1 Optimal tile size for matrix multiplication

The purpose of this test is to measure the performance characteristics of Biohadoop for a given problem. The problem is to find an optimal tile size for the tiled matrix multiplication.

A matrix multiplication can be performed in different ways. The most obvious one is the standard algorithm:

```
for i = 1 to n
  for j = 1 to n
    for k = 1 to n
      C(i,j) = C(i,j) + A(i,k) * B(k,j)
```

The matrix multiplication can be improved by loop tiling (!!!cite!!!). The computation is performed on smaller blocks (tiles) of the matrices:

```
for i0 = 1 to n, step blocksize_i
  for j0 = 1 to n, step blocksize_j
    for k0 = 1 to n, step blocksize_i
      for i = i0 to min(i0 + blocksize_i, n)
        for j = j0 to min(j0 + blocksize_j, n)
          for k = k0 to min(k0 + blocksize_i, n)
            C(i,j) = C(i,j) + A(i,k) * B(k,j)
```

If the blocks are small enough, they fit into the L1 CPU cache, which results in a speed up (!!!cite or show result!!!). Theoretically, the optimal block size (in terms of computation speed) can be computed by considering cache and block sizes (!!!example?!!!). But in a real environment, there are many other factors that influence the optimal tile size. One factor is the cache usage by the underlying OS. Another factor is the existence of L2 or L3 caches and their sizes. So we need to find an alternative way to get optimal tile sizes.

An optimization algorithm can be used to find the (near) optimal tile size. In this case, the optimization is done using a GA. Each individual in the population represents a tile size, where the tile size is defined by the values `blocksize_i`, `blocksize_j` and `blocksize_k`. The fitness function is computed as the time it takes to multiply the matrices A and B using a given tile size. The goal is to find an optimal tile size, that minimizes the time to compute a matrix multiplication.

Each GA iteration computes the fitness values of the child population (!!!reference GA in chapter!!!). This fitness values, together with the values of the current population, are then used to determine the next iteration's GA population. Remember, to compute the fitness value for a given tile size, we have to do a matrix multiplication. So we have to compute $population\_size$ matrix multiplications in each iteration, and $iterations * population\_size$ matrix multiplications altogether.

The computation of the fitness function takes the most time in the optimization (!!show data to support this statement!!). It can be parallelized easily, as

different matrix multiplications don't depend on each other. So it is the perfect candidate for parallelization, which is done with Biohadoop.

On startup, each worker receives a full copy of the matrices A and B to minimize network transfers later on. Then, in each GA iteration, the GA distributes the tile sizes of the child population to the workers. The workers compute the matrix multiplication on A and B using the provided tile size. The time spent on this computation is returned to the GA, which considers it as the fitness value for the given tile size.

By distributing a full copy of matrices A and B in the initial phase, the network transmission is minimized during the work phase. In the work phase, only the tile size (Integer) is sent to a worker, which returns the time spent for the matrix multiplication (Long). This way, the network is not the bottleneck when trying to scale out (!!!add some numbers about network usage during work phase, to support this statement!!!).

Every worker runs in a distinct container, and so does the GA algorithm. For this tests, each container consist of 1 CPU and 128MB RAM. In the results shown below, the container for the GA algorithm is excluded, those containers are worker only containers.

1.1 Algorithm parameters

- Matrix size = 128x128
- Min tile size = 1
- Max tile size = 128
- GA population = 50
- GA iterations = 1000
- GA mutation factor = ± 32

1.2 Hardware

Cluster of 6 machines, each machine has the following spec:

- Intel Core2 Duo CPU E8200 @ 2.66GHz (2x2,66Ghz, no hyperthreading)
- 6MB shared L2 cache, 32KB L1 data cache, 32KB L1 instruction cache
- 4GB (2x2GB) DDR2 RAM @ 667MHz

All machines are connected through switched 1GB Ethernet network. The ping latency is about 0,12ms.

1.3 Results

(!!!! WHAT ABOUT WORKERS THAT RUN ON THE SAME MACHINE AS APPLICATIONMASTER?!!!!) The tests were performed using the algorithm settings shown in section 1.1. The hardware specification can be found in section 1.2. The tests differ in the number of used worker containers (and thus in the number of workers), that range from 1 to 18. To get more reliable results, each test was repeated 3 times.

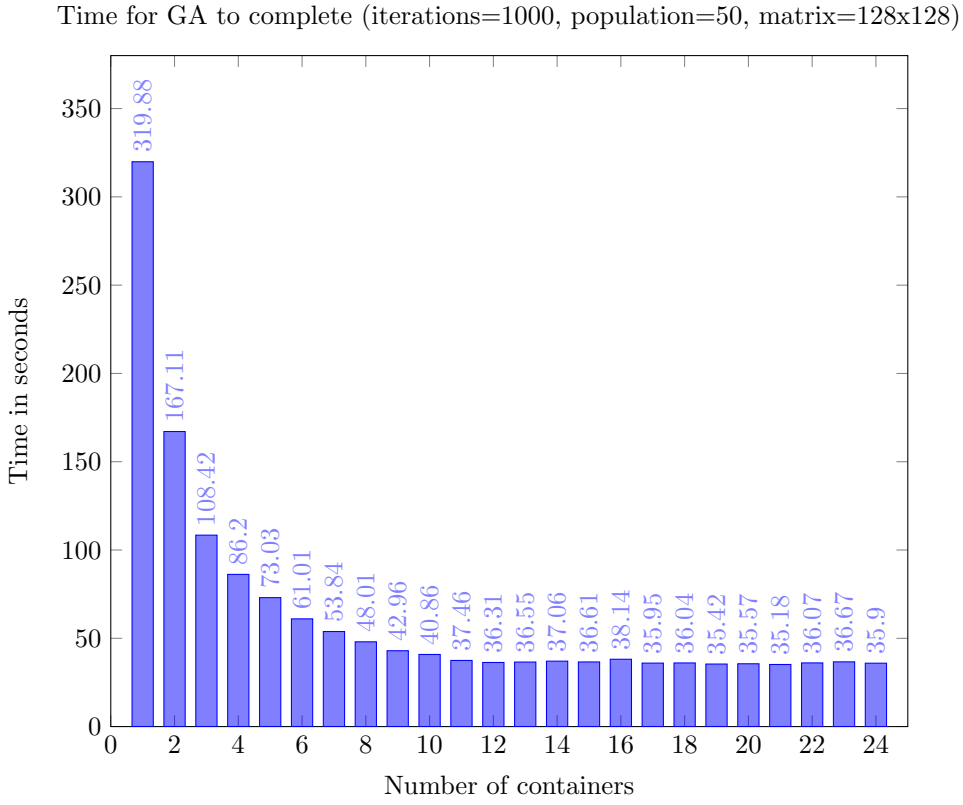

Figure 1: Find an optimal tile size for the tiled matrix multiplication. The x-axis shows the number of containers (workers), the y-axis shows the time it took to finish. iterations=1000, population=50, matrix size=128x128

As we can see from figure 1 and 2, the problem scales up to 12 containers. After that, the performance remains the same, with small fluctuations in both directions (!!!explain reasons for fluctuations: matrix multiplication takes different time for different tile sizes, if during GA iterations bad tiles sizes are chosen, iteration takes longer. other reasons: CPU cache, multiprocess OS, network)!!!). This seems plausible, as the whole cluster provides 12 physical cores. Each container allocates one core, and each matrix multiplication consumes almost 100%

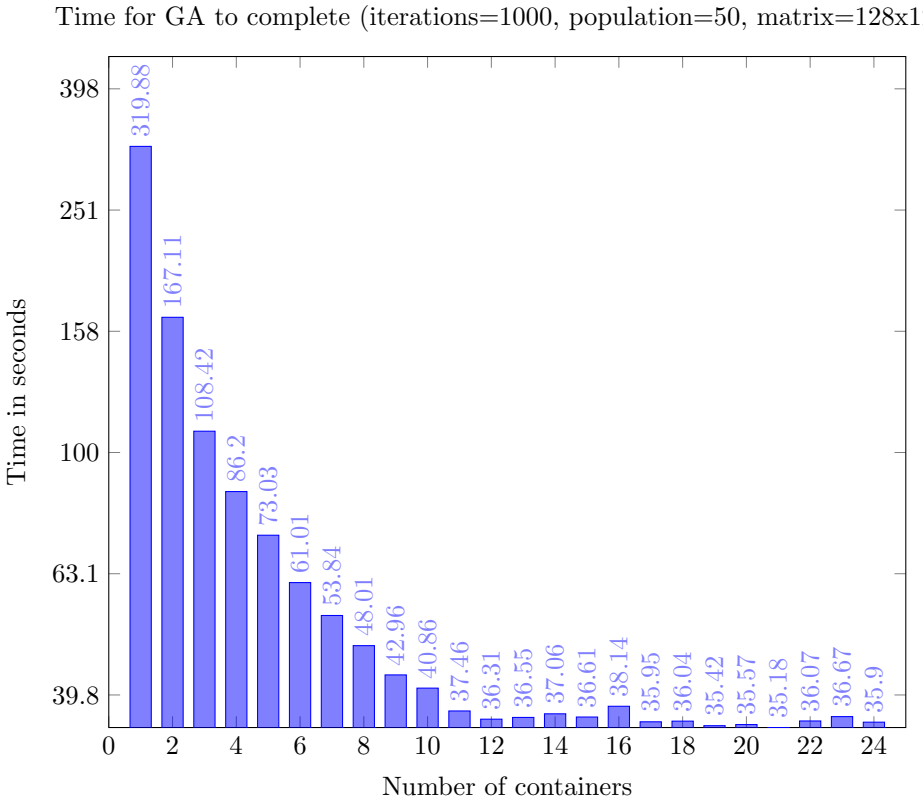

Figure 2: Find an optimal tile size for the tiled matrix multiplication. The x-axis shows the number of containers (workers), the y-axis shows the time it took to finish on a logarithmic scale. iterations=1000, population=50, matrix size=128x128

of a given core (!!! add data !!!). After the 12 cores are fully occupied, each new container has to compete for resources with the other containers, that run on the same machine - the cluster is fully occupied. Figure 2 shows the results 1in logarithmic form - the data is the same as in figure 1.

(!!! compare scale of this test problem with other implementations? !!!)

Reasons for the speed up degradation in the interval from 1 to 12 are (!!!need to prove with data!!!):

- The GA itself needs some time to run. This part of the algorithm runs sequentially (!!!Amdahls Law!!!). The size of the sequential part depends on the GA population size and the matrix size (!!!measure with different population/matrix sizes and compare!!!)
- The GA itself runs on one of the cluster machines, consuming resources (!!!Idea: run Biohadoop on single node and force workers to run on other

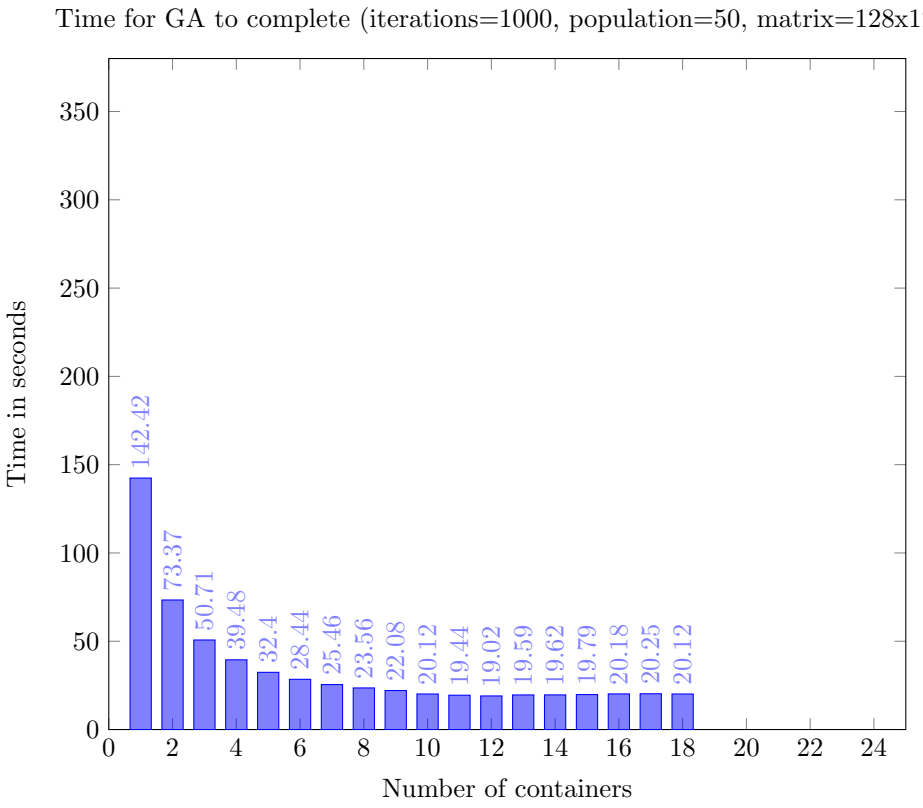

Figure 3: Find an optimal tile size for the tiled matrix multiplication. The x-axis shows the number of containers (workers), the y-axis shows the time it took to finish. iterations=1000, population=50, matrix size=128x128

nodes (use rack affinity); Compare results with this result!!!)

- The communication between the machines takes some time (netty pipeline, OS, network) (!!!to get data, measure echo speed with same messages as matrix mul, but no algorithm overhead!!!)

1.4 TODO

- Find time for sequential part (GA) and parallel part (matrix mul + network). Compute max. speedup. Discuss results and compare with test data
- Discuss speed degradation issues in further detail
- replace figure 2 with a table, showing the speeups?

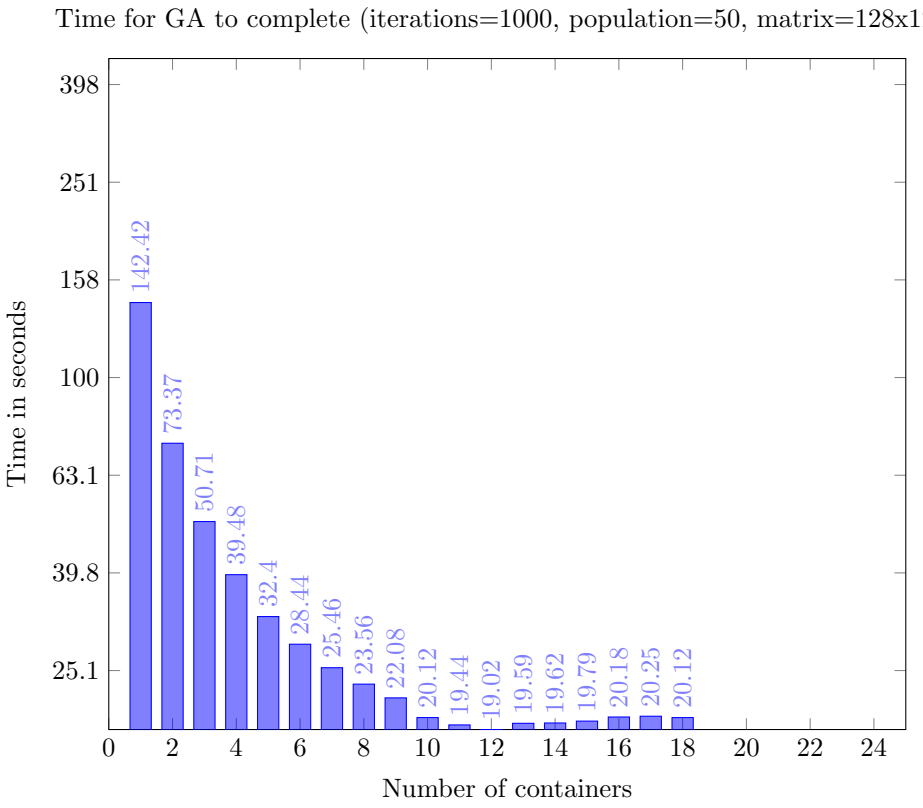

Figure 4: Find an optimal tile size for the tiled matrix multiplication. The x-axis shows the number of containers (workers), the y-axis shows the time it took to finish on a logarithmic scale. iterations=1000, population=50, matrix size=128x128

- ???

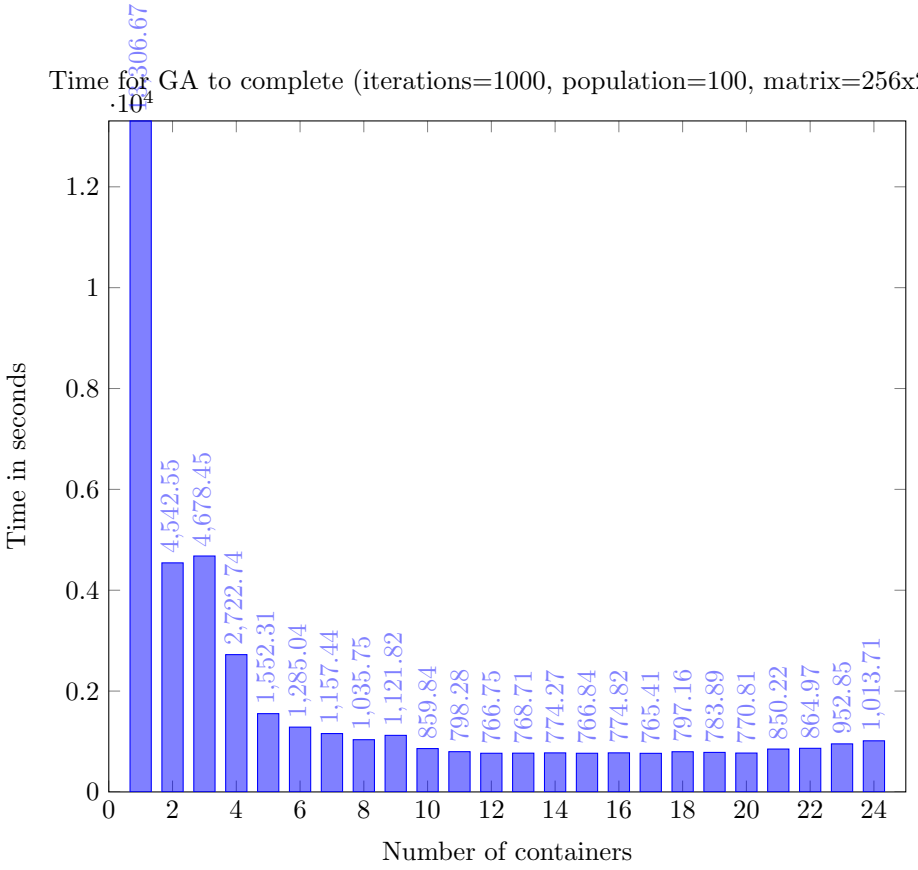

Figure 5: Find an optimal tile size for the tiled matrix multiplication. The x-axis shows the number of containers (workers), the y-axis shows the time it took to finish. iterations=1000, population=100, matrix size=256x256

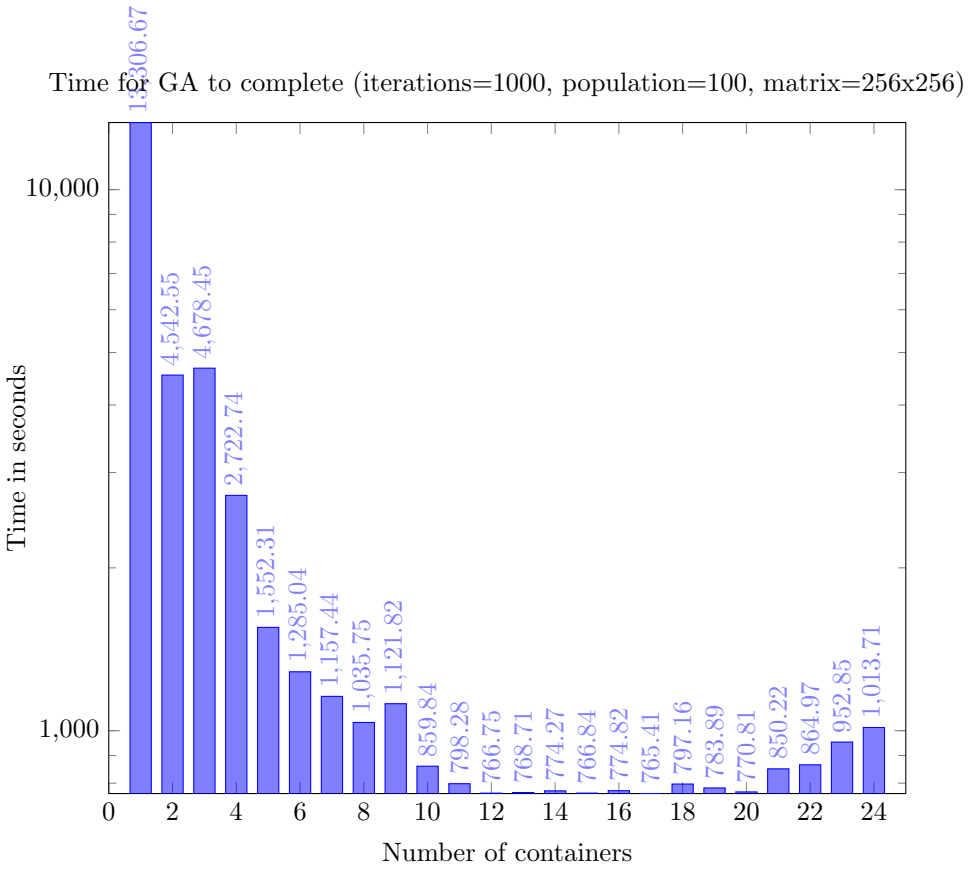

Figure 6: Find an optimal tile size for the tiled matrix multiplication. The x-axis shows the number of containers (workers), the y-axis shows the time it took to finish on a logarithmic scale. iterations=1000, population=100, matrix size=256x256