Работа №2

Решение задачи теплопроводности с использованием PyCUDA / Numba / CUDA (C/C++)

После подключения к HPC был перенесен исходный скрипт (скрипт писался в VSC .cpp, но компилировался и запускался с .cu, локально так же компилировался и запускался с .cu)
Был подключен модуль для CUDA - `module load nvidia_sdk/nvhpc/23.5`

Далее файл компилировался :

nvcc hw_1.cu -o hw_1.out

И запускался на узле A следующим образом

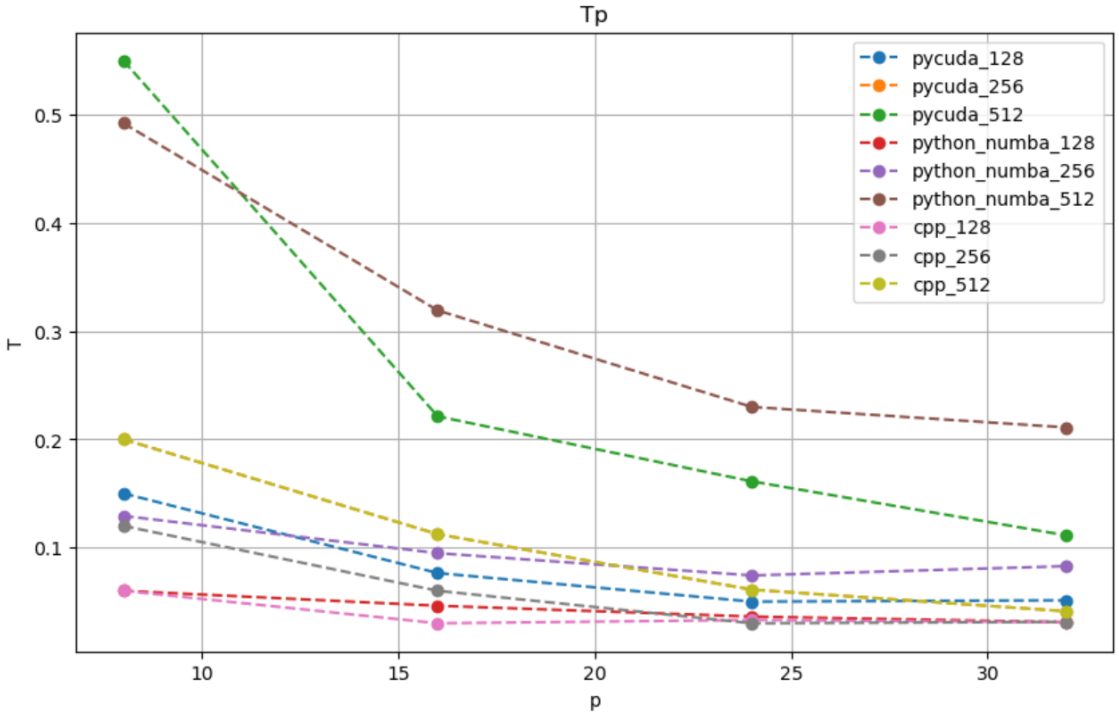
srun -n 1 --gpus=1 --constraint="type_a" hw_1.out

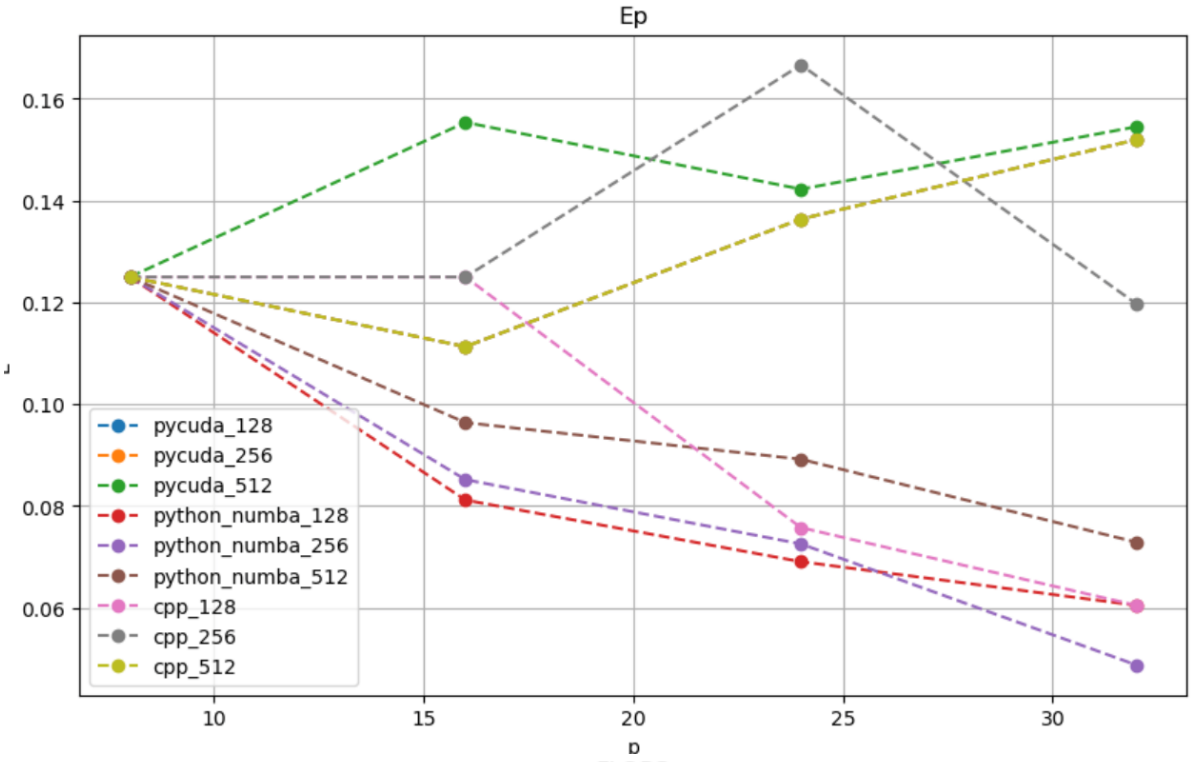
Реализация для каждого из методов лежит в соответствующем файле.

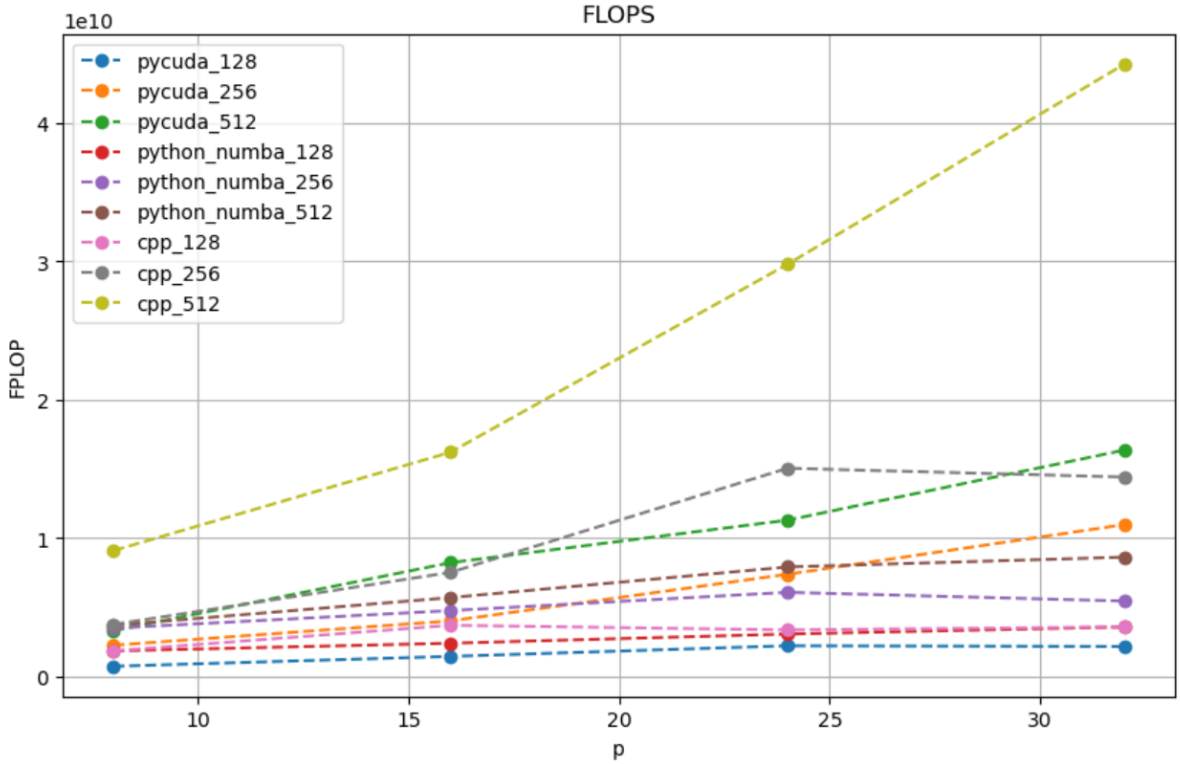
Расчеты PyCUDA / Numba были произведены локально с использованием RTX 4060 8 GB.

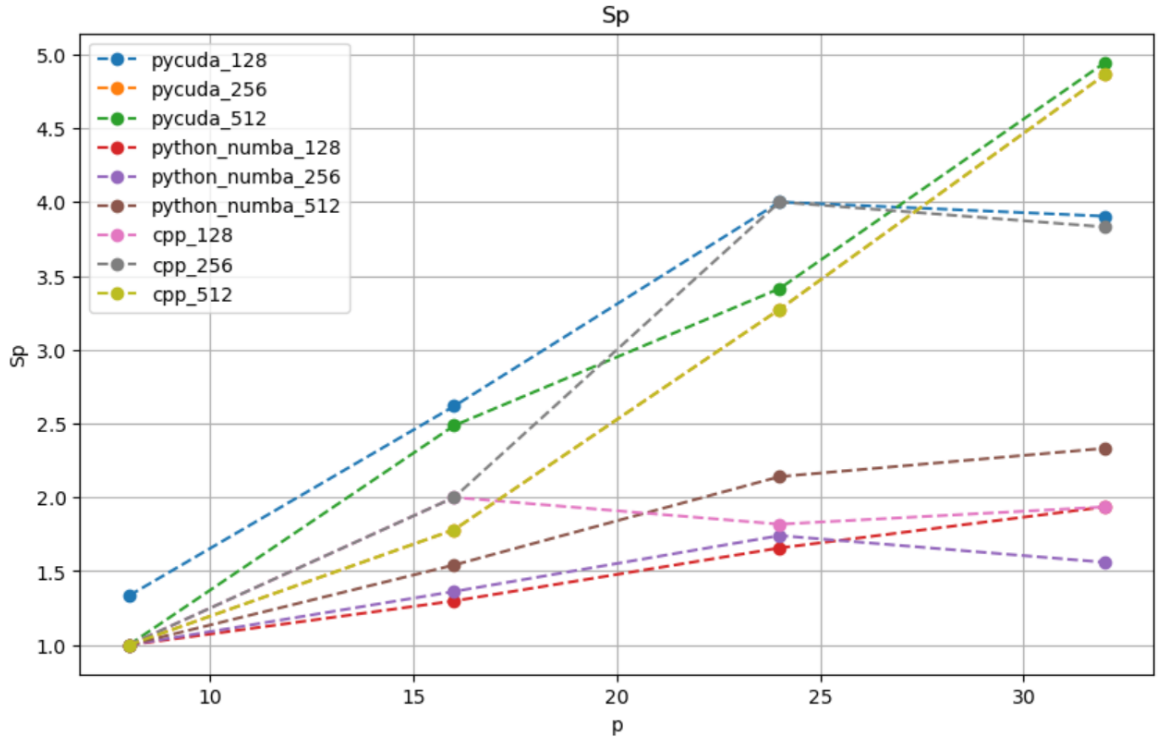
Графики при разных n и размеров блока (block-size) согласно CUDA + для Numba с использованием njit произведен отдельный расчет при разных p.

На рис. Ниже можно посмотреть все интересующие нас значения для разных n / block-size (p) и step = 1000

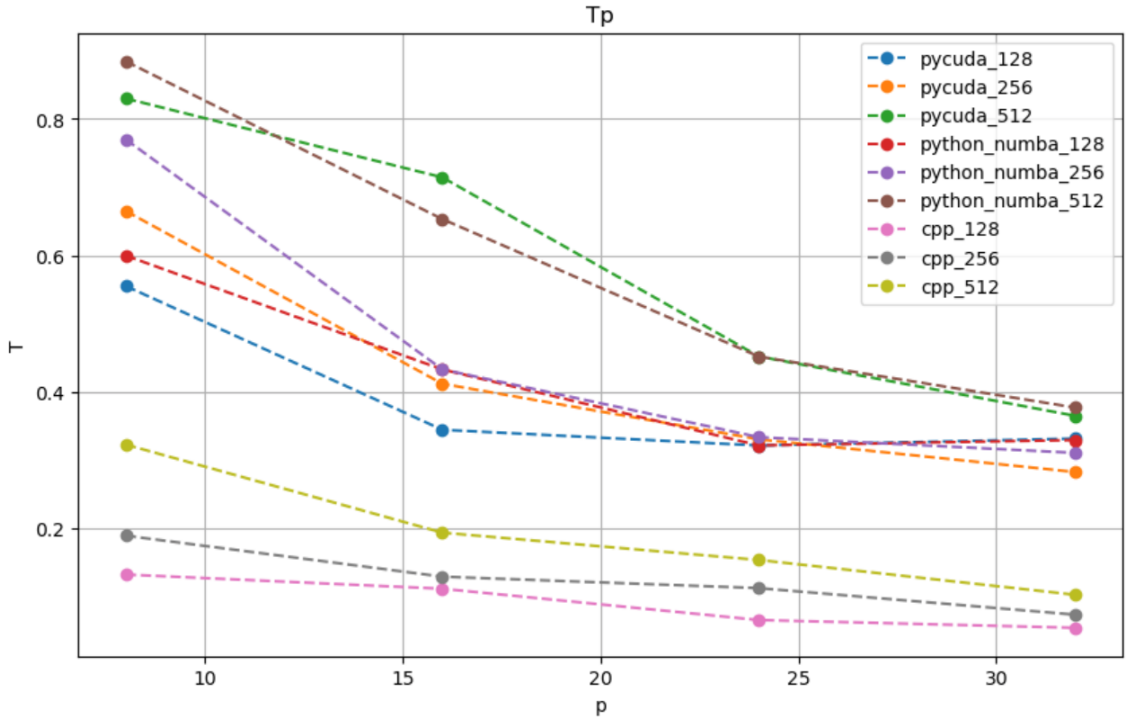


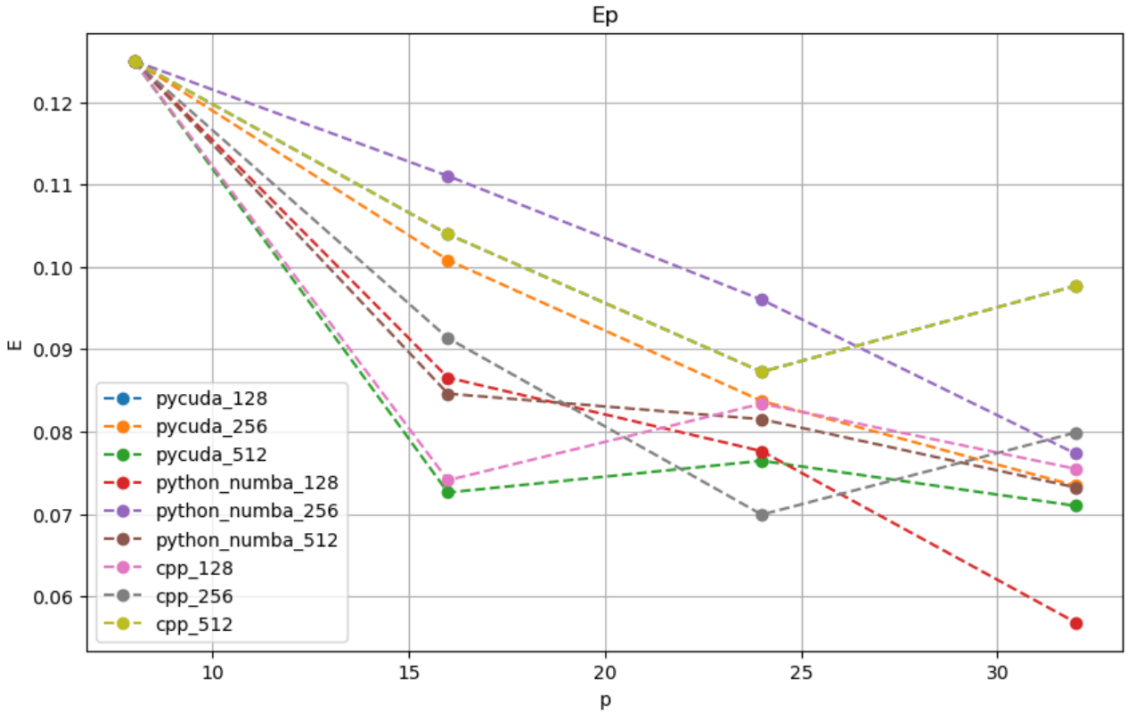

Ep
0.16
0.14
0.12
0.10
0.08
0.06
10
15
20
25
30
p
pycuda_128
pycuda_256
pycuda_512
python_numba_128
python_numba_256
python_numba_512
cpp_128
cpp_256
cpp_512

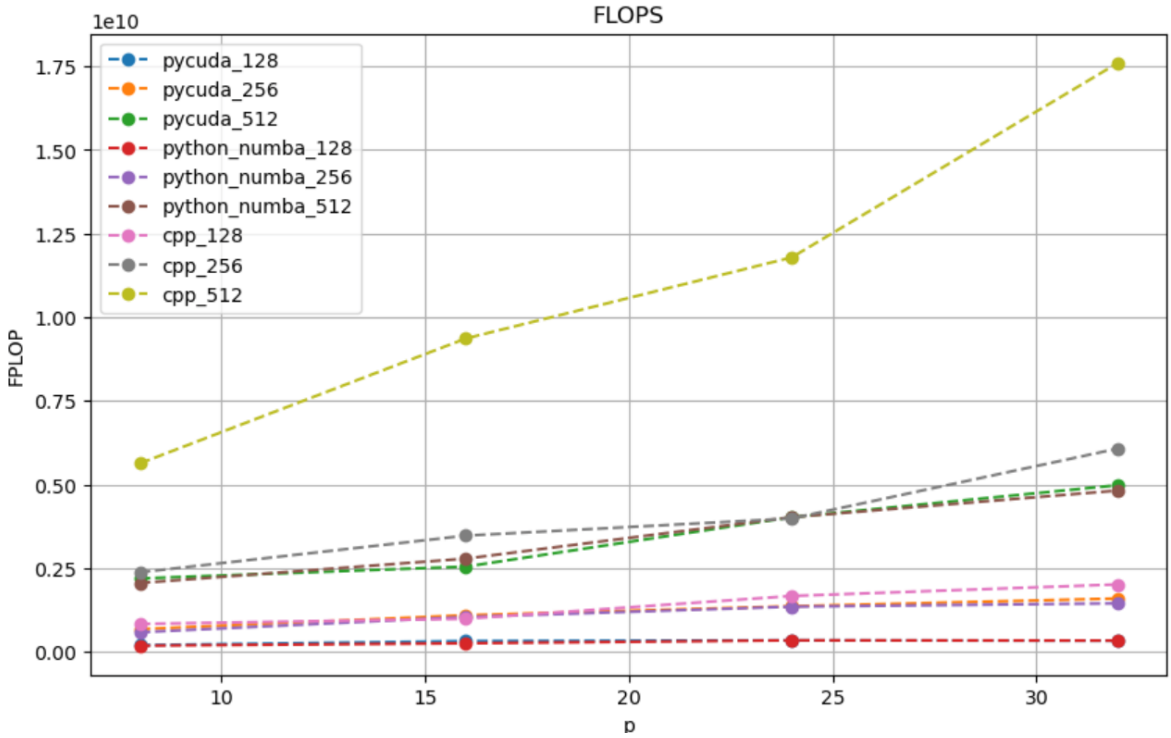

FLOPS
1e10
FPLOP
4
3
2
1
0
10
15
20
25
30
p
pycuda_128
pycuda_256
pycuda_512
python_numba_128
python_numba_256
python_numba_512
cpp_128
cpp_256
cpp_512

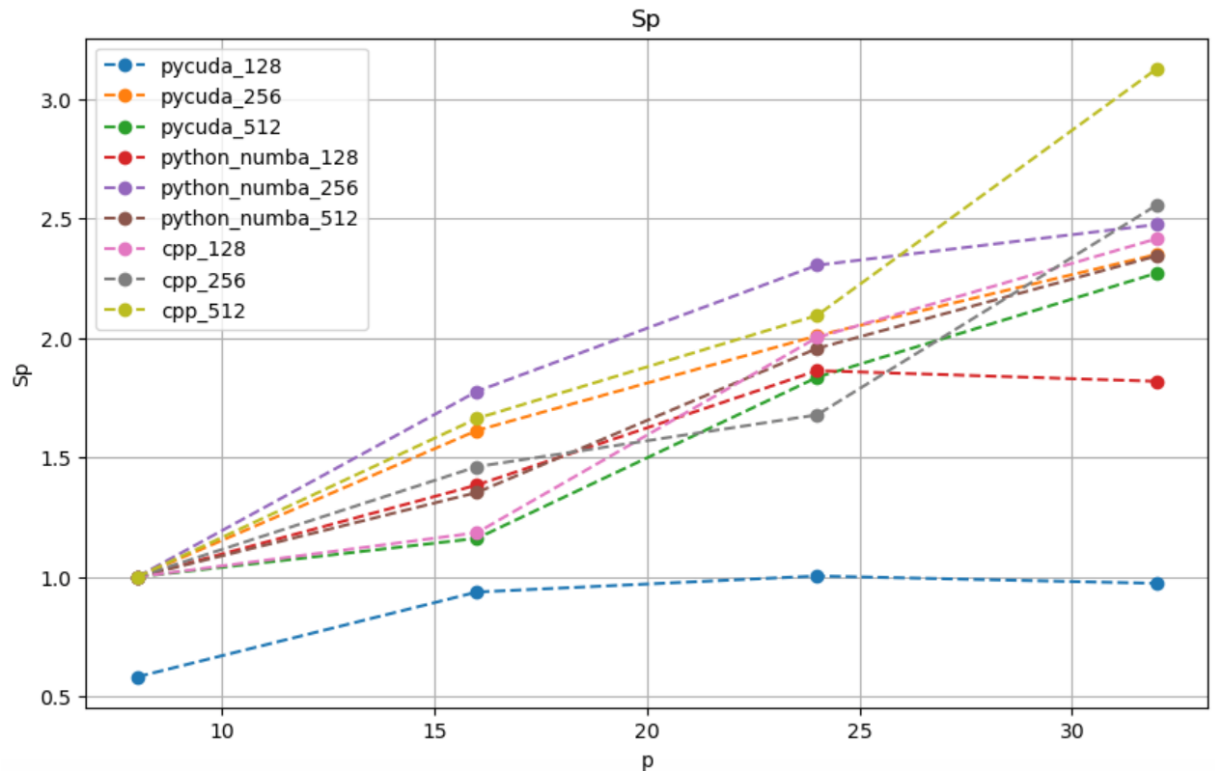


Дополнительно выведен график флопс, чтобы можно было посмотреть на эффективность того или иного подхода. CPP(cuda C/C++) дает самые лучшие результаты, а оно и не удивительно, так как уровень оптимизаций у Numba ниже чем у компилятора nvcc в C++ / C. Аналогично с Pycuda, это лишь обертки . Pycuda позволяет работать с синтаксисом cuda, но менее оптимизирован, чем на C/C++. Для Шага 10000 имеем следующие результаты:




Ep
E
p
pycuda_128
pycuda_256
pycuda_512
python_numba_128
python_numba_256
python_numba_512
cpp_128
cpp_256
cpp_512


FLOPS
1e10
FPLOP
p
pycuda_128
pycuda_256
pycuda_512
python_numba_128
python_numba_256
python_numba_512
cpp_128
cpp_256
cpp_512



В данном случае, CPP , конечно, в разы быстрее отрабатывает. При увеличении шага значения будут все еще больше и большее расходиться. Однозначно, библиотеки под Python удобнее в плане реализации, не требуют очень сложных зависимостей при компиляции, однако теряется производительность. При больших задачах это существенно влияет.

Стоит так же отметить, что локально Pycuda ставится довольно сложно, без дополнительных зависимостей от VS и CUDA toolkit проблематично запустить расчеты. Numba в njit в этом плане гораздо удобнее, не смотря, что расчеты производятся на CPU, они не так критично отклоняются от GPU при малых значениях. Как только ставится Pycuda в режиме. Cuda_jit в Numba можно расчеты в разы быстрее производить.