2021 31st International Conference on Field-Programmable Logic and Applications (FPL) | 978-1-6654-3759-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/FPL53798.2021.00024

Communication-avoiding micro-architecture to compute Xcorr scores for peptide identification

Sumesh Kumar, Fahad Saeed
Knight Foundation School of Computing and Information Sciences
Florida International University (FIU)
Miami, FL USA 33199
{sumesh.kumar, fsaeed}@fiu.edu

***Abstract*—Database algorithms play a crucial part in systems biology studies by identifying proteins from mass spectrometry data. Many of these database search algorithms incur huge computational costs by computing similarity scores for each pair of sparse experimental spectrum and candidate theoretical spectrum vectors. Modern MS instrumentation techniques which are capable of generating high-resolution spectrometry data require comparison against an enormous search space, further emphasizing the need of efficient accelerators. Recent research has shown that the overall cost of scoring, and deducing peptides is dominated by the communication costs between different hierarchies of memory and processing units. However, these communication costs are seldom considered in accelerator-based architectures leading to inefficient DRAM accesses, and poor data-utilization due to irregular memory access patterns. In this paper, we propose a novel communication-avoiding micro-architecture to compute cross-correlation based similarity score by utilizing efficient local cache, and peptide pre-fetching to minimize DRAM accesses, and a custom-designed peptide broadcast bus to allow input reuse. An efficient bus arbitration scheme was designed, and implemented to minimize synchronization cost and exploit parallelism of processing elements. Our simulation results show that the proposed micro-architecture performs on average 24x better than a CPU implementation running on a 3.6 GHz Intel i7-4970 processor with 16GB memory.**

***Index Terms*—cross-correlation, protein identification, SEQUEST, accelerator, micro-architecture**

I. Introduction

Mass-spectrometry based analysis has been the preferred method for identification of proteins from complex biological samples [1]. The last two decades have seen tremendous developments in data acquisition and analysis techniques which have enabled many powerful proteomic applications. Database search algorithms such as SEQUEST [7], X!Tandem [6], and MSFragger [9] can now search high resolution mass-spectrometry data against an ever increasing protein database to produce high quality matches. This has drastically increased the compute load for existing implementations of the database search algorithms. In this regard, several studies have used parallelization strategies using high-performing compute clusters [10], [16], [18], GPUs [3], [11], [13], and FPGAs [4], [5], [15], [19] to speed up the computation process. However, a recent study suggests that the major bottleneck in mass-spectrometry based analysis is the cost of communication i.e cost of moving input and output data between different hierarchies of a system [17]. Thus, even though CPUs are operating at a much higher frequency, their performance gain for proteomics studies relies on efficiently utilizing system cache or some other input reuse technique [2] to minimize the number of DRAM accesses. Consequently, the implementation of Crux [12], state-of-the-art software for computing cross-correlation (Xcorr) scores, utilizes processor registers to store peptide fragment ions to allow peptide reuse. While this allows one-side data reuse, the cost of accessing experimental spectra from main memory is not minimized as generally CPU registers are not large enough to hold the entire experimental spectrum. On the other hand custom architectures using FPGAs can achieve better performance for memory bound applications by utilizing the abundant on-chip RAM resources and custom-designed communication minimizing pipelines to allow experimental spectrum reuse [14].

In this paper, we propose a communication-avoiding micro-architecture to accelerate the Xcorr score computation which achieves two-side data reuse by utilizing the on-chip RAM to cache an entire experimental spectrum and a peptide broadcast bus to decrease the number of DRAM accesses. Our experiments show that these optimizations result in 600x reduction in the average number of DRAM accesses compared with a no-caching approach and 24x times speed-up over Crux. The main contributions of this paper are as follows:

1) We implemented a block RAM based cache of size 2kB to store experimental spectra and minimize redundant DRAM accesses.
2) We pre-sorted the peptide database which allows the use of binary search to search candidate peptides. The search operation needs to be performed only once per spectrum as next peptide can be pre-fetched, hence achieving input locality.
3) To allow input reuse, we designed a peptide broadcast bus to make it accessible to all the processing elements.
4) We implemented a first-come first-serve (FCFS) based bus arbitration scheme to minimize the synchronization time of processing elements sharing the system bus.

The rest of the paper is organized as follows. Section II explains the background of Xcorr computation problem and related work. Section III describes the proposed architecture. Section IV presents the experimental results. Section V concludes the paper.

978-1-6654-3759-2/21/$31.00 ©2021 IEEE
DOI 10.1109/FPL53798.2021.00024

II. Background

A. Xcorr theoretical formulation

The Xcorr score between a theoretical spectrum vector $\boldsymbol{X}$ and an experimental spectrum vector $\boldsymbol{Y}$ of length n is defined in [8] as,

$$X_{corr} = \sum_{i=0}^{n-1} X[i]Y[i] - \frac{1}{151}\sum_{i=0}^{n-1}\sum_{\tau=-75}^{\tau=75} X[i]Y[i-\tau] \quad (1)$$

where τ is the amount by which vector is being serially shifted. However, SEQUEST implementation performs an optimization by pre-processing the experimental spectrum to perform dot product only once as described in [7] and summarized below,

$$\boldsymbol{Y_P} = \sum_{i=0}^{n-1}\left(Y[i] - \frac{1}{151}\sum_{\tau=-75}^{\tau=75} Y[i-\tau]\right) \quad (2)$$

using (2) reduces the Xcorr computation to

$$X_{corr} = \sum_{i=0}^{n-1} X[i]Y_P[i]$$

B. Related work

A significant amount of work has been done on acceleration of peptide deduction algorithms using FPGA based architectures. Bogdan, Coca and Beynon [5] designed a FPGA based accelerator for peptide deduction using Profound [20] algorithm by instantiating 48 parallel search processors to achieve 950× speed-up over a single core processor. Another study accelerated the scoring process of X!Tandem [15] by instantiating 6 score generation modules and one fragment ion generation module to achieve 17× speed-up over a CPU only implementation. In [19], a complete CPU-FPGA system which was based on [15] but included support for multiple FPGAs and achieved 10-fold speed-up for the entire search operation. To the best of our knowledge, there hasn't been any studies performed on accelerating the SEQUEST [3] algorithm, which is widely used in proteomics.

III. Proposed Architecture

The architectural setting of the heterogeneous computational system for Xcorr is shown in Fig. 1. Host CPU communicates with PCIe DMA via the PCIe link to transfer the experimental spectra from host memory to FPGA memory. A set of directly accessible core control registers, hold computation parameters, and control the operation. Each step of the algorithm takes place inside the processing element (PE) i.e. reading experimental spectrum vectors one by one, searching for candidate peptides, generating theoretical spectrum, computing dot product scores, and writing the results back to main memory. The system allows deployment of multiple PEs which execute the computations in a parallel and asynchronous manner. Since all the PEs share the same memory bus, we implemented a first come first serve (FCFS) based bus arbitration scheme to achieve maximum bandwidth utilization. The detailed view of the bus arbitration scheme is described in Fig. 3.

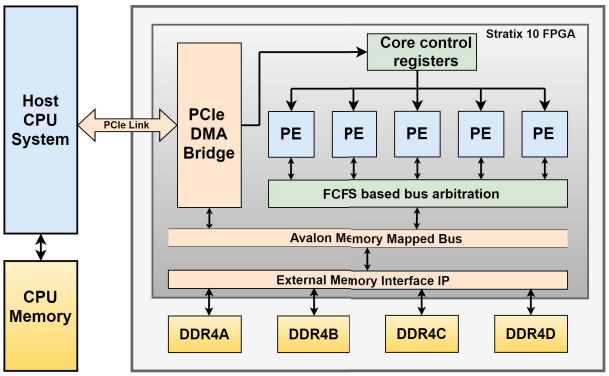

Fig. 1. Complete system architecture shows host CPU communicates with FPGA RAM via PCIe DMA bridge which is connected to Intel's Avalon memory mapped bus. Core registers module contains the computation parameters and is also used for FPGA-CPU communication. To allow efficient use of the Avalon memory mapped bus, all PEs are connected to FCFS based bus arbiter which is in turn connected to Avalon memory mapped bus.

A. Processing Element construction

Each PE takes over the computation of a single spectrum with all the candidate peptides. At the heart of a PE, sits a controller which determines the flow of computation as shown in Fig. 2. The controller copies an experimental spectrum in the form of m/z and ion intensity values from the external memory into on-chip RAM and starts the computation. Once the scores have been computed, they are collected in the on-chip RAM and a request for bus access is generated again to copy the scores into the DRAM.

B. Bus Arbiter Design

To ensure load-balancing among the PEs, we designed a bus-arbiter shown in Fig. 3 which aimed to minimize total wait time for all the PEs. All the PEs requiring access to the bus connect with the "bus request" signal which is connected to a wait counter register. The wait counter register keeps track of the wait time of every bus master so that the decision of contest for bus access is based on fairness i.e. access is granted to a master which has been wait for the longest.

C. Ion-Matching Kernel

To compute the dot product scores between experimental spectra and a candidate peptide, the processing element moves a 64-byte word from the on-chip RAM and a theoretically generated ion-pair from the candidate peptide to the peak-matching circuit. Each 64-byte word has 16 ion-pairs (using 16-bit floating point representation for intensity and 16 bit binary representation for m/z) from the experimental spectrum which is stored in 16 32-bit registers inside the peak-matching circuit. The m/z value of the theoretical ion is compared with all the experimental m/z values using a set of 16 parallel comparators as shown in Fig. 4 and the corresponding matching intensity value of the peak is multiplied and accumulated in the output register. Once all the ions are traversed, the final

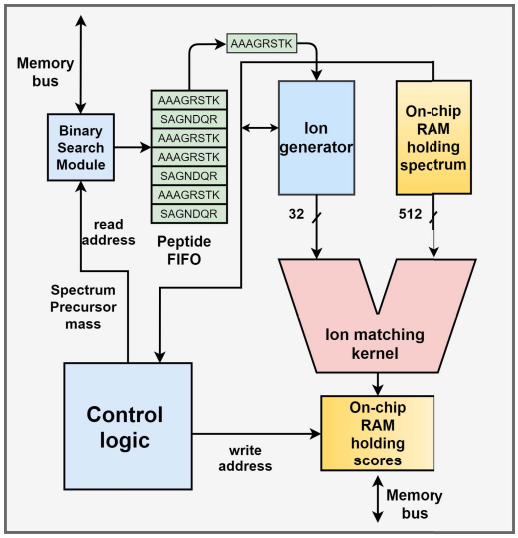

Fig. 2. Detailed internal construction of a single processing element. At the heart is the control logic which controls the function of all the sub-modules in the figure. Binary search module fetches a candidate peptide and stores it in a peptide FIFO. Ion generator reads the peptide and generates fragment ions. A 512 bit packet containing 16 32-bit ion mz and intensity pair values along with a 32-bit theoretical ion and intensity pair values are fed to the ion-matching kernel which finds the matching peak and stores the partial score in on-chip RAM.

Xcorr score is sent to the local on-chip RAM and the process repeats for the next candidate peptide.

IV. Experiments

A. Methodology

We designed the entire hardware using Intel Quartus Pro and Qsys system builder for Intel Stratix 10 FPGA. VHDL description was compiled using Quartus Pro to verify the maximum operable frequency of 200MHz. To evaluate the timing performance of our design, we implemented a cycle accurate simulator in python which mimicked the exact timing response of the hardware. In our simulator, we modeled each sub-module as a class whose data objects represented the internal and external signals of the module and a *clock-event()* method which updated the signals whenever a clock edge occurred.

For our experiments, we used the PXD000612 dataset from PRIDE database which contained 90494 experimental spectra to score against human proteome dataset containing 669964 peptides. The experimental spectra were stored in the compressed sparse row (CSR) format with ion m/z value as the data index and ion intensity value as data element. Ion m/z values were stored in a 16 bit binary format and ion intensity values were represented using 16 bit half-precision floating point format.

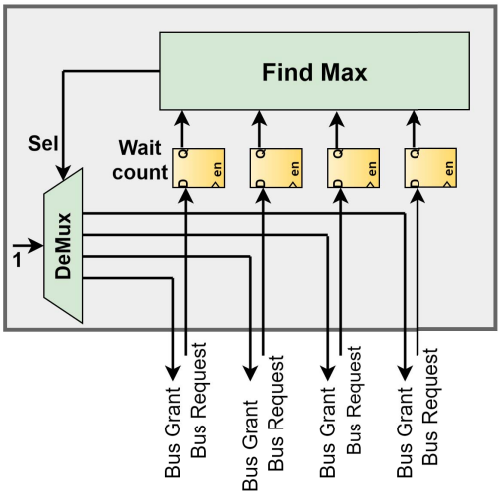

Fig. 3. Brief logic description of bus arbitration module. Bus request lines from all the PEs are coming into the arbiter. When a PE is denied service, its wait count register is incremented, dynamically increasing its priority for the next turn. Find max module is a comparator tree which finds which registers has the maximum value and grants access to the corresponding PE.

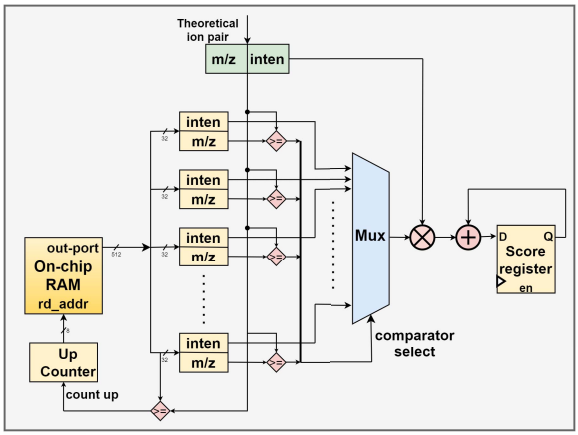

Fig. 4. The ion-matching circuit receives a 512 bit packet containing 16 experimental ions which are all compared with a theoretical ion in one cycle. The matched ions are multiplied and accumulated in the score register. If the theoretical ion is outside the range of current experimental ions, next packet is requested from the onchip-RAM by incrementing the counter.

B. Results

The performance gains in our design come from a combination of optimizations which minimize DRAM accesses and allow input reuse by using an on-chip RAM as a local cache. To find the optimal cache size, we performed a design space exploration for four different cache sizes along with the number of instantiated PEs in the design. The results of these experiments are presented in Fig. 5.

We analyzed the performance of our design by elaborating

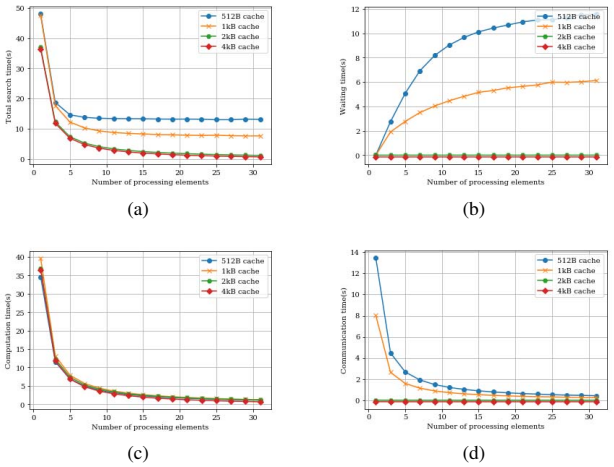

Fig. 5. (a), Total computation time vs number of instantiated PEs for cache size of 512B, 1kB, 2kB, and 4kB is shown. The search time decreases consistently with increasing number PEs for cache size 2kB and 4kB but its saturates for 512B and 1kB after 6 PEs due to increased memory requests. (b) Average synchronization time per PE in the system. This is the time spent by PE waiting for memory bus access. The wait time increases exponentially for cache size below 2kB. (c) Average computation time per PE in the system. This is the time spent on actual computation which decreases with the increasing number of PEs but is not affected by cache size. (d) Average I/O time per PE is also higher for cache size below 2kB. However there is little difference in performance between 2kB and 4kB.

the total processing time spent on computation and communication. To understand the effect of cache size, we further divide the communication time in terms of I/O and waiting time. We define these terms as **Average computation time**: average time each PE spends on computing dot product. **Average I/O time**: average time spent by each PE on DRAM read/write operations. **Average waiting time**: average time each PE spends on waiting to get access to system bus. The total processing time for dot product computations is shown in Fig. 5a, it is evident that increasing the number of PEs from 1 to 31 displayed significant speed up until 15 processing elements for a cache size of 2kB and 4kB, while for cache size below 2kB the speed up is plateaued after 6 processing elements.

Fig. 5b, 5c, and 5d show the breakdown of the total processing time in terms of computation, I/O and waiting time for a single PE. Fig. 5b shows that waiting time is zero for 1 PE as memory bus is not being shared. As the number of PEs are increased, there is an almost exponential increase in the average waiting time of a PE for cache-sizes below 2kB. The waiting times for 2kB and 4kB cache stay constant even when 31 PEs are instantiated. The average computation time per PE is not impacted by the size of cache, but it decreases sharply when processing elements are increased to 11. Fig. 5d shows that the total I/O time i.e. total number of DRAM accesses are orders of magnitude greater for cache size below 2kB. Table I further illustrates that increasing cache-size from 1kB to 2kB results in 600× reduction in the average I/O time.

Based on our experiments, in our final design we instantiated 16 processing elements to achieve maximum performance from the system. We compared the total search time of our design with Crux [12] for 6 different values of precursor mass window. Table II presents the total run-time of Crux running on a 3.6GHz Intel i7-4970 processor with 16GB of system memory and the run-time of our proposed hardware accelerator running at 200MHz clock frequency.

TABLE I
EFFECT OF CACHE SIZE ON AVERAGE I/O AND SYNCHRONIZATION TIME WHILE USING 16 PES

Cache size	I/O time	Waiting time	Total communication time
512B	1.01s	11.57s	12.58s
1kB	0.52s	5.19s	5.71s
2kB	0.86ms	2.2ms	3.06ms
4kB	0.84ms	2.1ms	2.95ms

TABLE II
RUN-TIME COMPARISON OF FPGA ACCELERATOR WITH CRUX RUNNING ON 3.6GHZ INTEL I7-4970 USING 8 THREADS AND 16GB MEMORY.

Precursor mass Tolerance (Da)	Dot product operations	*Crux*	Hardware Accelerator	Relative Speed-up
1.5	162.79M	53.2s	1.25s	42
3	325.41M	75s	2.45s	30
5	541.42M	86s	4.10s	21
10	1.07B	139s	7.782s	20
25	1.99B	304s	20.75s	15
50	3.076B	648s	39.6s	16

V. CONCLUSION

In this paper we designed, and developed an efficient communication-avoiding micro-architecture. By using extensive experimentation, we demonstrated the applicability of custom hardware design approach to accelerate crucial memory bound problems in MS based omics. We presented optimizations for input reuse at all stages of the computation including cache implementation, pre-fetching, and input broadcasting. Although the system was designed for SEQUEST, it can easily be applied for other scoring techniques which involve dot product computation with little modification. Our simulation results suggest that our design is scalable for up-to 32 PEs with linear speed-ups. In future, we plan to extend this work to include the complete peptide identification process involving protein digestion, and build towards a general purpose proteomics processor.

ACKNOWLEDGMENT

Research reported in this paper was supported by NIGMS of the National Institutes of Health under award number: R01GM134384. Fahad Saeed was further supported by the National Science Foundations (NSF) under the Award Numbers NSF CAREER OAC-1925960. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health or the National Science Foundation.

REFERENCES

[1] Ruedi Aebersold and Matthias Mann. Mass spectrometry-based proteomics. *Nature*, 422(6928):198–207, 2003.

[2] Shuichi Asano, Tsutomu Maruyama, and Yoshiki Yamaguchi. Performance comparison of fpga, gpu and cpu in image processing. In *2009 international conference on field programmable logic and applications*, pages 126–131. IEEE, 2009.

[3] Lydia Ashleigh Baumgardner, Avinash Kumar Shanmugam, Henry Lam, Jimmy K Eng, and Daniel B Martin. Fast parallel tandem mass spectral library searching using gpu hardware acceleration. *Journal of proteome research*, 10(6):2882–2888, 2011.

[4] Fabiola Casasopra, Gea Bianchi, Gianluca C Durelli, and Marco D Santambrogio. Parallel protein identification using an fpga-based solution. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 295–299. IEEE, 2016.

[5] Daniel Coca, Istvan Bogdan, and Robert J Beynon. Parallel fpga search engine for protein identification. *Bioinformatics: High Performance Parallel Computer Architectures*, page 313, 2010.

[6] Robertson Craig and Ronald C Beavis. Tandem: matching proteins with tandem mass spectra. *Bioinformatics*, 20(9):1466–1467, 2004.

[7] Jimmy K Eng, Bernd Fischer, Jonas Grossmann, and Michael J MacCoss. A fast sequest cross correlation algorithm. *Journal of proteome research*, 7(10):4598–4602, 2008.

[8] Jimmy K Eng, Ashley L McCormack, and John R Yates. An approach to correlate tandem mass spectral data of peptides with amino acid sequences in a protein database. *Journal of the american society for mass spectrometry*, 5(11):976–989, 1994.

[9] Andy T Kong, Felipe V Leprevost, Dmitry M Avtonomov, Dattatreya Mellacheruvu, and Alexey I Nesvizhskii. Msfragger: ultrafast and comprehensive peptide identification in mass spectrometry–based proteomics. *Nature methods*, 14(5):513–520, 2017.

[10] Chuang Li, Kenli Li, Tao Chen, Yunping Zhu, and Qiang He. Swtandem: a highly efficient tool for large-scale peptide identification with parallel spectrum dot product on sunway taihulight. *Bioinformatics*, 35(19):3861–3863, 2019.

[11] You Li, Hao Chi, Leihao Xia, and Xiaowen Chu. Accelerating the scoring module of mass spectrometry-based peptide identification using gpus. *BMC bioinformatics*, 15(1):1–11, 2014.

[12] Sean McIlwain, Kaipo Tamura, Attila Kertesz-Farkas, Charles E Grant, Benjamin Diament, Barbara Frewen, J Jeffry Howbert, Michael R Hoopmann, Lukas Kall, Jimmy K Eng, et al. Crux: rapid open source protein tandem mass spectrometry analysis. *Journal of proteome research*, 13(10):4488–4491, 2014.

[13] Jeffrey A Milloy, Brendan K Faherty, and Scott A Gerber. Tempest: Gpu-cpu computing for high-throughput database spectral matching. *Journal of proteome research*, 11(7):3581–3591, 2012.

[14] Eriko Nurvitadhi, Jaewoong Sim, David Sheffield, Asit Mishra, Srivatsan Krishnan, and Debbie Marr. Accelerating recurrent neural networks in analytics servers: Comparison of fpga, cpu, gpu, and asic. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–4. IEEE, 2016.

[15] Jin Qiu, Ping Kang, Li Ding, Yipeng Yuan, Wenbo Yin, and Lingli Wang. Fpga acceleration of the scoring process of x! tandem for protein identification. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–4. IEEE, 2017.

[16] Rovshan G Sadygov, Jimmy Eng, Eberhard Durr, Anita Saraf, Hayes McDonald, Michael J MacCoss, and John R Yates. Code developments to improve the efficiency of automated ms/ms spectra interpretation. *Journal of proteome research*, 1(3):211–215, 2002.

[17] Fahad Saeed. Communication lower-bounds for distributed-memory computations for mass spectrometry based omics data, 2020. unpublished.

[18] Leheng Wang, Wenping Wang, Hao Chi, Yanjie Wu, You Li, Yan Fu, Chen Zhou, Ruixiang Sun, Haipeng Wang, Chao Liu, et al. An efficient parallelization of phosphorylated peptide and protein identification. *Rapid Communications in Mass Spectrometry*, 24(12):1791–1798, 2010.

[19] Moucheng Yang, Tao Chen, Xuegong Zhou, Liang Zhao, Yunping Zhu, and Lingli Wang. A complete cpu-fpga architecture for protein identification with tandem mass spectrometry. In *2019 International Conference on Field-Programmable Technology (ICFPT)*, pages 295–298. IEEE, 2019.

[20] Wenzhu Zhang and Brian T Chait. Profound: an expert system for protein identification using mass spectrometric peptide mapping information. *Analytical chemistry*, 72(11):2482–2489, 2000.