Massively Parallelized DNA Motif Search on the Reconfigurable Hardware Platform COPACOBANA

Jan Schröder, Lars Wienbrandt, Gerd Pfeiffer, and Manfred Schimmler

Department of Computer Science, Christian-Albrechts-University of Kiel, Germany
{jasc,lwi,gp,masch}@informatik.uni-kiel.de

Abstract. An enhanced version of an existing motif search algorithm BMA is presented. Motif searching is a computationally expensive task which is frequently performed in DNA sequence analysis. The algorithm has been tailored to fit on the COPACOBANA architecture, which is a massively parallel machine consisting of 120 FPGA chips. The performance gained exceeds that of a standard PC by a factor of over 1,650 and speeds up the time intensive search for motifs in DNA sequences. In terms of energy consumption COPACOBANA needs 1/400 of the energy of a PC implementation.

Keywords: Motif finding, DNA sequence analysis, FPGA, High Performance Reconfigurable Computing (HPRC).

1 Introduction

The discovery of regulatory sequences in DNA - called motif-finding - is one of the most challenging problems in the field of bioinformatics. In fact there are problem instances of motif-finding which are unsolvable by current techniques. There are two reasons that make this problem so difficult: firstly, the parameters of a given problem instance (like sequence length, motif length, grade of mutation) can make it impossible to identify motifs due to background noise. Secondly, it is computationally expensive. So a precise algorithm can fail to discover a motif in a given sequence because its execution time exceeds rational means. We address both problems with a new approach to motif searching making use of a novel massively parallel architecture to speed up the execution time.

Motif searching has been an issue in many publications of the last ten years. As the most popular approaches to this topic we reference MEME [16] [17] [18] and the similar Gibbs sampler [14] [15] which iteratively develops matrices representing motifs of the input sequence using the expectation maximization technique; the projection algorithm [13] [20] which creates a representation of the highly conserved region over all motif instances; and CONSENSUS [21] - a greedy approach which constructs likely motif candidates by aligning only small parts of the genome at a time.

M. Chetty, A. Ngom, and S. Ahmad (Eds.): PRIB 2008, LNBI 5265, pp. 436–447, 2008.

The algorithm IGOM (Iterative Generation of position frequency matrices) has been published in [22]. This method iteratively develops a set of strings which are likely to be instances of an underlying motif by featuring two new ideas. It makes use of the structure of position frequency matrices of already known motifs which imply a distribution on only one or two nucleotides in each position rather than all four of them [19]. The *sp*-model has been introduced in [22] to describe this restriction. This observation is utilized to develop a very precise description of a kernel of the motif in the first few iterations of the algorithm. This has the advantage that the likelihood of false positives which fit to this description although not belonging to the motif is minimized. Regulatory sequences that match the observations of the *sp*-model (for example the *SigmaB* regulator in *Bacillus subtilis* [24]) are discovered easier and more accurately by this algorithm compared to the other methods of motif searching.

The second key feature of IGOM is the surveillance of the expected false positives which could arise from loosening the description of the motif. The algorithm will only make those changes to the matrix where the quotient of valid new candidates divided by the expected number of random strings which fit this change - and appear in any sequence of the given length without relevance - is maximal. The authors describe this quotient with the term *signal to noise ratio* (SNR) because of its correlation to signal theory where one tries to maximize the signal opposing to the background noise of the medium.

Further improvements of the algorithm has been published in [23]. The main idea of this publication is a Boolean representation of motif kernels. Instead of position frequency/weight matrices we use Boolean matrices to describe a motif. A value of "1" in a Boolean matrix (BM) considers the nucleotide to be a valid representation for a motif instance in the corresponding position [23]. It leads to a huge improvement of the complexity and makes this method highly applicable for special purpose architectures. Most of the methods in Bioinformatics gain performance when applied to special hardware because of the small alphabet sizes when dealing with DNA or protein sequences and simple operations on the input data. We chose to implement the IGOM/BMA algorithm in hardware because of its ideal qualifications:

1. The input data can be represented in a very efficient way with only two bits per nucleotide
2. The algorithm can be parallelized in an ideal way because of the independent search operations on the data.
3. The Boolean matrices used to represent motifs can be stored very efficiently in hardware allowing many processes working on a single FPGA chip simultaneously.

The amount of biological sequence information is increasing more rapidly [1] than the exponential performance growth of general purpose microprocessor-based computers. Due to this observation highly optimized special-purpose computers have been developed. Today, the technology of Field Programmable Gate Arrays (FPGAs) exhibit impressive performance compared to microprocessor-based machines, among other things in the field of bioinformatics. Successful

special purpose hardware are for example SPLASH 2 [2], JBits [3], BEE2 [4], XD1000 [5], RASC RC100 [6], and DeCypher [7]. The recent massively parallel FPGA-based architecture COPACOBANA [8] from *SCIENGINES* [9] is chosen as target for the proposed motif search algorithm. Taking advantage of the hardware architecture and the highly parallel nature of the algorithm we can accelerate this method with huge efficiency. Implementing the iterative development of motif kernels on the COPACOBANA we outperform a single desktop PC by a factor of over $1,650$. Taking into account the higher cost of the COPACOBANA, a cost performance ratio would be fairer for comparison. This leads to a performance per cost ratio up to five times higher compared to desktop PCs and of course accordingly faster execution time. Additionally the power consumption of PCs for the same task is much higher than that of COPACOBANA. We reach an energy efficiency more than 420 times better than standard PCs.

This paper is organized as follows. In chapter 2 the COPACOBANA hardware is specified, in chapter 3 the implemented algorithm is described. Chapter 4 will discuss the details concerning the implementation of the algorithm in hardware. Performance analysis, conclusion and outlook will follow in chapters 5 and 6.

2 COPACOBANA

The massively parallel computer COPACOBANA consists of 120 low cost FPGAs which are connected to a controller module by a bus system. It can be integrated in any standard Local Area Network (LAN) environment and is fully remotely controlled. Originally COPACOBANA has been developed as C*ost-*O*ptimized* PA*rallel* CO*de* B*reaker* in 2006. The goal was to break the 56-bit Data Encryption Standard (DES) in 10 days for production and material cost of less than $\$10,000$. [10] Actually it breaks DES in 7 days [8] in the mean. Due to the universality of FPGA-chips [11] this machine is suited for all kinds of fine grained parallel applications with low communication and memory requirements, and with special attention to the cost/performance ratio.

The FPGAs are of the type *Xilinx Spartan-3 1000* [12] (*XC3S1000*, speed grade -4, FTG256 packaging). Each comes with 1 million system gates, $17,280$ equivalent logic cells, $1,920$ Configurable Logic Blocks (CLBs) equivalent to $7,680$ slices, 120 kbit distributed RAM, 432 kbit Block RAM (BRAM), 24 dedicated $18x18$ multipliers, and 4 digital clock managers (DCMs). Figure 1 depicts the data path of COPACOBANA. Pluggable cards in DIMM format are holding 6 FPGAs each. Twenty of these cards are plugged into slots of a common backplane together with a controller card. The latter is the interface to a host computer via Ethernet LAN. It is transferring data and controlling the single master bus system which is currently operating at up to 1 Mbit/s. The host computer is executing a front-end software which uses an Application Programming Interface (API) for accessing COPACOBANA. Additionally some parts of the target algorithm are implemented here which for instance are sequential, perform a post- or preprocessing, or access a hard drive. This software represents the highest control instance, because it initiates any action of the controller, hence

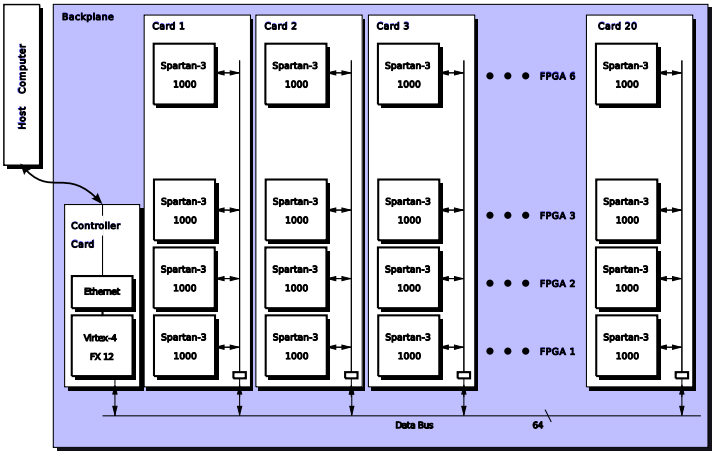

Fig. 1. COPACOBANA Data Path

it controls the entire machine. In other words, communication can not be initialized by one of the 120 slave FPGAs because COPACOBANA does not support interrupts. Therefore a static communication scheduling has to be considered for the host software.

The controller provides the following addressing modes. A single FPGA can be selected for writing and reading data. Any set of FPGAs on one card up to all 6 can be written to from the controller, and finally via broadcast the controller can write data to all 120 FPGAs. Each of the FPGAS can be configured to suit its purpose exactly: small processing units can be implemented on the chip that are designed only for one specific task.

3 Algorithm

In this section a description of the BMA algorithm is given. Since we aim for a massively parallel implementation (see section 4) it will be slightly modified with respect to the scoring function given in [22] and [23]. Given the input data - a whole genome or a particular set of sequences - and a fixed motif length $l2$ the algorithm will develop motif kernels in increasing order by the likelihood of their occurrence in a randomly distributed sequence. So assuming a normal distribution of the input data we are interested in the least likely occurrence of motif candidates in terms of over-representation. We will analyze the signal to noise ratio (SNR) to find those candidates:

1. The algorithm starts with a single string of the motif length and specifies the Boolean matrix.
2. Each iteration it will modify one column of the matrix so that two nucleotides will represent the given position of the motif - following the conclusions of the *sp*-model. The algorithm chooses the position in the matrix by analyzing

the SNR so it minimizes the probability of false positives and finds the best representation of the motif.
3. Beneath all the matrices generated each round (one for every start string) the best in terms of SNR are chosen and analyzed further if they are likely to represent a real motif in the organism represented by the input data. We will not discuss this third step in this paper since only the development of motif kernels is the time consuming part of the method which we apply to hardware.

For the sake of an efficient implementation we will restrict the algorithm in the following way. In each iteration there will be one change of the matrix - whether it is good or not in terms of SNR - and every change will add a "1" to a column of the matrix where there was exactly one "1" before. This has the great benefit that every matrix in the same round of the algorithm has exactly the same value for the expected noise. So SNR can be compared easily only by analyzing the number of candidates from the input data matching the describing matrices. We can take great advantage of this restriction in the implementation because it allows much simpler and smaller units processing the matrices.

3.1 Example

To illustrate how the algorithm works, we show a short example. Let the BM look like the first matrix illustrated in figure 2 after the first iteration of the algorithm. So the strings "GAAGT" and "GCAGT" match the matrix. In the second iteration all strings that match all but one position of the BM will contribute to a scoring matrix. For example, the string "GCAAT" would score for an "A" in the fourth column, whereas the string "AAAAT" would not contribute at all because it has too many mismatches in this iteration. After scoring all substrings of the genome in this manner a scoring matrix like the second matrix in figure 2 could arise. With the identified maximum of the scoring matrix (marked in the figure) the two strings "GAAAT" and "GCAAT" will be taken into the motif kernel forming a new BM, which is depicted as the third matrix in figure 2.

A	0	1	1	0	0	A	0	x	x	**4**	1	A	0	1	1	**1**	0
C	0	1	0	0	0	C	1	x	1	0	2	C	0	1	0	0	0
G	1	0	0	1	0	G	x	1	3	x	2	G	1	0	0	1	0
T	0	0	0	0	1	T	0	1	0	0	x	T	0	0	0	0	1

Fig. 2. Example: Boolean matrix at the beginning of an iteration, matching the strings "GAAGT" and "GCAGT", and a possible scoring matrix with the resulting new boolean matrix after the iteration.

4 Hardware Implementation

4.1 Parallel Processing Scheme

Since we are starting without any knowledge about possible motif candidates, the algorithm requires to analyze any possible position frequency matrix (PFM). For

a motif length of 12 nucleotides there are $4^{12} = 16,777,216$ such PFMs. There is no data dependency between any two of them. So, we can use a trivial parallelization scheme where a maximum number of PFMs is computed in parallel. COPACOBANA contains 120 FPGA chips. Each of them can be configured to provide 32 independent search entities. This accumulates to $3,840$ search entities to work concurrently.

The DNA is viewed as a sequence over the alphabet $\{A, C, G, T\}$. Every character can be represented with two bits. The restricted size of the local memory of the *Spartan-3* chips does not allow to store the complete DNA sequence in every search entity. Instead, it is provided by globally broadcasting it to all search entities character by character. Each entity continuously accumulates the relevant information to update its particular PFM using the globally broadcasted data stream.

Since 4^{12} is greater than $3,840$ it is necessary to compute the complete problem in $4^{12}/3,840 = 4,370$ subsequent identical computation runs. Each run requires a fixed number of iterations for updating the PFMs. It has turned out that more than six iterations do not provide useful results anymore. Therefore, the complete DNA sequence has to be broadcast to all processors a total number of $4,370 \cdot 6 = 26,220$ times. In our implementation, the DNA sequence is locally stored in the controller of COPACOBANA in order to reduce the traffic on the TCP/IP connection.

The PFM analysis is done in four steps:

1. The host application sends a command to initialize the search entities on the FPGAs. This command also provides the initialization matrix in form of an index. The index is in the range from 0 to $16,777,215$, each identifying a unique PFM.

The following steps will be repeated six times:

2. Each search entity scores all subsequences of the broadcasted DNA sequence against its own PFM.
3. The local results are read from the search entities. The best scores are stored in sorted lists on the host. There is one list for each iteration after initialization, so there will finally be six lists in this case. The number of best results saved is user defined.
4. Finally the host sends an update command to alter the position frequency matrices in the search entities.

After all six iterations have finished the next initialization is done with new indices, i.e. with new PFMs. The algorithm starts again with step 1. When the application has finished the lists with the user defined amount of best results for each of the six iterations are ready for further analysis.

4.2 FPGA Design

The main processing unit is the *search entity* which provides the core functionality of the algorithm. 32 search entities fit on a single FPGA chip and thus can work in parallel. In the following one of those is described in detail.

Every search entity has a unique identifier for individualization. The identifier is a natural number starting with zero. One search entity consists of an implementation of the boolean position frequency matrix and its matching functionality, a score counter and a counter for differing sequences, further called "difference counter".

Initially, the search entity receives an index which corresponds to a gene sequence of length 12, the expected motif size. By adding the identifier of the search entity to the incoming index every entity generates its individual initialization sequence. Hence, an FPGA has to be provided only one time with an initialization index to initialize 32 search entities at once. The initialization sequence is easily converted to the matrix structure by using lookup tables. "*A*" is "1000", "*C*" is "0100" etc.

An incoming gene sequence is matched with the position frequency matrix. If the sequence matches the score counter increments a locally stored score value. For every matrix position a counter is provided which is increased whenever the string under consideration has a mismatch in this position and it is the only mismatch with the PFM. This implicates that with a motif size of 12 we need 48 counters for each search entity. Given the fact that only one counter per search entity has to be accessed at maximum in one clock cycle, the counters can easily be stored in the local block RAM which is available in every *Spartan-3* FPGA.

The only data locally stored by the difference counter is the maximum counter value and a corresponding gene sequence which causes this counter to increment. This makes a matrix update fast and easy. The update command converts the sequence to the matrix structure like for the initialization followed by a simple or-operation on the old matrix. Every search entity provides its position frequency matrix and score as result.

The control of the motif search operation is realized by the *search control entity*. It manages the incoming control instructions and user data from the host application. The user data is read from the bus in 64 bit blocks which equals 32 characters. It is then provided to a FIFO buffer as a data stream. The buffer always provides a window of 12 characters as data input to the search entities. The search control entity also provides the best result of the search entities to the host application. Therefore the results are compared by their score. The comparison is made by comparators which are aligned to each search entity in a chain. Every comparator compares the result of its predecessor and one search entity. If the best result is fetched by the host application its score is cleared on the corresponding search entity. Hence, the second best result will be automatically provided to the host.

Figure 3 shows a simplified overview of the FPGA design.

4.3 Data and Control Flow on the FPGA

The search entities are organized on the FPGA in two chains due to the two rows of block RAM on the *Spartan-3*. All user and control data is buffered by one entity and provided to the successor in its chain in the next clock cycle. This keeps data paths short and permits higher frequencies. Except for the command

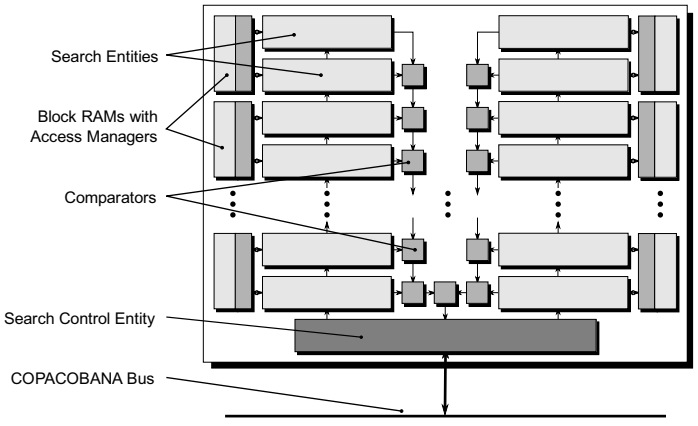

Fig. 3. FPGA design overview

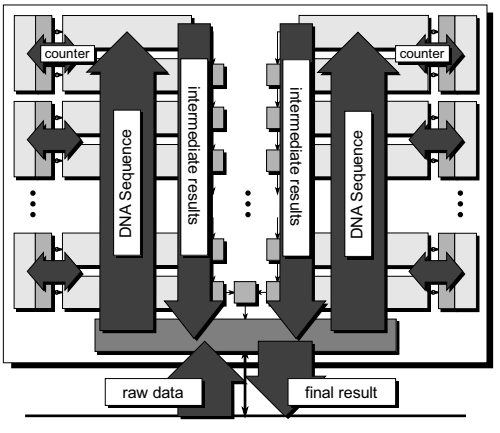

Fig. 4. FPGA dataflow overview

to read a result, all control instructions plus the user data from the host are provided by the search control entity directly to the first two search entities in the chains.

The comparators for the results are organized in two chains along the search entities as well. Every comparator compares the result of its predecessor and one search entity in one clock cycle. At the beginning of the chain the first comparators compare the results of the last two search entities. So the data flow of the results is contrary to the data flow of the sequence data. This again keeps data paths short because the maximum result of both chains is provided back to the search control entity after a final comparison. The signal to clear the best results score after being fetched by the host is routed through the comparator chain as well.

The overview of the design flow is shown in figure 4.

5 Performance Analysis and Conclusion

5.1 *C++* Implementation

For comparison the DNA motif search algorithm has been implemented in *C++*. It has been compiled with the *GNU Compiler Collection* (GCC) v4.0.2 and the "`-O3`"-flag for highest optimization. Additionally the "`-march=...`"-flag and other optimization flags like "`-msse3`" were set for the corresponding target architecture. The testing systems were a standard PC with an *Intel Pentium IV* at 2.8 GHz and a PC with an *AMD Thurion64 X2* dual core at 1.9 GHz running a *Linux* operating system, and a *Macintosh Pro* with two *Intel Xeon 5150* dual cores at 2.6 GHz running *Mac OS X*. The implementation uses static memory for the gene sequence, the score and the counter values for the missed matches. So no new memory is allocated dynamically at runtime except for new results in the lists. Since we store only 100 results for each iteration the allocations are very scarce and do not significantly delay the process. Actually the host application using COPACOBANA does the same. For the dual core architectures the task was equally divided into two processes.

Because the algorithm can not be parallelized for one processing core the application takes one initial position frequency matrix at a time. But we made one significant improvement which could not be applied to the parallelized solution. This application does not always perform six iterations per initialization index. It does a further iteration only if the position frequency matrix was updated in the preceding one. This causes a significant speedup for the iterative solution.

5.2 Performance

The applications were configured to analyze the DNA sequences of *Cowpox Virus* (280*k* bases), *Rickettsia canadensis str. McKiel* (1.2*M* bases) and *Bacillus subtilis* (5.9*M* bases) as an example. The desired motif size is 12 and the number of iterations is set to 6. Table 1 shows the duration of the computation and the speedup of COPACOBANA vs. the specified architectures. Figure 5 shows

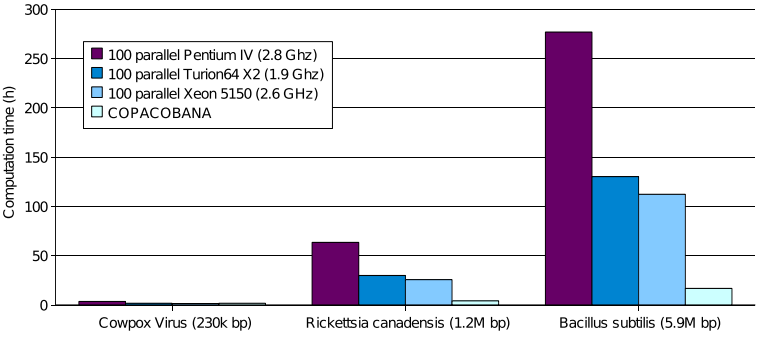

Fig. 5. Computation times for *Cowpox Virus*, *Rickettsia canadensis* and *Bacillus subtilis*

Table 1. computation times and speedups of the DNA motif search algorithm

		COPACOBANA	*Xeon 5150* 2.6 GHz dual core	*Thurion64 X2* 1.9 GHz dual core	*Pentium IV* 2.8 GHz single core
Cowpox	time	$1h40m$	$144h$ (6 d.)	$167h$ (7 d.)	$355h$ (14.8 d.)
Virus	speedup	1	> 86	> 100	> 210
Rickettsia	time	$4h10m$	$2,575h$ (107.3 d.)[1]	$2,987h$ (124.5 d.)[1]	$6,350h$ (264.6 d.)[1]
canadensis	speedup	1	> 615	> 715	$> 1,520$
Bacillus	time	$16h45m$	$11,236h$ (1.3 y.)[1]	$13,031h$ (1.5 y.)[1]	$27,700h$ (3.2 y.)[1]
subtilis	speedup	1	> 670	> 775	$> 1,650$

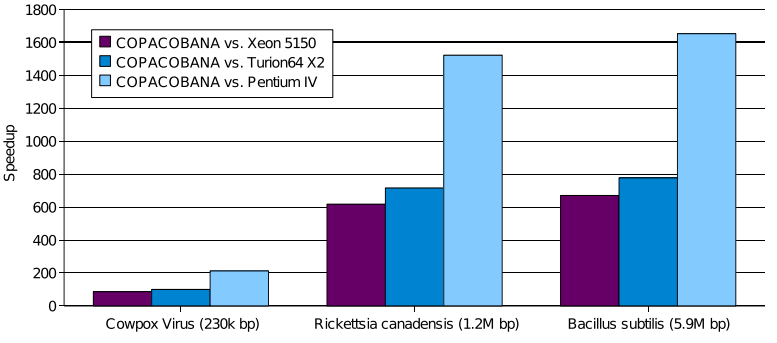

Fig. 6. COPACOBANA speedups vs. several processors

a graphical presentation of these computation times. To make the results more reasonable the computation times for the PCs are adapted to match 100 ideally parallelized PCs of each type in this figure. Figure 6 shows the speedups of COPACOBANA vs. one PC of the specified architecture.

5.3 Conclusion

Although significant improvements have been made to the iterative algorithm the parallel solution generates the same results in much shorter time. Previously nearly unreachable results due to the length of the computation time could now be afforded in less than one day. Unfortunately the drawback is the need of the special purpose hardware, but with the cost of €60,000 for a COPACOBANA and €200 for a standard PC the cost/performance ratio is only $(€60,000/€200)/1,650 = 0.18$. This means COPACOBANA is more than 5 times more cost effective than a standard PC. Another advantage of COPACOBANA is energy efficiency. Due to the short computation time and only 600W power consumption it consumes about 10.5 kWh to calculate the motif candidates for *Bacillus subtilis*. In contrast standard PCs consume about 4,155

[1] This value is computed by measuring a small part and extrapolating the duration.

kWh for the same task at 150W per PC. Assuming energy costs of €0.20/kWh this would be €831.00 against €2.10. This is about 400 times the energy costs of COPACOBANA. Easy calculation shows that the hardware price of COPACOBANA would be payed for energy costs of a PC cluster solving only 100 problems of the size of calculating motif candidates of *Bacillus subtilis*. Additionally the costs to build a cluster out of several PCs to reach the performance of one COPACOBANA are not considered. These are costs for connection cables, switches and the place to deposit these components. Furthermore the knowledge to get such a PC cluster working with this application has to be paid for as well.

6 Outlook

The performance analysis for COPACOBANA is made with a slow controller having a very little bandwidth of approximately 1 Mbit/s. There is already a new controller module under development which reaches a bandwidth of about 100 Mbit/s. First tests with this application reached 5 to 7 times the speed of the slow controller. Hence the controller is still the bottleneck of this application and even greater speedups could be reached easily. This leads to another improvement of the cost performance ratio and energy efficiency.

With the speedup gained by the COPACOBANA implementation we can intensify motif searching on real datasets. We will put it to use by analyzing motifs in virus datasets in close collaboration with the medical institute of the Free University of Berlin.

References

1. Meyer, F.: Genome Sequencing vs. Moore's Law. In Cyber Challenges for the Next Decade. CTWatch Quarterly 2, 1–2 (2006)
2. Hoang, D.T.: Searching Genetic Databases on SPLASH 2. In: Proc. Workshop on FPGAs for Custom Computing Machines (1993)
3. Guccione, S.A., Keller, E.: Gene Matching Using JBits. In: Proc. 12th Field Programmable Logic and Applications. Springer, Berlin (2002)
4. Herbordt, M.C., Model, J., Sukhwani, B., Gu, Y., Van Court, T.: Single pass streaming BLAST on FPGAs. In: Parallel Computing. Special issue on High-Performance Computing Using Accelerators, vol. 33, pp. 741–756 (2007)
5. XtremeData, Inc., `http://www.xtremedatainc.com/`
6. Silicon Graphics, Inc., `http://www.sgi.com/products/rasc/`
7. Time Logic Corp., `http://www.timelogic.com/`
8. COPACOBANA Research Project, `http://www.COPACOBANA.org/`
9. SCIENGINES Corp., `http://www.sciengines.com/`
10. Kumar, S., Paar, C., Pelzl, J., Pfeiffer, G., Rupp, A., Schimmler, M.: How to Break DES for € 8,980. In: 2nd Workshop on Special-purpose Hardware for Attacking Cryptographic Systems - SHARCS, April 3-4, Cologne, Germany (2006)
11. DeHon, A.: The Density Advantage of Configurable Computing. IEEE Computer 33(4), 41–49 (2000)
12. Xilinx Inc., `http://www.xilinx.com/`

13. Buhler, J., Tompa, M.: Finding motifs using random projections. J. Comput. Biol. 9, 225–242 (2002)
14. Lawrence, C.E., Reilly, A.A.: An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. Proteins 7, 41–51 (1990)
15. Lawrence, C.E., Altschul, S.F., Boguski, M.S., Liu, J.S., Neuwald, A.F., Wootton, J.C.: Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment. Science 262, 208–214 (1993)
16. Bailey, T.L., Elkan, C.: Fitting a mixture model by expectation maximization to discover motifs in biopolymers, UCSD Technical Report, CS94-351. University of California at San Diego (March 1994)
17. Bailey, T.L., Elkan, C.: Unsupervised Learning of Multiple Motifs in Biopolymers using EM. Machine Learning 21(1-2), 51–80 (1995)
18. Redhead, E., Bailey, T.L.: Discriminative motif discovery in DNA and protein sequences using the DEME Algorithm. BMC Bioinformatics 8, 385 (2007)
19. RegTransBase, `http://regtransbase.lbl.gov/`
20. Varghese, G., Raphael, B., Lung-Tien Liu, A.: Uniform Projection Method for Motif Discovery in DNA Sequences. IEEE Transactions On Computational Biology And Bioinformatics 1(2) (April-June 2004)
21. Hertz, G., Stormo, G.: Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. Bioinformatics 15(7-8), 563–577 (1999)
22. Schröder, J., Schimmler, M., Tischer, K., Schröder, H.: IGOM - Iterative Generation of Position Frequency Matrices (submitted, 2008), `http://www.informatik.uni-kiel.de/fileadmin/arbeitsgruppen/technical_cs/Files-Jan/IGOM_paper.pdf`
23. Schröder, J., Schimmler, M., Tischer, K., Schröder, H.: BMA - Boolean Matrices as Model for Motif Kernels. In: 2008 International Conference on Bioinformatics, Computational Biology, Genomics, and Chemoinformatics (BCBGC 2008) (July 2008), `http://www.informatik.uni-kiel.de/fileadmin/arbeitsgruppen/technical_cs/Files-Jan/paper_bcbgc125.pdf`
24. Petersohn, A., Brigulla, M., Haas, A., Hoheisel, J., Völker, U., Hecker, M.: Global Analysis of the General Stress Response of Bacillus subtilis. Journal Of Bacteriology, 5617–5631(October 2001)