A FIELD PROGRAMMABLE GATE ARRAY CO-PROCESSOR FOR THE BASIC LOCAL ALIGNMENT SEARCH TOOL

Jack Coyne, Jeff Allred, Vincent Natoli, William Lynch

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

ABSTRACT

We present the architecture of a field programmable gate array (FPGA) based system that provides acceleration of a Basic Local Alignment Search Tool (BLAST) algorithm for bioinformatic sequence analysis. A front side bus (FSB) attached FPGA is employed in the construction of a co-processor that provides acceleration of subroutines relevant to BLAST. A modified version of the National Center for Biotechnology Information (NCBI) BLAST software package takes advantage of the co-processor when exercised with the appropriate set of parameters. Two portions of the blastn (nucleotide alignment) algorithm are accelerated in hardware: 1) A modified hash table approach is used to compare a query sequence to a database sequence in order to find all exact matches of 11 nucleotides in length. 2) A dynamic programming method is used to extend these matches into high-scoring alignments indicating the origins and structures of locally maximal segment pairs (MSPs).

1. INTRODUCTION

The Basic Local Alignment Search Tool (BLAST) family of algorithms is widely used for analysis of bioinformatic data sets. BLAST allows molecular biologists to analyze protein and nucleotide (DNA, RNA) sequences for structural similarities, which might indicate functional similarities or be of other significance to scientific research. In designing BLAST, its authors took steps that reduce number of computations required during the sequence alignment process while having a minimal impact on the accuracy of its results. Their work [1] shows that we may construct a table that allows us to quickly compare two sequences to find segments scoring above some minimum value. The authors of BLAST call attention to the fact that very low-scoring (short) segment pair matches occur with high frequency, even in unrelated sequences, and should therefore be ignored. They show that by using this kind of selectivity in searching "BLAST is an order of magnitude faster then existing sequence comparison tools of comparable sensitivity".

Initial matches are extend using a dynamic programming approach, which is hastened by choosing to discontinue extension attempts if the alignment score drops to certain level below the current local maximum score. Available software packages implement BLAST with many variations and with parameters for optimizing the alignment process for individual data sets; however, the fundamental approach is simple, and consists of the following three steps:

1. generate a table of n-element sequence segments that would score greater than T when compared against the query sequence
2. scan the database sequence for segments that match entries in the table of high scoring alignments
3. 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. The current implementation performs the initial phase of the search by narrowing the scope to starting points that are a exact matches of 11-nucleotides. The second phase employs dynamic programming, with an adjustable drop-off criterion, for gapped extension to a locally maximal segment pair (MSP).

2. HASH TABLE SEARCHING

In the first phase of BLAST, a hash table of depth 4^N is constructed using the query sequence. The table allows each n-element segment of the database sequence to be compared against the entire query sequence using a single look-up operation. The construction of the table occurs as follows:

1. step through the query sequence 1 nucleotide at a time
2. at each step, form a word using the next N nucleotides
3. using the word from step 2 as a direct index into the table, mark the entry as valid (because the occurrence of this segment in the database would constitute a viable alignment seed)
4. populate this entry in the hash table with the query offset at which the segment of interest occurred

5. 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

For example, if the query offset 100 pointed to the segment ACGTACGT, this could be represented in binary as 0001101100011011, or in decimal as 6939. Therefore, hash table entry 6939 would be set to 100, and marked as a valid hit. If offset 500 was also followed by the sequence ACGTACGT, then the values 100 and 500 would be placed in the next available location in the overflow table, and the hash table would be set to point to that overflow entry. An additional field in the overflow table is used to indicate the end of each overflow entry group.

The default minimum length for a match in blastn is 11 nucleotides. If we construct a table with an 8-nucleotide index, then we may scan through the database with a stride of 4 nucleotides and be certain that at least one step in the scan will overlap with each match of 11 or more nucleotides. This has the benefit of being a byte-aligned operation, along with the obvious saving associated with using a stride greater than one. Once a match of length 8 has been found, we can easily check the neighboring nucleotides to determine if the segment belongs to a match of length 11.

For the co-processing method described here, we have adopted a second type of hash table for use in conjunction with the one described above. Because we wish to minimize the requirements for bandwidth in retrieving results from the FPGA, it is advantageous to do the extension to 11 nucleotides within the FPGA, thereby reducing the probability of finding a hit. This is accomplished by setting the FPGA hash and overflow table entries to contain the 6 nucleotides that neighbor the query segment to the left and right so that we may do the initial lookup and the extension in one pass. When an 11-mer is found, the database offset is sent to software so that the table index maybe reconstructed for use with the original software lookup table. In this manner, the FPGA is used as a filter that narrows focus of the software routines down to a small subset of the original search space. The hash table look-up occurs in two stages that together replace the NCBI BLAST [2] routine "s_BlastNaScanSubject_8_4". These stages, which are described in the following subsections, also implement the portion of "s_BlastSmallNaExtend" for extending matches to 11 nucleotides.

2.1. 8-mer Filtering

During the first stage (illustrated in Figure 1) a narrow lookup table of dimensions 2-bits-by-64k is used to determine whether or not the current word constitutes a hit, and if so wether it is an overflow hit. By keeping this table narrow, we are able to replicate it several times in order to process multiple database segments simultaneously. When a hit occurs on any processing channel, the sequence segment is pushed into a FIFO for processing by the second stage. Hits may occur simultaneously on multiple channels. A small ROM is used as a priority encoder for selection of which hit should be serviced on the current cycle, and to determine when the FIFO read pointer should be incremented. Based on which channel is being serviced, an appropriate byte offset is added to the sequence offset value. The byte offset also controls a multiplexer, which selects the originating sequence segment, and its neighbors, as input to the second stage.

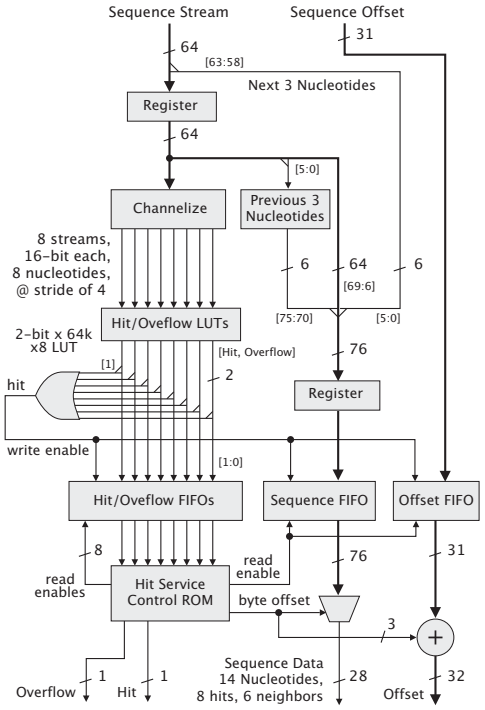

Fig. 1. Stage 1 of the hash table lookup

2.2. Hash Lookup and Extension

During stage 2, entries corresponding to hits generated by stage 1 are retrieved from the hash and overflow tables. The hash table is 12 to 16-bit wide, and 64k deep, and therefore requires a significant portion of FPGA memory resources. The minimum width is 12 because we must store the 6 neighbors of the query segment described by the table entry. Because the table must also store indexes into the overflow table, a width of 16-bits is used to accommodate overflow tables with up to 64k entries. Stage 2 is illustrated in Figure 2.

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

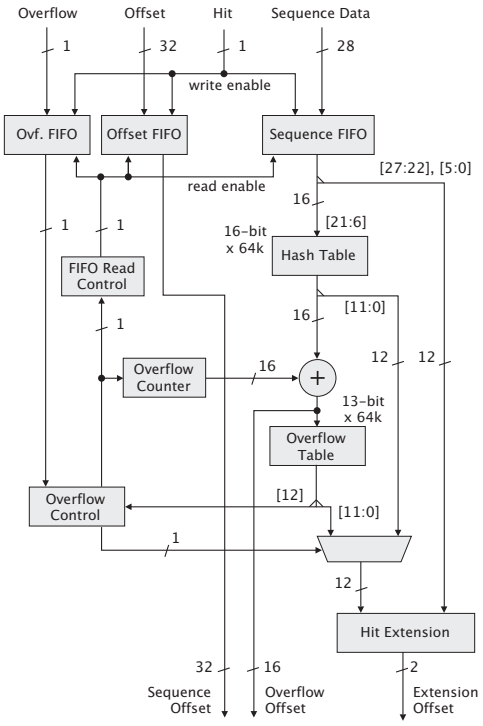

Fig. 2. Stage 2 of the hash table lookup

the hit. Matches are passed back to software along with the offset at which they occurred. 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. An adder and a counter are used to generate subsequent overflow table indices until a terminating word (0xFFFF) is reached at the end of the overflow group. Comparison with neighbors occurs for all overflow table entries corresponding to the database segment that originally triggered the hit. 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. For shorter queries, the probability of finding an 11-mer may be low enough so that the FIFOs are never full, and the database may be streamed through the two stages at full rate. For larger queries, the hash table population may be more dense, leading to more hits. In this case the FIFOs may become full, causing the stream rate to be throttled back in order to avoid FIFO overflow.

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 per second. Each overflow entry requires a lookup cycle, and there is a 4 cycle penalty for each overflow burst due to pipeline latency, however, because hits are sparsely distributed the rate of 8 billion should typically be achieved. The Altera Stratix III E260 provides enough memory to allow potential replication of the hash and overflow tables. The device can theoretically achieve rates of up 32 billion nucleotides per second. This translates to over 64 Gbps in bandwidth requirements for a fully utilized device. The co-processor has been designed using a front side bus (FSB) attached FPGA module from XtremeData, Inc, and the Intel Accelerator Abstraction Layer (AAL). The module was providing 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 current implementation of the BLAST co-processor uses an 8 or 16 channel lookup stage, with remaining resources and bandwidth allocated to gapped alignment phase.

3. GAPPED ALIGNMENT

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. Because alignments often span across multiple 11-mers, the software routine continuously removes from the list any 11-mers that have already been covered by previous extensions. The cycle of extending a 11-mer, followed by culling of the 11-mer list, repeats until the list is empty.

The alignment scoring process is exhaustive within the area scanned, however, the area is limited in two ways: 1) we have already narrowed the search down by choosing a starting point that achieves at least the minimum allowed score, and 2) the extension process will only continue until while the score remains above the selected drop off level. Apart from this truncation, the process is almost identical to the Smith-Waterman method [3]: matches are rewarded by increasing an alignment's score, while mismatch and gaps are penalized by decreasing its score.

A grid can be used to represent all possible alignments by assigning left-to-right movement as traversal of the database, and top-to-bottom movement as traversal of the query sequence. Movement along the diagonal corresponds to an alignment with few gaps, and will therefore tend to generate higher scores. Movement in one direction, but not the other, corresponds to a gap and causes a reduction score. The BLAST approach is to terminate the extension process if the alignment score falls too far below the current local maximum. This translates into focusing the search on the area near the diagonal, where higher scores tend to occur.

The scores for each cell V_{ij} can be computed recursively, using the scores of its neighbors to the top, left, and diagonal. Using σ_{ij} to represent the reward/penalty of a sequence match/mismatch between i^{th} and j^{th} nucleotides from the query and database respectively, α to represent the penalty

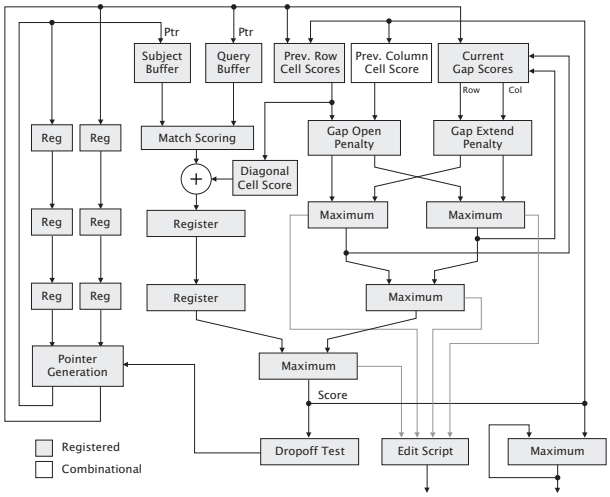

Fig. 3. Gapped alignment machine

of opening a gap, β as the penalty for extending a gap, and T_{ij} and L_{ij} as the gap scores from above and to the left, the score for cell (i,j) can be computed as:

$$V_{ij} = max(L_{ij}, T_{ij}, V_{i,j-1} + \sigma(D_j)) \quad (1)$$

$$L_{ij} = max(V_{i,j-1} - \alpha, L_{i,j-1} - \beta) \quad (2)$$

$$T_{ij} = max(V_{i-1,j} - \alpha, T_{i-1,j} - \beta) \quad (3)$$

This computation can be accomplished in a streaming manner if the dependencies are observed. For example, we may compute all scores along the first row, providing we address them in the order $j = 1...j_{max}$. We may then proceed to the next row, having just computed the prerequisites for is scoring. In fact, we may begin processing the first element in row 2 immediately after scoring the first element in row 1. In this manner, we may propagate a counter-diagonal wavefront of computation diagonally through the scoring grid.

In order to implement the maximum test, as described in (1)-(3), a pipelined tree of max functions was used. This allows us to achieve a high clock rate at the cost of a pipeline latency, which forces us to wait 4 cycles before starting the computation of the next element's score. Fortunately, we may place two jobs in the pipeline after the first has completed: one for the next cell to the right, and one for the cell below. On the third pass through the pipe we may process 3 jobs. By the 4th pass, the pipeline is full. The result of this arrangement is the simultaneous processing of 4 scoring grid rows. The drop-off test is conducted on the 4th row; when the score for this row falls D points below the overall maximum of the scoring grid, the pipeline is flushed and we begin scoring the next set of 4 rows. Similarly, if the first score in a row falls below this drop-off level, then scoring of the next set of rows will begin 1 column to the right. As the rows are scored an edit script is generated that indicates the outcomes of the maximum tests from (1)-(3). The edit script is then used by software to determine the structure of the high scoring alignment.

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 value selected of the drop-off value, which effects how frequently the pipeline is flushed. Because the software gapped alignment stage works on a list of w-mers that is culled after each w-mer is extended, the parallel implementation typically does not achieve peak performance. If we process K w-mers in parallel, there is a non-zero probability that we will be extending one or more w-mers that would have been culled, had we processed the list sequentially. The results of these extensions are discarded. We are in the process of analyzing this behavior and modifying NCBI BLAST such that we may process the list in an optimal order.

4. CONCLUSION

We have described a co-processor 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, and can achieve about 8 billion cell updates per second. This translates to about 0.5 million full extensions per second, but this is highly dependent upon the selected drop-off value, and the characteristics of the data set. Future work will involve balancing the performance of the stages, tuning performance to individual data sets, supporting additional stride and word widths, supporting protein sequence alignment, and using external memory to overcome device limitations.

5. REFERENCES

[1] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, pp. 403–410, 1990.

[2] NCBI, "Basic local alignment search tool," http://blast.ncbi.nlm.nih.gov/blast.

[3] T. Smith and M. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, pp. 195–197, 19981.