IN-SOCKET FPGA IMPLEMENTATION OF BIOINFORMATIC ALGORITHMS USING THE INTEL AAL

Jack Coyne, Jeffrey Allred, William Lynch and Vincent Natoli

Stone Ridge Technology, 2107 Laurel Bush Rd., Bel Air, MD 21015

ABSTRACT

We demonstrate the implementation of two important bioinformatic algorithms in the XtremeData XD2000i FSB module using the Intel Accelerator Abstraction Layer (AAL). We have modified SSEARCH35 and NCBI BLAST, industry standard open-source implementations of the Smith-Waterman and BLAST algorithms respectively, to transparently introduce a hardware accelerated option to users.

1. INTRODUCTION

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 on sequences of nucleic acids drawn from the set of four possibilities Adenine ('A'), Guanine ('G'), Cytosine ('C') and Thymine ('T') for DNA. Two critical algorithms in this endeavor are Smith-Waterman and BLAST. Smith-Waterman is a dynamic programming algorithm which finds the exact optimal local alignment, typically between a shorter query string inside a longer database string, BLAST has a similar role but provides results using an approximation that qualifies them with a probability or likelihood.

2. 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. Each compute FPGA is provided with 8MB of 350MHz QDRII+ RAM scratchpad memory.

3. 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[1, 2]. QuickAssist defines 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.

4. SMITH-WATERMAN

The Smith-Waterman dynamic programming algorithm[3] progresses a calculation front that is perpendicular to the diagonal of the sequence matrix formed by the database vs the query. It lends itself naturally to a systolic array with data streaming through element by element. It lends itself well to hardware implementation and has been addressed on different hardware platforms by others[4, 5, 6, 7].

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. The processing element design is illustrated in Figure 1.

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

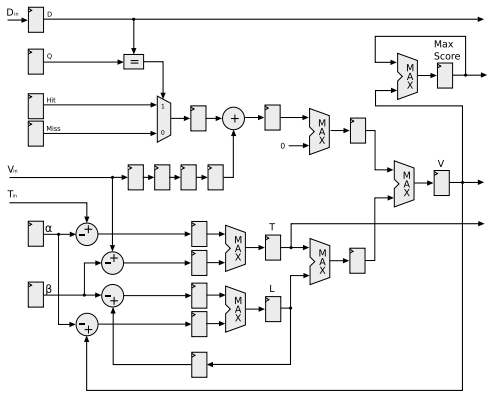

Fig. 1. Smith-Waterman Processing Element Block Diagram.

wide. The SSEARCH35[8] component of FASTA was modified to support the hardware implementation for our work.

5. BLAST

The fundamental approach of BLAST consists of the following three steps *i*) generate a table of n-element sequence segments that would score greater than a chosen minimum when compared against the query sequence *ii*) scan the database sequence for segments that match entries in the table of high scoring alignments *iii*) extend matches in both directions to find the alignment with a locally maximal score. For this work, we have focused our efforts on accelerating a specific configuration of blastn, the BLAST routine for comparing nucleotide sequences.

In the first phase of BLAST, a hash table of depth 4^N is constructed using the query sequence. The construction of the table occurs as follows: *i*) step through the query sequence 1 nucleotide at a time *ii*) at each step, form a word using the next N nucleotides *iii*) using the word from step (*ii*) as a direct index into the table, mark the entry as valid *iv*) populate this entry in the hash table with the query offset at which the segment of interest occurred *v*) for indexes that occur multiple times, generate an overflow table containing all offsets, and set the entry of the primary hash table to point to their location in the overflow table. The design is illustrated in Figure 2. When a hit occurs on any processing channel, the sequence segment is pushed into a FIFO for processing by the second stage. During stage 2, entries corresponding to hits generated by stage 1 are retrieved from the hash and overflow tables. Stage 2 is illustrated in Figure 3.

Hits from stage 1 are buffered in a set of FIFOs. If a hit retrieved from the buffer is not an overflow hit, then it is looked up directly in the hash table to obtain the sequence segment neighbors for that entry. These are then compared against the neighbors of the database segment that triggered

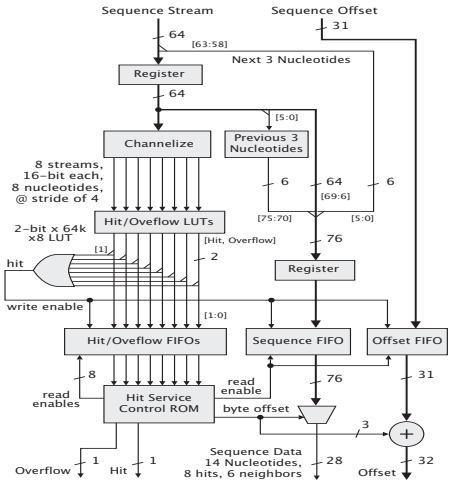

Fig. 2. Stage 1 of the hash table lookup

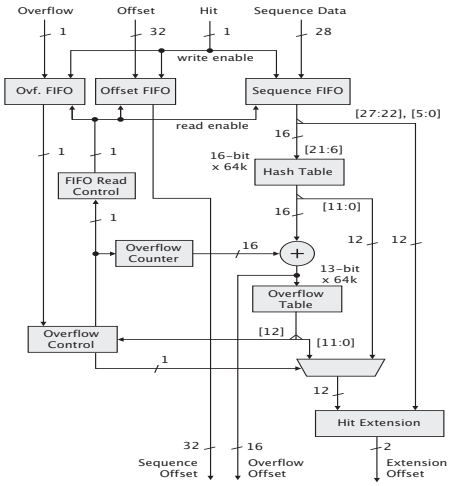

Fig. 3. Stage 2 of the hash table lookup

the hit. When a hit retrieved from the buffer is marked as an overflow, the entry from the hash table is used as an index into the overflow table. When matches are found, the overflow table offsets are returned to software. Once an overflow lookup has completed, the next hit is retrieved from the input FIFOs, and the lookup process repeats.

The exact matches of 11 nucleotides that were found during the hash table based search become the inputs to the gapped alignment phase. In this phase, gaps and mismatches may enter into the alignment. Alignments that achieve a local maximum score are returned. The process is almost identical to the Smith-Waterman method where matches are rewarded by increasing an alignment's score, while mismatch and gaps are penalized by decreasing its score. The gapped alignment machine design is illustrated in Figure 4.

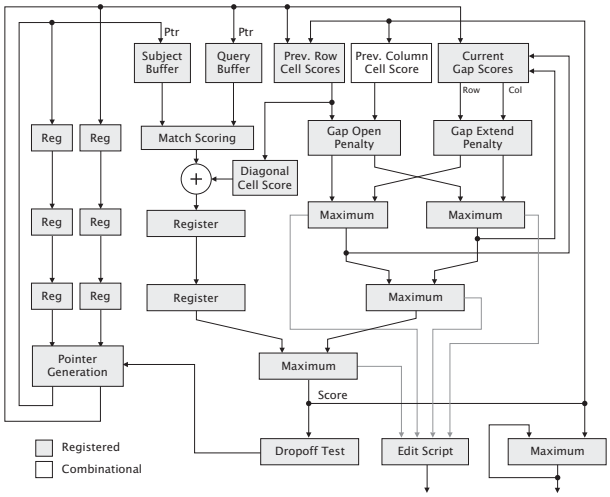

Fig. 4. Gapped alignment machine

6. RESULTS AND CONCLUSION

We have ported the Smith-Waterman and BLAST algorithms for DNA local sequencing to an XtremeData XD2000i FSB-FPGA module using the Intel® AAL middleware layer.

For Smith-Waterman, our design is a linear systolic array of processing elements that operate along the rows of the similarity matrix. The total design occupies 86% of the chip and operates at 200MHz and performs nine billion cell updates per second (bcups). Performance data was recorded for very large database strings against queries that were a few hundred base pairs long.Our application is found to be limited by computational intensity and not bandwidth considerations.

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.

We have also described a hardware design that provides acceleration of subroutines relevant to the BLAST nucleotide sequence alignment algorithm. The co-processor is capable of detecting 11-mer matches between a query sequence and a database sequence at a rate of 16 billion nucleotides per second, with up to 500 million matches per second. The remaining resources in the co-processor are used for gapped extension of sequence segment alignments. The gapped extension stage can achieve about 8 billion cell updates per second. This translates to about 0.5 million full extensions per second.

The performance of the current implementation is limited by on chip memory. The 8-channel design can process sequences at a rate of 8 billion nucleotides per second (when striding by 4), with up to 250 million complete lookups (matches processed) per second. The module provided about 25 Gbps of throughput when the co-processor was being developed, and is specified for up to 64 Gbps when taking full advantage of the bus.

The gapped alignment machine requires 600 logic blocks and 1100 registers. It requires 624kbits of block memory, which is mostly consumed by the edit script. Limited by memory, we can place approximately 20 machines on the Stratix III E260, and achieve about 15 billion cell updates per second at a clock rate of 200 MHz, depending on the selected drop-off value, which effects how frequently the pipeline is flushed.

7. REFERENCES

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

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

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

[4] ltera Corp, White Paper, *Implementation of the Smith-Waterman Algorithm on a reconfigurable supercomputing platform*, eptember 2007

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

[6] . Hirschberg, A linear space algorithm for computing maximal common subsequences, *Communication of ACM*, textbf18:341-343, 1975

[7] . 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

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