Smith-Waterman Implementation on a FSB-FPGA module using the Intel Accelerator Abstraction Layer

Jeff Allred, Jack Coyne
William Lynch and Vincent Natoli
Stone Ridge Technology
2107 Laurel Bush Road
Bel Air, MD 21015
Email: jallred@stoneridgetechnology.com

Joseph Grecco
Intel Corporation
77 Reed Road
Hudson, MA 01749
Email: joe.grecco@intel.com

Joel Morrissette
Intel Corporation
5300 NE Elam Young Parkway
Hillsboro, OR 97124
Email: joel.morrissette@intel.com

***Abstract*—The Smith-Waterman algorithm is employed in the field of Bioinformatics to find optimal local alignments of two DNA or protein sequences. It is a classic example of a dynamic programming algorithm. Because it is highly parallel both spatially and temporally and because the fundamental data structure is compact, Smith-Waterman lends itself very well to operation on an FPGA. Here we demonstrate an implementation of this important algorithm in a novel FSB module using the Intel Accelerator Abstraction Layer (AAL), a newly released software middleware layer. We have modified SSEARCH35, an industry standard open-source implementation of the Smith-Waterman algorithm, to transparently introduce a hardware accelerated option to users. We demonstrate performance of nine billion cell updates per second and discuss further opportunities for performance improvement.**

I. Introduction

In recent years a number of competing technologies for software acceleration have emerged. While it is still unclear what sustained role they will hold on future computing platforms it is clear that there are important applications that present an opportunity for acceleration on non-traditional platforms. Reconfigurable hardware such as FPGAs, for example, has been made available to users by several High Performance Computing (HPC) hardware vendors and used effectively to accelerate a wide variety of important algorithms. Among the most recent tools available for reconfigurable computing are socket replacement modules (FPGA platforms that substitute directly for a CPU in-socket) and middleware software abstraction layers.

In this work we address the Smith-Waterman algorithm[1], a highly parallel Bioinformatic algorithm that lends itself well to implementation on FPGAs. The hardware platform targeted is the XtremeData XD2000i FSB module. Using AAL, we have integrated our hardware accelerated Smith-Waterman function into SSEARCH35[2], an open source industry standard.

II. XtremeData XD2000i FSB Module

The XtremeData, Inc. XD2000i hardware accelerator is a direct hardware replacement for the Intel® Xeon® microprocessor in dual-socket server and blade systems. Plugging directly into the Xeon Front Side Bus (FSB) via one of the system's CPU sockets, the XD2000i provides three Altera® EP3SL150 Field Programmable Gate Arrays (FPGAs) in a tiered topology, each providing 142,000 equivalent logic elements (LE). The "bridge" FPGA abstracts the FSB through an encrypted core, providing 8.5GB/s FSB access and two local 9.6GB/s dedicated ports to the secondary "compute" FPGAs. The compute FPGAs are configured at runtime with user-defined logic to provide hardware-assisted acceleration as required by the host applications running on the Xeon® processor(s) in the system. Each compute FPGA is provided with 8MB of 350MHz QDRII+ RAM scratchpad memory. A block diagram of the FSB module appears in Fig 1.

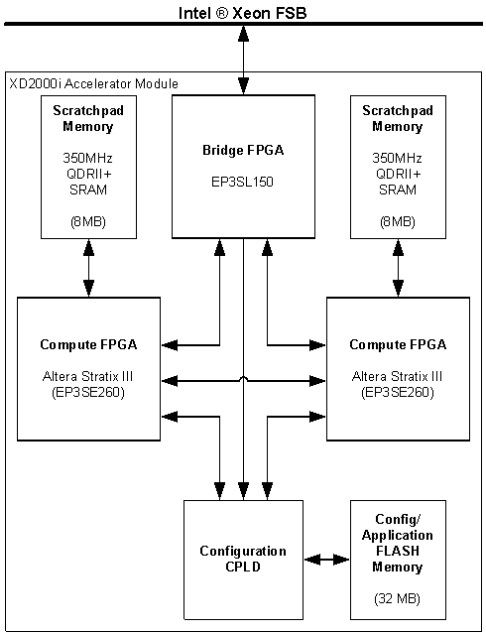

Fig. 1. Block Functional Diagram of Xtreme Data XD2000i Module

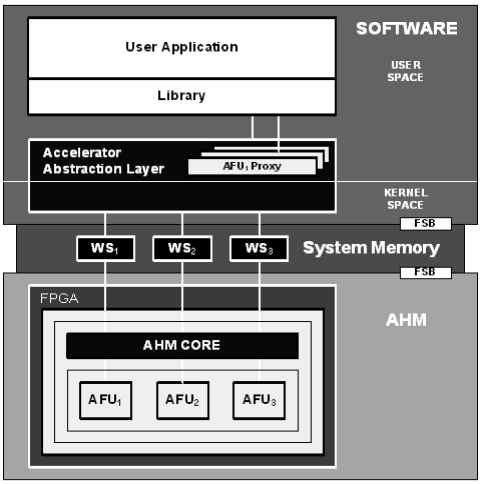

Fig. 2. Intel Accelerator Abstraction Layer software stack

III. Intel Accelerator Abstraction Layer (AAL)

Intel® QuickAssist Technology is a comprehensive initiative that consists of a family of interrelated Intel® and industry standard technologies that enable optimized use and deployment of accelerators on Intel® platforms[3], [4]. QuickAssist facilitates the integration of hardware accelerators into Intel® platforms by defining a common usage paradigm through the Accelerator Abstraction Layer (AAL). AAL defines a common software framework (Service APIs) and primitives for accessing QuickAssist accelerators that have been discovered and registered via the Accelerator Abstraction Services (AAS). Each accelerator is accessed via its existing interconnect interfaces through Accelerator Interface Adaptors (AIAs), which provide a standardized and hardware-agnostic interface to the underlying Accelerator Function Units (AFUs) implemented in hardware. In this manner, AAL provides a single programming model that supports multiple simultaneous applications on multiple compute cores. Figure 2 presents a block diagram of the AAL software stack.

By presenting a uniform discovery mechanism and messaging interface, AAL provides a migration path across multiple families and generations of accelerator implementations with minimal additional development effort. AAL is particularly well suited to asynchronous and massively parallel applications like the kind found in Smith-Waterman, enabling the developer to realize linear performance gains with the integration of additional accelerator resources.

From the software developer's perspective AAL presents a uniform interface to a library of accelerated functions. AAL manages the physical hardware on which the algorithm executes allowing the user to experience the benefit of hardware acceleration transparently. The hardware developer is similarly presented with a uniform interface this time in the bridge FPGA controlling the FSB interaction. By developing to this interface and registering the application with AAL, the hardware becomes available to software developers.

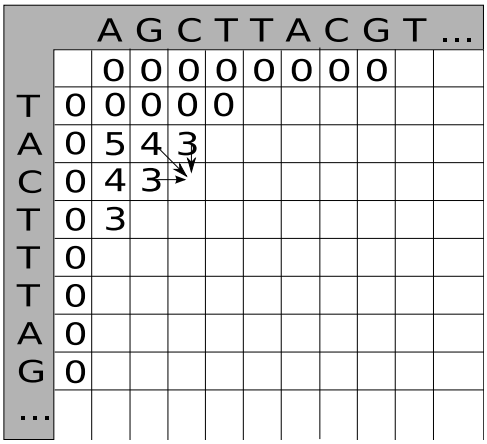

Fig. 3. Smith-Waterman Similarity Matrix

IV. Smith-Waterman Algorithm

The problem of sequence similarity is recurrent in the fields of Genetics and Bioinformatics. When a new gene is discovered its role and function may be inferred by its similarity to known sequences. The general problem is then to gain a measure of similarity between two separate sequences of 'letters' drawn from an 'alphabet'. In genetics the focus is usually on sequences of nucleic acids drawn from the set of four possibilities Adenine ('A'), Guanine ('G'), Cytosine ('C') and Thymine ('T') for DNA or sequences of amino acids that make up proteins drawn from an alphabet of 20 possible amino acids. A typical problem will search for the best match of a query string inside a much larger database string. Methods for solving this problem with various additional boundary conditions are well known and drawn from the class of dynamic programming methods. Smith-Waterman is a dynamic programming method which finds the best local alignment, typically between a shorter query string inside a longer database string.

The problem statement is to find the best local alignment of the string Q within the string D, where the length of Q is n and the length of D is m. The strings may be represented as $Q_1Q_2Q_3 \ldots Q_n$ and $D_1D_2D_3 \ldots D_m$.

Thus we wish to find a subsequence inside of Q which maximizes a metric of similarity with some portion of the string D. The Smith-Waterman algorithm and its derivatives such as that of Gotoh[6] and Hirchberg[7] solve this problem by constructing a similarity matrix, the elements of which each hold a score that is calculated recursively. The score of any individual cell in the matrix, V_{ij} is a function of the scores of its neighbors to the top, left and top-left diagonal, in addition to D_j and Q_i. The recursion relation is given by:

$$V_{ij} = max \begin{cases} 0 \\ V_{i-1,j-1} + \sigma(Q_i, D_j) \\ V_{i-1,j} + \sigma(Q_i, -) \\ V_{i,j-1} + \sigma(-, D_j) \end{cases}$$

The operation of the algorithm is best illustrated by considering the similarity matrix shown by example in Figure 3.

The matrix is formed with the string D as the first row at the top of the matrix and the string Q as the left-most column. The left-top corner is not used so the matrix is aligned in such a way that the diagonal elements connect equivalently positioned characters in Q and D.

The scoring matrix σ determines the relative weighting of matches, mismatches, deletions and insertions. A sample scoring matrix for DNA sequencing is:

$$\sigma = \begin{pmatrix} 5 & -1 \\ -1 & -4 \end{pmatrix}$$

This scoring matrix increments the score by 5 for a match, decrements by 4 for a mismatch and penalizes insertions and deletions by 1. Figure 3 shows the score filled in for the first few cells in the upper left corner of a similarity matrix.

The basic Smith-Waterman algorithm described above penalizes the initiation and the extension of a gap equivalently. It is frequently the case that a larger penalty is applied to the initialization of a gap then to its extension. This affine gap model introduced by Gotoh[6] is accomplished by a relatively easy modification to the basic recursion relations by introducing two additional parameters, the left gap and the top gap. The new recursion relation for the affine gap model is given by:

$$\begin{aligned} V_{ij} &= max \begin{cases} 0 \\ L_{ij} \\ T_{ij} \\ V_{i-1,j-1} + \sigma(Q_i, D_j) \end{cases} \\ L_{ij} &= max \begin{cases} V_{i,j-1} - \alpha \\ L_{i,j-1} - \beta \end{cases} \\ T_{ij} &= max \begin{cases} V_{i-1,j} - \alpha \\ T_{i-1,j} - \beta \end{cases} \end{aligned}$$

After completing the score calculation for the similarity matrix, the highest scoring cell is identified as the terminating point of the optimal alignment. To obtain the alignment itself one must backtrace from the high scoring cell to the point where the score becomes exactly zero. Backtracing requires each cell to store the information defining which of the three possible max values was chosen for the cell's score i.e. the information about which of the three possible neighbor cells originated its score value. In SSEARCH35, the similarity matrix is divided into tiles along the direction of the database string which is typically very large. The Smith-Waterman score is calculated for each of the tiles and the maximum is recorded. The actual backtrace is done in software by rescoring the maximum score tile and keeping track of the required alignment data.

V. Smith-Waterman FPGA Implementation

The Smith-Waterman dynamic programming algorithm progresses a calculation front that is perpendicular to the diagonal of the sequence matrix. It lends itself naturally to a systolic array with data streaming through element by element.

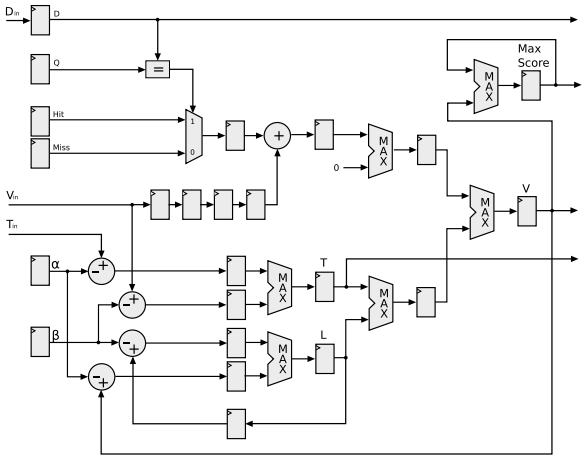

Fig. 4. Smith-Waterman Processing Element Block Diagram.

Each of our processing elements represents one particular row of the matrix and is responsible for the calculation of the scores of all elements in its row. The calculation of a score is dependent on the scores above it and to the left. This allows a march down the row, storing the result in a register for the element below and using the results in the register of the processing element above. The input and output rate of each processing element is fixed and equal, so there is no need for buffering data.

Each processing element includes an input register that is fed from the output of the previous processing element. This input register has a control field to specify the current action for the processing element and a data field. Inbound and outbound datapaths of the processing element are 64 bits wide.

The most important record type is the "Process Data" command. This record supplies the D, V, L and T from the previous row's machine. The current score is stored and put in the next processing element. The current gap based scores are put in L and the next processing elements T values. The "Process Data" command takes four cycles to complete and is the longest of all the possible commands. The benefit of extending the other commands to four pipeline stages is that there is no need to buffer data between stages. This dramatically reduces resource usage.

The design includes five records that contain operational codes and data. *Initialize Alpha and Beta Registers*: This record supplies the scoring values Alpha and Beta. *Hit and Miss Registers*: This record supplies the scoring values Hit and Miss. *Initialize Q Register*: This record supplies query string value Q to each machine. *Initialize VDiagonal Register*: This record supplies the initial V Value in the upper left diagonal cell for the first row of the first machine. *Process Cell*: This record instructs the core to process and score a single cell of the entire array.

VI. Smith-Waterman Software Implementation

Our goal in this project was to employ AAL and the FSB module to accelerate an application code that is commonly used by practitioners of Bioinformatics. We chose

the application SSEARCH35, a component of the FASTA package. The vast majority of execution time was spent in the FLOCAL_ALIGN function which executes the Smith-Waterman algorithm.

SSEARCH35 was designed to accommodate multiple implementations of FLOCAL_ALIGN and defines a standard approach to add an alternative implementation. The method involves editing three functions. They include a one-time startup call (init_work), a one-time shutdown call (close_work), and an execute call (do_work) that is called many times.

In the startup call, we allocate page locked memory for the execute call, initialize AAL, and obtain an exclusive lock on the hardware device. Likewise, in the shutdown call, we release the device lock, and release our memory back to the operating system.

The do_work() function is modified to call our hardware replacement for FLOCAL_ALIGN. In our implementation, we can send an arbitrary length of the database, however, we can only send one character of the query string per processing engine. If, for example, the query string is 300 characters long, and there are only 100 processing engines, we would create three tiles to work over. In this case, each tile will compute the scores for 100 characters of the query string and the entire length of the database string. The information that is returned from the last row of the previous tile is the exact starting data for the next tile.

VII. Results and Conclusion

We have ported the Smith-Waterman algorithm, a foundational algorithm for DNA sequencing to an XtremeData XD2000i FSB-FPGA module using the Intel® AAL middleware layer.

Our design is a linear systolic array of processing elements that operate along the rows of the similarity matrix. The current design fits 300 processing elements each of which requires 287 equivalent logic elements and 467 registers. The total design occupies 86% of the chip and operates at 200MHz. It performs nine billion cell updates per second (bcups).

The Smith Waterman algorithm has been implemented on FPGAs in previous work[8][9]. The uniqueness of this work is found in the use of the new FSB module and integration with the AAL middleware layer. The most relevant comparison is with the work by Zhang et. al. [8] which claims 25 bcups. Our performance data was recorded for very large database strings against queries that were a few hundred base pairs long. The SSEARCH35 software tiles the similarity matrix in the direction of the database string so there is some overhead in moving from tile to tile which accounts for some if not all of the difference between our observed performance and our theoretical performance. In addition the design tested used 20 bits to represent the score. This could be easily reduced to shrink the size of the PE's and fit more on the chip. With these considerations, our results are comparable to those previously published.

Our application is found to be limited by computational intensity and not bandwidth considerations. The XD2000i module has an optimal latency of 0.5μsec for a roundtrip data transfer from memory through the module and back to memory. Actual results range up to twice that amount. Burst characteristics for the module likewise range up to twice that of sustained bandwidth. Smith Waterman as an application is extremely computationally intense another attribute that makes it a good candidate for hardware acceleration. For each byte of data transferred into the device, many calculations are done. In practice this meant that the application was never found waiting or starved for data. The bandwidth and latency characteristics of the XD2000i module were sufficient to supply the FPGA with a constant stream of data. Breaks in the data stream were of course introduced intentionally by the software tiling of the similarity matrix.

The implementation of AAL in our user code was easily accomplished and will be well within the reach of those with application development experience. As a middleware layer, AAL is fairly lightweight and resulted in no discernible performance degradation. Since AAL is implemented as an asynchronous message passing system it allows for the host to perform other operations while waiting for the FPGA call to return. This contrasts with the typical use-case for co-processors in which the host is blocked during co-processing activities. The development required an effort equivalent to that of a more direct non-AAL implementation addressing the module driver. With broader adoption, the advantages of the interface will become more apparent as the same interface can address a variety of different accelerating platforms.

A number of design changes are currently being investigated to improve the performance further. These include operating on the second module FPGA (2x), unrolling the inner pipeline (4x), increasing the clock (1.5x) and reducing the score resolution from 20 bits to 8 bits (2.5x). These improvements can theoretically achieve a 30x improvement over our current implementation. A further factor of 1.6 is also possible from the difference in our achieved performance vs. theoretical maximum. These estimates put a theoretical performance maximum at about 450 bcups.

References

[1] T. F. Smith and M.S. Waterman, Identification of common molecular subsequences, *J. Mol. Biol.*, **147:195-197**, 1995.

[2] W. Pearson and D. Lipman, *Improved tools for biological sequence comparison*, Proc. Natl. Acad. Sci USA, Apr 85(8):2444-8, 1988

[3] Intel Corporation, White Paper, Enabling Consistent Platform-Level Services for Tightly Coupled Accelerators

[4] Intel Corporation, *Intel Quick Assist Technology FSB-FPGA Accelerator Platform Software Developer's Manual*

[5] Altera Corp, White Paper, *Implementation of the Smith-Waterman Algorithm on a reconfigurable supercomputing platform*, September 2007

[6] O. Gotoh, An improved algorithm for matching biological sequences, *J. Mol. Biol.*, **162:705-708**, 1982

[7] D. Hirschberg, A linear space algorithm for computing maximal common subsequences, *Communication of ACM*, **18:341-343**, 1975

[8] P. Zhang, G. Tan and G. Gao, Implementation of the Smith-Waterman algorithm on a reconfigurable supercomputing platform. Proc. Int. Workshop on High-performance Reconfigurable Computing Technology and applications (HPRCTA), pg 39-48, New York, NY, USA, 2007

[9] TimeLogic Corp, www.timelogic.com