Energy Efficiency of Sequence Alignment Tools – Software and Hardware Perspectives

Michał Kierzynka[1,2,*], Lars Kosmann[3], Micha vor dem Berge[4], Stefan Krupop[4], Jens Hagemeyer[5], René Griessl[5], Meysam Peykanu[5], Ariel Oleksiak[1,2]

[1]Poznań Supercomputing and Networking Center, Poland
[2]Poznań University of Technology, Poland
[3]OFFIS, Oldenburg, Germany
[4]Christmann Informationstechnik + Medien GmbH Co. KG, Germany
[5]CITEC, Bielefeld University, Germany
[*]corresponding author: michal.kierzynka@man.poznan.pl

Abstract

Pairwise sequence alignment is ubiquitous in modern bioinformatics. It may be performed either explicitly, e.g. to find the most similar sequences in a database, or implicitly as a hidden building block of more complex methods, e.g. for reads mapping. The alignment algorithms have been widely investigated over the last few years, mainly with respect to their speed. However, no attention was given to their energy efficiency, which is becoming critical in high performance computing and cloud environment. We compare the energy efficiency of the most established software tools performing exact pairwise sequence alignment on various computational architectures: CPU, GPU and Intel Xeon Phi. The results show that the energy consumption may differ as much as nearly 5 times. Substantial differences are reported even for different implementations running on the same hardware. Moreover, we present an FPGA implementation of one of the tested tools – G-DNA, and show how it outperforms all the others on the energy efficiency front. Finally, some details regarding the special RECS®|Box servers used in our study are outlined. This hardware is designed and manufactured within the FiPS project by the Bielefeld University and Christmann Informationstechnik + Medien with a special purpose to deliver highly heterogeneous computational environment supporting energy efficiency and green ICT.

Key words: sequence alignment, energy efficiency, FiPS project, heterogeneous hardware, bioinformatics, FPGA

1 Introduction

Sequence alignment is one of the most common and the most frequently applied operations in computational biology. This statement becomes even more authorized if we realize that many higher level algorithms, e.g. sequence mapping or phylogenetic tree construction, use this simple operation as a building block. There are a number of algorithms performing the alignment procedure, both heuristic and exact. The former were often used in the past when limited computational power was the main limiting factor. On the other hand, exact algorithms have become more popular in the recent years, mainly with the advent of high performance software implementations. However, the exponential growth of the number of sequences in databases and from individual experiments is still posing a real challenge, especially in the context of Next Generation Sequencing (NGS). Moreover, the increase in the amount of data is faster than the rate of improvement of microprocessors. Therefore, scientists are constantly working on ways to improve the existing software tools. This also includes the research associated with different hardware architectures which was deeply explored in the recent years. The most popular accelerating hardware in this area is probably GPU (graphics processing unit), with multiple implementations of different alignment scenarios, e.g. [1, 2, 3, 4, 5]. Other hardware architectures that were found useful include: FPGAs (field-programmable gate arrays) [6, 7], IBM's Cell BE [8, 9], Intel Xeon Phi [10] alongside with traditional CPUs (central processing units) with their SIMD capabilities [11, 12].

One common goal of all the high-throughput implementations of sequence alignment is obviously to maximize the performance. Therefore, scientists tend to compare their software tools in this respect. However, from a data center or cloud provider perspective it is also crucial to maximize the performance achieved per energy used, i.e. the energy efficiency of the software. From this standpoint, sequence alignment is a prime example to look at. First, because it is ubiquitous in modern bioinformatics analysis, which nowadays is carried out more frequently in high performance computing (HPC) centers. Second, as there are already multiple high quality implementations of this building block algorithm on multiple architectures. In this paper we compare the energy efficiency of various state-of-the-art pairwise sequence alignment tools alongside with their pure performance. The results may be of key importance not only for those managing data centers, but also for scientists implementing complex processing pipelines and programmers starting to consider their platform of choice. One of the goals of this work is also to raise awareness within the community about the energy-efficiency driven development and tools selection.

An important aspect is also the fact that this research was conducted as part of FiPS – an EU-founded project entitled: "Developing Hardware and Design Methodologies for Heterogeneous Low Power Field Programmable Servers". The main goal of the project is to develop highly heterogeneous low power servers (called RECS$^{®}$|Box) for HPC centers and cloud applications, with a special emphasis placed onto FPGA modules. Alongside with the hardware, the project partners have developed tools helping the programmers to port their applications to heterogeneous environment [13]. Interestingly, the hardware is developed in close collaboration with application partners, who adapt their domain-specific software tools to achieve more energy-efficient implementations. One of the applications that are considered in the project is G-DNA [5] – a GPU-based software for pairwise sequence alignment. We investigate whether any improvement in its energy efficiency is possible, given that the application was already highly optimized for GPU. In particular, an FPGA implementation of this tool is proposed. Since the comparative results are very promising, the article also presents some of the experience and results achieved in this context. Furthermore, details regarding the RECS$^{®}$|Box hardware design and development are outlined too. This hardware deserves special attention as it facilitates development of energy efficient applications due to its hardware configuration and software monitoring solutions.

The rest of the paper is organized in the following way. Section 2 shortly outlines the basic idea behind sequence alignment algorithms. Section 3 first briefly presents the software tools for pairwise sequence alignment that are compared in our study, and then refers to literature related to energy efficiency. Section 4 presents the methodology and results of the comparative study. The next section describes the FPGA implementation of G-DNA as well as details regarding the architecture of energy efficient RECS®|Box hardware. Finally, conclusions are outlined in Section 6.

2 Dynamic programming algorithms

There are three basic sequence alignment methods based on the dynamic programming, namely: the Needleman-Wunsch algorithm (NW) [14] for global alignment, its semi-global version, and the Smith-Waterman algorithm (SW) [15] for local alignment. These algorithms work in a similar way, and differ only with respect to the boundary conditions. Additionally, each algorithm may compute only the so called alignment score, i.e. quantitative information about the similarity of sequences, or the full alignment by performing the backtracking step. Moreover, the algorithms may use either linear or affine gap penalties. This section explains the basic idea behind these methods on the example of NW with affine gap penalties. For more detailed analysis of alignment algorithms see [3].

2.1 The Needleman-Wunsch algorithm

Let us first define the following notation:

- A – an alphabet, i.e. a set of characters (nucleotides or amino acids),
- $s_i(k) \in A$ – k-th character of the i-th sequence,
- $SM(s_i(k) \in A, s_j(l) \in A)$ – substitution value for a given pair of characters,
- G_{open}, G_{ext} – gap opening and gap extension penalties,
- H – a matrix with partial alignment scores,
- E, F – auxiliary matrices with partial alignment scores indicating vertical and horizontal gap continuation, respectively.

The Needleman-Wunsch algorithm fills the dynamic programming matrix H according to a similarity function expressed in Formula 1. The matrix is of size $n+1 \times m+1$, where n is the number of characters in the first sequence s_1 and m – in the second sequence s_2. The similarity function is based on a score of substitution between any two characters which is typically defined by a *substitution matrix*. Gap penalties, i.e. G_{open} and G_{ext}, are applied to reflect the cost associated with insertion/deletion mutations.

$$H_{i,j} = \max \left\{ \begin{array}{c} E_{i,j} \\ F_{i,j} \\ H_{i-1,j-1} + SM(s_1(i), s_2(j)) \end{array} \right\} \tag{1}$$

$$E_{i,j} = \max \left\{ \begin{array}{c} E_{i,j-1} - G_{ext} \\ H_{i,j-1} - G_{open} \end{array} \right\} \tag{2}$$

$$F_{i,j} = \max \left\{ \begin{array}{c} F_{i-1,j} - G_{ext} \\ H_{i-1,j} - G_{open} \end{array} \right\} \tag{3}$$

where $i = 1..n$ and $j = 1..m$. Additionally, the first row and the first column of matrix H are filled according to the following formulas:

$$H_{0,0} = 0 \tag{4}$$

$$H_{i,0} = -G_{open} - (i-1) \cdot G_{ext} \tag{5}$$

$$H_{0,j} = -G_{open} - (j-1) \cdot G_{ext} \tag{6}$$

where $i = 1..n$ and $j = 1..m$. Moreover, the first rows and columns of matrices E and F are initialized with $-\infty$.

At this point the optimal alignment score is already known and can be found in cell $H(n,m)$. However, the actual alignment of the two sequences, i.e. the relative arrangement of subsequent characters, is not known yet. This may be computed by the optional *backtracking* procedure which performs backward moves starting from $H(n,m)$ until the first cell, i.e. $H(0,0)$, is reached. A single move is performed to the neighboring cell that contributed to maximum value in Formula 1. If the algorithm moves diagonally, two characters are aligned. If the move is performed to the upper cell, a gap character is inserted into sequence s_1 in the alignment. In a similar way a gap is added to sequence s_2 every time the algorithm moves to the left.

2.2 Performance analysis

The performance of the dynamic programming implementations, including NW and SW, is usually defined in the number of cell updates per second (CUPS). It refers to the number of cells in matrix H that can be calculated in one second, including the time needed to perform all side operations like computation of matrices E and F or performing the backtracking procedure, if needed. The performance can be easily calculated if we divide the number of cells computed in matrix H by the overall runtime of the algorithm:

$$\frac{\sum_{i=1}^{n} length_{ai} \cdot length_{bi}}{t \cdot 10^9} \; [GCUPS] \tag{7}$$

where n is the number of pairwise alignments to perform, $length_{ai}$ and $length_{bi}$ are the lengths of the corresponding sequences to align, t represents the time in seconds and the result is given in giga (10^9) CUPS.

3 Related work

3.1 Sequence alignment tools

One of the main goals of this paper is to compare the energy efficiency of state-of-the-art algorithms performing pairwise sequence alignment. In most cases this kind of software is designed to perform a database scan, i.e. one query sequence is aligned to all the sequences from a database in order to find the best scoring matches [4, 10, 11, 12]. Other scenarios are possible as well, e.g. when all sequences are aligned with all the others [3, 16], or when only selected pairs of sequences are aligned [5], or when the alignment concerns very long sequences [17], etc. Although the underlying algorithm may always be the same, e.g. the Smith-Waterman algorithm, the parallelization and optimization techniques as well as the resulting performance may differ depending on the scenario. In order to provide a fair comparison, we decided to choose the widest class of tools currently used, i.e. those implementations that compute the alignment score between relatively short sequences (possibly from NGS) using the exact algorithm based on dynamic programming. These kind of tools are often used for protein database scans, or constitute a common building block of more complex algorithms, e.g. for DNA *de novo* assembly [18], reads mapping [19], or multiple sequence alignment [20]. Furthermore, we chose to focus on the implementations based on the idea of dynamic programming, as they may be fairly compared with respect to power consumption and efficiency. Heuristics would obviously result in higher performance, but then there is the question about quality of alignment, which is out of the scope of this work. The following paragraphs summarize the key features of tools selected for our comparison. Note that all of them are freely available and open source.

SSEARCH tool from the package FASTA version 36.3.7 implements the SSE2 accelerated Smith-Waterman search developed by Michael Farrar in 2007 [11]. The algorithm is parallelized using Single-Instruction Multiple-Data (SIMD) instructions. For this reason it was long considered as the fastest tool when it comes to CPU implementations. SSEARCH addresses the protein database search scenario in which the algorithm is further parallelized to align multiple pairs of sequences using multiple CPU cores.

Another CPU-based implementation of Smith-Waterman algorithm is SWIPE developed by Torbjørn Rognes in 2011 [12]. It also uses SIMD vector processing capabilities, but this time requires the SSSE3 instruction set to be available in the processor. Depending on the length of query sequence the software was reported to run up to six times faster than Farrar's approach.

CUDASW++ 3.0 [4] is a fast Smith-Waterman protein database search algorithm which utilizes both CPU and GPU SIMD instructions to carry out the computations. On the GPU side it makes use of CUDA PTX SIMD video instructions (available in the NVIDIA Kepler architecture) to achieve more data parallelism as compared to the SIMT execution model. On the CPU side the software uses the SWIPE implementation to boost the performance.

G-DNA [5] was designed to perform alignment of selected pairs of nucleotide sequences, which is a common scenario in *de novo* assembly or reads mapping. The software implements a semi-global version of the Needleman-Wunsch algorithm and uses multiple GPUs to compute the score and shift for each selected pair of sequences. This implementation was analyzed by the FiPS consortium for further improvements in energy efficiency.

Finally, SWAPHI [10] is a parallel algorithm accelerating the Smith-Waterman protein database search on Intel Xeon Phi coprocessors. The high speed of pairwise alignment is achieved by effective utilization of both coarse-grained and fine-grained parallelism coming from many processing cores and from the 512-bit wide SIMD vectors, respectively.

All the above-mentioned implementations report exceptional performance using either CPU, GPU or Intel Xeon Phi. Some of them are often used in research institutes and data centers around the world. However, none of the authors has reported the energy efficiency of their implementation. No such comparison was made in literature either. This certainly requires investigation which is presented in this paper.

3.2 Work related to energy efficiency

A key part of this work is to present the energy efficiency aspect of the sequence alignment problem, and how it may be further improved with the FPGA architecture. Interestingly, attempts to accelerate the execution of sequence alignment with the use of FPGA devices started earlier than any other optimization strategies involving GPUs or SIMD instructions available in CPUs. There were also several papers published in this area. One example may be [21], which presents an implementation of the Smith-Waterman algorithm on FPGA. The authors were able to achieve up to 11 GCUPS (for a specific sequence length) back in 2004, at the time when standard CPU implementations were running at around 50 MCUPS. However, the energy efficiency aspect was not presented. Similar results were reported one year later in [22], this time though on more realistic data sets. Another attempt was made in 2007 [6]. Although the authors claim they achieved 160-fold speedup, the actual performance of the FPGA implementation was only around 25 MCUPS. In [23] in turn the authors implemented an equivalent of SSEARCH in version 35 from the FASTA package using Intel Accelerator Abstraction Layer achieving up to 9 GCUPS. More recently, in 2013, another promising result was presented [7] with an implementation on Xilinx Virtex-7 FPGA achieving 40-fold speedup as compared to a base non-optimized CPU code. Unfortunately, no information was provided about the performance expressed in the number of cell updates per second (CUPS). Summing up, none of the above-mentioned papers addresses the problem of power consumption, including the recent review article about sequence alignment on FPGA [24]. Instead, what all those papers have in common is the recommendation for FPGA as an interesting platform for sequence alignment.

One paper that focuses on the energy efficiency of an FPGA implementation of the Smith-Waterman algorithm was published in 2012 [25]. The authors presented power consumption and performance of their implementation depending on the utilization of processing elements (PEs). This was achieved by scaling the linear systolic array design for various numbers of PEs and measuring the dynamic power and performance values. The results demonstrate that initially the performance per unit Watt increased with growing number of PEs, but stabilized and finally decreased after further increasing of the number of PEs. The maximum performance achieved per unit Watt was between 200 and 250 MCUPS/Watt and the total performance within this power efficiency was around 7.6 GCUPS. Nevertheless, no comparison was made to other state-of-the-art implementations.

However, a paper that deserves special attention was only recently published in 2015 [26].

The authors compare their implementation of the Smith-Waterman algorithm on Zynq SoC 7100 FPGA to two other embedded processors, namely: ARM Cortex-A9 running Farrar's implementation [11] and application-specific processor (ASIP) running BioBlaze [7]. In addition, a desktop Intel Core i7 CPU also running Farrar's implementation was used for reference. The energy efficiency was measured in Mega Cell Updates per Joule (MCUPJ). With 460 MCUPJ the described FPGA implementation achieved 3.18x and 1.25x better energy efficiency than the ARM Cortex-A9 and the dedicated ASIP processor, respectively. Surprisingly, the authors claim to achieve 6.95x better energy efficiency compared to the code running on Intel i7, which is not consistent with our experiments (cf. Section 4). Furthermore, our tests show that the state-of-the-art alignment algorithms are in most cases more energy efficient than the implementation in question and are also more powerful when it comes to pure performance.

Another interesting work in this area was published in 2012 [27]. The authors carried out a comparative study between three different acceleration technologies: FPGAs, GPU as well as IBM's Cell BE, and compared them to a traditional CPU. Comparison criteria included speed, energy consumption, and purchase and development costs. In order to compare the development time the authors prepared their own implementations of the Smith-Waterman algorithm on all these hardware architectures. However, judging by the performance achieved, they were far from state-of-the-art. As a result, although the paper presents an interesting glimpse into the costs associated with each type of hardware, the results may be misleading due to the human factor (development time and poor performance achieved). On the contrary, by presenting the performance and energy efficiency of the top-class implementations in this article, we should eliminate this issue and make a more objective comparison. Finally, it is worth noting that according to the study presented in [27], FPGAs outperform all the other platforms on performance per watt criterion, which once again is a promising results in favor of reconfigurable hardware. For this reason, we decided to investigate whether it is possible to further improve the energy efficiency of G-DNA – one of our review implementations, using the FPGA architecture and the RECS®|Box hardware developed within FiPS.

4 Energy efficiency of sequence alignment tools

4.1 Testing methodology

All selected software tools for pairwise sequence alignment (cf. Section 3.1) were tested on real-life and sufficiently large data sets. The implementations designed to perform protein database scans were tested on the latest (as for June 2015) release of UniProt [28] which is a comprehensive and freely accessible database of protein sequence and functional information and often serves as a benchmark in this kind of tests. Several lengths of query sequence were selected to provide an insight into possible impact on results. As for the software processing the nucleotide sequences, NGS data sets from Illumina HiSeq 2000 (C. elegans N2, DRA000967) and 454 GS FLX Titanium sequencers (E.coli H260, SRX079673) were used for short and long sequence tests, respectively.

For each algorithm the performance and energy consumption results presented in this section are averaged over approx. 100 measurements. In order to capture a sufficient number of measurements, each implementation was run several times. The performance is expressed in Cell Updates per Second (CUPS) which is the most common measure for algorithms based on dynamic programming. The energy efficiency is given in Cell Updates per Joule. The measurements of energy were performed using hardware-specific tools. For Intel CPUs, this was done with Intel Power Gadget using Runtime Average Power Limiting (RAPL) capability of Intel CPUs, as it allows for precise measurements. The complete support is provided only for Intel Xeon line, and therefore desktop CPUs could not be considered. To make a reasonable comparison between different hardware architectures, the power consumption was measured for the whole CPU socket as well as for memory (DRAM). The reason is that for many architectures, e.g. GPU or Intel Xeon Phi, one cannot distinguish between energy used by the processing unit and corresponding memory modules (which also use a considerable amount of energy). On the other hand, the measurements of energy taken by the whole computer system would count in the power used by devices that are not required by individual software tools in our tests. Therefore, for algorithms running on accelerators, we measured only the energy used by these devices. In the case of NVIDIA GPUs, this was achieved using NVIDIA Management Library (NVML) and the *nvidia-smi* tool. The NVIDIA Tesla product line was chosen due to its support for power measurement (GeForce gaming series are not supported). For the Intel Xeon Phi the energy measurements were done with Intel Xeon Phi Coprocessor Platform Status Panel, i.e. *micsmc* tool. As for FPGA, in our case it is integrated in the RECS®|Box system via a COM Express compute board that also holds DDR memory and additional hardware needed to perform computations. This board includes monitoring devices that were used to measure the power consumption of the integrated FPGA and peripheral devices. This is described in detail in Section 5.4. The measurement of the whole compute board energy including its necessary peripheral hardware was done to measure the whole alignment process on the platform.

One additional remark concerning the CUDASW++ 3.1 software is that by default apart from GPU it also uses the CPU-based SWIPE implementation to boost the performance. As SWIPE is tested as a separate application, we decided to run CUDASW++ 3.1 in its GPU only mode to obtain more fine-grained results and thus clearly distinguish between these two architectures.

The tests were run on the RECS®|Box hardware as well as on the *moss* computational cluster located at Poznań Supercomputing and Networking Center. The latter was chosen due to the availability of specific hardware and software combination. A brief specification of the hardware used to carry out the computational tests is given below:

- Intel Xeon E5-2670 2.60GHz, 8 $\times$ 32GB DDR3 at 1333MHz, with Hyper-Threading disabled
- NVIDIA Tesla K20m and K40c
- Intel Xeon Phi 5110P
- Xilinx Zynq XC7Z045

4.2 Comparative study

The main goal of this section is to compare the performance and energy efficiency of the state-of-the-art implementations of pairwise sequence alignment described in Section 3.1. Additionally, as the G-DNA implementation was analyzed by the FiPS consortium for further improvements in energy efficiency, the results obtained in this area are presented as well.

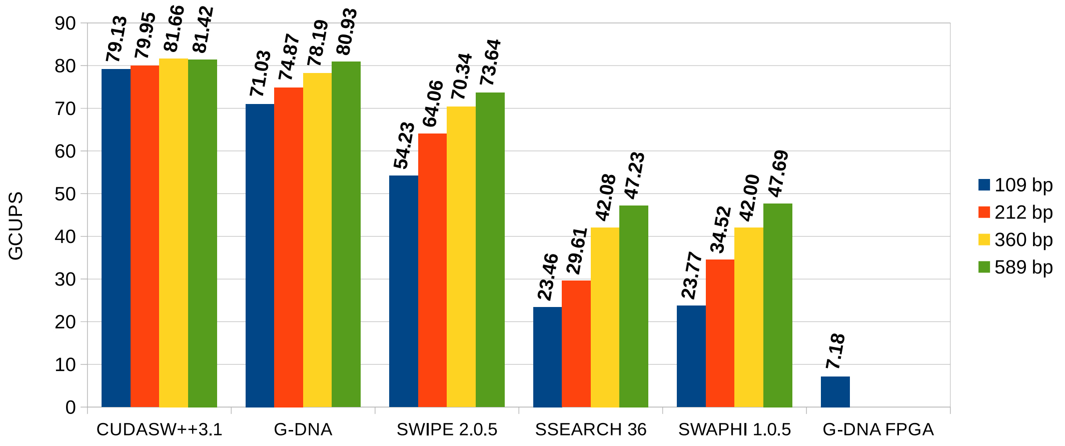

Figure 1: Average performance (in GCUPS) for each implementation depending on the sequence length (in base pairs).

Figure 1 presents the average performance of each implementation depending on the sequence length. The results are expressed in Giga Cell Updates Per Second (GCUPS). It is plain to see that the best performing implementations are the GPU ones, i.e. CUDASW++ 3.1 and G-DNA. Moreover, the performance of the former remains almost on the same level, regardless of the length of the query sequence. In contrast, the general trend is that the performance increases with growing sequence length. This is easy to explain, as the ratio of pure computations to data initialization and transfer increases with longer sequences. On the CPU front, SWIPE 2.0.5 clearly outperforms the implementation in SSEARCH 36, both utilizing all 8 CPU cores. The latter, surprisingly, has a very similar performance to SWAPHI 1.0.5 running on Intel Xeon Phi. Finally, the FPGA implementation of G-DNA supports sequences up to 112 bp only (cf. Section 5). Correspondingly, not all the sequence lengths could be tested. Nevertheless, the short sequences (around 100 – 110 bp) constitute the most common use case in many practical applications (e.g. mapping of NGS reads). When it comes to performance, with slightly more than 7 GCUPS it certainly cannot compete with the other tools. However, the comparison of pure performance is not the most important in this paper.

Figure 2 presents the average energy efficiency of each implementation depending on the sequence length. The results are expressed in Giga Cell Updates Per Joule (GCUPJ). Likewise in the case of performance, the energy efficiency of both GPU-based implementations is very good. It varies between 0.614 and 0.687 GCUPJ, as measured on NVIDIA Tesla K40 GPU. This time though G-DNA performs slightly better compared to CUDASW++ 3.1. The average energy efficiency on NVIDIA Tesla K20 was somewhat lower, and for comparison purposes is presented in Figure 3. The energy efficiency of the CPU implementations strongly correlates

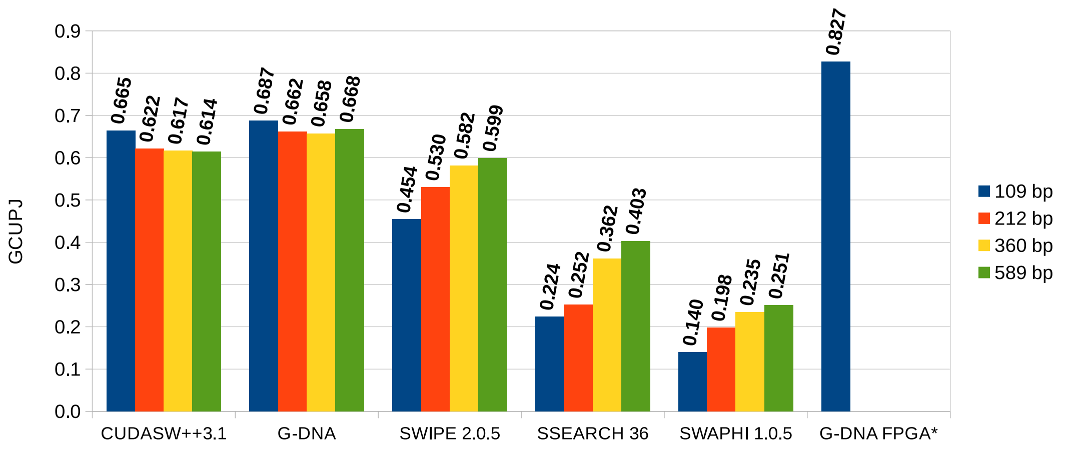

Figure 2: Average energy efficiency (in GCUPJ) for each implementation depending on the sequence length (in base pairs).

with their performance. This is due to the fact that both applications need roughly the same power to run. As a consequence SWIPE 2.0.5 is some 48 % - 110 % more energy efficient than SSEARCH 36, depending on the query length. Surprisingly, SWAPHI 1.0.5 has the lowest energy efficiency. The major contributing factor is the high power consumption of the Intel Xeon Phi coprocessor, which makes it hard to achieve good efficiency results on this platform. Interestingly, the FPGA implementation of G-DNA with its 0.827 GCUPJ outperformed all the other implementations by a large margin. This is mainly due to a very low power consumption of the board with FPGA module and a particular application to hardware fit. This results justify the wide interest in FPGA that up to now was mainly attributed to its speed (cf. Section 3.2).

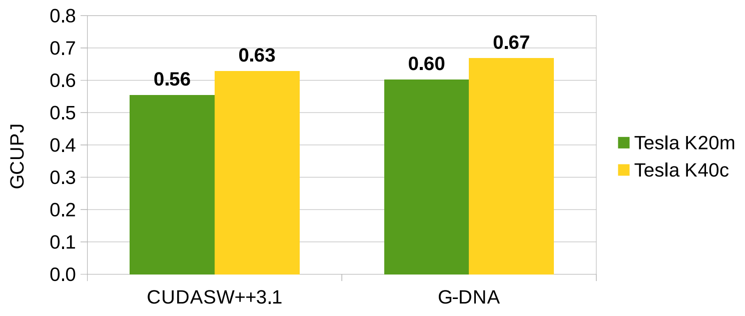

Figure 3: Average energy efficiency of the GPU-based sequence alignment tools depending on the GPU model.

Another interesting outcome of this comparison is that GPU-based implementations achieve the highest energy efficiency for short sequences whereas CPU applications strongly prefer longer sequences. Although this may be hard to explain without a deep analysis of the source code of individual applications, our observation is that the CPU power remains almost on the same level regardless of the sequence length. In contrast, the power draw of GPUs is much more sensitive with this respect and varies with sequence length.

Additionally, it is worth noting that in many cases, within a given hardware architecture, the faster software we have, the more energy efficient it is. In other words, by comparing only the performance we may usually order the software tools from the most to the least energy efficient ones. This is particularly evident in the case of CPU, whose power draw primarily depends on the number of running threads. This in general is also true for other architectures, with some minor exceptions. However, such straightforward comparison does not work when comparing implementations on different architectures. For example, a significantly faster implementation may turn out to be much less power efficient, as exemplified above. In this case, empirical comparison become indispensable. Even more difficult is a complete evaluation of cost effectiveness of individual solutions, an attempt of which is presented later on.

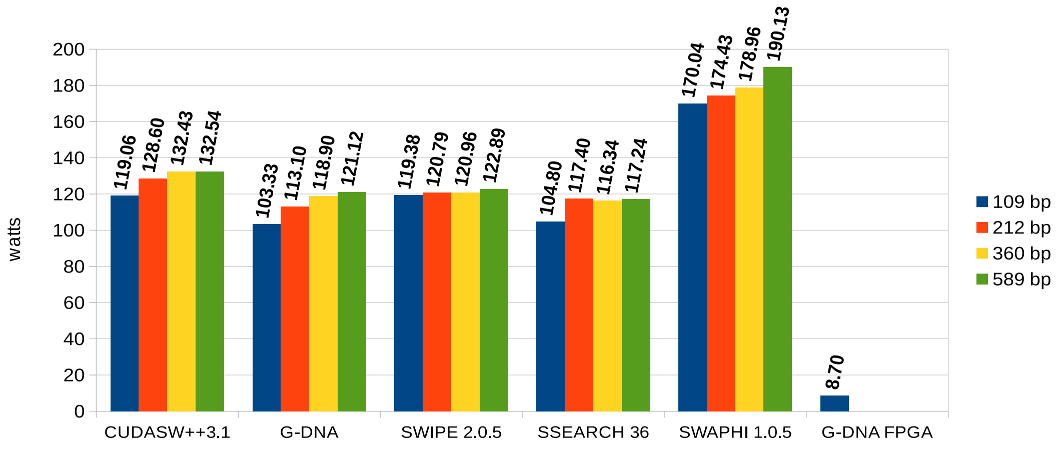

Figure 4: Average power draw (in watts) for each implementation depending on the sequence length (in base pairs).

For completeness, Figure 4 presents the average power draw of each implementation depending on the sequence length. The highest power consumption was observed in the case of Intel Xeon Phi, between 170 an 190 watts. In contrast, the board with FPGA module only required 8.7 watts to run the pairwise alignment. Both GPU and CPU have a very similar power consumption, however as pointed out, GPU is more sensitive to sequence length and demonstrates more variation in power draw. Additionally, in the case of CPU, we may distinguish between the power used by the CPU socket and the power used by memory modules. Figure 5 presents a more detailed insight into the power distribution between these two. It is also worth noting that although SWIPE utilized slightly more power in total, both CPU-based applications are comparable with respect to power consumption.

Finally, Table 1 presents the average power draw of individual processing units in their idle state. It is plain to see that the Intel Xeon Phi has much higher idle power draw as compared to the other architectures. This certainly should be considered by data centers that experience uneven server load. In the case of CPUs, although the power consumed by the CPU socket and memory may be measured separately, for the sake of justice, the total power was always used to calculate the energy efficiency, as explained in Section 4.1. Furthermore, it should

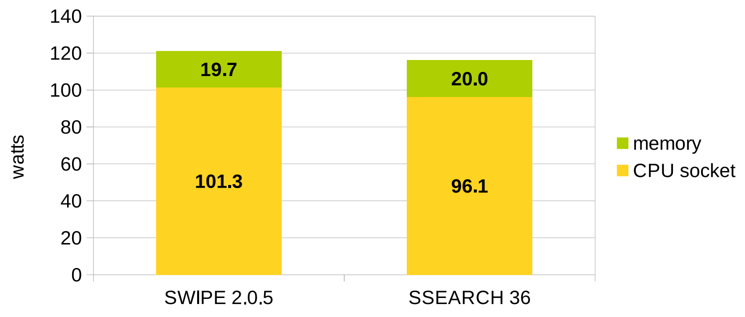

Figure 5: Average power consumed by the CPU socket and memory depending on application.

be stressed that the interpretation of the Thermal Design Power (TDP) values presented in Table 1 differ depending on the manufacturer. For Intel processors, including Xeon Phi, TDP represents the average power dissipated when operating at base frequency with all cores active under an Intel-defined, high-complexity workload. In contrast, values presented for NVIDIA GPUs refer to the actual power limits defined by the manufacturer, and are therefore less likely to be achieved by a real life application. Likewise, in the case of Xilinx the TDP value represents the maximum theoretical power of the whole board. This include the Zynq processor itself (16 W), the DRAM (3 W), the crosspoint switches (5.5 W) and other necessary hardware.

Table 1: The average power draw (in watts) of individual processing units measured in idle state. Thermal Design Power (TDP) values are given for reference. The reference prices of the hardware are as of 2015. * price calculated for large quantities, including a RECS®|Box compute board.

	total	processor	DRAM	TDP	price [€]
Intel Xeon E5-2670	7.8	4.2	3.5	115	1450
NVIDIA Tesla K20m	27.6	-	-	225	2569
NVIDIA Tesla K40c	37.2	-	-	235	2896
Intel Xeon Phi 5110P	96.5	-	-	225	2277
Xilinx Zynq XC7Z045	8.2	-	-	26	847*

4.3 Global impact and costs comparison

It is extremely difficult to gather world-wide or even country-wide statistics regarding the use of sequence alignment tools wrt. the total runtime or power used, not to mention the hardware architecture. The same concerns HPC centers, even though different alignment tools are becoming part of standard software installations. However, if we look at the map of genomic centers[1], we can easily spot over 330 genomic institutions capable of sequencing

[1] *http://omicsmaps.com* – Next Generation Genomics: World Map of High-throughput Sequencers

human genomes, possessing over 5000 sequencers for this purpose, as of 2015 (note that these numbers are growing rapidly). Therefore, in this section we attempt to provide cost estimates from a single genomic center perspective, assuming 1000 human genome mapping operations monthly. The number of alignments to perform was calculated on a real-life data (140 GB of raw sequences, each 100 bp long). In order to calculate costs we used the data obtained from our experiments and combined these with average electrical power cost in EU according to Eurostat (0.202 € / kWh, as of 2014). The results are presented in Table 2. Note that this comparison is intended only as an exemplary costs comparison, which can vary depending on a number of criteria.

Table 2: Time, hardware quantities and costs associated with running each implementation. Computational time and the number of kWh are presented for one month. For the GPU-based software NVIDIA Tesla K40 was selected.

Implementation	Computational time [h]	# devices needed	Running power [kWh]	Idle power [kWh]	5-year energy cost [€]	5-year total cost [€]
CUDASW++ 3.1	368.6	1	43.86	13.07	690.02	3586.02
G-DNA	410.6	1	42.46	11.51	654.04	3550.04
SWIPE 2.0.5	537.8	1	64.24	1.42	795.86	2245.86
SSEARCH 36	1243.3	2	130.21	1.53	1596.72	4496.72
SWAPHI 1.0.5	1227.0	2	208.33	20.55	2774.08	7328.08
G-DNA FPGA	4062.2	6	35.27	2.11	453.07	5535.07

There is no doubt that the largest fraction of costs comes from the hardware. The number of devices was calculated to keep up with a constant amount of computations per month. Therefore, the faster the implementation the fewer processing units are required (e.g. compare SWIPE with SSEARCH). Note that we count in only the price of processing units (plus compute board in the case of FPGA), to avoid server configuration preferences. As for the energy consumption, it is divided into this coming from the runtime and from the idle states. The latter is also important as the hardware most often works in servers. As a matter of fact, we can observe that the energy costs vary substantially, from 453.07 to 2774.08 €. Even though, the FPGA is the most energy efficient implementation, it is not very fast. Hence, a lot needs to be invested in the hardware. However, one needs to keep in mind that the price of electrical power is constantly rising (in EU over 52 % between 2005 and 2014, according to Eurostat), whereas the price of hardware tends to have the opposite tendency. In the comparison, all implementations are compared in their current state of the development. The FPGA-based implementation of G-DNA is based on high level synthesis, as discussed in Section 5, and consequently still has a lot of potential for optimization, while all the other implementations are mature implementations which are already quite optimized. Finally, although the most cost effective implementation can vary depending on the defined workload, the winner in our case is SWIPE running on a CPU.

5 FPGA-based implementation and hardware platform

The Needleman-Wunsch and the Smith-Waterman algorithms benefit from effective pipelining structures, and parallel processing of individual cells. Therefore, general purpose processors are somewhat limited in the possible performance that can be achieved. On the contrary, FPGAs offer flexible hardware, that can be shaped into minimal cell processing elements in a pipeline of optimal length. This performance benefit requires a specialized development. The typical workflow starts with the functional description of the algorithm, normally delivered as high level source code. Then expert knowledge is needed to port the functional code into a synthesizable implementation in a hardware description language. This leads to major transformations of the algorithm. Subsequently, a synthesis tool performs several synthesis steps to register transfer level (RTL) and down to gate level, resulting in a binary configuration file that is loaded into the FPGA and defines the exact functionality of each logic cell. During every step additional simulation is needed, to verify functionality through all abstraction layers, and functional equivalence to the initial algorithmic description.

In the last few years, so called high level synthesis (HLS) tools have become popular in hardware development. These tools rise the abstraction layer onto nearly algorithmic level. They offer implementation of the hardware description in a high level language like C, C++ or SystemC. This enables the engineer to reuse initial functional descriptions and perform minor refactoring resulting in synthesizable high level code. Nevertheless, expert knowledge is needed as in fact most tools define a subset of the programming language only which can be seen as domain specific language (DSL). Still, the opportunity to reduce porting effort and reuse prototype source code is provided.

The FiPS project has developed the FiPS flow [13] which enables the developer to identify kernel in a larger application that can be outsourced on different hardware architectures. The FiPS workflow automatically extracted and characterized the main kernel in the CPU version of the G-DNA application and additionally suggested FPGA as a possible and efficient architecture. In order to avoid manual reimplementation HLS tools were used to port the kernel to the FPGA hardware.

5.1 Processing elements

The extracted G-DNA kernel is CPU-specific and synthesis of this code without intervention results in poor FPGA area consumption and performance. Its imperative structure complicates introduction of optimization techniques during synthesis process. Therefore, a careful refactoring with respect to the given kernel structure is needed, in order to keep intervention as small as possible, but enable the synthesis tool to use hardware optimization to reduce area and improve performance.

G-DNA implements a division of the algorithm into subproblems. The alignment of nucleotides is segmented into submatrices which are calculated on their own. An outer structure organizes the submatrices calculation in order to reduce the memory effort from quadratic to linear memory consumption. In our FPGA implementation this segmentation is utilized to introduce an efficient pipelining, behaving as systolic array. Individual submatrices are implemented as processing elements (PEs). Such pipelining idea has already been known in literature for some time, except that a single PEe usually implements a single matrix cell, i.e.

alignment of two nucleotides only. Therefore, in our approach individual PEs perform much more calculations.

Contrary to previous implementations [29], [30], [31], G-DNA already implements a technique to reduce the number of memory accesses performed. This technique contributed to derive the PE design for the FPGA implementation. A nucleotide is encoded as two or three bit value, depending on the type of sequence used. These representations are chained into a 32-bit integer which assembles a subsequence of 10 to 16 nucleotides which can be fetched in one memory call. On the downside of this separation, the last row and column of each submatrix have to be handed over as input for adjacent PEs, in order to provide correct alignment. This is done by two internal arrays of 8-bit integers providing the output vectors.

5.2 Systolic array and pipelining

The outer structure integrating several PEs to align full sequences was optimized to provide effective pipelining. This was achieved by transforming the original loop structure into a systolic array. As usually, this is a matrix of homogeneous data processing units (DPUs), in this case the PEs. Together they build a network where every DPU represents a pipeline stage. The input data consist of two input streams, i.e. sequences, which are processed in successive order by each stage of the two-dimensional pipeline. The result consist of two output streams at the last PEs. FPGAs are well suited for systolic arrays as the structure of individual programmable logic cells and flexible routing allows implementation of PEs and coupling without architectural limitations.

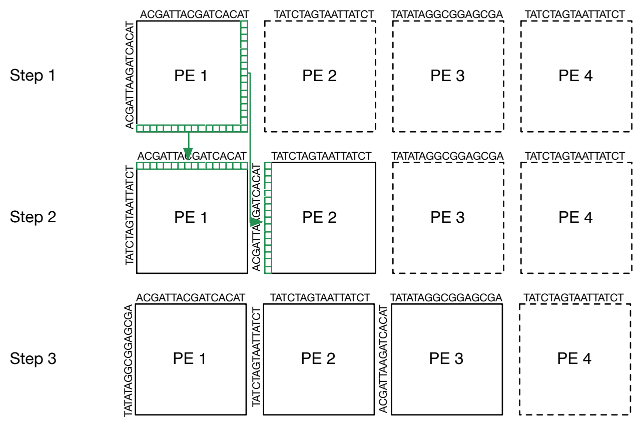

Figure 6: An example presenting the pipeline stages with systolic array and the first three alignment steps of two 64 nucleotide long sequences. In the first step only PE 1 is active. In the second step part of the sequence is handed to PE 2, additionally with the output vector from PE 1. Both PEs continue to work in parallel. This progresses until all submatrices have been processed.

The systolic structure allows reuse of PEs at their position in the pipeline. An example structure and the three initial processing steps for two 64 nucleotide long sequences are presented in Figure 6. In this particular case, the reuse of PEs reduces their number to four. In the first step only PE 1 is active and performs the alignment of the first 16 nucleotides. The output column is handed over to PE 2, while the output row remains at PE 1 for reuse. This is marked with arrows in the figure. These steps are repeated until the whole sequence is processed within six additional steps.

Afterwards, the resulting output arrays are processed in order to provide the alignment score and shift values, which are returned by the kernel. Both numbers are encoded in a single 32-bit unsigned integer and written to the output memory. Likewise in the previous step, the encoding is done to reduce memory footprint.

5.3 System integration

The hardware structure of the Zynq SoC is described in Section 5.4. Due to the implementation decisions, the length of sequences needs to be a multiple of 16, or the sequence may be filled up to this length. For the common case, a length of 112 nucleotides was chosen, which results in 49 PEs. With these architecture, area requirements of the cores allows occupation of 95.9 % of the FPGAs programmable logic, by integrating 11 individual cores. Detailed logic utilization is presented in Table 3. To provide functionality the cores require block RAM and communication structure which are included. The communication is realized by an integrated AXI bus, which connects every core to its sequence buffer and to the integrated ARM dual core processor. This setup allows individual access to each core by a management software, running on one of the ARM cores, which can also handle communication within the RECS®|Box server. The efficiency of this implementation has been demonstrated in Section 4.2.

Table 3: Programmable logic utilization for Zynq XC7Z045

	Used	Available	Utilization
Slice LUTs	209641	218600	95.90 %
LUT as Logic	208805	218600	95.51 %
LUT as Memory	836	70400	1.18 %
Slice Registers	171635	437200	39.25 %
Block RAM Tile	13	545	2.38 %

5.4 Hardware platform

The RECS®|Box hardware is designed and manufactured by the Bielefeld University and Christmann Informationstechnik + Medien company. Utilizing the RECS®|Box platform concept enables a seamless integration of general purpose processors, embedded processors, FPGAs, GPUs, and multi/many-core processors into a single scalable and modular server architecture. Since the platform focuses on high energy efficiency and on high resource efficiency, a tight integration with FPGAs has been developed and is presented in detail within this section. Importantly, the integrated monitoring and management functionalities of the RECS®|Box architecture are vital for the evaluation of energy efficiency of software tools. It is implemented by distributed microcontrollers placed on every board inside a RECS®|Box

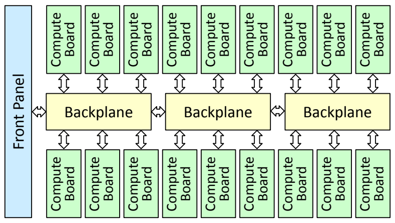

Figure 7: Structural overview of the RECS®|Box system

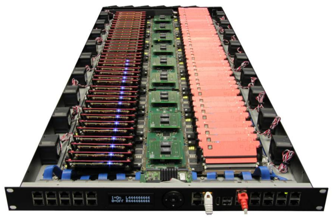

Figure 8: Example of a RECS®|Box system equipped with 72 ARM compute boards

system. All the controllers are connected via an I^{2}C-based management bus. An ARM microcomputer on the front panel runs the main management software and controls the distributed microcontrollers. It also provides access to the management system for the user via Gigabit Ethernet. In this paper, we used the integrated monitoring and management functionalities to measure the energy consumption of sequence alignment software. The RECS®|Box architecture is designed to provide a heterogeneous scale out approach (horizontal scaling). The architecture as well as the monitoring and management tools are developed to allow provisioning of the RECS®|Box system at rack or data center level, while still maintaining a comprehensive management solution across the RECS®|Box cluster. Compared to commercially available scale out approaches like HP moonshot [32], the RECS®|Box architecture offers a higher integration density, allowing more servers per rack while still supporting heterogeneous populations with compute boards of different architectures.

The RECS®|Box integrates up to 18 compute boards, each equipped with up to four microserver boards in one rack unit enclosure (cf. Figures 7 and 8). Microserver boards for Intel and AMD x86/x64 CPUs, eCPUs as well as for FPGAs have been realized and can be flexibly combined into a heterogeneous cluster server system. Communication between the compute boards is facilitated via a central backplane that offers switched Gigabit Ethernet as well as a dedicated, multi-standard interconnection infrastructure offering a bandwidth of up to 40 Gbit/s per compute board. Details of the hardware architecture can be found in [33].

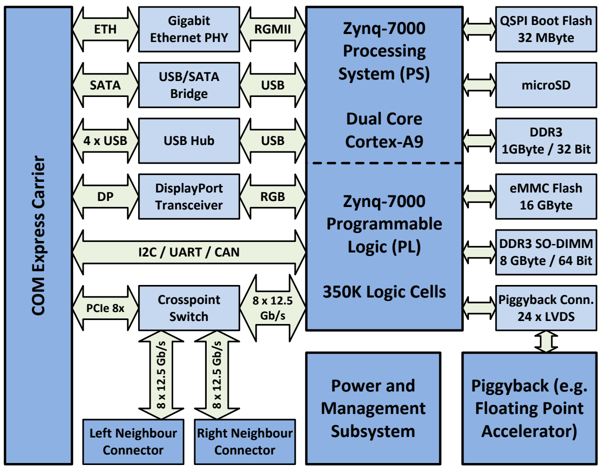

Figure 9: Architecture of the Zynq-based COM Express microserver

To achieve high resource and energy efficiency, the FPGA modules have been tightly integrated into the RECS®|Box server architecture. The FPGA-based microservers are integrated into the RECS®|Box platform using the COM Express standard. The Xilinx Zynq-7000 SoC (system on a chip) was chosen for integration. The Zynq-7000 family is based on the Xilinx all Programmable SoC architecture which integrates a dual-core ARM Cortex-A9 processing system and 28 nm Xilinx programmable logic in a single device. The Zynq-based COM Express microserver consists of a wide variety of building blocks enabling tight integration with the RECS®|Box system as well as its utilization for embedded applications. For the integration with the COM Express standard [34], the Zynq-7000 SoC was extended by additional I/O components to provide all required interfaces. Figure 9 gives an overview of the architecture of the module and its I/O structure. In addition to the interfaces defined in the COM Express module standard, high-speed serial communication between the different Zynq-based microservers is supported by utilizing the serial high-speed transceivers of the Xilinx Zynq devices. Depending on the device, the bandwidth of these transceivers reaches up to 12.5 Gbit/s per channel, where each lane consists of an RX and a TX channel allowing full duplex communication. These are connected via asynchronous crosspoint switches to enable highly flexible communication topologies between the different modules. The use of the serial high-speed transceivers enables a dedicated, high-speed (up to 100 GBit/s) low latency (300 ns) communication infrastructure between the Zynq-based microservers.

6 Conclusions

In this paper we compared top-class implementations performing pairwise sequence alignment, namely: SWIPE 2.0.5, SSEARCH 36, CUDASW++ 3.1, G-DNA and SWAPHI 1.0.5. The tools were selected in a way to allow for a fair comparison (algorithms based on dynamic programming, optimized for short sequences) and to span multiple hardware architectures. The results show that differences in energy efficiency are substantial. For example: CUDASW++ 3.1 running on a GPU can perform the same database scan as SWAPHI 1.0.5 running on Intel Xeon Phi, only using 4.75 times less energy. Significant differences were observed also within the same architecture, e.g. SWIPE 2.0.5 was shown to be up to 110 % more energy efficient as compared to SSEARCH 36, both running on the same multi-core CPU. Such information may be of great importance to those designing bioinformatic pipelines or complex software that involves sequence alignment. HPC centers may benefit from these results as well by giving higher priority or visibility to more energy efficient tools. The reduction in power consumption may be noticeable, given that bioinformatic analysis has become a major application for some of the high-performance installations.

The whole work has, however, a wider context. It was performed as part of the FiPS project that focuses on RECS®|Box hardware design and its evaluation. These energy efficient servers provide the user with advanced monitoring infrastructure, allowing for detailed power measurements and analysis. G-DNA was one of the applications selected to evaluate the hardware and the software stack developed within the project. Although the G-DNA tool was already highly optimized for a GPU, we were able to improve its energy efficiency by 20 % using the RECS®|Box hardware with FPGA module. Even though, the current implementation has some limitations regarding the maximum sequence length, we have proved that the energy efficient hardware constitutes an interesting alternative to traditional servers. Yet above all, by presenting this work we would like to encourage more researchers to design and use the software in a more energy-oriented way.

Acknowledgment

This research was supported by the EU Seventh Framework Programme FP7/2007–2013 under grant agreement no. FP7-ICT-2013-10 (609757). Webpage: www.fips-project.eu. This work was also funded as part of the Cluster of Excellence Cognitive Interaction Technology 'CITEC' (EXC 277), Bielefeld University, and supported by the PL-Grid Infrastructure in Poland.

References

[1] S. Manavski and G. Valle, "CUDA compatible gpu cards as efficient hardware accelerators for smith-waterman sequence alignment," *BMC Bioinformatics*, no. 9, 2008.

[2] L. Ligowski and W. Rudnicki, "An efficient implementation of smith waterman algorithm on GPU using CUDA, for massively parallel scanning of sequence databases," *IPDPS*, pp. 1–8, 2009.

[3] J. Blazewicz, W. Frohmberg, M. Kierzynka, E. Pesch, and P. Wojciechowski, "Protein alignment algorithms with an efficient backtracking routine on multiple GPUs," *BMC Bioinformatics*, vol. 12:181, 2011.

[4] Y. Liu, A. Wirawan, and B. Schmidt, "CUDASW++ 3.0: accelerating Smith-Waterman protein database search by coupling CPU and GPU SIMD instructions," *BMC Bioinformatics*, vol. 14:117, 2013.

[5] W. Frohmberg, M. Kierzynka, J. Blazewicz, P. Gawron, and P. Wojciechowski, "G-DNA – a highly efficient multi-GPU/MPI tool for aligning nucleotide reads," *Bull. Pol. Ac.: Tech.*, vol. 61, pp. 989–992, 2013.

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

[7] N. Neves, N. Sebastiao, A. Patricio, D. Matos, P. Tomas, P. Flores, and N. Roma, "BioBlaze: Multi-Core SIMD ASIP for DNA Sequence Alignment," *IEEE 24th International Conference on Application-Specific Systems, Architectures and Processors*, pp. 241–244, 2013.

[8] M. Farrar, "Optimizing Smith-Waterman for the Cell Broadband Engine," 2008.

[9] A. Szalkowski, C. Ledergerber, P. Krahenbuhl, and D. C., "SWPS3 – fast multi-threaded vectorized smith-waterman for IBM cell/b.e. and x86/SSE2," *BMC Research Notes*, no. 1, p. 107, 2008.

[10] Y. Liu and B. Schmidt, “SWAPHI: Smith-Waterman protein database search on Xeon Phi coprocessors,” *IEEE 25th International Conference on Application-specific Systems, Architectures and Processors*, pp. 184–185, 2014.

[11] M. Farrar, “Striped Smith-Waterman speeds database searches six times over other SIMD implementations,” *Bioinformatics*, vol. 23, no. 2, pp. 156–161, 2007.

[12] T. Rognes, “Faster Smith-Waterman database searches with inter-sequence SIMD parallelisation,” *BMC Bioinformatics*, vol. 12:221, 2011.

[13] Y. Lhuillier, J.-M. Philippe, A. Guerre, M. Kierzynka, and A. Oleksiak, “Parallel architecture benchmarking: from embedded computing to HPC, a FiPS project perspective,” *2014 International Conference on Embedded and Ubiquitous Computing, IEEE*, pp. 154–161, 2014.

[14] S. Needleman and C. Wunsch, “A general method applicable to the search for similarities in the amino acid sequence of two proteins,” *J Mol Biol 48 (3): 443-3*, vol. 48, pp. 443–53, 1970.

[15] T. Smith and M. Waterman, “Identification of common molecular subsequences,” *Journal of Molecular Biology 147: 195-97*, vol. 147, pp. 195–97, 1981.

[16] W. Frohmberg, M. Kierzynka, J. Blazewicz, and P. Wojciechowski, “G-PAS 2.0 - an improved version of protein alignment tool with an efficient backtracking routine on multiple GPUs,” *Bull. Pol. Ac.: Tech.*, vol. 60, pp. 491–494, 2012.

[17] E. De Sandes, G. Miranda, A. De Melo, X. Martorell, and E. Ayguade, “CUDAlign 3.0: Parallel Biological Sequence Comparison in Large GPU Clusters,” *Cluster, Cloud and Grid Computing, 2014 14th IEEE/ACM*, pp. 160–169, 2014.

[18] J. Blazewicz, W. Frohmberg, P. Gawron, M. Kasprzak, M. Kierzynka, A. Swiercz, and P. Wojciechowski, “DNA Sequence Assembly Involving an Acyclic Graph Model,” *Foundations of Computing and Decision Sciences*, vol. 38, pp. 25–34, 2013.

[19] B. Langmead and S. Salzberg, “Fast gapped-read alignment with Bowtie 2,” *Nature Methods*, vol. 9, pp. 357–359, 2012.

[20] J. Blazewicz, W. Frohmberg, M. Kierzynka, and P. Wojciechowski, “G-MSA – A GPU-based, fast and accurate algorithm for multiple sequence alignment,” *J. Parallel. Distr. Com.*, vol. 73, pp. 32–41, 2013.

[21] S. Dydel and P. Bala, “Large Scale Protein Sequence Alignment Using FPGA Reprogrammable Logic Devices,” *LNCS*, vol. 3203, pp. 23–32, 2004.

[22] T. Oliver, B. Schmidt, and M. D.L., “Reconfigurable architectures for bio-sequence database scanning on FPGAs,” *IEEE Trans. Circuit Syst. II*, no. 52, pp. 851–55, 2005.

[23] J. Allred, J. Coyne, W. Lynch, V. Natoli, J. Grecco, and J. Morrissette, “Smith-Waterman implementation on a FSB-FPGA module using the Intel Accelerator Abstraction Layer,” *IEEE International Symposium on Parallel and Distributed Processing, 2009*, 2009.

[24] X. Chang, F. Escobar, C. Valderrama, and V. Robert, "Exploring Sequence Alignment Algorithms on FPGA-based Heterogeneous Architectures," in *IWBBIO*, (Granada, Spain), pp. 330–341, 7-9 April 2014.

[25] L. Hasan and H. Zafar, "Performance versus Power Analysis for Bioinformatics Sequence Alignment," *Journal of applied research and technology*, vol. 10, no. 6, pp. 920–928, 2012.

[26] M. Cruz, P. Tomas, and N. Roma, "Energy-Efficient Architecture for DP Local Sequence Alignment: Exploiting ILP and DLP," *Lecture Notes in Computer Science*, vol. 9044, pp. 194–206, 2015.

[27] K. Benkrid, A. Akoglu, C. Ling, Y. Song, Y. Liu, and X. Tian, "High Performance Biological Pairwise Sequence Alignment: FPGA versus GPU versus Cell BE versus GPP," *International Journal of Reconfigurable Computing*, 2012.

[28] T. U. Consortium, "UniProt: a hub for protein information," *Nucleic Acids Res.*, vol. 43, pp. D204–D212, 2015.

[29] T. Oliver, B. Schmidt, and D. Maskell, "Reconfigurable architectures for bio-sequence database scanning on fpgas," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol. 52, pp. 851–855, Dec 2005.

[30] Y. Yamaguchi, H. Tsoi, and W. Luk, "Fpga-based smith-waterman algorithm: Analysis and novel design," in *Reconfigurable Computing: Architectures, Tools and Applications* (A. Koch, R. Krishnamurthy, J. McAllister, R. Woods, and T. El-Ghazawi, eds.), vol. 6578 of *Lecture Notes in Computer Science*, pp. 181–192, Springer Berlin Heidelberg, 2011.

[31] P. Zhang, G. Tan, and G. R. Gao, "Implementation of the smith-waterman algorithm on a reconfigurable supercomputing platform," in *Proceedings of the 1st International Workshop on High-performance Reconfigurable Computing Technology and Applications: Held in Conjunction with SC07*, HPRCTA '07, (New York, NY, USA), pp. 39–48, ACM, 2007.

[32] HP, "HP Moonshot System." *Available at* `http://www8.hp.com/us/en/products/servers/moonshot/`. Accessed: 13-August-2015.

[33] R. Griessl, M. Peykanu, J. Hagemeyer, M. Porrmann, S. Krupop, M. Berge, T. Kiesel, and W. Christmann, "A scalable server architecture for next-generation heterogeneous compute clusters," in *12th IEEE Int. Conf. on Embedded and Ubiquitous Computing (EUC)*, pp. 146–153, Aug 2014.

[34] PICMG, "PICMG COM.0 R2.1 - Com Express Module Base Specification." *Available at* `http://www.picmg.org`. Accessed: 13-August-2015.