid, dely - ymid);
}
```

Практическая часть

Время работы программ я замерял на квадратных матрицах N × N, где N от 1000 до 16200 (максимальный размер, который позволило устройство) с шагом 100. Ниже приведен сравнительный график:

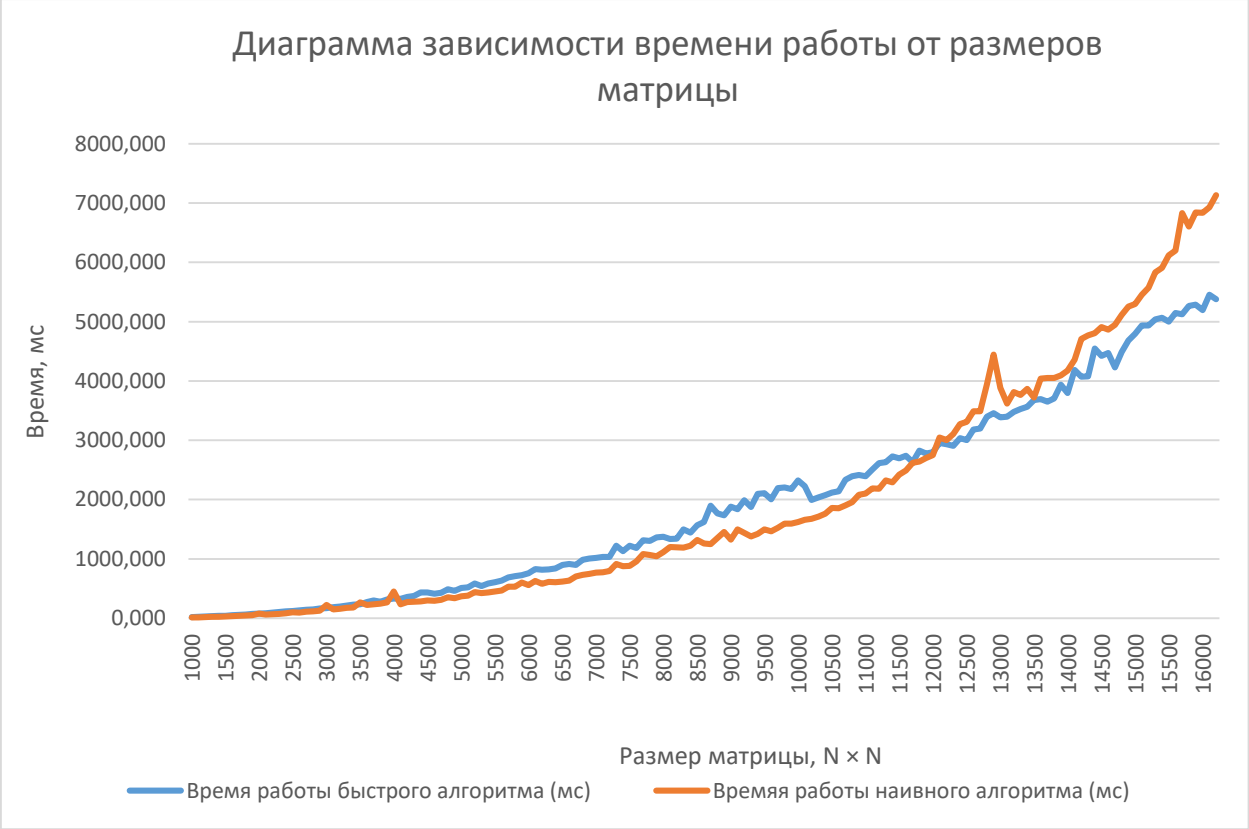

Рисунок 1 – Среднее время работы на матрице N × N

На графиках можно заметить, что при размере матрицы в районе 12000 × 12000 кеш независимый алгоритм начинает работать быстрее наивного, а на матрице размера 16200 × 16200 работает почти в полтора раза быстрее.

Все замеры были произведены встроенными в Code::Blocks средствами компилятор MinGW GCC C++11. В замер времени входит заполнение некоторой матрицы подходящего размера, используя некую исходную матрицу, созданную вне таймера. К отчету будет приложен файл с программой, таблица с результатами и файл с измерениями времени работы. Для подсчета среднего времени я использовал 6 измерений времени и брал среднее.