Accelerating String Matching on MIC Architecture for Motif Extraction

Solon P. Pissis[1,2](✉), Christian Goll[2], Pavlos Pavlidis[3], and Alexandros Stamatakis[2]

[1] Florida Museum of Natural History, University of Florida, Gainesville, USA
solon.pissis@kcl.ac.uk
[2] Heidelberg Institute for Theoretical Studies, Heidelberg, Germany
[3] Foundation for Research and Technology – Hellas, Iraklio, Greece

Abstract. Identifying repeated factors that occur in a string of letters or common factors that occur in a set of strings represents an important task in computer science and biology. Such patterns are called *motifs*, and the process of identifying them is called *motif extraction*. In biology, motifs may correspond to functional elements in DNA, RNA, or protein molecules. In this article, we orchestrate `MoTeX`, a high-performance computing tool for MoTif eXtraction from large-scale datasets, on Many Integrated Core (MIC) architecture. `MoTeX` uses state-of-the-art algorithms for solving the fixed-length approximate string-matching problem. It comes in three flavors: a standard CPU version; an OpenMP version; and an MPI version. We compare the performance of our MIC implementation to the corresponding CPU version of `MoTeX`. Our MIC implementation accelerates the computations by a factor of ten compared to the CPU version. We also compare the performance of our MIC implementation to the corresponding OpenMP version of `MoTeX` running on modern Multicore architectures. Our MIC implementation accelerates the computations by a factor of two compared to the OpenMP version.

Keywords: Motif extraction · HPC · MIC architecture

1 Introduction

Identifying repeated factors that occur in a string of letters or common factors that occur in a set of strings represents an important task in computer science and biology. Such patterns are called *motifs*, and the process of identifying them is called *motif extraction*. Motif extraction has numerous direct applications in areas that require some form of *text mining*, that is, the process of deriving reliable information from text [5]. Here we focus on its application to molecular biology.

In biological applications, motifs correspond to functional and/or conserved DNA, RNA, or protein sequences. Alternatively, they may correspond to (recently, in evolutionary terms) duplicated genomic regions, such as transposable elements or even whole genes. It is mandatory to allow for a certain number of mismatches between different occurrences of the same motif since both, single nucleotide polymorphisms, as well as errors introduced by wet-lab sequencing platforms might have occurred. Hence, molecules that encode the same or related functions do not necessarily have *exactly* identical sequences.

A DNA motif is defined as a sequence of nucleic acids that has a specific biological function (e.g., a DNA binding site for a regulatory protein). The pattern can be fairly short, 5 to 20 base-pairs (bp) long, and is known to occur in different genes [7], or several times within the same gene [9]. The DNA motif extraction problem is the task of detecting overrepresented motifs as well as conserved motifs in a set of orthologous DNA sequences. Such conserved motifs may, for instance, be potential candidates for transcription factor binding sites.

A single *motif* is a string of letters (word) on an alphabet Σ. Given an integer error threshold e, a motif on Σ is said to *e-occur* in a string s on Σ, if the motif and a factor (substring) of y differ by a distance of e. In accordance with the pioneering work of Sagot [10], we formally define the common motifs problem as follows:

The *common motifs* problem takes as input a set $s_1, \ldots, s_N$ of strings on Σ, where $N \geq 2$, the quorum $1 \leq q \leq N$, the maximal allowed distance (error threshold) e, and the length k for the motifs. It consists in determining all motifs of length k, such that each motif e-occurs in at least q input strings. Such motifs are called *valid*. The values for k, e, and q are determined empirically.

In accordance with [3], motif extraction algorithms can be divided into two major classes: (1) *word-based* (string-based) methods that mostly rely on exhaustive enumeration, that is, counting and comparing oligonucleotide sequence (k-mer) frequencies; and (2) *probabilistic sequence* models, where the model parameters are estimated using maximum-likelihood or Bayesian inference methods.

Here, we focus on word-based methods for motif extraction. A plethora of word-based tools for motif extraction, such as RISO [6,10], YMF [11], Weeder [7], and RISOTTO [1], have already been released. The comprehensive study by Tompa *et al.* [12] compared thirteen different word-based and probabilistic methods on real and synthetic datasets, and identified Weeder and YMF—which are both word-based—as the most effective methods for motif extraction.

Very recently, we have introduced `MoTeX`, the first word-based HPC tool for MoTif eXtraction from large-scale datasets [8]. It can be used to tackle the common motifs problem by making a stricter assumption on motif validity, which we will elaborate later on. `MoTeX` is based on string algorithms for solving the so-called fixed-length approximate string-matching problem under the edit distance model [4] and under the Hamming distance model [2]. Given that $k \leq w$, where w is the size of the computer word (in practice, $w = 64$ or $w = 128$), the time complexity of this approach is $\mathcal{O}(N^2n^2)$ for the common motifs problem, where n is the average sequence length. The analogous parallel time complexity is $\mathcal{O}(N^2n^2/p)$, where p is the number of available processors.

Hence, `MoTeX` exhibits the following advantages: under the realistic assumption that $k \leq w$, time complexity does not depend on (i) the length k for the motifs (ii) the size $|\Sigma|$ of the alphabet, or (iii) the maximal allowed distance e. Given the stricter assumption on motif validity, it is guaranteed to find globally optimal solutions. Furthermore, the size of the output is linear with respect to the size of the input. In addition, MoTeX can identify motifs either under the edit distance model or the Hamming distance model. Finally, apart from the standard CPU version, MoTeX comes in two HPC flavors: the OpenMP-based version that supports the *symmetric multiprocessing programming* (SMP) paradigm; and the MPI-based version that supports the *message-passing programming* (MPP) paradigm.

In [8], we demonstrated that MoTeX can alleviate the shortcomings of current state-of-the-art tools for motif extraction from large-scale datasets. We showed how the quadratic time complexity of MoTeX can be slashed, in theory and in practice, by using parallel computations. The extensive experimental results presented in [8] are promising, both in terms of accuracy under statistical measures of significance as well as efficiency; a fact that suggests that further research and development of MoTeX is desirable. For instance, the MPI version of MoTeX requires about one hour to process the full upstream *Homo sapiens* genes dataset using 1056 processors, for *any* value of k and e, while current sequential programmes require more than two months for this task.

Our contribution. Many Integrated Core (MIC) architecture combines many cores onto a single chip, the *coprocessor*. One can write parallel programs, using the SMP paradigm, that can *offload* sections of code to run on the coprocessor—or alternatively, that run natively on the coprocessor. The compiler provides the language extensions to facilitate programming for MIC architecture such as *pragmas* to control the data transfer between the *host* CPU and the coprocessor. In this article, we orchestrate MoTeX on MIC architecture. We compare the performance of our MIC implementation, running on a single chip of MIC architecture, to the corresponding CPU version of MoTeX running on the host CPU. Our MIC implementation, using the full single-chip potential, accelerates the computations by a factor of ten compared to the CPU version. We also compare the performance of our MIC implementation to the corresponding OpenMP version of MoTeX running on a single chip of modern Multicore architectures. Our MIC implementation accelerates the computations by a factor of two compared to the OpenMP version, both using the full single-chip potential.

2 Definitions and Notation

In this section, in order to provide an overview of the algorithms used later on, we give a few definitions.

An *alphabet* Σ is a finite non-empty set whose elements are called *letters*. A *string* on an alphabet Σ is a finite, possibly empty, sequence of elements of Σ. The zero-letter sequence is called the *empty string*, and is denoted by ε. The *length* of a string x is defined as the length of the sequence associated with the string x, and is denoted by $|x|$. We denote by $x[i]$, for all $1 \leq i \leq |x|$, the letter at index i of x. Each index i, for all $1 \leq i \leq |x|$, is a position in x when $x \neq \varepsilon$. It follows that the ith letter of x is the letter at position i in x, and that $x = x[1 \mathinner{.\,.} |x|]$.

A string x is a *factor* of a string y if there exist two strings u and v, such that $y = uxv$. Let the strings x, y, u, and v, such that $y = uxv$. If $u = \varepsilon$, then x is a *prefix* of y. If $v = \varepsilon$, then x is a *suffix* of y.

Let x be a non-empty string and y be a string. We say that there exists an (exact) *occurrence* of x in y, or, more simply, that x *occurs* (exactly) in y, when x is a factor of y. Every occurrence of x can be characterised by a position in y. Thus we say that x occurs at the *starting position* i in y when $y[i \mathinner{.\,.} i+|x|-1] = x$. It is sometimes more suitable to consider the *ending position* $i + |x| - 1$.

The *edit distance*, denoted by $\delta_E(x, y)$, for two strings x and y is defined as the minimum total cost of operations required to transform string x into string y. For simplicity, we only count the number of edit operations and consider that the cost of each edit operation is 1. The allowed operations are the following:

- `ins`: insert a letter in y, not present in x; $(\varepsilon, b),\ b \neq \varepsilon$;
- `del`: delete a letter in y, present in x; $(a, \varepsilon),\ a \neq \varepsilon$;
- `sub`: substitute a letter in y with a letter in x; $(a, b),\ a \neq b,\ a, b \neq \varepsilon$.

The *Hamming distance* δ_H is only defined on strings of the same length. For two strings x and y, $\delta_H(x, y)$ is the number of positions in which the two strings differ, that is, have different letters.

3 Algorithms

In this section, we first formally define the *fixed-length approximate string-matching* problem under the edit distance model and under the Hamming distance model. We then provide a brief description and analysis of the sequential algorithms to solve it. Finally, we show how the common motifs problem can be reduced to the fixed-length approximate string-matching problem, by using a stricter assumption than the one in the initial problem definition for the validity of motifs.

Problem 1 (Edit distance). Given a string x of length m, a string y of length n, an integer k, and an integer $e < k$, find all factors of y, which are at an edit distance less than, or equal to, e from every factor of fixed length k of x.

Problem 2 (Hamming distance). Given a string x of length m, a string y of length n, an integer k, and an integer $e < k$, find all the factors of y, which are at a Hamming distance less than, or equal to, e from every factor of fixed length k of x.

Let $\mathsf{D}[0 \mathinner{.\,.} n, 0 \mathinner{.\,.} m]$ be a dynamic programming (DP) matrix, where $\mathsf{D}[i, j]$ contains the edit distance between some factor $y[i' \mathinner{.\,.} i]$ of y, for some $1 \leq i' \leq i$, and factor $x[\max\{1, j-k+1\} \mathinner{.\,.} j]$ of x, for all $1 \leq i \leq n$, $1 \leq j \leq m$. This matrix can be obtained through a straightforward $\mathcal{O}(kmn)$-time algorithm by constructing DP matrices $\mathsf{D}^s[0 \mathinner{.\,.} n, 0 \mathinner{.\,.} k]$, for all $1 \leq s \leq m-k+1$, where $\mathsf{D}^s[i, j]$ is the edit distance between some factor of y ending at $y[i]$ and the prefix of length j of $x[s \mathinner{.\,.} s+k-1]$. We obtain D by collating D^1 and the last row of D^s, for all $2 \leq s \leq m-k+1$. We say that $x[\max\{1, j-k+1\} \mathinner{.\,.} j]$ e-occurs in y ending at $y[i]$ *iff* $\mathsf{D}[i, j] \leq e$, for all $1 \leq j \leq m$, $1 \leq i \leq n$.

Table 1. Matrix D and matrix M

		0	1	2	3	4	5	6	7
		ϵ	G	G	G	T	C	T	A
0	ϵ	0	1	2	3	3	3	3	3
1	G	0	0	1	2	2	2	3	3
2	G	0	0	0	1	1	2	3	3
3	G	0	0	0	0	1	2	3	3
4	T	0	1	1	1	0	1	2	2
5	C	0	1	2	2	1	0	1	2
6	T	0	1	2	3	2	1	0	1
7	A	0	1	2	3	3	2	1	0

(a) Matrix D for $x := y :=$ GGGTCTA and $k := 3$

		0	1	2	3	4	5	6	7
		ϵ	G	T	G	A	A	C	T
0	ϵ	0	1	2	3	3	3	3	3
1	G	0	0	2	2	3	3	3	3
2	T	0	1	0	3	2	3	3	2
3	C	0	1	2	1	3	2	2	3
4	A	0	1	2	3	1	2	3	2
5	C	0	1	2	3	3	2	1	3
6	G	0	0	2	2	3	3	2	1
7	T	0	1	0	3	2	3	3	2

(b) Matrix M for $x :=$ GTCACGT, $y :=$ GTGAACT, and $k := 3$

Example 1. Let the string $x :=$ GGGTCTA, the string $y := x$, and $k := 3$. Table 1a illustrates matrix D. Consider, for instance, the case where $j = 6$. Column 6 contains all e-occurrences of factor $x[4 \mathinner{.\,.} 6] =$ TCT, that is the factor of length $k = 3$ ending at position 6 of x, in y. Cell $\mathsf{D}[4, 6] = 2$, tells us that there exists some factor $y[i' \mathinner{.\,.} 4]$ of y, such that, for $i' = 2$, $\delta_E(y[2 \mathinner{.\,.} 4], x[4 \mathinner{.\,.} 6]) = 2$.

Iliopoulos, Mouchard, and Pinzon devised MaxShift [4], an algorithm with time complexity $\mathcal{O}(m\lceil k/w \rceil n)$, where w is the size of the computer word. By using word-level parallelism MaxShift can compute matrix D efficiently. The algorithm requires constant time for computing each cell $\mathsf{D}[i, j]$ by using word-level operations, assuming that $k \leq w$. In the general case it requires $\mathcal{O}(\lceil k/w \rceil)$ time. Hence, algorithm MaxShift requires time $\mathcal{O}(mn)$, under the assumption that $k \leq w$. The space complexity is $\mathcal{O}(m)$ since each row of D only depends on the immediately preceding row.

Theorem 1 ([4]). *Given a string x of length m, a string y of length n, an integer k, and the size of the computer word w, matrix D can be computed in time $\mathcal{O}(m\lceil k/w \rceil n)$.*

Let $\mathsf{M}[0 \mathinner{.\,.} n, 0 \mathinner{.\,.} m]$ be a DP matrix, where $\mathsf{M}[i, j]$ contains the Hamming distance between factor $y[\max\{1, i - k + 1\} \mathinner{.\,.} i]$ of y and factor $x[\max\{1, j - k + 1\} \mathinner{.\,.} j]$ of x, for all $1 \leq i \leq n$, $1 \leq j \leq m$. Crochemore, Iliopoulos, and Pissis devised an analogous algorithm [2] that solves the analogous problem under the Hamming distance model with the same time and space complexity.

Theorem 2 ([2]). *Given a string x of length m, a string y of length n, an integer k, and the size of the computer word w, matrix M can be computed in time $\mathcal{O}(m\lceil k/w \rceil n)$.*

Example 2. Let the string $x :=$ GTGAACT, the string $y :=$ GTCACGT, and $k := 3$. Table 1b illustrates matrix M. Consider, for instance, the case where $j = 7$.

Column 7 contains all e-occurrences of factor $x[5\,..\,7] = \texttt{ACT}$, that is, the factor of length $k = 3$ ending at position 7 of x, in y. Cell $\mathsf{M}[6,7] = 1$, tells us that $\delta_H(y[4\,..\,6], x[5\,..\,7]) = 1$.

By making the following stricter assumption for motif validity, the common motifs problem can be directly and efficiently solved using the above algorithms.

Definition 1. *A valid motif is called* strictly valid iff *it occurs* exactly*, at least once, in (any of) the input strings.*

Consider, for instance, the DNA alphabet $\Sigma = \{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{T}\}$. The number of possible DNA motifs of length $k := 10$ is $|\Sigma|^k = 1{,}048{,}576$. Given a dataset with a size of $\approx$1 MB, the possibility that there exists a motif that is valid, but not strictly valid, is rather unlikely. In other words, given such a dataset, the possibility that there exists a pattern which does not occur exactly, at least once, in the dataset *and* it also satisfies *all* the restrictions imposed by the input parameters, is rather unlikely.

The common motifs problem (detecting strictly valid motifs) can be directly solved by solving the fixed-length approximate string-matching problem for all N^2 pairs of the N input strings. Consider, for example, the common motifs problem under the Hamming distance model. We use an array u_r for each input string s_r, such that for all $1 \leq r \leq N$, $k \leq j \leq |s_r|$, $u_r[j]$ contains the total number of strings in which motif $s_r[j-k+1\,..\,j]$ e-occurs; we set $u_r[j] := 0$, for all $0 \leq j < k$. Array u_r, for all $1 \leq r \leq N$, can easily be computed, by computing matrix M for pair (s_r, s_t), for all $1 \leq t \leq N$. While computing matrix M, we increment $u_r[j]$ only once *iff* $\mathsf{M}[i,j] \leq e$, for some $k \leq i \leq |s_t|$; as soon as we have computed the N different matrices M for s_r, it suffices to iterate over array u_r and report $s_r[j-k+1\,..\,j]$, for all $k \leq j \leq |s_r|$, as a strictly valid motif *iff* $u_r[j] \geq q$. An array v_r, such that $v_r[j]$, for all $1 \leq j \leq |s_r|$, denoting the total number of e-occurrences of motif $s_r[j-k+1\,..\,j]$ in $s_1, \ldots, s_N$ can also be maintained. Maintaining arrays u_r and v_r does not induce additional costs. Therefore, the common motifs problem can be solved in time $\mathcal{O}(n^2)$ per matrix, where n is the average length of the N strings, thus $\mathcal{O}(N^2n^2)$ in total.

Example 3. Let the input strings $s_r := \texttt{GTGAACT}$, $s_t := \texttt{GTCACGT}$, $e := 1$, $q := 2$, and $k := 3$. Further, let the current state of arrays u_r and v_r be:

$$\begin{array}{r@{\,:\,}l} j & 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7 \\ u_r[j] & 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0 \\ v_r[j] & 0\ 0\ 0\ 0\ 2\ 0\ 2\ 0 \end{array}$$

Table 1b illustrates matrix M. Arrays u_r and v_r are:

$$\begin{array}{r@{\,:\,}l} j & 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7 \\ u_r[j] & 0\ 0\ 0\ 1\ 2\ 0\ 2\ 1 \\ v_r[j] & 0\ 0\ 0\ 1\ 3\ 0\ 3\ 1 \end{array}$$

and so the strictly valid motifs are $s_r[2\,..\,4] = \texttt{TGA}$ and $s_r[4\,..\,6] = \texttt{AAC}$.

4 Implementation

`MoTeX` is implemented as a programme that solves the common motifs problem for strictly valid motifs. `MoTeX` was implemented in the C programming language under a GNU/Linux system. It is distributed under the GNU General Public License (GPL). The open-source code and documentation are available at http://www.exelixis-lab.org/motex. The mandatory input parameters are:

– a file with the N input strings in FASTA format (sequences);
– the length k for the motifs;
– the distance d ($d := 0$ for Hamming distance *or* $d := 1$ for edit distance);
– the maximal allowed distance e;
– the quorum $q' \leq 100$ (%) as the ratio of quorum q to N.

Given these parameters, `MoTeX` outputs a text file containing the strictly valid motifs. For each reported motif, it also outputs the total number of sequences in which the motif e-occurs at least once and the total number of its e-occurrences.

Apart from the standard CPU version, `MoTeX` comes in two HPC flavors: the OpenMP version for shared memory systems and the MPI version for distributed memory systems. The user can choose the best-suited version depending on: the total size of the input sequences; the nature of the input dataset, for instance, a few very long sequences or many relatively short sequences; the available HPC architecture; and the number $p > 1$ of available processors.

Here we focus on the case when $p \leq N$, where N is the number of input sequences; that is, we have a large number of relatively short sequences. The user can choose any of the two HPC versions. `MoTeX` evenly distributes the computation of the N^2 distinct DP matrices for the N input sequences in a straightforward manner across the p processors. Therefore, if $p \leq N$, the common motifs problem for strictly valid motifs can be solved in parallel in time $\mathcal{O}(N^2 n^2/p)$.

4.1 MIC Implementation

Our MIC implementation is a parallel program that uses the SMP paradigm by offloading sections of the code to run on the coprocessor. First, we used the compiler option `offload-attribute-target` to flag every global routine and global data object in the source file with the offload attribute `target(mic)`.

The compiler supports two programming models: a *non-shared memory* model and a *virtual-shared memory model*. In our implementation, we used the non-shared memory model which is appropriate for dealing with flat data structures such as scalars, arrays, and structures that are bit-wise copyable. Data in this model is copied back and forth between the host CPU and the coprocessor around regions of the offloaded code. The data selected for transfer is a combination of variables implicitly transferred because they are lexically referenced within offload constructs, and variables explicitly listed in clauses in the pragma. Only pointers to non-pointer types are supported—pointers to pointer variables are not supported. Arrays are supported provided the array element type is a scalar or bitwise copyable structure or class—so arrays of pointers are not supported. We therefore defined the following flat data structures:

- Sequence $\mathsf{S} = s_1 s_2 \dots s_N$ of size nN, where s_i is an input sequence, for all $1 \leq i \leq N$, and n is the average sequence length;
- Array L of size N, such that $\mathsf{L}[i]$ stores the length of s_{i+1}, for all $0 \leq i < N$;
- Array I of size N, such that $\mathsf{I}[i]$ stores the starting position of s_{i+1} in S, for all $0 \leq i < N$;
- Array U of size nN, such that $\mathsf{U}[\mathsf{I}[i] \mathinner{.\,.} \mathsf{L}[i] - 1]$ stores array u (see Sect. 3 for details) of s_{i+1}, for all $0 \leq i < N$;
- Array V of size nN, such that $\mathsf{V}[\mathsf{I}[i] \mathinner{.\,.} \mathsf{L}[i] - 1]$ stores array v (see Sect. 3 for details) of s_{i+1}, for all $0 \leq i < N$.

Then we placed the offload pragma before the code block computing the N^2 distinct DP matrices for the N input sequences. While the instruction sets for the host CPU and the coprocessor are similar, they do not share the same system memory. This means that the variables used by the code block must exist on both the host CPU and coprocessor. To ensure that they do, the pragmas use specifiers to define the variables to copy between the host CPU and coprocessor:

- `in` specifier defines a variable as strictly an input to the coprocessor. The value is not copied back to the host CPU. S, L, and I were defined by `in`;
- `out` specifier defines a variable as strictly an output of the coprocessor. U and V were defined by `out`.

Therefore it becomes obvious that S, L, I, U, and V are copied *either* back *or* forth between the host CPU and the coprocessor at most once.

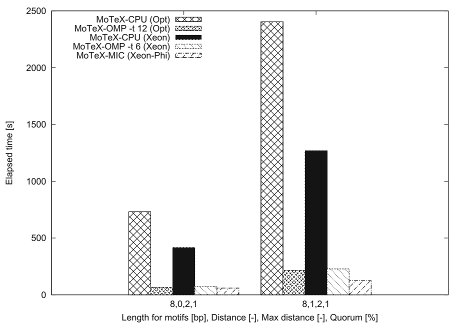
Finally, the code-block statement following the offload pragma is converted into an outlined function that runs on the coprocessor. This code is permitted to call other functions. In order to ensure that these called functions are also available on the coprocessor, we marked the functions to be called by the offloaded code block with the special function attribute `__declspec(target (mic))`.

5 Experimental Results

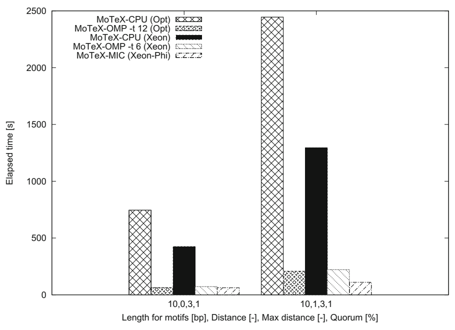
The following experiments were conducted on a GNU/Linux system running on:

- **Multicore architecture I:** a single AMD Opteron 6174 Magny-Cours CPU at 2.20 GHz with 12 cores;
- **Multicore architecture II:** a single Intel Xeon CPU E5-2630 0 at 2.30 GHz with 6 cores;
- **MIC architecture:** a single Intel Xeon host CPU E5-2630 0 at 2.30 GHz with a single Intel Xeon Phi 5110P coprocessor at 1.053 GHz with 60 cores.

We evaluated the time performance of our MIC implementation, denoted by `MoTeX-MIC (Xeon-Phi)`, running on the host CPU *and* the coprocessor against: (i) the standard CPU version of `MoTeX`, denoted by `MoTeX-CPU (Opt)`, and



(a) 1062 sequences of *Bacillus subtilis*



(b) 200 sequences of *Homo sapiens*

Fig. 1. Elapsed-time comparisons for the common motifs problem

(ii) the OpenMP version with 12 threads, denoted by `MoTeX-OMP -t 12 (Opt)`, both running on the Multicore architecture I; and (iii) the standard CPU version of `MoTeX`, denoted by `MoTeX-CPU (Xeon)`, and (iv) the OpenMP version with 6 threads, denoted by `MoTeX-OMP -t 6 (Xeon)`, both running on the Multicore architecture II. As input datasets for the programmes, we used 1062 upstream sequences of *Bacillus subtilis* genes of total size 240 KB and 200 and 1200 upstream sequences of *Homo sapiens* genes of total size 240 KB and 1.2 MB, respectively, obtained from the ENSEMBL database. We measured the elapsed time for each programme for different combinations of input parameters. In particular, we provided different values for the motif length k, the distance d used, the maximal allowed distance e, and the quorum q' as a proportion of sequences in the input dataset.

As depicted in Figs. 1 and 2, our MIC implementation accelerates the computations by a factor of ten compared to the corresponding CPU version and by a factor of two compared to the OpenMP version. We observe that, similar to other architectures such as GPGPU, for the same input dataset, the speedup gained

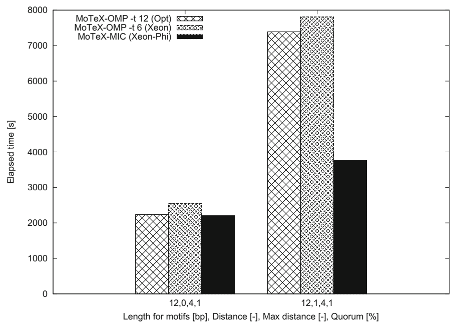


Fig. 2. Elapsed-time comparison for the common motifs problem using 1200 sequences of *Homo sapiens*

from our MIC implementation increases as the ratio of the workload to the size of the data transferred between the host CPU and the coprocessor increases. For instance, notice that in Fig. 2, while the time efficiency of `MoTeX-OMP -t 12 (Opt)`, `MoTeX-OMP -t 6 (Xeon)`, and `MoTeX-MIC (Xeon-Phi)` is similar for Hamming distance, the later programme becomes faster by a factor of two with the same dataset for edit distance, which is computationally more intensive.

References

1. Pisanti, N., Carvalho, A.M., Marsan, L., Sagot, M.-F.: RISOTTO: fast extraction of Motifs with mismatches. In: Correa, J., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 757–768. Springer, Heidelberg (2006)
2. Crochemore, M., Iliopoulos, C.S., Pissis, S.P.: A parallel algorithm for fixed-length approximate string-matching with k-mismatches. In: Elomaa, T., Mannila, H., Orponen, P. (eds.) Ukkonen Festschrift 2010. LNCS, vol. 6060, pp. 92–101. Springer, Heidelberg (2010)
3. Das, M., Dai, H.K.: A survey of DNA motif finding algorithms. BMC Bioinform. **8**(Suppl 7), S21+ (2007)
4. Iliopoulos, C.S., Mouchard, L., Pinzon, Y.J.: The Max-Shift algorithm for approximate string matching. In: Brodal, G., Frigioni, D., Marchetti-Spaccamela, A. (eds.) WAE 2001. LNCS, vol. 2141, pp. 13–25. Springer, Heidelberg (2001)
5. Lothaire, M. (ed.): Applied Combinatorics on Words. Cambridge University Press, New York (2005)
6. Marsan, L., Sagot, M.F.: Algorithms for extracting structured motifs using a suffix tree with an application to promoter and regulatory site consensus identification. J. Comput. Biol. J. Comput. Mol. Cell Biol. **7**(3–4), 345–362 (2000)
7. Pavesi, G., Mereghetti, P., Mauri, G., Pesole, G.: Weeder web: discovery of transcription factor binding sites in a set of sequences from co-regulated genes. Nucleic Acids Res. **32**(Web-Server-Issue), 199–203 (2004)
8. Pissis, S.P., Stamatakis, A., Pavlidis, P.: MoTeX: a word-based HPC tool for MoTif eXtraction. In: Gao, J. (ed.) Fourth ACM International Conference on Bioinformatics and Computational Biology (ACM-BCB 2013), pp. 13–22 (2013)
9. Rombauts, S., Déhais, P., Van Montagu, M., Rouzé, P.: PlantCARE, a plant cis-acting regulatory element database. Nucleic Acids Res. **27**(1), 295–296 (1999)
10. Sagot, M.-F.: Spelling approximate repeated or common Motifs using a suffix tree. In: Lucchesi, C., Moura, A.V. (eds.) LATIN 1998. LNCS, vol. 1380, pp. 374–390. Springer, Heidelberg (1998)
11. Sinha, S., Tompa, M.: YMF: a program for discovery of novel transcription factor binding sites by statistical verrepresentation. Nucleic Acids Res. **31**(13), 3586–3588 (2003)
12. Tompa, M., Li, N., Bailey, T.L., Church, G.M., De Moor, B., Eskin, E., Favorov, A.V., Frith, M.C., Fu, Y., Kent, W.J., Makeev, V.J., Mironov, A.A., Noble, W.S., Pavesi, G., Pesole, G., Regnier, M., Simonis, N., Sinha, S., Thijs, G., van Helden, J., Vandenbogaert, M., Weng, Z., Workman, C., Ye, C., Zhu, Z.: Assessing computational tools for the discovery of transcription factor binding sites. Nat. Biotechnol. **23**(1), 137–144 (2005)