Chapter 6

Evaluation

Biohadoops purpose is to facilitate the implementation of parallel algorithms on Hadoop. It is expected that the execution time of an algorithm reduces if it is parallelized (assuming the algorithm is suitable for parallelization). As this cannot be guaranteed without proper measurement, this chapter is devoted to the study of Biohadoops speedup characteristics.

Two bio-inspired optimization algorithms are used as benchmarks. Both use Biohadoop and its task system to solve a test problem. The algorithms and test problems are described in section 6.1.

The algorithms are executed on a Hadoop cluster to study how their execution times change when their problem sizes and the number of workers change. The benchmark details can be found in section 6.2, the results are presented in section 6.2.

6.1 Test problems

Both implemented algorithms are part of the GA family. They differ in the number of objectives that they can handle. While NSGA-II is used to solve the MOP in section 6.1.1, a simple GA is used to solve the SOP in section 6.1.2.

6.1.1 ZDT-3

The first implemented optimization algorithm is NSGA-II that is used to find optimal solutions for the Zitzler–Deb–Thiele's function nr. 3 [54]. ZDT-3 is a well known MOP and therefore suited as a test problem. The task is to find an approximation to the optimal Pareto Front, given in figure 6.1.

The implementation uses Biohadoop workers to create and evaluate the offsprings. Simulated Binary Crossover (SBX) and Parameter based mutation [55] are used for the offspring creation. The fitness is computed using the ZDT-3 function. The selection of the fittest individuals for the next population is based on ranking and crowding distance and is performed on the master.

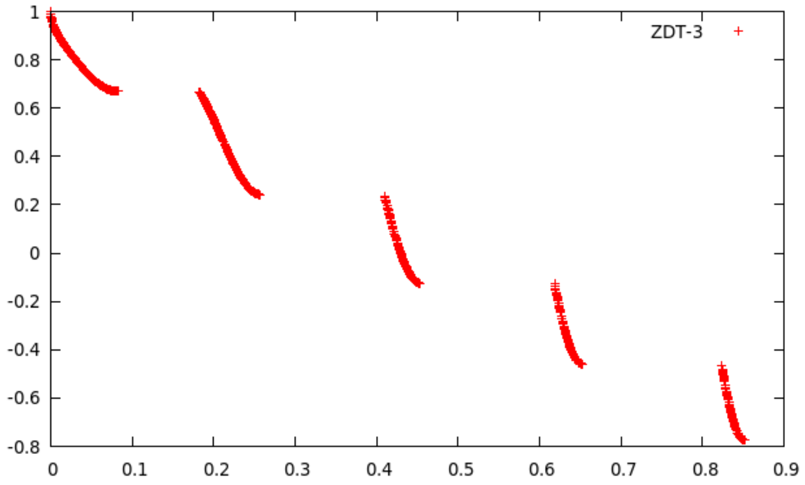

Figure 6.1: Optimal Pareto Front for ZDT-3

6.1.2 Tiled matrix multiplication

The second benchmark implements a GA to solve the SOP of finding optimal tile sizes for the tiled matrix multiplication. The objective is to minimize the execution time for a 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 m
    for k = 1 to l
      C(i,j) = C(i,j) + A(i,k) * B(k,j)
```

The matrix multiplication can be improved by loop tiling [56]. The computation is performed on smaller blocks (tiles) of the matrices:

```
for i0 = 1 to n, step blocksize_i
  for j0 = 1 to m, step blocksize_j
    for k0 = 1 to l, step blocksize_k
      for i = i0 to min(i0 + blocksize_i, n)
        for j = j0 to min(j0 + blocksize_j, m)
          for k = k0 to min(k0 + blocksize_k, l)
            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 speedup. For example, the average of five consequent test multiplications for two matrices of size 1024x1024 took 18.245 seconds for the simple matrix multiplication and 2.048 seconds for the tiled multiplication with tile sizes $i = 16$,

$j = 16$ and $k = 16$. This numbers show that it is appropriate to use the tiled approach for the matrix multiplication. But the speed of the tiled matrix multiplication depends heavily on the tile sizes, the same tiled multiplication as above with tile sizes of $i = 1$, $j = 1$ and $k = 1$ took 22.690 seconds to finish, an increase of more than 10 times compared to a good tile size. Because of the number of possible tile size combinations and the time it takes to execute a matrix multiplication (e.g. matrix size=1024, 2 seconds for a matrix multiplication: $1024^3 * 2 = 2e9$ seconds), it is not feasible to do an exhaustive search for the optimal tile sizes.

An optimization algorithm can be used to find the (near) optimal tile sizes for the different loops. In this case, the optimization is done using a GA. The implementation uses Biohadoops workers to create and evaluate an offspring. For the offspring creation, Simulated Binary Crossover (SBX) and Parameter based mutation are used. The fitness is computed as the time it takes to multiply two matrices using a given tile size. The selection of the fittest individuals for the next population is performed on the master.

6.2 Benchmarks

The task of the benchmarks is to find the speedup characteristics of Biohadoop. The assumption is that the execution time of an algorithm depends on the problem size and the number of workers. To evaluate this assumption, the algorithms presented in section 6.1 are executed with different problem sizes and different numbers of workers. The execution times are measured and used to calculate the speedup using the formula $S = T_S/T_P$, where S is the speedup, T_S is the time for a serial execution and T_P is the time for parallel execution. The results are discussed in section 6.3.

A benchmark is defined by a given algorithm (e.g. NSGA-II) and its settings (e.g. number of workers, number of iterations, etc.). All performed benchmarks have the following settings in common:

- the number of iterations is set to 250
- the population size is set to 100
- the distribution index n_c for the SBX crossover is set to 20
- the distribution index n_m for the mutation is set to 20
- the mutation probability for each offspring value is set to $1/n$, i.e. on average one offspring value is mutated

The test problems have also exclusive settings that only apply to them.

For ZDT-3 this is the genome size. The genome size corresponds to the number of values of an individual and the dimension of the solution space. Each individual is represented by its genome. ZDT-3 can handle any genome size, changing this number influences two properties of the ZDT-3 benchmark. First, increasing the number of genomes also increases the computation effort for the workers that generate new offsprings and compute their fitness. This results from the fact that the workers generate new individuals using the parent genomes and that the ZDT-3 algorithm, used for the fitness computation, loops over all genomes. Second, the genome size influences the amount of data that has to be transferred between the master and the workers. Each worker repeatedly receives two parent individuals and returns an offspring and its computed fitness. The amount of data send between master and workers is therefore related to the gnome size of each individual.

The exclusive setting of the tiled matrix multiplication is the matrix size. It influences the number of computations that need to be performed for a full matrix multiplication and therefore the execution time. In contrast to the first problem, the matrix size has no influence on the amount of data that has to be transferred between the master and the workers. The matrices are part of the "initial data" (see chapter 4.3.3) and therefore transferred exactly once to every worker. The task data consists of two parent individuals that are transferred from the master to the workers to create a new offspring and compute its fitness. The data transferred from a worker to the master contains the offspring and its computed fitness value. Each individual consists of its tile sizes for i, j and k.

The genome size for different ZDT-3 benchmarks is set to 10, 100, 1000 and 10000. The matrix size for the tiled matrix multiplication is set to 128x128 and 256x256. The execution time for each setting is measured for a number of workers that range from 1 to 15. Each of this benchmarks is repeated five times to improve the reliability of the results, making it 300 benchmark runs for ZDT-3 (4 genome sizes * 15 worker setting * 5 repetitions) and 150 benchmark runs for the tiled matrix multiplication (2 tile sizes * 15 worker settings * 5 repetitions).

All experiments were performed on a Hadoop cluster with 6 identical machines. Each machine has the following specifications:

- 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

The machines are directly connected to the same Switch through a 1Gb (Gigabit) Ethernet network.

6.3 Results

The time a Biohadoop application takes to execute is composed of the time Biohadoop needs to start up and the algorithm execution time. The start up begins with Biohadoops submission to Hadoop and ends when the algorithms `run` method is invoked. The algorithm execution time starts with the invocation of the algorithms `run` method and ends when this method returns.

The distinction between start up time and algorithm execution time is made because the main part of the start up time is spent between the application submission to YARN and the beginning of its execution. It is not possible to predict when an application is executed by Hadoop, it depends on different factors like the available cluster resources. To minimize the impact of this uncertainty, the following measurements are based on the algorithm execution time, without the application start up time. The start up time over all benchmarks range from 2.378s to 6.147s, with a median of 3.864s, a 25% quartile of 3.145s and a 75% quartile of 4.258s. The mean was 3.781s. This start up times are close to each other, because the used cluster was completely dedicated to the benchmarks. The start up may take longer when the cluster usage is higher.

Table 6.1 gives an impression how well the benchmark problems are suited to parallelization by showing the maximum theoretical speedup. The theoretical speedup was calculated using the formula $S = T/(T - t_p)$, where S is the speedup, T the algorithm execution time and t_p is the time spent in code parts that are parallelized using Biohadoops task system. T and t_p were taken from the average benchmark times with one worker.

Test problem	Theoretical speedup
NSGA-II, 10 genomes	7.028
NSGA-II, 100 genomes	7.600
NSGA-II, 1000 genomes	12.008
NSGA-II, 10000 genomes	11.124
128x128 tiled mul	81.359
256x256 tiled mul	212.169

Table 6.1: Theoretical speedups

The algorithm execution time results for the benchmarks can be found in the boxplots in figures 6.2, 6.3, 6.4 and 6.5 for the ZDT-3 benchmarks and figures

6.6 and 6.7 for the tiled matrix multiplication. The number of workers is plotted on the x-axis, the algorithm execution time is plotted on the y-axis.

6.3.1 Influence of YARN container placement

The first thing to note when looking at the figures is that the five benchmark times for a given setting (e.g. NSGA-II, 10 genomes) and one worker are very different. The explanation for this effect can be found in the YARN container placement. If a worker container is executed on the same machine as the master container, they communicate without using the physical network. The result is a huge performance gain, as can be seen for example in figure 6.3. In the single worker benchmarks, 4 out of 5 benchmarks executed with both the master and worker container running on the same machine, resulting in a 50% better performance (9,761s average) compared to the fifth benchmark (14.164s) where the master and worker were executed on different machines.

The number of worker containers running on the same machine as the master can also have a negative effect on the execution times. This is especially true if the master is already at the limit of the machines resources and must share them with the workers. An example for this can be found in figure 6.2 for 8 workers. Two worker containers were executed on the same machine as the master during 2 out of 5 benchmarks. The execution time results were 87.977s and 88.014s. In the remaining 3 benchmarks, only one worker container was executed on the same machine as the master, leaving more resources to the master. This results in execution times of 68.423s on average, a difference of more than 20%.

So it depends on the available resources of a machine if the execution of worker containers on the same machine as the master container provides benefits or drawbacks. If a resource like CPU or network is already at its limit, additional worker containers slow the whole Biohadoop execution down. If there are enough resources available, the execution of worker containers on the same machine as the master provides benefits, as the communication between the master and the workers can be performed without network usage.

The location of the YARN containers can currently not be influenced, but discussions by the YARN developers suggest that future versions of YARN will support this feature.

6.3.2 ZDT-3

The next thing to notice are the speedups for the ZDT-3 benchmarks. ZDT-3 is not well suited for parallelization as can be seen from the theoretical speedups in table 6.1, but the results are even worse than expected, with maximum speedups of 1.619 for 10 genomes, 1.513 for 100 genomes, 2.498 for 1000 and 2.479 for

10000 genomes. Figure 6.8 shows the speedup results of the ZDT-3 benchmarks together with the speedups for the tiled matrix multiplication.

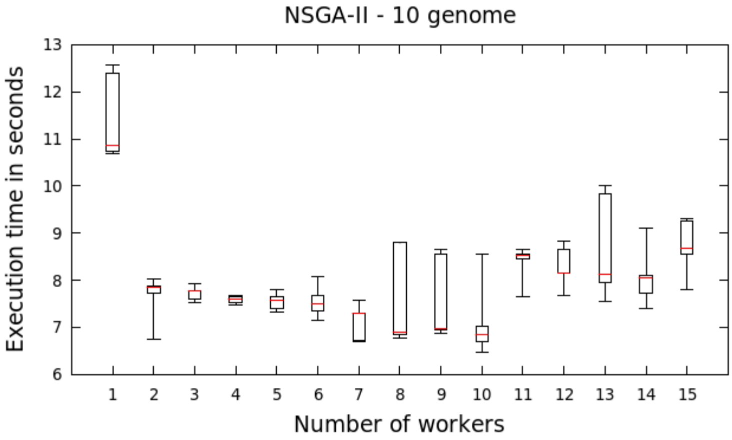

Figure 6.2: ZDT-3 execution times for a genome size of 10

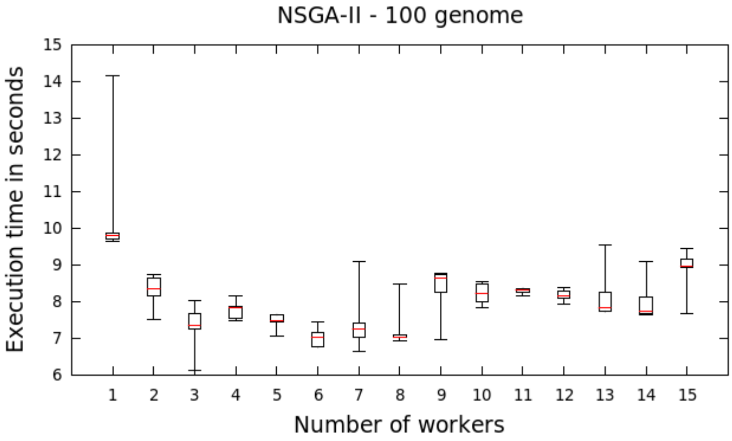

Figure 6.3: ZDT-3 execution times for a genome size of 100

The ZDT-3 benchmarks seem to suffer from the lack of one or more resources (bound by the resources), which prohibits further speedup increases. The investigations show that the ZDT-3 benchmarks are not bound by memory, i.e. memory issues don't slow the execution down. All benchmarks start with 256MB of Java heap memory, which is enough for the containers to execute without causing excessive garbage collections. This was established by using the tool jvisualvm (delivered with Java) for the ZDT-3 benchmark with 10000 genomes. The memory usage for the master container is at about 100MB to 150MB, the

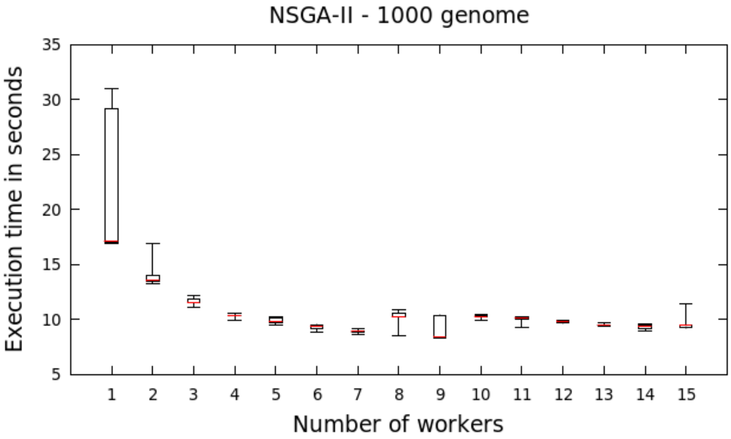

Figure 6.4: ZDT-3 execution times for a genome size of 1000

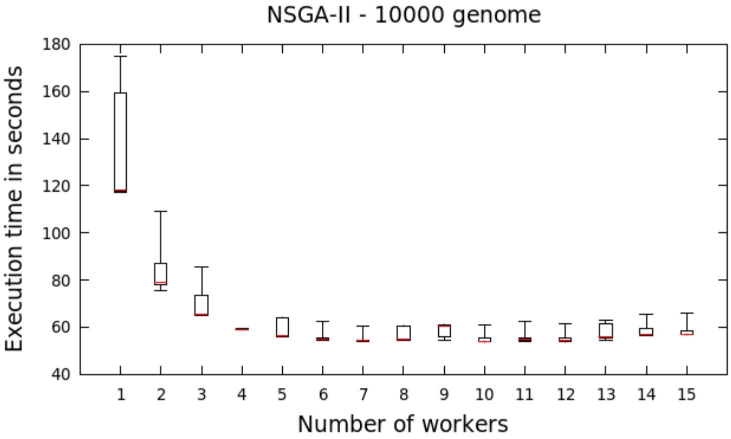

Figure 6.5: ZDT-3 execution times for a genome size of 10000

GC activity, a good indicator for memory problems, ranges from 3% to 5.5% of the CPU time, with an average of 3.8%. The memory usage for a worker is even lower and lies in the range of 5MB to 30MB. This numbers show no significant memory problems.

The next step is to investigate the network performance. Calculations give a first hint to understand if the bad speedups can be explained with the saturation of the 1Gb network (a small "b" denotes bits, a big "B" denotes bytes, e.g. 1Gb = 1 gigabit, 1GB = 1 gigabyte). In each benchmark, 250 iterations on 100 individuals are performed, resulting in 25000 tasks. The tasks are send from the master to the workers and the workers return the results. The task data send from the master to the worker contains two individuals. Each individual

genomes	data (Mb)	theoretical transfer time (s)	fastest algorithm execution time (s)
10	32	0.032	7.072
100	320	0.32	7.031
1000	3200	3.2	8.910
10000	32000	32	55.475

Table 6.2: Amount of network data sent from master to workers, theoretical transfer time and fastest algorithm execution

consists of its genome, where each value in the genome is of type `double` (8 bytes). For a genome size of 10, this makes 2 (parents) * 10 (genomes) * 8 (bytes) * 25000 (tasks) = 4000000 bytes (4MB) or 32Mb of data that needs to be transferred from the master to the workers during the benchmark. The size of the results send from the workers to the master is about the half, as it consists of an individual (the offspring) and its fitness (the fitness is composed of two `double` values). This fact allows to use the outgoing data amount as upper bound for the network usage: if the outgoing data rate doesn't exceed the network bandwidth, this will be true also for the incoming data. Table 6.2 shows the results for all genome sizes together with the best algorithm execution times. One can see from the table that the benchmark data can be transferred on the 1Gb Ethernet network during the according fastest algorithm execution time.

Additional experiments were performed to improve the confidence in the calculations and to establish the true achievable data rate for the network, given different message sizes. The experiments measure the peak network bandwidth using a small Java program and iftop. The Java program uses the same communication techniques as Biohadoop (Netty + Kryo) and performs repeated request / response cycles between a master and several workers. The exchanged messages consist of 20, 200, 2000 or 20000 `double` values, corresponding to two parent individuals in the according ZDT-3 benchmarks. The resulting peak bandwidth was 134Mb/s for 20, 489Mb/s for 200, 901Mb/s for 2000 and 552Mb/s for 20000 `double` values. The CPU on the master was the limiting factor for 20, 200 and 20000 `double` values. For 2000 `double` values, the network was saturated at 901Mb/s and therefore the limiting factor. No studies were performed to explain why the experiments delivered the best results with 2000 values as this lies out of the scope of this thesis.

One phenomena regarding the network bandwidth needs further investigation. The above measurements show a peak data rate of 552Mb/s for the case of 20000 `double` values. If this data rate is taken as base for the ZDT-3 benchmark with

10000 genomes, one can calculate that more than 55s are needed to exchange 32Gb of data over the network between the master and its workers (32000Mb / 552Mb/s = 57.97s). The explanation can be found once more in the YARN container placement. The 552Mb/s peak bandwidth is the data rate that is send through the Ethernet port to the cluster, but worker containers that run on the same machine as the master don't use this port for communication. Instead, they communicate through the local interface. iftop showed an additional combined data transfer rate of 400Mb/s (send and receive data rates are added) on the local interface when workers were running on the same machine as the master. This gives an aggregated peak data rate of 700Mb/s - 800Mb/s for the outgoing traffic, which is fast enough to transmit 32Gb of data in less than 55s. The execution times were higher in cases where no workers executed on the same machine as the master.

The calculations and additional experiments show that the network is fast enough to transfer the ZDT-3 benchmark data. The reason for the bad ZDT-3 speedup results lie elsewhere.

This leads to the assumption that the benchmarks are CPU bound which was confirmed through observations of the CPU usage of the master. In the case of 10 and 100 genomes the CPU limit was reached by the master with two workers, for 1000 and 10000 genomes the limit was reached with four workers.

The high CPU utilization is caused by two effects: the first one is the object serialization/deserialization overhead that ranges between 30% to 40% for genome sizes of 10 and goes up to 60% to 70% for a genome size of 10000. Small genome sizes mean a high rate of exchanged messages and serializations/deserializations. Large genome sizes reduce the rate of exchanged messages but increase the amount of work for a single serialization/deserialization.

The second effect is a direct consequence of computational small worker tasks like in the case of 10 to 100 genomes: the master performs (beside the communication aspects) the algorithms for ranking and crowding distance. The workers return their results fast as the computation is not intense. The master has to compute therefore the ranking and crowding distance at short intervals. This results in a CPU utilization of about 25% to 30% only for this computations.

In conclusion, the ZDT-3 benchmarks are CPU bound by the master due to the small computational effort on the workers and the resulting fast exchange of many small messages. Increased genome sizes provide better speedup results, but are again limited by the CPU of the master, as they have higher demands for object serialization/deserialization. The performance of the 1Gb network and the available memory are sufficient to not slow down the ZDT-3 benchmarks.

6.3.3 Tiled matrix multiplication

The theoretical speedups for the tiled matrix multiplication promise better results (see table 6.1) as matrix multiplications are compute intense and clearly dominate the algorithm execution time. Figure 6.6 and 6.7 show the execution times. One can see that the execution times decrease with the number of workers. This scales until 12 workers, after which the execution times remain the same or increase slightly. The reason for this is that the cluster offers 12 CPU cores in total. When all cores are fully utilized, which happens with 12 workers, additional workers have to share CPU resources. This negatively impacts the execution times. So the tiled matrix multiplication is CPU bound by the workers.

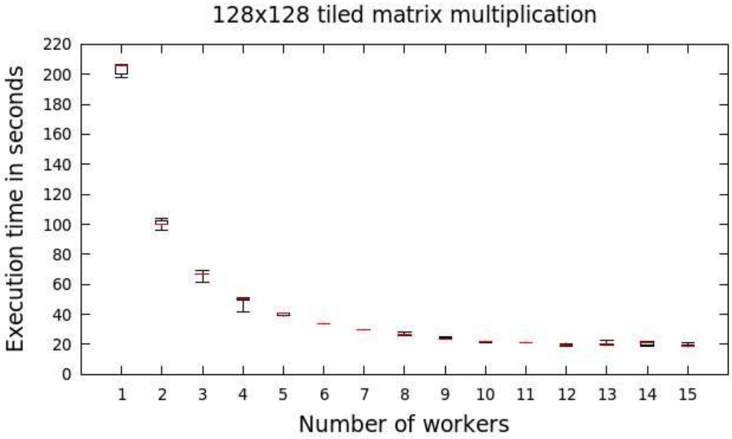

Figure 6.6: Tiled matrix multiplication execution times for a matrix size of 128x128

An additional advantage of the tiled matrix multiplication benchmarks over the ZDT-3 benchmarks is the small amount of data that needs to be transmitted. Like in the ZDT-3 benchmarks, each task data consists of two parents that are sent from the master to the worker, the result is an offspring with its fitness value. In contrast to ZDT-3, where an individual consists of a number of `double` values according to its genome size, a tiled matrix multiplication individual consists of the tile sizes for the i, j and k loop. Each of them is a single `integer` with 4 bytes. The total amount of data that needs to be transmitted from the master to the workers is therefore $2 * 3 * 4 * 25000 = 600000$ bytes or 4.8Mb. Together with the computational intense tasks of matrix multiplication on the workers (leads to lower network usage) and the absence of time consuming ranking and crowding distance algorithms on the master, this provides speedups of up to 10.507 for 128x128 matrices and 7.961 for 256x256 matrices.

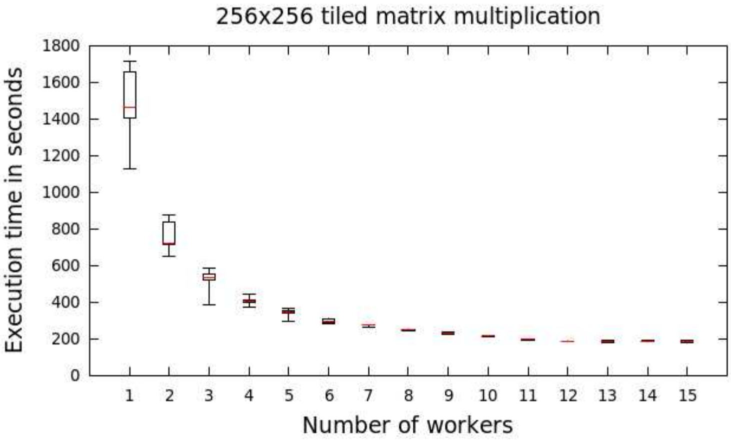

Figure 6.7: Tiled matrix multiplication execution times for a matrix size of 256x256

The reason for the better performance of the 128x128 benchmark over the 256x256 benchmark is unknown. A possible explanation is that the tile sizes are taken from a bigger range (256 instead if 128) which makes it more likely that bad tile sizes are chosen. This is although pure speculation.

6.3.4 Speedups

Figure 6.8 depicts the speedups for all test problems with respect to increasing worker sizes. The ZDT-3 benchmarks show poor results. This is not surprising as the maximum theoretical speedups of this problems are small (see table 6.1) and the communication overhead is bigger compared to the tiled matrix multiplication. The only unexpected outcome was that the benchmarks scale very bad with a maximum speedup of 2.498 for 1000 genomes. The reason is that the ZDT-3 benchmarks are CPU bound by the master, as the investigations in section 6.3.2 show.

The tiled matrix multiplications demonstrate better results, the maximum speedup was 10.507 for a matrix size of 128x128. In this case, the speedup grows near linear or even slightly better than linear with the number of workers. That a speedup is better than linear is usually suspicious but can be explained by the fact that each benchmark was repeated five times and the average times of this five executions were taken to compute the speedups. Five executions seem to be to small to get smooth results, especially when taking into account that the YARN container placement has big influences on the execution times.

The speedups for the 128x128 tiled matrix multiplication increase until a worker size of 12 is reached. At this point, no more improvements are achieved. The reason for this is the limited number of CPUs in the cluster.

The speedup for the 256x256 tiled matrix multiplication benchmark is worse compared to the 128x128 tiled matrix multiplication, although it also grows nearly linear until 12 workers.

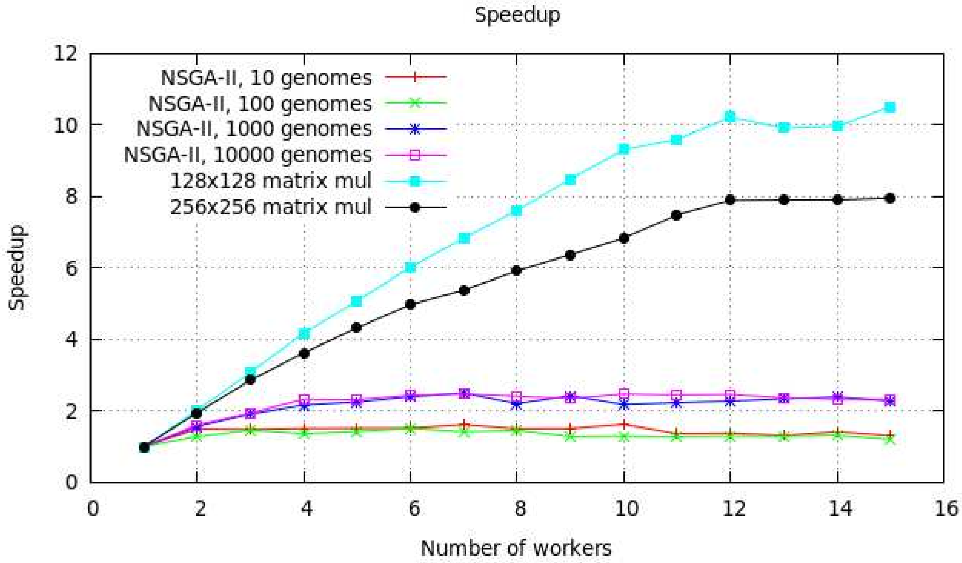

Figure 6.8: Speedups for ZDT-3 and tiled matrix multiplications