**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики-процессов управления**

**Кафедра “фундаментальная информатика и информационные технологии”**

**ОТЧЕТ**

**по лабораторной работе**

**по дисциплине «Введение в суперкомпьютерные вычисления»**

**на тему:**

**«Open Multi-Processing»**

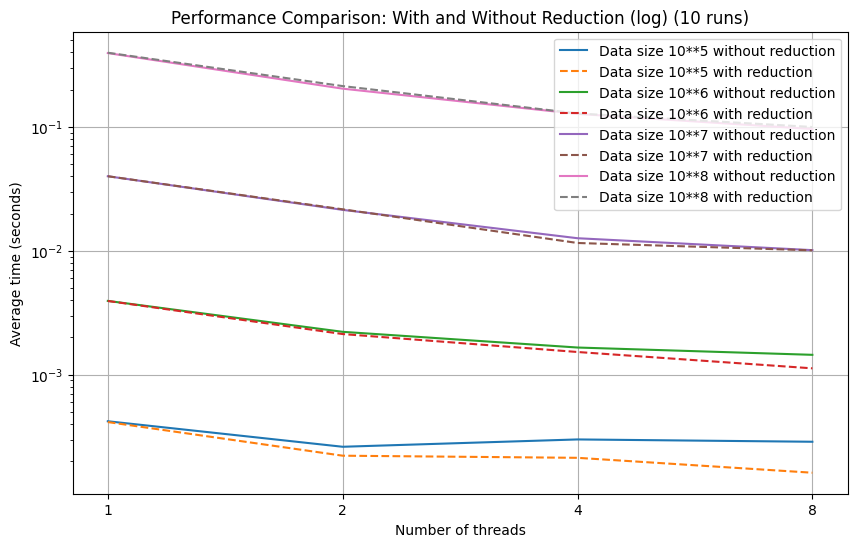
| Студент гр. 22Б15-пу |  | Глухов К.Д. |
| --- | --- | --- |
| Преподаватель |  | Корхов В.В. |

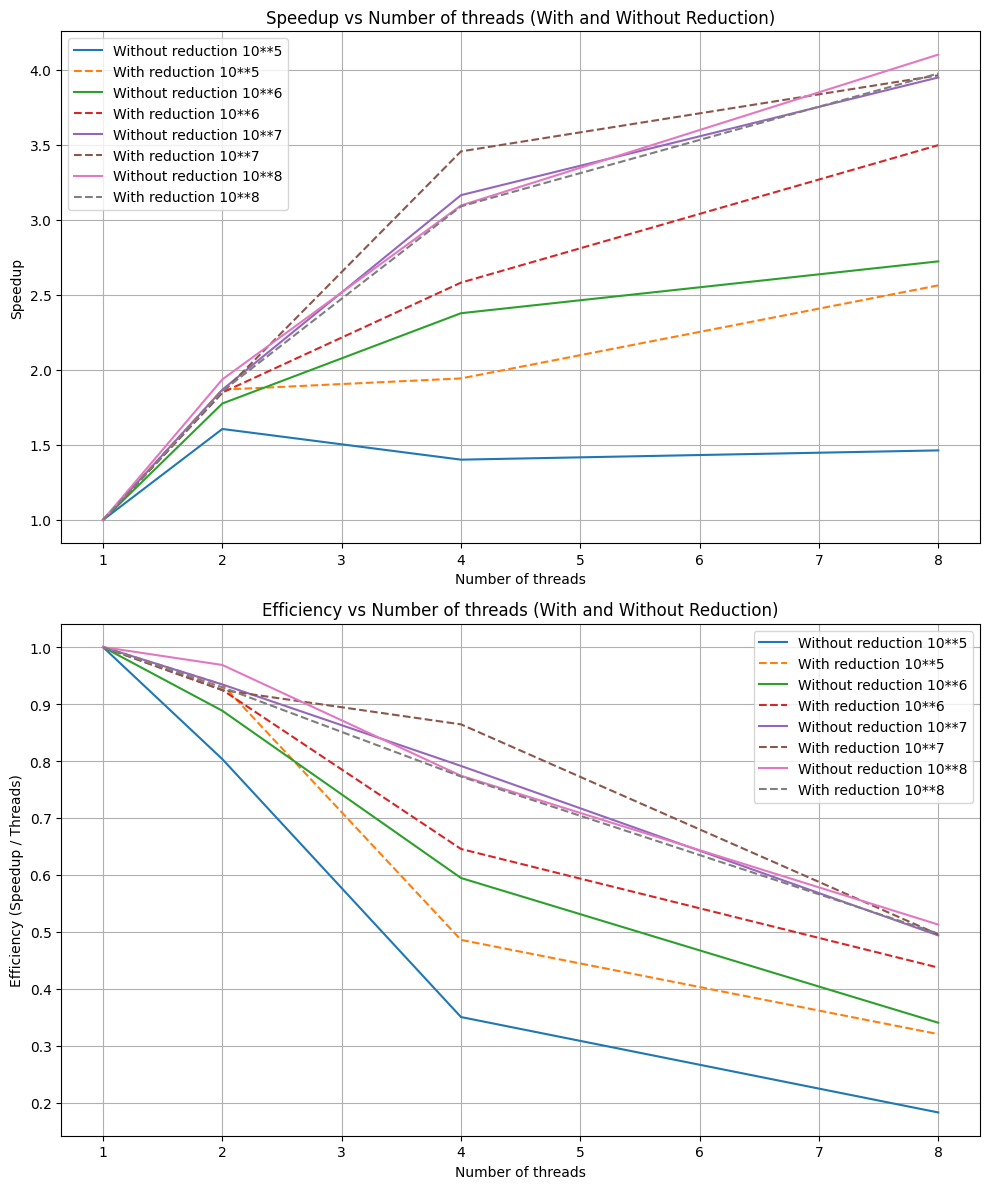
**Санкт-Петербург**

**2024 г**

# **Задание 1.**

Разработайте программу для нахождения минимального (максимального) значения среди элементов вектора. Рассмотреть вариант с использованием редукции (если используемая версия OpenMP поддерживает ее) и без редукции. (1 балл)



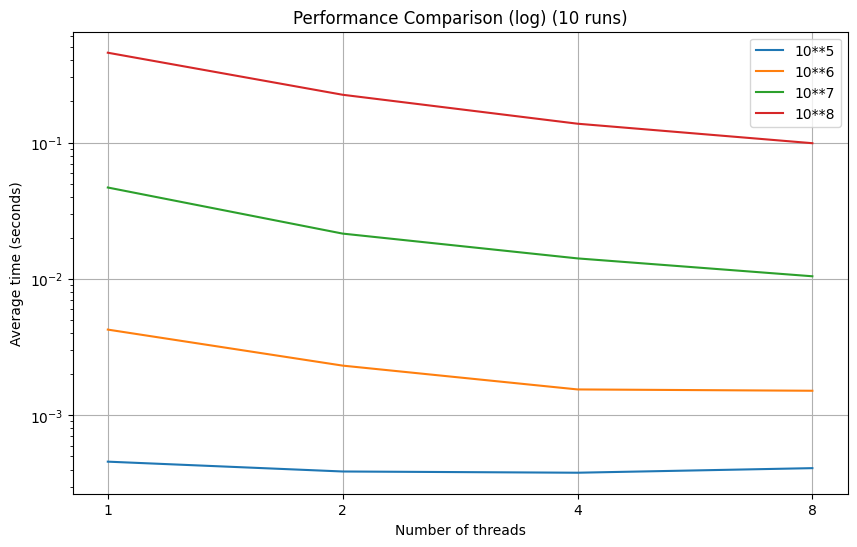


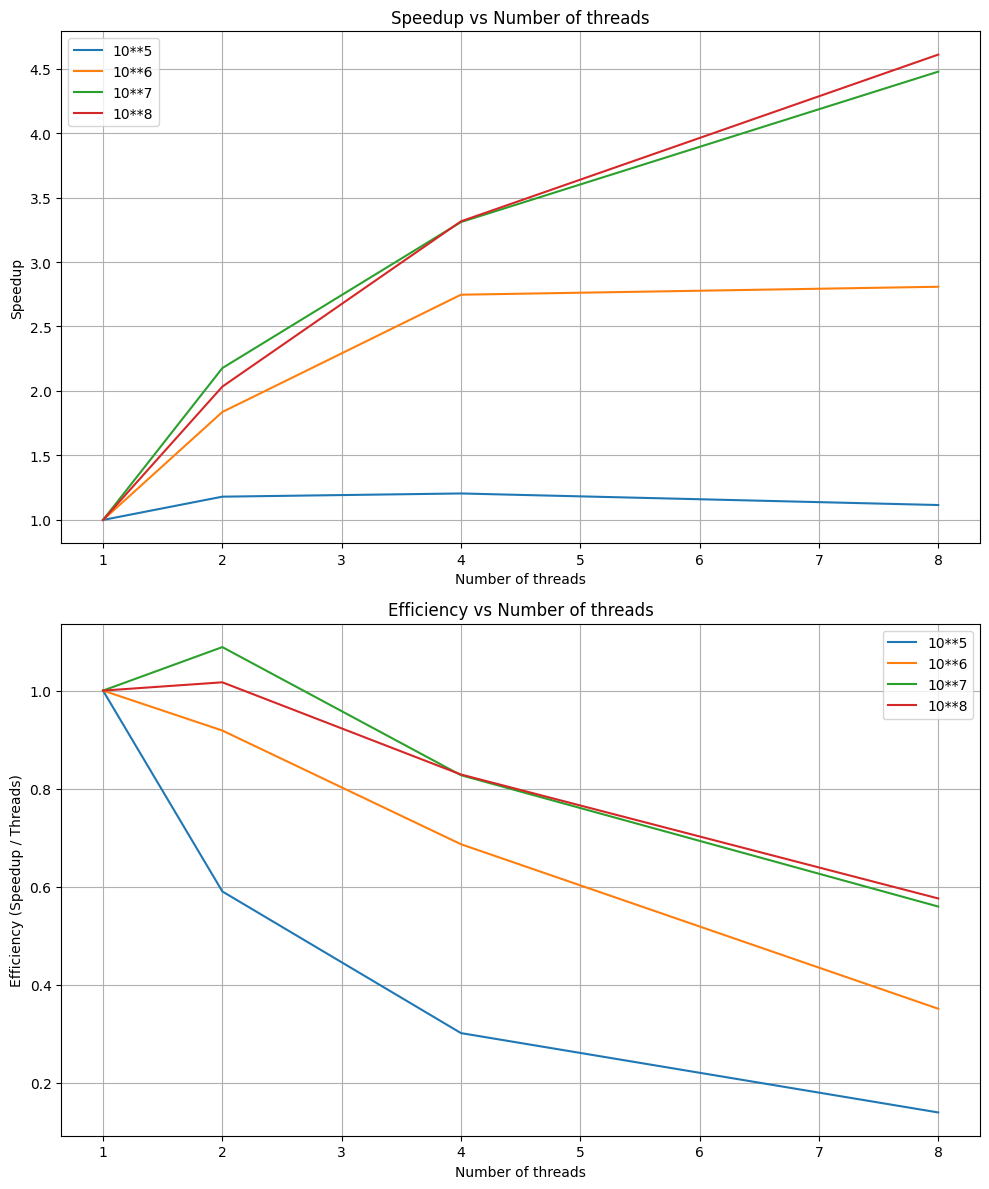
# **Вывод.**

Видно что во всех случаях программа использующая редукцию работает быстрее, самый большой прирост виден при малом объеме данных.

# **Задание 2.**

Разработайте программу для вычисления скалярного произведения двух векторов. (1 балл)





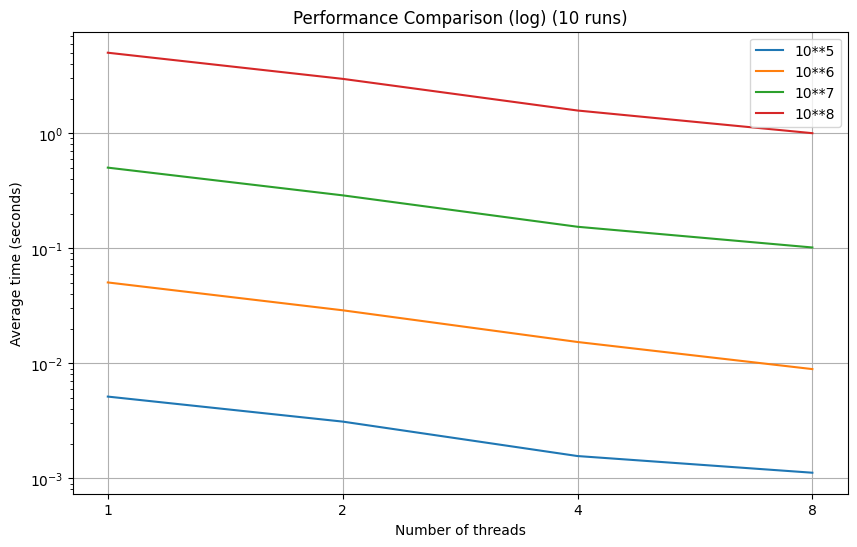
# **Вывод.**

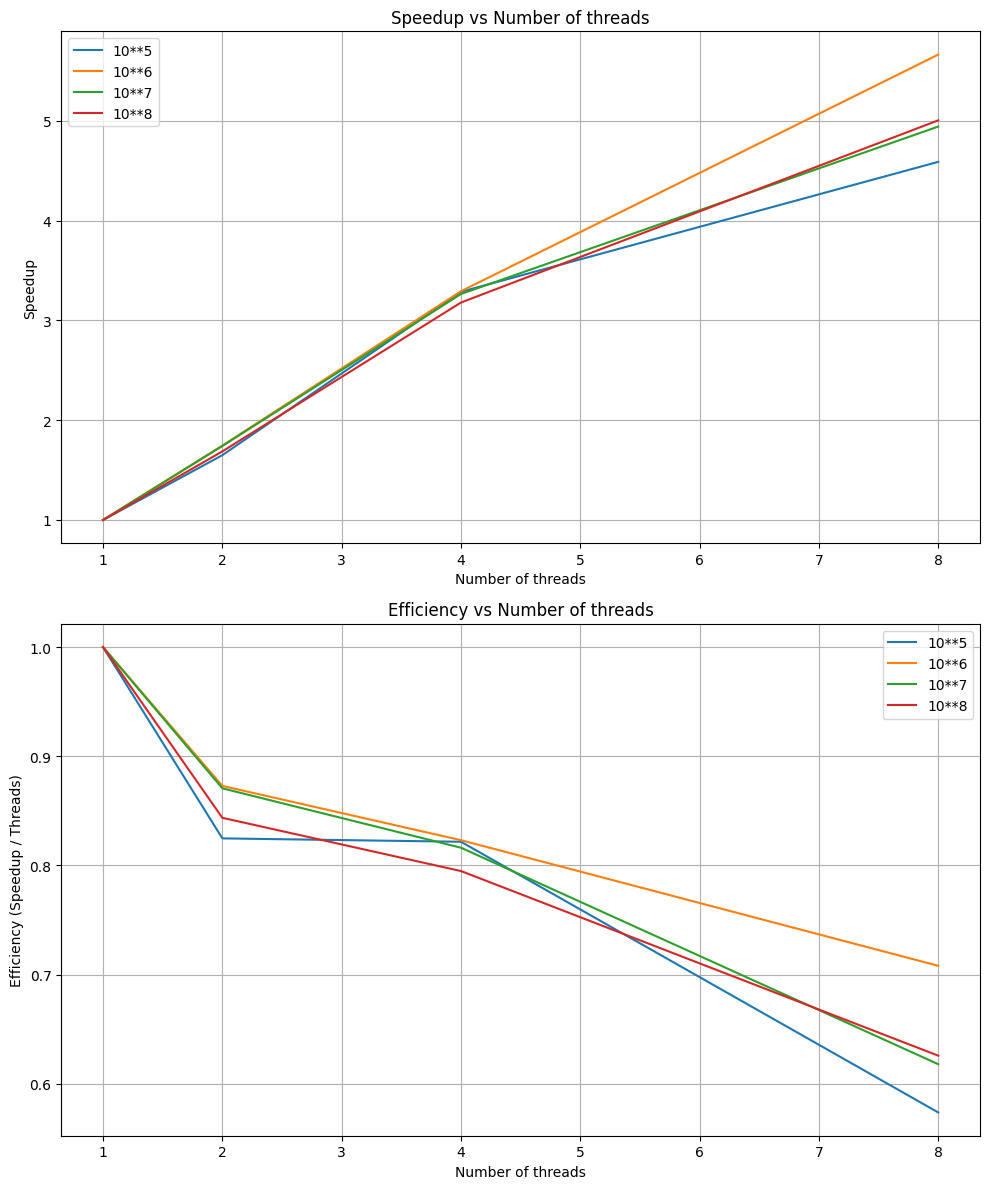
Видно что изменение количества процессоров в случае решения задачи с высокой степенью параллелизма приводит к почти линейному результату в случае больших объемов данных. Почти линейному, так как есть накладные расходы на передачу данных в силу малого количества процессоров, нет возможности проверить ситуацию, когда ускорение решения падает от количества процессоров.

**Задание 3.**

Разработайте программу для задачи вычисления определенного интеграла с использованием метода прямоугольников (1 балл)







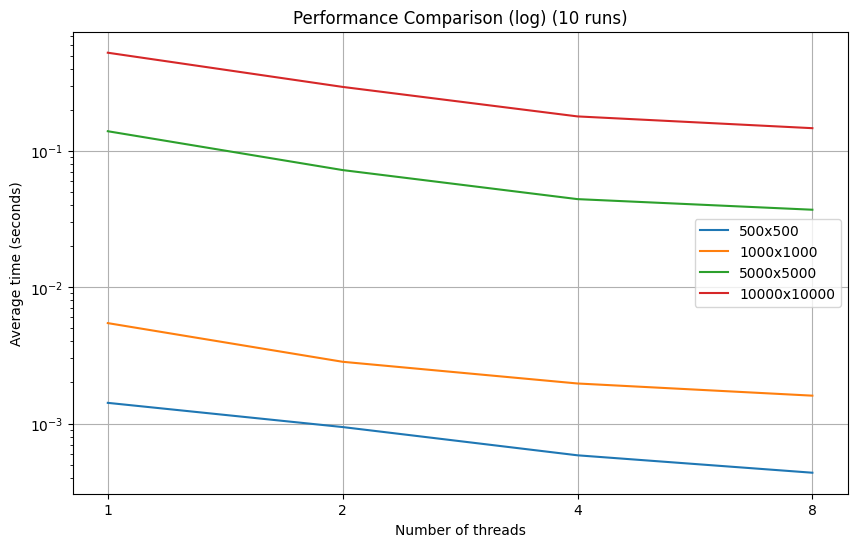
# **Вывод.**

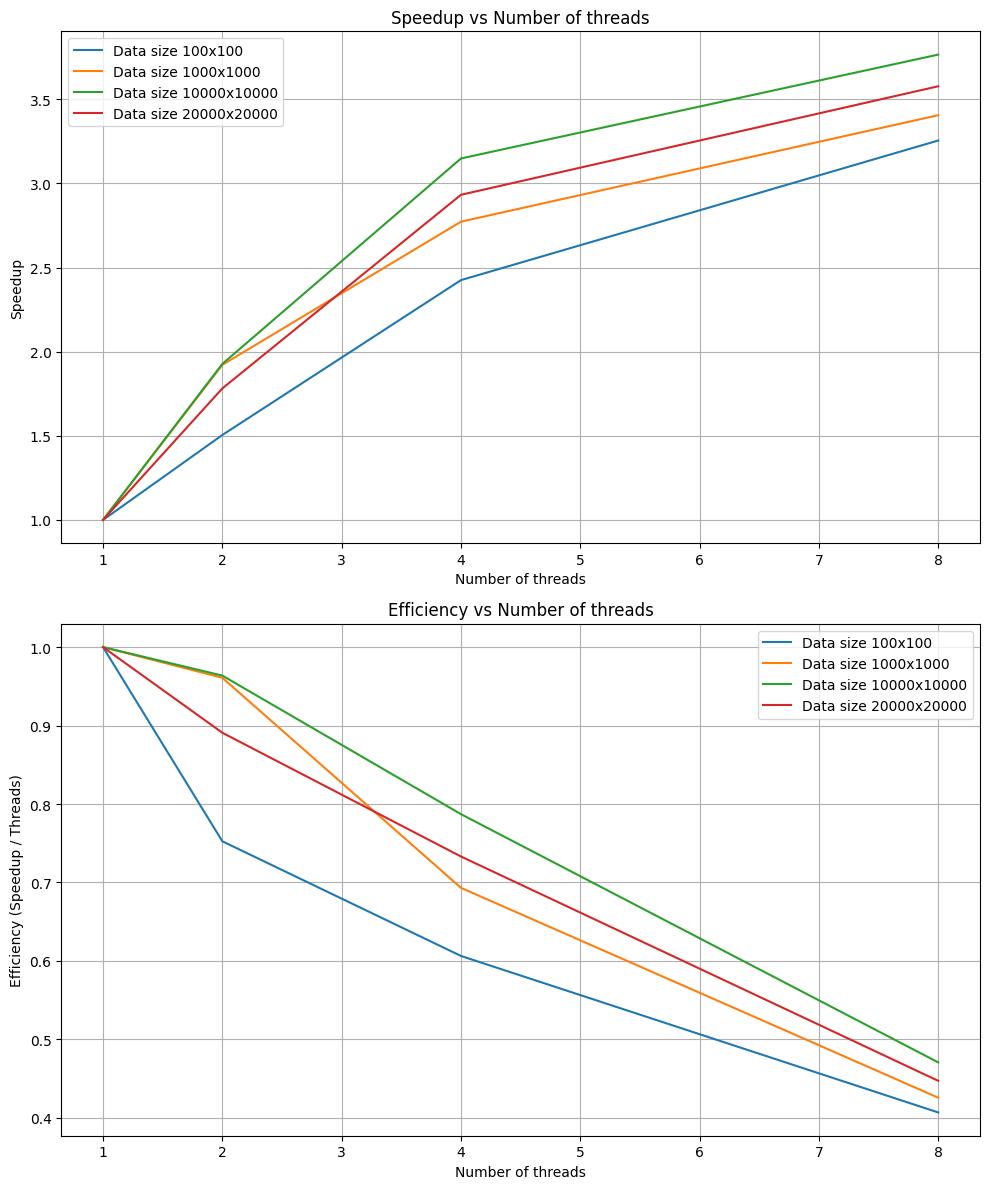
Аналогичен выводу из задания 2, но в этом случае лучшее ускорение показал меньший объем данных.

**Задание 4.**

Разработайте программу решения задачи поиска максимального значения среди минимальных элементов строк матрицы (такая задача имеет место для решения матричных игр) (2 балла)





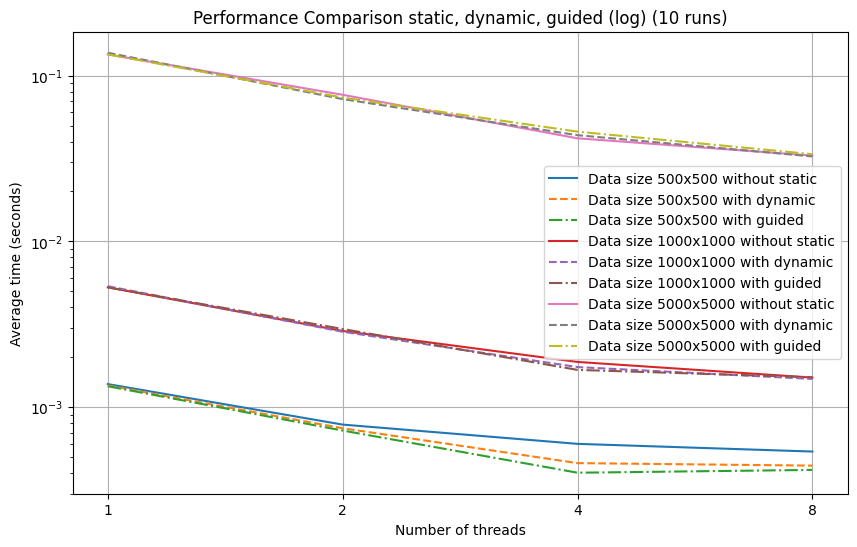


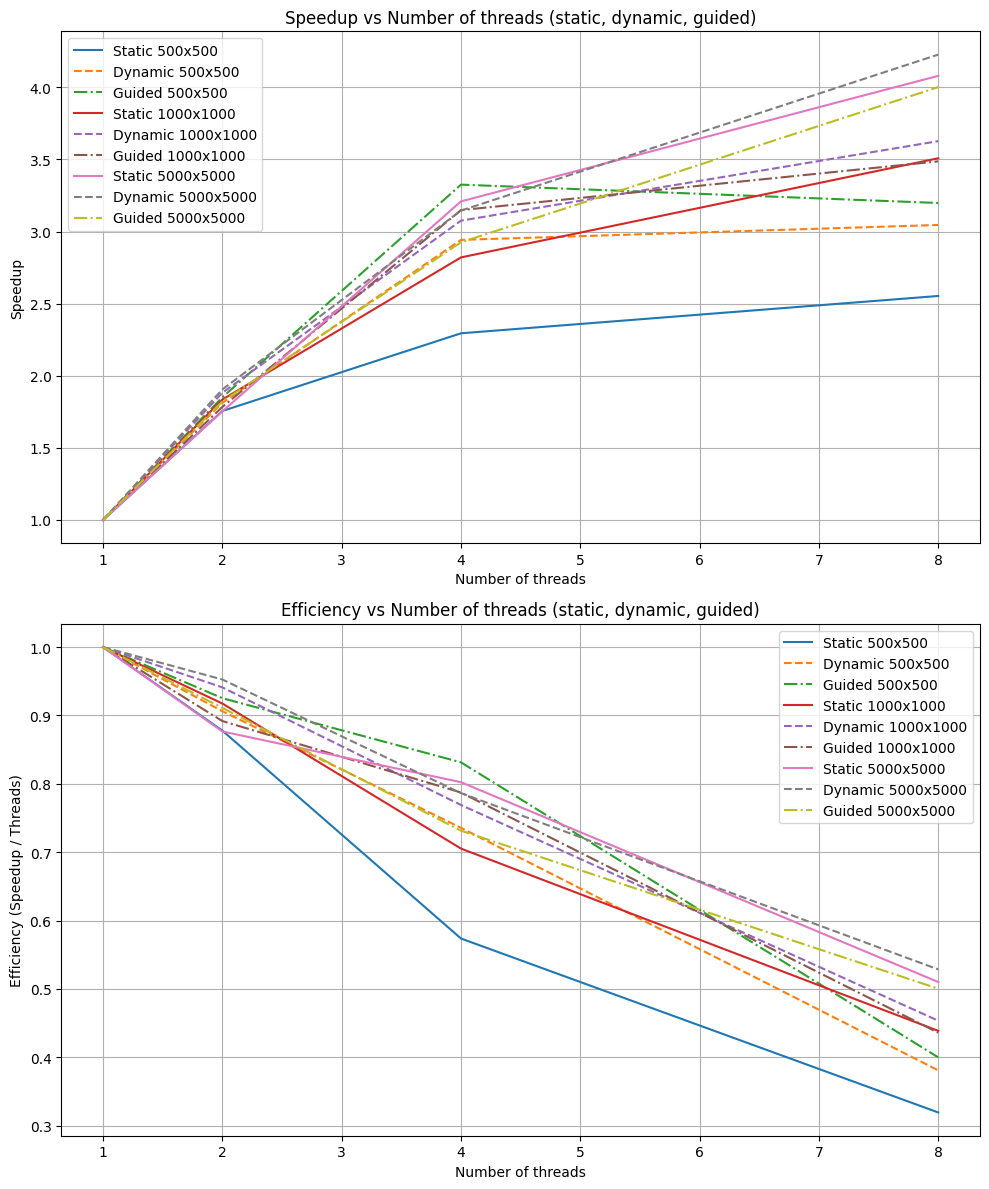
# **Вывод.**

Аналогичен выводу из задания 2.

**Задание 5.**

Разработайте программу для задачи 4 при использовании матриц специального типа (ленточных, треугольных и т. п.). Определите время выполнения программы и оцените получаемое ускорение. Выполните вычислительные эксперименты при разных правилах распределения итераций между потоками и сравните эффективность параллельных вычислений (выполнение таких экспериментов целесообразно выполнить для задач, в которых вычислительная трудоемкость итераций циклов различна). (2 балла)



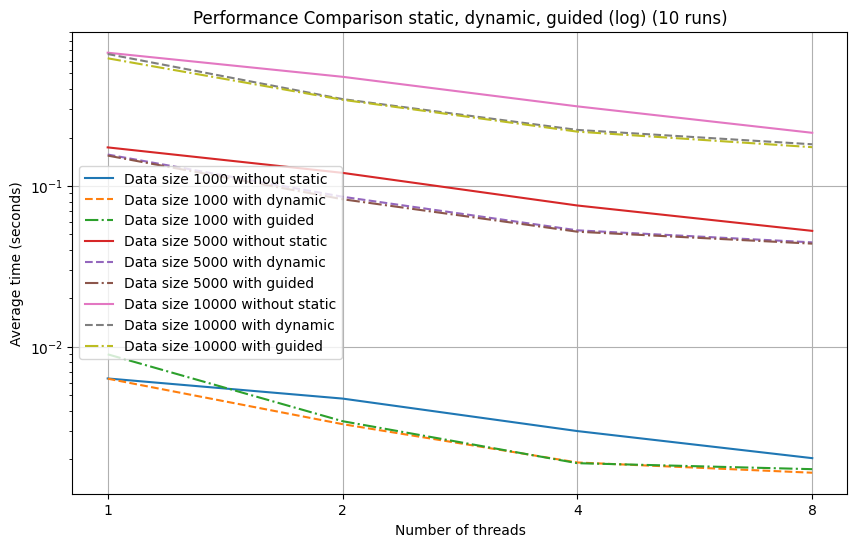


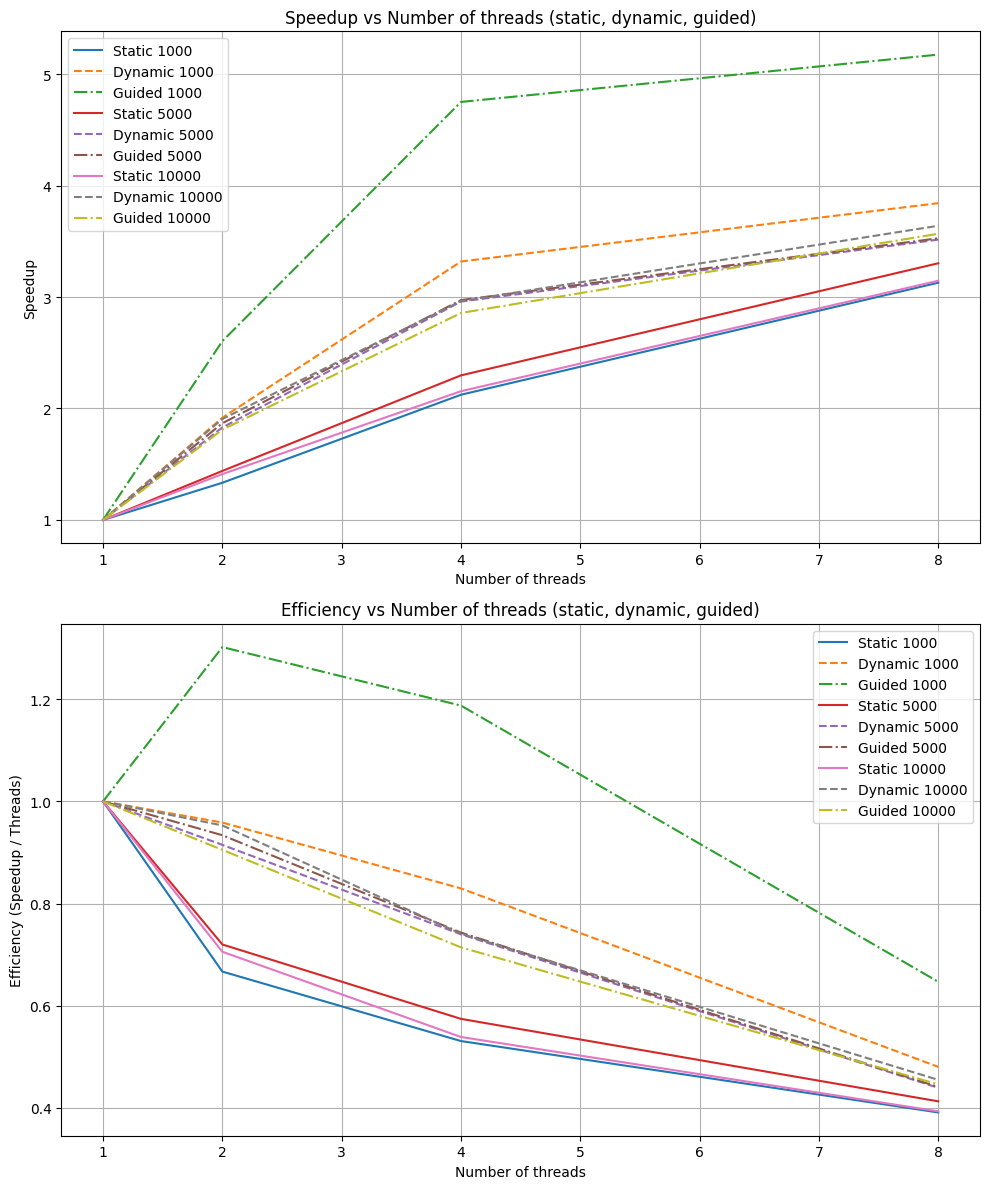
**Вывод.**

В решение поставленной задачи в случае треугольной матрицы, dynamic выдавал всегда лучшие результаты чем static. При этом, если объем данных был мал, то guided работал быстрее dynamic, с ростом объема данных dynamic стал быстрее guided.

**Задание 6.**

Проведите исследование режимов распределения итераций между потоками цикла for (static, dynamic, guided): придумайте цикл с неравномерной нагрузкой итераций (например, на некоторых итерациях генерировать случайные числа) и посмотрите, как будет изменяться время работы такой программы при различных режимах работы цикла for. (2 балла)



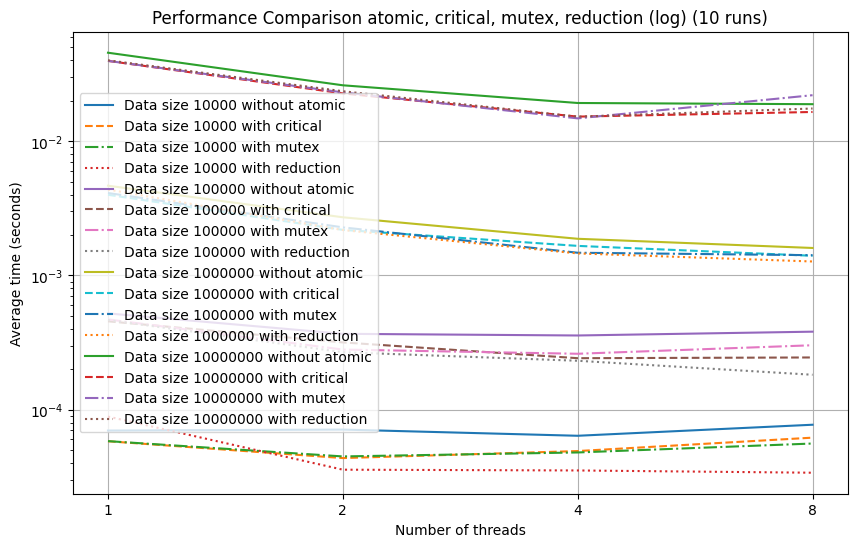
****

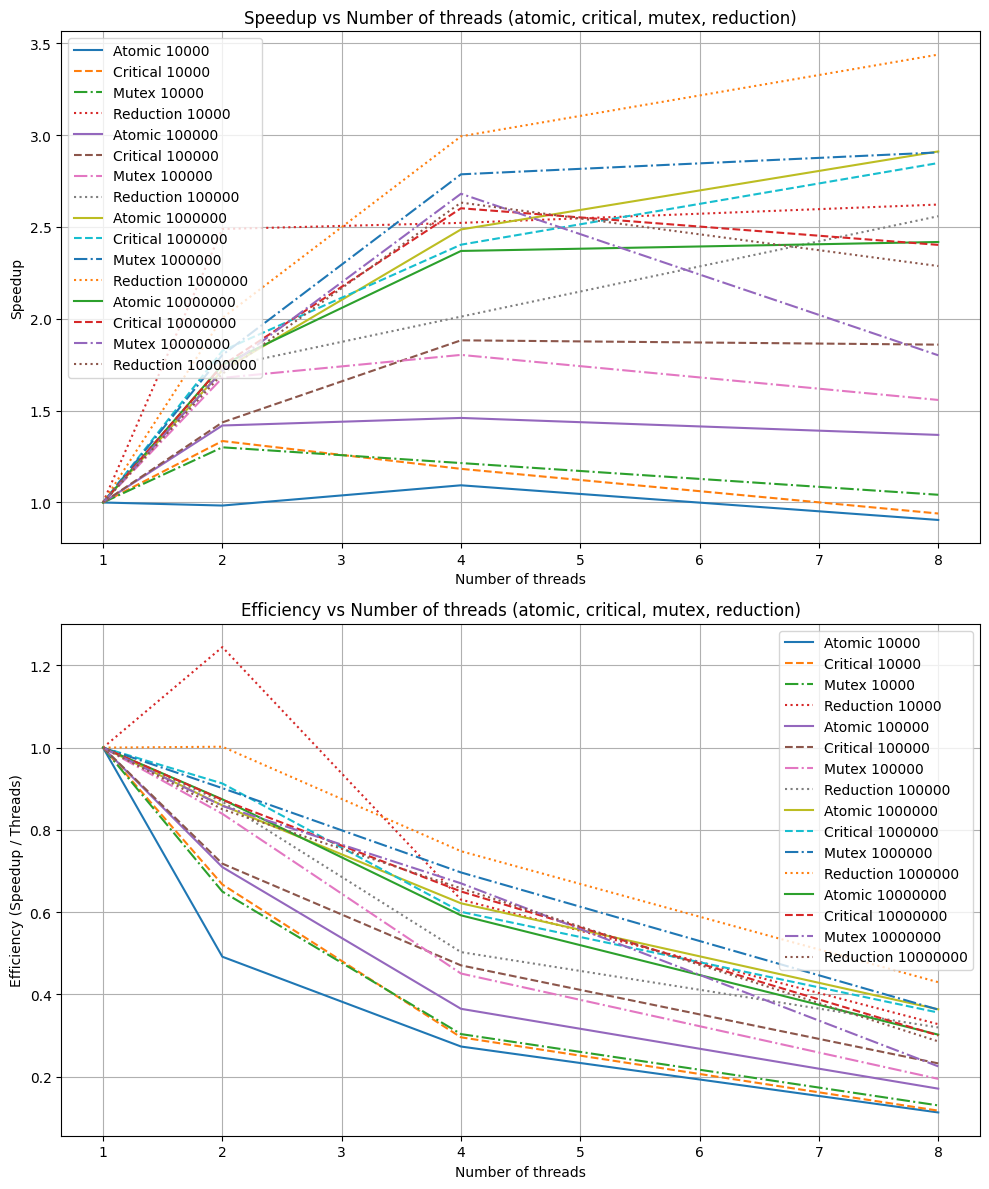
**Вывод.**

За пример была взята операция с числами Фибоначчи. Сразу видно что static работает хуже всех в этом случае.

**Задание 7.**

Реализуйте операцию редукции с использованием разных способов организации взаимоисключения (атомарные операции, критические секции, синхронизацию при помощи замков). Оцените эффективность разных подходов. Сравните полученные результаты с быстродействием операции редукции, выполняемой посредством параметра reduction директивы for. (2 балла)





**Вывод.**

Иерархия производительности: Reduction>>Mutex>Critical>Atomic

**Задание 8.**

Разработайте программу для вычисления скалярного произведения для последовательного набора векторов (исходные данные можно подготовить заранее в отдельном файле). Ввод векторов и вычисление их произведения следует организовать как две раздельные задачи, для распараллеливания которых используйте директиву sections. (3 балла)

**# Average execution time (parallel sections): 2.00009e-05 seconds**

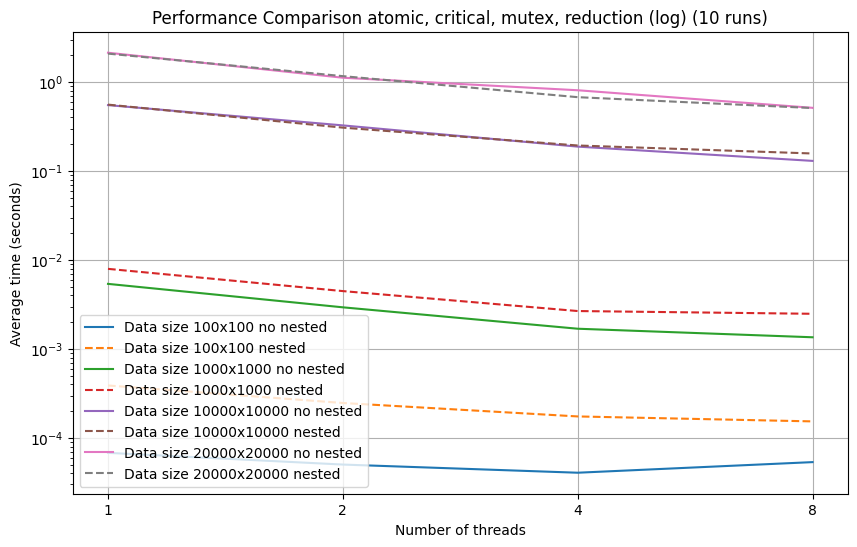
**# Average execution time (force computation): 9.99928e-06 seconds**

**Вывод.**

Sections работает хуже чем простая реализация, но позволяет разделять чтение и вычисления, что полезно в задачах где одновременно считываем и проводим вычисления.

**Задание 9.**

Уточните, поддерживает ли используемый Вами компилятор вложенные параллельные фрагменты. При наличии такой поддержки разработайте программу с использованием и без использования вложенного параллелизма (достаточно разработать программу для одной из задач, например, задачи 4) с использованием распараллеливания циклов разного уровня вложенности. Выполните вычислительные эксперименты и оцените эффективность разных подходов. (2 балла)



# 

**Вывод.**

Вложенный параллелизм показывает себя лучше на больших объемах данных, при уменьшении объема данных решение без вложенного параллелизма производится медленнее. (Отвал по памяти, не могу показать но могу предположить)

**Github.**

<https://github.com/YukiTheKilla/supercomputers>