Домашнее задание №1
Липатов Данила МСМТ243
(пункты 1,2,3,4)

После подключения к HPC был перенесен файл с исходным скриптом
Далее файл компилировался :
g++ dz_last.cpp -O3 -o dz_last.out -fopenmp

С уровнем оптимизации O3

Далее выполнялась команда запуска out файла через srun
srun --cpus-per-task=16 --nodes=1 --constraint=”type_d” dz_last.out

Далее полученные результаты обрабатываем в JupyterNotebook для визуализации.

P.S. В среднем на каждую матрицу приходилось 10 запусков и результаты для каждого потока усреднялись.

Получаем следующие графики:

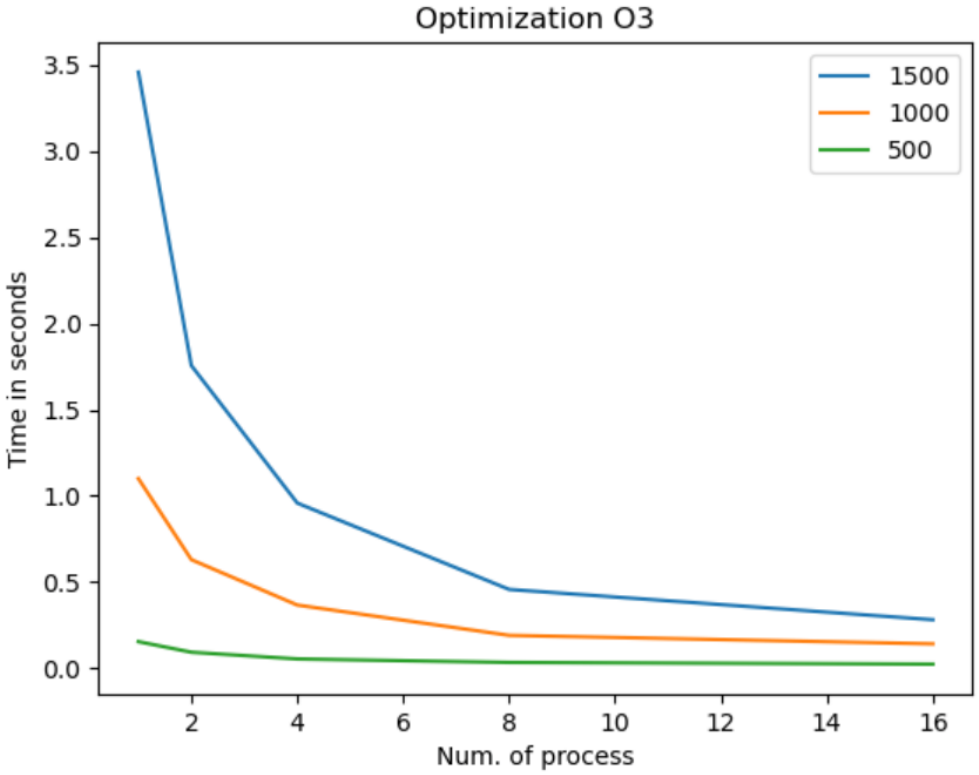


Рис. 1.1 Зависимость времени выполнения от кол-ва потоков.

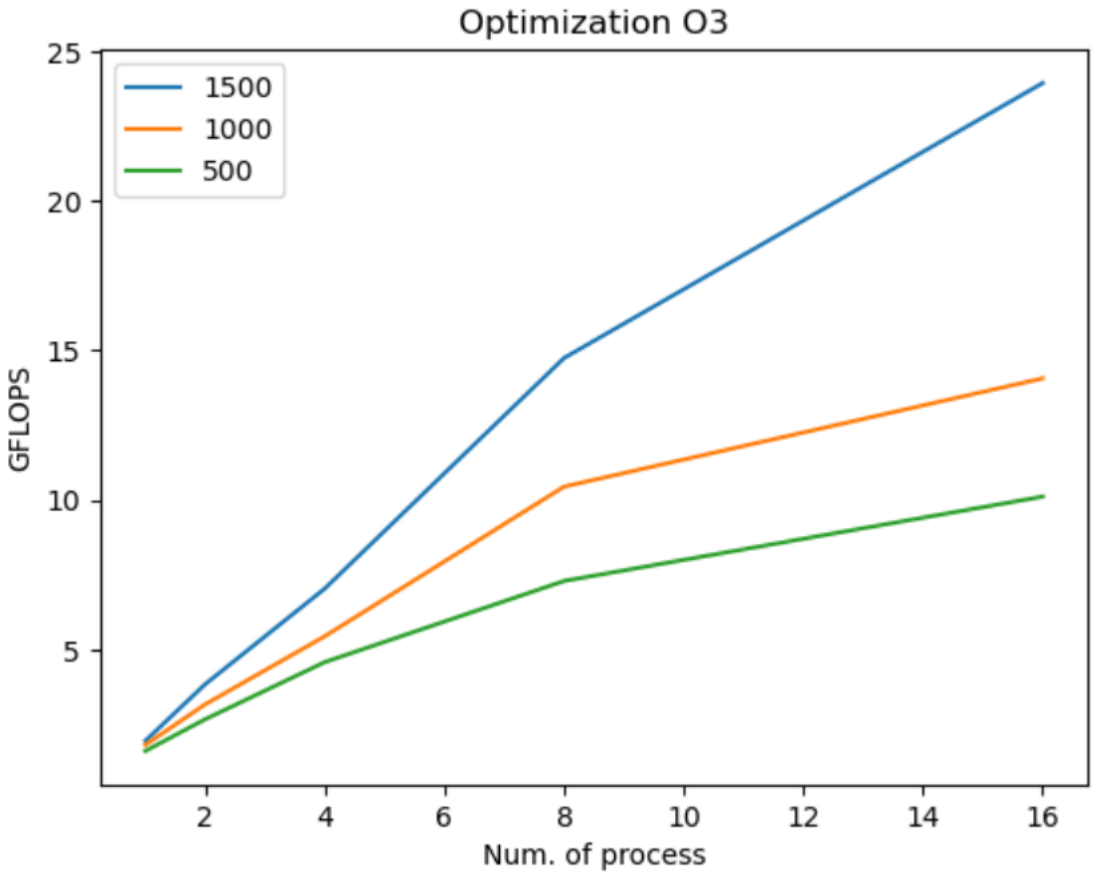


Рис. 1.2 Зависимость GFLOPS от кол-ва потоков.

порядок итерации “ijk”

Блочное умножение матриц

Так как reduction/schedule совместно с collapse не дал большого преимущества, было решено пребегнуть к иному алгоритму: блочному перемножению матриц. Самым оптимальным размером блока на основе испытаний получился – 32.

Графики:

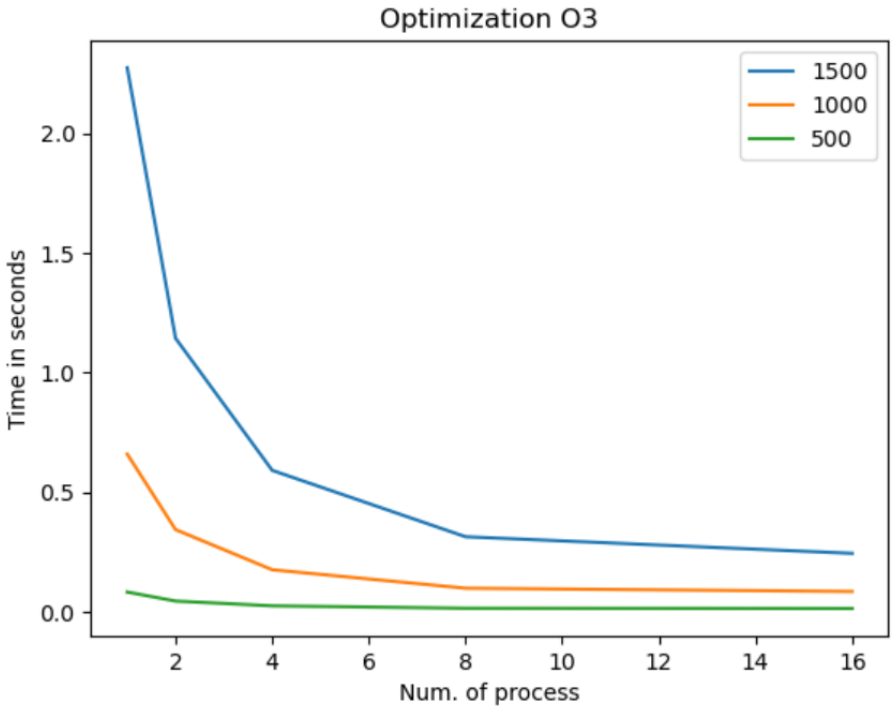


Рис. 1.3 Зависимость времени от кол-ва потоков

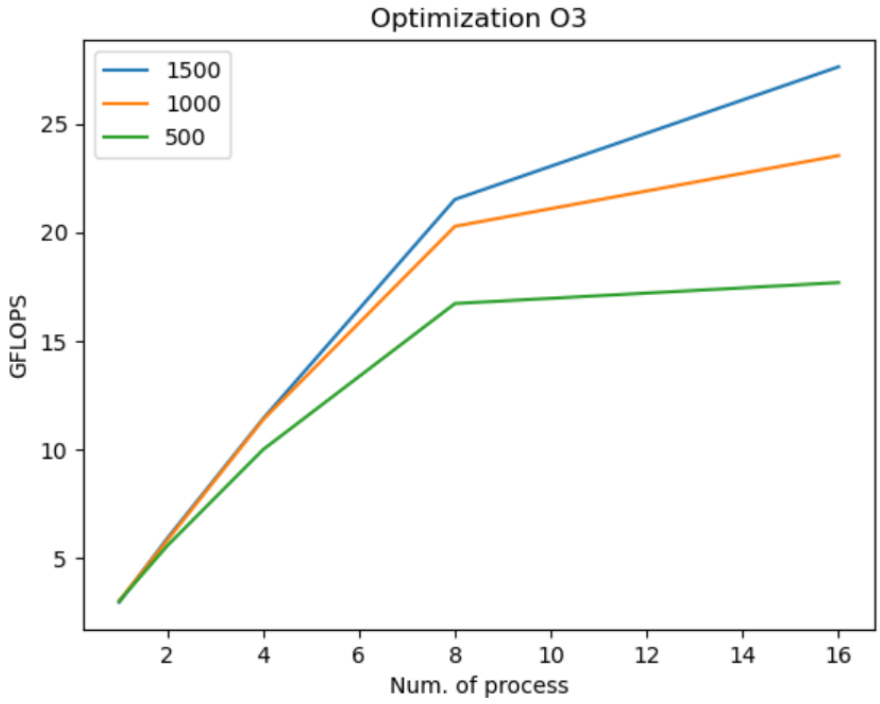
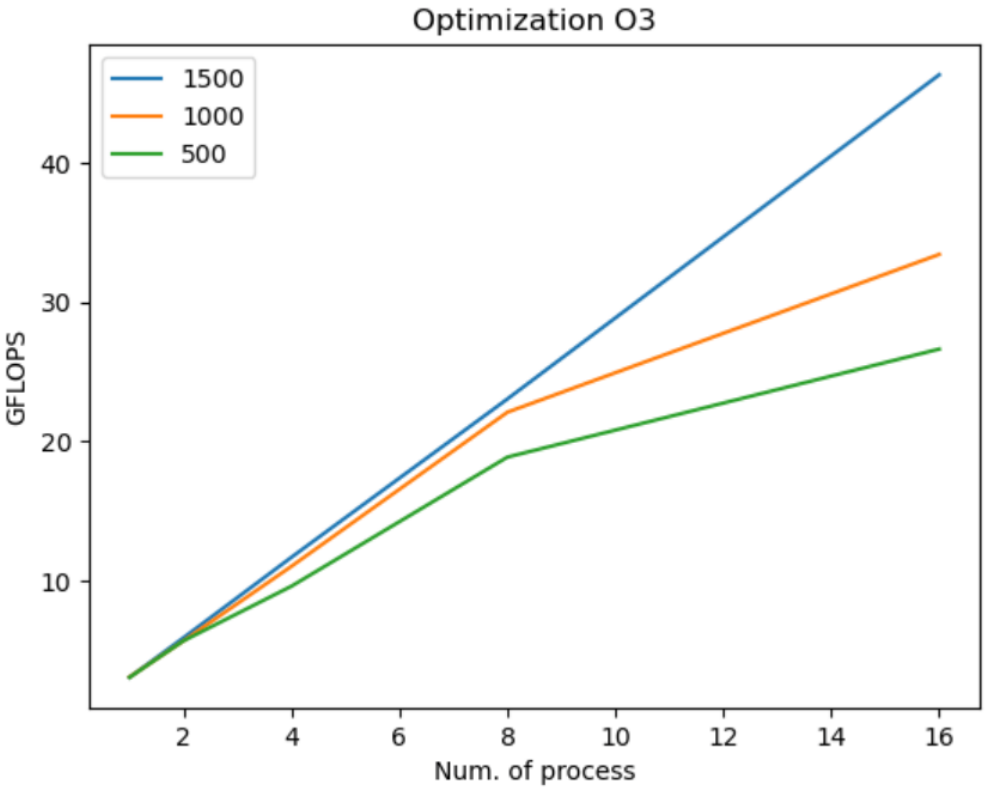
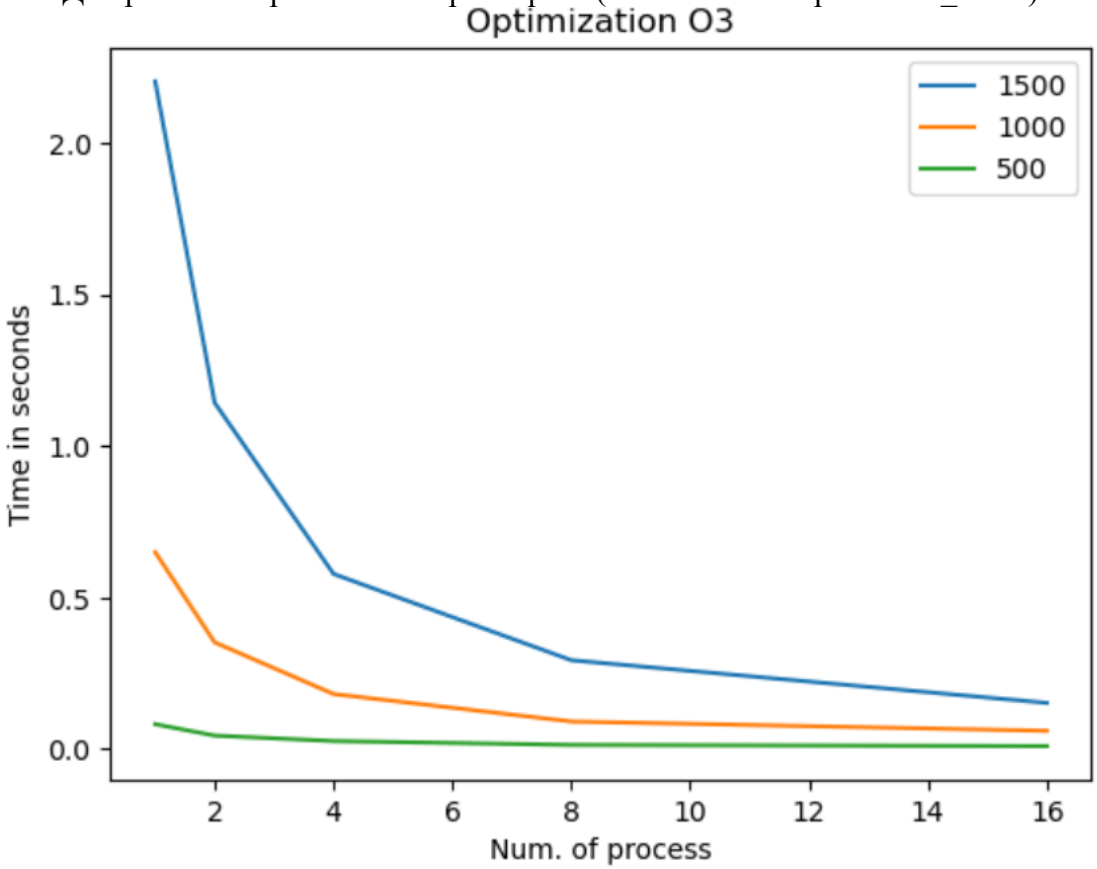


Рис. 1.4 Зависимость GFLOPS от кол-ва потоков



Доп рис. Эксперементы с параметрами(schedule + collapse + size_block)



Доп рис. Эксперементы с параметрами(schedule + collapse + size_block)

Вывод:
Данный алгоритм в свзязке с schedule(dynamic) позволил оптимизировать выполнение перемножения матриц

OpenBlas

Далее были подключены модули cblas.h / mkl.h
Для cblas результаты работы следующие (сравнивалось время выполнения)

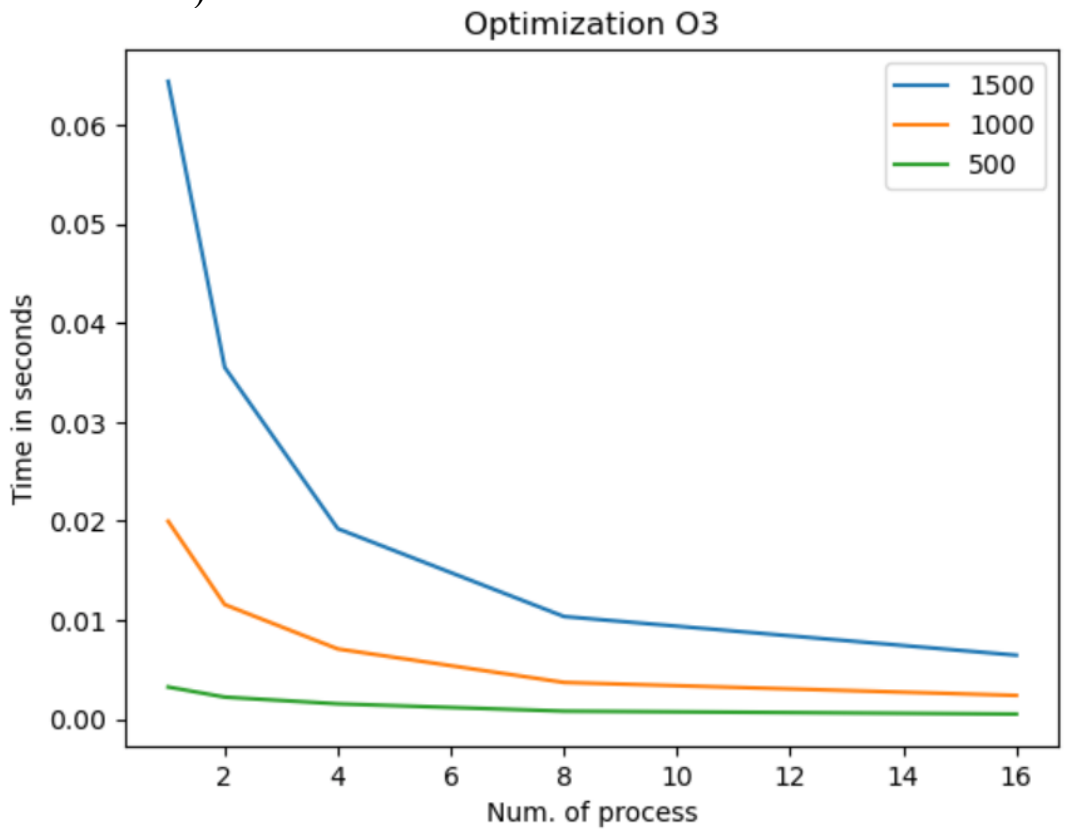


Рис. 1.5 Зависимость времени выполнения от кол-ва потоков.

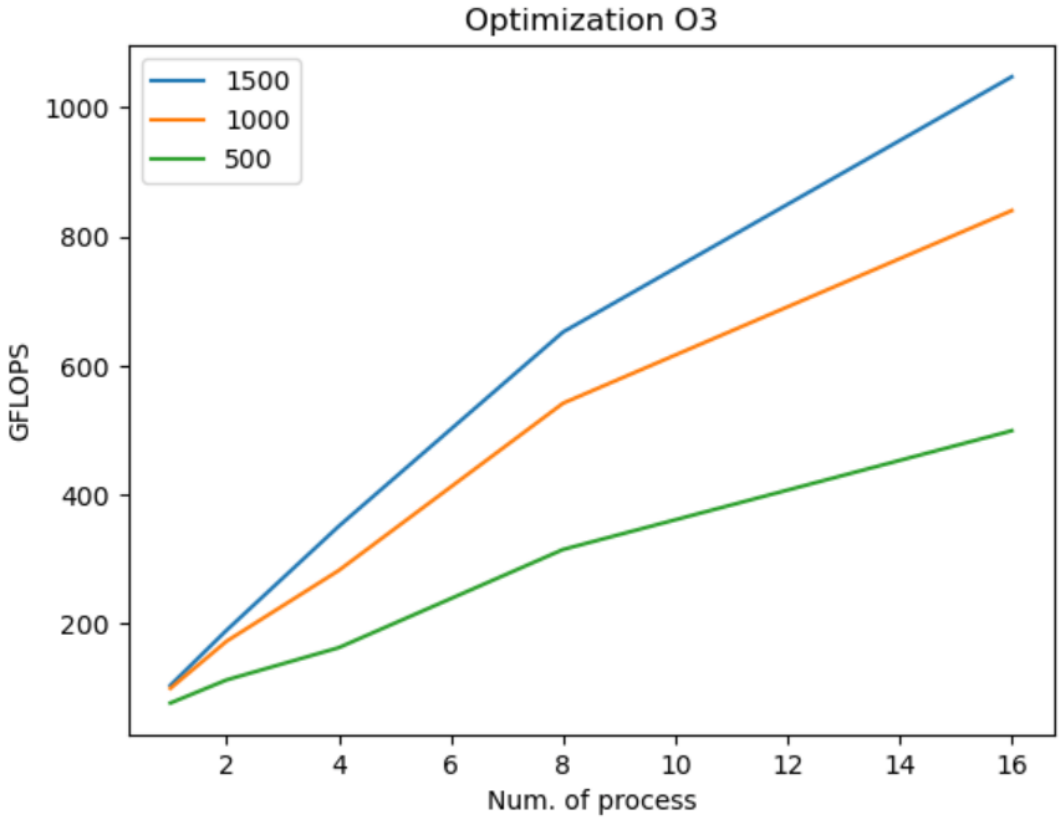


Рис. 1.6 Зависимость GFLOPS от кол-ва потоков.

Время выполнения распараллеливания в отличии от cblas_dgemm дольше в 116 раз в первом случае и дольше 33 во втором.
Так же производилось сравнение матриц после перемножения (дабы избежать вопроса гонки данных и правильности написания алгоритма перемножения)

```
Parallel
36.000000 90.000000 24.000000 117.000000
86.000000 134.000000 41.000000 94.000000
58.000000 96.000000 27.000000 103.000000
62.000000 124.000000 42.000000 120.000000
BLAS_
36.000000 90.000000 24.000000 117.000000
86.000000 134.000000 41.000000 94.000000
58.000000 96.000000 27.000000 103.000000
62.000000 124.000000 42.000000 120.000000
```

Рис. 1.7 Итоговые матрицы для каждого из подходов.

Во избежание неточных результатов и корректного сравнивания матриц (для каждого метода был отдельный файл), рандомное присваивание элементов для матрицы A и B (srand(time(0));) не использовалось.

Как можно заметить, на малом порядке размеров матрицы результаты сходятся (если бы была гонка данных, то тут бы ее можно было заметить)

MKL

Аналогичный алгоритм был произведен для MKL, согласно мануалу, который доступен на просторах интернета (ссылочка на мануал по MKL).

Однако в отличии от туториала загрузки openBLAS из ДЗ, mkl подгружался иным способом:

1- `module load INTEL/oneAPI_2022`
2- `module load mkl/2022.2.1`
3- `Этап компиляции (g++ dz_last.cpp -O3 -o dz_last.out -fopenmp -lmkl_rt)`
4- `Соответственно запуск на расчет: srun --cpus-per-task=16 --nodes=1 --constraint="type_d" dz_last.out`

В отличии от `INTEL/parallel_studio_xe_2020_ce` модуль `INTEL/oneAPI_2022` нашел расположение `mkl.h` и смог его подключить без передачи пути хардами.

Результаты выполнения:

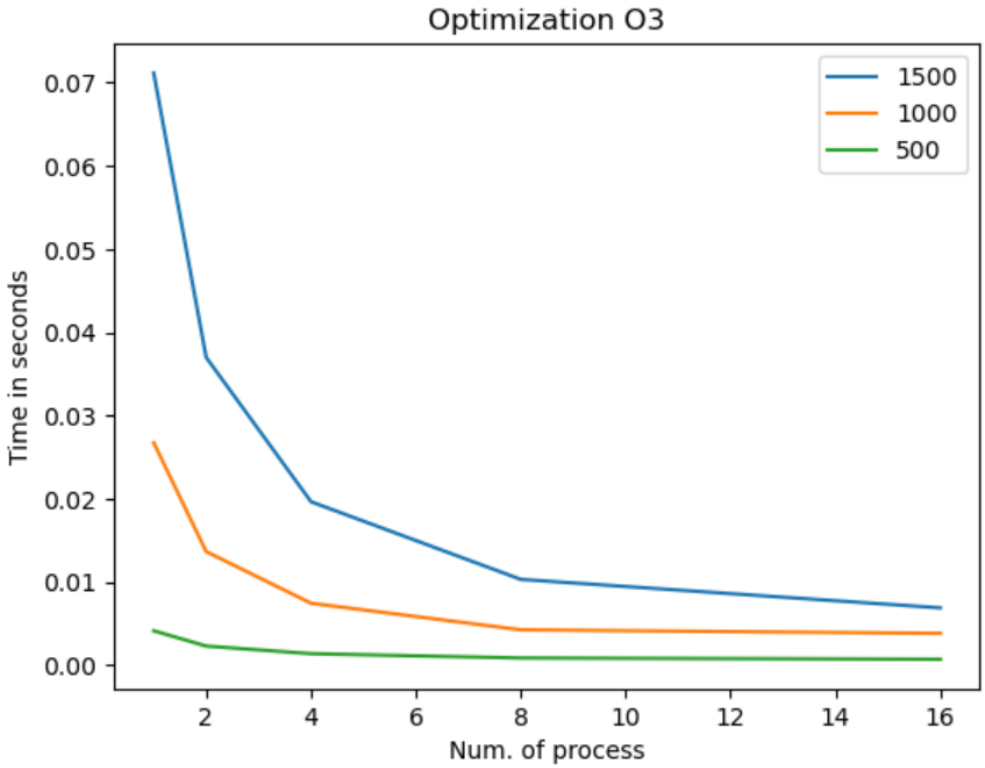


Рис. 1.8 Результат времени выполнения от кол-ва потоков.

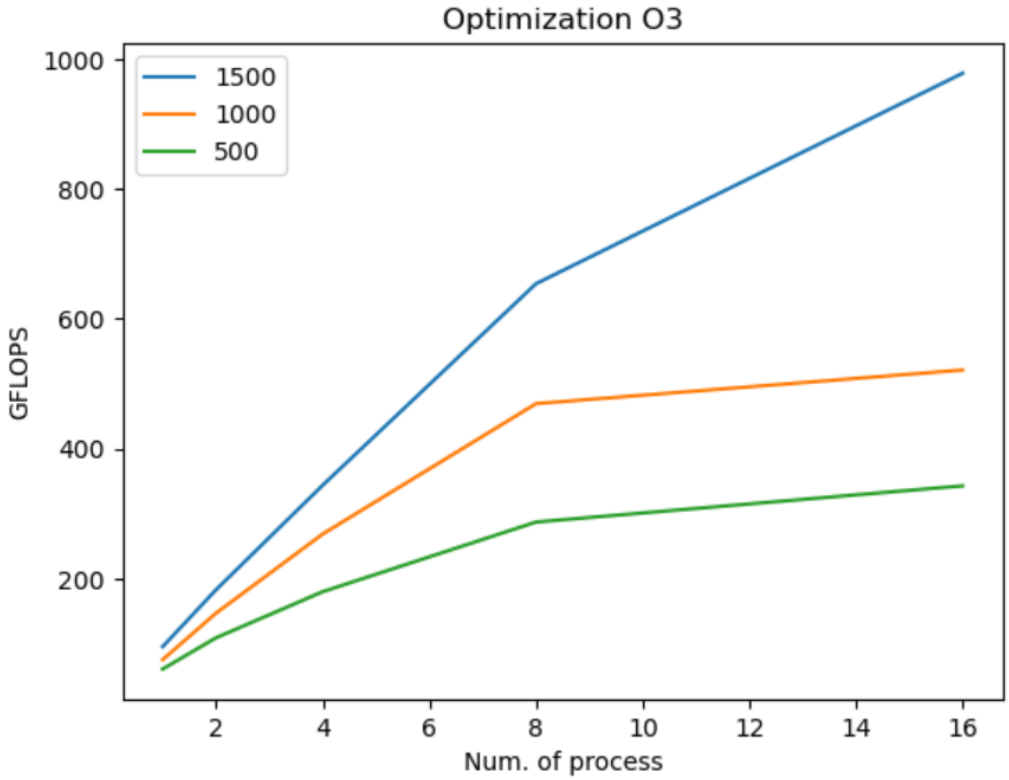


Рис. 1.9 Результат GFLOPS от кол-ва потоков.

Результаты выполнения перемножения матриц в разы быстрее, чем у параллельного выполнения, так же оба метода показали относительную стабильность в выполнении перемножения матриц.

Так же интересно было пронаблюдать зависит ли производительность от уровня оптимизации (O3 vs O0)
Результаты для MKL:

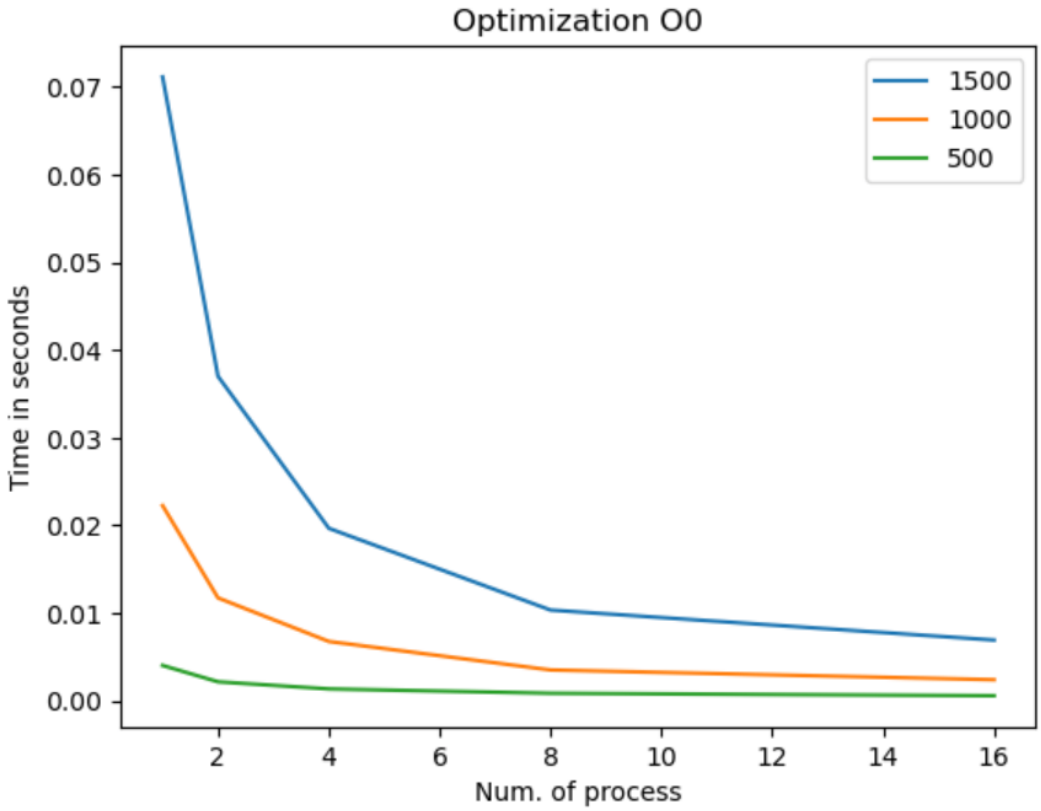


Рис. 1.10 Результат времени выполнения от кол-ва потоков.

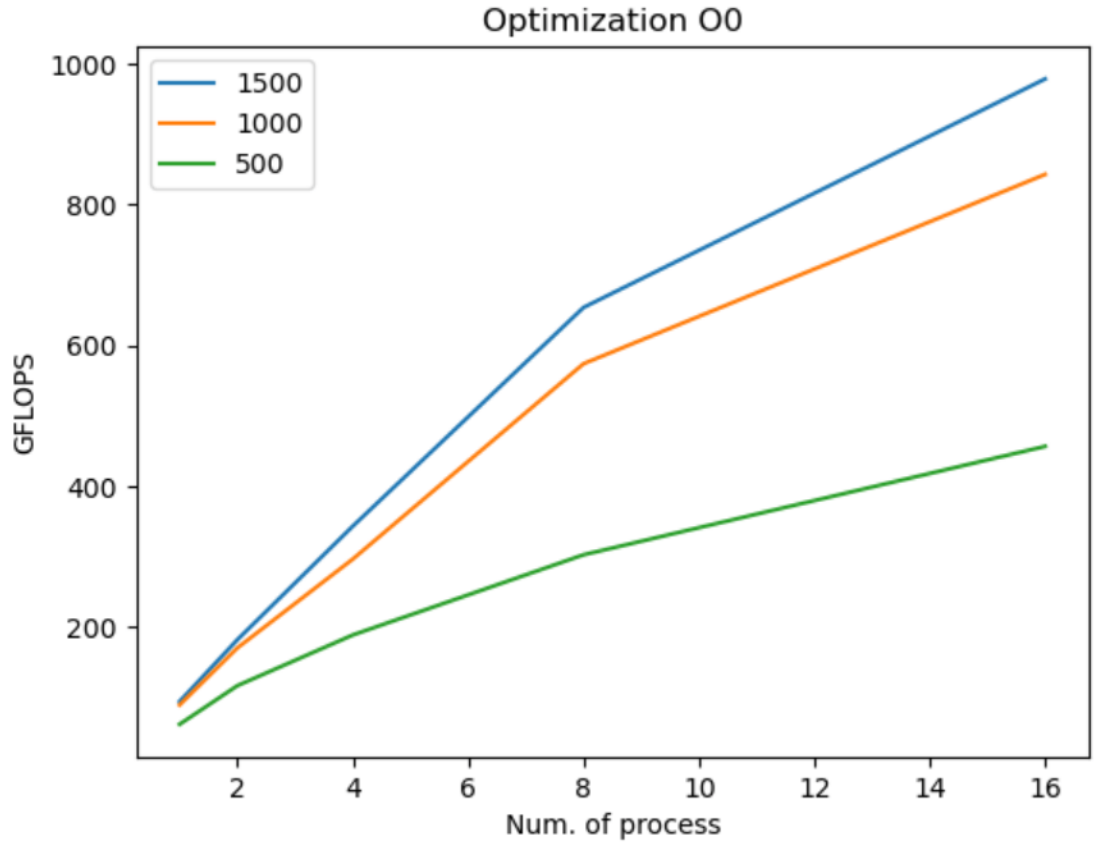


Рис. 1.11 Результат GFLOPS от кол-ва потоков.

Профилирование

Был произведен «замер» по времени скрипта на отдельных функциях и в целом, результаты в соответствующих txt файлах