Cluster architectures and Calculations

Assignment 2

CT-Reconstruction

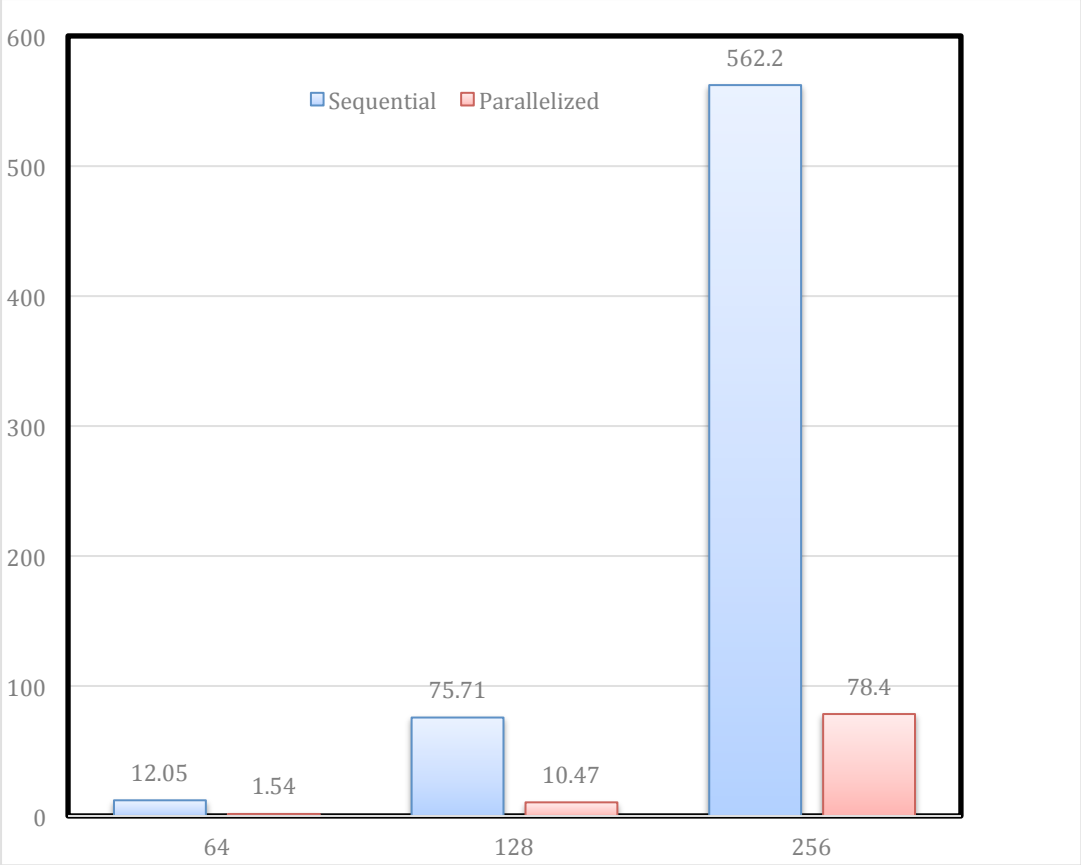
Tzemis Evangelos

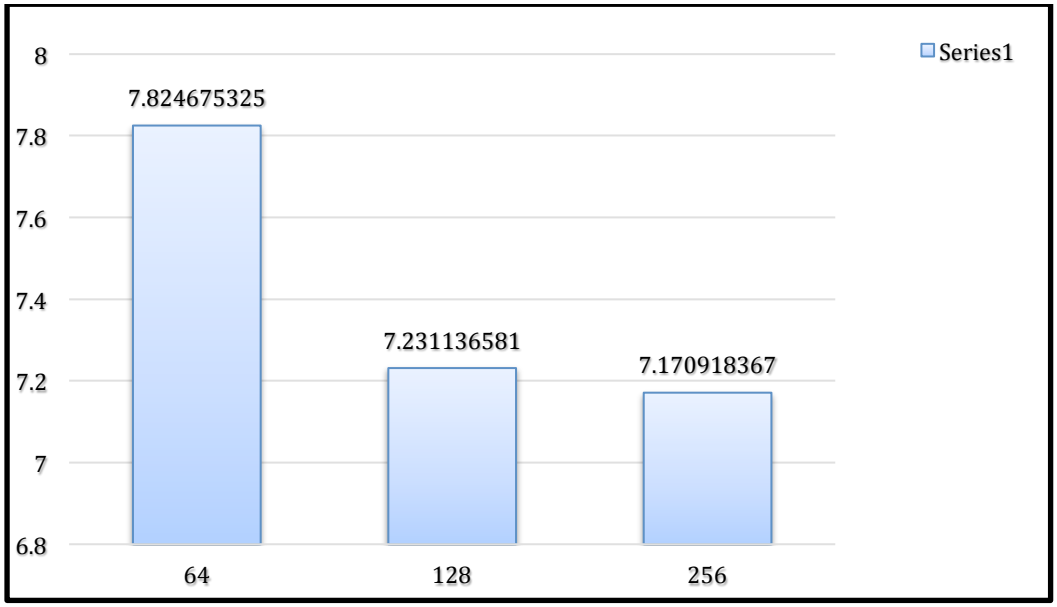
khb107@alumni.ku.dk

Method for parallelizing sequential code

I decided to parallelize the sequential code by parallelizing the outer projection loop. I create 16 processes which they are supposed to work for an amount of projections which is the workload they have. For each projection they also have a loop for all z_voxel pixels. Which for 64 and 128 takes a small amount of time but for 256 it might take longer. In this loop they need to access the result array to add some data to it. So we need to explicit define what is shared and how we are managing conflicts and race conditions.
The thing that processes share is a list of 16 (N,N,N) arrays. In this way each process writes data to its own process-specific array. With this approach we can overcome locking mechanism by using more memory. All other instances are copied when the process is created (because they are not shared) and thus there is no conflict. Finally the arrays in this list are reduced by adding their elements (after all processes have joined) in one single (N,N,N) array which is the final 3d result array. I have tried also dividing in 32 processes which gave different-worse results. I don't think that my solution is the fastest one, since I though about ways of improving the code however for the time I invest on it I think it is ok.

Improvements: another approach I implemented was to share only one (N,N,N) shmarray and use locks to avoid race conditions and conflicts. However I did not found an obvious way of locking specifying elements in an array. By this I mean instead of using a lock when writing to an array use an individual lock for every element of the array. In this way we would have been able to actually use locks whenever two processes would try to access the same index of this array.


600
500
400
300
200
100
0
Sequential
Parallelized
562.2
12.05
1.54
75.71
10.47
78.4
64
128
256


8
7.8
7.6
7.4
7.2
7
6.8
Series1
7.824675325
7.231136581
7.170918367
64
128
256