Bioinformatics
doi.10.1093/bioinformatics/xxxxxx
Advance Access Publication Date: Day Month Year
Original paper

OXFORD

Sequence analysis

Acceleration of Nucleotide Semi-Global Alignment with Adaptive Banded Dynamic Programming

Hajime Suzuki [1] **and Masahiro Kasahara** [1,*]

[1] Department of Computational Biology and Medical Sciences, Graduate School of Frontier Sciences, the University of Tokyo, Kashiwa City, Chiba, 277-8561, Japan.

* To whom correspondence should be addressed.

Associate Editor: XXXXXXX

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Abstract

Motivation: Pairwise alignment of nucleotide sequences has been calculated in practice by the seed-and-extend strategy, where we enumerate seeds (shared patterns) between sequences and then extend the seeds by a Smith-Waterman-like semi-global dynamic programming to obtain full pairwise alignments. With the advent of massively parallel short read sequencers, algorithms and data structures for efficiently finding seeds had been explored extensively. However, recent advances in single-molecule sequencing technologies enabled us to obtain millions of reads, each of which is orders of magnitude longer than those output by the short-read sequencers, demanding a faster algorithm for the extension step that dominates the computation time in pairwise local alignment. Our goal is to design a faster extension algorithm which overcomes the two major drawbacks of the single-molecule sequencers that the sequencing error rates is high (e.g., 10-15 %) and insertions and deletions are more frequent than substitutions are.

Results: We propose an adaptive banded dynamic programming (DP) algorithm for calculating pairwise semi-global alignment of nucleotide sequences that allows a relatively high insertion or deletion rate while maintaining the band width to some small constant (e.g., 32 cells). On every band advancing operation, cells at the forefront of the band are calculated simultaneously without mutual dependencies, allowing an efficient Single-Instruction-Multiple-Data (SIMD) parallelization. We show by an experiment that our algorithm runs approximately 8 times faster than the extension alignment algorithm in NCBI BLAST+ retaining the similar sensitivity and accuracy. The results indicate that the algorithm is capable of replacing extension alignment routines in the existing nucleotide local alignment programs.

Availability: The implementation of the algorithm and the benchmarking scripts are available at https://github.com/ocxtal/adaptivebandbench.

Contact: mkasa@k.u-tokyo.ac.jp

1 Introduction

In the past decade, technological improvement in the DNA sequencing field has been remarkable. Single-molecule sequencers, often called third generation sequencers, achieved more than ten-fold improvement in their read lengths. The commercially available third generation sequencers such as PacBio Sequel and Oxford Nanopore MinION can yield reads of 20 kb or even longer, whereas the longest practical read lengths of the Sanger sequencers were around 1 kbp. They demand us to process huge amount of reads longer than 20 kb, but fast algorithms for such long reads were not explored well before the third generation sequencers became common.

As most genome analyses using massively parallel sequencers start with aligning reads against themselves (for *de novo* assembly in whole genome re-sequencing researches) or reference sequences (for other reference-guided analyses, e.g. exome sequencing and RNA-seq), faster algorithms for pairwise alignment are crucial in accelerating most types of genome analyses. However, it has been recently proved that the near-quadratic time bounds for computing the edit distance cannot not be improved and would be tight (unless strong exponential time hypothesis is false; Backurs and Indyk (2015)). Thus, it is reasonable to design

fast heuristic algorithms for pairwise alignment. Current typical heuristic algorithms for pairwise local alignment first find short exact matches called "seeds" and then extend them using a semi-global variant of pairwise local alignment algorithms such as Smith-Waterman-Gotoh algorithm (SWG; Smith and Waterman (1981); Gotoh (1982)). This idea, the seed-and-extend strategy, was employed in classical Basic Local Alignment Search Tool (BLAST; Altschul *et al.* (1990)) and also recently in long-read alignment programs for third-generation sequencers, such as BWA-MEM (Li, 2013), BLASR (Chaisson and Tesler, 2012), DALIGNER (Myers, 2014), and GraphMap (Sović *et al.*, 2016).

In the era of second generation sequencers, such as Illumina and Ion Torrent, fast algorithms for finding seed matches were explored because the most time consuming step was in the seed matching stage. Approximate string matching algorithms based on Baeza-Yates algorithm (Navarro and Ricardo, 1998) were used in the early days (MAQ (Li *et al.*, 2008) and BFAST (Homer *et al.*, 2009)), and later an exact substring matching algorithm based on Burrows-Wheeler transform (Burrows and Wheeler, 1994) and an auxiliary data structure proposed by Ferragina and Manzini (Ferragina and Manzini, 2000) were adopted in many sequence alignment programs, such as BWA (Li and Durbin, 2009) and Bowtie2 (Langmead and Salzberg, 2012).

However, as the read length increases, the proportion of the extension step in the computation time of pairwise alignment has been getting larger and therefore a faster extension algorithm is demanded. Not only the longer read length leads to a longer computation time for the extension step, it also forces us to use a smaller seed length to allow an order of magnitude more frequent insertions and deletions in alignments, putting a significant computational burden on the extension step. Commonly used techniques for accelerating the extension alignment only calculate the values of the cells in a small region of the matrix in the SWG algorithm; we first estimate by heuristic a region where the optimal path of the pairwise alignment may go through, and only compute the values of the cells in the region. As the region gets smaller and smaller, we run a higher risk of missing the optimal path (alignment) but the computation time for the extension step also becomes smaller. This technique is first proposed by Chao *et al.* (1992), and is later adopted in many local alignment programs (BWA-MEM (Li, 2013) and BLASR (Chaisson and Tesler, 2012)) as "banded DP".

Another heuristic for the extension step was introduced in BLAST, which we denote by the "BLAST X-drop DP algorithm". The BLAST X-drop DP algorithm continues to extend alignments until all the cells in the forefront have a score less than the current maximum minus X by implicitly assuming that valid alignments do not include a part of alignments that scores to a value under $-X$. The algorithm successfully reduces the region in the DP matrix to be calculated when the X is sufficiently small.

Another approaches previously taken to accelarate the SWG algorithm by a constant factor include utilizing Single-Instruction-Multiple-Data (SIMD) operations that increases the number of cells processed per unit operation. Examples of such approaches include the Wozniak's (Wozniak, 1997), Rognes's (Rognes and Seeberg, 2000), and Farrar's (Farrar, 2007) approach. The Farrar's striped vectorization (parallelization) successfully accelerated calculation of a rectangular DP matrix of the SWG algorithm, and it was adopted in many local alignment programs (MOSAIK2 (Lee *et al.*, 2014), BWA (Li and Durbin, 2009) and Bowtie2 (Langmead and Salzberg, 2012)) and libraries (SSW library (Zhao *et al.*, 2013), Parasail (Daily, 2016)).

Here, we propose a new algorithm, adaptive banded DP. It uses SIMD instructions on general-purpose processors in a different way from previous ones, and supports a X-drop-like heuristic to accurately identify the end of alignments. We demonstrate that it is the fastest affine-gap penalty semi-global alignment algorithm. It calculates the semi-global alignment of nucleotide sequences with a 4×4 substitution matrix and an affine-gap penalty function 8 times faster than the X-drop DP algorithm implemented in the BLAST. The algorithm successfully reports the optimal alignment score and its corresponding alignment path at nearly 100 % of probablity when aligning sequences with the error mode of single-molecule sequencers with common substitution matrices and gap penalties. Notably, the indel tolerance is so high that a simulation with our algorithm (the cell width: 32) were able to successfully align reads with a 27 base contiguous gap without missing matches after the gap.

2 Methods

2.1 Semi-global alignment of nucleotide sequences

First we give a definition of the nucleotide semi-global alignment problem. Let $a = a_0 a_1 ... a_{|a|-1}$ and $b = b_0 b_1 ... b_{|b|-1}$ be strings over an alphabet $\Sigma = \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}$. The problem formulation is to calculate a coordinate (n, m) and a corresponding "alignment", or an edit path from $(0, 0)$ to (n, m) which consists of $\{\mathrm{match}, \mathrm{insertion}, \mathrm{deletion}\}$, that maximizes the sum of substitution scores and gap penalties. The substitution scores are defined over a pair of alphabets: $s(p, q)$ where $p, q \in \Sigma$ (called "score matrix"), and the gap penalty function is expressed in an integer linear form: $g(k) = G_o + k \cdot G_e$ where $G_o \geq 0, G_e > 0$ and k is the length of contiguous gaps (called "affine-gap penalty function"). The problem appears as a subproblem in the extension stage of the seed-and-extend algorithm.

This formulation of the semi-global alignment problem is usually solved with a variant of the Smith-Waterman-Gotoh (Smith and Waterman, 1981; Gotoh, 1982) algorithm, where the initial values (the scores at the top and left edges in the DP matrix) are modified to the gap penalties from the origin. This modification fixes the staring cell of resulting alignments to the origin of the matrix. The end of alignments remains open (not fixed) as in the original SWG algorithm; it starts traceback from the cell with the maximum score. We use a general 4×4 score matrix throughout the paper unless specified, but the match-mismatch scoring model, where a score matrix is characterized by a pair of integers, (M, X), where M is a match score and X is a penalty score, is a special case of the general score matrix, and therefore most discussions hereafter hold true also for the match-mismatch scoring model. The recurrence relations of the DP matrices used in the SWG algorithm are shown in Equation 1, 2, and 3, where S is a score matrix, E and F to calculate gap penalities in the horizontal and vertical directions respectively.

$$E[i,j] = \begin{cases} -G_o - i \cdot G_e & (j = 0) \\ -\inf & (i = 0) \\ \max \begin{cases} S[i-1,j] - G_o - G_e \\ E[i-1,j] - G_e \end{cases} & (i \neq 0, j \neq 0) \end{cases} \quad (1)$$

$$F[i,j] = \begin{cases} -G_o - j \cdot G_e & (i = 0) \\ -\inf & (j = 0) \\ \max \begin{cases} S[i,j-1] - G_o - G_e \\ F[i,j-1] - G_e \end{cases} & (i \neq 0, j \neq 0) \end{cases} \quad (2)$$

$$S[i,j] = \begin{cases} 0 & (i = 0, j = 0) \\ -G_o - j \cdot G_e & (i = 0, j \neq 0) \\ -G_o - i \cdot G_e & (i \neq 0, j = 0) \\ \max \begin{cases} S[i-1,j-1] + s(a_{i-1}, b_{j-1}) \\ E[i,j] \\ F[i,j] \end{cases} & (i \neq 0, j \neq 0) \end{cases} \quad (3)$$

We use the minimum or a sufficiently small value within the range of the integer type used in our implementation, as $-\inf$ cannot be represented

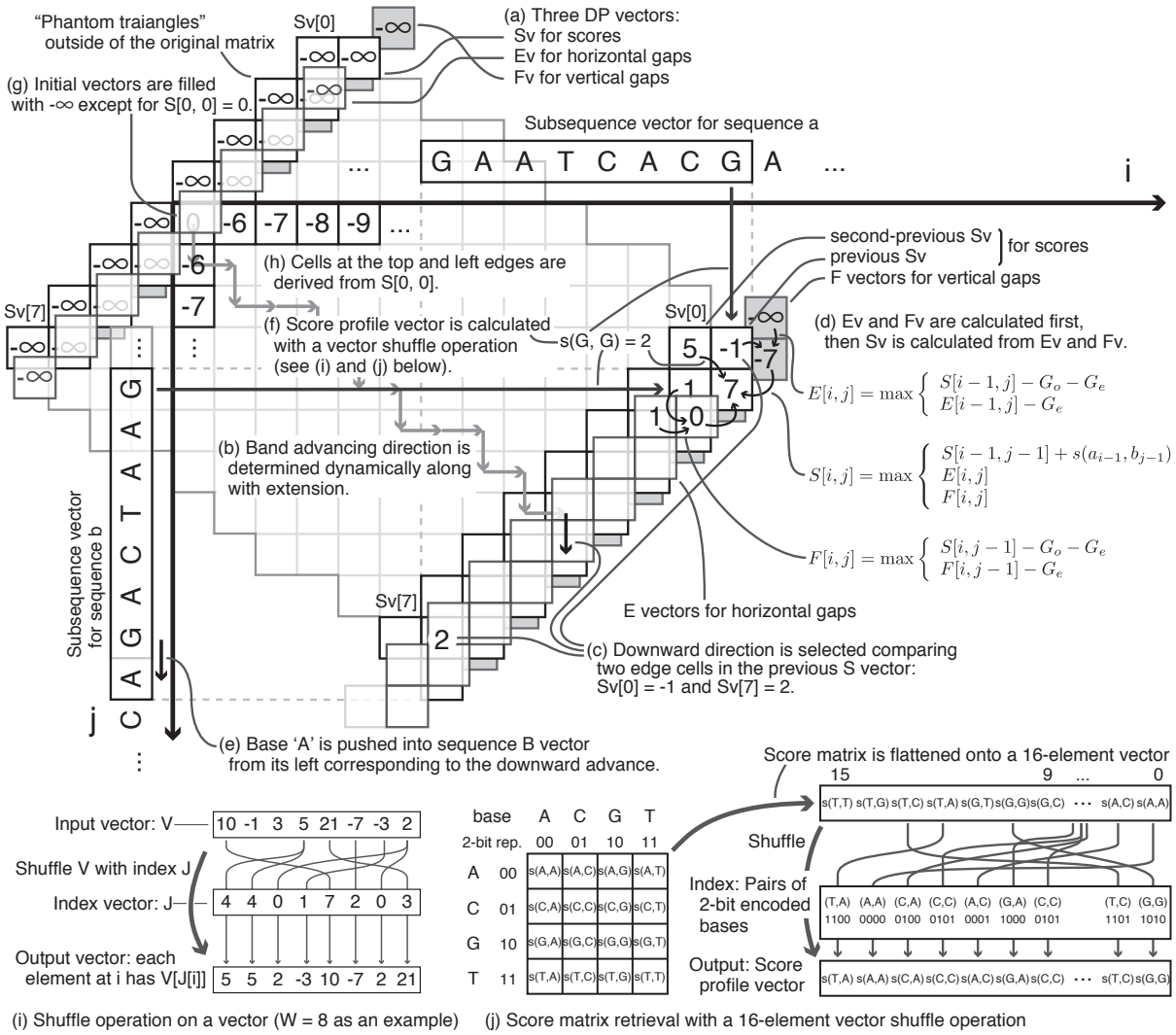


Fig. 1. Overview of the adaptive banded DP algorithm, where $W = 8$. (a) Vector placement: Three DP vectors, S_V, E_V, and F_V are placed in the anti-diagonal direction of the DP matrix, holding the corresponding part of the original S, E, and F matrices. (b, c) Determining the advancing direction of the band: The advancing direction of the band is determined on every vector update by comparing the two edge cells in the S_V vector. (d) Vector update procedure: The vector update procedure consists of three update operations, each of which corresponds to the update formula of the original semi-global alignment algorithm (Eq 1, 2, and 3). (e) Subsequence vectors: Two subsequence vectors are placed on the top and left sides of the matrix. They move according to the advancing direction of the band. (f, i, j) Score profile vector calculation: The score profile vector, an array of $s(\cdot, \cdot)$ values, is calculated using a vector shuffle instruction (i), which is an indexed element retrieval on a vector. The index vector and the input element vector are respectively composed of pairs of 2-bit encoded bases, and the flattened 4×4 substitution matrix (j). (g, h) Initialization of vectors: The three, S_V, E_V, and F_V vectors and the second-previous S_V vector are initialized with $-\inf$ except for $S[0, 0] = 0$. This setting results in proper initial values aligning on the two top and left lines.

in regular integer types. Note that the formulation of the SWG algorithm here might be somewhat different from ones that appear in previous papers, but it is mathematically equivalent to the corrected version of the original SWG algorithm (Flouri *et al.*, 2015).

2.2 Adaptive banded DP

In 1992, Chao *et.al.* proposed a method called banded DP as an acceleration technique for the global alignment DP algorithms (Chao *et al.*, 1992). It reduced the time and memory necessary in computation by narrowing the region of the DP matrix to calculate. The idea was later simplified to fill a pre-determined, diagonally placed rectangle band with a constant width, which we call "static banded DP" hereafter. The simplified algorithm is applied to many semi-global alignment implementations such as SeqAn (Döring *et al.*, 2008) and recently Parasail library (Daily, 2016), the refinement alignment in SSW library (Zhao *et al.*, 2013), and the extension alignment in BWA-MEM (Li, 2013) and BLASR (Chaisson and Tesler, 2012). A vector-oriented parallelization of the static banded DP was proposed by Kimura et.al. (Kimura *et al.*, 2012) in their static-banded edit-distance algorithm. They adopted anti-diagonally placed vectors with a constant width (e.g., 64 cells) to calculate the cells in a vector simultaneously.

Our adaptive banded DP algorithm adopts a similar band-narrowing approach, but in contrast to the static banded DP where the narrowed region is determined statically (i.e., before filling cells in the DP matrix), our algorithm determins the narrowed region dynamically as we calculate cells in the DP matrix in our algorithm. Figure 1 illustrates the overview of our algorithm. The banded region with a constant width (the number

of anti-diagonally aligned cells; denoted as W) is created by pushing the forefront DP vectors (the anti-diagonally placed vectors) toward the diagonal direction. The forefront DP vectors, S_V, E_V, and F_V hold the cells in an anti-diagonal line (Fig 1(a)). At each step, the three forefront DP vectors move to either of the two directions, rightward or downward. On every step, the advancing direction of the forefront vectors is determined by comparing the two edge cells ($S_V[0]$ and $S_V[W-1]$) to keep the difference between the two edge cells in the next forefront S_V vector smaller. In other words, the next vectors are placed rightward when $S_V[0] \geq S_V[W-1]$ and downward when $S_V[0] < S_V[W-1]$ (Fig 1(b, c)). When the forefront vectors move, the E_V, F_V, or S_V are updated according to the formula (Fig 1(d)). First, the new E_V and F_V vectors are derived from the previous E_V, F_V, and S_V vectors, then the new S_V vector is calculated from the current E_V and F_V vectors, the second-previous S_V vector, and a score profile vector (an array of substitution scores). As each operation has no dependencies between the cells in the forefront vectors, the whole update procedures can be implemented with Single-Instruction-Multiple-Data (SIMD) instructions keeping the vectors on SIMD registers.

In our algorithm, the score profile vector is also generated in a SIMD-vectorized manner leveraging a vector shuffle operation. The vector shuffle operation can retrieve multiple elements (e.g., 16 elements) from a given array of the fixed size (e.g., 16) in a single operation; it does a simple table-lookup from the 16-element array multiple times (e.g., 16 times) in parallel (Fig 1(i)). We concatenate the pair of 2-bit encoded bases into one value ranging from 0 to 15, and use the array of such values as the index vector and a flattened 4×4 score matrix as the input vector of the shuffle operation (Fig 1(f, j)). To efficiently generate the index vector, we retain two subsequences of length W on vector registers with each base represented in a 2-bit binary encoding, where $\{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}$ is mapped to $\{00, 01, 10, 11\}$ respectively. Every time the forefront vector moves, either of the two sequence vectors is shifted left or right by one according to the advancing direction (Fig 1(e)). The conversion from an ASCII code to the 2-bit encoded binary is performed when a base comes into the sequence vector; the conversion is done with a well-known simple formula $0\mathrm{x}03 \cdot ((c \gg 2) \oplus (c \gg 1))$ where c is a 8-bit ASCII-encoded character and $\cdot$, $\oplus$, and $\gg$ are respectively bitwise AND, bitwise XOR, and logical shift operations. To our knowledge, the technique is not published in literature, nor could we find the origin of the technique. Note that the conversion works correctly regardless of the case of an input character since the output 2-bit pattern only depends on a subset of the input bits that are identical between an uppercase character and its corresponding lowercase character.

The head of the band, the top-left triangular corner of the matrix, is handled in a special way. We added two phantom traiangular regions to reshape the corner to have a constant width. The initial vectors are placed at the top-left edge of the augumented band, whose center cell[1] is aligned to the cell at the origin $(0, 0)$. The initial values in the vectors are set to $-\inf$, or a sufficiently small value in the range of the cell variable, except that the cell at $(0, 0)$ is set to 0 in order to derive proper initial values on the first column ($i = 0$) and the first row ($j = 0$) in the DP matrix. Figure 1(f) shows an example of the initial and derived values with the score parameters $(M, X, G_o, G_e) = (2, 3, 5, 1)$.

Finally, we describe a heuristic to terminate the band extension, which is simlilar to an algorithm introduced in the BLAST X-drop DP algorithm. To avoid unfruitful extension through an unmatched region beyond the true end of the matches, the BLAST (Altschul *et al.*, 1990) and programs developed later such as BWA-MEM (Li, 2013) and LAST (Kiełbasa *et al.*, 2011) adopted a heuristic algorithm called X-drop termination in their semi-global alignment DP routines. It terminates the extension when all the forefront cell scores are smaller than the current maximum score by at least X. Since there is no way to calculate the maximum values in

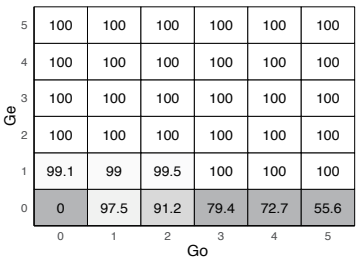
[1] Strictly speaking, we have two center cells when W is odd, but here we define the center as either of them.

```
/* initialize sequence vectors */
av <- { 0 }
bv <- { 0 }
for (i <- 0 .. BW / 2) {
    av <- shift left av
    av[0] <- (a[i]>>2) ^ (a[i]>>1)
}
for (j <- 0 .. BW / 2) {
    bv <- shift right bv
    bv[BW / 2] <- (b[j]>>2) ^ (b[j]>>1)
}

/* initialize score vectors */
ppv[0 .. BW] <- -inf
pv[0 .. BW] <- -inf, pv[BW / 2] = 0
ev[0 .. BW] <- -inf
fv[0 .. BW] <- -inf

/* initialize X-drop variable */
center_max <- pv[BW / 2]

while (until the end of the band) {
    /* X-drop termination test */
    if (pv[BW / 2] < center_max - X) {
        break
    }

    /* dynamic direction determination */
    if (pv[BW - 1] > pv[0]) {
        dir <- DOWN
    } else {
        dir <- LEFT
    }

    /* update vectors */
    if (dir is DOWN) {
        j <- j + 1
        bv <- shift right bv
        bv[BW - 1] <- 0x03 & ((b[j]>>2) ^ (b[j]>>1))

        uv <- pv
        lv <- shift right pv
        fv <- shift right fv

        if (previous direction is down) {
            ppv <- shift right ppv
        }
    } else {
        i <- i + 1
        av <- shift left av
        av[0] <- 0x03 & ((a[i]>>2) ^ (a[i]>>1))

        uv <- shift left pv
        lv <- pv
        ev <- shift left ev

        if (previous direction is right) {
            ppv <- shift left ppv
        }
    }
    ev <- max(ev - Ge, lv - Gi - Ge)
    fv <- max(fv - Ge, uv - Gi - Ge)
    scv <- shuffle(matrix, av | (bv<<2))
    cv <- max(ppv + scv, ev, fv)

    /* save vectors for use in traceback */
    store(cv) store(ev) store(fv)
    ppv <- pv
    pv <- cv

    /* update the X-drop variable */
    center_max = max(center_max, pv[BW/2])
}
```

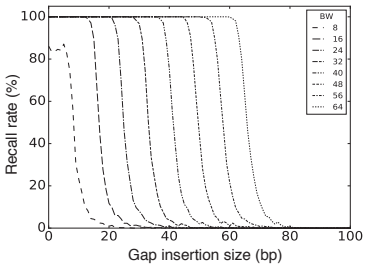
Fig. 2. Pseudocode of the Adaptive Banded DP algorithm. The ppv, pv, ev, and fv represent the second-previous S_V vector, the previous S_V, E_V, and F_V vector, respectively. The two subsequence vectors are denoted as av and bv. The X-drop threshold is denoted as X. The three binary operator on vectors, $+$, $-$, and max, are element-wise addition, subtraction, and maximum, respectively. The shift left and shift right operator shift elements in a vector leftward and rightward by one column. The shuffle operation takes an element vector as the first argument and an index vector as the second argument. The $\ll$ operator in the shuffle represents the bitwise left shift on each element in the vector.

Read num.	1185
Depth	0.254
Read length mean (bp)	998.9
Read length SD (bp)	28.0
Read length min (bp)	950
Read length max (bp)	1050
Read accuracy mean	0.8497
Read accuracy SD	0.0089
Substitution rate.	0.0151
Insertion rate.	0.0903
Deletion rate.	0.0447

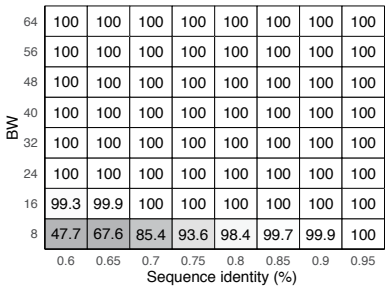
(a) Simulated read profile



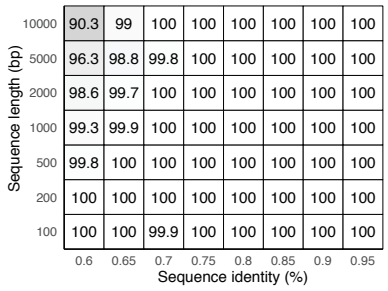
(c) Recall rate with different gap penalties



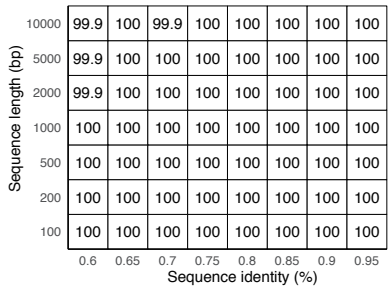
(e) Recall rate with different gap insertion sizes



(b) Recall rate with different band widths and sequence identities



W = 16



W = 24

(d) Recall rate with different sequence lengths and identities (left: W = 16, right: W = 24)

Fig. 3. Results of the recall benchmarks (a) Statistic profiles of one sample of the simulated read sets: The length and the identity were set to $1,000$ bp and 0.85, with their (SD, max, min) set to $(50, 1050, 950)$ and $(0.01, 0.86, 0.84)$, respectively. (b) Recall rate (in percent) with different band widths and sequence identities: The mean sequence length L and score parameters were set to $1,000$ and $(1, 2, 2, 1)$. 100.0 is displayed as 100. (c) Recall rate (in percent) with different gap penalties: The match award and mismatch penalty were fixed at $(M, X) = (2, 2)$. The mean sequence length and identity were set to $1,000$ and 0.75. 100.0 is displayed as 100. (d) Recall rate (in percent) with different sequence lengths and identities with two bandwidths: The result of $W = 16$ is shown in the left and $W = 24$ in the right. The scoring parameters were set to $(1, 2, 2, 1)$. 100.0 is dispalyed as 100. (e) Recall rate (in percent) with different gap insertion sizes: The mean sequence length and identitiy were set to $1,000$ and 0.75. A single contiguous gap with the length within $[0, 100]$ was inserted to the reference-side sequence at a position within $[100, 600]$ from its head. The scoring parameters were set to $(M, X, G_o, G_e) = (1, 2, 2, 1)$.

a vector efficiently, we adopted a slightly modified version of the X-drop heuristic to avoid the inefficiency. Our algorithm does not find the maximum value in the forefront vector as the original X-drop heuristic does. Instead, the extension is terminated when the score of the center cell at the forefront of the band becomes smaller than the current maximum score in the previously calculated center cells by at least X.

The whole adaptive banded DP algorithm is shown in pseudocode in Figure 2, where ppv, pv, ev, and fv correspond to the second-previous S_V vector, previous S_V, E_V, and F_V vectors in Figure 1. The two subsequence vectors are denoted as av and bv.

2.3 Relation to existing algorithms

Our approach using the anti-diagonal vector for parallelization is similar to the parallel SWG algorithm with SIMD instructions by Wozniak (1997). The Wozniak's algorithm, which mainly targeted protein sequences, had a bottleneck in its serial (unparallelized) lookups of a score matrix and later superseded by more efficient parallel algorithms by Rognes and Seeberg (2000) and Farrar (2007). In the Rognes's algorithm and the Farrar's algorithm, the DP vectors are vertical (or horizontal) in the DP matrix so that the score profile vector can be calculated by just choosing (loading) one of the precalculated four (or 20) score profile vectors that corresponds to the four nucleotides (20 amino acids). The precalculated score profile vectors were effective in eliminating the serial score-matrix lookups in the Wozniak's algorithm, while entailed the precalculation overhead and the additional memory consumption for the vectors.

The score profile vector calculation with a SIMD shuffle operation is first proposed by Wang *et al.* (2014) in their XSW program, which adopted the Farrar's SIMD-vectorized SWG algorithm. They use a pair of 16-element vector shuffle operations to calculate a 16-element score vector from a single row of 26×26 score matrix in an on-the-fly manner. The parallel score vector calculation in our algorithm can be considered a further extension of the Wang's approach; it accepts arbitrary base pairs as the input and eliminating the indexed memory access required in the Wang's algorithm when fetching a row from the score matrix. We discovered that the 4×4 score matrix for nucleotide alignment perfectly fit in a single 16-element vector while the score matrix for protein alignment does not. This enabled us to combine the anti-diagonal-parallel approach and the vectorized score profile calculation in the efficient way.

We also found that a hardware-based semi-global alignment algorithm proposed in the patent by McMillen and Ruehle (2015) also adopted a constant-width band that moves dynamically according to the values in calculated cells; however, the further details about how to move the forefront vector are not described in the mode of operation of invention. We speculate that their method moves the forefront vector so that the cell with the maximum value comes closer to the center so our approach is likely to be different in that aspect.

3 Results

We implemented our adaptive banded algorithm for x86_64 processors with Streaming SIMD Extension 4.1 (SSE4.1) instruction sets. We used the 16-bit-wide variables; eight values were retained in a single xmm SIMD register and they are processed simultaneously during the extension. The band width W was set to multiple of 8 and determined at the compile time by passing the constant value as a macro definition to the compiler.

All the benchmarking programs were implemented in the C programming language and compiled with gcc-4.9.3.

3.1 Recall benchmarks

Since our algorithm only calculates the cells in the narrowed regions, our algorithm may miss the optimal alignment that could be identified by the original semi-global DP algorithm that calculates the full DP matrix. To show the sensitivity of our algorithm is similar to that of the original semi-global DP algorithm, we compared by simulation the optimal score and the corresponding alignment identified by our algorithm and the original semi-global DP algorithm under several conditions. In the simulation, we generated simulated reads from a reference genome, and then aligned the simluated reads against the reference genome using the both algorithms, mimicking the situation in resequencing studies. We presumed that two parameters, the band width W and the score parameters (match score and mismatch score) have strong effects on the accuracy and sensitity of alignment. Throughout the experiments, we used the simple match-mismatch score model, which is represented as a tuple of four non-negative integers (M, X, G_o, G_e), where M and X are a match award and a mismatch penalty and G_o and G_e are the coefficients of the affine-gap penalty function. Remind that the match-mismatch score model is a special case of the 4×4 score matrix model as we described in the method section. Sets of $1,000$ sequence pairs were generated from the *Escherichia coli* reference genome (accession no.: NC000913) with PBSIM, a PacBio long read simulator (Ono *et al.*, 2013). Each pair consists of a simulated PacBio read and the corresponding region in the reference genome, to each of which a random sequence of 200 bp is appended in order to mimick the situation in semi-global alignment. In the following experiments, we characterized a set of sequences by the mean read length L and the mean identity I. The maximum, the minimum and the standard deviation of these two parametes were set to $1.05L$, $0.95L$ and $0.05L$, and $I + 0.01$, $I - 0.01$ and 0.01, respectively. The statistics of one of the sets of simulated reads are shown in Table (a) in Figure 3 as an example.

3.1.1 Effect of the band width on the recall rate

We first evaluated the effect of the band width W on the recall rate of the algorithm. Sequence pairs with various identities ranging in $[0.6, 0.95]$ were aligned by the adaptive banded DP algorithm with band widths ranging from 8 to 64 with a step of 8. The mean sequence length and the scoring parameters were set to $L = 1,000$ and $(M, X, G_o, G_e) = (1, 2, 2, 1)$. The result is shown in Figure 3(b), showing that our algorithm can perfectly identify the optimal scores and paths when the band width is greater than or equal to 24 and when the sequence identity is 0.6 or greater. The results are still nearly perfect when $W = 16$, while the recall rate drops significantly with $W = 8$ and sequence identities lower than 0.75.

3.1.2 Effect of gap penalties on the recall rate

The algorithm examines only the scores of the edge cells in the forefront vector when deciding the advancing direction. It might be possible that our algorithm might fail in capturing the optimal path in the narrowed band especially when the scores in the S vector is almost flat, which is likely to happen when the gap penalty is small or even zero. We evaluated the effect of small gap penalties on its recall. Both the gap open penalty G_o and the gap extension penalty G_e were independently varied from 0 to 5, with the match award and the mismatch penalty fixed to $(M, X) = (2, 2)$. The result (Fig 3(c)) shows that setting the gap extension penalty to zero severely degraded the recall rate. A small but non-zero gap insertion penalty did not affect on the sensitivity at large when the gap extension penalty is set to a value larger than or equal to 2, indicating that the algorithm also works well with the linear-gap-penalty function, where the gap penalty function is expressed in a form of $g(k) = k \cdot G$ as well as more general affine-gap-penalty functions. We also note that the parameter combinations with non-perfect recalls are impractical and thus will not be used in real analysis; (1) $G_e = 0$ implies that we can insert almost arbitrarily large gaps without incurring a good penalty, and (2) $(G_o, G_e) = (0, 1), (1, 1), (2, 1)$ all have a positive expected score for alignment of two random sequences (four bases occurring equally and independently; Vingron and Waterman (1994)).

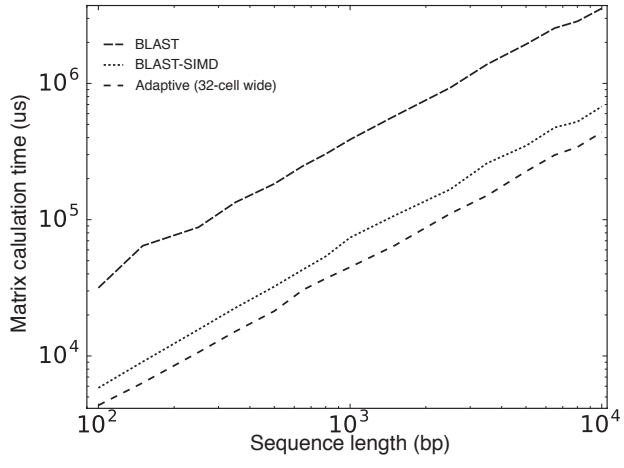


Fig. 4. Matrix calculation time (normalized to a single function call) of three algorithms, the BLAST X-drop DP (BLAST), the SIMD-vectorized BLAST X-drop DP (BLAST-SIMD), and the 32-cell-wide SIMD-vectorized adaptive banded DP (Adaptive) with different sequence length L.

3.1.3 Aligning longer sequences

Next we evaluated the effect of the lengths of query seqences on the recall rate. If we use a statically banded DP, in which the band is determined before starting to fill in values in cells, the band width we need in order to capture the optimal path in the narrowed band grows in $O(\sqrt{n})$ where n is the expected number of indel errors in a read, assuming that insertions and deletions occur at the same probability and independently. We hypothesized that a certain fixed-size band width is sufficient for capturing the optimal path in empirical settings when we use the adaptive banded DP algorithm. Sets of simulated sequences of various lengths ranging $[100, 10000]$ were given to our algorithm with three different band widths, $W = 16$, $W = 24$, and $W = 32$, and then compared the results with one given by the full semi-global alignment to see if our algorithm can output the same result to one by the full semi-global alignment algorithm. The mean sequence identitiy varied between $[0.6, 0.95]$ and the score parameters were set to $(1, 2, 2, 1)$. The results (Fig 3(d)) show that the 16-cell-wide and 24-cell-wide band may not be sufficient for aligning long (e.g., > 500) sequences with lower identity (~ 0.7) perfectly. The 32-cell-wide band perfectly output the same result as the full semi-global alignment algorithm.

3.1.4 Indel tolerance

Our algorithm is anticipated to fail in finding the optimal score when a large insertion or deletion (indel) appear in the true pairwise alignment due to the nature of banded DP algorithms. We evaluated the impact of indels in query sequences on the recall rate. We simulated an insertion by inserting a random sequence of a length specified by a parameter to the reference-side sequences. Sequence pairs of a simulated PacBio read and its corresponding region in the reference genome were generated with

parameters $L = 1,000$ and $I = 0.75$, then a random (the same as before) sequence with a certain length (denoted by l) ranging $[0, 100]$ was inserted to the reference-side sequence at a position within $[100, 600]$ bp from its 5'-end. Considering that our algorithm runs basically symmetrically between two sequences to be aligned, only insertion was tested but it does not lose the generality at large. The score parameters $(M, X, G_o, G_e) = (1, 2, 2, 1)$ were used. The result (Fig 3(d)) suggests that our algorithm can find the optimal score at an almost 100 % probability when the length of the inserted gap, l is less than $W - 4$.

3.2 Speed benchmark

We compared the matrix calculation performance of our algorithm to the well-known BLAST X-drop DP algorithm found in the NCBI BLAST+ package (version 2.2.31+). Since the original implementation had many complex arguments and conditional branches, we reimplemented the algorithm in a simple form for the comparison purpose. We also prepared a SIMD-vectorized variant of the BLAST X-drop DP. The scoring parameters and the X-drop threshold passed to the implementations were $(M, X, G_o, G_e) = (1, 2, 2, 1)$ and 30, respectively. The mean sequence identitiy was set to $I = 0.75$ and the mean length was varied within $[100, 10000]$. The program was compiled by gcc-4.9.3 with an optimization flag -O3. The result, shown in Figure 4, revealed that our adaptive banded DP algorithm was the fastest, being uniformly 8 times and 3-4 times faster than the BLAST X-drop DP and the SIMD-vectorized X-drop DP.

4 Discussions

We proposed the adaptive banded DP algorithm for nucleotide alignment that calculates only the cells in a dynamically-determined constant-wide band in the DP matrix. The use of a SIMD shuffle instruction for generating the score profile vector enabled us to execute the whole update operation in the vectorized form while adopting anti-diagonally placed vectors to eliminate dependencies between elements in a single vector, which was not achieved in the previous SIMD-vectorized DP algorithms. The experiment showed that our algorithm can output almost perfect alignments in a sense that the output alignments are equivalent to ones by the full semi-global alignment.

Although we have not provided integration yet, our algorithm can be integrated immediately with existing nucleotide local alignment programs such as nucleotide BLAST (Altschul *et al.*, 1990), BWA-MEM (Li, 2013), and BLASR (Chaisson and Tesler, 2012) to accelerate the extension step. It is also worth mentioning that the 16-bit-wide implementation of our algorithm, with a small modification, is sufficient for supporting (virtually) infinitely long query sequences. That is, the actual scores in each vector can be represented with a pair of a potentially large base value of 64-bits and an offset value of 16-bits from the base value so that the elements (i.e., offset values) in vectors fit within the 16-bit range, as we can guarantee that the range of scores in each vector can be bounded to a certain (not so large) range, assuming that the absolute values of score parameters are small integers. One possible implementation is based on a recusive function call, where the 64-bit offset value is initially set to 0 and updated when an overflow is detected in one of the cells in vectors; the 16-bit values in the vectors are subtracted by a certain value, e.g. INT16_MAX (or 32767), and the adjusted difference is accumulated into the base value. The remaining query sequences will be processed in the next tail recursive call. This modification also makes it possible for the algorithm to be applied to alignment in comparative genomics, where the alignment of long nucleotide sequences (e.g. contigs or chromosomes of a few megabases) may be required.

On the other hand, an application of our algorithm for protein sequence alignment has a difficulty in calculating the score vector because the score matrix for amino acids will not fit in a single SIMD register. Both further algorithmic engineering and much wider SIMD registers are probably required to make the adaptive banded DP algorithm run effeciently for protein alignment.

Finally, we would like to point out that it is possible to port our implementation to other architectures because we took a care for portability. Power and AArch64 architectures have their own SIMD instruction sets, AltiVec and NEON, both of which have basic arithmetic and 16-element shuffle operations required for our algorithm. The current GPU architectures (e.g. one by NVIDIA), whose synchronous threads are regarded as SIMD lanes, are also expected to efficiently handle the adaptive band DP algorithm without much hassle.

5 Availability of Code

The implementation of the algorithm and the benchmarking scripts are available at https://github.com/ocxtal/adaptivebandbench.

6 Acknowledgements

This work was supported in part by MEXT KAKENHI Grant Number 16H06279. We also thank Professor Shinichi Morishita for his kind and insightful comment on this manuscript.

References

Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. (1990). Basic local alignment search tool. *Journal of molecular biology*, **215**(3), 403–410.

Backurs, A. and Indyk, P. (2015). Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, pages 51–58. ACM.

Burrows, M. and Wheeler, D. J. (1994). A block-sorting lossless data compression algorithm.

Chaisson, M. J. and Tesler, G. (2012). Mapping single molecule sequencing reads using basic local alignment with successive refinement (BLASR): application and theory. *BMC bioinformatics*, **13**(1), 238.

Chao, K.-M., Pearson, W. R., and Miller, W. (1992). Aligning two sequences within a specified diagonal band. *Computer applications in the biosciences: CABIOS*, **8**(5), 481–487.

Daily, J. (2016). Parasail: SIMD C library for global, semi-global, and local pairwise sequence alignments. *BMC bioinformatics*, **17**(1), 81.

Döring, A., Weese, D., Rausch, T., and Reinert, K. (2008). SeqAn an efficient, generic C++ library for sequence analysis. *BMC bioinformatics*, **9**(1), 11.

Farrar, M. (2007). Striped Smith–Waterman speeds database searches six times over other SIMD implementations. *Bioinformatics*, **23**(2), 156–161.

Ferragina, P. and Manzini, G. (2000). Opportunistic data structures with applications. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 390–398. IEEE.

Flouri, T., Kobert, K., Rognes, T., and Stamatakis, A. (2015). Are all global alignment algorithms and implementations correct? *bioRxiv*.

Gotoh, O. (1982). An improved algorithm for matching biological sequences. *Journal of molecular biology*, **162**(3), 705–708.

Homer, N., Merriman, B., and Nelson, S. F. (2009). BFAST: An alignment tool for large scale genome resequencing. **4**(11), e7767.

Kiełbasa, S. M., Wan, R., Sato, K., Horton, P., and Frith, M. C. (2011). Adaptive seeds tame genomic sequence comparison. *Genome research*, **21**(3), 487–493.

Kimura, K., Koike, A., and Nakai, K. (2012). A bit-parallel dynamic programming algorithm suitable for DNA sequence alignment. *Journal of bioinformatics and computational biology*, **10**(04), 1250002.

Langmead, B. and Salzberg, S. L. (2012). Fast gapped-read alignment with Bowtie 2. *Nature methods*, **9**(4), 357–359.

Lee, W.-P., Stromberg, M. P., Ward, A., Stewart, C., Garrison, E. P., and Marth, G. T. (2014). MOSAIK: a hash-based algorithm for accurate next-generation sequencing short-read mapping. *PloS one*, **9**(3), e90581.

Li, H. (2013). Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv preprint arXiv:1303.3997*.

Li, H. and Durbin, R. (2009). Fast and accurate short read alignment with Burrows–Wheeler transform. *Bioinformatics*, **25**(14), 1754–1760.

Li, H., Ruan, J., and Durbin, R. (2008). Mapping short dna sequencing reads and calling variants using mapping quality scores. *Genome research*, **18**, 1851–1858.

McMillen, R. and Ruehle, M. (2015). Bioinformatics systems, apparatuses, and methods executed on an integrated circuit processing platform. US Patent 9,014,989.

Myers, G. (2014). Efficient local alignment discovery amongst noisy long reads. In *International Workshop on Algorithms in Bioinformatics*, pages 52–67. Springer.

Navarro, G. and Ricardo, B.-Y. (1998). A practical q-gram index for text retrieval allowing errors. *CLEI Electronic Journal*, **1**(2).

Ono, Y., Asai, K., and Hamada, M. (2013). PBSIM: PacBio reads simulator – toward accurate genome assembly. *Bioinformatics*, **29**(1), 119–121.

Rognes, T. and Seeberg, E. (2000). Six-fold speed-up of Smith–Waterman sequence database searches using parallel processing on common microprocessors. *Bioinformatics*, **16**(8), 699–706.

Smith, T. F. and Waterman, M. S. (1981). Identification of common molecular subsequences. *Journal of molecular biology*, **147**(1), 195–197.

Sović, I., Šikić, M., Wilm, A., Fenlon, S. N., Chen, S., and Nagarajan, N. (2016). Fast and sensitive mapping of nanopore sequencing reads with GraphMap. *Nature communications*, **7**.

Vingron, M. and Waterman, M. S. (1994). Sequence alignment and penalty choice: Review of concepts, case studies and implications. *Journal of molecular biology*, **235**(1), 1–12.

Wang, L., Chan, Y., Duan, X., Lan, H., Meng, X., and Liu, W. (2014). XSW: Accelerating biological database search on Xeon Phi. In *Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*, pages 950–957. IEEE.

Wozniak, A. (1997). Using video-oriented instructions to speed up sequence comparison. *Computer applications in the biosciences: CABIOS*, **13**(2), 145–150.

Zhao, M., Lee, W.-P., Garrison, E. P., and Marth, G. T. (2013). SSW library: An SIMD Smith-Waterman C/C++ library for use in genomic applications.