PoC-Align: An Open-Source Alignment Accelerator using FPGAs

Thomas B. Preußer and Oliver Knodel and Rainer G. Spallek
Department of Computer Science
Technische Universität Dresden
Dresden, Germany
{thomas.preusser, oliver.knodel, rainer.spallek}@tu-dresden.de

Abstract—**The mapping of reads, i.e. short DNA base pair strings, to large genome databases has become a critical operation for genetic analysis and diagnosis. The underlying alignment operation essentially is a string search tolerating some character mismatches and possibly character deletions or insertions with respect to a reference genome. Its output comprises the locations within the reference that are likely to correspond to the mapped DNA snippet.**

This paper describes PoC-Align, an alignment infrastructure using FPGA accelerators. It is an extension of our preceding FPGA aligner [1], which has been enhanced to tolerate alignment gaps (insertions and deletions) and to be more customizable though generic parameters. In addition to the descriptions of the implementation of these extensions, we also name the mainly software-carried enhancements, such as the support of mapping paired-end reads, that are implemented on top of the FPGA accelerator. Providing a thorough overview on the complete infrastructure, we aim at advertising the disclosure of the sources of our solution and hope to encourage other groups to use and extend this platform.

Keywords*—*Short-Read Mapping; Sequence Alignment; FPGA

I. Introduction

The mapping of reads, i.e. short DNA base pair strings, to large genome databases has become a critical operation for genetic analysis and diagnosis. Completely sequenced genomes are available for several species, including humans. These references enable an analytic comparison with newly sequenced individuals in order to shed light on the descent or the evolutionary linage of a population or to determine the individual genetic disposition for diseases or other critical conditions [2]. Providing the base for interpretation, the correct mapping of the sequenced DNA fragments to their corresponding locations in the reference genome is extremely important.

The cost-effective standard analysis nowadays relies on the fast automated sequencing of millions of short fragments obtained from the multiplied relevant DNA material [3]. The mapping of this vast number of reads requires an extremely efficient search within the large reference, which must be tolerant of the individual genetic variance reflected by base mismatches, insertions or deletions (gaps).

Typical genome databases consist of many billions of base pairs (bp) and may have a storage size of several GByte. Each bp is represented by one of the four nucleotides A, G, T(=U) or C. These establish the alphabet of the genome string. Contemporary next-generation sequencers as by Applied Biosystems, Illumina and Roche produce millions of short reads with a length of 30–100 bp in a single run [4]. The mapping obtained from only one experiment may take hours or even days with conventional software tools. This motivated the development of designated short-read mappers, which often display a reduced tolerance of mismatches and do not allow gaps [5]. They apply search heuristics to increase their processing speed sacrificing accuracy. Thus, short-read mappers can generally not guarantee to locate every mapping of a read, which would be valid under the set mismatch constraints.

This paper describes the enhancements of a previously published FPGA aligner [1] and the infrastructure built around it. Similar to Bowtie 2, the hardware mapping was split into (a) the computation of candidate positions based on matching subsections of the read, and (b) their filtering by an an end-to-end alignment unit. On top of the hardware accelerator, a client-server software infrastructure provides a high-level Java API and ready-to-use clients to handle standard data formats of this application domain, such as FASTA and SAM.

In the remainder of this paper, the next section presents the state of the art of different mapping approaches. Sec. III describes the overall PoC-Align infrastructure before Sec. IV will detail the revised hardware mapping approach and the incentives for novel features. The customization options of the generic accelerator design are explained by Sec. V. Representative configurations for common off-the-shelf boards are shown before Sec. VII summarizes and concludes this paper.

II. State of the Art

A. Traditional Mappers

The classic *exact* read mapping approach is the Smith-Waterman algorithm [6], which computes a score matrix whose dimensions correspond to the lengths of the database and of the read. The algorithm is able to tolerate an arbitrary number of mismatches and gaps. It can compute the bound for the score of the best alignment even without an explicitly given matching constraints. Despite its computational complexity, the Smith-Waterman algorithm is still the method of choice if exact results are required, which enumerate *every* valid mapping location. Highly optimized software implementations are freely available [7], [8]. Implementations for GPUs and also FPGAs have been reported [9]–[12].

The pioneer among the heuristic mappers is BLAST [13]. BLAST searches the database for seeds, i.e. substrings taken from a read. It then attempts to extend and combine these partial mappings to form a full alignment. BLAST efficiently maps long sequences. However, it suffers from reads that are short themselves and becomes less attractive as the complexity rather shifts to the sheer number of reads. This struggle to scale down to short read length and up to a great number of reads is known as the short-read mapping problem.

B. Short-Read Mappers

The short-read mapping problem has promoted the development of better suited heuristics as in short-read mappers like Bowtie [14], Maq [15], PASS [16] and RazerS [17].

Many short-read mappers do not operate on the reference genome database directly. They rather perform a significant data transformation such as the Burrows-Wheeler transform used by Bowtie. Bowtie uses this representation to search for perfect matches of the read using a Ferragina-Manzini (FM) index [18]. This highly efficient and fast search only inspects relevant parts of the original database. However, mismatches can only be tolerated through expensive backtracking. Gaps cannot be handled at all. The reworked revision Bowtie 2 [19] adds the capability to tolerate some gaps by using the FM search for seeds only. If multiple seeds are located in mutual proximity, the determined location is considered a candidate mapping. An evaluation by an end-to-end alignment of the read against the candidate database segment finally determines whether or not the location is worth to be reported.

Maq and PASS are based on spaced-seed indexing. Fixed-length sequences taken from every database position are chopped into seeds, which are stored in a large lookup table. A read to be mapped is also chopped up. While some pieces must match the seeds in the lookup table exactly, Maq tolerates mismatches by allowing additional unmatched space. Maq regularly reports locations with more mismatches than specified. Another disadvantage of the current Maq implementation is that it only reports one mapping location per read. The spaced-seed indexing is mainly used by older mappers and is significantly slower than a search using an FM index.

RazerS uses the *q-gram counting* strategy [20] to identify potentially matching regions, which are then treated in detail. In contrast to Bowtie, more mismatches as well as gaps can be tolerated.

The speed and accuracy of all short-read mappers depends strongly on the read length and the number of tolerated mismatches. They do not guarantee to report all valid locations. Typically, they are able to detect at least one of the valid mappings of about 70%-90% of the mappable reads [5]. The other way around, between 10%-30% of mappable reads are falsely reported to not map at all.

C. Mappers on Parallel Platforms

Parallel hardware platforms such as GPUs and FPGAs promise a tremendous performance boost for suitable parallel applications. These platforms offer a huge potential for concurrency, which is hard and expensive to reach with full-blown general-purpose processors. However, this potential benefit requires a harder implementation effort on a lower, often even structural abstraction level. This is especially true for FPGA designs.

The short-read mapping can be naturally parallelized simply due to the great number of reads to map. Orthogonal parallelization approaches as by assigning different database slices to concurrent search engines are possible but not necessary. For expectedly large database slices, such approaches would even be counterproductive when these slices exceed the capacity of the local storage.

There are FPGA and GPU implementations of the Smith-Waterman algorithm [9]–[11] and accelerators for BLAST [21]–[23] available. These mappers boost performance using large numbers of parallel computation units. Also, short-read mappers have been implemented on parallel platforms such as FPGAs. Tang et al. [24] introduced a heterogeneous architecture with a FPGA accelerator using a hash-index method. The most time-consuming and basic operations are transferred from the host CPU to up to 100 specialized processing elements on a Xilinx Virtex-5 LX330 FPGA. Arram et al. [25] and also Cheema et al. [26] designed FPGA accelerators using an FM search after a Burrows-Wheeler transform.

Olson et al. [27] have proposed an FPGA mapper, which combines a seed-based search within a database index with a following regional Smith-Waterman alignment to verify the detected candidate positions. They report a noteworthy speedup of an order of magnitude compared to Bowtie. However, they also assume a high-end system comprising eight Xilinx Virtex-6 LX240T FPGAs with 4 GB of DDR3 RAM attached to each one of them. This large memory space is mostly needed for buffering the database seed index, which is greatly inflated in comparison to the original database representation.

Our alignment infrastructure extends the systolic FPGA-based alignment accelerator described before [1]. Similar to Bowtie 2, we now use the original alignment approach for the search of smaller read fragments to identify candidate mappings. A succeeding Smith-Waterman alignment against the obtained database location determines, which of these candidates qualify as valid mappings. This measure enables the mapping process to tolerate gaps that do not impact more than a configurable number of read segments at the same time.

III. The PoC-Align Infrastructure

Fig. 1 provides an overview over our alignment infrastructure. Its central component is a server application implemented in Java. This server is responsible for the management of the alignment resources and provides administrative and client interfaces. The text-based administrative interface is accessed through an arbitrary terminal program that can connect to a local TCP socket. The provided functionality comprises:

- administration: attach- and detachment of FPGA accelerators; loading and unloading of available genome databases.
- statistics: view alignment statistics and workload distribution.
- diagnostics: list managed resources, attached clients, active JVM threads etc.
- assistance: an overview of all available commands is printed upon the command `help` or `?`.

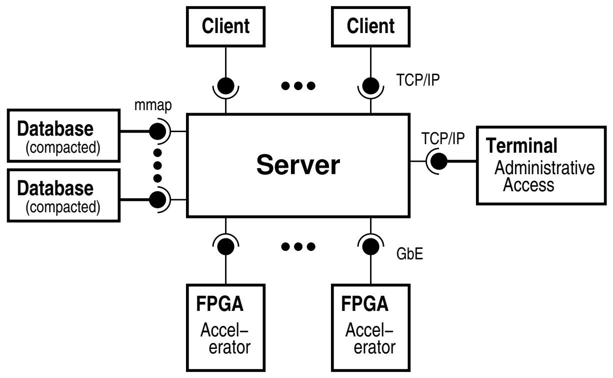

Fig. 1. Infrastructure Overview

The communication with the FPGA accelerators is based on the exchange of raw packets across a GigaBit Ethernet link. The implemented protocol provides a sequence and flow control based on sequence numbers. The announcement of the next sequence number expected by the other end implements a cumulative acknowledgement, which is piggy-backed within each transmission. An additional `NAK` flag is used to trigger an immediate retransmission, instead of waiting for the timeout, in the case that the communication partner has detected a lost or broken packet directly.

Significant backpressure across the Ethernet link must be sustained to throttle the host when streaming the reference database to the FPGA. The smooth operation under these conditions requires a more sophisticated flow control. This was implemented on the basis of credits granted by the FPGA. The overall packet format with the optional `Credit` field only used for the upstream communication from the FPGA is shown in Fig. 2. A `RawSocket` class with appropriate native methods makes the raw Ethernet communication accessible to the Java implementation of the server.

Growing space in the input buffers is actively announced by the FPGA even if no result packet is ready for transmission. Even when idling, the accelerator transmits heartbeat packets. These packets simplify the attachment to a server as they convey the relevant geometric dimensions of the specific accelerator instance. The server detects an accelerator simply by listening for these heartbeat packets on the available interfaces.

The databases provided for alignment are memory-mapped from files that are pre-formatted for efficient processing. The representation of the genome sequences is reduced to two bits per bp. Regions of neutral `N`s are eliminated as they cannot yield sensible mapping locations. They are, however, counted for the correct reporting of result positions. This is achieved through an offset map that translates the positions reported by the accelerator on the basis of the compacted database into the real positions within the original inflated database.

After the accelerator has been programmed for specific reads, the mapping process is performed against the compacted database being streamed to the FPGA device. The server performs this streaming via a `RawSocket`, which utilizes a native `sendmsg` system call for the data transmission. This system call is able to assemble a packet to transmit from several data buffers. This avoids an explicit assembly by a copying operation in user space. The packets are built from two parts: the header and the payload. Each of them is represented by its own direct `ByteBuffer`. When streaming the database, the payloads are simply defined as a slice of the memory-mapped database file.

Also clients connect to the server so as to post mapping jobs. The communication across a TCP/IP link allows local and remote accesses. The application-level protocol is implemented by the abstract *`Client`* class, which provides a high-level API to subclasses for querying the available databases and issuing mapping jobs on them. An abstract callback method must be implemented for receiving the mapping results. The PoC-Align distribution includes (a) a `TestClient`, which randomly derives mutated queries typically from the same database and reports the mapping success, and (b) a `FastaClient`, which reads the queries from a FASTA text file and dumps the located mappings to a SAM file. A specialization of this client allows to process paired-end queries from a FASTA file. Then two adjacent queries form a pair of read segments with an approximately known gap between them. Command-line options are available to specify parallel or antiparallel segment directions and whether or not the second segment is from the complement DNA strain.

IV. Hardware Mapper Design

The key component of the PoC-Align infrastructure is the hardware mapper. The initial determination of candidate locations is realized by an extremely parallel computation matching the currently configured reads against each position of the database. The matching units, each responsible for one read, are combined into daisies that share the same storage for the streamed database segment. In practice, these daisies must be limited to comprise at most 16 matchers so as to maintain a competitive clock rate. Thus, further concurrency is achieved by implementing a daisy chain, in which the individual daisies are separate clock-enable domains isolated by FIFOs.

The matchers operate in between two parallel streams. The token stream taking the path on the left side of Fig. 3 carries both (a) the read configuration to program the matchers and (b) the reference database. Each token consists of three bits, the first of which is a command flag. If set, the remaining two bits control the operation mode so as to start or stop the read configuration or the database streaming. The read configuration is performed daisy by daisy starting from the beginning of the chain. The first unconfigured daisy grabs a configuration changing the opening `CFG_START` token into a `CFG_STOP` to avoid duplicated configurations in upstream daisies. Once configured, a daisy passes `CFG_START` tokens without reconfiguring itself so that the complete chain can be configured with reads to map. Only after having seen a database stream wrapped within `DB_START` and `DB_STOP`, a daisy will respond to a `CFG_START` token again.

Tokens with a cleared command flag contain two data bits. Within a database stream, these two bits encode one bp. Within a configuration stream, they represent two parallel configuration streams, which are simply shifted into the matchers. One of the bits configures the odd-numbered matchers, the other one the even-numbered matchers of a daisy.

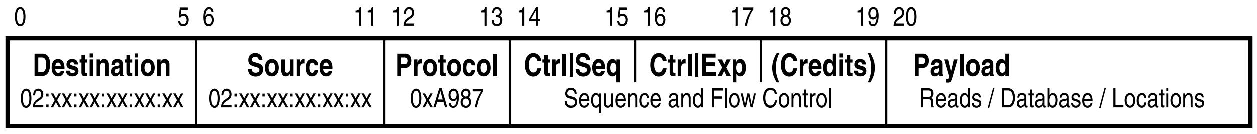

Fig. 2. Format of Ethernet Packets used in FPGA Communication

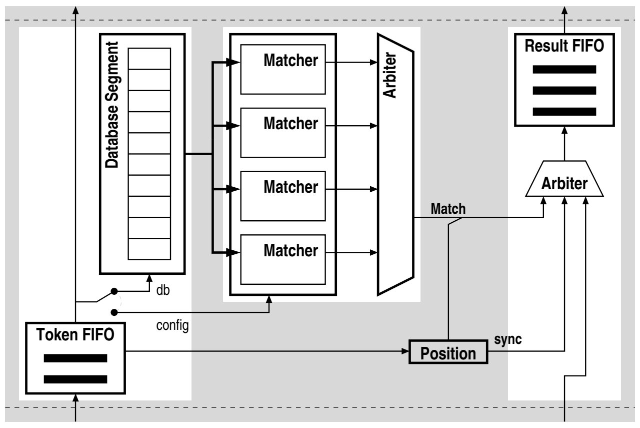

Fig. 3. Computation of Candidate Locations

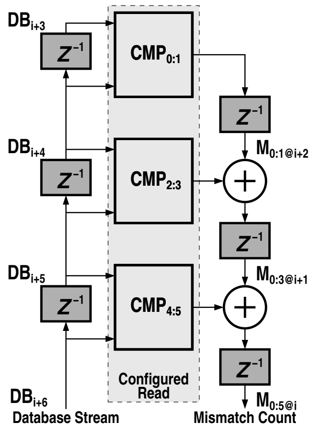

Fig. 4. Systolic Architecture of Matcher Segments

The matchers compute candidate mapping locations as the database streams by. The candidate read is identified by the matcher coordinates, i.e. the number of the daisy in the chain and the number of the matcher within the daisy. The location is encoded as an offset within the current database block. This offset encoding reduces the bit width required for the result stream. In fact, a constant offset width of a few bits suffices irrespective of the size of the database. The association to a database block is achieved by `sync` messages embedded into the result stream, which flows upwards along the right side of Fig. 3. These `sync` messages pace the computation of upstream daisies as their forwarding is synchronized with the overflow of the local position counter. The true position of a reported result is determined by counting the `sync` messages preceding it in the result stream:

$$\text{preceding\_syncs} \times \text{block\_size} + \text{offset}$$

Note that the parallel token and result streams differ from the original architecture, which used anti-parallel streams [1]. This invention decouples the width of the local position counters from the database size and enforces some ordering on the reported results, which cannot cross block boundaries. This new property is important for the efficient implementation of the Smith-Waterman-like alignment that evaluates and filters the candidates before they are reported to the host.

A small block size reduces the bit width of the result stream. However, widths below 8 bits are hardly advisable as the result stream will be dominated by `sync` messages. This occupies bandwidth and eventually impacts concurrency due to its synchronizing nature. At the end of the result stream, the positions reported back to the host are recoded assuming a different block size. This is done so that the 32-bit result words are utilized optimally irrespective of the internal dimensioning of the position counters.

The matchers are based on the original systolic architecture as depicted in Fig. 4 [1]. In contrast to that approach, the configured read is no longer compared to the current database segment as a whole. Rather, it is segmented into a configurable number of divisions, which are matched individually. A local minimum in the mismatch count of a division remains active over several cycles as determined by a `SLACK` parameter. This allows to associate matching segments of a read even in the presence of a gap. The selection of a candidate is based on adding the active mismatch counts of the divisions. The produced sum, however, only comprises half of these counts selecting the smallest inputs. This way, divisions affected by a gap and producing high mismatch counts are ignored. This relaxed treatment of mismatches is why a conclusive filtering of the candidates by an explicit alignment is necessary.

The internal computation of the mismatch counts uses saturating arithmetic – interpreting the maximum value as infinity. Thus, the associated data paths can be limited to three bits in practice. The individual comparators of a matcher are implemented by runtime-configurable LUTs using CFGLUT5 device primitives [29]. They produce a 2-bit result representing the number of mismatches found between the current two database positions and the two configured read positions. The read configuration is computed by the host and shifted into the LUTs before the reference database is streamed by. The used scheme is illustrated by Fig. 5, which shows the configuration for a comparator that is to represent the read sequence `AG`.

As depicted in Fig. 6, the aligning filter is located at the end of the daisy chain. It evaluates the reported candidates by aligning the identified read against the reported database location. This requires the availability of the actual read and database contents. These are reconstructed from the token stream, which terminates in this unit. The configuration blocks are decoded into a plain bp sequence and stored within on-chip RAM. As the neutral base `N` is supported for read

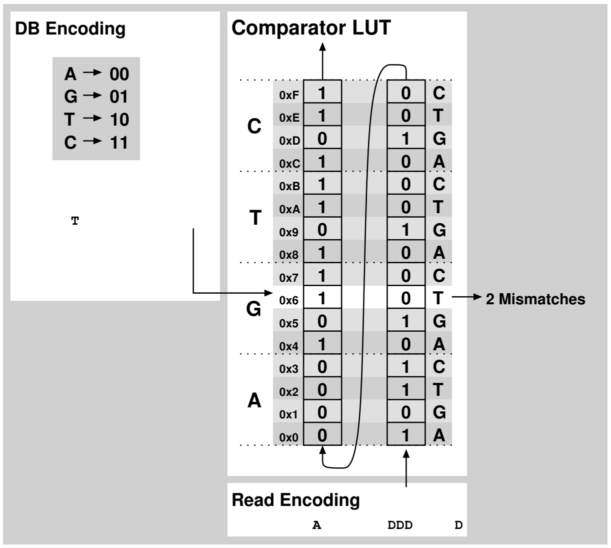

Fig. 5. Example Comparator Configuration [28]

CHAINLEN	Length of daisy chain: production designs should simply try to fill the device for highest concurrency.
HBREADTH	Half the number of the matchers per daisy: values between 4 and 8 are good trade-offs in terms of resource sharing and the achievable clock frequency.
READ_LEN	Length of the matched reads: must be a multiple of $2\times$READ_DIV.
READ_DIV	The number of divisions the reads are segmented into for matching.
OFS_BITS	The internal bit width of the position counters: values around 10 offer sensible trade-offs between resource use and implied synchronization.
MIS_BITS	The bit width of the mismatch counts: a value of 3 is sufficient for most practical purposes.
SLACK	The number of positions, by which matching read divisions may be offset from their expected relative location.

TABLE I. GENERICS FOR CUSTOM DESIGN CONFIGURATION

contents, each read symbol is represented by three bits there. No recoding is necessary for the database stream. However, it is not sensible to collect the complete database. Instead, only the database blocks currently in use are kept within a cyclic buffer. A stored block can be discarded and overwritten when all candidates preceding the sync announcing the next block have been processed. Note that all resources used for result filtering will be eliminated from the design if the matchers only implement a single divisions. Gaps cannot be handled in this configuration.

V. DESIGN CUSTOMIZATION

Tab. I summarizes the generic parameters available for customizing the geometry and the sensitivity of a particular accelerator instance. The number of reads that can be mapped by one run is determined by the product CHAINLEN $\times$ $2 \cdot$ HBREADTH. While more resources can be shared within a daisy, the HBREADTH parameter should typically not be increased beyond 8 as this impacts the achievable clock frequency heavily. Instead, CHAINLEN should be increased to fill up a device for productive use. A clock frequency of 200 MHz is typically achievable with such a geometric configuration.

The parameters READ_LEN and READ_DIV determine the length of the reads that can be matched by the hardware and the number of divisions used in the matching process. The server software is truncates or splits longer reads and extends

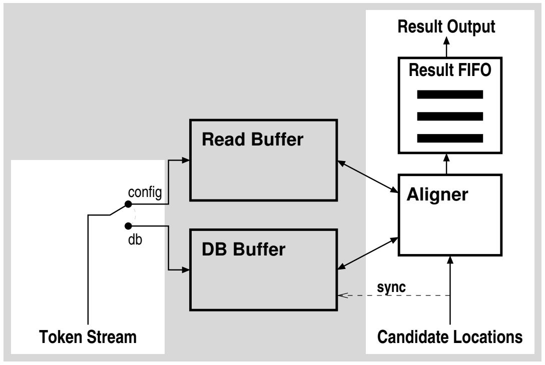

Fig. 6. Aligning Result Filter

shorter ones by neutral Ns that do not contribute mismatches. Longer reads will then require some postprocessing in software to validate the partial mappings in the context of the complete read. In practice, READ_LEN values around 50 bp appeared most useful as the number of reported mapping locations is small enough to make a postprocessing of longer reads in software feasible. Assuming a READ_DIV of 3, a READ_LEN of 48 is a sensible value.

The parameters OFS_BITS and MIS_BITS dimension internal data paths. A MIS_BITS of 3 is sufficient for practical purposes and ideal for the utilization of the 6-input LUTs found on the supported Xilinx devices of generation 5 or newer.

The SLACK parameter determines the sensitivity of the mapper in the context of gaps. It should be chosen as small as possible since it heavily affects the resource consumption by the matchers and by the aligning result filter. This is because a greater slack requires the regional alignment to be performed against a larger window of the database.

The server application allows to utilize several hardware accelerators with significantly different slack values. These are combined into a processing chain where reads are mapped on the low-slack accelerator first. Only if no acceptable mapping was found, it is mapped again on the high-slack accelerator. The policy determining whether or not an additional mapping attempt is made on a more sensitive accelerator is defined by the client.

Tab. II summarizes the design sizes and clock frequencies that can be realized on a selection of Virtex devices. The resulting performance is the product of the matcher count and the achieved clock frequency. For all listed implementations, a matcher supports a read length of 48 bp and eight matchers are combined into one daisy. Thus, the contrasted mapper geometries only differ in the lengths of the daisy chains that fit onto the respective devices with two alterantive READ_DIV configurations. The table also gives an overview of the device utilization.

VI. SOURCE DISTRIBUTION

The sources of the complete infrastructure are released under the Apache License 2.0 in the context of the PoC library at http://poc.inf.tu-dresden.de/wiki/doku.php?id=align.

TABLE II. PLATFORM COMPARISON OF VARIOUS VIRTEX DEVICES

Board	Device	Generic Parameters				Matchers	Clock Frequency	Utilization (Occupied Slices)	Performance Product
		CHAINLEN	HBREADTH	READ_LEN	READ_DIV				
ML505	XC5VLX50T	21	4	48	1	168	200 MHz	6,907 (95%)	33,600
		17	4	48	3	136	200 MHz	6,864 (95%)	27,200
ML605	XC6VLX240T	130	4	48	1	1,040	250 MHz	36,297 (96%)	260,000
		102	4	48	3	816	250 MHz	35,074 (93%)	204,000
VC707	XC7VX485T	225	4	48	1	1,800	250 MHz	67,998 (89%)	450,000
		170	4	48	3	1,360	250 MHz	68,333 (90%)	340,000

VII. CONCLUSIONS

This paper has presented PoC-Align, an open-source alignment infrastructure utilizing FPGA accelerators in the process of read mapping. The underlying hardware mapper is an adaptation of a previously proposed approach, which was further enhanced to allow the mapping of reads containing gaps. The hardware mapper is highly customizable. A powerful server application enables the easy utilization of attached accelerators locally or remotely through a high-level Java API. The server refines the functionality offered by the accelerators by providing more complex mapping solutions such as for long reads or paired-end reads. We hope that the open-source release of PoC-Align will create a greater community of users who will become a valuable source of feedback for further development. We also enable this community to contribute themselves to the evolution of PoC-Align.

REFERENCES

[1] T. B. Preußer, O. Knodel, and R. G. Spallek, "Short-read mapping by a systolic custom FPGA computation," in *IEEE 20th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM 2012)*. IEEE, May 2012, pp. 169–176.

[2] J. Reis-Filho, "Next-generation sequencing," *Breast Cancer Research*, vol. 11, no. Suppl 3, p. S12, 2009.

[3] M. Margulies, M. Egholm, W. E. Altman, S. Attiya, J. S. Bader, L. A. Bemben, J. Berka, M. S. Braverman, Y.-J. Chen, Z. Chen, S. B. Dewell, L. Du, J. M. Fierro, and Gome, "Genome sequencing in microfabricated high-density picolitre reactors," *Nature*, vol. 437, no. 7057, pp. 376–380, 2005.

[4] E. R. Mardis, "The impact of next-generation sequencing technology on genetics," *Trends in Genetics*, vol. 24, no. 3, pp. 133–41, 2008.

[5] C. Trapnell and S. L. Salzberg, "How to map billions of short reads onto genomes," *Nature Biotechnology*, vol. 27, no. 5, pp. 455–457, 2009.

[6] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, pp. 195–197, Mar 1981.

[7] T. Rognes and E. Seeberg, "Six-fold speed-up of Smith-Waterman sequence database searches using parallel processing on common microprocessors," *Bioinformatics*, vol. 16, no. 8, pp. 699–706, 2000.

[8] U. of Virginia, "FASTA sequence comparison at the U. of Virginia," 2011, http://fasta.bioch.virginia.edu/fasta_www2/fasta_intro.shtml.

[9] I. Li, W. Shum, and K. Truong, "160-fold acceleration of the Smith-Waterman algorithm using a field programmable gate array (FPGA)," *BMC Bioinformatics*, vol. 8, no. 1, p. 185, 2007.

[10] T. Oliver, B. Schmidt, and D. Maskell, "Hyper customized processors for bio-sequence database scanning on FPGAs," in *Proceedings of the 2005 ACM/SIGDA 13th International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '05. New York, NY, USA: ACM, 2005, pp. 229–237.

[11] S. A. Manavski and G. Valle, "CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment," *BMC Bioinformatics*, vol. 9, no. Suppl 2, p. S10, Mar 2008.

[12] C. W. Yu, K. H. Kwong, K. H. Lee, and P. H. W. Leong, "A Smith-Waterman systolic cell," in *13th International Workshop on Field Programmable Logic and Applications FPL 2003*. Springer, 2003, pp. 375–384.

[13] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, no. 3, pp. 403–410, Oct 1990.

[14] B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg, "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome," *Genome Biology*, vol. 10, no. 3, p. R25, 2009.

[15] H. Li, "Maq: Mapping and assembly with qualities," 2008, http://maq.sourceforge.net/.

[16] D. Campagna, A. Albiero, A. Bilardi, E. Caniato, C. Forcato, S. Manavski, N. Vitulo, and G. Valle, "PASS: a program to align short sequences," *Bioinformatics*, vol. 25, no. 7, pp. 967–968, 2009.

[17] D. Weese, A.-K. Emde, T. Rausch, A. Döring, and K. Reinert, "RazerS - fast read mapping with sensitivity control," *Genome Research*, vol. 19, no. 9, pp. 1646–1654, Jul 2009.

[18] P. Ferragina and G. Manzini, "Opportunistic data structures with applications," in *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, ser. FOCS '00. Washington, DC, USA: IEEE Computer Society, 2000, pp. 390–.

[19] B. Langmead and S. L. Salzberg, "Fast gapped-read alignment with Bowtie 2," *Nature Methods*, vol. 9, no. 4, pp. 357–359, Apr 2012.

[20] O. Owolabi and D. R. McGregor, "Fast approximate string matching," *Software: Practice and Experience*, vol. 18, no. 4, pp. 387–393, 1988.

[21] M. C. Herbordt, J. Model, B. Sukhwani, Y. Gu, and T. VanCourt, "Single pass streaming BLAST on FPGAs," *Parallel Computing*, vol. 33, no. 10-11, pp. 741–756, 2007.

[22] E. Sotiriades, C. Kozanitis, and A. Dollas, "FPGA based architecture for DNA sequence comparison and database search," in *Proceedings of the 20th International Conference on Parallel and Distributed Processing*, ser. IPDPS'06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 193–193.

[23] B. C. Lam, C. Pascoe, S. Schaecher, H. Lam, and A. D. George, "BSW: FPGA-accelerated BLAST-Wrapped Smith-Waterman aligner." in *ReConFig*, 2013, pp. 1–7.

[24] W. Tang, W. Wang, B. Duan, C. Zhang, G. Tan, P. Zhang, and N. Sun, "Accelerating millions of short reads mapping on a heterogeneous architecture with FPGA accelerator," in *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*. IEEE Computer Society, 2012, pp. 184–187.

[25] J. Arram, K. H. Tsoi, W. Luk, and P. Jiang, "Reconfigurable acceleration of short read mapping," in *Field-Programmable Custom Computing Machines (FCCM), 2013 IEEE 21st Annual International Symposium on*. IEEE Computer Society, 2013, pp. 210–217.

[26] U. I. Cheema and A. A. Khokhar, "A high performance architecture for computing burrows-wheeler transform on FPGAs." in *ReConFig*, 2013, pp. 1–6.

[27] C. B. Olson, M. Kim, C. Clauson, B. Kogon, C. Ebeling, S. Hauck, and W. L. Ruzzo, "Hardware acceleration of short read mapping," in *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*. IEEE Computer Society, 2012, pp. 161–168.

[28] O. Knodel, T. B. Preußer, and R. G. Spallek, "Next-generation massively parallel short-read mapping on FPGAs," in *22nd IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP 2011)*, Sep 2011.

[29] Xilinx, "Virtex-6 libraries guide for HDL designs," http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_3/virtex6_hdl.pdf, 2010.