Национальный исследовательский ядерный университет «МИФИ»
Кафедра №42
«Криптология и кибербезопасность»

Лабораторная работа №3 «Реализация алгоритма с использованием технологии OpenMP».

Саменков Андрей
Б20-515

4 октября 2022

1 Описание используемой рабочей среды

1. Процессор: Ryzen 5 5500U 6 cores 12 threads
2. ОЗУ: 8gb DDR4
3. Система: Linux Fedora 36 Workstation x64
4. Компилятор: G++ (GCC) 12.2.1 20220819
5. Версия OpenMP: 4.5(201511)

2 Алгоритм

Общая идея сортировки Шелла состоит в сравнении на начальных стадиях сортировки пар значений, располагаемых достаточно далеко друг от друга в упорядочиваемом наборе данных. Сначала определяем шаг с которым проходим по массиву. Потом запускаем for по остаткам при делении на величину шага. Т.к. последовательности с разными остатками не пересекаются мы можем распаралеллить данный for. Далее мы в каждой подпоследовательнсоти выполняем сортировку вставками.

3 Блок схема

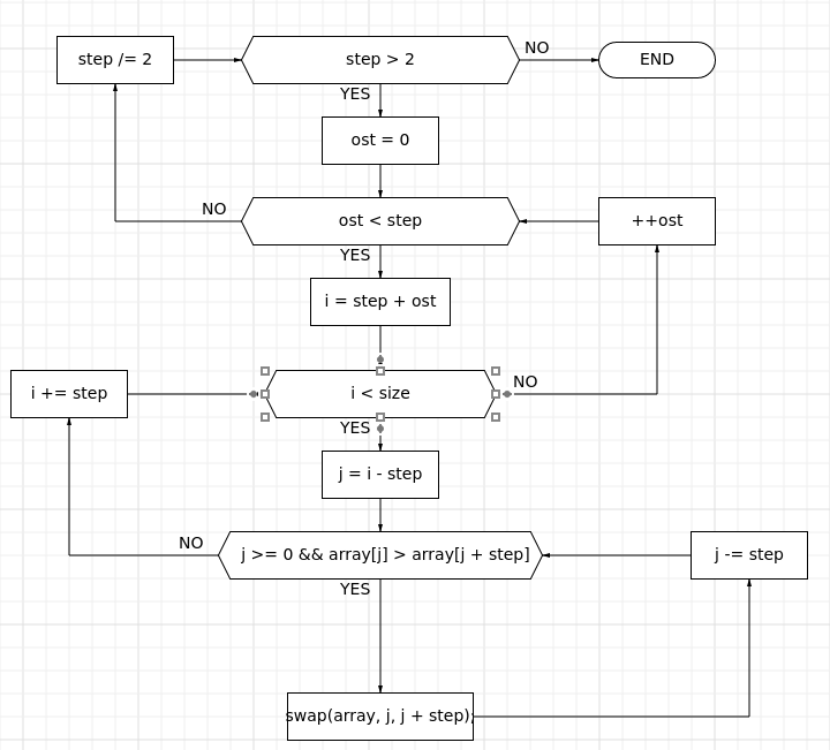

Рис. 1: Блок схема алгоритма.

4 OpenMP

$\#pragma\ omp\ parallel\ for\ num\_threads(threads)\ shared(a,\ m,\ n)\ private(i)\ default(none)$

1. *omp parallel* – показывает, что следующий блок кода будет выполняться с использованием нескольких потоков

2. $num\_threads(threads\_num)$ – задаёт количество потоков равное значению переменной threads

3. $shared(a,\ m,\ n)$ - указывает, что переменные a, m, n должны совместно использоваться для всех потоков. Если данный параметр не будет указана, то её аргументам будут присвоены правила по умолчанию (в данном случае они будут являться общими, так как все объявлены вне блока параллельных вычислений)

4. $private(i)$ - указывает что переменная i должна быть свою для каждого потока

5. $default(none)$ - требует, чтобы каждая переменная, на которую ссылаются в блоке параллельных вычислений и которая не имеет предопределенного атрибута совместного использования данных, имела свой атрибут, явно определенный ранее.

6. for – показывает, что следующий цикл for будет выполняться параллельно, несколькими потоками сразу.

5 Анализ эффективности и ускорения

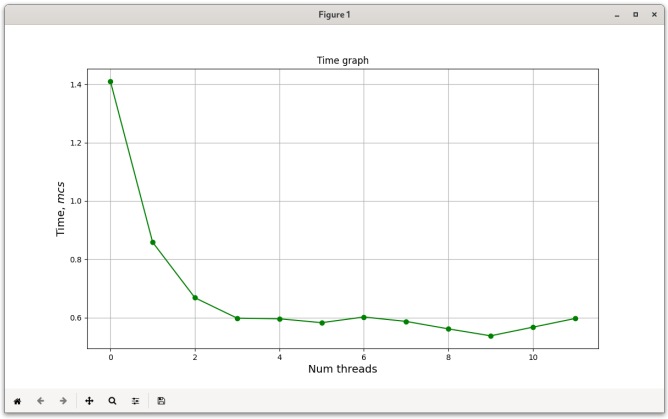

Рис. 2: График зависимости времени выполнения алгоритма от колличества потоков.

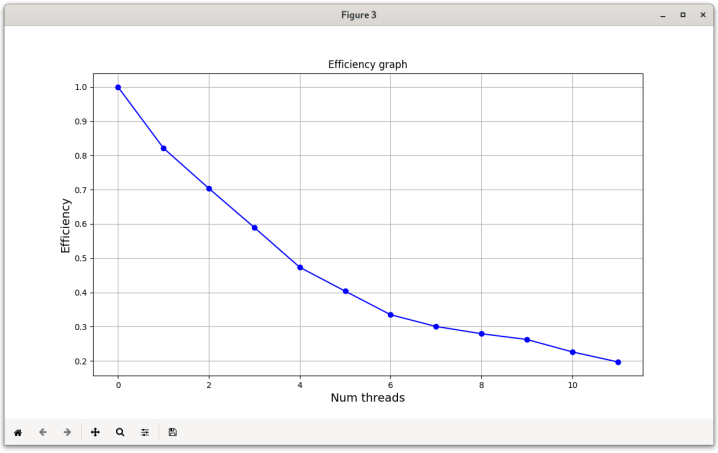

Рис. 3: График эффективности алгоритма от колличества потоков.

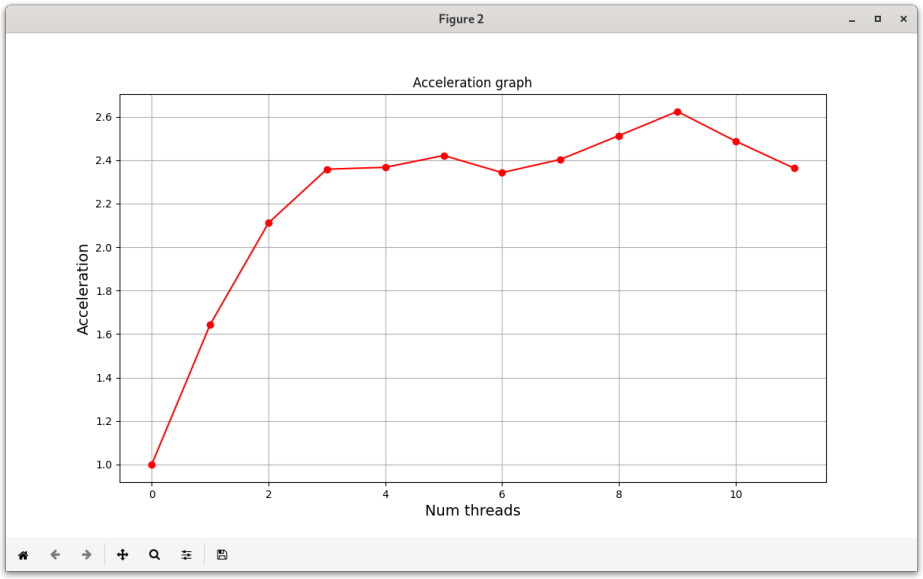

Рис. 4: График зависимости ускорения от колличества потоков.

6 Код программы

```
void insertionSort(int a[], int n, int stride) {
    for (int j = stride; j < n; j += stride) {
        int key = a[j];
        int i = j - stride;
        while (i >= 0 && a[i] > key) {
            a[i + stride] = a[i];
            i -= stride;
        }
        a[i + stride] = key;
    }
}

void shellSort(int a[], int n, int threads)
{
    int i, m;

    for(m = n/2; m > 0; m /= 2)
    {
            #pragma omp parallel for num_threads(threads) shared(a, m, n) private(i) default(none)
            for(i = 0; i < m; i++)
                insertionSort(&(a[i]), n-i, m);
    }
}
```

7 Заключение

В рамках данной работы был изучен приведённый алгоритм, оценена его вычислительная сложность в однопоточном варианте, а также была построена программа, выполняющая параллельные вычисления. Для успешного выполнения лабораторной работы был изучен и использован ряд директив OpenMP. По результатам работы программы были построены графики зависимости времени работы, ускорения и эффективности от числа потоков. По данным графикам видно, что параллельные вычисления дают большой выигрыш по времени в работе программы, при увеличении числа потоков до 4-х, а дальше ускорение не сильно растет, а иногда и падает.