



Факультет программной инженерии и компьютерной техники
Параллельные вычисления

Лабораторная работа №4
Метод доверительных интервалов при измерении времени
выполнения параллельной OpenMP-программы

Преподаватель: Жданов Андрей Дмитриевич
Выполнил: студент: Кульбако Артемий Юрьевич, Р4115

Санкт-Петербург
2023

ОГЛАВЛЕНИЕ

ОГЛАВЛЕНИЕ	2
ЗАДАНИЕ.....	3
ВЫПОЛНЕНИЕ.....	5
ИСХОДНЫЙ КОД	8

ЗАДАНИЕ

1. В программе, полученной в результате выполнения ЛР-3, так изменить этап Generate, чтобы генерируемый набор случайных чисел не зависел от количества потоков, выполняющих программу. Например, на каждой итерации i перед вызовом `rand_r` можно вызывать функцию ``srand(f(i))``, где f – произвольно выбранная функция. Можно придумать и использовать любой другой способ.
2. Заменить вызовы функции ``gettimeofday`` на ``omp_get_wtime``.
3. Распараллелить вычисления на этапе Sort, для чего выполнить сортировку в два этапа:
 - Отсортировать первую и вторую половину массива в двух независимых нитях (можно использовать OpenMP-директиву ``parallel sections``);
 - Объединить отсортированные половины в единый массив.
4. Написать функцию, которая один раз в секунду выводит в консоль сообщение о текущем проценте завершения работы программы. Указанную функцию необходимо запустить в отдельном потоке, параллельно работающем с основным вычислительным циклом.
5. Обеспечить прямую совместимость (forward compatibility) написанной параллельной программы. Для этого все вызываемые функции вида ``omp_*`` можно условно переопределить в препроцессорных директивах, например, так:

```
#ifdef _OPENMP
    #include "omp.h"
#else
    int omp_get_num_procs() { return 1; }
#endif
```

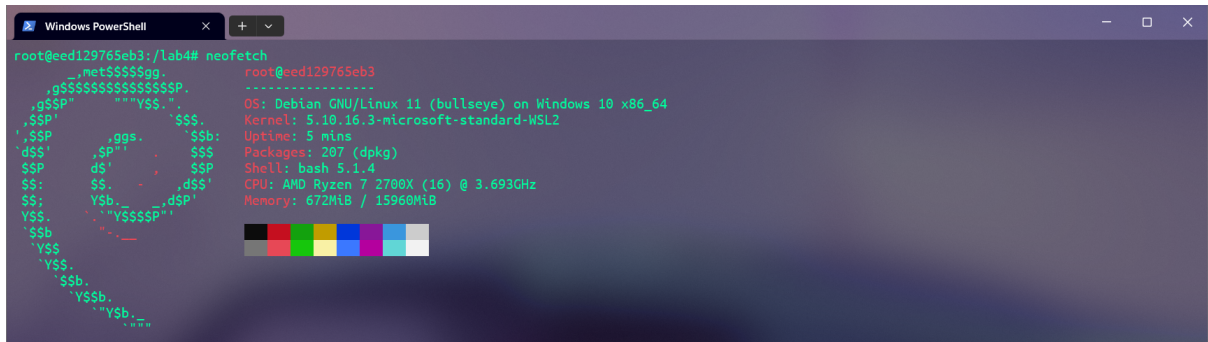
6. Провести эксперименты, варьируя N от $\min(N_x/2, N_1)$ до N_2 , где значения N_1 и N_2 взять из ЛР-1, а N_x – это такое значение N , при котором накладные расходы на распараллеливание превышают выигрыш от распараллеливания. Написать отчёт о проделанной работе. Подготовиться к устным вопросам на защите.
7. Необязательное задание на «четвёрку» и «пятёрку». Уменьшить количество итераций основного цикла с 100 до 10

и провести эксперименты, замеряя время выполнения следующими методами:

- Использование минимального из десяти полученных замеров;
- Расчёт по десяти измерениям доверительного интервала с уровнем доверия 95%. Привести графики параллельного ускорения для обоих методов в одной системе координат, при этом нижнюю и верхнюю границу доверительного интервала следует привести двумя независимыми графиками.

8. Необязательное задание на «пятёрку»: в п.3 задания на этапе Sort выполнить параллельную сортировку не двух частей массива, а k частей в k нитях (тредах), где k – это количество процессоров (ядер) в системе, которое становится известным только на этапе выполнения программы с помощью команды ``k = omp_get_num_procs()``.

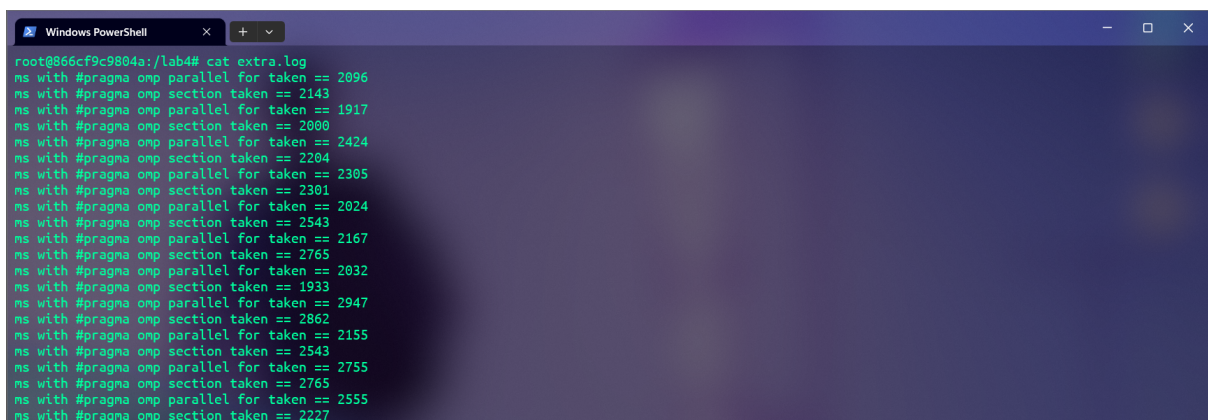
ВЫПОЛНЕНИЕ



```
Windows PowerShell
root@eed129765eb3:/lab4# neofetch
root@eed129765eb3
-----
OS: Debian GNU/Linux 11 (bullseye) on Windows 10 x86_64
Kernel: 5.10.16.3-microsoft-standard-WSL2
Uptime: 5 mins
Packages: 207 (dpkg)
Shell: bash 5.1.4
CPU: AMD Ryzen 7 2700X (16) @ 3.693GHz
Memory: 672MiB / 15960MiB
```

Рис. 1 - Характеристики машины

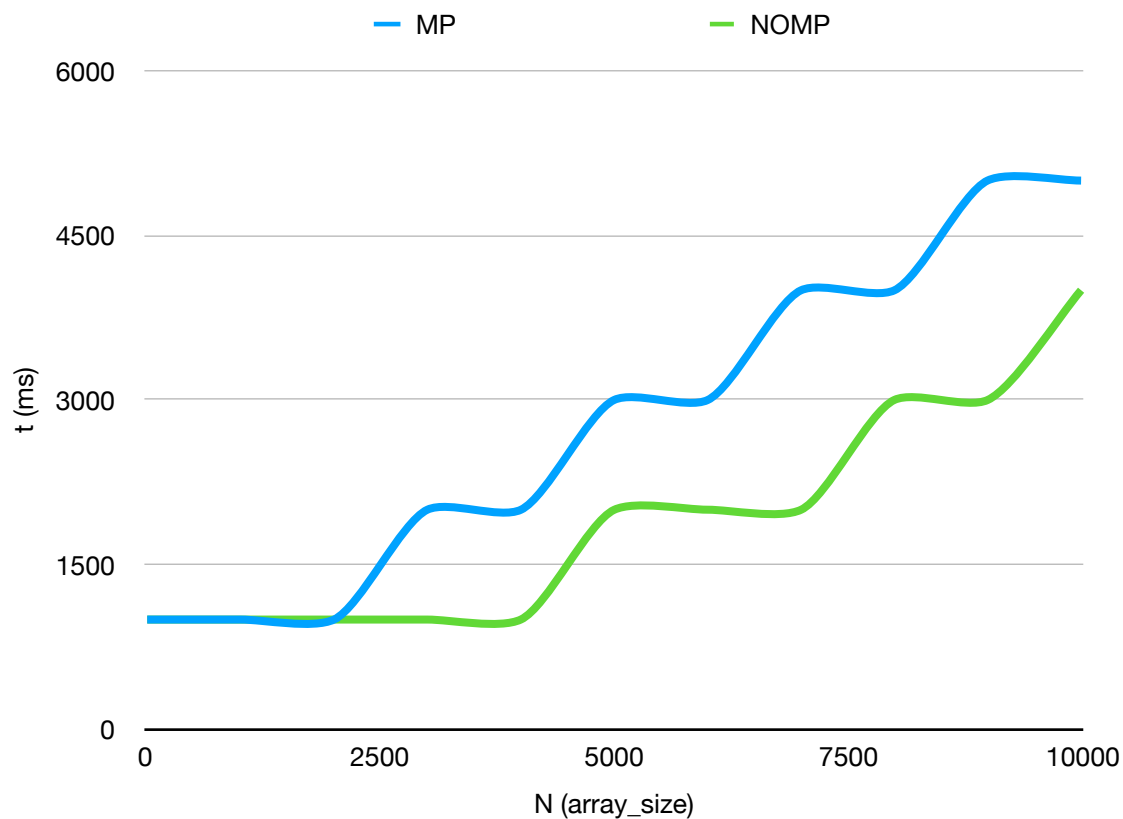
В первую очередь, я выполнил своё индивидуальное задание: проверил, создаёт ли лишнюю нагрузку использование директивы `pragma omp parallel for` на каждый цикл по сравнению с использованием механизма секций (`pragma omp section`). Исследование показало, что два подхода равнозначны: где-то победило первый вариант, где-то второй.



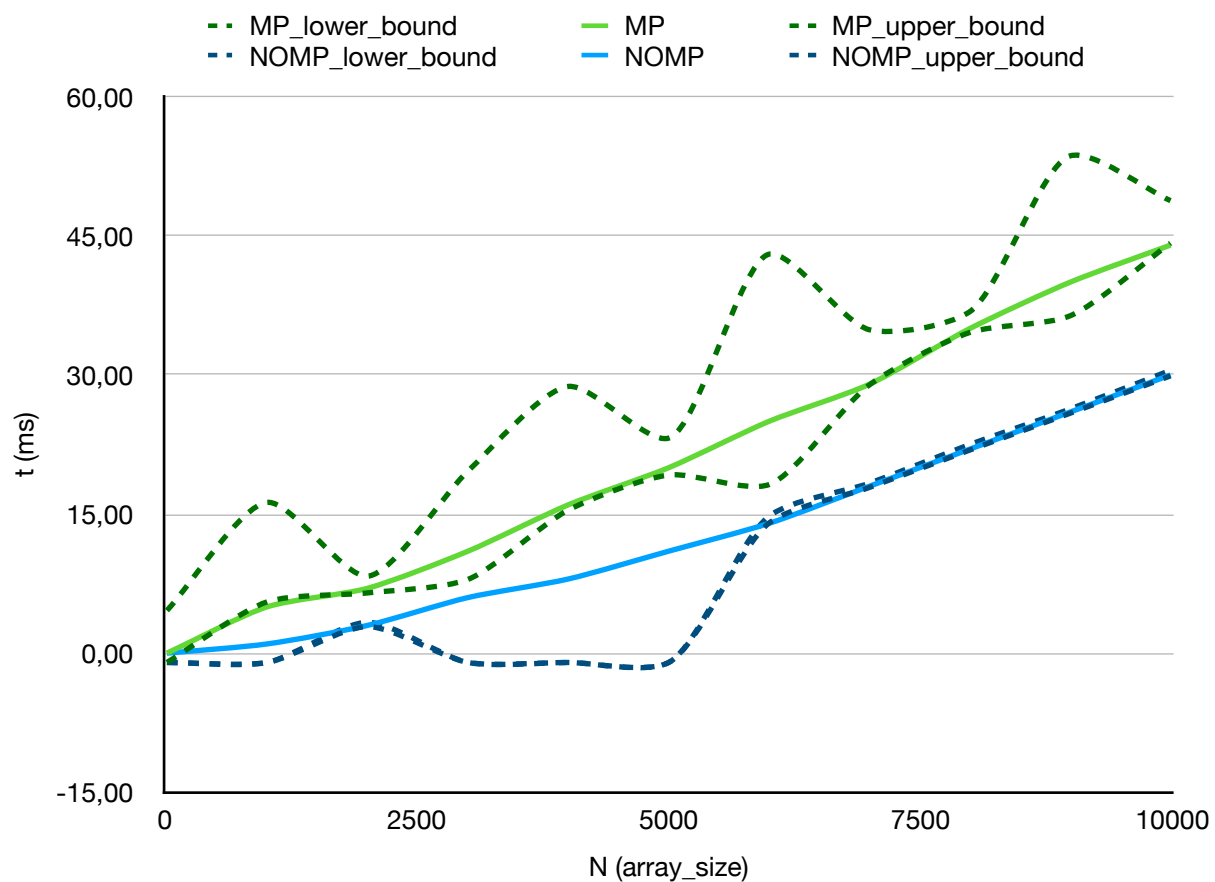
```
Windows PowerShell
root@8866cf9c9804a:/lab4# cat extra.log
ns with #pragma omp parallel for taken == 2096
ns with #pragma omp section taken == 2143
ns with #pragma omp parallel for taken == 1917
ns with #pragma omp section taken == 2000
ns with #pragma omp parallel for taken == 2424
ns with #pragma omp section taken == 2204
ns with #pragma omp parallel for taken == 2305
ns with #pragma omp section taken == 2301
ns with #pragma omp parallel for taken == 2024
ns with #pragma omp section taken == 2543
ns with #pragma omp parallel for taken == 2167
ns with #pragma omp section taken == 2765
ns with #pragma omp parallel for taken == 2032
ns with #pragma omp section taken == 1933
ns with #pragma omp parallel for taken == 2947
ns with #pragma omp section taken == 2862
ns with #pragma omp parallel for taken == 2155
ns with #pragma omp section taken == 2543
ns with #pragma omp parallel for taken == 2755
ns with #pragma omp section taken == 2765
ns with #pragma omp parallel for taken == 2555
ns with #pragma omp section taken == 2227
```

Рис. 2 - Сравнение директив `omp`

Теперь приступим к основному заданию:



На диаграмме можно видеть, влияние накладных расходов OpenMP на скорость выполнения программы. При малом количестве элементов массива, время выполнения с директивами OpenMP больше, чем без них. С увеличением количества элементов, разрыв сокращается, в конце концов, версия с OpenMP станет быстрее (на графике этот момент не отражён, т.к. по условия работы, время выполнения ограничено 10 сек, при таком сценарии обгон не происходит).



Такою же ситуация мы наблюдаем, если сократить число экспериментов с 100 до 10 и провести выборку по самому быстрому прогону (эксперименту). Пунктирной линией обозначены границы доверительных интервалов для MP и NOMP тестов.

ИСХОДНЫЙ КОД

Таблицы в формате csv и исходной код программы и скриптом для тестирования доступен на: <https://github.com/testpassword/Parallel-computing/tree/master/lab4-09.04.23>:

