FPGA-based hardware acceleration for local complexity analysis of massive genomic data

Agathoklis Papadopoulos, Ioannis Kirmitzoglou, Vasilis J. Promponas, Theocharis Theocharides

A R T I C L E   I N F O

Keywords:
Bioinformatics
Reconfigurable parallel architectures
Genomics
Hardware acceleration

A B S T R A C T

While genomics have significantly advanced modern biological achievements, it requires extensive computational power, traditionally employed on large-scale cluster machines as well as multi-core systems. However, emerging research results show that FPGA-based acceleration of algorithms for genomic applications greatly improves the performance and energy efficiency when compared to multi-core systems and clusters. In this work, we present a parallel, hardware acceleration architecture of the CAST (Complexity Analysis of Sequence Tracts) algorithm, employed by biologists for complexity analysis of protein sequences encoded in genomic data. CAST is used for detecting (and subsequently masking) low-complexity regions (LCRs) in protein sequences. We designed and implemented the CAST accelerator architecture and built an FPGA prototype, with the purpose of benchmarking its performance against serial and multithreaded implementations of the CAST algorithm in software. The proposed architecture achieves remarkable speedup compared to both serial and multithreaded software CAST implementations ranging from approx. 100x–5000x, depending on the system configuration and the dataset features, such as low-complexity content and sequence length distribution. Such performance may enable complex analyses of voluminous sequence datasets, and has the potential to interoperate with other hardware architectures for protein sequence analysis.

© 2012 Elsevier B.V. All rights reserved.

1. Introduction

Genomics—the study of the complete genomes of biological species (including human)—has revolutionized the way biological research is currently performed. This is further evident with the novel high-throughput sequencing technologies that enable whole genome DNA sequencing of simple unicellular organisms in a matter of days [1]. Moreover, anticipating the "$1000 genome", researchers expect that genomic science will soon become a central part of clinical practice [2]. As huge amounts of sequence data are currently being produced worldwide at an increasing pace, extensive downstream computational analysis is required. Typical computational pipelines for genomics feature a computationally intensive sequence comparison component; this is justified by the empirical observation that genes and proteins with similar sequences usually perform similar functions. Therefore, sequence similarity search serves for inferring functional and structural analogy for biological macromolecules [3,4].

Traditionally, such complex algorithms have been implemented on high-performance computing clusters and multiprocessor systems [5], and recently, with the emergence of manycore architectures, on clusters equipped with such CPUs [5]. However, there has been a tremendous amount of effort in designing efficient systems that can take advantage of the inherent parallelism opportunities of such algorithms; one of the most powerful supercomputers in the world (IBM Blue Gene) has been designed with the objective of performing large-scale protein folding simulations [6]. Furthermore, the specialized nature of algorithms targeting bioinformatics, limits the number of end-users for such platforms, thus the costs of possible custom hardware solutions tend to be extremely high. As such, alternative technologies that can better balance the cost and performance constraints can be more efficient for targeting the bioinformatics research communities.

Emerging technologies such as general purpose computing on graphics processing units (GPGPU) and domain-specific hardware accelerators running on field-programmable gate arrays (FPGAs) are prime candidates for improving performance of bioinformatics applications. GPGPU methodologies have been recently developed to facilitate the BLAST (Basic Local Alignment Search Tool) algorithm and the resulting GPU-BLAST exhibited a 4x speedup [7]. However, reconfigurable hardware implementations have been proven capable of outperforming high-end GPGPU...
systems, even when running on low-budget FPGA boards [8,9], especially when taking the power consumption into consideration. Given the cost-performance benefits of FPGAs, recent advances in computational biology suggest that reconfigurable domain-specific hardware might be indeed a powerful alternative to cluster-based supercomputers [10].

Such architectures employ the flexibility of general-purpose computing that biologists can utilize without large learning curves, as well as the hardware suitable for parallel implementations of the algorithms that general-purpose processors lack. Thus, FPGAs can balance out performance, manufacturing costs and hardware resource constraints; furthermore, they offer higher programmability than ASICs and are easier accessible to non-expert end-users. Among the most benefited algorithms when mapped on such architectures, include those for processing massive genomic data which typically involve some form of sequence comparison.

Sequence comparison is usually performed by aligning sequences, i.e. by algorithmically identifying the optimal correspondence of individual positions (residues) between the compared sequences, given a scoring scheme. When a newly identified protein sequence is submitted for comparison against a database of already known proteins, the similarity score alone is not sufficient to pinpoint important biological relationships for functional inference. Thus, robust statistical computations [11] have been used to reliably identify those similarities that are unlikely to have arisen simply by chance in a haystack of unrelated hits. Such measures involve complex data processing, with unpredictable data flow behavior; data dependencies involved in such algorithms are therefore a significant drawback when employing traditional superscalar and multi-threaded or multicore CPU architectures. Furthermore, memory bandwidth and memory management is another drawback.

From a biological perspective, some of the basic assumptions related to such applications do not hold for a significant fraction of known proteins. For example, for obtaining reliable estimates of the statistical significance of scores observed in a series of pairwise sequence comparisons, it is postulated that the compared sequences are random; they are generated by sampling from an amino acid residue distribution, i.e. the distribution of the protein sequence database. However, real biomolecular sequences deviate from this “ideal” distribution, especially in cases where skewed local composition is observed [12,13]. The latter observation was based on a working definition of the so-called Low Complexity Regions (LCRs) in amino acid sequences. More specifically, a measure of local compositional complexity was introduced, calculated by applying an entropy-like computation on composition vectors. This approach was followed in the SEG suite of programs as a two-pass procedure, where candidate LCRs are identified during the first pass, followed by an optimization step [13].

Based on the concept of LCRs, important improvements have been suggested by the bioinformatics community for improving the computational behavior of sequence comparison algorithms. In particular, identification of LCRs is usually followed by a procedure known as masking, with the objective of canceling out the effect of local compositional bias on scoring protein sequence comparisons. It is worth mentioning that proteins with LCRs seem to play important roles in several natural biological processes (including human disease) [14]. However, this work mainly focuses on the application of massive LCR detection in large protein datasets for masking, as a preprocessing step in sequence database search.

Alternative approaches for LCR detection and masking have been proposed, although not necessarily based on the same definition. A popular method is the CAST algorithm [15] (more details provided in Section 2). Masking protein sequences with CAST has been empirically shown to result into similarity searches of superior specificity (low false positive rate) without sacrificing sensitivity [15]. However, a major obstacle against the wider use of the CAST algorithm is its relatively low computational performance (compared to SEG) due to its iterative nature. With the exponential growth of sequence databases [16], even the fastest algorithms for sequence comparison fall short.

This paper therefore presents a parallel hardware architecture to accelerate sequence comparison bioinformatics algorithms, and in particular the CAST algorithm. The proposed architecture extends our previously introduced FPGA-based module initially presented in [17]. The extended architecture is capable of identifying and masking the LCRs of multiple protein sequences in parallel and, includes a complete high-speed communication channel for transferring and receiving proteomic data. Furthermore, this paper presents additional results stemming from added benchmark sequences, and provides a detailed comparison between the FPGA-based hardware accelerator and the corresponding software acceleration methodologies involving multicore and multithreaded architectures. The paper is organized as follows: Section 2 presents a brief background on the CAST algorithm and relevant related work. Section 3 presents the proposed architecture in detail, and an experimental prototype on an FPGA along with the evaluation results, is presented in Section 4. Section 5 concludes the paper.

2. Background – Related work

2.1. Background

Alignment-based pair-wise comparison of biological macromolecular sequences is a routine computational procedure practiced daily in most biological research laboratories throughout the...
world. Thus, a large set of tools have been developed over the years trying to provide improved solutions to the sequence comparison problem over the traditional dynamic programming algorithms. Rapid and sensitive algorithms, such as the BLAST heuristic algorithm [18], have become the most widely used tools for homology searches in sequence databases. The wide utilization of the BLAST suite of programs is clearly reflected by the fact that the two key methodological papers describing the methods [18,19] have been collectively cited more than 60,000 times [20].

When high sequence similarity is detected between a query sequence of unknown function and an annotated database entry, reliable function prediction for the query can be obtained. However, LCRs may result in erroneous function predictions, as the high score observed is rather due to the bias effect and not due to genuine homology. Thus, masking LCRs can significantly improve the reliability of homology detection and the quality of function prediction.

The CAST algorithm [15] is an iterative method for identifying and masking LCRs in protein sequences and has shown higher quality results over other proposed methods, such as SEG [13]. In principle, the CAST algorithm compares protein sequences against an artificial database consisting of 20 degenerate protein sequences of arbitrary length, each one being a homopolymer based on one of the 20 natural amino acid residue types. CAST identifies LCRs in a single linear pass for each amino acid type (Fig. 1) memory and computations required being linear to the input size. This feature is a consequence of the careful reformulation of the Smith–Waterman local sequence alignment algorithm [14], by taking into account that homopolymers do not carry positional information, and gaps are not permitted. The most remarkable feature of CAST is that not only it detects LCRs, but also identifies the type of amino acid residue causing the bias. Thus, selective masking can be performed in a more subtle and specific manner compared to other masking methods (e.g. [13]).

The LCR detection step requires the use of a substitution matrix, where matching non-identical residue types may give positive scores as well. Thus, an LCR biased in one residue type may lead to high scores for biases of a similar but not identical type (e.g. arginine/lysine). Therefore, CAST iteratively performs LCR detection and masking steps to prevent unnecessary masking due to cross-dependencies between amino acid residue types. An empirically defined threshold value $T$ for the similarity score serves as a LCR selection criterion. With the use of the BLOSUM62 substitution matrix, the optimal value $T=40$ is used. In practice, a variant of BLOSUM62 serves as the default scoring matrix: the scores of each residue type against the neutral type ‘X’, are computed as the mean value of the amino acid substitution scores for the respective residue type.

The algorithm shown in Fig. 1 receives as input a protein sequence, and, searches for the LCR candidates (highest scoring segments—HSS) of each natural amino acid type. It then selects the HSS with the maximum score, and if that score is less than the threshold $T$, it ends outputting discovered LCRs; otherwise, it replaces each occurrence of the max scoring residue type in the highest scoring segment region with an ‘X’ (i.e. a neutral amino acid) and iterates through the updated sequence. For each discovered LCR its residue type, the sequential position (start and end) and computed score are reported. Further details of the algorithm can be found in [15].

2.2. Related work

The protein database has been growing exponentially over the years, and the execution time of existing tools implemented traditionally in software grows exponentially even on high-end computer systems [21]. Recently; however, application-specific reconfigurable hardware solutions, utilizing FPGAs have emerged as promising alternatives. Several researchers proposed architectures for bioinformatics on FPGAs that exhibit performance improvements, such as those in [22–26].

The reconfigurable architecture of the BLAST algorithm proposed in [22] showed speedups of over 45x over the NCBI BLAST software. The architecture proposed for BLASTp in [23] showed remarkable speedups up to 1400x over the software implementations of the same algorithm. Along the same lines, the FPGA-based implementation of PSI-BLAST proposed in [24] showed speedup over 20x compared to the existing software solutions. In a relevant sequence analysis problem, an FPGA accelerator of the GOR algorithm [25] used for protein secondary structure prediction, showed speedup factors above 430x over the original GOR and more than 110x over the multi-threaded software version [26]. Recently, a Network-on-Chip-based hardware accelerator for biological sequence alignment reported speedup over three orders of magnitude over traditional CPU architectures [10]. Preprocessing tools implemented on hardware can further reduce execution times since major tools—such as BLAST—have already been implemented on reconfigurable fabric with astonishing results. A prefiltering approach for further improving performance of BLAST is already implemented on reconfigurable fabric [27].

To the best of our knowledge, the architecture we proposed in [17] was the first attempt to implement a hardware acceleration architecture that can be used for identifying and masking LCRs in protein sequences. In this work, we expanded our architecture by developing a complete system prototype that can massively

---

1 BLOSUM62 is one of the substitution matrices commonly used for calculating scores between evolutionarily divergent protein sequences. While it is possible to use any other substitution matrix, BLOSUM62 is the substitution matrix of choice for this work.
identify and mask LCRs. The necessary I/O communication and sequence allocation schemes has been developed as well in order to optimize the overall system performance in actual real world trials.

3. Proposed architecture

The proposed hardware architecture is based on custom-built modular components as building blocks. This design approach makes the proposed architecture highly scalable and implementable on other platforms, such as custom ASICs. As such, the system can be easily expanded and upgraded to address emerging computational and algorithmic needs, stemming from both enhanced datasets and algorithmic adjustments/improvements. The proposed architecture is designed to be seamlessly integrated in existing computational systems already used by the end-users. The protein sequences and the results are transferred-to and received-from the proposed architecture by utilizing a standard, high-speed communication channel such as PCI-Express or Gigabit Ethernet. An example setup where the FPGA board configured with the proposed architecture is installed on the PCI-Express bus of a host system is shown in Fig. 3.

The CAST algorithm can be executed over multiple protein sequences in parallel by using multiple instances of the hardware architecture initially presented in [17]. These Sequence Processing Units (SPUs) receive data streams prepared by a specialized allocation unit in order to achieve high load balancing and minimized total execution time. The system is receiving the protein sequence dataset from the host PC via a memory-mapped high-speed I/O channel (for example PCI-Express). This I/O scheme is selected because it is natively supported by modern operating systems and provides the end-users (who usually do not have computer engineering background) with a friendly programming environment.

3.1. Sequence processing unit—SPU

An SPU of the proposed hardware architecture—shown in Fig. 2—consists of a series of interconnected processing elements (PEs) that take advantage of the natively parallel characteristics of the CAST algorithm. The SPU receives a stream of protein sequence represented in FASTA format\(^2\), compares the stream with the twenty degenerate sequences, extracts current iteration’s LCR, masks it and iterates until all LCRs are discovered. As soon as all LCRs are discovered, the SPU outputs each of the sequence’s LCRs score and position in the sequence.

An SPU consists of three different types of PEs: a front-end unit, a number of CAST processing units and a back-end unit. The input sequence under consideration is streamed into the SPU through an I/O receiver and each unit of the SPU propagates the necessary signals to the next one in a pipelined manner until the propagated stream reaches the back-end unit. The back-end unit then sends it to the I/O transmitter or redirects the updated stream back for the next iteration.

3.1.1. Front-end unit

The front-end unit (FEU) is responsible for receiving one ASCII symbol of the input sequence per cycle from the allocation unit, and to generate the control signals needed by the CAST processing units. FEU is responsible for generating the signals required for communicating with the allocation unit as well. The received symbol is recorded by the ASCII decoder unit to a lesser bit representation in order to reduce the hardware resources needed. The Control Finite State Machine (FSM) of the FEU checks each received symbol and if it marks the beginning or the ending of a sequence, the unit generates the appropriate control signals which are propagated to the CAST units. The FEU also facilitates the logic necessary to control the number of iterations needed. Whenever the need for another iteration occurs, the unit signals the allocation unit to hold any new pending sequences and instead re-feeds the updated sequence received from the back-end unit back to the SPU architecture.

3.1.2. CAST unit

The CAST processing units perform the CAST algorithm computations. Each SPU has twenty identical CAST units, each one responsible for executing the CAST algorithm for one of the twenty degenerate homopolymers. They are interconnected in a row-wise pipelined manner. Each unit calculates the high scoring segment (HSS)—this region is indicating a potential LCR—of the input sequence for their respective amino acid residue type. A CAST unit consists of a local memory block holding the subset of the substitution matrix referring to its respective amino acid residue, a CAST core performing the score calculations needed to detect the boundaries of the HSS, and a shift/scan unit for propagating the data and control signals needed by the adjacent CAST unit.

The local memory block is initialized with the substitution values from a centralized memory block holding the complete matrix prior the arrival of the first input sequence. The substitution matrices used in Bioinformatics are rather small in size (i.e. the BLOSUM62 variant used in this work, requires less than 500 bytes) and can be stored in a centralized multi-ported memory block. However, the number of accesses to the centralized memory block to fetch the substitution values needed in the calculations for all of CAST units each cycle is a potential bottleneck. Hence, the decision to store the substitution values in small local memory blocks in every CAST unit was made, so that the substitution score data can be accessed asynchronously and as fast as possible. This design approach eliminates the memory access times of the centralized memory, as the data needed is fetched and processed in one cycle.

The CAST core—shown in Fig. 4—derives the score for the sequence streamed in and holds the statistics needed for identifying HSS and its score. The score accumulates every cycle the substitution scores of each amino acid in the sequence processed so far. If the accumulated score falls below zero, the score is

---

\(^2\) FASTA format is using English alphabet letters to represent amino acids in proteins, represented in ASCII text [31].
registered as zero and remains zero until the cycle where a positive substitution score arrives. This score value is compared against the current max score every cycle as well. The max score register value is increased if needed, and it is also compared against the threshold set by the CAST algorithm. The HSS boundaries can be retrieved just by storing HSS’s beginning and ending positions in the sequence. A counter measuring the position of the currently processed symbol in the input sequence is used for that purpose. The CAST core has modules which are used for marking the HSS boundaries using a sliding window approach. The end position of the HSS is the position where the current max score has retrieved. The shift and scan unit transmits the boundaries of the LCR, first the beginning and then the ending position.

The Shift & Scan unit is responsible for propagating the signals needed by the adjacent CAST units. The actions of all Shift & Scan units are decided by the control signals propagated through their respective CAST Unit. Every Shift & Scan unit operates in two modes: shift and scan. When in shift mode, the unit propagates the stream without changes while the SPU is computing the HSS. When in scan mode, the unit propagates the highest value between its CAST Unit’s max score and the propagated max score received from the left adjacent CAST unit. This method allows the back-end unit to receive the sequence’s maximum HSS score as soon as possible. Whenever the feedback unit informs the corresponding CAST unit that its respective amino acid HSS is actually the current iteration’s LCR, the shift & scan unit transmits the boundaries of the LCR, first the beginning and then the ending position.

### 3.1.3. Back-end unit

The back-end unit (BEU) decides whether the analysis of the current sequence is complete and propagates the results to the external I/O controller or updates the sequence and redirects it back to the FEU for calculating the next LCR. This decision is taken by the BEU’s control unit and is based on the current iteration having an amino acid residue HSS with higher value than the threshold parameter of the CAST algorithm. The symbols streamed by the last CAST unit while in calculation mode, are stored in a local Line Memory of a size equal to the size of the maximum sequence length used in the software implementations of the algorithm. This memory’s size can be easily increased to accommodate larger protein sequences. Line Memory is implemented as a dual-port memory block. The stream representing the sequence is deemed necessary to be stored locally as the nature of the CAST algorithm requires a number of iterations dynamically decided at runtime. Upon receiving confirmation that all symbols of the sequence are processed, the control unit waits for the HSS with the maximum score to be propagated through the architecture and marks it as current iteration’s LCR. It then transmits the necessary signals to the feedback unit to start reading the sequence from the Line Memory and stream it either back to the FEU for another round of calculations, or to the SPU output connected to an external I/O transmitter. The sequence is dynamically updated by the feedback unit as the symbols corresponding to the highest scoring amino-acid are replaced with ‘X’ while passing through the feedback unit.

### 3.2. Multi-SPU system

The SPU described above is efficiently executing the CAST algorithm over each sequence stream received. The performance of the hardware can be further boosted by integrating multiple SPUs in a single system. The inherent modularity makes the system highly scalable and a system featuring four SPUs is presented in Fig. 5. The number of SPUs present in each system is limited only by the I/O scheme and the available hardware resources—in the case of using FPGAs. As such, systems having eight or more SPUs are feasible as well. The SPUs are independent of each other and each one receives a different protein sequence stream. Each SPU is initially loaded with the substitution matrix values, executes the CAST algorithm over the protein sequence stream presented to its input port as described in the previous section, and exports the results in a dedicated FIFO which is used to present the results to the host PC via the high-speed I/O channel. A system of the proposed architecture featuring multiple SPUs needs to have a specialized allocation unit to properly partition the received protein dataset, and to prepare each SPU’s input stream.

### 3.3. Data allocation unit

A multi-SPU system integrated with a high-speed I/O communication channel capable of providing multiple symbols per cycle
needs an efficient way to partition the received protein sequence
data among the SPUs. As such, the host operating system (OS) can
be programmed to transmit the protein data in a scheme that is
efficient and requires minimum modifications by the proposed
architecture. Assuming that the protein sequences are received in
a predetermined format through the high-speed I/O communica-
tion channel, the system's allocation unit is responsible to divide
the sequences to the SPUs. The allocation scheme used, affects
the multi-SPU system's execution time for each protein database,
as unbalanced load sharing between the SPUs can hinder
performance.

The allocation unit can be designed as the one shown in Fig. 6
where a simple allocation unit, designed using FIFOs, for a system
having four SPUs is presented. This allocation unit was designed
with emphasis on fitting on the targeted FPGA board. The host OS
arbitrarily splits the protein sequence database in four parts and
provides an ASCII symbol taken simultaneously from each part per
cycle on the PCI-Express communication channel. As such, four
different sequences are fetched in parallel and are stored in input
FIFOs driving each SPU. Each FIFO has all necessary control signals
like read/write enable, full and empty connected either on the I/O
receiver module or the SPU; however those signals are omitted from
Fig. 6 for clarity. The abovementioned allocation scheme is also used
in evaluating the proposed hardware architecture.

Higher-complexity resource allocation schemes can take bet-
ter advantage of the inherent parallelism; however, these
schemes can fit only on larger FPGAs. The design of such
higher-complexity units is a challenge, as the execution time of
a given dataset is greatly affected be the number of iterations
needed for each sequence and that is not known beforehand. As
such, a static allocation scheme is not efficient, and intuitively an
arbitrary partition of the dataset among the SPUs can cause great
variations in execution times. The effects of arbitrarily partition-
ing the data among the SPUs using the resource allocation unit of
Fig. 6, is extensively discussed in Section 4.4.1.

3.4. Overall system data flow

The local memory blocks belonging to the CAST units of each
SPU are first initialized with their respective substitution matrix
subsets. As soon as the initialization is completed, the system is
ready to receive data from the high-speed I/O communication
channel. The host system begins transmitting the protein
sequential to FPGA, and the allocation unit stores them locally.
As soon as an SPU requests a sequence for processing, the
allocation unit allocates an unprocessed sequence to the idle
SPU following the selected allocation scheme. The newly allocated
sequence is then propagated as input stream to the SPU.

Each stream received by an SPU is processed one ASCII symbol
per cycle. The FEU of the SPU recognizes the beginning of the
sequence, sets the appropriate control signals and starts propa-
gating the symbols to the adjacent CAST units. When a new
sequence is streamed in these CAST units, each unit first resets
and waits for the symbols to arrive. Next, each unit searches
through its local memory block for the symbol's substitution
value, and it then updates the necessary internal registers for the
scores and HSS boundaries for each symbol propagated through.

The BEU of the SPU stores the streamed sequence to its Line
Memory block and waits for the end of sequence signal. When
this happens, the BEU generates the appropriate signals for the
CAST units and waits to receive the maximum scoring HSS. The
maximum score is propagated through the architecture, and,
when it reaches the BEU, is marked as current iteration's LCR.
The CAST unit having that LCR then, propagates the region
boundaries to BEU. If the current iteration's LCR score is higher
than the CAST algorithm threshold, the BEU updates the sequence
stored in its Line Memory and signals the FEU to hold any new
incoming sequences. The updated sequence is then fed back to

Please cite this article as: A. Papadopoulos, et al., FPGA-based hardware acceleration for local complexity analysis of massive
 genomic data, INTEGRATION, the VLSI journal (2012), http://dx.doi.org/10.1016/j.vlsi.2012.10.003
the FEU for the next iteration. However, if the threshold has not been reached, the BEU sends the LCR boundaries and scores to the SPU’s output FIFO. In the latter case, the SPU is free to receive the next protein sequence from the allocation unit.

4. Experimental platform and results

4.1. Experimental platform

A prototype of the proposed hardware architecture was implemented for a Virtex-5 LX110T FPGA running at a 100 MHz clock, and evaluated using real protein databases [28]. The initial evaluation results of a single SPU design were also presented in [17]. In this work, we evaluate a complete multi-SPU system, which includes the I/O module and the host machine. We implemented a PCI-Express communication channel using the Xilinx LogicCORE EndPoint Block Plus for PCI-Express IP core [29]. The allocation unit was built using the Xilinx CoreGenerator FIFO cores [30], following the scheme presented in Fig. 6. The complete system was downloaded on a Xilinx University Program XUP-V5-LX110T Evaluation Platform (cost ~$750) and the board was mounted on a host PC (Intel i3/4GB RAM), as shown in Fig. 3.

Every SPU has the local memory blocks of its corresponding CAST units mapped as distributed RAM on the FPGA fabric for more flexibility and speed. The line memory of each BEU is implemented as a standard dual-port memory and is mapped on the BRAM resources of the FPGA. The FPGA system was integrated to the host PC through the PCI-Express channel interface (see Fig. 3). Obviously, the use of the external communication channel adds significant overheads; however, we strongly believe that giving results for a complete multi-SPU system strengthens the actual impact of the proposed architecture and the presented results. It is of course implied that implementation of custom-built (but not necessarily standard) communication modules, optimized for the proposed CAST hardware architecture, will minimize the impact of communication overheads on the execution time, but the design of such systems is left as future work.

The original CAST implementation presented in [8] provides an initial comparison reference; however, comparing a highly parallel hardware architecture with a sequential single-threaded software implementation can be considered unfair. Thus, a multi-threaded version of CAST (mCAST) was developed in order to achieve better utilization of the resources present in multi-core CPUs [17]. Further improvements to the multi-threaded version that extends the initial version presented in [17], as part of this work, are also discussed next.

4.2. mCAST 2.0

The initial mCAST presented in [17] was simply a straightforward parallelization of the single-threaded algorithm of [8]. In [17], the identification of the HSS for each amino acid type was a two-step process. The first forward, step is the actual comparison of the sequence against the homopolymer and it results in a vector of scores for each position in the sequence. During the second, backward step, regions spanning from a score of zero to a local peak score are detected and homogenized; this results in a vector containing regions of uniform scores out of which the one with the higher score is picked as the HSS.

In the mCAST 2.0 (the version used in this work for comparison purposes), we optimized the execution path and the backward step is only executed if the maximum score detected in the forward step is equal or greater than the score threshold. This almost halves the execution time for the detection of each HSS and since CAST is an iterative algorithm the overall speed-up is, in many cases, much better.

We tested the efficiency of the improved multi-threaded mCAST 2.0 on a Intel Core i7 720 @ 2.66 GHZ/8 GB RAM system running Windows Server 2008. The mCAST2.0 outperformed the initial multi-threaded software by halving the execution time on average, even reaching a 9x speedup in some test cases. As such, we use the superior mCAST 2.0 software in order to fairly evaluate the proposed hardware architecture.

4.3. Proteomic Benchmarks

To compare the proposed hardware architecture against the mCAST 2.0 software implementation, we used a number of datasets stemming from four actual protein sequence databases 3. The first database belongs to Haemophilus influenza bacteria (haem database) and the second consists of protein sequences from the malaria parasite genome Plasmodium falciparum (p. falciparum database). The third and fourth databases are a reference viral database and a unified protein database from NCBI used for benchmarking bioinformatics applications (Viral.1.protein database, uniprot.sprot database). The haem database was also partitioned in smaller sequence databases (labeled Case I to Case V) in order to have different sized datasets and datasets with different sequence lengths. Table 1 lists all the test databases used during the evaluation process.

4.4. Hardware evaluation

We evaluated the proposed hardware architecture designed for executing the CAST algorithm by using three different system builds: a single-SPU architecture, a Quad-SPU and finally an Octal-SPU system. All evaluation configurations have a complete 64 bit/per cycle PCI-Express I/O communication channel. We evaluated the octal-SPU system—which consumed almost all available resources on the FPGA evaluation platform—in order to observe the scalability of the proposed architecture.

We compare the results of the proposed hardware architecture configurations against the software mCAST 2.0 implementation running on the Intel Core i7 system which characteristics are given in Section 4.2. Moreover, we run the SEG algorithm on the Intel Core i7 system for all test datasets. Comparing the results of the proposed architecture against SEG algorithm is essential, as SEG algorithm is a common option for LCR masking used by the BLAST suite. The execution times for both mCAST 2.0 and SEG have been calculated while suppressing I/O system calls for fair comparison with the proposed hardware architecture; the software under these conditions focuses on reading inputs and computing the algorithm, similarly with the hardware approach.

Table 2 shows the comparisons between the execution time achieved by mCAST 2.0 and SEG, against the execution time of the three evaluation configurations of the proposed architecture. The performance results of all hardware configurations outperform both the optimized mCAST 2.0 and SEG despite running at a lower clock of 100 MHz. In fact, results are orders of magnitude better for all test sequences. The merits of designing a hardware architecture for accelerating the masking of LCRs in protein sequences are made clear when comparing the execution time results for the Haem, p. falciparum, Viral.1.protein and Uniprot.sprot databases. These cases are actual protein databases provided by NCBI which are used extensively for benchmarking bioinformatics applications. The single-SPU configuration yields speedups higher than 30x over SEG and over two orders of magnitude over

---

3 The datasets used in the evaluation of the proposed system can be downloaded from [32].
and over 5000x speedups, respectively. These astonishing results, on a dataset from the Uniprot.sprot database, show over 2500x speedup over mCAST 2.0 and 104x speedup over SEG. Results on the hardware approach, as the proposed quad-SPU hardware has a significant speedup of compositionally biased regions, demonstrate the capabilities of the proposed architecture configurations. The simple allocation scheme used in the evaluation of the system is proven adequate for each SPU and the average utilization percentage relative to the calculation of the SPU utilization is 3.20%.

Table 2 indicates the limitations of LCR masking on general-purpose high-end processors, as they cannot efficiently execute algorithms used in bioinformatics due to memory bandwidth issues and limitations in available levels of parallelism, despite the coding improvements made to the software implementations. On the other hand, a custom-built hardware system running even on a cheap FPGA board can significantly accelerate performance. Unbalanced input streams will cause some of the SPUs to finish their load early while other SPUs will be struggling under heavier loads; this hinders the overall execution time.

Table 3 provides the execution times of every SPU when the proposed architecture configurations follow the allocation scheme of Fig. 6. We also calculated the utilization percentage for each SPU and the average utilization percentage relative to the longest running SPU for each configuration. The simple allocation unit shown in Fig. 6 is proven adequate to justify the proposed system, despite the fact that the arbitrary allocation of the sequences caused unbalanced utilization of the CAST instances and negatively affected performance results, as shown in Table 3. For instance, the load of the quad-SPU configuration is greatly unbalanced while it was executing the haem dataset. The arbitrary partition of the dataset by the OS of the host PC has placed heavy load on SPU-0 and SPU-1 while the rest SPUs are underutilized (utilization 13% and 14% for SPU-1 and SPU-2 respectively). The overall execution time for the dataset is calculated as 1.68 ms (time needed for all SPUs to finish their load) while a properly balanced allocation should have yielded execution time below 0.9 ms.

The simple allocation scheme used in the evaluation of the proposed hardware architecture offers more balanced SPU utilization as we increase the number of SPUs in the system. Table 3 shows that the octal-SPU configuration is well balanced in most cases. Splitting each dataset in a larger number of partitions statistically increases the probability to achieve even load among the SPUs. As such, a system featuring eight or more SPUs has no need for a complex allocation unit.

4.4.1. Allocation unit—observations and discussion

The allocation scheme for partitioning the received sequences to the SPUs is a design aspect that needs to be evaluated as well. A poor allocation scheme will cause significant variation between the SPUs execution times. Unbalanced input streams will cause some of the SPUs to finish their load early while other SPUs will be struggling under heavier loads; this hinders the overall execution time.

Table 1

<table>
<thead>
<tr>
<th>Dataset name</th>
<th>Number of sequences</th>
<th>Average length</th>
<th>% LCRs</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case I</td>
<td>1</td>
<td>1695</td>
<td>100(12.51)</td>
<td>Longest sequence of haem database</td>
</tr>
<tr>
<td>Case II</td>
<td>236</td>
<td>311.6</td>
<td>7.62 (0.59)</td>
<td>Random sequences from haem database</td>
</tr>
<tr>
<td>Case III</td>
<td>478</td>
<td>299.2</td>
<td>8.57 (0.69)</td>
<td>Random sequences from haem database</td>
</tr>
<tr>
<td>Case IV</td>
<td>258</td>
<td>292.3</td>
<td>7.75 (0.58)</td>
<td>Random sequences from haem database</td>
</tr>
<tr>
<td>Case V</td>
<td>113</td>
<td>764</td>
<td>23.89 (1.28)</td>
<td>Sequences longer than 600 from haem database</td>
</tr>
<tr>
<td>Haem</td>
<td>1743</td>
<td>305.1</td>
<td>7.97 (0.64)</td>
<td>All sequences from haem database</td>
</tr>
<tr>
<td>p. falciparum</td>
<td>5491</td>
<td>755.9</td>
<td>6.74 (1.68)</td>
<td>All sequences from p. falciparum database</td>
</tr>
<tr>
<td>Viral.1.protein</td>
<td>101,537</td>
<td>274.4</td>
<td>3.96 (7.84)</td>
<td>All sequences from viral.1.protein reference database</td>
</tr>
<tr>
<td>Uniprot.sprot</td>
<td>537,593</td>
<td>354.9</td>
<td>3.08 (7.36)</td>
<td>All sequences from uniprot.sprot reference database</td>
</tr>
</tbody>
</table>

mCAST 2.0. The quad-SPU architecture outperforms SEG with two orders of magnitude speedup and mCAST 2.0 is outperformed by several thousand times for most test cases. The octal-SPU configuration has shown astonishing results as well, proving that the proposed architecture is indeed highly scalable as this configuration does halve the execution times when compared to the quad-SPU configuration.

Results on the p. falciparum database, which has a large number of compositionally biased regions, demonstrate the capabilities of the hardware approach, as the proposed quad-SPU hardware has a 350x speedup over mCAST 2.0 and 104x speedup over SEG. Results of the quad- and octal-SPU configurations on the over-half-a-million-sequences long Uniprot.sprot database, show over 2500x and over 5000x speedups, respectively. These astonishing results prove that the proposed system for identifying and masking LCRs is capable of handling efficiently the massive protein datasets nowadays analyzed in most biology labs worldwide.

Table 2

<table>
<thead>
<tr>
<th>Dataset name</th>
<th>Number of sequences</th>
<th>Average length</th>
<th>% LCRs</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case I</td>
<td>1</td>
<td>1695</td>
<td>100(12.51)</td>
<td>Longest sequence of haem database</td>
</tr>
<tr>
<td>Case II</td>
<td>236</td>
<td>311.6</td>
<td>7.62 (0.59)</td>
<td>Random sequences from haem database</td>
</tr>
<tr>
<td>Case III</td>
<td>478</td>
<td>299.2</td>
<td>8.57 (0.69)</td>
<td>Random sequences from haem database</td>
</tr>
<tr>
<td>Case IV</td>
<td>258</td>
<td>292.3</td>
<td>7.75 (0.58)</td>
<td>Random sequences from haem database</td>
</tr>
<tr>
<td>Case V</td>
<td>113</td>
<td>764</td>
<td>23.89 (1.28)</td>
<td>Sequences longer than 600 from haem database</td>
</tr>
<tr>
<td>Haem</td>
<td>1743</td>
<td>305.1</td>
<td>7.97 (0.64)</td>
<td>All sequences from haem database</td>
</tr>
<tr>
<td>p. falciparum</td>
<td>5491</td>
<td>755.9</td>
<td>6.74 (1.68)</td>
<td>All sequences from p. falciparum database</td>
</tr>
<tr>
<td>Viral.1.protein</td>
<td>101,537</td>
<td>274.4</td>
<td>3.96 (7.84)</td>
<td>All sequences from viral.1.protein reference database</td>
</tr>
<tr>
<td>Uniprot.sprot</td>
<td>537,593</td>
<td>354.9</td>
<td>3.08 (7.36)</td>
<td>All sequences from uniprot.sprot reference database</td>
</tr>
</tbody>
</table>

- a Speedup of Quad-SPU H/W over single-SPU H/W architecture.
- b Speedup of Octal-SPU H/W over single-SPU H/W architecture.
- c Speedup of single-SPU H/W over mCAST.
- d Speedup of Quad-SPU H/W architecture over mCAST.
- e Speedup of Octal-SPU H/W architecture over mCAST.
- f Speedup of single-SPU H/W architecture over SEG.
- g Speedup of Quad-SPU H/W architecture over SEG.
- h Speedup of Octal-SPU H/W architecture over SEG.

Table 3

<table>
<thead>
<tr>
<th>Dataset name</th>
<th>Number of sequences</th>
<th>Average length</th>
<th>% LCRs</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case I</td>
<td>1</td>
<td>1695</td>
<td>100(12.51)</td>
<td>Longest sequence of haem database</td>
</tr>
<tr>
<td>Case II</td>
<td>236</td>
<td>311.6</td>
<td>7.62 (0.59)</td>
<td>Random sequences from haem database</td>
</tr>
<tr>
<td>Case III</td>
<td>478</td>
<td>299.2</td>
<td>8.57 (0.69)</td>
<td>Random sequences from haem database</td>
</tr>
<tr>
<td>Case IV</td>
<td>258</td>
<td>292.3</td>
<td>7.75 (0.58)</td>
<td>Random sequences from haem database</td>
</tr>
<tr>
<td>Case V</td>
<td>113</td>
<td>764</td>
<td>23.89 (1.28)</td>
<td>Sequences longer than 600 from haem database</td>
</tr>
<tr>
<td>Haem</td>
<td>1743</td>
<td>305.1</td>
<td>7.97 (0.64)</td>
<td>All sequences from haem database</td>
</tr>
<tr>
<td>p. falciparum</td>
<td>5491</td>
<td>755.9</td>
<td>6.74 (1.68)</td>
<td>All sequences from p. falciparum database</td>
</tr>
<tr>
<td>Viral.1.protein</td>
<td>101,537</td>
<td>274.4</td>
<td>3.96 (7.84)</td>
<td>All sequences from viral.1.protein reference database</td>
</tr>
<tr>
<td>Uniprot.sprot</td>
<td>537,593</td>
<td>354.9</td>
<td>3.08 (7.36)</td>
<td>All sequences from uniprot.sprot reference database</td>
</tr>
</tbody>
</table>

- a Speedup of Quad-SPU H/W over single-SPU H/W architecture.
- b Speedup of Octal-SPU H/W over single-SPU H/W architecture.
- c Speedup of single-SPU H/W over mCAST.
- d Speedup of Quad-SPU H/W architecture over mCAST.
- e Speedup of Octal-SPU H/W architecture over mCAST.
- f Speedup of single-SPU H/W architecture over SEG.
- g Speedup of Quad-SPU H/W architecture over SEG.
- h Speedup of Octal-SPU H/W architecture over SEG.

Please cite this article as: A. Papadopoulos, et al., FPGA-based hardware acceleration for local complexity analysis of massive genomic data, INTEGRATION, the VLSI journal (2012), http://dx.doi.org/10.1016/j.vlsi.2012.10.003
The average power consumption was estimated using the Xilinx XPower tool over the proposed architecture’s post-route synthesis results for the Xilinx Virtex-5 LX110T FPGA. Table 4 provides the FPGA resources needed for mapping the proposed architecture on a relatively cheap, off-the-shelf FPGA (see Section 4.1 for details).

We provide the FPGA resources needed to map each SPU, the simple allocation unit of Fig. 6 and the PCI-Express communication module integrated in the proposed architecture. The PCI-Express communication module integrates the required hardware over 500x and occupies around 50% of the FPGA used in the current implementation. Such overheads are small enough to allow other FPGA-based bioinformatics software tools on the FPGA fabric as well.

The proposed architecture accelerates over 100x the execution time of the improved multi-threaded CAST algorithm using just 10% of the LX110T FPGA for implementing a single-SPU system. Such overheads are small enough to allow other FPGA-based bioinformatics algorithms such as BLAST to be combined with CAST on the same FPGA, further enhancing the performance of such systems. The quad-SPU system yields average speedups of over 500x and occupies around 50% of the FPGA used in the evaluation process, which leaves enough resources for mapping other bioinformatics software tools on the FPGA fabric as well.

The PCI-Express communication module integrated in the proposed architecture enables the utilization of multiple FPGA boards on the same host PC, each running either an instance of a multi-SPU architecture or other FPGA-based bioinformatics applications like BLAST.

4.5. Power consumption

The average power consumption was estimated using the Xilinx XPower tool over the proposed architecture’s post-route simulation model. The results show that an 4-SPUs system consumes less than 1.8 W of dynamic power on average. Taking the minimized execution time into account as well, the proposed system’s energy consumption gains rate well in the range of the other FPGA-based architectures studied in [9] where FPGAs are proven to use orders of magnitude less energy than their GPU- and CPU-based counterparts.

4.6. Hardware synthesis results for Xilinx Virtex-5 FPGA

We lastly give the hardware synthesis results in Table 4, in order to give a complete picture of the required hardware resources needed for mapping the proposed architecture on a relatively cheap, off-the-shelf FPGA (see Section 4.1 for details). We provide the FPGA resources needed to map each SPU, the simple allocation unit of Fig. 6 and the PCI-Express communication modules while following the memory mapping considerations discussed in Section 4.1. The synthesis results for the complete quad-SPU and octal-SPU systems are also provided.

The proposed architecture accelerates over 100x the execution time of the improved multi-threaded CAST algorithm using just 10% of the LX110T FPGA for implementing a single-SPU system. Such overheads are small enough to allow other FPGA-based bioinformatics algorithms such as BLAST to be combined with CAST on the same FPGA, further enhancing the performance of such systems. The quad-SPU system yields average speedups of over 500x and occupies around 50% of the FPGA used in the evaluation process, which leaves enough resources for mapping other bioinformatics software tools on the FPGA fabric as well.

The PCI-Express communication module integrated in the proposed architecture enables the utilization of multiple FPGA boards on the same host PC, each running either an instance of a multi-SPU architecture or other FPGA-based bioinformatics applications like BLAST.

5. Conclusions

This paper presented an FPGA-based hardware acceleration architecture for accelerated masking of LCRs in protein sequences. The proposed architecture implements the CAST algorithm, enabling very fast and high-quality selective masking in very large genomic datasets. We observe that the speedup is over 500x and occupies around 50% of the FPGA used in the evaluation process, which leaves enough resources for mapping other bioinformatics software tools on the FPGA fabric as well.

The PCI-Express communication module integrated in the proposed architecture enables the utilization of multiple FPGA boards on the same host PC, each running either an instance of a multi-SPU architecture or other FPGA-based bioinformatics applications like BLAST.
the proposed architecture presents speedups of 10 to 200 times faster.

We plan on combining the proposed CAST hardware design with the BLAST hardware architectures developed in literature [23]. We expect this approach will further increase performance gains for protein alignment analysis while yielding high quality results. Issues, such as communication schemes, data transformations, dynamic load balancing for fully utilizing hardware resources and I/O communication channels for achieving higher performance, as well as power consumption will be addressed.

Acknowledgments

This work was co-funded by the European Regional Development Fund of the Republic of Cyprus through the Research Promotion Foundation programmes (PENEK/ENIXS/0308/77) and (TIENEK/0311/32).

References


A. Papadopoulos et al. / INTEGRATION, the VLSI journal 1 (2012), 10–19

[9] Joannis Kirmitsoglou holds a B.Sc. degree in Physics and a Ph.D. in Biological Sciences both from the Aristotle University of Thessaloniki, Greece. Having dealt with marine biodiversity informatics his research interests expanded to also include bioinformatics. Currently, he builds and applies computational tools, web services and machine learning methods for proteins sequence analysis and genome and protein evolution. He also continues to develop tools and services related to biodiversity informatics.