Parallel algorithms for degenerate and weighted sequences derived from high throughput sequencing technologies

Costas S. Iliopoulos[1,2], Mirka Miller[1,3,4], and Solon P. Pissis[1]

[1] Dept. of Computer Science, King's College London, London WC2R 2LS, England
csi@dcs.kcl.ac.uk, solon.pissis@kcl.ac.uk
[2] Digital Ecosystems & Business Intelligence Institute, Curtin University, GPO Box U1987 Perth WA 6845, Australia
[3] School of Electrical Engineering and Computer Science, The University of Newcastle Callaghan NSW 2308, Australia
[4] Dept. of Mathematics, University of West Bohemia, Pilsen, Czech Republic
mirka.miller@newcastle.edu.au

Abstract. Novel high throughput sequencing technologies have redefined the way genome sequencing is performed. They are able to produce millions of short sequences in a single experiment and with a much lower cost than previous methods. In this paper, we address the problem of efficiently mapping and classifying millions of degenerate and weighted sequences to a reference genome, based on whether they occur exactly once in the genome or not, and by taking into consideration probability scores. In particular, we design parallel algorithms for *Massive Exact and Approximate Unique Pattern Matching* for degenerate and weighted sequences derived from high throughput sequencing technologies.

Keywords: parallel algorithms, string algorithms, high throughput sequencing technologies

1 Introduction

The computational biology applications that have been developed for decades are strongly related to the technology that generates the data they consider. The algorithms, and the application parameters, are tuned in such a way that they abolished intrinsic limitations of the technology. As an example, the length of the data to be processed, or the quality/error rate that accompanies this data, are crucial elements that are considered for choosing the appropriate data structure for preprocessing, storing, analyzing, and comparing sequences. Moreover, the way solutions are improved reflects both computer science and biotechnology advances.

Among the large number of equipment that produce data, the DNA sequencers play a central role. DNA sequencing is the generic term for all biochemical methods that determine the order of the nucleotide bases in a DNA sequence. It consists of obtaining (generally relatively short) fragments of a DNA sequence (typically less than a thousand bp - base pair). The Sanger sequencing method [17,18] has been the workhorse technology for DNA sequencing for almost 30 years. It has been slowly replaced by technologies that used different colored fluorescent dyes [16,22] and polyacrylamide gels. Later, the gels were replaced by capillaries, increasing the length of individual obtained fragments from 450 to 850 bp. Despite the many technological advances, obtaining the complete sequence of a genome was carried out in very large

Costas S. Iliopoulos, Mirka Miller, Solon P. Pissis: Parallel algorithms for degenerate and weighted sequences derived from high throughput sequencing technologies, pp. 249–262.
Proceedings of PSC 2009, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-04403-2

dedicated "sequencing factories", which require hundreds of automatic sequencers using highly automated pipelines.

Along the years, sophisticated algorithms have been developed for assembling whole genomes, from a simple bacterial genome [6] to the human genome [7]. These algorithms were following the progress of the sequencing technologies, and were fully taking into account all the biases introduced by the equipment.

Very recent advances, based either on sequencing-by-synthesis (SBS) or on hybridization and ligation, are producing millions of short reads overnight. Depending on the technology (454 Life Science, Solexa/Illumina or Polony Sequencing, to name a few), the size of the fragments can range from a dozen of base pairs to several hundreds.

These high throughput sequencing technologies have the potential to assemble a bacterial genome during a single experiment and at a moderate cost [8] and are aimed in sequencing DNA genomic sequences. One such technology, Pyrosequencing^{TM}, massively parallelises the sequencing via microchip sensors and nanofluids, and it produces reads that are approximately 200 bp long, and may not improve beyond 300 bp in the near future [24]. In contrast, the technology developed by the Solexa/Illumina [2], generates millions of very short mate-pair reads ranging from 25 [26] to 50 [8] bp long, although in the future this number may be increased to 75. The results of these new technologies mark the beginning of a new era of high throughput short read sequencing that moves away form the traditional Sanger methods. The common denominator of these technologies is the fact that they are able to produce a massive amount of relatively short reads. Due to this massive amount of data generated by the above systems, efficient algorithms for mapping short sequences to a reference genome are in great demand.

Popular alignment programs like *BLAST* or *BLAT* are not successful because they focus on the alignment of fewer and longer sequences [11]. Recently, a new thread of applications addressing the short sequences mapping problem has been devised for this particular objective. These applications are based on the pigeonhole principle, and make use of hashing and short key indexing techniques.

ELAND is the mapping algorithm developed as part of the Illumina pipeline. It is optimized to map very short reads of $20 - 32$ bp ignoring additional bases when the reads are longer, whilst allowing at most two mismatches between the read and the genomic sequence [21]. *SOAP* [14] supports multi-threaded parallel computing and allows up to two mismatches, or a gap of 1-3 bp without any other mismatch. *SeqMap* [11] allows up to 5 mixed mismatches and inserted/deleted nucleotides in mapping.

RMAP [21] and *MAQ* [13] are ungapped mapping programs, which take read qualities, base position probabilities, and mate-pair information into account. Their strategies resemble the strategies developed by Ewing *et al.* in [5], where at each position of the reads a quality score is assigned, which encodes the probability that the base at that position is either rightly or wrongly positioned.

The last two applications show the necessity for a measure of accuracy concerning the mapping methods. Accuracy can be quantified in terms of sensitivity and specificity. Possible causes of limitations in the accuracy of these experiments include sequencing errors arising from any part of the high throughput experiment, variation between sampled genome that generated the reads and the reference genome, as well as ambiguities caused by repeats in the reference genome [21].

Therefore, the limitations of the equipment used, or the natural polymorphisms that can be observed between individual samples can give rise to uncertain sequences, where in some positions more than one nucleotide can be present. These sequences, where more than one base are possible in certain positions, are called *degenerate* or *indeterminate* sequences.

Figure 1 presents the sequence logo of a degenerate DNA sequence, which is the consensus DNA sequence derived from different reads at the same location. In this consensus degenerate sequence, one can note that in some positions more than one base occurs, and in fact all bases (A, C, G, T in the case of DNA sequences) may occur.

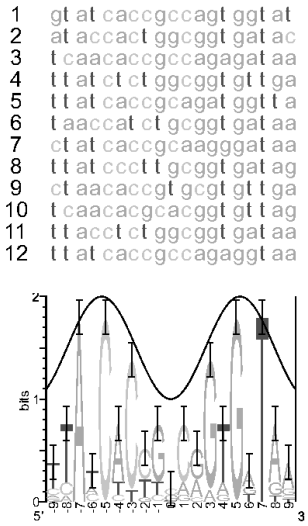

Figure 1. A sequence logo of a biological degenerate sequence. Picture taken from [19].

Degenerate string pattern matching has mainly been handled by bit mapping techniques (Shift-Or method) [1,28]. These techniques have been used to find matches for a degenerate pattern in a string [9], and the *agrep* utility [27] has been virtually one of the few practical algorithms available for degenerate pattern matching.

Very often, each position of a sequence is accompanied by probabilities of each base occuring in the specific position. In the case of the high throughput experiments, these quality scores, which accompany the raw sequence data, describe the confidence of bases in each read [21]. The sequencing quality scores assign a probability to the four possible nucleotides for each sequenced base. Bases with low quality scores are more likely to be sequencing errors. These sequences, where the probability of every symbol's occurrence at every location is given, are called *weighted* sequences.

Weighted sequences are also used to represent relatively short sequences such as binding sites, as well as long sequences such as protein families profiles [3]. Additionally, they have been used to represent complete chromosome sequences that were obtained using the traditional method of whole-genome shotgun strategy.

In this paper, we present parallel algorithms for addressing the problem of efficiently mapping uniquely occuring short reads to a reference genome. In particular, we design parallel algorithms for *Massive Exact and Approximate Unique Pattern Matching* for degenerate and weighted sequences derived from high throughput se-

quencing technologies. Our approach differs from the above mapping programs in three key points:

- it preprocesses the genomic sequence based on the reads length, by using word-level parallelism, before mapping the reads to it. This provides efficiency to the method.
- it does not index and hash the reads, but instead it converts each read to a unique arithmetic value. This results to a much higher sensitivity in terms of the number of reads perfectly mapped to the reference genome.
- it directly classifies the mapped reads into unique and duplicate matches, i.e. into reads that occur exactly once in the genome and into reads that occur more than once. The uniqueness of a mapped read guarantees an adequate placement on the sequence, and provides anchors that will be used for placing mate-pair reads, and other connected reads as well. It also identifies something that is totally region specific, while most of the genome is repetitive.

The rest of the paper is structured as follows. In Section 2, we present the preliminaries. In Section 3, we define the problem of Massive Exact and Approximate Unique Pattern Matching for degenerate and weighted sequences. In Section 4 and Section 5, we present the parallel algorithms for solving the exact and the approximate case, respectively. Finally, we briefly conclude with some future work in Section 6.

2 Preliminaries

A *string* is a sequence of zero or more symbols from an alphabet Σ. The set of all strings over Σ is denoted by Σ^*. The length of a string x is denoted by $|x|$. The *empty* string, that is the string of length zero, is denoted by ϵ. The i-th symbol of a string x is denoted by $x[i]$.

A string w is a *substring* of x if $x = uwv$, where $u, v \in \Sigma^*$. We denote by $x[i \dots j]$ the substring of x that starts at position i and ends at position j. Conversely, x is called a *superstring* of w. A string w is a *prefix* of x if $x = wy$, for $y \in \Sigma^*$. Similarly, w is a *suffix* of x if $x = yw$, for $y \in \Sigma^*$.

In this work, we are considering the finite alphabet Σ for DNA sequences, where $\Sigma = \{A, C, G, T\}$.

A *degenerate* string is a sequence $t = t[1 \dots n]$, where $t[i] \subseteq \Sigma$ for each i. When a position of the string is degenerate, and it can match more than one element from the alphabet Σ, we say that this position has *non-solid* symbol. If in a position only one element of the alphabet Σ is present, we refer to this symbol as *solid*.

A *weighted* string over alphabet Σ is a sequence $s = s[1 \dots n]$ of sets of couples. In particular, each $s[i]$ is a set $((q_1, \pi_i(q_1)), (q_2, \pi_i(q_2)), \dots, (q_{|\Sigma|}, \pi_i(q_{|\Sigma|}))$, where $\pi_i(q_j)$ is the occurrence probability of character q_j at position i. A symbol q_j occurs at position i of a weighted sequence $s = s[1 \dots n]$ if and only if the probability of occurrence of symbol q_j at position i is greater than zero, i.e. $\pi_i(q_j) > 0$. For every position $1 \leq i \leq n$, $\sum_{j=1}^{|\Sigma|} \pi_i(q_j) = 1$. For example, $\binom{A\ 0.8}{C\ 0.2}$ is a non-solid symbol, implying that base A occurs with probability 80 % and C with probability 20 %.

3 Problems definition

We denote the generated short reads as the set $p_0, p_1, \dots, p_{r-1}$ and we call them *patterns*. Notice that r is a very large integer number ($r > 10^7$). Due to the massive

amount of data, specialized solutions are needed to various sequencing-related problems. The length ℓ of each pattern is nowadays typically between 25 and 50 bp long, and we denote that constant range, without loss of generality, as $\ell_{min} \leq \ell \leq \ell_{max}$. We assume that the data is derived from high quality sequencing methods and therefore we will consider patterns with at most $\mu = 3$ non-solid symbols. We are given a genomic solid sequence $t = t[1 \dots n]$ and a positive threshold $k \geq 0$.

We define the *Massive Exact and Approximate Unique Pattern Matching* problem for degenerate and weighted sequences as follows.

Problem 1.
Find whether the degenerate pattern $p_i = p_i[1 \dots \ell]$, for all $0 \leq i < r$, of length $\ell_{min} \leq \ell \leq \ell_{max}$, with at most μ non-solid symbols, occurs with at most k-mismatches in $t = t[1 \dots n]$, exactly once.

Problem 2.
Find whether the weighted pattern $p_i = p_i[1 \dots \ell]$, for all $0 \leq i < r$, of length $\ell_{min} \leq \ell \leq \ell_{max}$, with at most μ non-solid symbols, occurs with at most k-mismatches in $t = t[1 \dots n]$, exactly once, with probability at least c, if $\prod_{i=1}^{\ell} \pi_i(q_i) \geq c$.

We mainly focus on the following classes of both problems:

Class 1. p_i occurs in t exactly once
Class 2. p_i occurs with at most 1-mismatch in t, exactly once
Class 3. p_i occurs with at most 2-mismatches in t, exactly once

Class 2 and Class 3 correspond to cases where the pattern either contains a sequencing error (quality score associated with read is indicating it), or a small difference between a mutant and the reference genome, which will have an impact on the proteins that have to be translated, as explored in [23,25].

4 Massive Exact Unique Pattern Matching in Parallel

In this section, we solve the problem of *Class 1*. The focus is to find occurrences of pattern p_i, for all $0 \leq i < r$, in text $t = t[1 \dots n]$. In particular, we are interested in whether p_i occurs in t exactly once.

The proposed algorithm makes use of the message-passing paradigm, by using p processing elements. The following assumptions for the model of communications in the parallel computer are made. The parallel computer comprises a number of nodes. Each node comprises one or several identical processors interconnected by a switched communication network. The time taken to send a message of size n between any two nodes is independent of the distance between nodes and can be modelled as $t_{comm} = t_s + nt_w$, where t_s is the latency or start-up time of the message, and t_w is the transfer time per data. The links between two nodes are full-duplex and single-ported: a message can be transferred in both directions by the link at the same time, and only one message can be sent and one message can be received at the same time.

In addition, the proposed parallel algorithm makes use of word-level parallelism by compacting strings into single computer words that we call *signatures*. We get the signature $\sigma(x)$ of a string x, by transforming it to its binary equivalent using 2-bits-per-base encoding of the DNA alphabet (see Table 1), and packing its decimal value into a computer word (see Table 2).

The idea of employing signatures is long known to computer scientists, introduced by Dömölki in [4] in 1964 for his Shift-Or algorithm, a string matching algorithm based on only few bitwise logical operations. The most well known application is the *four Russians* algorithm, which packs rows of boolean matrices into computer words speeding up boolean matrix multiplication. A randomised version of *fingerprints* (modulo a prime number) was employed by Karp and Rabin in [12] for solving the pattern matching problem. Their method cannot be used in our pattern matching problem as our signatures are small and, thus, there is no practical speed up by reducing it modulo a prime number.

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

Table 1. Binary Encoding of DNA alphabet

String x	A G C A T
Binary form	0 0 1 0 0 1 0 0 1 1
Signature $\sigma(x)$	1 4 7

Table 2. Signature of $AGCAT$

Our aim is to preprocess text t and create two sets of lists $\Lambda_{\ell_{min}}, \ldots, \Lambda_{\ell_{max}}$ and $\Lambda'_{\ell_{min}}, \ldots, \Lambda'_{\ell_{max}}$. Each list Λ_ℓ, for all $\ell_{min} \leq \ell \leq \ell_{max}$, holds each duplicate substring of length ℓ of t. Each list Λ'_ℓ, for all $\ell_{min} \leq \ell \leq \ell_{max}$, holds each unique substring of length ℓ of t.

An outline of the parallel algorithm, for all $\ell_{min} \leq \ell \leq \ell_{max}$, is as follows:

Problem Partitioning. We use a *data decomposition* approach to partition the text t with the sliding window mechanism into a set of substrings $z_1, z_2, \ldots, z_{n-\ell+1}$, where $z_i = t[i \ldots i+\ell-1]$, for all $1 \leq i \leq n-\ell+1$.

Step 1. We assume that text t is stored locally on the master processor. We make sure that the load is evenly balanced by distributing $z_1, z_2, \ldots, z_{n-\ell+1}$ among the p available processors. Each processor ρ_q, for all $0 \leq q < p$, is allocated a fair amount a_q of substrings, as shown in Equation 1.

$$a_q = \begin{cases} \lceil \frac{n-\ell+1}{p} \rceil, & \text{if } q < n-\ell+1 \mod p \\ \lfloor \frac{n-\ell+1}{p} \rfloor, & otherwise \end{cases} \tag{1}$$

We denote $z_{first_q}, \ldots, z_{last_q}$ as the set of the allocated substrings of length ℓ of processor ρ_q.

Example. Table 3 shows the processors allocation for the case of $t = GGGTCTA$, $\ell = 3$ and $p = 3$.

Step 2. Each processor ρ_q compacts each allocated substring z_i, for all $first_q \leq i \leq last_q$, into a signature $\sigma(z_i)$, packs it in a couple $(i, \sigma(z_i))$, where i represents the matching position of z_i in t, and adds the couple to a local list Z_q. Notice that, as

ρ_q	a_q	$first_q$	$last_q$	Allocated substrings
ρ_0	2	1	2	$z_1 = GGG$, $z_2 = GGT$
ρ_1	2	3	4	$z_3 = GTC$, $z_4 = TCT$
ρ_2	1	5	5	$z_5 = CTA$

Table 3. Processors allocation for $t = GGGTCTA$, $\ell = 3$ and $p = 3$

soon as we compact z_{first_q} into $\sigma(z_{first_q})$, then each $\sigma(z_i)$, for all $first_q + 1 \leq i \leq last_q$, can be retrieved in constant time (using "shift"-type of operation).

Step 3. We sort the local lists Z_q based on the signature's field, in parallel, using *Parallel Sorting by Regular Sampling* (PSRS) [20], a practical parallel deterministic sorting algorithm. Notice that parallel sorting means rearranging the elements of the local lists Z_q, so that each processor ρ_q still has a fair amount in Z_q, but with the smallest signatures stored in sorted order by processor ρ_0, ρ_1 etc.

Step 4. Each processor ρ_q runs sequentially through its sorted list Z_q and checks whether the signatures in $Z_q[x]$ and $Z_q[x+1]$ are equal, for all $0 \leq x < |A_q| - 1$. If they are equal, then ρ_q adds $Z_q[x]$ to a new list L_q. If not, then $Z_q[x]$ is added to a new list L'_q.

Step 5. Each processor ρ_q, for all $1 \leq q < p$, sends the first element in Z_q to the neighbour processor ρ_{q-1}. Then, each processor ρ_q, for all $0 \leq q < p-1$, compares the signature of the last element in Z_q, to the signature of the element received from processor ρ_{q+1}. If they are equal, then processor ρ_q adds the element to the list L_q, else it is added to the list L'_q.

Step 6. We perform a *gather* operation, in which processor ρ_0 collects a unique message, local list L_q, from each processor ρ_q, for all $1 \leq q < p$, and stores each local list L_q in rank order, resulting in a new combined sorted list Λ_ℓ. We do the same with the local list L'_q, resulting in a new sorted combined list Λ'_ℓ. Processor ρ_0 performs a *one-to-all* broadcast to send both lists Λ_ℓ and Λ'_ℓ to all other processors.

Main Step. Assume that the two sets of lists $\Lambda_{\ell_{min}}, \ldots, \Lambda_{\ell_{max}}$ and $\Lambda'_{\ell_{min}}, \ldots, \Lambda'_{\ell_{max}}$ are created and stored on each processor ρ_q. We extend the set of patterns $p_0, p_1, \ldots, p_{r-1}$ to a new set $p'_0, p'_1, \ldots, p'_{r'-1}$, $r < r'$, as follows.

We make sure that each processor ρ_q is allocated a fair amount of query patterns from the set $p_0, p_1, \ldots, p_{r-1}$, in a similar way as in step 1.

1. **Problem 1.** For each degenerate pattern p_i of length ℓ with λ non-solid symbols, such that $\lambda \leq \mu$, we create $\prod_{j=1}^{\ell} |p_i[j]|$ new patterns, each differing in λ positions.
2. **Problem 2.** For each weighted pattern p_i of length ℓ with λ non-solid symbols, such that $\lambda \leq \mu$, we create $\prod_{j=1}^{\ell} |p_i[j]|$ new patterns, each differing in λ positions. We select each of those patterns, say $s = s[1 \ldots \ell]$, with $s[1] = (q_1, \pi_1(q_1))$, $s[2] = (q_2, \pi_2(q_2)), \ldots, s[\ell] = (q_\ell, \pi_\ell(q_\ell))$, that satisfy $\prod_{j=1}^{\ell} \pi_j(q_j) \geq c$.

Then, each processor can determine, by using a binary search, whether an allocated pattern p'_i of length ℓ occurs in t exactly once, in $\mathcal{O}(\log n)$ time. If $\sigma(p'_i) \in \Lambda'_\ell$, then p'_i is a unique pattern, and the algorithm returns its matching position in t. If $\sigma(p'_i) \in \Lambda_\ell$, then p'_i occurs in t more than once. If $\sigma(p'_i) \notin \Lambda_\ell$ and $\sigma(p'_i) \notin \Lambda'_\ell$, then p'_i does not occur in t.

Notice that in a case when $2\ell > w$, where w is the word size of the machine (e.g. 32 or 64 in practice), our algorithm can easily be adopted by storing the signatures in $\lceil 2\ell/w \rceil$ computer words.

Theorem 1. *Given the text $t = t[1 \dots n]$, the set of patterns $p_0, p_1, \dots, p_{r-1}$, the length of each pattern $\ell_{min} \leq \ell \leq \ell_{max}$, the word size of the machine w, and the number of processors p, the parallel algorithm solves the Class 1 of Problem 1 and Problem 2 in $\mathcal{O}(\lceil \ell_{max}/w \rceil (\frac{n}{p} \log \frac{n}{p} + \frac{r}{p} \ell_{max} \log n))$ computation time and $\mathcal{O}(n \log p + r)$ communication time.*

Proof. In step 1, assuming that the text t is kept locally on master processor, the data distribution can be done in $\mathcal{O}(t_s \log p + t_w \frac{n}{p}(p-1))$ communication time. In step 2, each processor creates a fair amount of signatures in $\mathcal{O}(\lceil \ell_{max}/w \rceil \frac{n}{p})$ computation time. In step 3, the *PSRS* algorithm can be executed in $\mathcal{O}(\lceil \ell_{max}/w \rceil \frac{n}{p} \log \frac{n}{p})$ computation time, where $n \geq p^3$, and $\mathcal{O}(n/\sqrt{p})$ communication time [20]. In step 4, the sequential run through the local list Z_q takes $\mathcal{O}(\lceil \ell_{max}/w \rceil \frac{n}{p})$ computation time. Step 5 involves $\mathcal{O}(1)$ point-to-point simple message transfers and comparisons. In step 6, the *gather* operation can be done in $\mathcal{O}(t_s \log p + t_w \frac{n}{p}(p-1))$, and the *one-to-all* broadcast take $\mathcal{O}((t_s + t_w n) \log p)$ communication time.

Assuming that the two sets (of a constant number) of lists are created, the main step runs in $\mathcal{O}(\lceil \ell_{max}/w \rceil \frac{r}{p} \ell_{max} \log n)$ computation time, for the binary search, and $\mathcal{O}(t_s \log p + t_w \frac{r}{p}(p-1))$ communication time, for the patterns distribution. Notice that, since $|\Sigma| = 4$ and $\mu = 3$, the number of the new created patterns is treated as constant.

Hence, asymptotically, the overall time is $\mathcal{O}(\lceil \ell_{max}/w \rceil (\frac{n}{p} \log \frac{n}{p} + \frac{r}{p} \ell_{max} \log n))$ computation time, and $\mathcal{O}(n \log p + r)$ communication time. □

5 Massive Approximate Unique Pattern Matching in Parallel

In this section we solve the problem of *Class 2* and *Class 3*. The focus is to find occurrences of p_i, for all $0 \leq i < r$, in text $t = t[1 \dots n]$ with at most k-mismatches. In particular, we are interested in whether p_i occurs with at most 1-mismatch in t exactly once for the problem of *Class 2*, or with at most 2-mismatches exactly once for the problem of *Class 3*.

The proposed parallel algorithm makes use of the message-passing paradigm, by using p processing elements, and word-level parallelism, by compacting strings into signatures, and applying a bit-vector algorithm for efficient approximate string matching with mismatches.

Our aim is to preprocess text t and create two sets of lists $\Lambda_{\ell_{min}}, \dots, \Lambda_{\ell_{max}}$ and $\Lambda'_{\ell_{min}}, \dots, \Lambda'_{\ell_{max}}$. Each list Λ_ℓ, for all $\ell_{min} \leq \ell \leq \ell_{max}$, holds each duplicate substring of length ℓ of t with at most k-mismatches. Each list Λ'_ℓ, for all $\ell_{min} \leq \ell \leq \ell_{max}$, holds each unique substring of length ℓ of t.

5.1 The bit-vector algorithm for fixed-length approximate string matching with k-mismatches

Iliopoulos, Mouchard and Pinzon in [10] presented the Max-Shift algorithm, a bit-vector algorithm that solves the *fixed-length approximate string matching* problem:

given a text t of length n, a pattern ρ of length m and an integer ℓ, compute the optimal alignment of all substrings of ρ of length ℓ and a substring of t. The focus of the MAX-SHIFT algorithm is on computing matrix D', which contains the best scores of the alignments of all substrings of pattern ρ of length ℓ and any contiguous substring of the text t.

The MAX-SHIFT algorithm makes use of word-level parallelism in order to compute matrix D' efficiently, similar to the manner used by Myers in [15]. The algorithm is based on the $\mathcal{O}(1)$ time computation of each $D'[i,j]$ by using bit-vector operations, under the assumption that $\ell \leq w$, where w is the number of bits in a machine word or $\mathcal{O}(\ell/w)$-time for the general case. The algorithm maintains a bit-vector matrix $B[0\ldots m, 0\ldots n]$, where the bit integer $B[i,j]$, holds the binary encoding of the path in D' to obtain the optimal alignment at $i,\ j$ with the differences occurring as leftmost as possible.

Here the key idea is to devise a bit-vector algorithm for the *fixed-length approximate string matching with at most k-mismatches* problem: given a text t of length n, a pattern ρ of length m and an integer ℓ, find all substrings of ρ of length ℓ that match any contiguous substring of t of length ℓ with at most k-mismatches. If we assign ρ=t, we can extract all the duplicate substrings of length ℓ of t with at most k-mismatches. The focus is on computing matrix M, which contains the number of mismatches of all substrings of pattern ρ of length ℓ and any contiguous substring of the text t of length ℓ.

Example. Let the text $t = \rho = GGGTCTA$ and $\ell = 3$. Table 4 shows the matrix M.

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

Table 4. Matrix M for $t = \rho = GGGTCTA$ and $\ell = 3$

We maintain the bit-vector $B[i,j] = b_\ell \cdots b_1$, where $b_\lambda = 1,\ 1 \leq \lambda \leq \ell$, if there is a mismatch of a contiguous substring of the text $t[i-\ell+1\ldots i]$ and $t[j-\ell+1\ldots j]$ in the λ^{th} position. Otherwise we set $b_\lambda = 0$.

Given the restraint that the integer ℓ is less than the length of the computer word w, then the bit-vector operations allow to update each entry of the matrix B in constant time (using "shift"-type of operation on the bit-vector). The maintenance of the bit-vector is done via operations defined as follows:

1. $shiftc(x)$: shifts and truncates the leftmost bit of x.
2. $\delta_H(x,y)$: returns the minimum number of replacements required to transform x into y

The Bit-Vector-Mismatches algorithm for computing the bit-vector matrix B and matrix M is outlined in Figure 2.

```
Bit-Vector-Mismatches
▷Input: t, n, ρ, m, ℓ
▷Output: B, M
1  begin
2    ▷ Initialization
3    B[0...m, 0] ← min(i, ℓ) 1's; B[0, 0...n] ← 0
4    M[0...m, 0] ← min(i, ℓ); M[0, 0...n] ← 0
5    ▷ Matrix B and Matrix M computation
6    for i ← 1 until m do
7      for j ← 1 until n do
8          B[i, j] ← shiftc(B[i − 1, j − 1]) OR δ_H(ρ[i], t[j])
9          M[i, j] ← ones(B[i, j])
10 end
```

Figure 2. The Bit-Vector-Mismatches algorithm for computing matrix B and matrix M

Example. Let the text $t = \rho = GGGTCTA$ and $\ell = 3$. Table 5 shows the bit-vector matrix B. Consider the case when $i = 7$ and $j = 5$. Cell $B[7, 5] = 101$ denotes that substrings $t[3 \ldots 5] = CTA$ and $t[5 \ldots 7] = GTC$ have a mismatch in position 1, a match in position 2, and a mismatch in position 3, resulting in a total of two mismatches, as shown in cell $M[7, 5]$.

		0	1	2	3	4	5	6	7
		ϵ	G	G	G	T	C	T	A
0	ϵ	0	0	0	0	0	0	0	0
1	G	1	0	0	0	1	1	1	1
2	G	11	10	00	00	01	11	11	11
3	G	111	110	100	000	001	011	111	111
4	T	111	111	101	001	000	011	110	111
5	C	111	111	111	011	011	000	111	101
6	T	111	111	111	111	110	111	000	111
7	A	111	111	111	111	111	101	111	000

Table 5. The bit-vector matrix B for $t = \rho = GGGTCTA$ and $\ell = 3$

Assume that the bit-vector matrix $B[0 \ldots m, 0 \ldots n]$ is given. We can use the function $ones(v)$, which returns the number of 1's (bits set on) in the bit-vector v, to compute matrix M (see Figure 2, line 9).

5.2 The parallel algorithm

The key idea behind parallelising the Bit-Vector-Mismatches algorithm, is that cell $B[i,j]$ can be computed only in terms of $B[i-1,j-1]$ (see Figure 2, line 8).

An outline of the parallel algorithm, for all $\ell_{min} \leq \ell \leq \ell_{max}$, is as follows:

Problem Partitioning. We use a *functional decomposition* approach, in which the initial focus is on the computation that is to be performed rather than on the data manipulated by the computation. We assume that the text t (and the pattern $\rho = t$) is stored locally on each processor. This can be done by using a *one-to-all* broadcast operation in $(t_s + t_w n)\log p$ communication time, which is asymptotically $\mathcal{O}(n \log p)$. We partition the problem of computing matrix B (and M) into a set of diagonal vectors $\Delta_0, \Delta_1, \ldots, \Delta_{n+m}$, as shown in Equation 2.

$$\Delta_\nu[x] = \begin{cases} B[\nu - x,\ x] \ :\ 0 \leq x \leq \nu, & \text{(a)} \\ B[m - x,\ \nu - m + x] \ :\ 0 \leq x < m + 1, & \text{(b)} \\ B[m - x,\ \nu - m + x] \ :\ 0 \leq x < n + m - \nu + 1, & \text{(c)} \end{cases} \tag{2}$$

where,
(a) if $0 \leq \nu < m$
(b) if $m \leq \nu < n$
(c) if $n \leq \nu < n + m + 1$

Step 1. We make sure that the load is evenly balanced among the p available processors in each diagonal Δ_ν. Each processor ρ_q, for all $0 \leq q < p$, is allocated a fair amount $a_q[\nu]$ of cells in each diagonal Δ_ν, as shown in Equation 3.

$$a_q[\nu] = \begin{cases} \lceil \frac{|\Delta_\nu|}{p} \rceil, & \text{if } q < |\Delta_\nu| \mod p \\ \lfloor \frac{|\Delta_\nu|}{p} \rfloor, & otherwise \end{cases} \tag{3}$$

We denote $\Delta_\nu[first_q[\nu]], \ldots, \Delta_\nu[last_q[\nu]]$ as the set of the allocated cells of processor ρ_q in diagonal Δ_ν.

Step 2. Each processor ρ_q computes each allocated cell $\Delta_\nu[x]$, for all $first_q[\nu] \leq x \leq last_q[\nu]$, in each diagonal Δ_ν, using the Bit-Vector-Mismatches algorithm.

Step 3. It is possible that in a certain diagonal Δ_ν, $\nu > 0$, a processor will need a cell or a pair of cells, which were not computed on its local memory in diagonal $\Delta_{\nu-1}$. We need a communication pattern in each diagonal Δ_ν, for all $0 \leq \nu < n + m$, which minimises the data exchange between the processors. It is obvious, that in each diagonal, each processor needs only to communicate with its neighbours (boundary cells swaps).

Step 4. On every occasion a processor ρ_q computes a cell $M[i,j] \leq k$, where $i \geq \ell$ and $j \geq \ell$, we notice two possible cases:

1. if $M[i,j] = 0$ and $i = j$, then substring $t[i - \ell + 1 \ldots i]$ occurs in t at least once. We compact substring $t[i - \ell + 1 \ldots i]$ into a signature $\sigma(t[i - \ell + 1 \ldots i])$, pack it in a couple $(i - \ell + 1, \sigma(t[i - \ell + 1 \ldots i]))$, and add the couple to a new list Z_q.

2. if $i \neq j$, then substrings $t[i-\ell+1 \dots i]$ and $t[j-\ell+1 \dots j]$ are considered to be duplicates with at most k-mismatches. We compact both substrings into the signatures $\sigma(t[i-\ell+1 \dots i])$ and $\sigma(t[j-\ell+1 \dots j])$, pack them in couples $(i-\ell+1, \sigma(t[i-\ell+1 \dots i]))$ and $(j-\ell+1, \sigma(t[j-\ell+1 \dots j]))$, and add the couples to the list Z_q.

Step 5. Assume that the diagonal supersteps $\Delta_0, \Delta_1, \dots, \Delta_{n+m}$ are executed. The local lists Z_q are constructed, and so, we follow the steps 3-6 of the parallel algorithm in Section 4.

Main step. Assume that the two sets of lists $\Lambda_{\ell_{min}}, \dots, \Lambda_{\ell_{max}}$ and $\Lambda'_{\ell_{min}}, \dots, \Lambda'_{\ell_{max}}$ are created and stored on each processor ρ_q. We extend the set of patterns $p_0, p_1, \dots, p_{r-1}$ to a new set $p'_0, p'_1, \dots, p'_{r'-1}$, $r < r'$, as in Section 4. Then, each processor can determine by using a binary search, whether an allocated pattern p'_i of length ℓ occurs in t exactly once, in $\mathcal{O}(\log n)$ time. If $\sigma(p'_i) \in \Lambda'$, then p'_i is a unique pattern with at most k-mismatches, and the algorithm returns its matching position in t. If $\sigma(p'_i) \in \Lambda_\ell$, then p'_i occurs in t more than once.

Notice that, in a case where $\sigma(p'_i) \notin \Lambda_\ell$ and $\sigma(p'_i) \notin \Lambda'_\ell$, then p'_i does not occur in t, and we can check whether the k-mismatches occur inside the pattern p'_i as follows.

1. **Class 2 and Class 3.** We construct a new set of patterns x_j, for all $0 \leq j < |\Sigma|.\ell$, differing from p'_i in one position, and we compact each x_j into a signature $\sigma(x_j)$. If $\sigma(x_j) \in \Lambda'_\ell$, then p'_i is a unique pattern with at most 1-mismatch, and the algorithm returns its matching position in t. If $\sigma(x_j) \in \Lambda_\ell$, then we discard pattern p'_i as it has to occur in t exactly once. If $\sigma(x_j) \notin \Lambda_\ell$ and $\sigma(x_j) \notin \Lambda'_\ell$ then p'_i does not occur in t.
2. **Class 3.** We construct a new set of patterns y_j, for all $0 \leq j < |\Sigma|^2.\binom{\ell}{2}$, differing from p'_i in two positions, and we compact each y_j into a signature $\sigma(y_j)$. If $\sigma(y_j) \in \Lambda'_\ell$, then p'_i is a unique pattern with at most 2-mismatches, and the algorithm returns its matching position in t. If $\sigma(y_j) \in \Lambda_\ell$, then we discard pattern p'_i as it has to occur in t exactly once. If $\sigma(y_j) \notin \Lambda_\ell$ and $\sigma(y_j) \notin \Lambda'_\ell$ then p'_i does not occur in t.

In general, for the problem of k-mismatches, for each pattern p'_i of length ℓ that does not occur in t, we construct k new sets of patterns, each containing $|\Sigma|^\lambda.\binom{\ell}{\lambda}$ patterns differing from p'_i in λ positions, for all $1 \leq \lambda \leq k$.

Theorem 2. *Given the text $t = t[1 \dots n]$, the set of patterns $p_0, p_1, \dots, p_{r-1}$, the length of each pattern $\ell_{min} \leq \ell \leq \ell_{max}$, the word size of the machine w, and the number of processors p, the parallel algorithm solves the Class 2 and Class 3 of Problem 1 and Problem 2 in $\mathcal{O}(\lceil \ell_{max}/w \rceil (\frac{n^2}{p} + \frac{\ell^3_{max} r}{p} \log p))$ computation time, and $\mathcal{O}(n \log p + r)$ communication time.*

Proof. We partition the problem of computing matrix B (and M) into a set of $n+m+1$ diagonal vectors, thus $\mathcal{O}(n)$ supersteps (since $n = m$). In step 1, the allocation can be done in $\mathcal{O}(1)$ time. In step 2, the cells computation requires $\mathcal{O}(\lceil \ell_{max}/w \rceil \frac{n}{p})$ time. In step 3, the data exchange between the processors involves $\mathcal{O}(1)$ point-to-point simple message transfers. In step 4, the local lists Z_q are constructed in $\mathcal{O}(\lceil \ell_{max}/w \rceil \frac{n}{p})$ time. Assuming that the diagonal supersteps are executed, step 5 can be done in $\mathcal{O}(\lceil \ell_{max}/w \rceil (\frac{n}{p} \log \frac{n}{p}))$ computation time, and $\mathcal{O}(n \log p)$ communication time (see Section 4).

Assuming that the two sets (of a constant number) of lists are created, the main step runs in $\mathcal{O}(\lceil \ell_{max}/w \rceil (\frac{\ell_{max}^3 r}{p} \log n))$ computation time, for the binary search, and $\mathcal{O}(t_s \log p + t_w \frac{r}{p}(p-1))$ communication time, for the patterns distribution.

Hence, asymptotically, the overall time is $\mathcal{O}(\lceil \ell_{max}/w \rceil (\frac{n^2}{p} + \frac{\ell_{max}^3 r}{p} \log p))$ computation time, and $\mathcal{O}(n \log p + r)$ communication time. □

Also, the space complexity can be reduced to $\mathcal{O}(n)$ by noting that each diagonal Δ_ν depends only on diagonal $\Delta_{\nu-1}$.

6 Conclusion

In this paper, we have presented parallel algorithms to tackle the data emerging from the new high throughput sequencing technologies in biology. The new technologies produce a huge number of very short sequences and these sequences need to be classified, tagged and recognised as parts of a reference genome. Our algorithms can manipulate this data for degenerate and weighted sequences for Massive Exact and Approximate Unique Pattern matching. Implementation of the algorithms described in this paper is under way and will be presented in the near future.

References

1. R. Baeza-Yates and G. Gonnet: *A new approach to text searching.* Communications of the ACM, 35 1992, pp. 74–82.
2. S. Bennett: *Solexa ltd.* Pharmacogenomics, 5(4) 2004, pp. 433–438.
3. M. Christodoulakis, C. S. Iliopoulos, L. Mouchard, K. Perdikuri, A. Tsakalidis, and K. Tsichlas: *Computation of repetitions and regularities on biological weighted sequences.* Journal of Computational Biology, 13(6) 2006, pp. 1214–1231.
4. B. Dömölki: *An algorithm for syntactic analysis.* Computational Linguistics, 8 1964, pp. 29–46.
5. B. Ewing, L. Hillier, M. C. Wendl, and P. Green: *Base-Calling of Automated Sequencer Traces UsingPhred.I. Accuracy Assessment.* Genome Research, 8(3) 1998, pp. 175–185.
6. R. D. Fleischmann, M. D. Adams, O. White, R. A. Clayton, E. F. Kirkness, A. R. Kerlavage, C. J. Bult, J. F. Tomb, B. A. Dougherty, and J. M. Merrick: *Whole-genome random sequencing and assembly of Haemophilus influenzae.* Science, 269 1995, pp. 496–512.
7. C. Genomics: *The sequence of the human genome.* Science, 291 2001, pp. 1304–1351.
8. D. Hernandez, P. Francois, L. Farinelli, M. Osteras, and J. Schrenzel: *De novo bacterial genome sequencing: millions of very short reads assembled on a desktop computer.* Genome Res, March 2008.
9. J. Holub, W. F. Smyth, and S. Wang: *Fast pattern-matching on indeterminate strings.* J. of Discrete Algorithms, 6(1) 2008, pp. 37–50.
10. C. S. Iliopoulos, L. Mouchard, and Y. Pinzon: *The max-shift algorithm for approximate string matching*, in Proceedings of the 5th Workshop on Algorithm Engineering (WAE'01), Aarhus, Denmark, 2001, G.S. Brodal and D. Frigioni and A.Marchetti-Spaccamela, eds., pp. 13–25.
11. H. Jiang and W. H. Wong: *Seqmap: mapping massive amount of oligonucleotides to the genome.* Bioinformatics, 24(20) 2008, pp. 2395–2396.
12. R. M. Karp and M. O. Rabin: *Efficient randomized pattern-matching algorithms.* IBM J. Res. Dev., 31(2) 1987, pp. 249–260.
13. H. Li, J. Ruan, and R. Durbin: *Mapping short DNA sequencing reads and calling variants using mapping quality scores.* Genome Research, 18(11) 2008, pp. 1851–1858.
14. R. Li, Y. Li, K. Kristiansen, and J. Wang: *SOAP: short oligonucleotide alignment program.* Bioinformatics, 24(5) 2008, pp. 713–714.

15. E. W. Myers: *A fast bit-vector algorithm for approximate string matching based on dynamic progamming.* Journal of the ACM 46, 1999, pp. 395–415.
16. J. M. Prober, G. L. Trainor, R. J. Dam, F. W. Hobbs, C. W. Robertson, R. J. Zagursky, A. J. Cocuzza, M. A. Jensen, and K. Baumeister: *A system for rapid DNA sequencing with fluorescent chain-terminating dideoxynucleotides.* Science, 238 1987, pp. 336–341.
17. F. Sanger and A. R. Coulson: *A rapid method for determining sequences in DNA by primed synthesis with DNA polymerase.* J. Mol. Biol., 94 1975, pp. 441–448.
18. F. Sanger, S. Nicklen, and A. R. Coulson: *DNA sequencing with chain-terminating inhibitors.* Proc. Natl. Acad. Sci. USA, 74 1977, pp. 5463–5467.
19. M. C. Shaner, I. M. Blair, and T. D. Schneider: *Sequence logos: A powerful, yet simple, tool*, in Proceedings of the Twenty-Sixth Annual Hawaii International Conference on System Sciences, Volume 1: Architecture and Biotechnology Computing, T. N. Mudge, V. Milutinovic, and L. Hunter, eds., IEEE Computer Society Press, 1993, pp. 813–821.
20. H. Shi and J. Schaeffer: *Parallel sorting by regular sampling.* J. Parallel Distrib. Comput., 14(4) 1992, pp. 361–372.
21. A. Smith, Z. Xuan, and M. Zhang: *Using quality scores and longer reads improves accuracy of solexa read mapping.* BMC Bioinformatics, 9(1) 2008, p. 128.
22. L. M. Smith, J. Z. Sanders, R. J. Kaiser, P. Hughes, C. Dodd, C. R. Connell, C. Heiner, S. B. Kent, and L. E. Hood: *Fluorescence detection in automated DNA sequence analysis.* Nature, 321 1986, pp. 674–679.
23. M. Sultan, M. H. Schulz, H. Richard, A. Magen, A. Klingenhoff, M. Scherf, M. Seifert, T. Borodina, A. Soldatov, D. Parkhomchuk, D. Schmidt, S. O'Keeffe, S. Haas, M. Vingron, H. Lehrach, and M.-L. Yaspo: *A Global View of Gene Activity and Alternative Splicing by Deep Sequencing of the Human Transcriptome.* Science, 321(5891) 2008, pp. 956–960.
24. A. Sundquist, M. Ronaghi, H. Tang, P. Pevzner, and S. Batzoglou: *Whole-genome sequencing and assembly with high-throughput, short-read technologies.* PLoS ONE, 2 2007.
25. Z. Wang, M. Gerstein, and M. Snyder: *Rna-seq: a revolutionary tool for transcriptomics.* Nature reviews. Genetics, November 2008.
26. R. L. Warren, G. G. Sutton, S. J. Jones, and R. A. Holt: *Assembling millions of short DNA sequences using SSAKE.* Bioinformatics, 23(4) February 2007, pp. 500–501.
27. S. Wu and U. Manber: *Agrep- a fast approximate pattern-matching tool.* Proceedings USENIX Winter 1992 Technical Conference, San Francisco, CA, 1992, pp. 153–162.
28. S. Wu and U. Manber: *Fast text searching: allowing errors.* Commun. ACM, 35(10) 1992, pp. 83–91.