Int J Parallel Prog (2017) 45:1420–1460
DOI 10.1007/s10766-017-0495-0

3D-Stacked Many-Core Architecture for Biological Sequence Analysis Problems

Pei Liu[1] · Ahmed Hemani[1] · Kolin Paul[2] · Christian Weis[3] · Matthias Jung[3] · Norbert Wehn[3]

Received: 14 June 2016 / Accepted: 23 February 2017 / Published online: 11 April 2017

Abstract Sequence analysis plays extremely important role in bioinformatics, and most applications of which have compute intensive kernels consuming over 70% of total execution time. By exploiting the compute intensive execution stages of popular sequence analysis applications, we present and evaluate a VLSI architecture with a focus on those that target at biological sequences directly, including pairwise sequence alignment, multiple sequence alignment, database search, and short read sequence mappings. Based on coarse grained reconfigurable array we propose the use of many-core and 3D-stacked technologies to gain further improvement over memory subsystem, which gives another order of magnitude speedup from high bandwidth and

✉ Pei Liu
peiliu@kth.se

Ahmed Hemani
hemani@kth.se

Kolin Paul
kolin@cse.iitd.ac.in

Christian Weis
weis@eit.uni-kl.de

Matthias Jung
jungma@eit.uni-kl.de

Norbert Wehn
wehn@eit.uni-kl.de

[1] Department of Electronic System, School of ICT, KTH Royal Instaitute of Technology, 16440 Stockholm, Sweden

[2] Department of Computer Science and Engineering, Indian Institute of Technology, New Delhi, India

[3] Microelectronic Systems Design Research Group, University of Kaiserslautern, Kaiserslautern, Germany

low access latency. We analyze our approach in terms of its throughput and efficiency for different application mappings. Initial experimental results are evaluated from a stripped down implementation in a commodity FPGA, and then we scale the results to estimate the performance of our architecture with 9 layers of 70 mm^2 stacked wafers in 45-nm process. We demonstrate numerous estimated speedups better than corresponding existed hardware accelerator platforms for at least 40 times for the entire range of applications and datasets of interest. In comparison, the alternative FPGA based accelerators deliver only improvement for single application, while GPGPUs perform not well enough on accelerating program kernel with random memory access and integer addition/comparison operations.

Keywords Accelerator architectures · Application specific integrated circuits · Bioinformatics · Computational biology · Coprocessors · Reconfigurable architectures · Three-dimensional integrated circuits

1 Introduction

Sequence analysis in bioinformatics field is an enabling technology that allows researchers to process biological sequence data. It finds similarity between biological sequences in same type, e.g., *deoxyribonucleic acids* (DNA), *ribonucleic acids* (RNA), and *amino acids* (protein). There have been a large number of programs developed with diverse approaches to solve biological sequence analysis problems, including pairwise sequence alignment, multiple sequence alignment (MSA), database search, genomic analysis, motif finding, short read sequence mapping and so on.

Similar with other scientific research domains, a dominant portion of overall execution time is consumed by a small number of kernel functions in biological sequence analysis applications, thus exporting those functions to hardware accelerators, e.g., FPGAs and GPGPUs, becomes a hot topic in recent years [1–30]. It's particularly noticeable that those hardware accelerators rarely reach typical biology labs; bioinformatics researchers still prefer using general purpose platforms such as workstations, clusters, supercomputers, or clouds.

One of the key reasons is that previous researches mostly focused on accelerating individual processing stages of a specific application, thus researchers facing real problems do not have the flexibility to adopt different accelerated workflows. Typical bioinformatics researches usually replicates multiple workflows using various similar applications of the same purpose in each stage to ensure high sensitivity. Meanwhile, there is no significant "best" or "worst" program among others in biological analysis, since the results vary in practice.

Although bioinformatics researchers prefer general purpose platforms, the high-end workstations cannot keep pace with rapidly growing problem size. *Next Generation Sequencing* (NGS) now allow genomes to be sequenced more quickly and cheaply than ever before, thus the size of sequence databases is increasing exponentially, as well as the problem size of database oriented sequence analysis.

Cloud computing is somehow expected to be a solution for high performance computing, but its general availability has caused a big waste of time during data distribution. For every specific task, all the necessary data have to be transferred from storage server nodes to computing nodes. This would consume a lot of time in case of large data, e.g., 70× coverage for whole genome sequencing would be over 330 GB in size, compressed. Therefore cloud computing is less efficiency than dedicated platform in such cases [31].

In order to address both performance and availability gap of biological sequence analysis problems, we turn to architectural innovation. Our goal is to provide an effective solution for multiple applications in biological sequence analysis workflow, giving computational biology researchers flexible choices over popular approaches.

The remainder of this paper is organized as follows. Section 2 details the contributions of our paper. Section 3 introduces a series of applications in biological sequence analysis, and presents a performance study that some common elements could be figured out within their compute intensive kernels. Section 4 describes an efficient implementation of coarse grained reconfigurable array (CGRA) using integer addition/comparison operations, while Sect. 5 introduces a many-core architecture with 3D stacked DRAM so that CGRA based sub-processor could benefit from both circuit-switching and exclusive DRAM access policy. We analyze our approach in terms of its throughput and efficiency for application mappings in Sect. 6. Experimental results are presented in Sect. 7. Our architecture (with its concomitant software support) is the first that accommodates a series of applications covering different stages of biological sequence analysis workflow while providing both high performance and energy efficiency. Estimated results show that our design outperforms all known approaches for the entire range of applications and datasets of interest. Section 8 introduces related work, and Sect. 9 offers conclusions.

2 Contributions

We present a novel architecture for biological sequence analysis problems, using CGRA, many-core and 3D-stacked technologies. In summary we make the following contributions:

- We present an analysis of multiple applications in different subfields of biological sequence analysis. We categorize those applications with computation stages and data flow, then identify the kernel functions which consumes a dominant portion of overall execution time. Then we summarize the similarity of operations and datapath of target applications, enabling the use of a reconfigurable architecture to suite all the requirements.
- We describe a reconfigurable *processing element* (PE) which is designed to facilitate both simplification and scalability with integer operations. An array of PEs forming a CGRA as a processing core can be reconfigured to accelerate target applications, while shared buffers are efficiently interconnected to maximize throughput and minimize congestion.
- We introduce a memory access policy to maximize the throughput of local DRAM access in sub-processor while devoid bus congestion within a circuit-switching many-core system. Benefit from this policy, 3D-stacked DRAM has been adopted, which provides both high bandwidth and low access latency.

- We present that our architecture is specially customized for biological sequence analysis problems in terms of both operations and datapath. The compute intensive kernel functions of target biological sequence analysis applications can be well mapped to our architecture.
- We present results from prototyping stripped down implementation of our design on a commodity FPGA, representing the performance of a single processing core of our architecture. Then we scale the results to give a plausible estimation of our design with many-core and 3D stacked architecture. Based on conservative area estimation and simulation results, we show that our design achieves the highest performance and greatest energy efficiency comparing with all other known accelerators.

3 Biological Sequence Analysis Applications

In this Section, we introduce all the biological sequence analysis algorithms and applications that have been successfully accelerated with our architecture.

3.1 Pairwise Sequence Alignment

The pairwise sequence alignment algorithms are the most important components of biological sequence analysis field; most famous sequence analysis applications have incorporated with one or more types of pairwise sequence alignment algorithms as part of the processing flow [32–36]. There are two major classes in pairwise sequence alignment: *global* and *local* alignment. In case of *global* sequence alignment, two individual sequences of similar length are aligned from one end to another, while *local* sequence alignment only relatively align the subsequences within the originals that present the highest similarity.

For sequence a and b with length m and n respectively, adopting pairwise sequence alignment would build a distance score matrix filled with alignment scores at first, where each element is the best alignment score representing the evolution distance between the segments in the two individual sequences. Needleman and Wunsch [37] is the most popular *global* alignment algorithm. For *Needleman–Wunsch* algorithm using *affine gap*, the distance scores between sequence a and b are given recursively to fill the distance score matrix as Algorithm 1:

Algorithm 1 Distance score matrix filling stage of Needleman-Wunsch Algorithm (Affine Gap)

```
 1: $F_{0,0} = M_{0,0} = I_{0,0} = D_{0,0} = 0$          // Initialization
 2: for $i = 1$ to $H, j = 1$ to $L$
 3:         $M_{i,0} = -e + M_{i-1,0}$ , $M_{0,j} = -e + M_{0,j-1}$ // Initialization
 4: end $i, j$
 5: for $i = 1$ to $H$ , $j = 1$ to $L$
 6:     $M_{i,j} = \max\{ M_{i-1,j-1}, I_{i-1,j-1}, D_{i-1,j-1}\} + S_{i,j}$ // Matching
 7:     $I_{i,j} = \max\{ M_{i-1,j} - d, I_{i-1,j} - e \}$          // Insertion
 8:     $D_{i,j} = \max\{ M_{i,j-1} - d, D_{i,j-1} - e \}$          // Deletion
 9:     $F_{i,j} = \max\{ M_{i,j}, I_{i,j}, D_{i,j} \}$             // Filling
10: end $i, j$
```

- For a matching event between segment i from sequence a and segment j from sequence b, which means that i and j are identical character symbol, the matching score $M_{i,j}$ is given by the maximum alignment score of segment pair at ($i - 1$, $j - 1$) plus $S_{i,j}$, where $S_{i,j}$ is the *substitution matrix* [38] score with entry of residues i, j .
- For an insertion event between i and j, which means that j is likely to be an inserted symbol character that different from $i - 1$ and i, the insertion score $I_{i,j}$ is given by the maximum value between the matching score of segment pair ($i - 1$, j) minus a gap open d and the insertion score of which minus a gap extension e.
- For a deletion event between i and j, which means that a character symbol in b is likely being deleted and j is the same as $i - 1$, the deletion score $D_{i,j}$ is given by the maximum value between matching score of segment pair (i, $j - 1$) minus a gap open d and insertion score of which minus a gap extension e.
- The distance score matrix element at location (i, j) is then filled by the maximum value of corresponding matching, insertion and deletion scores.

The gap penalties depend on the length of the gap and are generally assumed to be independent of the gap residues. For *linear gap*, the value is given by a constant penalty d times a linear function of gap length g as (1). For *affine gap*, the constant penalty d is predefined as well, while a linear penalty is given to subsequent gap extensions as (2).

$$Penalty_{linear} = d \times g \tag{1}$$

$$Penalty_{affine} = d + e \times (g - 1) \tag{2}$$

Smith and Waterman [39] algorithm is the standard model for *local* sequence alignment. The distance score matrix filling stage of *Smith–Waterman* is slightly different from *Needleman–Wunsch* as Algorithm 2:

Algorithm 2 Distance score matrix filling stage of *Smith-Waterman* Algorithm (Affine Gap)

```
 1: $F_{0,0} = M_{0,0} = I_{0,0} = D_{0,0} = 0$                    // Initialization
 2: for $i = 1$ to $H, j = 1$ to $L$
 3:       $M_{i,0} = 0$ , $M_{0,j} = 0$                             // Initialization
 4: end $i, j$
 5: for $i = 1$ to $H$ , $j = 1$ to $L$
 6:     $M_{i,j} = \max\{\{\{ M_{i-1,j-1}, I_{i-1,j-1}, D_{i-1,j-1}\} + S_{i,j} \}, 0\}$ // M
 7:     $I_{i,j} = \max\{ M_{i-1,j} - d, I_{i-1,j} - e \}$            // Insertion
 8:     $D_{i,j} = \max\{ M_{i,j-1} - d, D_{i,j-1} - e \}$            // Deletion
 9:     $F_{i,j} = \max\{ M_{i,j}, I_{i,j}, D_{i,j} \}$                 // Filling
10: end $i, j$
```

During the distance score matrix filling stage, there is not of much differences between *Needleman–Wunsch* and *Smith–Waterman* algorithm in the form of computation. Gapped penalties are replaced by "0" during initialization, and another "0" is added for comparison in case of matching event. The sequential execution flow of the distance score filling stage for sequences using either *global* or *local* sequence alignment is demonstrated as Fig. 1.

After generation of the distance score matrix, both *global* and *local* sequence alignment algorithms adopt a *backtracking* stage to find the final optimal alignment result. A path of relatively max values from the bottom-right to top-left cell of the score matrix has to be recovered. For *Needleman–Wunsch* algorithm, the best *global* alignment chain is established by tracking back diagonally from position (m, n), which must be the element with global maximum score, to (1,1) of the filled m-by-n distance score matrix, sequentially given by (3) and demonstrated as Fig. 2a:

$$Backtrack_{i,j} = \max\{F_{i-1,j}, F_{i,j-1}, F_{i-1,j-1}\} \tag{3}$$

And for *Smith–Waterman*, the starting point of best *local* alignment chain is given by the global maximum distance score as (4), and then tracking back diagonally using (3) till "0" as (5) and Fig. 2b. Those zeros that different from *Needleman–Wunsch*

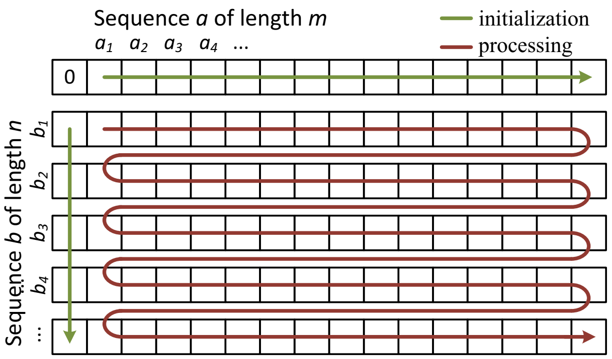

Sequence a of length m
a_1 a_2 a_3 a_4 ...
initialization
processing
0
Sequence b of length n
b_1
b_2
b_3
b_4
...

Fig. 1 Sequential execution of distance score matrix filling stage in pairwise sequence alignment

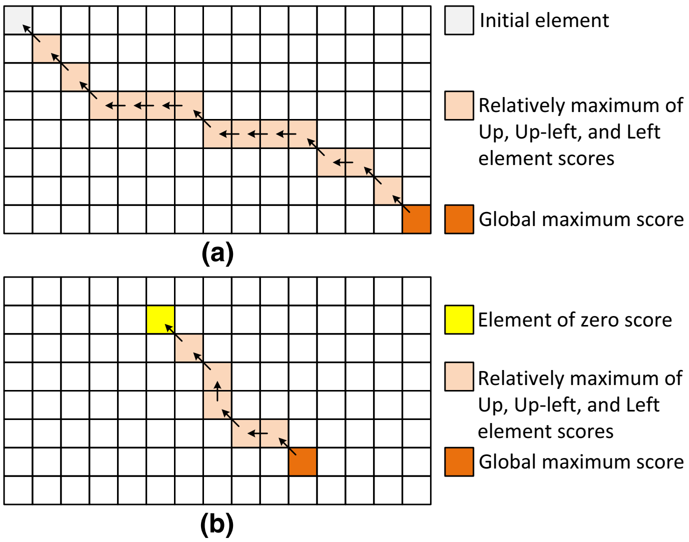

Initial element
Relatively maximum of Up, Up-left, and Left element scores
Global maximum score
(a)
Element of zero score
Relatively maximum of Up, Up-left, and Left element scores
Global maximum score
(b)

Fig. 2 Backtracking stage of **a** global sequence alignment, and **b** local sequence alignment

algorithm would "reset" the alignment in a region where the other option element scores are negative values, which does not help to find an aligned subsequence.

$$Start_{local} = \max_{i,j=1}^{m,n}\{M_{i,j}\} \tag{4}$$

$$Stop_{local} = 0 \tag{5}$$

3.2 Multiple Sequence Alignment

Multiple sequence alignment is used to discover conserved subsequences across multiple long sequences of similar length, so that sequence homology can be inferred to assess the shared evolutionary origins. The MSA problem is an extension of pairwise sequence alignment, which is NP-Hard, made it impractical to give an exact solution. Considering homologous sequences are evolutionarily related, MSA programs often incorporate heuristics progressive alignment to solve this problem. Those popular multiple sequence alignment applications such as ClustalW [32] and T-Coffee [33] shares the same heuristic idea consists of three main stages:

- All sequences to be processed are pairwise aligned with each other in order to give corresponding similarity distance score matrices, representing the divergence of each pair of sequences. This stage is performed with deterministic method using distance score matrix filling of *local* sequence alignment.
- A phylogenetic-like guided tree is calculated from the distance score matrices with its leaves representing sequences and sub sequence groups. The branch lengths reflect sequence divergences along with tree topology.
- All the sequences are progressively aligned according to the branching order of the guide tree, from leaf to root. Profiles are often created for tree branches instead

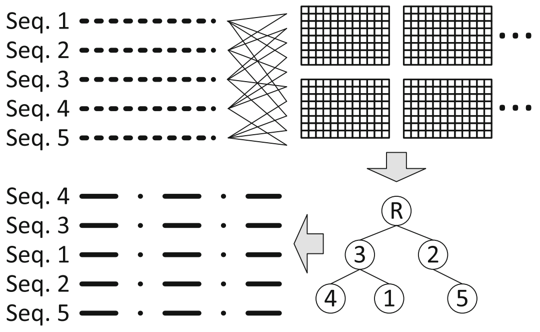


Fig. 3 Processing flow of multiple sequence alignment

of aligning two subgroups of sequences directly, but the alignment processes of profile-wise, profile-sequence and sequence-wise are all equivalent to pairwise sequence alignment. The tree topology indicates the order of pairwise sequence alignment for left and right node/branch of each ancestor tree node.

Therefore solving MSA problem can be seen as solving a series of pairwise sequence alignment problems and sub-problems, along with building a heuristic tree. An example of five sequences of similar length being processed with MSA is shown as Fig. 3.

3.3 Database Search

Database searching is used to uncover homologous sequences, which might be the most frequently used applications in computational biology. Considering the exponentially growing size of biological sequence databases, it's not applicable to use deterministic approach such as pairwise sequence alignment directly on database searching, therefore heuristic approaches have been introduced to find a suspecting region to be further verified. The heuristic idea adopted in the most popular database searching toolset, BLAST [34], is that the pairwise sequence alignment would be executed only after two small regions of exact match has been found nearby.

During data preparation, a large database is partitioned with overriding area on partition edge, then the *k-mer* set in each partition is used to generate large hash tables respectively. *k-mer* means a group of substrings of length *k* belongs to the original string; all the *k-mers* are different from each other; the original sequence string can be formed as a combination of all its *k-mers* with overlapping and duplication.

After generation of the large hash tables for each database partition, corresponding smaller pointer tables are built as entry index for each large hash table. It's used to address a small range of entries in the large hash table, thus it's not necessary to traverse the whole large table on every access.

Searching against a indexed database using BLAST, the heuristic approach consists of three stages:

- For each query sequence, all the *k-mers* and their close similarities are obtained and tested against a user-defined threshold. Those passed strings are called *seed* strings. The *seed* length *k* is normally 3–5 for protein and 11 for DNA/RNA sequences. All

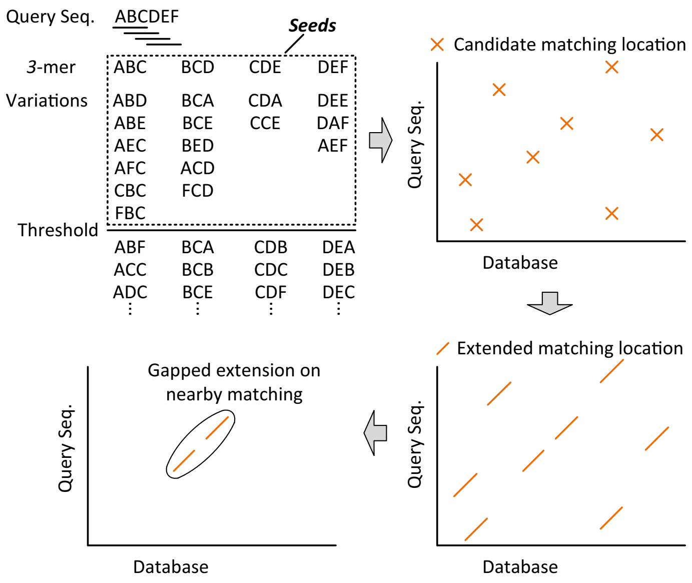


Fig. 4 Processing flow of BLAST for database searching

the *seeds* are used to build a *bloom filter* [40]. The original database or its partitions is inputted to test against the *bloom filter*, in order to get a series of candidate matching locations including fake-positives. For each candidate matching location, all the *seeds* are evaluated, which produces the exact *seed-location* matching pairs.

- For each *seed-location* matching pair, the *seed* is stepwise extended from both ends towards the query sequence, where pairwise distance scores are generated between *seed* extension and corresponding extension of matched database string. The extension does not stop until the accumulated total score reaches user-defined threshold.
- Subsequences with extension distance scores higher than user-defined threshold will be used in the third stage for evaluation. Pairwise *local* sequence alignment is used to verify the similarity between query sequence and database at the suspected regions that indicated by two nearby extensions.

An example of *seed* generation for query sequence ABCDEF is shown in Fig. 4, along with following processes of matching, ungapped extension and gapped extension in BFAST [35] for database searching.

3.4 Short Read Sequence Mapping

Short read sequence mapping is a general category of applications that finding the corresponding positions of short fragments to reconstruct the original form. NGS technologies randomly breaks down many sample organism copies of genome or protein into small pieces, then "read" in parallel and output as huge amount of randomly fragmented strings with equal length. It guarantees read quality by multiple times of redundant coverage over original organism sample. Those fragmented strings called "*read*" are typically representing 50–400 *base-pairs* (bp) in length for DNA. In order to reconstruct the original sequence structure of target organism sample, a mapping stage is introduced with two different approaches: de novo assembly and reference-guided assembly. In this paper we focus on the reference-guided assembly cases.

Reference-guided assembly is to reconstruct the original genome sequences of target organism sample when a sequence model is existed. Either indexing the input reads and then scanning through the reference model, or aligning each read independently against indexed reference sequence, nearly all the programs within this category share the same principle: index, search, and pairwise *local* sequence alignment. Rapidly developing sequencing technologies produces more and more read data than ever before, which make the read indexing approaches to be deprecated in real practice nowadays, thus we focus on database indexing methods. There are mainly two types of database index methods, hash-index and *Ferragina–Manzini index* (FM-index) [41].

Hash-index based mapping is a deterministic method. The reference sequence model is used to generate hash index and corresponding pointer tables during data preparation, which is very similar to the data preparation part of BLAST. During mapping stage, the computation is focused on searching indexed database using hashed *seed* of each short read, in order to determine the candidate matching location:

- For each *read*, the head part of each is obtained as *seed* string and properly hashed.
- *Prefix* part of each hashed *seed* is used to access the pointer table, and a valid hit would give an address range pointing to the large hash table. *Suffix* part of the hashed *seed* is then compared with a series hash values retrieved from the large hash table respectively. Each equivalent indicates an exact *seed* matching between the query sequence and the database, called *candidate matching location*.
- Pairwise *local* sequence alignment is executed between the complete short read and the reference sequence at each candidate matching location for verification.

The hash-index method requires large amount of memory space to store the index tables and the reference sequence. The software execution of typical hash-index based short read sequence mapping is demonstrated as Fig. 5.

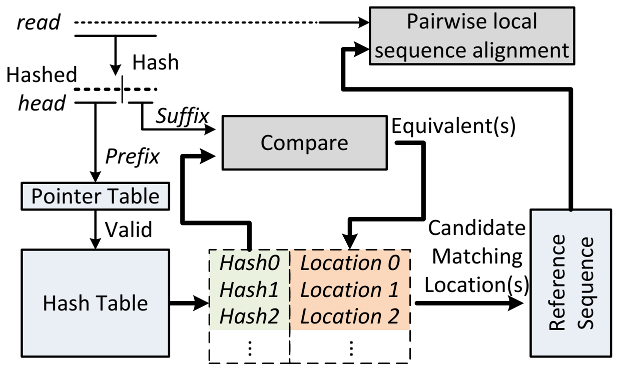


Fig. 5 Processing flow of hash-index based short read sequence mapping

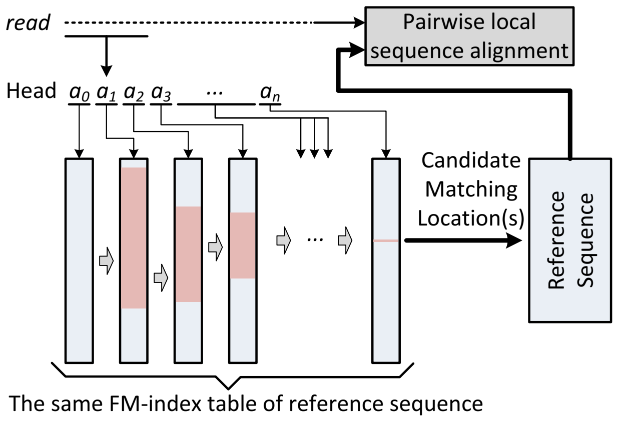

read
Pairwise local sequence alignment
Head a_0 a_1 a_2 a_3 ... a_n
Candidate Matching Location(s)
Reference Sequence
The same FM-index table of reference sequence

Fig. 6 Processing flow FM-index based short read sequence mapping

With the help of *Burrows–Wheeler transform* (BWT) [42], The FM-index guided assembly reduces the memory footprint by an order of magnitude with backward search. The FM-index is created by encoding the BWT table for *suffix array* [43] of the reference sequence model. In case of exact matching with FM-index, the execution is mainly in two stages:

- For each *read*, the head part is obtained as *seed* to query against the FM-index table. Sequentially evaluating every character of the *seed*, two pointers *top* and *bottom* are updated that specify the range of positions of the verified pattern in the *suffix array*. When the two pointers are equal or if *top* is less than *bottom*, the search is terminated indicating target *seed* does not exist in the reference sequence. After processing the last character of the *seed* without termination, the two pointers indicate a number of *candidate matching locations* where the *seed* string are located in reference sequence.
- Pairwise *local* sequence alignment is then executed between the *read* and original sequence model at each *candidate matching location* for verification.

Figure 6 demonstrates exact backward searching with FM-index based method without termination. In case of termination of backward searching, which indicates unmatched *seed* between the corresponding *read* and the reference sequence, this *read* will be marked as unsuccessful mapped.

For inexact matching search, which is the real case, it is required to exhaustively traverse all the possible symbol combinations for user-defined mismatch counts, thus the execution time is much more than exact matching. The processing flow would be a branched structure, which will be discussed in Sect. 6.

3.5 Application Analysis

Our target biological sequence analysis take original biological sequences (e.g., DNA, RNA, and protein) as input. A list of execution stages of those applications under analysis are shown in Table 1. It can be seen that most of the operations during

Table 1 Main execution stages of target applications

Target applications	Execution stages
Pairwise sequence alignment	Distance score matrix filling
	Backtracking
Multiple sequence alignment	Distance score matrix filling
	Guided tree construction
	Pairwise local sequence alignment
Database searching	Seed generation
	Seed matching[a]
	Seed extension[b]
	Pairwise local sequence alignment
Short read sequence mapping (hash-index)	Seed hashing
	2-hit string matching
	Pairwise local sequence alignment
Short read sequence mapping (FM-index)	Backward searching
	Pairwise local sequence alignment

[a] Bloom filter adopted in software execution
[b] Including distance score matrix filing

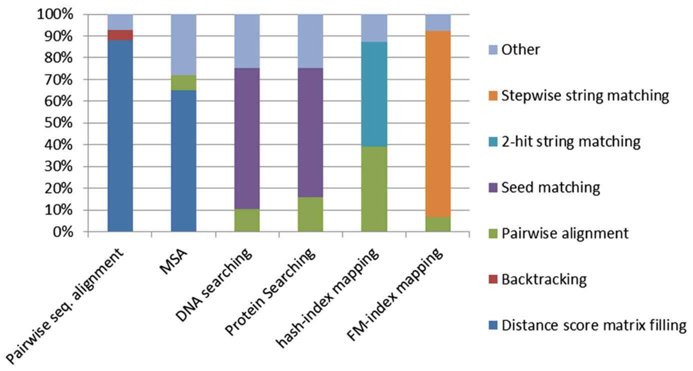


Fig. 7 Profiling of biological sequence analysis applications

execution are variations of pairwise sequence alignment or its sub-stages, and substring matchings. The profiling of those applications presenting the proportion of time spent in the different execution stages are shown as Fig. 7. It can be seen that a dominant portion of overall execution time is consumed by a few stages of all the applications of interest, which makes them to be the initial target for parallel acceleration. Meanwhile the rest stages are also not negligible, since the speedup of overall computation is limited by the time needed for the sequential fraction of the application as *Amdahl's law* [44].

The main operations within distance score matrix filling stage of pairwise sequence alignment are addition and comparison according to Algorithms 1 and 2, thus makes it

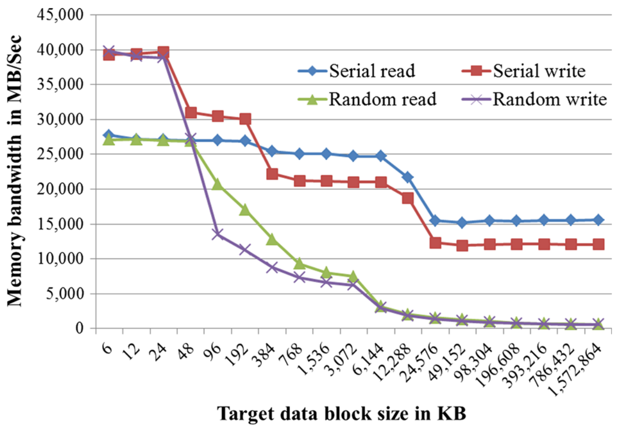


Fig. 8 Real memory bandwidth with different data block size on i7-4960×

a considerable basement to be further derived for other targeting computation stages. By exploiting the common feature between distance score matrix filling, backtracking, and string matching operations, we are able to propose a generalized architecture for acceleration of those compute intensive functions in different biological sequence analysis applications.

Figure 8 demonstrates benchmark results for memory bandwidth utilization generated by *Randmem* [45] on our i7-4960× platform, which has a theoretical maximum memory bandwidth of 51.2 GB/s (DDR3-1600 in 4 channels) [46,47]. For various size of data blocks stored in DDR3 memory, randomly reading from or writing to one 64 bit double/integer value is normalized as real memory bandwidth, which dropped to 575–1556 MB/s in case of target data blocks larger than on-chip cache size (15 MB on i7-4960× processor).

The principle of random cases in *Randmem* benchmark is similar with the case of string matching in biological sequence analysis. During the string matching, various tables stored in main buffer are randomly retrieved with tiny payload. It could hardly benefit from bursts, as the payload is very small on each access and the access address is unpredictable. This principle leads to extremely low memory bandwidth utilization in real, which is the major cause of the sharp drop over random access curves in Fig. 8. FPGA and GPGPU also suffer from this bottleneck, since each DDR2/3/4/5 SDRAM channel could only response to limited request at the same time. Therefore theoretical bandwidth does not yield high random access bandwidth in such extreme applications. Now we present a novel architecture in follow sections that would override the restriction of random memory access bandwidth while still remaining devoid of data congestion between sub-processors.

4 CGRA for Biological Sequence Analysis

Based on analysis introduced in Sect. 3, a coarse grained reconfigurable array is proposed in this section that would provide impressive performance boost while retain flexibility on accelerating most subfields of biological sequence analysis.

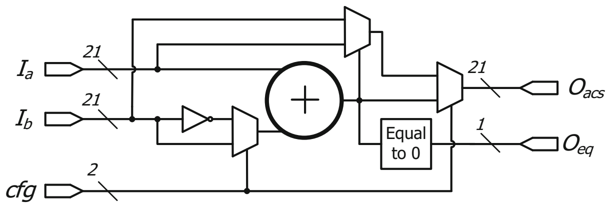
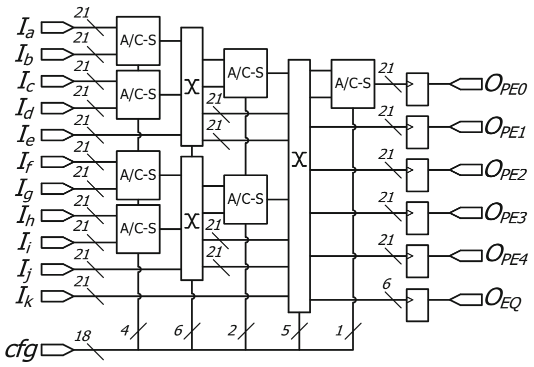


Fig. 9 Reconfigurable Add/Compare-Select (A/C-S) block

Fig. 10 Reconfigurable processing element



4.1 Reconfigurable Processing Element

Analysis of the computational needs in both operation and data-path reveals that those target applications involves elementary mathematical operations mainly in additions and comparisons. Based on this observation, we propose a reconfigurable two-input function block, shown in Fig. 9 as the basic building block of *processing element* (PE).

Similar to the well-known *add-compare-select* (ACS) unit for *viterbi* decoder, this building block also incorporate with *add*, *compare* and *select* functions while *add* and *compare* are composed together, formed as an ACS unit. It has two working modes for either signed addition or comparison operation, along with a constant output for equivalence flag.

By composition along with crossbars, 7 basic building blocks in 3 stages forms a PE as Fig. 10, which can be reconfigured:

- To address the varying additions and comparisons of the distance score matrix filling stage for pairwise sequence alignment algorithms.
- To address the varying comparisons of the backtracking stage for pairwise sequence alignment algorithms.
- To address the varying comparisons of the *seed* matching and *seed* extension stages for database searching.
- To address the varying comparisons of various string matching stages for short read sequence mapping applications.
- To be deployed as an array of arbitrary size to massively parallelize the computations.

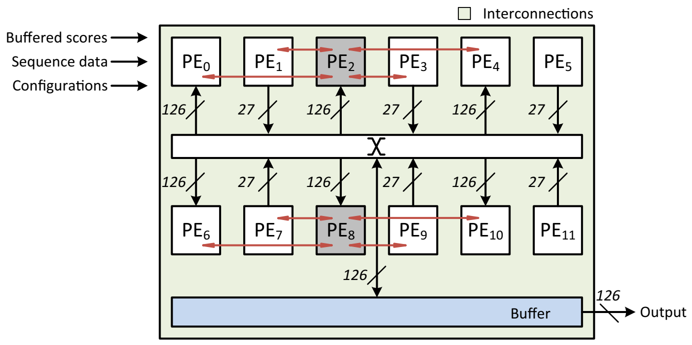


Fig. 11 Reconfigurable PE group with nearest neighbor interconnections and dedicated buffer

In Fig. 10, the ACS units and crossbars can be either statically configured or dynamically switched due to applications requirements. One level of register files are placed at the outputs with 21 bits for general ports and 6 bits for equivalence flags.

4.2 CGRA Interconnection

In order to take efficiency use of 128 bits width DRAM interface and reduce interconnection complexity, 12 PEs are grouped together along with a shared buffer block as Fig. 11. Within the same group, PEs within the reconfigurable group are interconnected with their nearest neighbor by leftwards and rightwards with distance one and two. Note that PE_5 and PE_6 are close neighbor, and the distance in Fig. 11 does not represent the interconnection distance.

The proposed CGRA is composed of an array of identical reconfigurable PE groups. Reconfigured in different working mode, the PE groups can be replicated multiple times to fulfill the requirements of target algorithms and applications introduced in Sect. 3. This is accomplished by an optimized MATRIX architecture [48].

The main idea of MATRIX architecture is to build an adaptable system which could reallocate general computing resources to meet application requirements. By analyzing the characteristics of biological sequence analysis applications, we are able to optimize the MATRIX architecture with a shallow pipeline and compact interconnection.

The routing architecture of our CGRA depicted as Fig. 12 is a hierarchical network with two levels, consisting different coverage type of interconnection. The levels are nearest neighbor connections and global lines. Despite the interconnection inside a PE group, the nearest neighbor connections link the PEs located at group edge to all its neighbors within distance two. Meanwhile, the global lines span the entire row of PE groups as shared inputs and outputs from/to sequencer.

4.3 Input and Output Buffers

There are various type of shared inputs among PEs during biological sequence analysis, such as *substitution matrices* for sequence alignments and *suffixes* of hashed *seed* strings for hash-index based database searching and short read sequence mapping.

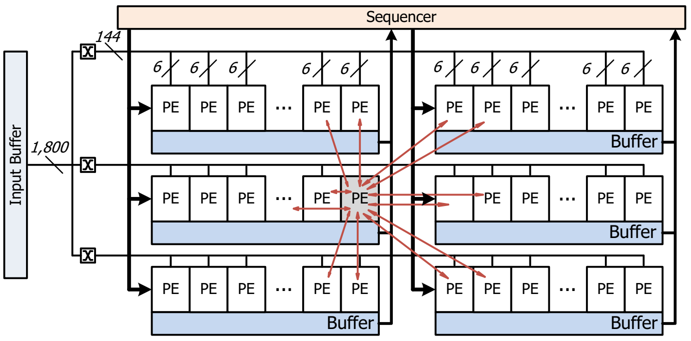


Fig. 12 CGRA with 6 PE groups and optimized nearest neighbor interconnections by group edge

Considering the maximum bit-width requirement of shared inputs, *substitution matrix* for *protein* sequence analysis: there are 24 *amino acids* represented by character symbols, thus a pre-computed score matrix with 24×24 elements is required for each step of computation. By observing the data structure of the *substitution matrices*, all of them consists of data elements within a value range between -32 and 32, and all of those 24×24 matrices are formed in symmetric.

Therefore the maximum shared data input among PEs are $(24 + 1) \times 24/2 = 300$ scores, each with signed 6 bits, thus a specialized interconnection for shared buffer input is shown in Fig. 12. 1800 bits shared input representing the whole *substitution matrix* lies outside CGRA, while the input *amino acid* string at each PE would retrieve the relative matrix scores through a 300–12 crossbar as shared inputs of corresponding PE groups ($12 \times 6 = 72$ bits in total), then the dynamically inputted *amino acid* symbols at each PE would pick the exact score for computation.

This static shared buffer distribution method is also part of the second level of hierarchical network in CGRA, which improves both timing and silicon efficiency. Similar cases such as *substitution matrices* for DNA or *suffixes* of hashed *seed* strings for hash-index based database searching and short read sequence mapping could take use of the same buffer block and interconnections but omitted in detailed description, since the overall.

Figure 11 also demonstrate private buffer block for a PE group. The buffer block provides two 126 bits I/O ports, so that PE number 0, 2, 4, 6, 8, and 10 could read from the buffer at once from crossbar, or a combination of 126 bits can be written from the outputs of PE number 1, 3, 5, 7, and 9 that each with 21 bits.

In case of distance score matrix filling stage in pairwise sequence alignment, it would generate a 21 bits distance score in each clock cycle for each segment pair. But the output of each PE to crossbar is 22 bits, which includes 1 bit that indicate the equivalence test results of the A/C-S units. The required buffer depth will be discussed in Sect. 6.

5 3D-Stacked Many-Core Architecture

Most of the modern parallel processors are many-core based designs; nearly all of them suffer from the high cost of on-chip communication that dominates the sili-

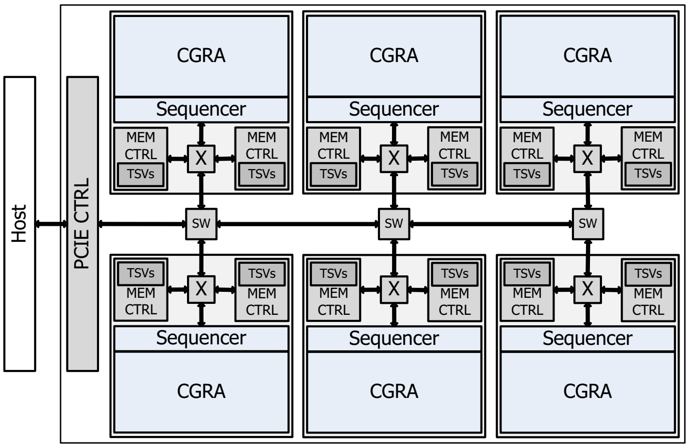


Fig. 13 Reconfigurable many-core processor

con area, power, and performance budgets. Benefit from word-sized computation and reconfigurable interconnection, CGRAs as spatial processors could eliminate some performance gap over interconnection, while the mismatch between performance density and available throughput still exists. Our architecture provides a solution for biological sequence analysis fields that could devoid of such gaps.

5.1 Sub-Processor with Integrated Memory Interface

The CGRA structure introduced in Fig. 12 scales very well in terms of array, while it's constrained by the external data throughput. For distance score matrix filling, classical approach demands high throughput over writing to external memory. For hash matching and string matching stages, large tables are randomly and frequently accessed simultaneously. By exploiting the behavior of those stages, we propose a sub-processor that integrates two DRAM controllers, which would benefit from well-balanced throughput versus performance density.

As shown in Fig. 13, each sub-processor is equipped with two DRAM controllers that interface with dedicated external DRAM channels respectively, where local accesses to DRAMs are routed to an integrated sequencer, and remote accesses are routed to circuit-switching bus. The system host writes data to those DRAMs and programs the sequencer on data preparation stage, then the sequencer takes control of the CGRA as well as DRAM controllers to accelerate specific processing stages of target applications. The generated results are stored in DRAMs and then read back by host after processing finished. Detailed application stage mappings are demonstrated in Sect. 6.

5.2 Circuit-Switching Many-Core Processor

Although packet-switching based *Network-on-Chip* (NoC) architecture dominates many-core designs, our sub-processors are interconnected alternatively using circuit-switching. The many-core architecture in 2D mesh is demonstrated as a 3 × 2 array in Fig. 13, where sub-processors are linked with circuit-switching bus and communicating with system host through PCIE interface.

By exploiting the data flow of target biological sequence analysis problems, we do not consider the remote data accesses between local DRAMs and neighborhood sub-processors. Remote accesses to DRAMs in a sub-processor are only requested from system host during data preparation and finishing stages, one sub-processor after another. There is also no interactive communication between sub-processors during computation. Therefore we are able to take advantage of low complexity and high bandwidth with circuit-switching bus without considering the coherence and congestion issues.

5.3 3D-Stacked DRAM with Scatter/Gather Read

According to JEDEC compliant DRAM model, a *read* request is executed in 3 phases:

- The target *row* and *bank* is activated during the *Activate* phase
- The *column* address is selected while a *Read* command is sent to select specific *column* of the entire *row,* and a latency of several clock cycles is required before the data reaches the DQ port.
- The *precharge* phase is needed only if the current *row* must be deactivated; as the *precharge* command recharge the consumed energy in this *row*, this bank is ready for a new *read* or *write* request on any *row*.

For 3D compliant DRAM model proposed by Loi and Benini [49], *Activate* and *Precharge* phases show the same latency as JEDEC since they rely on the physical interconnections, while the *Read/Write* phase is shorter due to improvement on the I/O interface and lean column decoder architecture. The protocol translation between processor and DRAM is still needed but the access time is much less than classical approaches. Our design follows this model with further optimizations.

Our many-core architecture works in classical ASIC design space, but its scalability is strictly limited by the physical availability of DRAM channels from PCB layout. If the accelerator system equipped with 4 DRAM channels per processor chip, similar to the existed commercial products, it would have only two sub-processors working together. 3D-stacking eliminates this limitations and we can enhance memory parallelism with novel memory architecture. An overview of the 3D-stacked technologies to be adopted in our architecture is illustrated as Fig. 14.

The cache line in general purpose processor helps reducing DRAM access latency by scheduled prefetching. It's not practical to adopt the cache line in our accelerator,

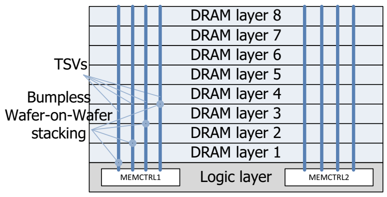


Fig. 14 3D-stacked DRAM layers above sub-processor

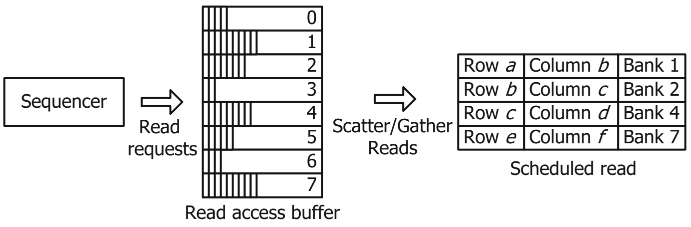

Sequencer
Read requests
0
1
2
3
4
5
6
7
Read access buffer
Scatter/Gather Reads
Row a | Column b | Bank 1
Row b | Column c | Bank 2
Row c | Column d | Bank 4
Row e | Column f | Bank 7
Scheduled read

Fig. 15 Scatter/Gather read policy in 3D-stacked DRAM

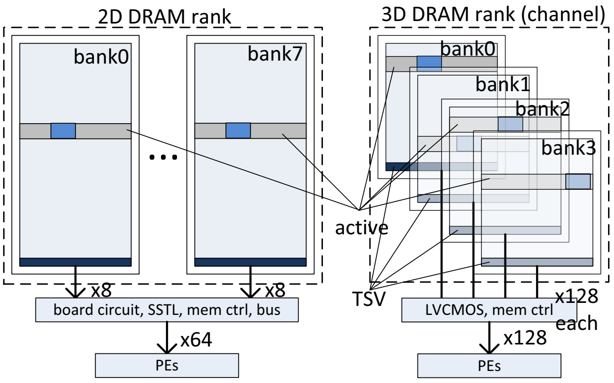


Fig. 16 Read accesses over typical 2D DRAM and our 3D-stacked DRAM

but it’s still possible to benefit from scheduled memory access due to the extreme low latency when accessing stacked DRAM layers. With a random read scheduler implemented in the memory controller, a *Scatter/Gather* read policy has been adopted in each DRAM channel as Fig. 15, thus the DRAM read is optimized for random or strided accesses in 128 bit, instead of the classical cache line access policy.

This customized policy allows PEs to directly address individual words in DRAM, which dramatically increased effective memory bandwidth of random read accesses. A comparison between random read accesses over classical 2D DRAM and our 3D-stacked SDRAM is shown as Fig. 16.

Figure 16 demonstrates the differences of random read access between classical 2D DRAM and our 3D DRAM, where the small blocks of active rows indicate data read. It depicted 3D-stacked DRAM with 4 banks for theory demonstration, while our designated architecture has 8 banks for each DRAM channel. We assume that all banks in a DRAM channel share a common 128-bit wide TSV data bus, a single channel is achieved by 8 banks in 8 layers, each bank with a size of 64 Mb.

For 2D DRAM rank, the data is distributed over all banks, and each access targets a small fraction of the same active address to all banks. A *Scatter/Gather* read policy in our 3D-stacked architecture allows accessing different banks at the same time, while the target column address can be different between each bank. Although data from different banks can’t be transferred altogether, the I/O width to each bank is exact the same as the channel width, therefore random read from different banks can be sequentially accomplished in continuous clock cycle, so that achieving much higher bandwidth efficiency.

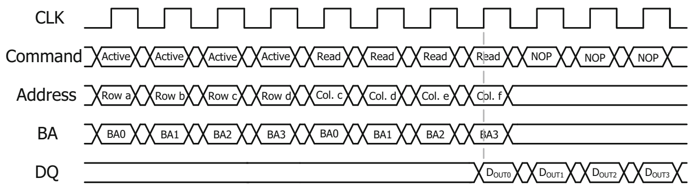


Fig. 17 Random read with burst length 1 in 3D-stacked SDRAM

Data-path latency is another factor that affects DRAM random access, as cache line won't be of much help in this case. Signal from classical 2D DRAM has to be transferred through PCB, SSTL pads, memory controller, and system bus before reaching PE, while in our 3D architecture it becomes much simpler and more efficient. We don't even have to consider the system bus latency in general 3D designs, as remote access to local DRAM channels is avoided by careful dataset partitioning and task scheduling.

For external DRAM channel with 8 banks, considering the most extreme random read case, 4 banks are selected and opened for reading the least data and then closed, the command order from memory controller ignoring NOP will be: Active0, Active1, Active2, Active3, Read0, Read1, Read2, Read3, Precharge0, Precharge1, Precharge2, Precharge3. Typical DDR3-1600 DRAM with customized memory controller adopting Scatter/Gather read policy would take a minimum of 28 clock cycles to execute this command string, which is 35 ns in total. The working mode can be BC4 or BL8, thus the data read from each bank can be $64 \times 4 = 256$ bits or $64 \times 8 = 512$ bits respectively.

According to SDRAM specification, data from a fixed-length *Read* burst can be followed immediately by data from another *Read* burst. Therefore our designated *Scatter/Gather* policy for random read access to 3D-stacked SDRAM banks can be demonstrated as Fig. 17, CL = 3.

After initial Active of four selected banks, read access with burst length 1 is issued to each open bank, and the DRAM output form is small data fragments of 128 bits from every open bank transferred in 4 continuous clock cycles. Assume the 3D-stacked SDRAM running at 333 MHz adopting the Scatter/Gather read policy, it would take 12 clock cycles, which is 36 ns in total to execute the same command string.

According to targeting biological sequence applications detailed in Sects. 3 and 6, random read from external DRAM is the most critical bottleneck, and 128 bit is the efficient requirement to the size of data over each read access. Therefore it can be concluded that DDR3-1600 and SDR-333 gives almost the identical performance over random read with customized memory controller. It must be figured out as well that 64 banks are required to achieve such performance in case of DDR3-1600. Considering energy efficiency and silicon efficiency between DDR3-1600 and SDR-333 in conjunction, we prefer using the latter in our 3D-stacked architecture.

Wide I/O DRAM has been standardized by JEDEC (JESD229) [50] as a 3D DRAM solution for embedded SoC systems with lots of low power consideration. In order to conform to the WIDE I/O standard, we use LVCMOS signaling for interconnection so that we could devoid of using complex SSTL interface, results in both silicon and energy efficiency. We adopt the *wafer-to-wafer* bumpless stacking process which provides aggressively small TSVs at an extremely fine pitch (5 μm). This approach

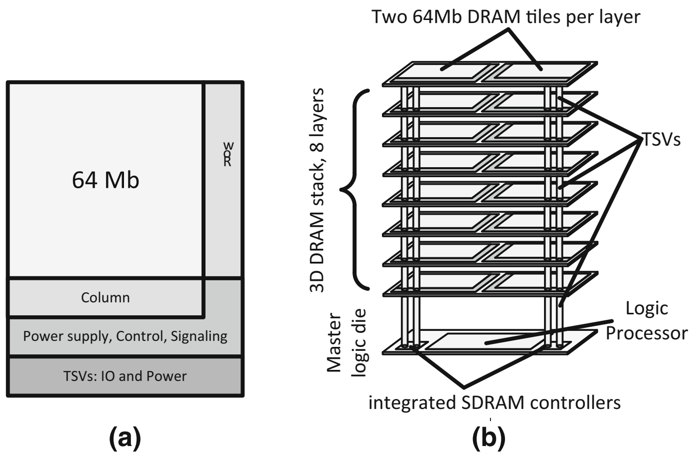


Fig. 18 **a** 3D DRAM core tile of 64 Mb, and **b** 3D-stacked architecture of single processing cube

provides much better silicon efficiency than microbump based die-to-die stacking with large pitch (20–40 μm).

Figure 18a shows the layout structure of a single DRAM core tile for 3D architecture, which forms the basis to compose 3D DRAM layers. The tile size is 64 Mb, and we assume 128 I/Os per tile (IOPT) and a page size of 1 kB. Figure 18b depicts a high-level view of our 3D integrated architecture over single stacked tile. Note that the two stack of DRAM memory tiles are accessed using an alternative DRAM interfacing protocol respectively, paired with dedicated memory controller.

Our design does not rely on NoC since all the mapped algorithm and application stages takes use of local data only. Although our design is many-core architecture on the horizontal logic plane, there is no remote access between local DRAMs and neighborhood processing cores. Each processing core works independently with two integrated memory controllers exclusively, which interfaces with a 3D-stacked DRAM of 64 MB in size respectively. This approach provides an extremely high bandwidth for fast local access without consideration of remote access during computation. In case of write or read from host processor, each local processing core will be accessed in a serialized queue, thus no congestion shall be considered on shared bus.

6 Application Mappings

Considering that all the target biological sequence analysis applications are composed of a few characterizable computing stages, the application mapping is introduced by individual kernel stage mappings over our architecture.

6.1 Distance Score Matrix Filling

In order to accelerate the distance score matrix filling stage in pairwise sequence alignment, the *wave-front* [51] method is a promising solution which has been adopted

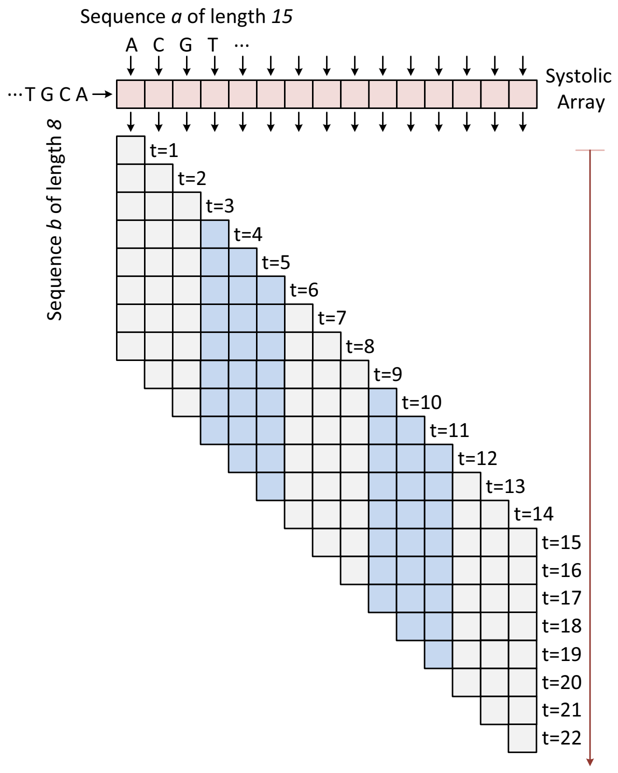


Fig. 19 Wave-front method for distance score matrix filling in parallel

in most hardware accelerators. The distance score matrix filling is defined to be the path from the top-left corner to the bottom-right corner of the matrix. The *wave-front* method reduces the time complexity of complete filling from $O(m \times n)$–$O(m \times n/p)$, where m and n is the length of the sequence pair to be aligned, p is the number of processing elements in parallel.

The *wave-front* method can be achieved by using a *systolic array* [52] with m depth of processing units, example of which is shown as Fig. 19.

Comparing with sequential implementation as Fig. 1, which would take at least $15 \times 8 = 120$ clock cycles for processing sequence pair with length 15 and 8 symbols respectively, the *wave-front* approach takes only 22 clock cycles. Mapping *wave-front* method to our accelerator for distance score matrix filling stage in *Needleman–Wunsch* and *Smith–Waterman* algorithms are shown as Figs. 20 and 21 respectively, both with *affine gap*. The unused logics are grayed out, and the configuration logics have been omitted since no dynamic reconfiguration is required during computation. Note that extra logics are added in mapped PEs for *Smith–Waterman* algorithm to give $Fmax_{i,j}$, which will be used in the backtracking stage.

The input ports labeled with 21 bits width are the efficient input width for each port. Although both *Needleman–Wunsch* and *Smith–Waterman* algorithms can be used for

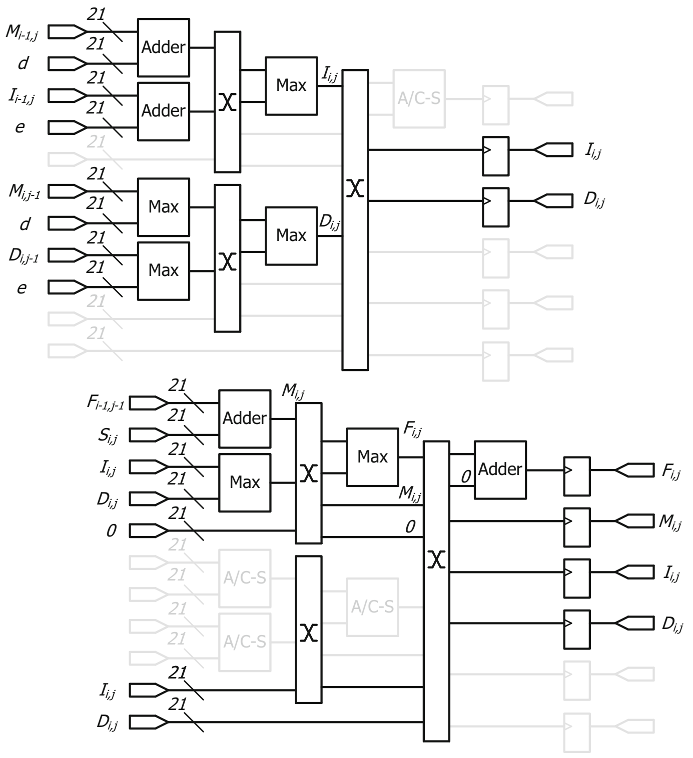


Fig. 20 Distance score matrix filling stage of Needleman–Wunsch algorithm mapped to two PEs (Affine gap)

protein sequences, the data structure of computation is not influenced by the type of sequence except the *scoring matrix*.

Adopting *systolic array* with m processing units requires $2m$ of PEs in our architecture at this stage. If m is less than PE group buffer depth d, n instances of *systolic array* can be mapped to our CGRA as Fig. 22a. Otherwise multiple *systolic arrays* can be composed to give buffer depth of $2d$ for alignment of long sequences as shown in Fig. 22b, while the effective number of PEs are halved in each PE group.

The initial biological sequence input can be encoded using 2 bits to represent DNA or RNA sequence symbols, or 5 bits for protein symbols. Assume m to be the array depth of our CGRA, which is the number of PE groups located at the first column of CGRA, and PE groups at the same column would share the same input from the sequencer block, then the effective CGRA width n can be theoretically given by (6) or (7) in case of DNA/RNA or protein sequences, respectively:

$$n = \frac{B_{eff\_in}}{f_{CGRA} \times m \times 2} \tag{6}$$

$$n = \frac{B_{eff\_in}}{f_{CGRA} \times m \times 5} \tag{7}$$

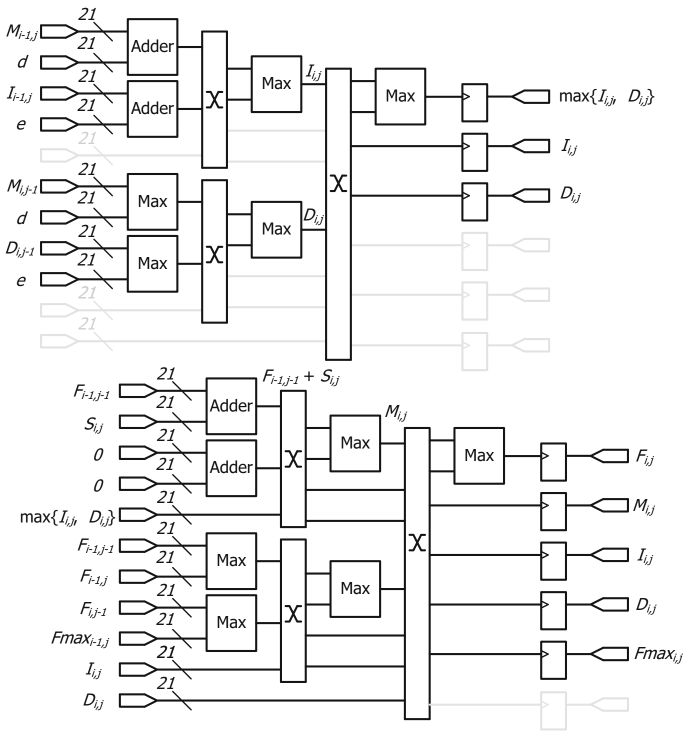


Fig. 21 Distance score matrix filling stage of Smith–Waterman algorithm mapped to two PEs (Affine gap)

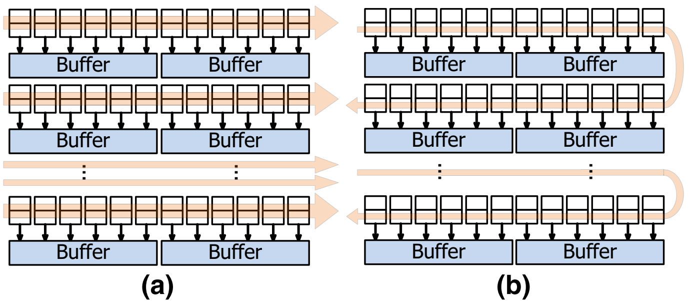


Fig. 22 Wave-front method for distance score matrix filling in parallel

where B_{eff_in} is the effective external dynamic input bandwidth, and f_{CGRA} is the clock frequency of CGRA. PEs in deeper stages of PE group could receive data spread by former stages, while the output of each PE group is stored in buffer to fill the distance score matrix, which would produce w bits of data to be stored in each computation cycle, where w is 21 denotes the output bit-width of a PE. In this case, the theoretical output bandwidth of our CGRA B_{CGRA_out} without buffer is given by (8):

$$B_{CGRA\_out} = m \times n \times f_{CGRA} \times 21 \tag{8}$$

Assume $B_{eff\_in} = B_{eff\_out}$, which is roughly the real case of sequential operations over external DRAM subsystem shown in Fig. 8, then:

$$B_{CGRA\_itout} = B_{eff\_out} \times 21/2 \quad \text{(DNA/RNA)} \tag{9}$$
$$B_{CGRA\_out} = B_{eff\_out} \times 21/5 \quad \text{(Protein)} \tag{10}$$

From (9) and (10), we can observe over 10 times mismatch for CGRA output bandwidth in case of DNA/RNA sequence, and over 4 times mismatch in case of protein sequence. This mismatch made classical approaches of distance score matrix filling benefit less than anticipated on massively paralleled accelerators if the distance score matrixes have to be transferred to external DRAM. This issue will be further discussed in next subsection.

6.2 Backtracking

During the backtracking stage of pairwise sequence alignment, the saved distance score matrix is evaluated to give the optimal alignment score. Assume the matrix is stored in external DRAM and evaluation can be assigned to individual PEs in parallel, the maximum number of active PEs $n_{backtrack}$ is estimated by (11), where another bandwidth mismatch exists as $n_{backtrack} < 4$ based on estimation of commodity ASIC performances.

$$n_{backtrack} = \frac{B_{eff\_in}}{f_{CGRA} \times 21} \tag{11}$$

In order to reduce the performance loss caused by the bandwidth mismatches, internal buffer with reasonable depth is coupled with PE group in our CGRA. In case of sequence pairs with length k and l, both $k, l \leqslant m$, then the buffer instances with depth m could store the complete distance scores produced by corresponding PE group.

In case of *Needleman–Wunsch global* sequence alignment, the distance score matrix is evaluated right after generation, and the backtracking chain is given sequentially in every clock cycle as (3) and Fig. 23. For sequence pairs with length k and l, both $k, l \leqslant m$, the footprint of reading buffered distance scores is separately colored for different PE groups. With the help of local buffer, it takes 20 clock cycles to finish the backtracking in Fig. 23, which is much faster than retrieving distance score matrix from external DRAM.

For *Smith–Waterman local* sequence alignment mapped to our architecture, the value of global maximum distance score F_{max} has been recorded after distance score matrix filling stage, which has been described in Fig. 19. Therefore the backtracking stage of *Smith–Waterman local* sequence alignment can be partial accelerated as Fig. 24a.

After distance score matrix filling, the individual PE groups within the same *systolic array* could compare distance scores stored in local buffer against F_{max} in parallel. When the corresponding matrix element with global maximum score value has been

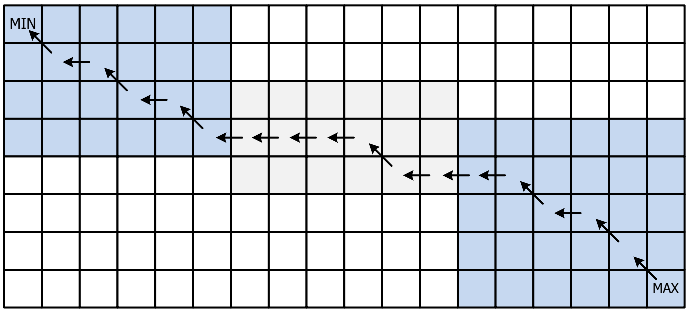


Fig. 23 Mapped backtracking stage of Needleman–Wunsch global sequence alignment using local buffer

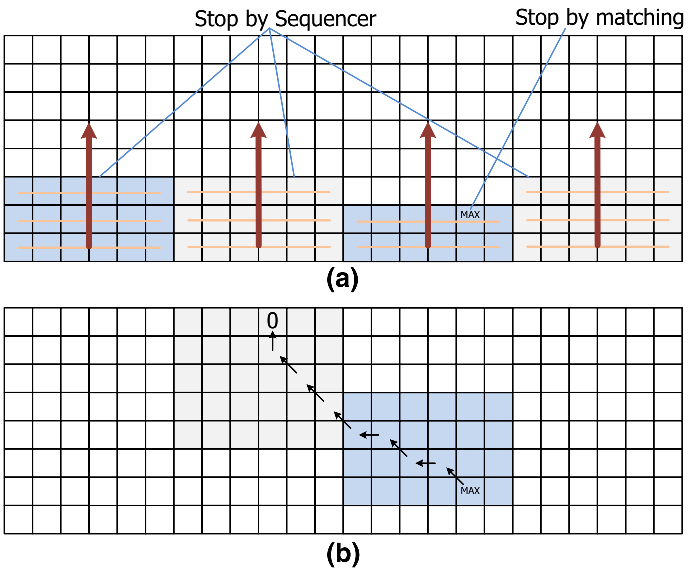
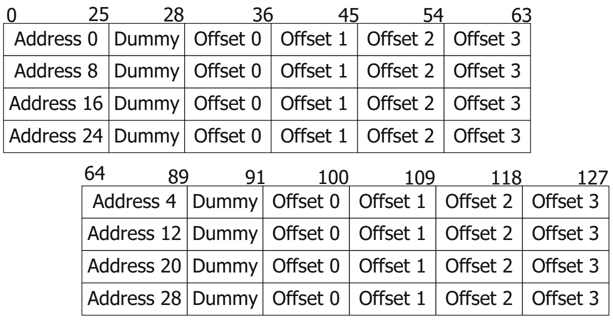


Fig. 24 Mapped backtracking stage Smith–Waterman local sequence alignment using local buffer

found, parallel comparisons are stopped by sequencer, and then backtracking starts at the element with global maximum value. Sequentially tracking back using (3), the process would stop until "0", demonstrated as Fig. 24b.

For long sequence pairs, the size of distance score matrix would increase by 2 orders of magnitude, so that $2m \times 4 = 8$m PEs and corresponding buffer instances would fit for k and $l \leq 2m$. In this case, the distance score matrix must be filled in sub-blocks, and the backtrack order of a sub-block is sequentially from bottom-right, while different sub-blocks are still evaluated in parallel.

From exploration of real experiments we found that the most common usage of pairwise sequence alignment is to align sequence strings with length less than 200, so that we set $m = 192$ in our current design. Thus our architecture has the ability to massively accelerate pairwise sequence alignment with maximum length of 384


0 25 28 36 45 54 63
64 89 91 100 109 118 127

Address 0	Dummy	Offset 0	Offset 1	Offset 2	Offset 3
Address 8	Dummy	Offset 0	Offset 1	Offset 2	Offset 3
Address 16	Dummy	Offset 0	Offset 1	Offset 2	Offset 3
Address 24	Dummy	Offset 0	Offset 1	Offset 2	Offset 3

Address 4	Dummy	Offset 0	Offset 1	Offset 2	Offset 3
Address 12	Dummy	Offset 0	Offset 1	Offset 2	Offset 3
Address 20	Dummy	Offset 0	Offset 1	Offset 2	Offset 3
Address 28	Dummy	Offset 0	Offset 1	Offset 2	Offset 3

Fig. 25 Offset data structure of pointer table for seed matching of BLAST

for both stages without output to external DRAM. For even longer sequence pairs, external DRAMs have to be used for temporary buffer.

6.3 Seed Matchings and Seed extension

The original *seed matching* stage introduced in Sect. 3 is not suitable for parallelization on many-core accelerator, since its process is almost sequential and the function blocks are rather large in terms of hardware logic. As the final goal of seed matching is to find out the exact *seed-location* matching pairs, we turn this problem into an equivalent solution using indexed table searching that can be executed in parallel:

- During data preparation, the original database is hashed according to hash functions and indexed to hash table. A pointer table is created using all the possible seeds as access entry, which stores the accessing range of the database hash table that corresponding to each seed.
- The *seeds* are used to access the pointer table, and a valid pointer table value indicates a series of exact *seed-location* matching pairs in the original database.

Considering the size of the pointer table, the length of *seed* for DNA/RNA sequence type is 10, and that of protein sequence is 3 or 4. Therefore the theoretical address range of pointer table can be 20 or 15 bits, thus it can be only stored in external DRAM.

The channel width of our proposed 3D-stacked DRAM is 128 bits, which is too big for single record of addressing to the hash table. We adopt a offset data structure as Fig. 25.

In order to access the hash index table, each record in pointer table gives an address entry following with four offset values. The offset values indicates the access range of hash index table records for current *seed*, and also can be used to give the access address for next *seeds*, though some computation is required:

$$Address_{n+1} = Address_n + offset_0 \quad (12)$$

$$Address_{n+2} = Address_n + offset_0 + offset_1 \quad (13)$$

$$Address_{n+3} = Address_n + offset_0 + offset_1 + offset_2 \quad (14)$$

After finding out all the exact *seed-location* matching pairs, the original software implementation of BLAST would execute a *ungapped seed extension* stage to find high score pairs (HSP). Starting by stepwise extending the *seed-location* matching from both end and evaluating the distance score, if two extended seeds are found nearby, a *gapped extension* stage will be executed to give exact local sequence alignment.

In case of our massively paralleled architecture, mapping *ungapped extension* stage is almost the same as mapping distance score matrix filling of local sequence alignment. The only difference from Fig. 21 is that F_{max} shall be compared with current $F_{i,j}$. If $F_{i,j} < F_{max}$, extension will be stopped and F_{max} will be compared with a threshold value $F_{threshold}$. Distances within the original database between all the extended *seeds* that with $F_{max} > F_{threshold}$ will be pairwise evaluated. Two or more extended *seeds* found close by indicates a HSP.

All the HSPs will be evaluated with *gapped extension*, which is an alternative version of pairwise *local* sequence alignment, called "*banded local* sequence alignment". It adopt a premise that the weak alignment results caused by long insertions or deletions shall be abandoned in final result, thus the alignment domain can be restricted within a narrow band along the diagonal curve of distance score matrix [53].

This approach highly reduced resource requirement on matrix filling stage. Assume a small band of width d, the valid distance score matrix is reduced from $m \times n$ to $(m^2 + n^2)^{1/2} \times d$. d can be narrowed to 3 or 2 in extreme case, as only 1 mismatch or exact match is allowed over alignment region, so that backtracking after matrix filling can be omitted due to existed unique score chain. An example of $d = 7$ is shown as Fig. 26a.

The PE Mapping of *banded local* sequence alignment is identical to Fig. 19 except the PEs on band edge, which has fixed "0" inputs for F, M, I, D values from the elements outside the banded area. Adopting *banded local* sequence alignment would filter out those alignments which exceed the banded area during computation as Fig. 26b, so that the sequence pair under process will be abandoned for any further process-

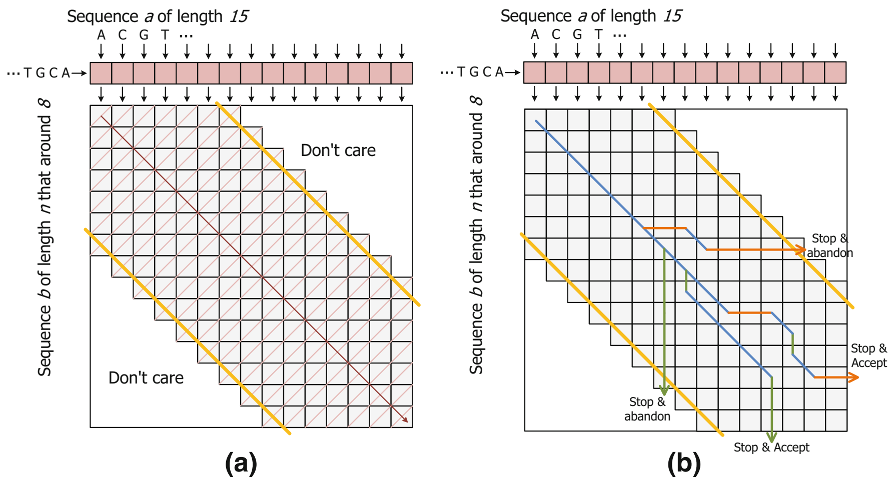


Fig. 26 Banded Smith–Waterman local sequence alignment

ing. Backtracking stage will be executed only for the sequence pairs that passed banded distance score matrix filling stage. This policy would save both computation resources and runtime comparing with the typical solution introduced in previous subsections.

6.4 2-Hit String Matching

The *2-hit string matching* stage in hash-index based short read sequence mapping shares the very similar idea to the *seed matching* stage in database searching: a valid hit to pointer table gives an access range to the corresponding hash table.

During *seed matching* stage, valid access to pointer table using a *seed* would give an address range of the records in the large hash table that contains *candidate matching locations*. These locations will be further verified with *ungapped extension*.

For 2-hit string matching, valid access to pointer table using *prefix* of a *seed* would give an address range of the records in the large hash table that contains *hash-location* pairs. The hash part of which will be compared with *suffix* of the *seed* for verification.

Despite using *distance score matrix filling* for *ungapped extension*, comparison between hashed *seed suffix* and hash value bucket can be easily mapped to PE as Fig. 27.

For short read sequence mapping, the number of isolated *reads* to be processed is tens of millions or even more in practical, in order to ensure enough redundancy to cover everywhere of the target sequences. Using head part of each *read* as a *seed*, the hashed *seed* prefix is isolated as well, and the input order of which is also unpredictable. Therefore accessing to pointer table is completely random read to DRAM and the payload is relatively small, which could benefit from the *Scatter/Gather* read policy.

Initial string matching is direct equivalence test between two values, thus string matchings performance is limited by I/O throughput. The hash table accessing in *2-hit string matching* retrieves *hash-location* pairs that each consists of 38–42 bits according

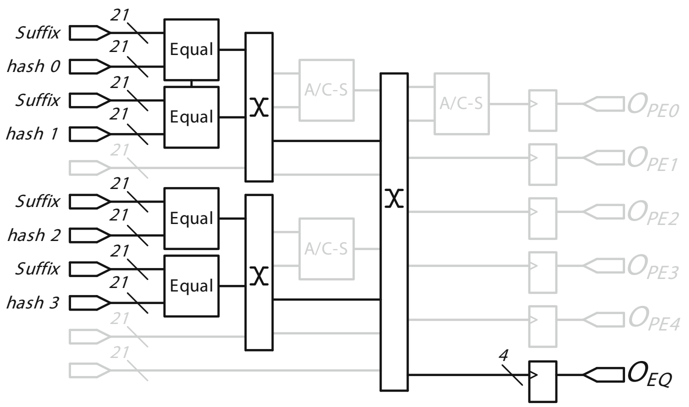


Fig. 27 Mapped PE for 2-hit string matching

to different *seed* length, and the number of *hash-location* pairs in each bucket could be any value between 1 and 12. Therefore reading from hash table could also benefit from Scatter/Gather read policy, since the burst read length for each access can be 1, 2, or 4 only.

6.5 Backward Searching

The FM-index for short read sequence mapping is a data structure which contains the BWT of the reference genome and corresponding *suffix array* checkpoints. Backward searching against FM-index would find out whether the read occurs in the reference genome and the corresponding location(s). Accelerating backward searching on FPGA has been proposed by Fernandez et al. [20] using Convey HC-1 to process the whole short read. We follow the similar idea over hardware design, but also incorporate the method used in BWA-SW [36] that searching against FM-index would find out whether the head part of a *read*, called *seed*, occurs in the reference genome and the corresponding location(s).

The Burrows–Wheeler transform of the reference genome is represented as two tables, called C-table and I-table. The I-table is an array with four entries that stores the position of the first occurrence of each nucleotide symbol in the reference genome after the reference genome has been sorted lexicographically. The C-table is a two dimensional matrix, with a number of rows equal to the length of the reference genome, and a number of columns equal to four as nucleotide symbol. Assume BTW(*ref*) to be the Burrow–Wheeler transform of the reference genome, entry i, j of the C-Table represents the number of occurrences of nucleotide symbol j in the prefix of BWT(*ref*) of length i.

During backward searching, the *seed* is gradually matched with the reference genome for every single nucleotide symbol s within the *seed* by accessing the FM-index using two pointers, called *top* and *bottom*, which are defined by (15) and (16):

$$top_{new} = C[top_{current}, s] + I[s] \tag{15}$$

$$bottom_{new} = C[bottom_{current}, s] + I[s] \tag{16}$$

These two pointers are updated at each processed character of the *seed*. If at any one time $top_{new} \leq bottom_{new}$, the search is terminated and the corresponding *read* is reported unmapped. Otherwise if the last character in the *seed* is reached, the range between current *top* and *bottom* indicates the number of occurrences of the *seed* in the reference genome. PE Mapping of backward searching is shown as Fig. 28. (15) and (16) have been implemented using two adders, while whether $top_{new} \leq bottom_{new}$ is also verified.

In case of inexact searching which is the realistic case, a limited number of mismatch events might be occurred during backward searching, which leads to branched executions of exact searching with insertions or deletion at the mismatched location. Our massively paralleled architecture could handle this problem by allocating free PEs with dynamic reconfiguration for branched computation as Fig. 29. The corresponding

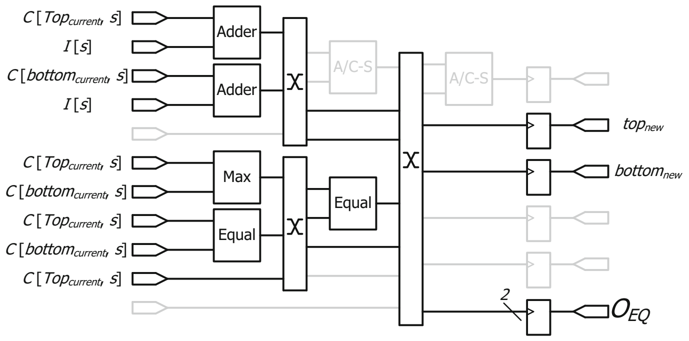
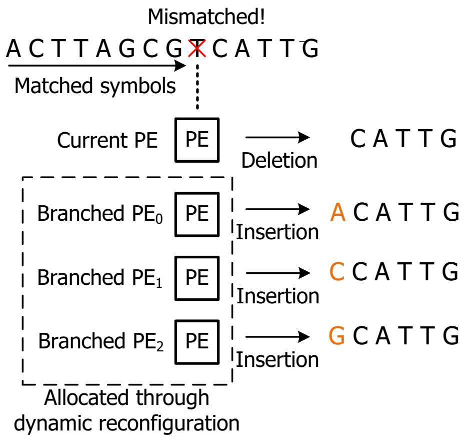


Fig. 28 Mapped PE for backward searching

Fig. 29 Mapped inexact backward searching



top and *bottom* pointers within each branch will be refreshed respectively according to the new forms of the *seed*.

The number of allowed mismatch can be defined by user, while 2 or 3 is common used. Larger allowed mismatch values would greatly impact the overall performance due to much more DRAM accesses. The mismatched location is not required to be specifically stored, since the inexact backward searching leads to reference genome locations that should be further verified by banded local sequence alignment.

After backward searching without termination, the values of *top* and *bottom* pointer as well as those between them will be used as addresses to access the *suffix array* stored in external DRAM to retrieve the corresponding matched seed location(s). Banded *local* sequence alignment, or so called *gapped extension*, will be executed for each *read* at each matched location between the *seed* and the reference genome.

During the whole backward matching stage, accessing C-table is completely random by each PE, thus the Scatter/Gather read policy would be of great help on increasing the effective read bandwidth.

6.6 Further Improvements

For seed matching, 2-hit string matching, and backward searching stages, the utilization ratio of CGRA is extremely low due to bandwidth mismatch, thus not many PEs are working actively. In order to make efficiency use of resources, the following pairwise local sequence alignment stages can be executed in parallel along with the matching stages. Since our architecture has two DRAM channels for each sub-processor, genome sequences required for pairwise sequence alignment can be stored in DRAM channel that different from the large tables, thus both the tables and corresponding genome sequences can be accessed simultaneously. The overlapped run time between matching stages and pairwise local sequence alignment stages provides another factor of performance improvements.

7 Experimental Results

In this section, we evaluate the performance and energy efficiency of our architecture. We don't report individual performance for pairwise sequence alignment since it has been integrated as important component in all the other applications.

7.1 Test Environments

Without loss of generality, we use an example design based on 45-nm silicon process. Previous research shown that a typical 3D DRAM core tile of 64 Mb occupies 1.92 mm^2 in 45 nm process node with a maximum frequency of 357 MHz [54]. With two DRAM core tiles of 64 Mb stacked on, we estimate that each silicon layer of our sub-processor occupies 4.2 mm^2 with 2 DRAM channels clocked at 333 MHz, and our 3D-stacked cube consists of 9 silicon layers.

The planned chip area of each layer is approximately 70 mm^2 which consists of $4 \times 4 = 16$ sub-processor cubes. Typical 45-nm process provides a density of 2000 K equivalent gates per 1 mm^2 [55], while our PE cost around 4000 gates. With 4.2 mm^2 of silicon area for logic layer, a sub-processor is synthesized with a clock frequency of 666 MHz and verified with post-synthesize simulation.

From synthesis area report, 72.3% of a sub-processor is occupied by 64 PE groups that each contains 12 PEs and a instances of 24 Kb buffer block, while we estimate that 5.6% is occupied by memory controllers and TSVs. The rest 22.1% area of sub-processor is occupied by shared input buffer, sequencer block and interconnections. The number of PEs in CGRA and buffer size of a sub-processor are well balanced by exploration of mapped biological sequence analysis application stages in Sect. 6, while the consumed silicon area fits well over process shrinking.

The wire model in CACTI-3DD [56] is used to estimate the 3D DRAM bus delay, where 1 mm wire introduces 0.087 and 0.03 ns delay for TSV at 45-nm process node. As the RC delay is proportional to wire length, the longest route from memory controller across 8 layers to the corner of DRAM layer is around 4.9 mm, thus introduces 0.42 ns delay. Comparing with DRAM clock period of 3.3 ns, the signal propagation across 3D layers does not incur extra bus delay in clock cycle.

We evaluated our architecture in classical design domain on terasIC DE5-NET FPGA board, which is equipped with Altera 5SGXEA7N2F45C2N Stratix V device. A stripped down version representing single sub-processor of our architecture has been synthesized to fit the DE5 board, which contains 768 PEs and two DRAM channels as well. Because of insufficient interconnection resources, we can only generate pre-configured netlists that equivalent to each specific working mode of CGRA, corresponding to each target biological sequence analysis application. Those FPGA implementations utilizes around 70% of LUTs, running at a clock frequency of 200 MHz. This category of implementations with two channels of 2 GB DDR3-1600 DRAM is listed as "FPGA" in the following evaluations.

The benchmark platform for software performance is a workstation equipped with Intel Core i7 4960$\times$ 3.60 GHz CPU, 64 GB DDR3-1600 RAM running Redhat Enterprise Linux v5.11 $\times$64 system. Sequential software performances as baseline are reported as "Seq." in all the evaluations, while multithreaded executions with 12 threads are reported as "MT". GPU implementations are evaluated on a nVIDIA Geforce Titan Black device with 980 MHz core frequency and 6 GB onboard DRAM, equipped on the same workstation.

All the source codes are compiled with GCC v4.4.7 and "$-$O3" optimization flag. CUDA Toolkit 6.5 is used to compile all the GPU source codes. Modified sequential software implementations are served as front-end to utilize our FPGA implementation as hardware accelerator. The host sessions for FPGA and 3D implementations are not working during acceleration, but some of the overhead processing in software is required for all the implementations, which is omitted here. For each specific test, all the implementations use the same default options such as gap penalties, gap extensions, and substitution matrices, in order to ensure equivalent complexity and quality of results.

7.2 Performance Scaling

The performances of our 3D-stacked many-core accelerator labeled as "3D" are estimated through proper scaling of FPGA generated results.

We recorded the memory traces through simulation of our original 3D-stacked design with each mapped application stages by a short period. By analysis of those memory traces, corresponding memory access patterns has been extracted.

Then we designed a function block and inserted it between the memory controller and sequencer block at the FPGA implementations, which slow down the DRAM interface to 100 MHz. The available memory size is also restricted by 64 MB for each DRAM channel.

Since the external DRAM is working at 800 MHz, it's easy to reform the memory access traces according to the designated patterns that occurs on 3D design. Therefore we received a sub-processor that has 2:1 clock ratio with external DRAM as well, and the computation traces shall be almost identical to a sub-processor in 3D design.

There is no communication between sub-processors in our 3D design, thus sequentially evaluating each partitioned problem with FPGA would give the isolated performance for each sub-processor running in parallel at working frequency of 200 MHz for core logic and 100 MHz for DRAM. Therefore for 16 partitioned problems that could been solved in parallel, pick the result with highest runtime and divide by 3.33 would give the rough performance estimation of the 3D-stacked accelerator.

This scaling approach does not affect the kernel function performance, and the overhead time consumed by disk I/O, data transaction between host and accelerator devices, data compression, and data decompression are not included in all the results.

7.3 Test Results

Sequential execution of T-Coffee v11 [33] is evaluated as the baseline of MSA. GPU accelerated T-Coffee called G-MSA [23] is evaluated; its performance is reported only for overall execution time since it's designed with conjunction of software multithreading. The dataset for MSA evaluation is the BAliBase v3 [57] benchmark set.

Figure 30 shows the execution times measured in seconds of multiple sequence alignment regarding distance score matrix filling and pairwise local sequence alignment stages. The overall runtime with respect of acceleration ratio to sequential implementation is shown as the secondary vertical axis.

Sequential performance of NCBI BLAST v2.2.8 [34] is the baseline of database searching. For DNA database searching using *blastn*, G-BLASTN [26] is evaluated for GPU, and we use the same datasets as the report of G-BLASTN to query NCBI *env_nt* database released in Oct. 2014. For protein database searching using *blastp*, same status applies to GPU-BLAST [24] and NCBI *env_nr* database.

Figure 31 shows the execution times measured in seconds of database searching using *blastn* and *blastp* of BLAST regarding seed matching and pairwise local sequence alignment stages. The overall runtime with respect of acceleration ratio to sequential implementation of *blastn* and *blastp* are shown as the secondary vertical axis respectively.

Sequential performance of BFAST v0.7.0a [35] is the baseline of hash-index based short read sequence mapping. There is no existed GPU implementation for hash-index based short read sequence mapping. NCBI run SRR867061 is chosen as the

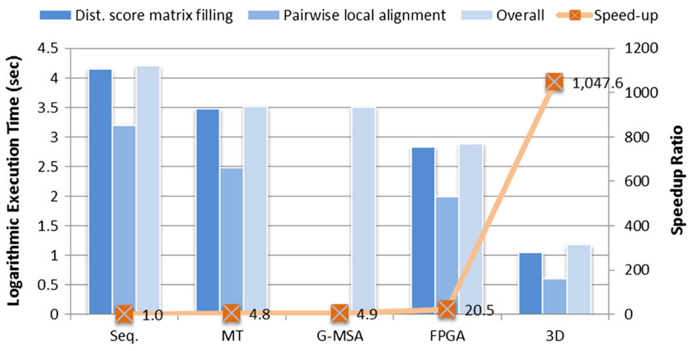


Fig. 30 Performance evaluation of multiple sequence alignment

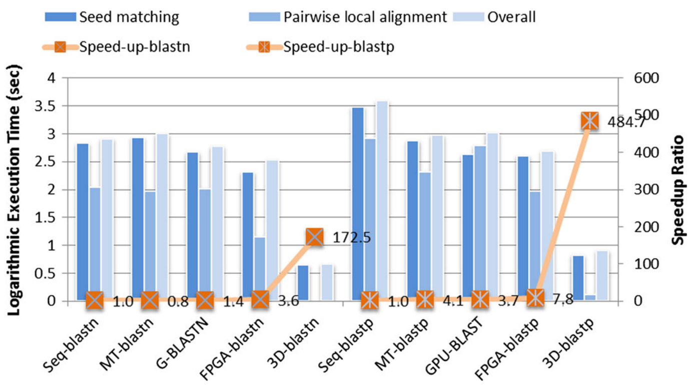


Fig. 31 Performance evaluation of database searching

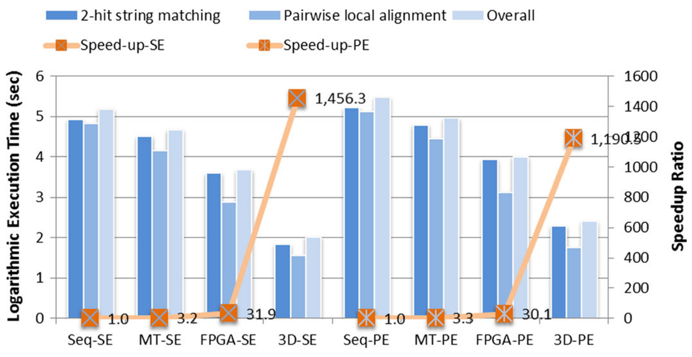


Fig. 32 Performance evaluation of hash-index based short read sequence mapping

dataset which represent the short read mapping problem in real practice. It consists of 63,877,967,101 paired-end short reads, each with 101 bp. Both single- and paired-end reads from this dataset are mapped against the human genome reference GRCh38 [58], and the corresponding results are labeled with "SE" and "PE" suffixes respectively.

Figure 32 shows the execution times measured in seconds of hash-index based short read sequence mapping with both single-end and paired-end dataset regarding 2-hit string matching and pairwise local sequence alignment stages. The overall runtime with respect of acceleration ratio to single-end and paired-end *reads* are shown as the secondary vertical axis respectively.

Sequential performance of BWA v0.6.2 [36] is the baseline of FM-index based short read sequence mapping. BarraCUDA r280 [28] is evaluated for GPU platform of FM-index case, which is the accelerated version of BWA. The short read dataset and reference genome are the same as hash-index based tests.

Figure 33 shows the execution times measured in seconds of FM-index based short read sequence mapping with both single-end and paired-end dataset regarding backward searching and pairwise local sequence alignment stages. The overall runtime with respect of acceleration ratio to single-end and paired-end *reads* are shown as the secondary vertical axis respectively.

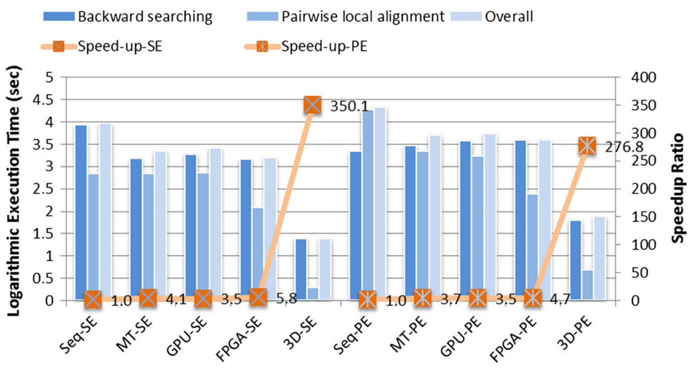


Fig. 33 Performance evaluation of FM-index based short read sequence mapping

It's not forced to complete the 2-hit string matching or backward searching for each *read* before executing pairwise local sequence alignment for verification, thus our architecture could take full use of CGRA resources with limited memory bandwidth available, shown as almost absence of time consumed by pairwise sequence alignment on FPGA and 3D cases of FM-index based short read sequence mapping in Fig. 33.

It can be figured out that the estimated performance of our 3D-stacked accelerator platform outperforms corresponding hardware accelerator solutions for at least 40 times.

7.4 Short Read Mapping Sensitivity

To measure mapping quality, the sensitivity is calculated by dividing the number of aligned short reads by the total number of short reads as (2):

$$Sensitivity = Num\_READ_{mapped}/Num\_READ_{overall} \tag{17}$$

Using the same attributes for seed generation and gap control, our implementations appear to be around the same sensitivity as original BFAST and BWA programs, labeled with "3D" prefix in Table 2.

Table 2 Sensitivity

SRR867061	Single end	Paired end	
	Sensitivity (%)	Sensitivity (%)	Confidence (%)
BFAST	88.85	90.77	86.65
3D-BFAST	88.87	90.81	86.65
BWA	90.15	96.53	96.49
BarraCUDA	91.02	96.61	96.55
3D-BWA	90.64	96.60	96.52

7.5 Memory Bandwidth Efficiency

The stacked DRAM layers gives single sub-processor the peak transfer rate of $128 \times 2 \times 333/8 = 10,656$ MB/s at a cost around 3.2 Watts based on CACTI-3DD models. This value is theoretically worse than that of DE5 board with two DDR3-1600 channels giving 25,600 MB/s, or the new WIDE I/O 2 [59] standard of 34 GB/s. But the idea of extremely low latency and high bandwidth over random access shines through: 3D-stacked SDR DRAM provides the best effort for low latency random memory access, while the sequential transfer rate is also sufficient.

According to the recorded DRAM access traces, in case of hash-index based short read sequence mapping, the effective read bandwidth on DRAM channel 0 of DE5 is approximately 450 MB/s, which consists of accesses to pointer table and original reference sequence. This value is much lower than the estimated value described in Sect. 5, since the memory controller provided by Altera IP is not optimized for random access.

According to simulated DRAM access traces of 3D SDRAM with 333 MHz, the random access is highly benefit from Scatter/Gather read policy, and the effective read bandwidth of DRAM channel 0 is about 1210 MB/s in case of hash-index based short read sequence mapping, pretty much higher than that of FPGA result.

7.6 Energy Efficiency

The logic layer of a sub-processor at 666 MHz on 45-nm process consumes around 3.1 Watts given by post-synthesis simulation. According to CACTI 3DD model, the stacked DRAM layers consume around 3.2 Watts for each sub-processor.

Therefore our example design would consume around $(3.1 + 3.2) \times 16 = 100.8$ Watts energy with 9 layers of 70 mm^2 silicon area. This power density is below the maximum value of thermal dissipation with 200 W/cm^2 as defined by *International Technology Roadmap for Semiconductors* [60], and liquid cooling approaches should work well.

8 Related Work

The first hardware accelerator for bioinformatics and computational biology can be traced back to last century, aiming to accelerate DNA sequence alignment [1]. During recent years, a significant amount of hardware accelerators [2–6] focused on dynamic programming algorithms such as the Needleman and Wunsch [37] and Smith and Waterman [39] algorithms.

For MSA problem, accelerating with FPGA has been well researched [7–10], while recent reports using GPU as accelerator gave very impressive results [22,23].

Applications based on seeds-and-extend that perform DNA sequence matching (BLAST-like) have also been explored for acceleration. There have been researches reported to accelerate all the computation stages of BLAST and its variations using both FPGA [11–14] and GPU [24–27].

As the rising of demands over personalized genetic analysis services, the recent researches focus has shifted to the analysis of NGS, and specifically to the problem of mapping short-reads to the reference genome. A great number of papers have been published that either use FPGAs [16–21,55], or GPUs [28–30].

A decent survey on hardware acceleration for most computational genomics and bioinformatics problems has been published recently [61].

9 Conclusions

The significant growth of biological sequence databases and the rising demands of personalized bioinformatics analytic services urge an innovation on computer architecture. Many accelerators have been proposed but hardly put in real use, as a wide range of applications shall be adopted in bioinformatics while current accelerators just focus on individual problems. Our proposed solution covers a wide range of popular applications that would accelerate the processing of most subfields over biological sequence analysis, including pairwise sequence alignment, MSA, database search, and short read sequence mappings.

By profiling of those algorithms and applications, the most execution time consuming stages have been figured out, while we find that those stages can be categorized into a few sub-problems. We propose a coarse grained reconfigurable fabric which consists of an array of simple basic building blocks that provides signed integer Add/Compare-Select functions. This architecture could efficiently support the acceleration of target applications by analysis of those sub-problems.

Moreover, with well-balanced PEs, buffers, data-paths and resource scheduling, the CGRA can be incorporated with many-core architecture and 3D-stacked technologies for massively parallelization while devoid of the scalability and congestion challenges on regular many-core designs. We implement a stripped down version of the many-core architecture on commodity FPGA, which represents an individual sub-processor along with dedicated DRAM channels. Based on well-balanced task partitions, this single-processor like portion could easily address the estimated overall performance of the many-core design, since it's ensured that no inter-task data exchanges between sub-processors during computation. It leverages a fact that with well-balanced customization of data partition and task scheduling, a reconfigurable fabric could reach the maximum silicon efficiency with limited data throughputs available.

References

1. Hoang, D.T.: Searching genetic databases on Splash 2. In: Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines, pp. 185–191. Napa (1993)

2. Caffarena, G., Bojanic, S., Lopez, J. A., Pedreira, C., Nieto-Taladriz, O.: High-speed systolic array for gene matching. In: Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays (FPGA '04). ACM, pp. 248–248. New York (2004)
3. Oliver, T.F., Schmidt, B., Maskell, D.L.: Reconfigurable architectures for bio-sequence database scanning on FPGAs. IEEE Trans. Circuits Syst. II Express Briefs **52**(12), 851–855 (2005)
4. Gok, M., Yilmaz, C.: Efficient cell designs for systolic Smith–Waterman implementations. In: International Conference on Field Programmable Logic and Applications, pp. 1–4. Madrid (2006)
5. Jiang, X., Liu, X., Xu, L., Zhang, P., Sun, N.: A reconfigurable accelerator for Smith–Waterman algorithm. IEEE Trans. Circuits Syst. II Express Briefs **54**(12), 1077–1081 (2007)
6. Benkrid, K., Liu, Y., Benkrid, A.: A highly parameterized and efficient FPGA-based skeleton for pairwise biological sequence alignment. IEEE Trans. Very Larg. Scale Integr. Syst. **17**(4), 561–570 (2009)
7. Lin, X., Peiheng, Z., Dongbo, B., Shengzhong, F., Ninghui, S.: To accelerate multiple sequence alignment using FPGAs. In: Eighth International Conference on High-Performance Computing in Asia-Pacific Region (HPCASIA'05), pp. 5–180. Beijing (2005)
8. Oliver, T., Schmidt, B., Maskell, D., Nathan, D., Clemens, R.: Multiple sequence alignment on an FPGA. In: 11th International Conference on Parallel and Distributed Systems (ICPADS'05), pp. 326–330. Fukuoka (2005)
9. Yilmaz, C., Gök, M.: An optimized system for multiple sequence alignment. In: International Conference on Reconfigurable Computing and FPGAs, 2009, pp. 178–182. Quintana Roo (2009)
10. Mahram, A., Herbordt, M.C.: FMSA: FPGA-accelerated ClustalW-based multiple sequence alignment through pipelined prefiltering. In: IEEE 20th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2012, pp. 177–183. Toronto (2012)
11. Jacob, A., Lancaster, J., Buhler, J., Chamberlain, R.D.: FPGA-accelerated seed generation in mercury BLASTP. In: 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2007), pp. 95–106. Napa (2007)
12. Sotiriades, E., Dollas, A.: Design space exploration for the BLAST algorithm implementation. In: 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2007), pp. 323–326. Napa (2007)
13. Kasap, S., Benkrid, K., Liu, Y.: High performance FPGA-based core for BLAST sequence alignment with the two-hit method. In: 8th IEEE International Conference on BioInformatics and BioEngineering BIBE, pp. 1–7. Athens (2008)
14. Chen, Y., Schmidt, B., Maskell, D.L.: Reconfigurable accelerator for the word-matching stage of BLASTN. IEEE Trans. Very Larg. Scale Integr. Syst. **21**(4), 659–669 (2013)
15. Olson, C.B. et al.: Hardware acceleration of short read mapping. In: IEEE 20th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 161–168. Toronto (2012)
16. Sogabe, Y., Maruyama, T.: An acceleration method of short read mapping using FPGA. In: International Conference on Field-Programmable Technology (FPT), pp. 350–353 (2013)
17. Chen, Y., Schmidt, B., Maksell, D.L.: An FPGA aligner for short read mapping. In: 22nd International Conference on Field Programmable Logic and Applications (FPL), pp. 511–514. Oslo (2012)
18. Arram, J., Tsoi, K.H., Luk, W., Jiang, P.: Reconfigurable acceleration of short read mapping. In: IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 210–217. Seattle (2013)
19. Chen, P., Wang, C., Li, X., Zhou, X.: Accelerating the next generation long read mapping with the FPGA-based system. IEEE/ACM Trans. Comput. Biol. Bioinform. **11**(5), 840–852 (2014)
20. Fernandez, E.B., Villarreal, J., Lonardi, S., Najjar, W.A.: FHAST: FPGA-based acceleration of Bowtie in hardware. IEEE/ACM Trans. Comput. Biol. Bioinform. **12**(5), 973–981 (2015)
21. Waidyasooriya, H.M., Hariyama, M.: Hardware-acceleration of short-read alignment based on the Burrows–Wheeler transform. IEEE Trans. Parallel Distrib. Syst. **27**(5), 1358–1372 (2016)
22. Liu, Y., Schmidt, B., Maskell, D.L.: MSA-CUDA: Multiple sequence alignment on graphics processing units with CUDA. In: 20th IEEE International Conference on Application-Specific Systems, Architectures and Processors, 2009, pp. 121–128. Boston (2009)
23. Blazewicz, J., et al.: G-MSA—A GPU-based, fast and accurate algorithm for multiple sequence alignment. J. Parallel Distrib. Comput. **73**(1), 32–41 (2013)
24. Vouzis, P.D., Sahinidis, N.V.: GPU-BLAST: using graphics processors to accelerate protein sequence alignment. Bioinformatics **27**(2), 182–188 (2011)

25. Liu, W., Schmidt, B., Liu, Y., Voss, G., Mueller-Wittig, W.: Mapping of BLASTP algorithm onto GPU clusters. In: IEEE 17th International Conference on Parallel and Distributed Systems (ICPADS), pp. 236–243. Tainan (2011)
26. Zhao, K., Chu, X.: G-BLASTN: accelerating nucleotide alignment by graphics processors. Bioinformatics **30**(10), 1384–1391 (2014)
27. Zhang, J., Wang, H., Feng, W.C.: cuBLASTP: Fine-grained parallelization of protein sequence search on CPU+GPU. In: IEEE/ACM Transactions on Computational Biology and Bioinformatics, vol. 99, pp. 1–1
28. Klus, P., et al.: BarraCUDA-a fast short read sequence aligner using graphics processing units. BMC Res. Notes **5**, 27 (2012)
29. Liu, Y., Schmidt, B.: CUSHAW2-GPU: empowering faster gapped short-read alignment using GPU computing. IEEE Des. Test **31**(1), 31–39 (2014)
30. Chacón, A., Marco-Sola, S., Espinosa, A., Ribeca, P., Moure, J.C.: Boosting the FM-index on the GPU: effective techniques to mitigate random memory access. In: IEEE/ACM Transactions on Computational Biology and Bioinformatics Sept.–Oct. 1, vol. 12(5), pp. 1048–1059 (2015)
31. Blaststation: Benchmark tests of NCBI Blast+ on Amazon EC2. http://www.blaststation.com/freestuff/en/benchmarkBlastCloud.html. Accessed 31 Dec 2016
32. Thompson, J.D., Higgins, D.G., Gibson, T.J.: CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. Nucl. Acids Res. **22**(22), 4673–4680 (1994)
33. Notredame, C., et al.: T-Coffee: a novel method for fast and accurate multiple sequence alignment. J. Mol. Biol. **302**(1), 205–217 (2000)
34. Altschul, S.F., et al.: Basic local alignment search tool. J. Mol. Biol. **215**(3), 403–410 (1990)
35. Homer, N., et al.: BFAST: an alignment tool for large scale genome resequencing. PLoS ONE **4**(11), e7767 (2009)
36. Li, H., Durbin, R.: Fast and accurate long-read alignment with Burrows–Wheeler transform. Bioinformatics **26**(5), 589–595 (2010). (PMC. Web. 9 June 2016)
37. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol. **48**(3), 443–453 (1970)
38. Altschul, S.F.: Amino acid substitution matrices from an information theoretic perspective. J. Mol. Biol. **219**(3), 555–565 (1991)
39. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. J. Mol. Biol. **147**(1), 195–197 (1981)
40. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7), 422–426 (1970)
41. Ferragina, P., Manzini, G.: Opportunistic data structures with applications. In: Proceedings of the 41st Annual Symposium on Foundations of Computer Science, pp. 390–398 (2000)
42. Burrows, M., Wheeler, D.J.: A block sorting lossless data compression algorithm. SRC Research Report 124, Digital Equipment Corporation, Palo Alto, California (1994)
43. Manber, U., Myers, G.: Suffix arrays: a new method for on-line string searches. SIAM J. Comput. **22**(5), 935–948 (1993)
44. Amdahl, G.M.: Validity of the single processor approach to achieving large scale computing capabilities, In: Proceedings of the AFIPS '67 Spring Joint Computer Conference, pp. 483–485 (1967)
45. Longbottom, R.: RandMem Benchmark. http://www.roylongbottom.org.uk/randmem%20results.htm. Accessed 31 Dec 2016
46. Intel$^{®}$ Core^{TM} i7 Processor. http://ark.intel.com/products/family/59143/. Accessed 31 Dec 2016
47. JEDEC: DDR3 SDRAM Standard, JESD79-3F. https://www.jedec.org/standards-documents/docs/jesd-79-3d (2012). Accessed 31 Dec 2016
48. Mirsky, E., DeHon, A.: MATRIX: a reconfigurable computing architecture with configurable instruction distribution and deployable resources. In: Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines, 1996, pp. 157–166 (1996)
49. Loi, I., Benini, L.: An efficient distributed memory interface for many-core platform with 3D stacked DRAM. In: Design, Automation and Test in Europe Conference and Exhibition (DATE), pp. 99–104 (2010)
50. JEDEC: Wide I/O Single Data Rate, JESD229. https://www.jedec.org/standards-documents/docs/jesd229 (2011). Accessed 31 Dec 2016

51. Edmiston, E., et al.: Parallel processing of biological sequence comparison algorithms. Int. J. Parallel Program **17**(3), 259–275 (1988)
52. Ibarra, O., Palis, M.: VLSI algorithms for solving recurrence equations and applications. IEEE Trans. Acoust. Speech Signal Process. **35**(7), 1046–1064 (1987)
53. Chao, K.M., et al.: Aligning two sequences within a specified diagonal band. Comput. Appl. Biosci. **8**(5), 481–487 (1992)
54. Weis, C., et al.: Exploration and optimization of 3-D integrated DRAM subsystems. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **32**(4), 597–610 (2013)
55. Europractice: Standard Cell Library TSMC. http://www.europractice-ic.com/libraries_TSMC.php. Accessed 31 Dec 2016
56. Chen, K., et al.: CACTI-3DD: Architecture-level modeling for 3D diestacked DRAM main memory. In: Design, Automation Test in Europe Conference Exhibition (DATE), pp. 33–38 (2012)
57. Thompson, J.D., et al.: BAliBASE 3.0: latest developments of the multiple sequence alignment benchmark. Proteins **61**(1), 127–136 (2005)
58. Miga, K.H., et al.: Centromere reference models for human chromosomes X and Y satellite arrays. Genome Res. **24**(4), 697–707 (2014)
59. JEDEC: Wide I/O 2 (WideIO2), JESD229-2. https://www.jedec.org/standards-documents/docs/jesd229-2 (2014). Accessed 31 Dec 2016
60. International Technology Roadmap for Semiconductors. http://www.itrs2.net. Accessed 31 Dec 2016
61. Aluru, S., Jammula, N.: A review of hardware acceleration for computational genomics. IEEE Des. Test **31**(1), 19–30 (2014)