Scalable Distributed Genetic Algorithm for Data Ordering Problem with Inversion Using MapReduce

Doina Logofatu[1] and Daniel Stamate[2]

[1] Computer Science Department of Frankfurt University of Applied Sciences 1 Nibelungenplatz 60318, Frankfurt am Main, Germany
[2] Department of Computing, Goldsmiths College, University of London, London SE146NW, UK
logofatu@fb2.fh-frankfurt.de

Abstract. We present in this work a scalable distributed genetic algorithm of Data Ordering Problem with Inversion using the MapReduce paradigm. This specific topic is appealing for reduction of the power dissipation in VLSI and in bioinformatics. The capacitance and the switching activity influence the power consumption on the software level. The ordering of the data sequences is an unconditional consequence of switching activity. An optimization problem related to this topic is the ordering of sequences such that the total number of transitions will be minimized – Data Ordering Problem (DOP). Adding the bus-invert paradigm, some sequences can be complemented. The resulting problem is the DOP with Inversion (DOPI). These ordering problems are NP-hard. We establish a scalable distributed genetic approach - MapReduce Parallel Genetic Algorithm (MRPGA) for DOPI, MRPGA_DOPI and draw comparisons with greedy algorithms. The proposed methods are estimated and experiments show the efficiency of MRPGA_DOPI.

Keywords: Data Ordering with Inversion, Low Power, Distributed Algorithm, Evolutionary Approaches, Transition Minimization, Greedy, MapReduce, Apache Hadoop.

1 Background and Motivation

Quicker access time and bigger storage capacity are actual demands for electronic devices ([10]). That's why the low-power topics have to be considered at the beginning of the design process ([11], [12], [14], [17]). The software component for the design of embedded systems is of importance for the power consumption of the circuit, since its optimization could lead to significant improvements ([12], [14]).

Often, the design complexity describes directly the power consumption of a circuit. It is a fact that nowadays the power consumption has grown, bringing also more complex design approaches. Reliability, cooling costs, packaging, and computer battery life are some domains with positive effects regarding a design based on low power consumption. For handling the power management issue, new directions and approaches are necessary. The capacitance and the switching activity influence the (software level) power consumption. An important design metric, the switching

L. Iliadis et al. (Eds.): AIAI 2014, IFIP AICT 436, pp. 325–334, 2014.

activity, characterizes the quality of an embedded system-on-chip design. The switching activity proves to be an important design metric, which describes the quality of an embedded system-on-chip design. It is directly related to the ordering of data strings. One first problem related to this topic is the ordering of data words for minimizing the total number of transitions *Data Ordering Problem* (DOP). This problem could be of interest for various practical situations, e.g. reordering and optimize DNA sequences or in the linguistic leveling [4, 7].

2 Problem Domain

Definition 2.1. *Hamming distance.* If a word w_r of length k is followed by the word w_s of the same length k, the total number of transitions is given by the number of bits that change. This is:

$$d(w_r, w_s) = \sum_{j=1}^{k} w_{rj} \oplus w_{sj} \tag{1}$$

also known as the *Hamming distance* between w_r and w_s. Here, the w_{rj} denotes the j^{th} bit of w_r, and $\oplus$ the XOR operation. For instance, *d(1010, 0100) = 3*. The number of transitions is, in fact, the *Hamming distance* between w_r and w_s.

Definition 2.2. *Total number of transitions.* The *total number of transitions* is the sum of number of transitions needed for the transmission of all the words w_1, w_2, …, w_n. It is denoted with N_T. If σ is a permutation of {1, 2, …, n}, then the total number of transitions will be:

$$N_T = \sum_{j=1}^{n-1} d(w_{\sigma(j)}, w_{\sigma(j+1)}) \cdot \tag{2}$$

Definition 2.3. *Phase-assignment.* The *phase-assignment* (polarity) δ is a function defined on {1,…, n} with values in {0, 1}, which specifies if a word is sent as it is or complemented (negated, inverted).

For example, the complemented word of 10011 is 01100. That is, if $\delta(i)$=0 then the word w_i is sent as it is, otherwise is sent his complement.

Definition 2.4. The Data Ordering Problem *(DOP)* is defined as the problem of finding a permutation σ of the words w_1, w_2, …, w_n such that the total number of transitions:

$$N_T = \sum_{j=1}^{n-1} d(w_{\sigma(j)}, w_{\sigma(j+1)}) \tag{3}$$

is minimized.

Definition 2.5. The *Data Ordering Problem with Inversion (DOPI)* is defined as the problem of finding a permutation σ and a phase-assignation δ of the words $w_1, w_2, \ldots, w_n$ such that the total number of transitions:

$$N_T = \sum_{j=1}^{n-1} d(w_{\sigma(j)}^{\delta(j)}, w_{\sigma(j+1)}^{\delta(j+1)}) \tag{4}$$

is minimized.

The problems DOP and DOPI are NP-hard [11, 12].

Note that the Hamming distance between two words with the same length provides the total number of dissimilarities between these words.

We denote the generalizations of these problems also with DOP and DOPI. Given a set of n words with the same length k over an alphabet $\mathcal{A}$, and in the case of DOPI also a permutation of length k which denotes the inversion (complementation) function, it is requested to provide a permutation of the given words (in the case of DOPI also an assignation of them) which minimizes the total number of dissimilarities (transitions).

3 Previous Work

The optimal method and two evolutionary approaches, MUT_DOPI (with mutation) and GA_DOPI (hybrid genetic algorithm) are proposed in [6], together with three new genetic operators Simple Cycle Mutation (SIM), Cycle OX and Cycle PMX. The two ordering problems are very similar to the *Traveling Salesman Problem* (TSP). For DOP, DOPI and TSP a efficient ordering of elements regarding to a given weight has to be determined. Since the two problems DOP and DOPI are NP-hard [11, 12], the optimal algorithms can only manage small instances. In the past years some approaches were introduced for the problems DOP and DOPI:

Spanning Tree/Minimum Matching (ST-MM) [11]
Double Spanning Tree (DST) [11]
Greedy Simple (GMS) [12]
Greedy Min (GM) [12]
Evolutionary Approaches [6]

The *Evolutionary Algorithms* (EAs) provide the best results quality. Such evolutionary methods return better outcome than the above-presented *Greedy Min*, but with much more time resources. Further, they have no capability to cope with very large data sets. EAs which produce high-quality optimizations are presented in [6], where are presented evolutionary algorithms for both DOP and DOPI.

4 Scalable Distributed Algorithm Using MapReduce

Our scalable distributed evolutionary algorithm uses the genetic algorithm GA_DOPI [6] and is realized with *Hadoop*. Apache [18] proposes Hadoop as an OpenSource

MapReduce [3] framework implementation. Hadoop is a batch data processing system for running applications and process large amounts of data in parallel, in a reliable and fault-tolerant manner on large clusters of compute nodes, usually running on commodity hardware. This system offers monitoring and status tools and provides an abstraction model for programming tasks. It supports automatic distribution and parallelization. Apache's *Hadoop* offers a distributed file system (*HDFS*) which creates multiple duplicates of data sets and distributes them on compute nodes throughout the cluster to enable reliable, rapid computations. The nodes for computation and for storage are usually the same. The application specifies the input/output locations, supply *map* and *reduce* methods and possibly invariant data.

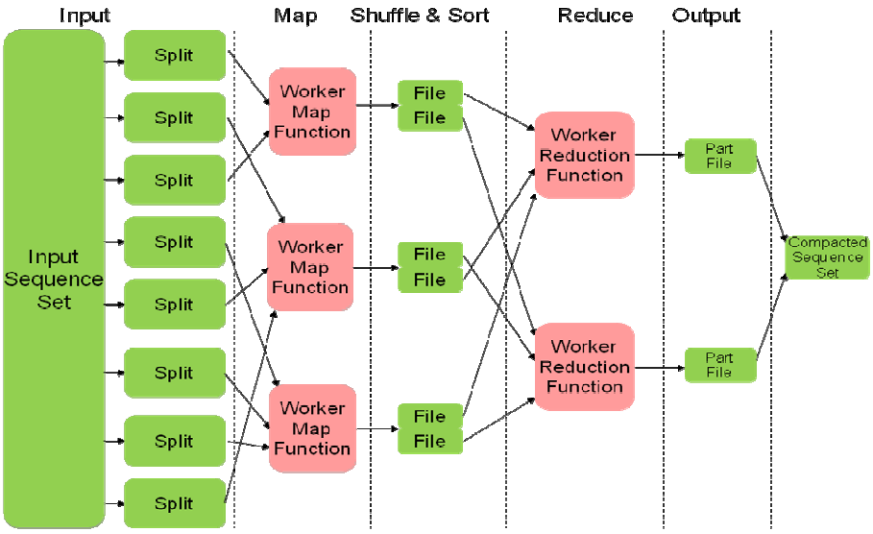


Fig. 1. MapReduce Dataflow

The scalable distributed evolutionary algorithm performs for a given number of generations. During every generation, it creates an arbitrary number of permutations of the current set of sequences, splits the set of permutations in subsets (*map* operation) and performs GA_DOPI [6] on all of them during the *reduce* operation. Lastly, it collects the best permutations (individuals) provided on each reducer node in the cluster and stick them together. Groups of initial individuals are initialized using the greedy approaches GMS and GM [7, 12]. The outcome represents the set of sequences for the next generation. The *map* operation associates for each given permutation a subset index in range 0 to *reductionFactor*−1 randomly, as output key. During the *Shuffle & Sort* action, the sequences get sorted after their indices, and each *reducer* node gets and executes GA_DOPI in one call over its subset of sequences with the same index.

```
ALGORITHM_ MRPGA_DOPI
    Initialization(size_of_population, rate_crossover, rate_mutation)
    Initialization(factior_reduction)
    Initialisation_GreedyMinSimplified_individuals()
    Initialization_GreedyMin_individuals()
    Initialization_Random_individuals()
    def MRPGA_MAP(initialIndex, pair<permutation, assignation>) = {
        reductionSetIndex = random(0.. reductionFactor−1)
        context.write(reductionSetIndex, pair<permutation, assignation>)
    }
    def MRPGA_REDUCE
            (reductionSetIndex, Iterable<pair<permutation, assignation>>)=
      {
          run_crossover(number_crossovers);
          run_mutation(number_mutatios);
          get_fitness(all_new_individuals);
          removal_WorstInviduals (number_crossovers+number_mutations);
          for (pair<new_perm, new_assign>: all_new_individuals)
                  context.write
                      (reductionSetIndex,pair<new_perm, new_assign>)
      }
      for( j← 1; j ≤ number_Generations; step 1)
          job← NewJob(MRPGA_MAP, MRPGA_REDUCE)
          job.configuration.set(populationHDSFPath)
          job.configuration.set(crossoverRate)
          job.configuration.set(mutationRate)
          job.configuration.setNumberReduceTasks(reductionFactor)
          job.submit_and_wait()
          collect_new_population
      end_for
      return bestIndividual
  END_ALGORITHM_MRPGA_DOP
```

Fig. 2. Pseudocode for MRPGA_DOPI

5 Realization Specifics

The Hadoop implementation for DOPI is the *Open Source* project “dopiSolver” on *sourceforge.net*. “dopiSolver.jar” contains the java binary application, as well as all the libraries depends on. The jar’s main method (*dopisolver.DopiSolverDriver*) submits the DopiSolver's Hadoop Job (*dopisolver.DopiSolverJob*). I order to launch the application use "hadoop jar dopiSolver.jar" and supply afterwards the solvers name: *dopi*.

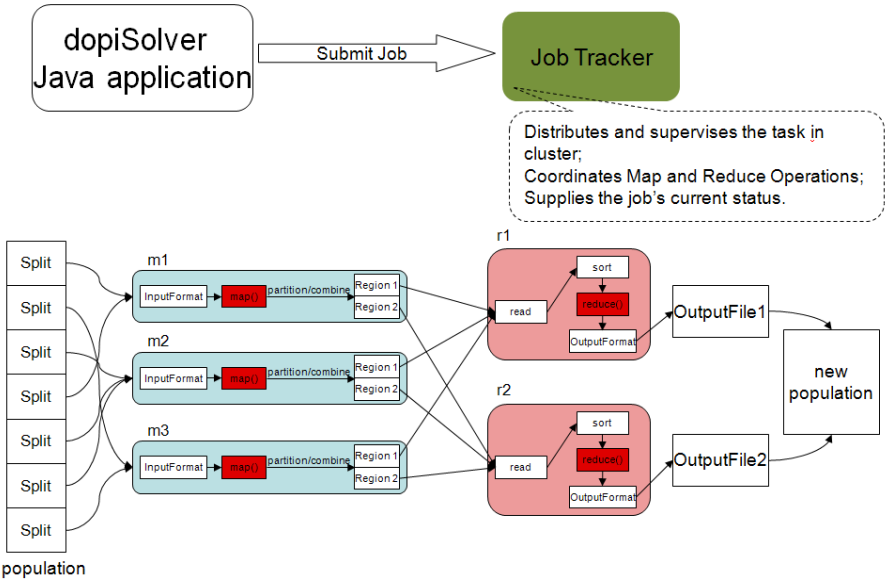


Fig. 3. Implementation of dopiSolver using Java and Apache's Hadoop

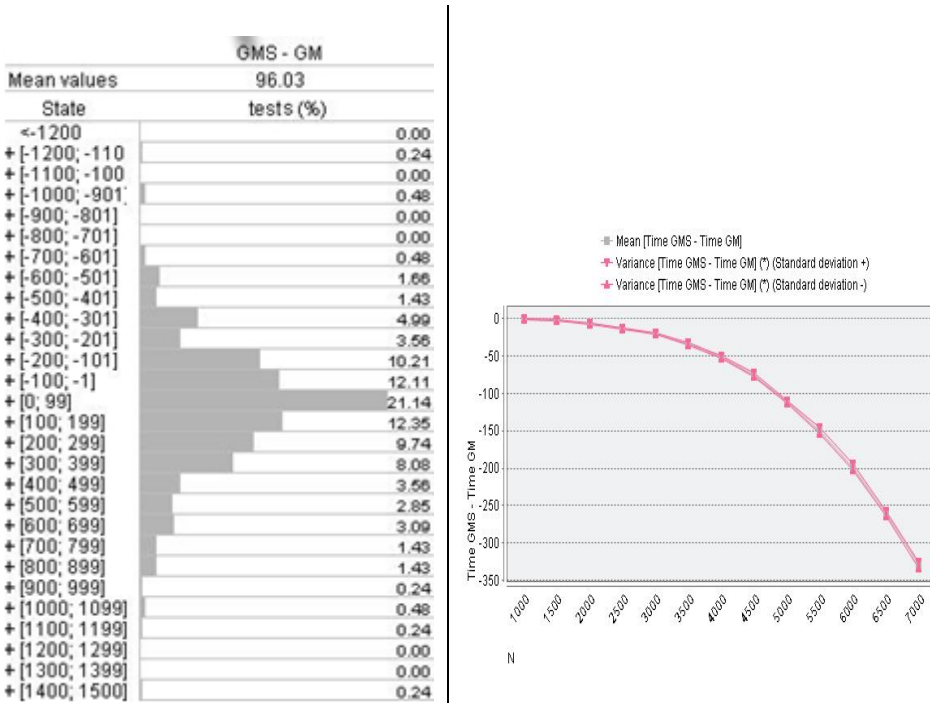
With "hadoop jar dcpSolver.jar dopi -h" you get the description of the application supported options. The source code is available on *sourceForge* under

https://dopisolver.svn.sourceforge.net/svnroot/dopisolver

(Subversion repository). The application depends on the libraries JUnit 4 (for test units), Apache Commons CLI2, and Apache Commons Math Version 2.0.

6 Experimental Results

Various sets of data were generated artificially with different parameters, like e.g. sequences set dimension (n, that is the number of words), length of the words (k). We start by performing the two greedy algorithms GM and GMS [6] on 421 different artificial generated data sets with $1000 \leq n \leq 7000$, $50 \leq k \leq 1000$. We do some experiments with the Greedy methods because we use them also in the initialization of the distributed algorithm. In Figure 5 we show the mean values for the difference GMS–GM on intervals and the mean (standard deviations $\pm$) for the execution time for all these experiments.



	GMS - GM
Mean values	96.03
State	tests (%)
<-1200	0.00
+ [-1200; -110	0.24
+ [-1100; -100	0.00
+ [-1000; -901	0.48
+ [-900; -801]	0.00
+ [-800; -701]	0.00
+ [-700; -601]	0.48
+ [-600; -501]	1.66
+ [-500; -401]	1.43
+ [-400; -301]	4.99
+ [-300; -201]	3.56
+ [-200; -101]	10.21
+ [-100; -1]	12.11
+ [0; 99]	21.14
+ [100; 199]	12.35
+ [200; 299]	9.74
+ [300; 399]	8.08
+ [400; 499]	3.56
+ [500; 599]	2.85
+ [600; 699]	3.09
+ [700; 799]	1.43
+ [800; 899]	1.43
+ [900; 999]	0.24
+ [1000; 1099]	0.48
+ [1100; 1199]	0.24
+ [1200; 1299]	0.00
+ [1300; 1399]	0.00
+ [1400; 1500]	0.24

Fig. 4. (a) Allocation on intervals of the difference GMS–GM within 421 experiments with $1000 \le n \le 7000$, $50 \le k \le 1000$ (*b*) Mean and variance for the execution time difference (GMS-GM)

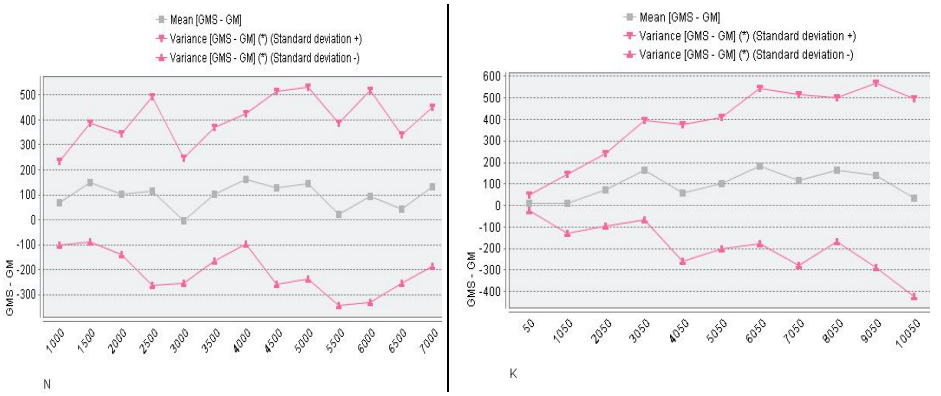
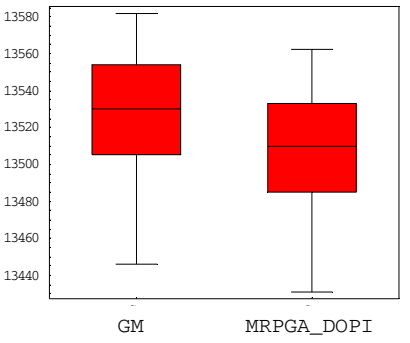
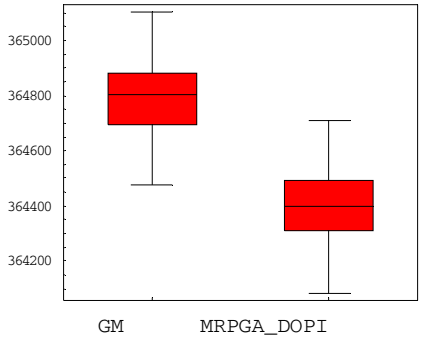


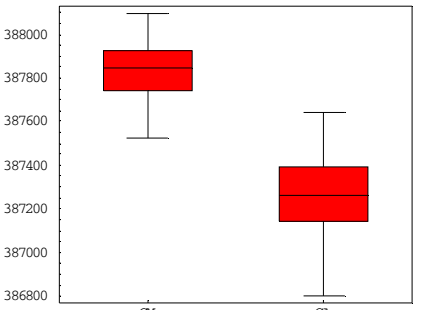
Fig. 5. Mean and variance for the difference GMS-GM for (*a*) values of *n* and *b)* values of *k*



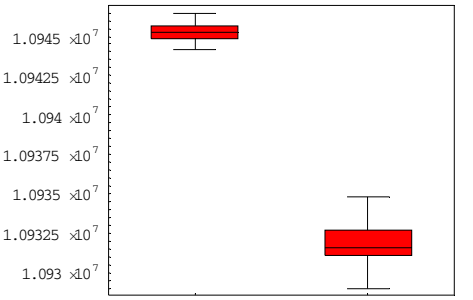
(a) 50 runs, n=100, k=200



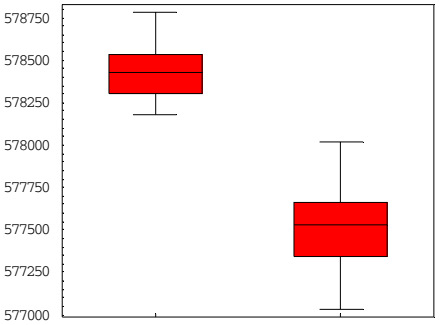
(b) 50 runs, n=100, k=5000



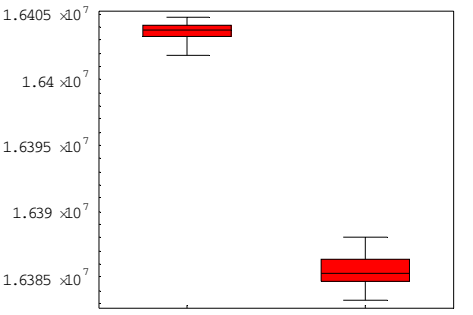
(c) 50 runs, n=3000, k=200



(d) 50 runs, n=3000, k=5000



(e) 50 runs, n=4500, k=200



(f) 50 runs, n=4500, k=5000

Fig. 6. Statistical boxplots with values for GM and MRPGA for 50 runs of the algorithms GM and MRPGA with n=100, 3000, 4500 and k = 200, 5000.

The mean values and student T-Tests illustrate the quality of results using MRPGA algorithm versus GM. Comparison in pairs (GM vs. MRPGA) concludes the null hypothesis rejection with significance level = 0.001.

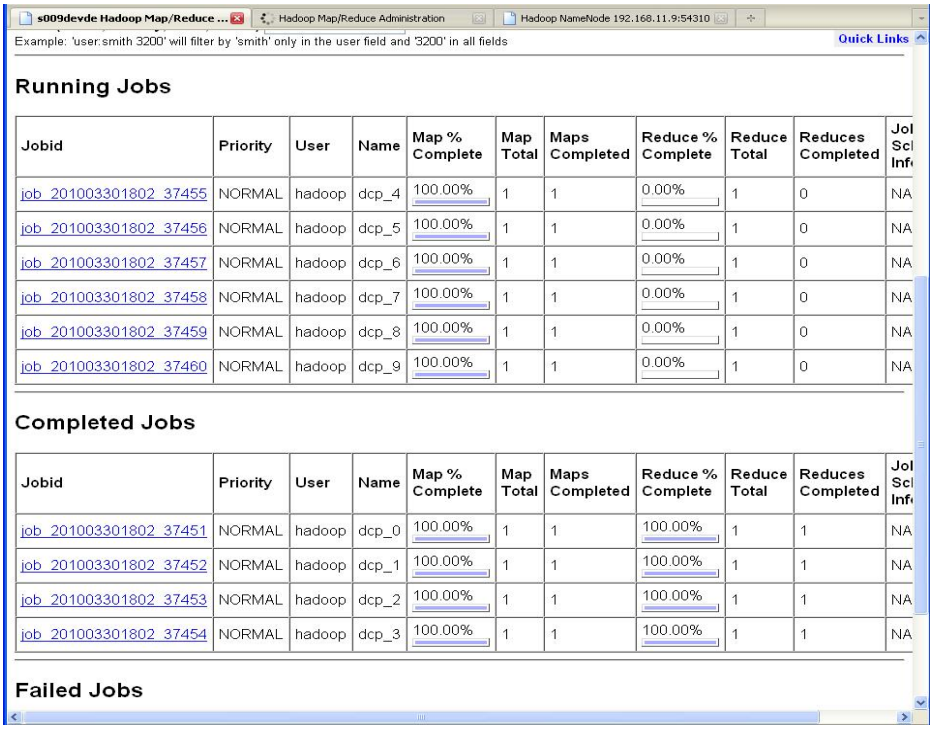


Fig. 7. Sample Hadoop Web Admin. The system offers monitoring and status tools and provides an abstraction model for programming tasks. It supports automatic distribution and parallelization.

7 Conclusions

Experiments executed on randomly generated data input indicated the efficiency of the proposed approach - MRPGA_DOPI - especially for specific attributes of the benchmarks. An application was written for the scalable distributed algorithm using the Hadoop implementation for MapReduce [11, 12]. Scalability, status reporting and monitoring, the fault toleration, the facility to work on commodity hardware, performing the computation there the data is, robustness, are all benefits of the Hadoop implementation. Next steps would be to perform experiments on larger data inputs, with various distributions, but also variants for the map or reduce functions. By performing the algorithms on real data inputs, e.g. from bioinformatics or system-on-chip design, will provide further knowledge. E.g. in bioinformatics, DOPI problem could be formulated using the Watson-Crick complements, for every nucleic acid:

$$\overline{A} = T, \overline{G} = C, \overline{C} = G, \overline{T} = A\,. \tag{5}$$

References

1. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. The MIT Press, Cambridge (2009)
2. Davis, L.: Handbook of Genetic Algorithms. van Nortand Reinhold, New York (1991)
3. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Communications of the ACM 51(1), 107–113 (2008)
4. Holland, J.H.: Adaption in Natural and Artificial Systems. University of Michigan Press, Ann Arbor (1975)
5. Llorà, X., Verma, A., Campbell, R.H., Goldberg, D.E.: When Huge is Routine: Scaling Genetic Algorithms and Estimation of Distribution Algorithms via Data-Intensive Computing. In: de Vega, F.F., Cantú-Paz, E. (eds.) Parallel and Distributed Computational Intelligence. SCI, vol. 269, pp. 11–41. Springer, Heidelberg (2010)
6. Logofatu, D., Drechsler, R.: Efficient Evolutionary Approaches for the Data Ordering Problem with Inversion. In: Rothlauf, F., Branke, J., Cagnoni, S., Costa, E., Cotta, C., Drechsler, R., Lutton, E., Machado, P., Moore, J.H., Romero, J., Smith, G.D., Squillero, G., Takagi, H. (eds.) EvoWorkshops 2006. LNCS, vol. 3907, pp. 320–331. Springer, Heidelberg (2006)
7. Logofătu, D., Gruber, M.: DNA Sequences Vectors and Their Ordering. In: AIP Conf. Proc., BICS 2008: Proceedings of the 1st International Conference on Bio-Inspired Computational Methods Used for Solving Difficult Problems: Development of Intelligent and Complex Systems, Tg. Mureş, vol. 1117(1), pp. 3–11 (2008)
8. Logofătu, D.: Grundlegende Algorithmen mit Java, pp. 65–98. Vieweg, Wiesbaden (2008)
9. Logofătu, D., Dumitrescu, D.: Parallel Evolutionary Approach of Compaction Problem Using MapReduce. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6239, pp. 361–370. Springer, Heidelberg (2010)
10. De Micheli, G.: Synthesis and Optimisation of Digital Circuits. McGraw-Hill, Inc. (1994)
11. Murgai, R., Fujita, M., Krishnan, S.C.: Data Sequencing for Minimum-Transition Transmission. In: IFIP Int'l Conf. on VLSI (1997)
12. Murgai, R., Fujita, M., Oliveira, A.: Using Complementation and Resequencing to Minimize Transitions. In: Design Automation Conf., pp. 694–697 (1998)
13. Oliver, I.M., Smith, D.J., Holland, J.R.C.: A Study of Permutation Crossover Operators on the Traveling Salesman Problem. In: Int'l Conference on Genetic Algorithms, pp. 384–390 (1994)
14. Shen, W.-Z., Lin, J.-Y., Wang, F.-W.: Transistor reordering rules for power reduction in CMOS gates. In: ASP Design Automation Conf., pp. 1–6 (1995)
15. Sotiriadis, P.P., Chandrakasan, A.: Low Power Bus Coding Techniques Considering Interwire Capacitances. In: Proc. of IEEE Conf. on Custom Integrated Circuits (CICC 2000), pp. 507–510 (2000)
16. Stan, M.R., Burleson, W.P.: Bus Invert Coding for Low-Power I/O. IEEE Tr. on VLSI Systems 3(1), 49–58 (1995)
17. Stan, M.R., Burleson, W.P.: Low-Power encodings for Global Communication in CMOS VLSI. IEEE Tr. on VLSI Systems 5(4), 444–455 (1997)
18. Apache Hadoop, web-link (2009), `http://hadoop.apache.org/`