Università della Svizzera italiana

Institute of Computing CI

High-Performance Computing **2022**

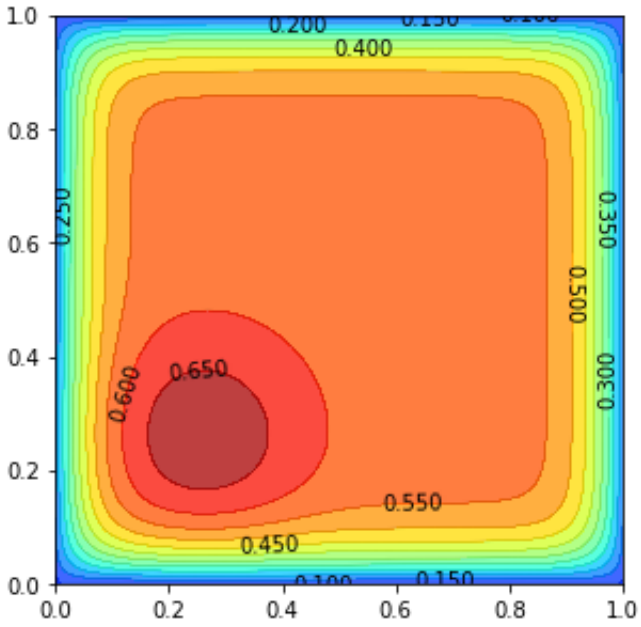
Student: Anthony Bugatto

Solution for Project 3

Due date: 09.11.2022, 23:59

1. Task: Implementing the linear algebra functions and the stencil operators [35 Points]

Using the parameters given in the specification, the results are approximately the same but with much higher efficiency and using one more iteration.

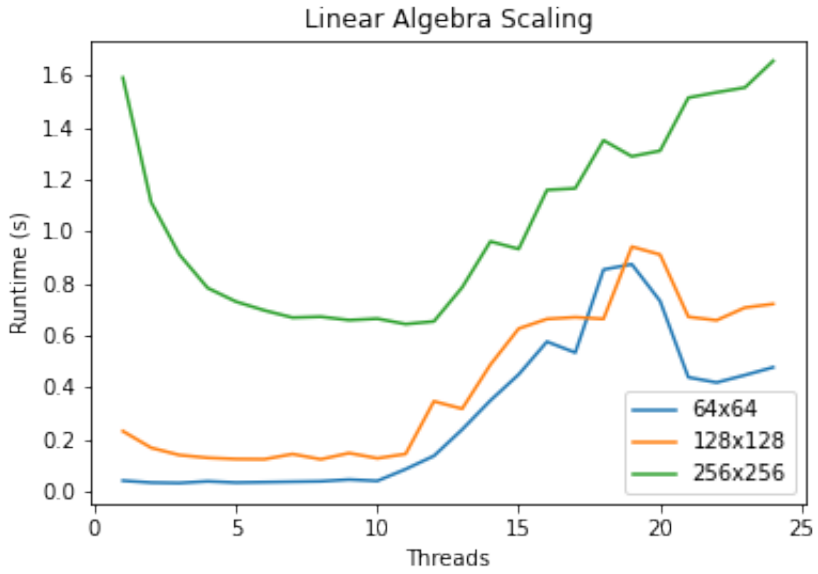



```
[stud07@icsnode20 Project3-code]$ ./main 128 100 0.005
================================================================================
                        Welcome to mini-stencil!
version   :: Serial C++
mesh      :: 128 * 128 dx = 0.00787402
time      :: 100 time steps from 0 .. 0.005
iteration :: CG 200, Newton 50, tolerance 1e-06
================================================================================
--------------------------------------------------------------------------------
simulation took 0.220552 seconds
1514 conjugate gradient iterations, at rate of 6864.6 iters/second
300 newton iterations
--------------------------------------------------------------------------------
```

2. Task: Adding OpenMP to the nonlinear PDE mini-app [50 Points]

```
[stud07@icsnode20 Project3-code]$ ./main 128 100 .005
================================================================================
                        Welcome to mini-stencil!
version   :: C++ OpenMP
threads   :: 5
mesh      :: 128 * 128 dx = 0.00787402
time      :: 100 time steps from 0 .. 0.005
iteration :: CG 200, Newton 50, tolerance 1e-06
================================================================================
```

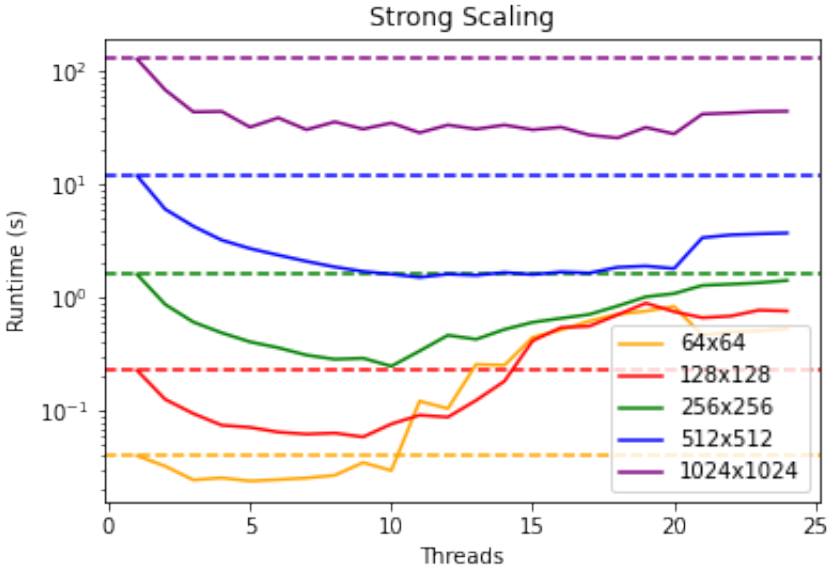
2.1. Linear Algebra Parallelization Results



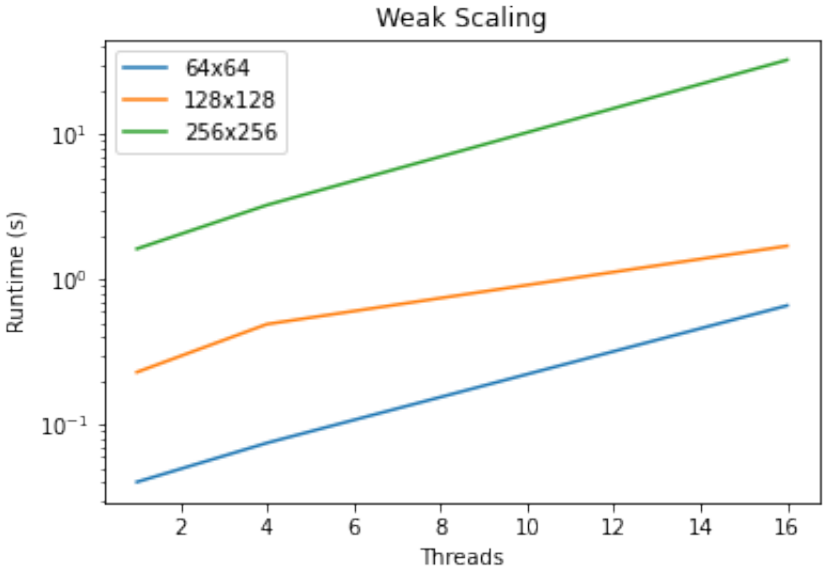
2.2. Kernel Parallelization Results

In order to parallelize the interior points we can simply add a omp parallel for pragma in front of the loop. However, parallelizing the boundary points can be a bit more of a challenge. After trying an omp parallel command in front of each loop and not improving the results, better results were found by making each of the four boundry regions: north, south, east, west, their own openMP section. The improvement wasn't large but added about a thousand iterations per second. One possible reason for the loop parallelization method for the boundaries not improving the runtime is that there were too few points for for the parallel efficiency to overcome increase the overhead added when breaking up the loops.

2.3. Strong Scaling



2.4. Weak Scaling



3. Bonus [5-10 Points]

SIMD instructions could make this computation significantly more efficient because it will allow us to take advantage of instruction level parallelism and run multiple independent computations on the same processor concurrently. This can be done by breaking up each thread into 'blocks,' each corresponding to a specified subset of computations that run in parallel via vectorized SIMD operations.