**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”**

**По теме: “Параллельное программирование openMP”**

**Студент группы 22.Б15-пу: Колосков В.Ю.**

**Преподаватель: Корхов В.В.**

**Санкт-Петербург**

**2024 г**

**Задание №1**

Формулировка

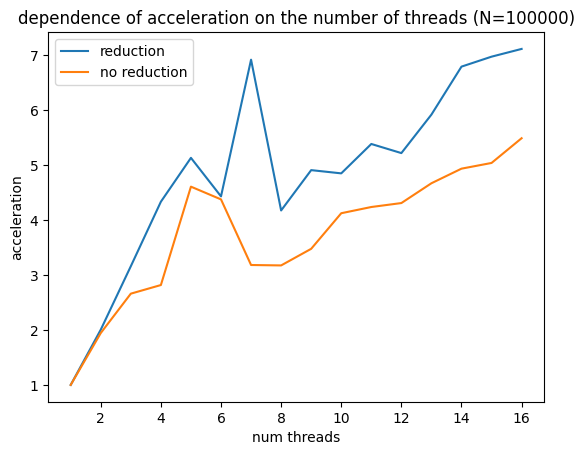
Разработайте программу для нахождения минимального (максимального) значения среди элементов вектора. Рассмотреть вариант с использованием редукции (если используемая версия OpenMP поддерживает ее) и без редукции.

Решение

Разбиваем цикл, в котором ищется максимум и минимум. Находим локальные максимумы и минимумы. Затем, благодаря редукции находим глобальные максимум и минимум.

В реализации без редукции используются критические секции, в которых находится глобальный максимум и минимум

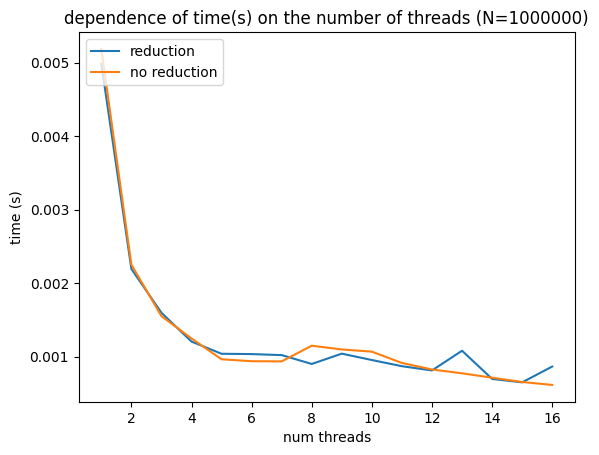
Анализ решений

Рассмотрим зависимости ускорения программы от числа потоков.

Из графика можно сделать два вывода:

1. Ускорение у программы с редукцией выше, чем ускорение у программы без редукции
2. При 5 – 7 потоках происходит всплеск в ускорении

Рассмотрим график зависимости времени работы программы от числа потоков



Из графика видно, что без редукции программа работает немного медленнее. Это связано с тем, что, при отсутствии редукции, необходимо делать больше синхронизаций.

**Задание №2**

Формулировка

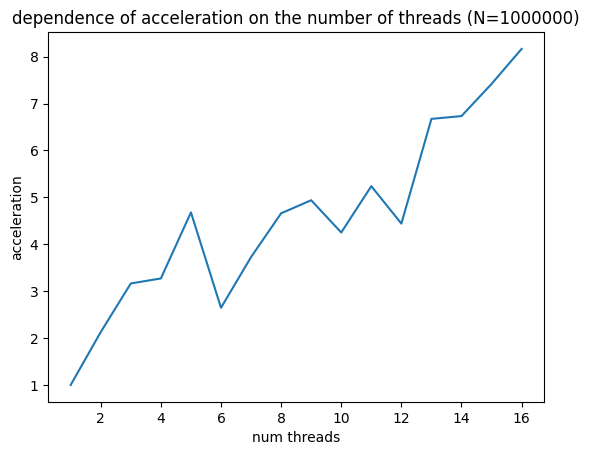
Разработайте программу для вычисления скалярного произведения двух векторов

Решение

Происходит распараллеливание цикла, в котором считается скалярное произведение. Затем, благодаря редукции, получаем ответ.

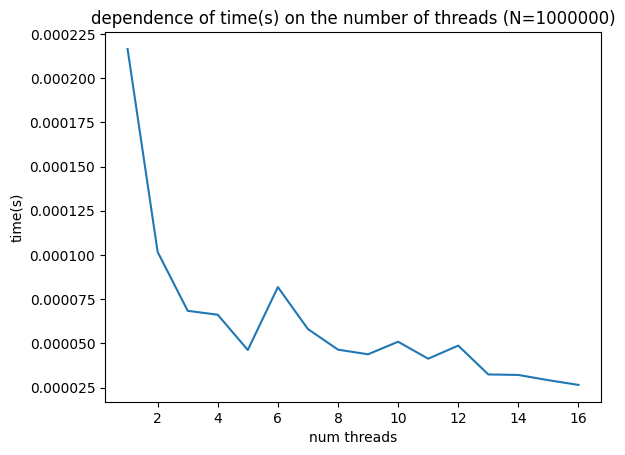
Анализ решения

Рассмотрим зависимости ускорения программы от числа потоков.



Из графика видно, что, в целом, ускорение непрерывно растет при увеличении числа потоков.

Рассмотрим график зависимости времени от количества потоков.



Из графика можно сделать вывод, что при увеличении числа потоков время работы уменьшается, что подтверждает данные из предыдущего графика.

**Задание 3**

Формулировка

Разработайте программу для задачи вычисления определенного интеграла с использованием метода прямоугольников

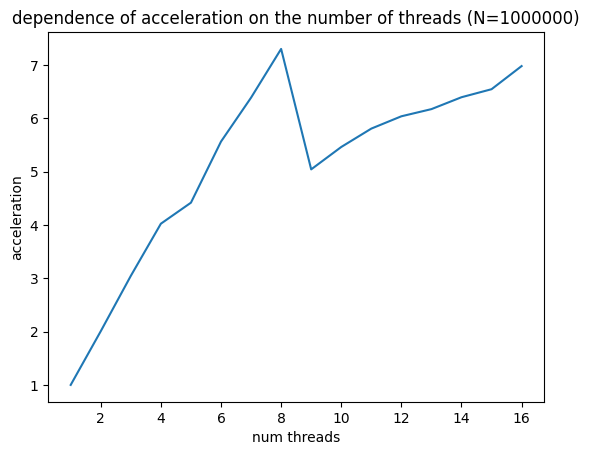
Решение

Для решения также разбивается цикл на несколько подциклов, в каждом из которых считается интеграл на определенном отрезке. Затем используется редукция.

Анализ решения

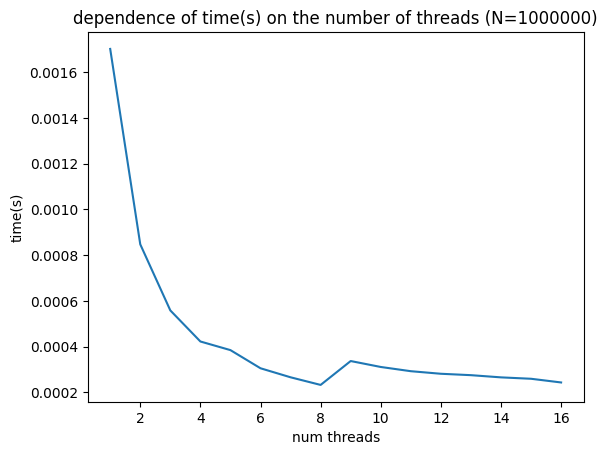
Для анализа решения взята функция y = x^3 +2x^2 – 5.

Рассмотрим зависимости ускорения программы от числа потоков.



Из графика можно сделать вывод, что максимальное ускорение достигается при 8 и 16 потоках.

Рассмотрим график зависимости времени от количества потоков.

 Данные на этом графике согласуются с данными на предыдущем графике.

**Задание 4**

Формулировка

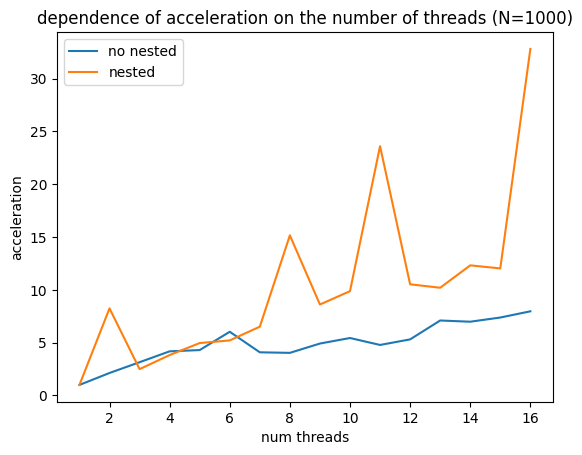
Разработайте программу решения задачи поиска максимального значения среди минимальных элементов строк матрицы

Решение

Для прохода матрицы есть 2 цикла. Во вложенном считаем минимумы и используем редукцию. Аналогично для цикла с максимумом.

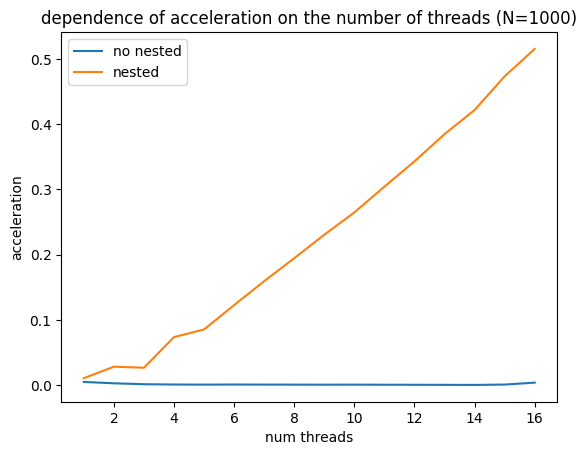
Анализ решения

Рассмотрим зависимость ускорения работы для количества потоков с включенным и выключенным nested.



Из графика видно, что при включенном nested ускорение не очень большое. При этом при выключенном, оно растет практически линейно.

Рассмотрим график зависимости времени работы от числа потоков



Из графика видно, что без nested работает сильно быстрее.

**Задание 5**

Формулировка

Разработайте программу для задачи 4 при использовании матриц специального типа. Выполните вычислительные эксперименты при разных правилах распределения итераций между потоками и сравните эффективность параллельных вычислений.

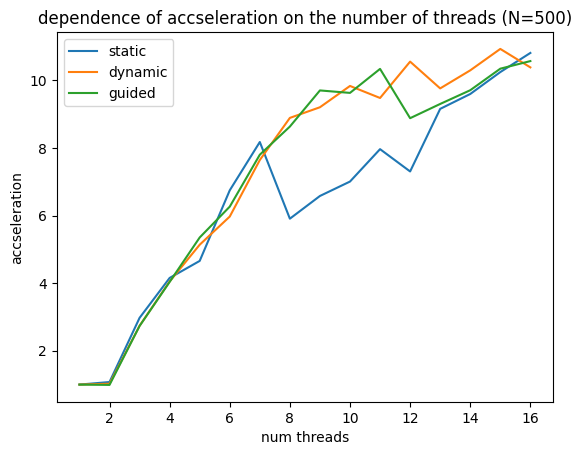
Решение

Используем оптимизацию в хранении ленточных и треугольных матриц. Для этого не будем хранить нули. Тем самым мы получаем неравномерные массивы, но это существенно экономит память и сводит алгоритм решения задачи к предыдущей.

Для прохода матрицы есть 2 цикла. Во вложенном считаем минимумы и используем редукцию. Аналогично для цикла с максимумом.

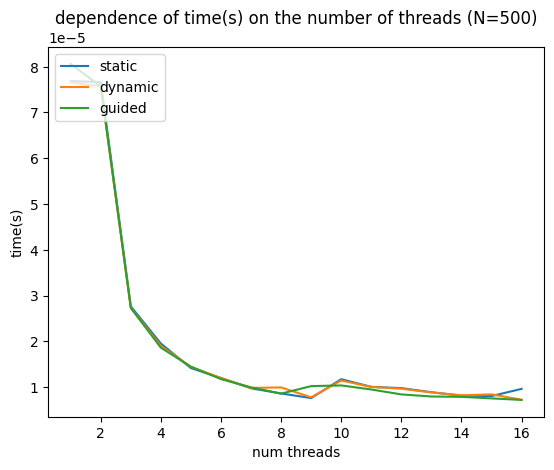
Анализ решения

Рассмотрим зависимость ускорения от числа потоков на ленточных матрицах



Из графика видно, что наибольшее ускорение у dynamic в потоках 12-16. На малом количестве потоков ускорение примерно одинаково.

Рассмотрим зависимость времени от количества потоков

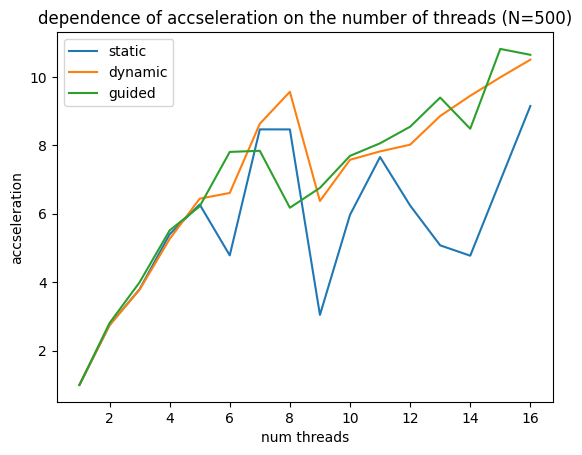


Из графика можно сделать вывод, что время работы при увеличении потоков различается между разными типами слабо, хотя guided работает немного быстрее.

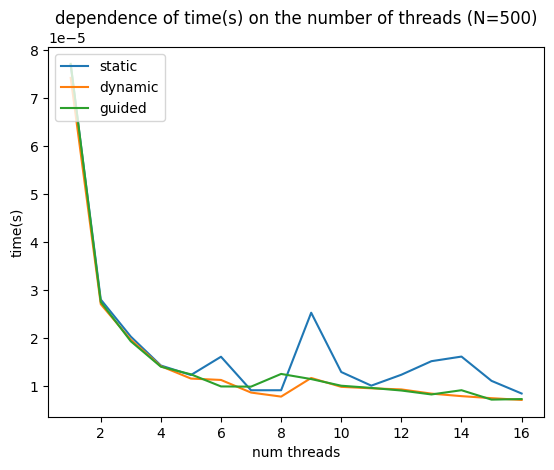
Рассмотрим работу программы на треугольных матрицах.

Рассмотрим зависимость ускорения от числа потоков.

Из графика видно, что наименьшее ускорение у static. У guided и dynamic примерно одинаковое ускорение, но у dynamic оно растет более стабильно



Рассмотрим зависимость времени работы программы от числа потоков



Из графика видно, что dynamic и guided работает наиболее быстро.

**Задание 6**

Задание

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.

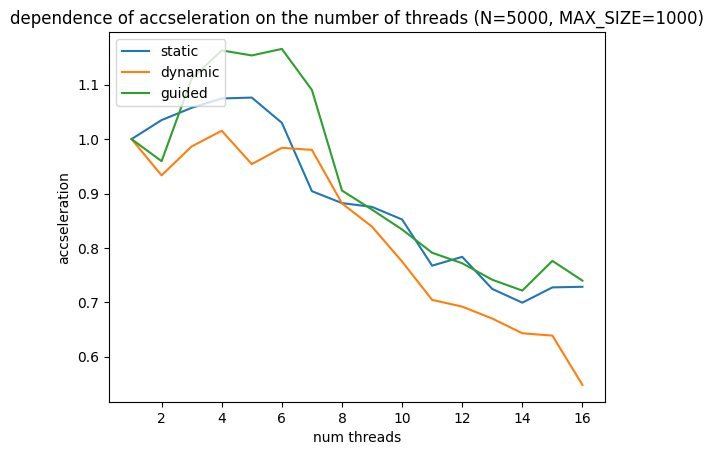
В моем случае программа генерирует N векторов произвольной размерности.

Решение

Распараллеливаем цикл, на каждой итерации которого генерируются пары векторов. В области critical записываем их в файл попарно.

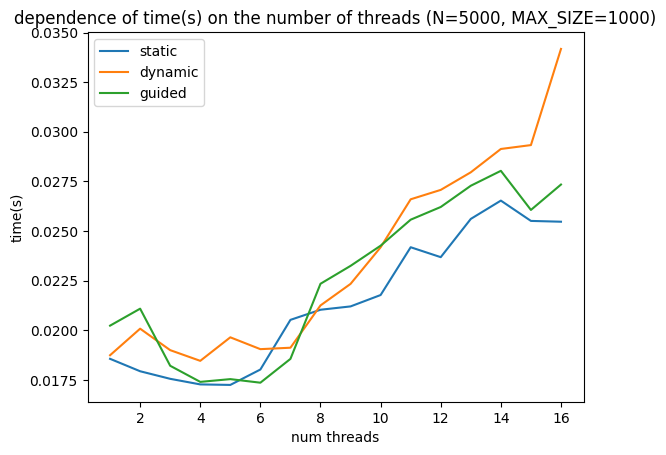
Анализ решения

Рассмотрим зависимость ускорения от количества потоков



Из приведенного графика можно сделать вывод, что наибольшее ускорение достигается при количестве потоков от 4 до 6. Наибольшее ускорение достигается при использовании guided, примерно 1.2. При увеличении количества потоков скорость работы программы меньше, чем без распараллеливания. Это связано с тем, что для безопасной записи используется область critical, что замедляет программу из-за необходимости производить дополнительные синхронизации.

Рассмотрим график зависимости времени работы программы от количества потоков.

 Из графика видно, что, при увеличении количества потоков, время работы растет, что согласуется с данными на предыдущем графике. Наименьшее время работы при использовании guided и static. Однако стоит отметить, что на 4-6 потоках у guided ускорение выше, следовательно, его более выгоднее использовать.

**Задание 7**

Задание

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.

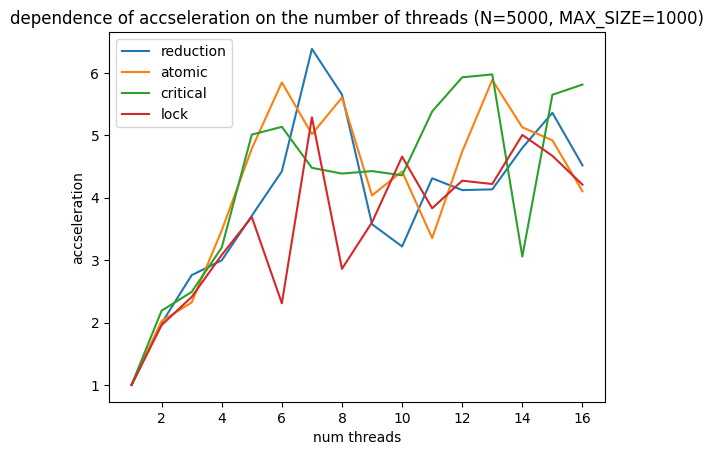
В моем случае задача состоит в подсчете суммы максимальных элементов строк матрицы

Решение

Распараллеливаем первый цикл. Вложенный цикл остается без распараллеливания. После одной итерации внешнего цикла производится редукция соответствующим способом

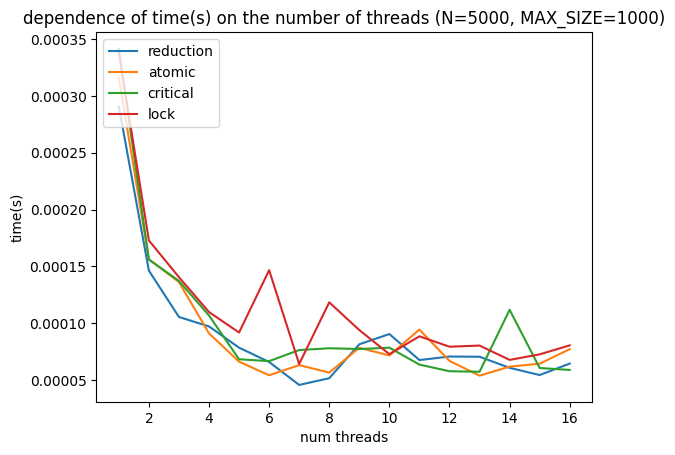
Анализ решения

Рассмотрим график зависимости ускорения от количества потоков



Из графика видно, что наибольшее ускорение достигается на разных количествах потоков для разных способов реализации редукции. Максимальное ускорение достигает у reduction при 7 потоках.

Рассмотрим график зависимости времени работы программы от количества потоков.

 Из графика видно, что наименьшее время при использовании reduction на 7 потоках, что соответствует максимальному ускорению.

Из приведенных выше двух графиков можно сделать вывод, что для задачи редукции лучше всего использовать reduction. Наибольшее время работы программы в основном достигается при использовании lock.

**Задание 8**

Формулировка

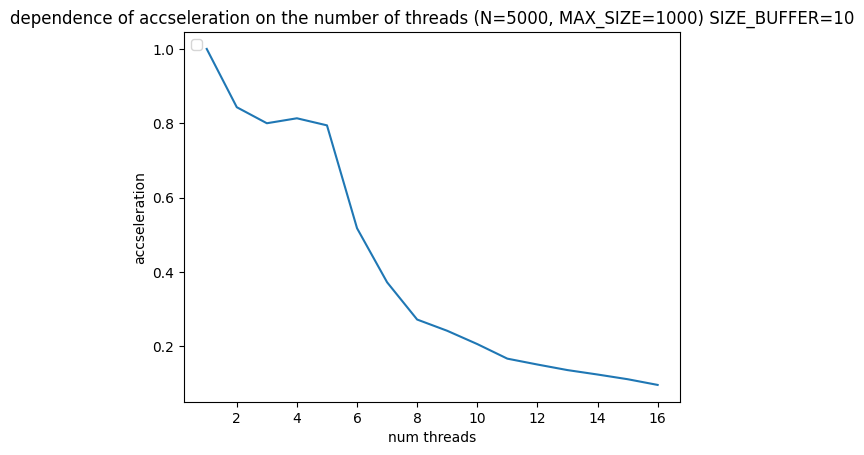
Разработайте программу для вычисления скалярного произведения для последовательного набора векторов (исходные данные можно подготовить заранее в отдельном файле). Ввод векторов и вычисление их произведения следует организовать как две раздельные задачи, для распараллеливания которых используйте директиву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

Решение

Для решения данной задачи необходимо дополнительно реализовать структуру данных очередь фиксированного размера. В нее в одной секции будут вкладываться пары векторов. Во второй секции вектора будут считываться. Если количество считанных пар векторов равно размеру очереди, то первая секция ждет, пока хотя бы 1 элемент будет взят из очереди. Если в очереди нет элементов, то вторая секция ожидает появления хотя бы одной пары. Во второй секции цикл для вычисления векторного произведения распараллеллен

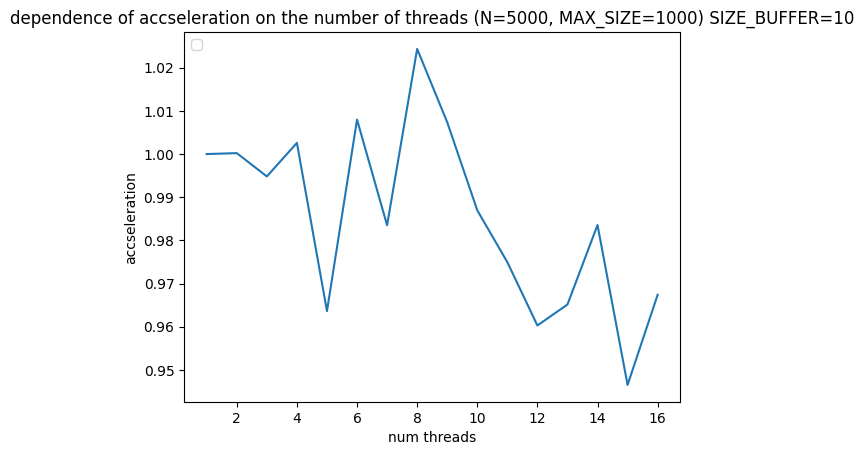
Анализ решения

Рассмотрим график зависимости ускорения работы программы от количества потоков при включенном nested

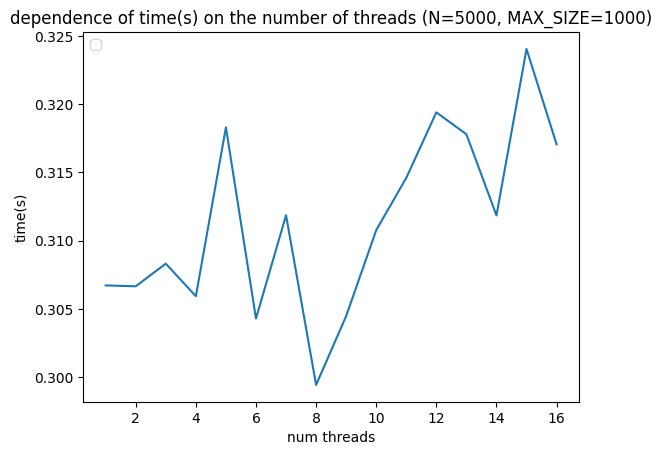


Из графика видно, что при увеличении числа потоков ускорение падает при включенном nested.

Рассмотрим ускорение от количества потоков при выключенном nested.



Из графика можно сделать вывод, что при увеличении числа потоков с 2 до 16 ускорение остается приблизительно одинаковым. Это связано с тем, что при отключении nested по сути работу могут выполнять только 2 потока. Однако с 8 по 16 потоки ускорение падает. Это связано с затратой времени на инициализацию потоков. Из графика можно сделать вывод, что в распараллеливание не имеет смысла.



Из графика можно сделать вывод, что при увеличении числа потоков с 1 до 8 время практически не изменяется, следовательно, распараллеливание не очень полезно. При работе 8-16 потоков время работы, в целом, увеличивается.

**Задание 9**

Формулировка

Исследовать, как влияет вложенный параллелизм на скорость работы программы.

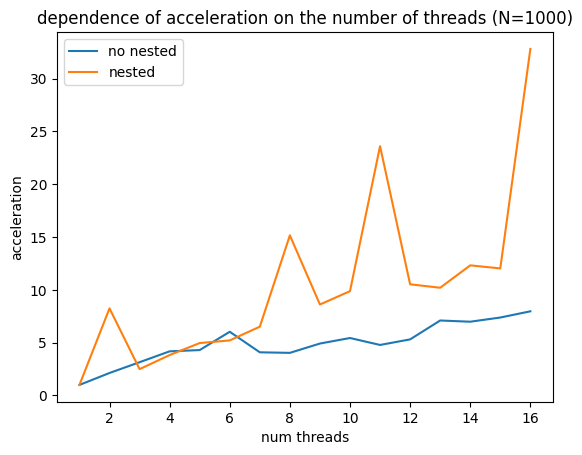
Для исследования взята программа из задания 4

Решение

Для прохода матрицы есть 2 цикла. Во вложенном считаем минимумы и используем редукцию. Аналогично для цикла с максимумом.

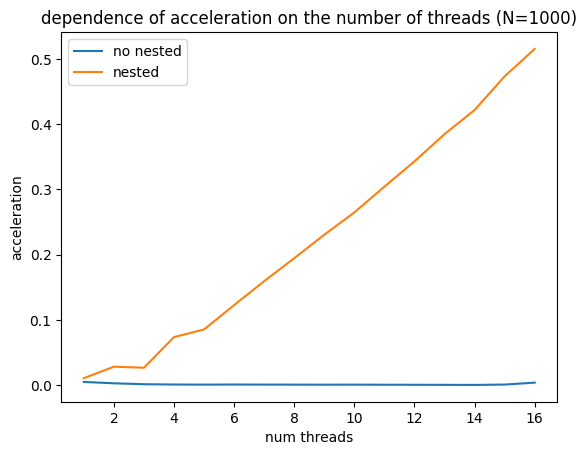
Анализ решения

Рассмотрим зависимость ускорения работы для количества потоков с включенным и выключенным nested.



Из графика видно, что при включенном nested ускорение не очень большое. При этом при выключенном, оно растет практически линейно.

Рассмотрим график зависимости времени работы от числа потоков



Из графика видно, что без nested работает сильно быстрее.