Next-Generation Massively Parallel Short-Read Mapping on FPGAs

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

***Abstract*—The mapping of DNA sequences to huge genome databases is an essential analysis task in modern molecular biology. Having linearized reference genomes available, the alignment of short DNA reads obtained from the sequencing of an individual genome against such a database provides a powerful diagnostic and analysis tool. In essence, this task amounts to a simple string search tolerating a certain number of mismatches to account for the diversity of individuals. The complexity of this process arises from the sheer size of the reference genome. It is further amplified by current next-generation sequencing technologies, which produce a huge number of increasingly short reads. These short reads hurt established alignment heuristics like BLAST severely.**

This paper proposes an FPGA-based custom computation, which performs the alignment of short DNA reads in a timely manner by the use of tremendous concurrency for reasonable costs. The special measures to achieve an extremely efficient and compact mapping of the computation to a Xilinx FPGA architecture are described. The presented approach also surpasses all software heuristics in the quality of its results. It guarantees to find *all* alignment locations of a read in the database while also allowing a freely adjustable character mismatch threshold. On the contrary, advanced fast alignment heuristics like Bowtie and Maq can only tolerate small mismatch maximums with a quick deterioration of the probability to detect existing valid alignments. The performance comparison with these widely used software tools also demonstrates that the proposed FPGA computation achieves its guaranteed exact results in very competitive time.

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

I. Introduction

Sequence alignment or mapping is a fundamental aspect in modern molecular biology. Complete sequencing of an organism gives the possibility to explore and understand genetic diseases and recent cancer genomes. The first step towards the sequencing of an organism is the comparison of its genome with large databases of identical or highly similar reference genomes. These databases typically contain many billion base pairs (bp) and have a data size of several GByte.

Searches in such databases must allow mismatches and the insertion of gaps. The alphabet for nucleotides consists of the four symbols A, G, T and C. The result of a mapping is the position of the sequence in the reference genome, with as few as possible mismatches and gaps.

The algorithms and tools used most frequently are Smith-Waterman [1] and BLAST [2]. These methods were developed for and work efficiently with sequences of 1,000 and more base pairs.

The Smith-Waterman algorithm calculates a matrix of possible matching scores and thereby finds the optimal alignment. Most implementations scan the score matrix on-the-fly during the iterative computation of its elements. Yet, the computational complexity is determined by the size of this matrix, whose dimensions correspond to the lengths of the aligned sequences. While Smith-Waterman is very feasible for the alignment of two short sequences, it renders very expensive for the mapping against a large database. It is still the algorithm of choice if exact results that enumerate *every* single valid alignment location are required.

BLAST takes a heuristic approach based on the search of short seeds within the database. It then attempts to form an alignment by combining and extending these subsequences to resemble the sought sequence closely. BLAST reaches high speeds for the mapping of long sequences in large databases and is the standard tool for the mapping of genomes today.

New next-generation sequencers by Applied Biosystems, Illumina, Roche and other companies produce millions of short sequences called *reads* with a length of 30–100 base pairs in a single run. Each complete sequencing involves many of these runs [3], [4]. For example, a recent cancer genome sequencing project [5] generated nearly 8 billion reads in 123 sequencing runs.

Traditional mapping tools neither scale up to the much greater number of reads nor do they scale down to the short sequence length. This issue is known as the short-read mapping problem. While BLAST is generally struggling with such short reads, the Smith-Waterman algorithm is simply too slow and needs too much memory to process millions of them.

Aligning the reads of an experiment can take hours and days with conventional software tools [6]. Therefore, new approaches were devised to cope efficiently with large amounts of data and short reads. These are called short-read mappers. Due to the processed short reads, they typically only allow a small number of mismatches and do not tolerate gaps [6].

II. Related Work

A. Software Short-Read Mappers

Several software mappers use searching heuristics in order to cope with the mapping of short reads. Prominent examples of such short-read mappers are Bowtie [7], SOAP2 [8], Maq [9], PASS [10] and RazerS [11].

Like many other new short-read mappers, Bowtie and SOAP2 operate in two steps. The first step is a transformation of the reference database by the Burrows-Wheeler transform. In the second step, the program searches exact matches using a Ferragina-Manzini index [12]. This search for perfect matches is highly efficient and very fast as it only searches parts of the database. Mismatches, however, can only be tolerated through expensive backtracking using trial and error.

Maq and PASS are based on spaced-seed indexing. For each database position, the following fixed-length sequence of base pairs is chopped into seeds, which are stored in a large lookup table. A read to be matched is also chopped. While some pieces must match the seeds in the lookup table exactly, Maq tolerates mismatches by allowing unmatched space between matching seeds. Maq regularly returns mapping 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 short-read mappers and is in most cases much slower than a search in a Burrows-Wheeler transformed database.

The RazerS tool uses the *q-gram counting* strategy [13] in a first step to locate potential match regions in the reference genome. In the second step, these regions of interest are 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 match all possible locations. They typically detect only about 70%–90% of the reads and not all of their valid mapping locations [14].

B. Alignment on Parallel Platforms

There are many different FPGA and GPU implementations of the Smith-Waterman algorithm [15]–[17] and BLAST [18], [19]. Yet, there are still no efficient solutions for the short-read mapping problem achieving a speed comparable to the software mappers.

Their irregularity in structure and memory access patterns makes it difficult to adopt the heuristic software mapping approaches in FPGA or GPU implementations. Thus, it appears more promising to exploit the regularity of simpler approaches like Smith-Waterman to achieve a larger fine-grained concurrency instead.

On the GPU, we implemented a simplified Smith-Waterman algorithm (GPUrSWA) in CUDA. As a designated short-read mapping tool, it does not tolerate indels. The calculation of the Smith-Waterman matrix is limited to the diagonal, reducing the number of necessary calculations. If the score reaches a given threshold the position of the read in the database is saved. The lack of gaps eliminates the need for backtracking on matrix path information. Every read is processed by an individual multiprocessor to achieve a high throughput by parallel processing of individual reads on the GPU. The data dependencies in the Smith-Waterman matrix are solved through a simple diagonal computation of the cells as in [17].

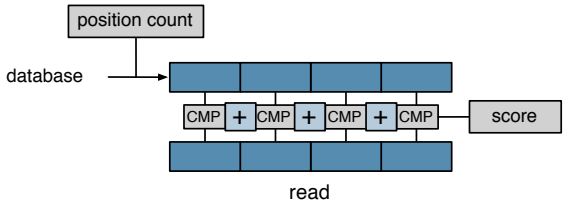

Figure 1. Naive string search algorithm with a shift register and a mismatch counter. With every cycle the database is shifted by one position.

Compared to the established Smith-Waterman tool SSEARCH, this implementation achieved a speedup of 10. Having this baseline implementation for a highly parallel platform, we set out to explore the potentials of an FPGA implementation. The derived design and its evaluation are detailed in the remainder of this paper.

III. Implementation

Our implementation uses a straightforward approach to search for short reads in a reference database. The algorithm is similar to the naive string search algorithm. Each read is compared sequentially to every database segment of the same length. In each cycle, the currently visible database segment will be shifted by one base-pair position. Fig. 1 depicts this basic approach. Its hardware implementation can easily match the complete read and count the mismatching locations in parallel. An arbitrary number of mismatches can be tolerated by a freely adjustable threshold triggering the logging of the examined database position. This design meets the demands of the short-read mapping very well.

The following subsections detail the implementation of this design on a Xilinx Virtex-6 XC6VLX240T FPGA device on an ML605 Evaluation Board.

A. Top Layer

Fig. 2 shows the top-level block diagram of the FPGA design. To achieve a high speed on the FPGA, it is necessary to provide many parallel modules. All of these work synchronously on the same database segment, which is stored in a global shift register. With each clock cycle, a new base pair encoded in 2 bits is loaded into this database register. With a system clock of 200 MHz, the required data rate is exactly 400 MBit/s.

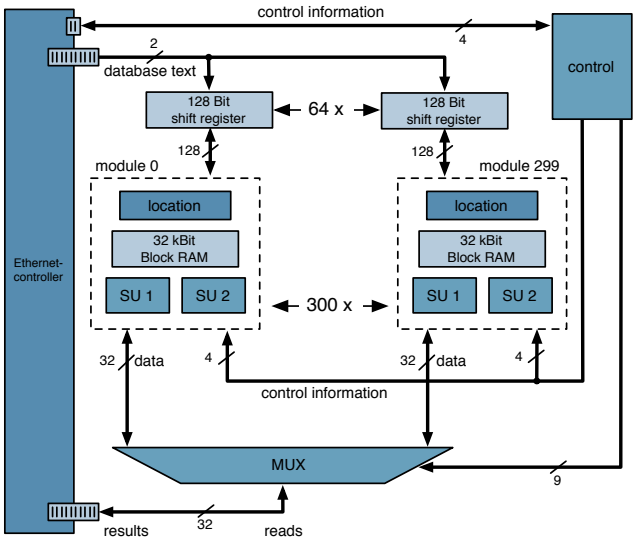

Figure 2. Basic structure of the top layer with search modules, database shift-registers and Ethernet controller.

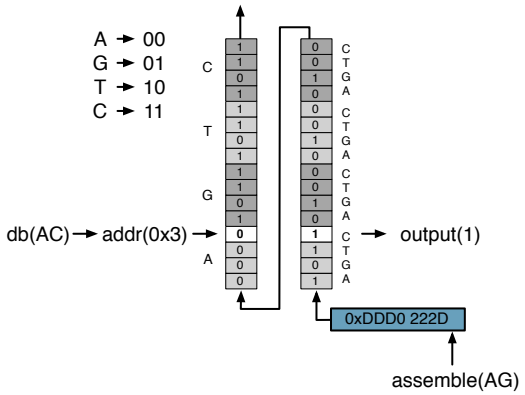

Figure 3. For the two base pairs of the read *AG*, the hexadecimal value *0xDDD0 222D* is configured into one Virtex-6 LUT. The database symbols *AC* address this LUT memory at position *0x3* to produce the output *01* signifying one mismatching base pair.

The simplest way to transfer the database from the host computer to the FPGA is by Gigabit Ethernet. Its data rate is high enough to stream the database directly from the host to the shift register. Thus, it is not necessary to store the database locally on the FPGA board. Because of that, the implementation is fully independent from the size of the database.

Like the database register, the position counter is accessible globally for all modules to reduce the resource demand.

To avoid concurrent memory accesses to the same memory, each module has its own local Block RAM to log the matching locations. This approach guarantees that every module can save its matches without conflicts with other ones. The word size of the Block RAM is configured to 32 bits, which allows to capture one 32-bit position within a single clock cycle.

On a Virtex-6 XC6VLX240T, 300 parallel modules can be realized. Each of them contains two search units, which can be operated in two different modes: They either match two short sequences of up to 64 base pairs independently, or they together match one double-length sequence of up to 128 base pairs. In the former case, the two ports of the dual-port Block RAM allow the search units to utilize the module memory without mutual interference as long as the occupied address ranges are kept disjoint.

B. Search Unit

A search unit must implement three essential functions:

1) store the read to match,
2) compare it to the current database segment, and
3) count the mismatches to trigger the match logging.

A straightforward approach could store the reads within registers and compare their base sequences directly with the current database segment. We propose an implementation that merges both of these beneficially by using distributed LUT RAM. While the RAM contents is initialized to a specific read in shift register (SRL) mode, the configured RAM produces local mismatch counts directly. The initialization of the RAM requires a recoding of the read information from its base pair representation into a sequence of functions mapping a part of the current database segment to a count of its contained mismatches. This read transformation is performed off-chip in software.

The LUT RAMs operate as a bunch of 16×2-ROMs, each of which directly yields the number of mismatches for an input of two adjacent base pairs applied together as a 4-bit address. Spanning two base pairs, the individual mismatch counts are, at most, two (2) so that they can be encoded in the two-digit binary result. If a read is shorter than the 64 base pairs supported by the hardware design, the extraneous locations are configured to always contribute a result of zero (0) mismatches. The configuration and operation of a single SRL 16×2 is summarized in Fig. 3.

The serial configuration of this memory utilizes designated shift lines available on the device. No general-purpose slice registers are occupied. And, last but not least, all of the LUT inputs can be designated to the continuously updated database segment since the reference read is encoded internally in the LUT configuration. Thus, also the combinational comparison circuit is compacted significantly. This extremely beneficial utilization of the available FPGA hardware is achieved by the use of the low-level `CFGLUT5` device primitive available for Virtex-5, Spartan-6 and Virtex-6 architectures [20]. It allows the use of a single 6-input LUT in a fashion comparable to a pair of two chained `SRL16` primitives of older Xilinx architectures.

Fig. 4 outlines a single search unit for a read of 64 base pairs. The 32 SRLs of a search unit are loaded in

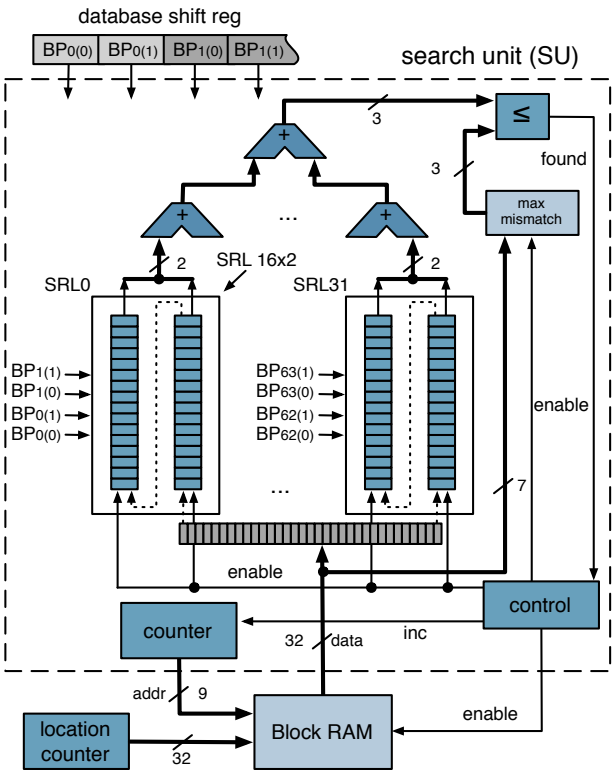

Figure 4. One search unit with 32 configuration LUT RAMs chained in a shift register configuration (SRL). These encode a read with up to 64 base pairs. Each of these SRL 16×2s covers two base pairs of the read. These are applied as 4-bit address to retrieve the corresponding mismatch count from the configured memory content. The tree combining the outputs of all the 32 SRLs computes the total mismatch count across the full read. If the result stays at or below a configured threshold, the corresponding database position will be logged into the local Block RAM.

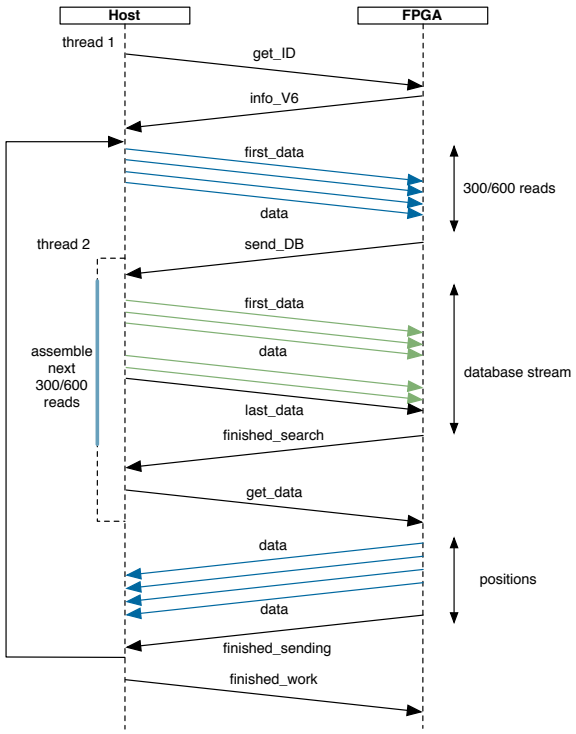

Figure 5. Communication between Host and FPGA. On the host, thread 1 manages the communication and thread 2 assembles the reads for the next iteration.

parallel through independent shift chains, which are fed from the 32-bit data output of the local Block RAM. This serves as a temporary configuration buffer when new reads are programmed into the device. Finally, the total number of mismatches for the current database segment must be computed. This is achieved by an addition tree with one pipeline stage to meet the clock rate of 200 MHz. The inputs to the tree are provided by the SRL outputs. If the total at the top lies at or below a configured threshold, the current 32-bit position of the database is logged into the local Block RAM of the search unit. The computation of the SRL configurations is performed offline in software.

C. Communication

The communication between the host and the FPGA is realized by a direct Gigabit Ethernet connection with RAW ethernet packages. To achieve a constant streaming of the database, it is important to reach a steady data rate of 400 MBit/s. The Ethernet module on the FPGA can forward received data directly to the Block RAM of a specific search unit or store it in a FIFO for the database shift register. It further recognizes the loss of packages and requests retransmissions accordingly. Fig. 5 gives a detailed overview of the communication.

D. Software

The software on the host has to transfer the reads, stream the database and receive the results as seen in Fig. 5. Each iteration can process up to 600 short or 300 double-length reads. For the processing of millions of reads, many iterations are necessary. To accelerate the streaming of the database and to achieve the targeted data rate of 400 MBit/s, the original ASCII database is transformed into a binary format, which is so reduced to one quarter of the original size. In parallel to the database streaming of one iteration, the reads of the next one are transformed into their corresponding SRL configurations.

If the local memory of a search unit is filled up, the host is signaled by the FPGA. The database streaming is interrupted so as to allow the accumulated match locations to be loaded off the device. Locations are issued in the Sequence Alignment/Map (SAM) format. [21]

The transformation and transmission of reads to the FPGA and the collection of the corresponding results require only one or two percent of the iteration run time. The remaining time is designated to the database search.

Table I
MAPPER SPEED AND ACCURACY (100,000 READS)

Mapper	One-Time DB Transform (min:sec)	Read Length: 50 bp (up to 3 errors) Search (min:sec)	Locations reported	Reads mapped
GPUrSWA	—	2866:40	2,306,253	100,000
Maq	0:08	11:00	76,874	76,800
SOAP2	4:22	0:16	163,913	76,873
PASS	—	4:32	455,862	99,336
RazerS	—	4:12	552,573	100,000
Bowtie	7:43	3:45	1,188,046	76,776
FPGA	0:07	3:44	2,306,253	100,000

IV. MEASUREMENTS AND RESULTS

We compare our FPGA implementation with the Maq, RazerS and Bowtie software heuristics and with the Smith-Waterman implementation on the Nvidia GTX 480 GPU. The latter is the only other exact algorithm besides the FPGA implementation and, thus, the only other one to guarantee to find *all* valid matching locations. The first sequence of the human genome (NCBI v.37) with 222,389,117 base pairs served as the benchmark database. The software heuristics were run on a Personal Computer with an Intel Core2 Duo CPU with 2.66 GHz and 4 GByte RAM.

Maq, Bowtie and the FPGA mapper require a one-time transformation of the database. The times required for this processing step are contrasted in the second column Tab. I. The only significant run times are observed for the Burrows-Wheeler transforms needed by SOAP2 and Bowtie. The simple translation of the database into binary format as performed for the FPGA mapper only runs for a few seconds – a negligible effort, in particular, for a one-time task.

For a performance comparison 100,000 reads were taken randomly as sequences of 50 base pairs from the reference genome and modified in up to 3 base pair locations. The number of injected errors was distributed uniformly from *0* to *3*. Maq and Bowtie were run with parameters to map as many locations as possible in acceptable time. For RazerS, the tolerance for indels was disabled so that its running time is not hurt by a feature not asked for.

The required execution times and obtained mapping results are summarized in the remaining columns of Tab. I and graphed in Fig. 6. Note that, in its current version, Maq does not identify all the matching locations of a read. It rather only outputs the number of reads that have mapped successfully somewhere. Thus, the numbers of mapped reads and reported locations coincide for this tool. All others are able to discover multiple valid mapping locations for a single read so that the number of reported locations is significantly higher than the number of mapped reads.

In terms of its running time, the FPGA mapper compares well to most software mappers. It essentially ties with Bowtie and beats RazerS and PASS clearly. Only SOAP2

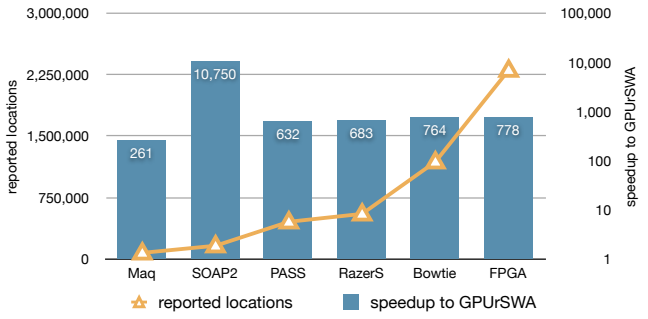

Figure 6. Comparison of the software and the FPGA mappers in terms of the speedups achieved over a GPU implementation of a simplified Smith-Waterman without indels (GPUrSWA) and in terms of the reported alignment locations. (cf. Tab. I)

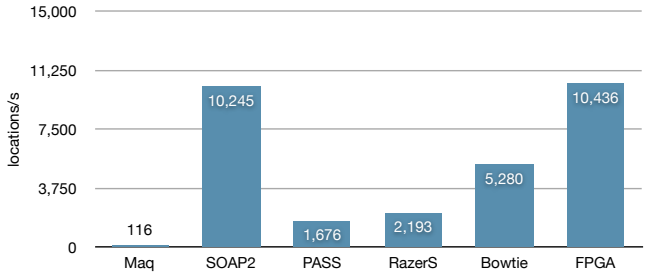

Figure 7. Number of reported locations per second between the software and the FPGA mappers. (cf. Tab. I)

achieves a significantly higher speedup. Maq is already significantly slower than these five. Although the Smith-Waterman algorithm was accelerated by the use of the GPU by an order of magnitude, it cannot keep up with any of the other mappers. It pays a high price for its computation of exact results. Only the FPGA implementation is able to combine exactness with high performance.

Regarding the quality of the obtained results, observe that Maq, Bowtie and SOAP2 fail to find any mapping for more than 20% of the reads although they are typically reported to achieve mapping rates of up to 90% for real experimental data. It turned out that the observed mapping rate is very sensitive to the actual number of errors contained in the reads. Even if the algorithms are configured to tolerate up to three mismatches, they are hardly capable to do so. Evidently, the individual genome diversity and the numbers of errors introduced during the experimental DNA sequencing are still rather small so that these mappers are acceptable in practical settings. Should future sequencing technologies, however, trade some accuracy for more speed, these software algorithms will no longer be useful.

It is noteworthy that RazerS is able to find, at least, one location for every read. However, it largely fails on alternative valid mapping locations. It only reports 20%

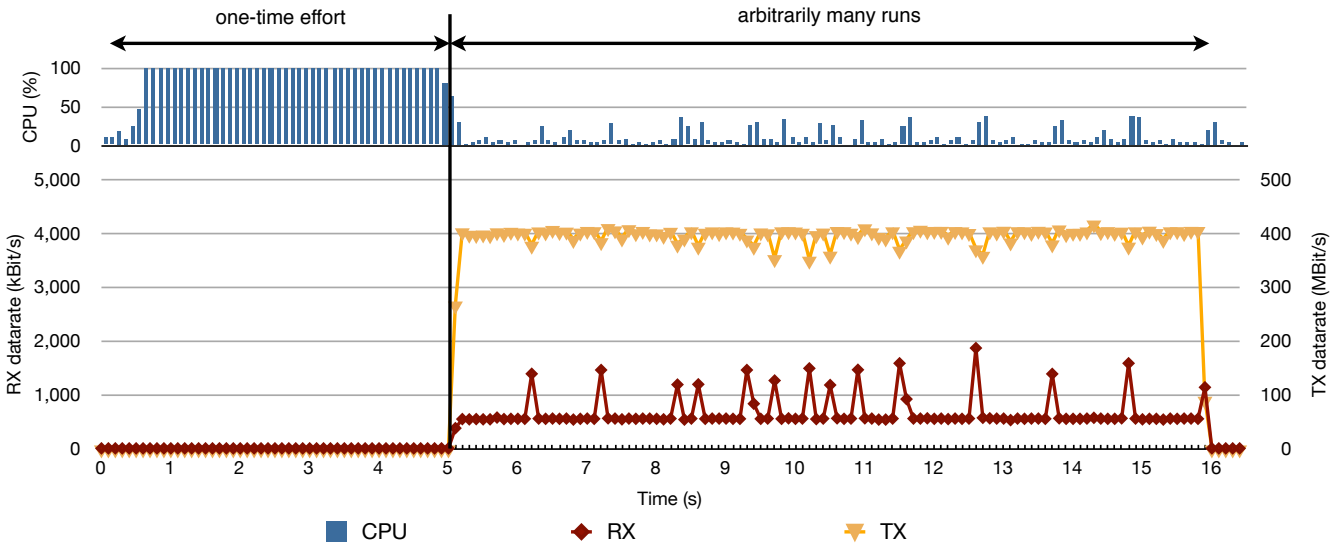

Figure 8. CPU utilization and induced network data transfer rate for the search of 6,000 reads in a database of 222,389,117 base pairs. The database transformation with the high CPU utilization is required only once per database. The CPU utilization by the on-the-fly transformation of the reads for the alignment is minor and shows short peaks of, at most, 30 percent.

of the existing locations, which is only half the number discovered by Bowtie, which reports 50% of them. SOAP2 finds only 14% of the existing locations, which is only a few more than Maq. Recall in contrast that both the Smith-Waterman algorithm and the FPGA mapper, guarantee a 100% mapping coverage within the configured mismatch threshold by design.

Fig. 7 illustrates the performance of the different mappers in terms of the total number of locations reported for each second of running time. This metric accounts for both speed and quality of the results. It clearly demonstrates the FPGA implementation to be superior to the software heuristics. Solely SOAP2 is able to gain a comparable score. Already Bowtie, the second best software heuristic, only achieves half of it. The scores further degrade steeply for RazerS, PASS and Maq, in this order.

Finally, we have traced the resource occupation of a run of the FPGA mapper. As shown in Fig. 8, the CPU is mainly needed for the one-time transformation of the database. During actual searches, its utilization is rather low as it only needs to transform and transfer the reads and collect the results. In contrast to software heuristics, a regular desktop PC will suffice and remain usable during a mapping run.

V. Conclusions

This paper has described and evaluated a short-read mapper implemented on an FPGA. This mapper does not rely on any search heuristics but achieves exact results with a competitive performance by the exploitation of massive concurrency in hardware. This is enabled by an extremely efficient low-level mapping of the basic comparators blocks to the available FPGA hardware.

The most noteworthy property of the implemented FPGA mapper is the quality of its results. In contrast to software heuristics, which regularly fail to map reads even with only a few mismatches, the FPGA mapper guarantees a 100% mapping rate within the configured mismatch threshold. Moreso, this threshold can be configured arbitrarily without any performance penalty. Additionally, the FPGA mapper can be configured to report the closest match for a given read without any pre-defined mismatch threshold.

The computation of guaranteed correct and complete mapping results by the FPGA mapper makes it a valuable baseline implementation for the evaluation of established and new software heuristics. Current comparisons between different software implementation are mostly relative lacking an objective judgment of the achieved absolute quality. Thus, also the development of software mappers benefits from the availability of our approach.

The performance of the FPGA mapper grows proportionally with the number of available search units. The design scales well and easily benefits from the continuing growth of FPGA devices. Ongoing research with a careful redesign of the search units indicates that the number of implementable units and their operating frequency can be pushed by about 50% each. This promises an internal performance boost by a factor above two. Also, multi-FPGA systems are evaluated for increased concurrency beyond the chip boundaries. Finally, designated solid-state drives for an FPGA-attached database storage are considered to increase the independence from the host computer.

REFERENCES

[1] T. Smith and M. Waterman, "Identification of common molecular subsequences," *Journal of molecular biology*, vol. 147, no. 1, pp. 195–197, 1981.

[2] S. Atschul, W. Gish, W. Miller, E. Myers, and D. Lipman, "Basic local alignment search tool," *J. Mol. Biol*, vol. 215, pp. 403–410, 1990.

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

[4] M. Margulies, M. Egholm, W. Altman, S. Attiya, J. Bader, L. Bemben, J. Berka, M. Braverman, Y. Chen, Z. Chen *et al.*, "Genome sequencing in microfabricated high-density picolitre reactors," *Nature*, vol. 437, no. 7057, pp. 376–380, 2005.

[5] T. Ley, E. Mardis, L. Ding, B. Fulton, M. McLellan, K. Chen, D. Dooling, B. Dunford-Shore, S. McGrath, M. Hickenbotham *et al.*, "Dna sequencing of a cytogenetically normal acute myeloid leukaemia genome," *Nature*, vol. 456, no. 7218, pp. 66–72, 2008.

[6] E. Mardis, "The impact of next-generation sequencing technology on genetics," *Trends in Genetics*, vol. 24, no. 3, pp. 133–141, 2008.

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

[8] R. Li, C. Yu, Y. Li, T. Lam, S. Yiu, K. Kristiansen, and J. Wang, "Soap2: an improved ultrafast tool for short read alignment," *Bioinformatics*, vol. 25, no. 15, p. 1966, 2009.

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

[10] 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, p. 967, 2009.

[11] D. Weese, A. Emde, T. Rausch, A. Döring, and K. Reinert, "Razers - fast read mapping with sensitivity control," *Genome research*, vol. 19, no. 9, p. 1646, 2009.

[12] P. Ferragina and G. Manzini, "Opportunistic data structures with applications," in *focs*. Published by the IEEE Computer Society, 2000, p. 390.

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

[14] C. Trapnell and S. Salzberg, "How to map billions of short reads onto genomes." *Nature biotechnology*, vol. 27, no. 5, p. 455, 2009.

[15] 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.

[16] 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*. ACM, 2005, pp. 229–237.

[17] S. 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, 2008.

[18] M. Herbordt, J. Model, B. Sukhwani, Y. Gu, and T. VanCourt, "Single pass streaming blast on fpgas," *Parallel computing*, 2007.

[19] E. Sotiriades, C. Kozanitis, and A. Dollas, "Fpga based architecture for dna sequence comparison and database search," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*. IEEE, 2006, pp. 8–pp.

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

[21] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, and R. Durbin, "The sequence alignment/map format and samtools," *Bioinformatics*, vol. 25, no. 16, p. 2078, 2009.