A Memory-Efficient Accelerator for DNA Sequence Alignment with Two-Piece Affine Gap Tracebacks

Jing-Ping Wu[1], Yi-Chien Lin[1], Ying-Wei Wu[1], Shih-Wei Hsieh[2], Ching-Hsuan Tai[2], Yi-Chang Lu[1,2]

[1]Department of Electrical Engineering, National Taiwan University, Taipei, 10617 Taiwan
[2]Graduate Institute of Electronics Engineering, National Taiwan University, Taipei, 10617 Taiwan
Email: {b05901014, b05901054, b05901100, r07943017, r08943177, yiclu}@ntu.edu.tw

***Abstract*—Previously, dynamic-programming-based DNA sequence aligners were mostly implemented with a penalty function of the one-piece affine gap model. When aligning sequences with longer gaps, the two-piece affine gap model provides better results at the cost of memory usage, which becomes an issue especially for aligners with memory-hungry traceback capabilities. In this paper, we design a memory-efficient scheme for traceback recording with the two-piece penalty scoring, so that the aligner can be realized on an ASIC. Our design is implemented with TSMC 40nm technology, and the proposed aligner can speed up pairwise alignment by 70X compared to the CPU approach.**

***Index Terms*—sequence alignment, Smith-Waterman, two-piece affine gap, traceback**

I. Introduction

Aligning two biological sequences is an important step in many biological and medical researches. With alignment results, researchers can evaluate similarities between sequences or determine the locations of variants. Dynamic-programming-based Smith-Waterman method with one-piece affine gap scoring is one of the most popular tool for sequence alignment (e.g. [1]). For a subject sequence with length m and a query sequence with length n, the conventional one-piece scoring model requires three matrices H, I, and D, all sized $m \times n$, to be filled using Eqs. (1) to (3) below. This process is usually referred as the dynamic-programming stage (DP stage) in sequence alignment.

$$H(i,j) = \max\{H(i-1,j-1) + S(i,j), \\ I(i,j), D(i,j), 0\}, \tag{1}$$

$$I(i,j) \ = \max\{H(i,j-1) - g_o,\ I(i,j-1) - g_e\}, \tag{2}$$

$$D(i,j) = \max\{H(i-1,j) - g_o,\ D(i-1,j) - g_e\}. \tag{3}$$

$S(i,j)$ gives the match score between the i^{th} nucleotide of the subject sequence and the j^{th} nucleotide of the query sequence. The g_o represents the penalty score of Gap Open, and the g_e represents the penalty scores of Gap Extension. Usually we have $g_o > g_e$, which indicates that the existence of a gap has more biological meaning than the length of a gap. Following this idea, in the cases where the gap is long, Gotoh *et al.* [2] proposed that the impact of lengths should be further discounted so that long gaps can be successfully identified. Therefore, a second set for penalty scoring (g'_o and g'_e) is introduced:

$$I'(i,j) = \max\{H(i,j-1) - g'_o,\ I'(i,j-1) - g'_e\}, \tag{4}$$

$$D'(i,j) = \max\{H(i-1,j) - g'_o,\ D'(i-1,j) - g'_e\}. \tag{5}$$

And Eq. (1) has to be revised into Eq. (6):

$$H(i,j) = \max\{H(i-1,j-1) + S(i,j), \\ I(i,j), D(i,j), I'(i,j), D'(i,j), 0\}, \tag{6}$$

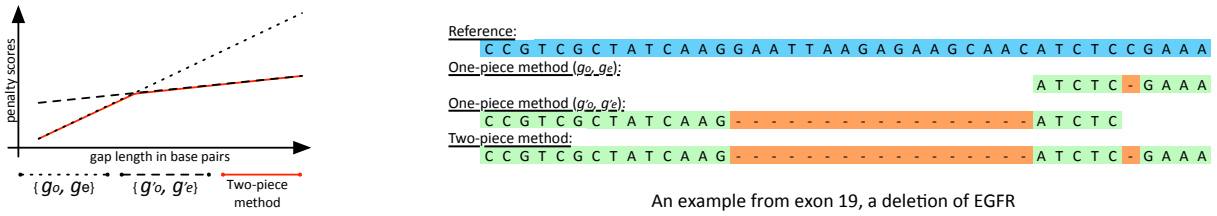
The traceback (TB) stage in sequence alignment is to determine the optimal alignment path, which can be done by retrieving records of cells calculated in the DP stage. In general, for an N-piece affine gap model, a naive implementation is to use k-bits to store the traceback information for each cell, where k is given by:

$$N = \lceil \log_2(2N+2) \rceil + 2N. \tag{7}$$

For each cell of traceback matrix T, $\lceil \log_2(2N+2) \rceil$ bits are allocated for Eq. (6), and additional $2N$ bits are allocated for the $2N$ equations generated from N sets of penalty scoring functions for insertion and deletion. Removing even one bit in each cell for traceback recording will make a big difference in memory usage.

Previous researches mainly focused on the DP stage, due to the time complexity of the algorithm. Lin *et al.* [4] has implemented a hardware that can handle both DP and TB stages based on the one-piece affine gap model. However, as mentioned above, the two-piece affine gap model is more preferred. Since memory usage is always a concern, in this work, we propose a novel scheme to reduce the traceback memory for the two-piece affine-gap model by 28%. In addition, we design a hardware aligner, including both DP and TB stages, based on this new scheme. 79 GCUPS is achieved by the hardware we implemented, which is a 71× speed-up compared to its CPU counterpart.

This work was partially supported by the Ministry of Science and Technology, Taiwan, under Grant number MOST 109-2221-E-002-177. The authors would like to thank Taiwan Semiconductor Research Institute for providing EDA Cloud services.



(a) (b)

Fig. 1: (a) Penalty scores calculated using different affine gap models. (b) An example of different alignment results using one-piece and two-piece affine gap scoring rules.

II. Lookup Table for Two-Piece Affine Gap Traceback

As suggested by [2], we have $g'_e < g_e$ and $(g_o + g_e) < (g'_o + g'_e)$, so that the set (g_o, g_e) can be used to handle shorter gaps, while (g'_o, g'_e) can be applied to cover longer gaps as shown in Fig. 1(a). $H(i,0) = H(0,j) = 0$ and $I(i,0) = I(0,j) = I'(i,0) = I'(0,j) = D(i,0) = D(0,j) = D'(i,0) = D'(0,j) = -\infty, \forall i,j$ are the boundary conditions used for Fig. 1(b).

For one-piece affine gap traceback, Lin *et al.* [4] proposed a 3-bit approach, which saves 25% of memory usage than the naive 4-bit approach. Following this idea, we find that if we compare the inequalities of $H_o = H(i,j) - g_o$, $I_e = I(i,j) - g_e$, $D_e = D(i,j) - g_e$, $H'_o = H(i,j) - g'_o$, $I'_e = I'(i,j) - g'_e$ and $D'_e = D'(i,j) - g'_e$ in advance, the total bits needed for recording two-piece traceback can be reduced. All possible conditions are summarized in TABLE I, which can be used to determine every current traceback direction based on its last traceback direction and the source of the current highest score. Combinations that are not possible for valid gap penalties are excluded from TABLE I.

In the upper part of TABLE I, the current highest score of (i,j) is the result from $H(i-1,j-1)+S(i,j)$ in Eq. (6). In such cases, for the cell (i,j), if the last traceback direction ($preTrace$) is $\mathcal{M}$ (at $(i+1,j+1)$), the traceback direction at (i,j) should be $\mathcal{M}$ as well. However, if $preTrace$ is not $\mathcal{M}$, we need to decide whether (i,j) is the starting position of a new gap or a continuation of an existing gap. Different from the conventional one-piece affine gap model, there are other four types of traceback directions in the two-piece affine gap model besides $\mathcal{M}$, which are $\mathcal{I}$, $\mathcal{D}$, $\mathcal{I}'$ and $\mathcal{D}'$. For the cases where $preTrace$ is $\mathcal{I}$ or $\mathcal{D}$, we can decide if (i,j) is the beginning of a new gap by the relationship of (H_o, I_e) or (H_o, D_e). For example, if $preTrace = \mathcal{I}$ and $H_o \geq I_e$, the current traceback direction would then be $\mathcal{M}$. On the other hand, if $I_e > H_o$, then the current traceback direction would be $\mathcal{I}$. A similar approach could be applied for the cases where $preTrace = \mathcal{D}$. As for the cases where $preTrace$ is $\mathcal{I}'$ or $\mathcal{D}'$, the same approach could also be applied, but H_o should be replaced by H'_o, I_e should be replaced by I'_e and D_e should be replaced by D'_e. To summarize, there is only one case if $preTrace = \mathcal{M}$, and only two possible cases each for

TABLE I: RULES FOR DETERMINING THE CURRENT TRACEBACK DIRECTION

score source	$H(i-1,j-1)+s(i,j)$					
$preTrace$	$\mathcal{M}$	$\mathcal{I}$	$\mathcal{D}$		$\mathcal{I}'$	$\mathcal{D}'$
$Ho \geq Ie \geq De$	$\mathcal{M}$	$\mathcal{M}$	$\mathcal{M}$	$Ho' \geq Ie' \geq De'$	$\mathcal{M}$	$\mathcal{M}$
$Ho \geq De > Ie$	$\mathcal{M}$	$\mathcal{M}$	$\mathcal{M}$	$Ho' \geq De' > Ie'$	$\mathcal{M}$	$\mathcal{M}$
$Ie > Ho \geq De$	$\mathcal{M}$	$\mathcal{I}$	$\mathcal{M}$	$Ie' > Ho' \geq De'$	$\mathcal{I}'$	$\mathcal{M}$
$Ie \geq De > Ho$	$\mathcal{M}$	$\mathcal{I}$	$\mathcal{D}$	$Ie' \geq De' > Ho'$	$\mathcal{I}'$	$\mathcal{D}'$
$De > Ho \geq Ie$	$\mathcal{M}$	$\mathcal{M}$	$\mathcal{D}$	$De' > Ho' \geq Ie'$	$\mathcal{M}$	$\mathcal{D}'$
$De > Ie > Ho$	$\mathcal{M}$	$\mathcal{I}$	$\mathcal{D}$	$De' > Ie' > Ho'$	$\mathcal{I}'$	$\mathcal{D}'$

$preTrace$	$\mathcal{M}$	$\mathcal{I}$	$\mathcal{I}'$	$\mathcal{D}$	$\mathcal{D}'$
$I(i,j)$	$\mathcal{I}$	$\mathcal{I}$	$\mathcal{I}'$	$\mathcal{D}$	$\mathcal{D}'$
$D(i,j)$	$\mathcal{D}$	$\mathcal{I}$	$\mathcal{I}'$	$\mathcal{D}$	$\mathcal{D}'$
$I'(i,j)$	$\mathcal{I}'$	$\mathcal{I}$	$\mathcal{I}'$	$\mathcal{D}$	$\mathcal{D}'$
$D'(i,j)$	$\mathcal{D}'$	$\mathcal{I}$	$\mathcal{I}'$	$\mathcal{D}$	$\mathcal{D}'$

the other types of directions. Therefore, there are 16 distinct combinations in total for the upper table in TABLE I.

When the source of the current score is either $I(i,j)$, $D(i,j)$, $I'(i,j)$ or $D'(i,j)$ in Eq. (6), the current traceback direction can be determined easily. As listed in the lower table in TABLE I, if $preTrace$ is $\mathcal{M}$, then the current traceback direction will be the same with the source of the current score. Otherwise, it will be the same with $preTrace$. This simplification is based on the fact that, for a certain gap in an alignment, the gap penalty is fixed within a gap, and which set of gap penalties is used by the gap is determined at the end of the gap. Thus, when $preTrace$ is $\mathcal{M}$ and the source of the current score is not from $H(i-1,j-1)$, it represents the end of a gap, and the current traceback direction can be uniquely determined by the source of the current score. Once the traceback direction is determined, it will not change until the next "match-cell" is encountered, reflecting the fact that the gap penalty will remain the same in a certain gap.

When elaborating TABLE I, we have made an assumption based on our observations. The assumption is that direct insertion-deletion transition will never present in DNA alignment. It is because that usually the penalty for mismatching is lower than the penalties for gap extension. Under this condition, direct insertion-deletion transition will not happen, hence TABLE I can be applied without leaving any combination. As mentioned earlier, a total of 16 combinations can be found in the upper table, and 4 combinations are in the lower table. For global alignments, 5 bits are enough to record all of

the 20 combinations. As for local alignments, 5 bits are also enough to cover all 20 + 1 combinations. Thus, we reduced the memory usage for traceback from 7-bit (given by Eq. (7)) to 5-bit, which is a 28% reduction. The algorithm used to determine the traceback path is shown as Algorithm 1. The symbol "$\nwarrow$", "$\leftarrow$" and "$\uparrow$" are the outputs from the traceback module, and the "+" symbol stands for concatenation. Although there are five different traceback directions used in the traceback algorithm, for the case $\mathcal{I}$ and $\mathcal{I}'$, both of them represent an insertion, which will cause the current position of traceback moves toward left. The only difference is the gap penalty they adopted, thus we can use one symbol "$\leftarrow$" to cover both cases. Based on the same reason, case $\mathcal{D}$ and $\mathcal{D}'$ can be covered by "$\uparrow$".

Algorithm 1: Two-piece Affine Gap Traceback

input : max index i, max index j,
traceback records T
output: alignment path

$Path \Leftarrow \emptyset, \quad PreTrace \Leftarrow \mathcal{M}$
while $(i \neq 0) \wedge (j \neq 0) \wedge (T(i,j) \neq$ "0") **do**
  $PreTrace \Leftarrow$ `TableLookUp` $(T(i,j), PreTrace)$
  switch $PreTrace$ **do**
    case $\mathcal{M}$: $Path \Leftarrow$ " $\nwarrow$ " $+ Path$, $i--$, $j--$
    case $\mathcal{I}$: $Path \Leftarrow$ " $\leftarrow$ " $+ Path$, $j--$
    case $\mathcal{D}$: $Path \Leftarrow$ " $\uparrow$ " $+ Path$, $i--$
    case $\mathcal{I}'$: $Path \Leftarrow$ " $\leftarrow$ " $+ Path$, $j--$
    case $\mathcal{D}'$: $Path \Leftarrow$ " $\uparrow$ " $+ Path$, $i--$
  end
end
alignment path $\Leftarrow Path$

III. Hardware Architecture

Dynamic Programming (DP) Module: The module consists of a PE array, one internal buffer, and 32 Echelon Shift Registers (ESR). The PE array consists of 512 PEs, which can process 512 cells located on the same anti-diagonal line at a time. The function of the internal buffer is to keep data on the block edge when the length of the subject sequence is longer than the number of PEs. The internal buffer is composed of two dual-port SRAM modules, which can support read and write simultaneously. The SRAM modules have 2048 words, and each word contains 80 bits. The ESR is a module used to synchronize output data generated by the PE array.

The PE array processes the alignment anti-diagonally. In one-piece alignment, each DP cell needs information from three sources, which include its upper, left and upper left cells. A PE of two-piece alignment needs two more scores, which come from its left and upper cells, but calculated with the second set of penalty scores. Thus, it will take score from five directions to implement two-piece alignment. It is also essential to record the max score and from which cell the max score comes, so that we can obtain the starting point of traceback.

Since cells will be updated anti-diagonally, cells in the same column will be updated in different cycles. The configuration of the ESR is shown in Fig. 3, where the numbers labeled on the cells of the DP matrix indicate which cycle each cell will be updated. In order to write data in the same column of the matrix into the SRAM at the same cycle, we need to use ESRs. Since each SRAM can store 16 directions in one cycle, each SRAM will read data from 16 PEs, so we need 32 = 512/16 SRAMs modules and 32 ESR. For data from n^{th} PE in every 16 PEs, $(16 - n)$ shifts are introduced before the data being written into the SRAM. As a consequence, output data generated in different cycles will be synchronized before written into the SRAM. As shown in Fig. 3, though cells in the same column will be updated in different cycles, it will also need to pass different numbers of shift registers, and the data will be written into the SRAM in the same cycle eventually. As a result, the n^{th} column of each SRAM will store data of the n^{th} column of DP matrix. The purpose of storing data to the DP matrix in this manner is to simplify the process of data prefetching in the Traceback module since it prefetchs data columnwisely.

Traceback Record Blocks: As the PE array keeps operating, each PE will generate traceback records following the rules summarized in TABLE I. These records are stored in Traceback Record Block (TRB), an array formed by SRAM modules. Each SRAM module is a single-port design of 2048 words, and there are 80 bits in each word. Since each traceback record takes 5 bits, one word can store the information from 16 traceback cells in one clock cycle. With 512 parallel PEs, 32 = 512/16 SRAM modules must operate simultaneously in one TRB to save data generated in every cycle. In order to pipeline DP and TB, two sets of TRB were used. The traceback record of the first alignment will be written into the first TRB, and the Traceback Module can start tracing back by reading data in the first TRB when the DP is done. In the same time, the second pass of DP can start immediately and write new traceback records into the second TRB without overwriting the data needed for the first traceback. Afterwards, the DP Module and Traceback Module will switch to read and write the other TRB.

Traceback (TB) Module: When performing traceback processes, a 16 $\times$ 5 = 80-bits long word will be fetched from TRB in one cycle. It should be noted that not every cell will be used in the traceback stage. Therefore, fetching cells that are not likely to be used is not desirable. We decide to fetch the cells dynamically based on the current position of traceback. Two memory blocks are used to store the data from TRB, and once the position of traceback nears the boundary of the block that is currently being used, the traceback module will record the current position, and then fill the other block based on this information. During this prefetch process, the memory blocks are filled from right to left, column by column. Since the traceback process always follows the directions toward left, up or upper-left, by filling the memory blocks in this order, the traceback process only needs to halt one or two cycles for the prefetch process. The prefetch address module resolves the cases where the stored position is not a multiple of 16. In such cases, the prefetch address module will request data from two

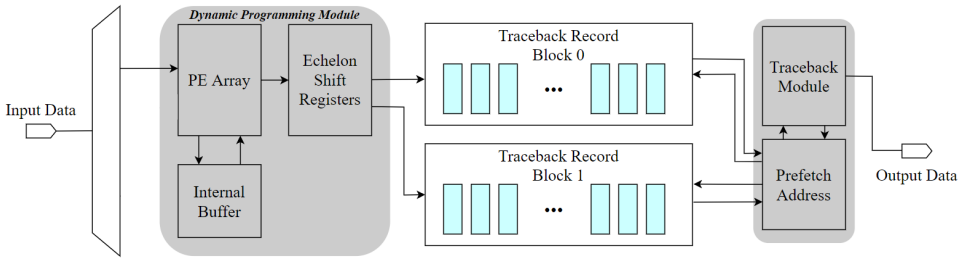


Fig. 2: The hardware architecture of the aligner with traceback.

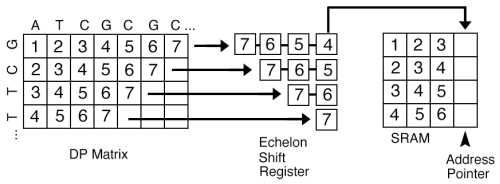


Fig. 3: Writing data through Echelon Shift Register.

SRAMs and send the rearranged data to the traceback module.

IV. Results

A. CPU and GPU

Our CPU code is a modified version of the KSW2 library from a versatile nucleotide sequence mapper, minimap2, developed by Li [5]. For fair comparison, the near-optimal banded Suzuki-Kasahara algorithm [6] in KSW2 is replaced by the original global optimal alignment version. When running on an Intel Core i7-7700 processor with 64 GB DRAM installed, a performance of 1.12 GCUPS (Giga Cell Updates Per Second) is achieved. Most GPU implementations of sequence alignment are based on one-piece affine gap scoring. We modified the state-of-the-art GPU based library GASAL2 from Ahmed *et al.* [7] to support the two-piece affine gap scoring scheme. When performing the global alignment with traceback computation, the original one-piece GASAL2 could reach 88.97 GCUPS, while the modified two-piece version can achieve 56.49 GCUPS. It is because the two-piece affine gap scoring requires roughly 65% more multiply-add operations and 72% more comparison instructions than the one-piece version. It should be noted that all of the GPU code in this work is compiled with CUDA version 9.1 and executed on a single NVIDIA Geforce GTX 1080 Ti GPU.

B. ASIC

The hardware is implemented with TSMC 40nm technology. According to the post-layout simulation, our hardware can achieve 79.65 GCUPS. We obtain a 71.11× speed-up compared with CPU and a 1.41× speed-up with GPU. The power consumption of CPU is 20.85 W and that of GPU is 193.47 W. In contrast, our hardware consumes only 1.81 W, which is 150× more efficient then the GPU platform. The chip specifications are summarized in TABLE II.

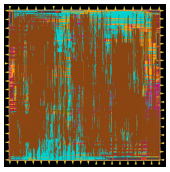


Fig. 4: The layout of the proposed hardware accelerator.

TABLE II: The Specifications of the Hardware Aligner

Cell library	TSMC 40 nm
Clock	308 MHz
Core area	38.46 mm^2
Chip area	38.96 mm^2
Gate count	1,122,876
Dual-Port SRAM usage	40 KB
Single-Port SRAM usage	5 MB
Power	1815 mW

TABLE III: Comparison Between Platforms

Devices	Speed (GCUPS)	Power (W)	Speedup	Efficiency (GCUPS/W)
CPU	1.12	20.85	1	0.0537
GPU	56.49	193.47	50.43	0.292
Ours	79.65	1.81	71.11	43.9

V. Conclusions

In this paper, we implement a hardware accelerator for DNA sequence alignment with the two-piece affine gap traceback. In our design, only 5 bits are needed to store the direction record of each DP cell, which uses 28% less memory than the naive implementation. Our design can align a sequence pair up to 2048×2048, and a 71× speed-up can be achieved compared to its CPU counterpart.

References

[1] O. Gotoh, "An improved algorithm for matching biological sequences," *Journal of Molecular Biology*, 162(3), pp. 705–708, 1982.

[2] O. Gotoh, "Optimal sequence alignment allowing for long gaps," *Bulletin of Mathematical Biology*, 52(3), pp. 359–373, 1990.

[3] X. Fei, Z. Dan, L. Lina, M. Xin, and Z. Chunlei, "FPGASW: accelerating large-scale smith-waterman sequence alignment application with backtracking on FPGA linear systolic array," *Interdisciplinary Sciences: Computational Life Sciences*, 10(1), pp. 176–188, 2018.

[4] M. J. Lin, Y. C. Li, and Y. C. Lu, "Hardware accelerator design for dynamic-programming-based protein sequence alignment with affine gap tracebacks," *2019 IEEE Biomedical Circuits and Systems Conference*, pp. 1–4, 2019.

[5] H. Li, "Minimap2: pairwise alignment for nucleotide sequences," *Bioinformatics*, 34(18), pp. 3094–3100, 2018.

[6] H. Suzuki and M. Kasahara, "Introducing difference recurrence relations for faster semi-global alignment of long sequences," *BMC Bioinformatics*, 19(1), pp. 33–47, 2018.

[7] N. Ahmed, J. Lévy, S. Ren, H. Mushtaq, K. Bertels, and Z. Al-Ars, "GASAL2: a GPU accelerated sequence alignment library for high-throughput NGS data," *BMC Bioinformatics*, 20(1), pp. 520, 2019.