Accelerating Motif Discovery: Motif Matching on Parallel Hardware

Geir Kjetil Sandve[1], Magnar Nedland[2], Øyvind Bø Syrstad[1], Lars Andreas Eidsheim[1], Osman Abul[3], and Finn Drabløs[3]

[1] Department of Computer and Information Science,
Norwegian University of Science and Technology, Trondheim, Norway
sandve@idi.ntnu.no, {syrstad, eidsheim}@stud.ntnu.no
[2] Interagon A.S.,
Trondheim, Norway
magnar.nedland@interagon.com
[3] Department of Cancer Research and Molecular Medicine,
Norwegian University of Science and Technology, Trondheim, Norway
{osman.abul, finn.drablos}@ntnu.no

Abstract. Discovery of motifs in biological sequences is an important problem, and several computational methods have been developed to date. One of the main limitations of the established motif discovery methods is that the running time is prohibitive for very large data sets, such as upstream regions of large sets of cell-cycle regulated genes. Parallel versions have been developed for some of these methods, but this requires supercomputers or large computer clusters. Here, we propose and define an abstract module PAMM (Parallel Acceleration of Motif Matching) with motif matching on parallel hardware in mind. As a proof-of-concept, we provide a concrete implementation of our approach called MAMA. The implementation is based on the MEME algorithm, and uses an implementation of PAMM based on specialized hardware to accelerate motif matching. Running MAMA on a standard PC with specialized hardware on a single PCI-card compares favorably to running parallel MEME on a cluster of 12 computers.

1 Introduction

Computational discovery of motifs in biological sequences has many important applications, the best known being discovery of transcription factor binding sites (TFBS) in DNA and active sites in proteins. More than a hundred methods have been developed for this problem, all with different strengths and characteristics. Methods that use probabilistic motifs (typically PWMs) are often favored because of their high expressibility. One of the best known and most widely used methods is MEME [1]. MEME is a flexible tool that uses Expectation Maximization (EM) to discover motifs as position weight matrices (PWMs) in both proteins and DNA.

One of the main limitations of current PWM-based motif discovery methods is that the running time is prohibitive for large datasets such as upstream regions of large sets of cell-cycle regulated genes. Parallel versions have been developed for some methods, for instance the paraMEME [2] version of MEME, but this typically requires supercomputers or computer clusters. Specialized hardware, such as Field Programmable Gate Arrays (FPGAs), may be a very viable alternative to this. FPGAs have previously been used in bioinformatics for instance to accelerate homology search [3], multiple sequence alignment [4] and phylogeny inference [5].

In this paper, we propose and define an abstract module PAMM (Parallel Acceleration of Motif Matching). Proposing the PAMM module serves two purposes. Firstly, it introduces acceleration of motif matching by parallel hardware to the motif discovery field. Secondly, PAMM serves as an interface between the development of modules for parallel matching of motifs and the development of algorithms that can make use of parallel motif matching.

As a first implementation of our methodology, we propose a method MAMA (Massively parallel Acceleration of the Meme Algorithm) that accelerates MEME by the use of an existing pattern matching hardware called the Pattern Matching Chip (PMC) [6]. The PMC can match a subset of regular expressions with massive parallelization[1]. Since this chip was not intended for weighted pattern matching, some transformations are needed when representing and matching motifs. Nonetheless, with these transformations in place we achieve very efficient matching of PWMs against sequences. Running MAMA on a standard PC with specialized hardware on a single PCI-card compares favorably to running paraMEME on a cluster of 12 computers.

2 Parallel Acceleration of Motif Matching

An ever increasing number of computing platforms offer capabilities for parallel execution of programs. Specialized hardware exists to relieve the main CPU of specific tasks, and FPGAs allow the creation of modules for application specific hardware acceleration. To allow the field of motif discovery to realize the full potential of modern computing hardware, the algorithms need to take advantage of this.

Here we propose and define an abstract module PAMM that can be used for accelerating motif discovery by matching motifs against sequences in parallel. The purpose of PAMM is to serve as an interface between development of modules for parallel matching of motifs and the development of algorithms that can make use of parallel motif matching. An overview of the PAMM module is presented in Figure 1. The input to PAMM is a set of motifs M and a set of sequences S, while the output depends on the requirements of the algorithm in question. Each motif is represented as a matrix. As the figure shows, there are two main parts in the PAMM module; a motif matcher and a post processing unit. The motif matcher calculates the match scores for each motif, while the post processing unit refines the results.

[1] More information at http://www.interagon.com

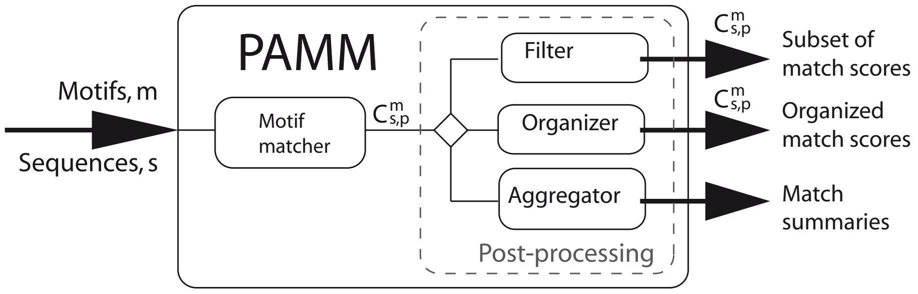


Fig. 1. The structure of the PAMM module

2.1 Motif Matching

The core of a PAMM implementation is a motif matcher that determines match scores $c^{m}_{s,p}$ for each motif m when aligned at each position p in each sequence s. As the number of motifs and sequences that can be processed in parallel will be limited in any practical implementation of the module, the algorithm must partition the inputs accordingly.

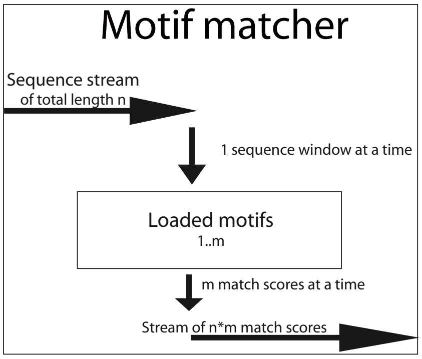
As a standard set-up we propose that a limited number of motifs are first loaded into the PAMM, and that sequence data are then streamed through. The motif matcher will continually calculate match scores for each motif against the sequences. When all motifs have been matched against the complete sequence data, a new set of motifs can be loaded into the module and matched against the sequences. As this means that the same sequences will typically be streamed through the PAMM many times, practical implementations could have an option to store a limited amount of sequence data in local memory to further accelerate matching and reduce bandwidth usage. This set-up is illustrated in Figure 2(a).

An alternative set-up could be to first load a limited amount of sequence data into the PAMM, and then stream motifs through the module. This could be an effective solution for cases with relatively short sequence data and large number of motifs. This setup is illustrated in Figure 2(b).

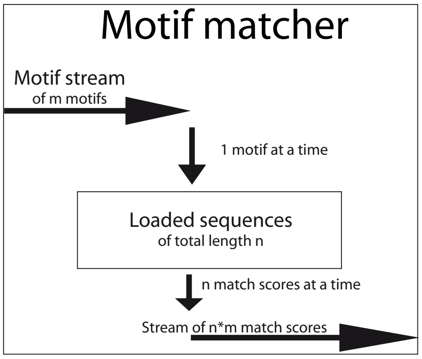
2.2 Post-processing of Match Scores

The number of results from the motif matcher is $|M| * |S|$, where M is the set of motifs and S is the set of all sequence data. This potentially large amount of results must somehow be processed by the system. By incorporating post processing, the number of results returned from a PAMM implementation can be reduced substantially. This reduces result processing in the algorithm module, as well as bandwidth requirements in the case where the PAMM and algorithm modules reside on different (sub)systems.

We envision three main branches of post processing for PAMM implementations; organizing, filtering, or aggregating (or a combination of these).



(a) Motif matcher with loaded motifs and sequences streamed through



(b) Motif matcher with loaded sequences and motifs streamed through

Fig. 2. Two possible set-ups of the motif matcher

An organizing post processor organizes the results in a way that facilitates efficient further processing of results outside the PAMM module. It could for instance return the match scores sorted by value. Although this does not decrease bandwidth usage, it may allow the CPU to process the results more efficiently.

A filtering post processor filters out uninteresting match scores to save processing time outside the PAMM module. It could for instance make the PAMM return only match scores above a threshold given for each motif. Although this discards some information, our own experiments (not presented here) show that the normalized match scores typically follow a distribution where most sequence offsets have a negligible likelihood of being motif locations. In combination with an organizing post processor, the k highest match scores could be returned, or all scores at most l lower than the highest match score.

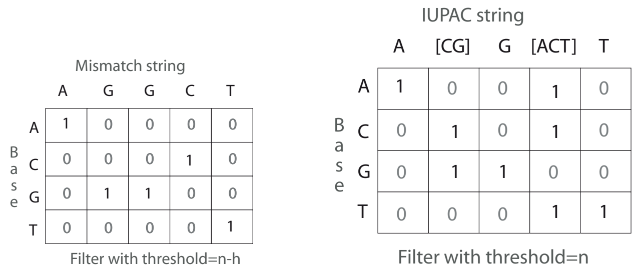
An aggregating post processor is tailored to a specific motif discovery algorithm and may be particularly (computationally) effective. If the PAMM is to be used in connection with stochastic optimization methods like Gibbs sampling, it can be set to return one sequence offset per sequence, with offsets chosen randomly based on the normalized probabilities of motif occurrences. Alternatively, if the PAMM is used in connection with EM methods, a new motif may be constructed from the match scores directly in hardware (maximization step of EM). This new motif would represent a weighted average of every window in the sequences, with windows weighted by the match score of a previous motif.

2.3 Motif Representations

The representation of a motif in PAMM is as a motif matrix $m \in M$ with element values $m_{i,x}$, where i is motif position and x is a symbol from the alphabet, i.e., $x \in \{A, C, G, T\}$. The element values represent individual scores for each symbol x from the alphabet at each position i in the motif. The motif is aligned against sequences as a sliding window. For a given alignment at position p in sequence

s, the score of motif position i is $m_{i,x}$, where x is the symbol at position $p+i$ in sequence s. The match score $c^m_{s,p}$ of the motif is the sum of scores at each motif position. This motif representation maps directly to PWMs (log-likelihood or log-odds) that are often used for motif discovery.

In addition to PWMs, strings allowing mismatches [7,8] (a consensus string allowing a certain Hamming distance to an occurrence) and IUPAC strings [9,10] (strings of characters and character classes) are commonly used models in motif discovery. Both of these can be represented by a motif matrix. For a motif matrix representing a mismatch string, elements $m_{i,x}$ corresponding to the consensus symbol at a position have value 1, and all other matrix elements are 0. Matrix scores $c >= n - h$ corresponds to a hit for the mismatch expression, where n is motif length and h is allowed number of mismatches. This is shown in Figure 3(a). For a motif matrix representing an IUPAC string, elements $m_{i,x}$ corresponding to symbols in the character class at a position are valued 1, and all other matrix elements are 0. Matrix scores $c = n$ corresponds to a hit for the IUPAC expression. This is shown in Figure 3(b)


Mismatch string
Base
Filter with threshold=n-h
IUPAC string
Base
Filter with threshold=n

	A	G	G	C	T
A	1	0	0	0	0
C	0	0	0	1	0
G	0	1	1	0	0
T	0	0	0	0	1

	A	[CG]	G	[ACT]	T
A	1	0	0	1	0
C	0	1	0	1	0
G	0	1	1	0	0
T	0	0	0	1	1

(a) mismatch strings (b) IUPAC strings

Fig. 3. Matrix representation of discrete motif models

Other and more complex motif models could also be represented with such a matrix (variants of Markov models and bayesian trees have for instance been used in motif discovery). This will typically require a larger motif matrix and some preprocessing of the sequence data. Such preprocessing could be done by additional hardware modules within the PAMM. The generality of the matrix representation makes it suitable as a standard motif representation for the PAMM module.

3 Practical Implementation

This section describes a motif discovery algorithm that uses a PAMM implementation to accelerate motif matching. To explore the potential of PAMM in motif discovery, we have used available hardware (PMC) to implement a PAMM module.

We have analyzed the running time of the MEME algorithm and developed a motif discovery algorithm MAMA based on MEME that uses the PAMM implementation for motif matching in the performance-critial parts. As this is a first implementation and a proof-of-concept, we have only made adjustments to the MEME algorithm that make it run faster while not altering which motifs are discovered.

3.1 Motif Discovery Using the PAMM Module

MEME is a a motif discovery algorithm based on Expectation Maximization (EM) that match motifs against sequences in the expectation step. Profiling of the MEME implementation showed that matching initial motifs (starting points) against sequences consumed most of the total running time. We have therefore made the necessary adjustments to allow parallel acceleration of this first iteration of MEME.

MEME Running Time. EM was first used for motif discovery by Lawrence *et al.* [11]. As EM is easily trapped in local minima, they used several random starting points (initial PWMs) for EM. This was improved in the MEME algorithm of Bailey and Elkan [1], which use every substring of a given length in the data set as starting point. More specifically, for every substring a PWM is constructed with a fixed weight to the elements in the matrix corresponding to symbols in the substring, and another, lower fixed weight to the other elements. As this typically amounts to very many starting points, they run EM for one iteration from each starting point, and then only continue with those PWMs that seem most promising.

Inspection of the MEME implementation[2] shows that specialized code is used for this first iteration, using dynamic programming to exploit overlap between starting points. PWMs generated from each substring in the data set are first matched against the sequences (expectation step). For each PWM, the sequence offsets are then sorted by match score and the k highest scoring offsets used to generate a PWM candidate for the next iteration (maximization step). Finally, the significance values for all candidate PWMs are computed, and the most significant ones kept and refined (iterated until convergence).

MEME tries a very large number of starting points in the first iteration, and only continues with a few most promising motifs. Our profiling showed that the first iteration amounted to around 97% of total running time in our tests, using data sets supplied with MEME, the TCM model, and otherwise default parameters. Although this number might vary for different test cases and parameter settings, it shows that the first iteration is the bottleneck when it comes to running time of the algorithm. Furthermore, matching motifs against sequences and sorting offset scores dominate the running time.

Exploration of Starting Points. As the first iteration dominates the running time of MEME, we have focused on accelerating this part. More specifically, we have used the PAMM module to match PWMs and sort offset scores in the first iteration, and left the remaining parts of MEME unaltered.

[2] Version 3.5.0, downloaded from http://meme.nbcr.net/downloads/

Exploration of starting points differs a bit from all other iterations in MEME. First, all matrix elements of starting point PWMs has one of two values: a fixed high value for elements corresponding to the symbol of the substring it is based on, and a fixed low value for every other element. Thus, all sequence windows at a given Hamming distance from the substring a PWM is based on will get the same PWM score. Ranking of sequence offsets based on PWM score will therefore in the first iteration be equal to ranking of sequences windows based on Hamming distance. Secondly, in a general EM iteration each sequence window is used in the maximization step (weighted by the expectation values). When maximizing the PWMs in the first iteration, however, only the sequence windows corresponding to the top k expectation values are used.

These properties are exploited in MAMA by using a PAMM implementation that represents motifs efficiently and returns sequence offsets sorted by match score. The motif discovery algorithm thus only needs to consider the first k sequence offsets returned by the PAMM implementation.

3.2 Implementation of the PAMM Module

We have implemented PAMM using available hardware for parallel pattern matching. This hardware, The Pattern Matching Chip (PMC) [6], is a multiple instruction single data (MISD) parallel hardware on a PCI card. One PCI-card can match up to one thousand simple patterns against 100 MB of sequences per second, and it is quite straightforward to set up searches. Because of its efficiency and ease of use, we have used the PMC for this first implementation of the PAMM module. The PMC implementation covers both motif matching and organization of match scores.

Motif Matching. As the PMC only supports binary matching of patterns, and integer summation, the PWM match scores need to be discretized. The discretization is based on the fact that the log-likelihood for any base pair in any location is in the interval $\left[log(\frac{\beta}{n+4\beta}), log(\frac{n+\beta}{n+4\beta})\right]$, where β is the pseudo-count and n is the number of motif sites, given as parameters to MEME. Instead of using a fixed granulation of the interval, we define a granulation parameterized with ϵ. Then, each value $m_{i,x}$ in the PWM m is represented by a number $c_{i,x} = \lfloor \frac{log(m_{i,x}) - log(\frac{\beta}{n+4\beta})}{\epsilon} \rfloor$ of processing elements (PEs) in the specialized hardware. The number of PEs matching a symbol of the alphabet at a given position is thus proportional to the log-likelihood value of that symbol at that position. When the PWM is aligned with a sequence window, the sum of PE match scores at a motif position then corresponds to the *score* at that position. Note that since only one of the four nucleotides can match at a position, the other three do not contribute to the *score*. Furthermore, as PWM log-likelihood is the the sum of log-likelihoods for each position, the total PWM score is given by the sum of *scores* of all positions.

Two optimizations are worth mentioning. First, if the minimum score $c_i = min_x(c_{i,x})$ at a given position i is higher than zero, we may subtract c_i from each score value at that position, and then add c_i to the score after the search. Secondly, if $c = max_{i,x}(c_{i,x})$ is the maximum score value of the motif, and more score values are close to c than are close to zero, we then use transformed score values $c'_{i,x} = c - c_{i,x}$ and compute total PWM score as: $c \cdot I - \sum_i \sum_x c'_{i,x}$, where i runs over all I positions of m. Both optimizations give equivalent results to the basic method while using less PEs on the PMC, thus allowing more matrices to be matched simultaneously.

The discretization method considered above can be used generally for matching arbitrary PWMs against sequences. The approximation accuracy clearly depends on the granulation parameter ϵ. As discussed in section 3.1, the PWMs are regular in the first iteration of MEME. Motif matching can then be done with degenerate use of discretization, thereby avoiding approximation problems. To ensure that MAMA gives the same results as MEME, we have therefore only used hardware-acceleration in the first iteration, and used a standard software solution for motif matching in the remaining iterations. Since the running time of MEME is strongly dominated by the first iteration, we still achieve significant speed-ups.

Organizing Match Scores. As the PMC provides massive parallelity, we are able to calculate expectation values for many PWMs in parallel. We also use this parallelity to scan each PWM against the sequences several times with different hit thresholds. By searching with several thresholds in parallel, we can make the PMC return sequence offsets sorted by decreasing match score. This corresponds to a PAMM organizing module for post-processing of match scores, and avoids CPU-intensive sorting of offsets after the expectation step.

4 Results

We have compared the performance of our hardware accelerated version MAMA with the CPU based version of MEME on data sets of different sizes. On all test referred to here we have used the TCM model of MEME, which is the most general model and presented as the main model in the original MEME article [1]. We ran our tests with the following hardware configuration:

– MAMA: 2.8 Ghz Pentium4 PC with 1 GB memory and the specialized hardware on a single PCI card.
– MEME: 2.8 Ghz Pentium4 PC with 1 GB memory.
– ParaMEME: a cluster of 12 computers, each 3.4 Ghz Pentium4 PC with 1 GB memory.

We evaluated the performance of MAMA on the largest data set (mini-drosoph) supplied with MEME and on 5 data sets of human promoter regions, consisting of from 100 to 1600 sequences of 5000 base pair length from cell cycle regulated genes (J.P.Diaz, in preparation). Data sets, sizes and running times are given in Table 1 for both MEME, paraMEME and MAMA. We see that MAMA gives a

Table 1. Results for MEME, paraMEME and MAMA on 6 data sets

		Running time (hours)		
Data set	Size (Mbp)	MEME	paraMEME	MAMA
mini-drosoph	0.5	2.6	0.19	0.27
hs_100	0.5	2.7	0.20	0.23
hs_200	1	11	0.87	0.50
hs_400	2	104	3.6	1.7
hs_800	4	X[3]	15	6.4
hs_1600	8	X[3]	64	13

significant speed-up compared to MEME on all datasets, and that the speed-up increases with data set size. On the 1 Mbp (Million base pairs) data set, MAMA is more than twenty times as fast as MEME, and on the 8 Mbp data set it is even four times as fast as paraMEME on the 12-computer cluster. For all data sets, standard MEME and the hardware-accelerated version MAMA discovers the same motifs.

5 Discussion and Conclusion

We have proposed an abstract module PAMM for parallel hardware-acceleration of motif discovery. This module could be used for acceleration of many different motif discovery methods. The acceleration could be especially large if post-processing of match scores is tailored to a specific algorithm.

As an exemplification and proof-of-concept we have developed a version of the MEME algorithm called MAMA that uses available hardware to implement a PAMM module. As shown in section 4, MAMA achieves a speed-up of more than a factor of 10 as compared to MEME on a single CPU. Our working implementation thus shows that the PAMM module indeed has a potential.

Furthermore, our work shows examples of both problematic issues and potential rewards in connection with hardware acceleration of algorithms within bioinformatics. Since we have implemented weighted motif matching on hardware that was not specifically built for that purpose, we had to do some transformations of the problem. The issues and solutions with regards to discretization and parallelization are relevant for many algorithmic solutions involving specialized hardware.

A natural continuation of the work presented in this paper is to develop a FPGA-based implementation of PAMM. Such a solution would be more readily available for practical use and further refinement by the scientific community. It could potentially also give even higher speed-ups. On the other hand, such a solution presumes a solution of representing PWMs on FPGA that is both efficient and flexible. We have ongoing work in this direction that shows promising results.

[3] Not tested due to excessive running times.

References

1. Bailey, T.L., Elkan, C.: Fitting a mixture model by expectation maximization to discover motifs in biopolymers. Proc. Conf. Intell. Syst. Mol. Biol. ISMB'94 (1994) 28–36
2. Grundy, W.N., Bailey, T.L., Elkan, C.P.: ParaMEME: a parallel implementation and a web interface for a DNA and protein motif discovery tool. Comput. Appl. Biosci. **12** (1996) 303–310
3. Yamaguchi, Y., Miyajima, Y., Maruyama, T., Konagaya, A.: High speed homology search using run-time reconfiguration. LNCS. Volume 2438. (2002) 281–291
4. Oliver, T., Schmidt, B., Nathan, D., Clemens, R., Maskell, D.: Using reconfigurable hardware to accelerate multiple sequence alignment with ClustalW. Bioinformatics **21**(16) (2005) 3431–3432
5. Mak, T.S.T., Lam, K.P.: Embedded computation of maximum-likelihood phylogeny inference using platform FPGA. In Proc. Comput. Systems Bioinformatics Conf. CSB'04, IEEE. (2004) 512–514
6. Halaas, A., Svingen, B., Nedland, M., Sætrom, P., Snøve Jr., O., Birkeland, O.R.: A recursive MISD architecture for pattern matching. IEEE Trans. Very Large Scale Integr. Syst. **12**(7) (2004) 727–734
7. Marsan, L., Sagot, M.F.: Extracting structured motifs using a suffix tree-algorithms and application to promoter consensus identification. In: Proc. 4th Int'l Conf. Comput. Mol. Bio. RECOMB'00, ACM Press (2000) 210–219
8. Blanchette, M., Tompa, M.: Discovery of regulatory elements by a computational method for phylogenetic footprinting. Genome Res. **12**(5) (2002) 739–748
9. Sinha, S., Tompa, M.: YMF: A program for discovery of novel transcription factor binding sites by statistical overrepresentation. Nucleic Acids Res. **31**(13) (2003) 3586–3588
10. Bortoluzzi, S., Coppe, A., Bisognin, A., Pizzi, C., Danieli, G.: A multistep bioinformatic approach detects putative regulatory elements in gene promoters. BMC Bioinformatics **6**(1) (2005) 121
11. 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**(1) (1990) 41–51