Accelerating NCBI BLAST

FPGA Supercomputing Coming of Age

Henrik Abelsson, Göran Sandberg and **Stefan Möhl**,
Mitrionics AB

ABSTRACT: *Using field programmable gate arrays (FPGAs) as coprocessors has long been a promising solution for accelerating software algorithms. With the development of an accelerated version of NCBI's BLAST application, FPGA supercomputing takes the step from proof-of-concept programs to the acceleration of full-blown production applications. In response to expressed interest from users of the NCBI BLAST application for comparing gene and protein sequences, Mitrionics has ported parts of the BLASTN algorithm to run on the Mitrion Virtual Processor in an FPGA. The work includes integration of the accelerated code into the original NCBI BLAST application, giving users access to their familiar tool, but with 10x – 20x performance improvements on gene searches. The Mitrion-C source code for the accelerated BLASTN algorithm, as well as the source code for the NCBI BLAST application, have been published by Mitrionics as Open Source as part of the Mitrion-C Open Bio Project. The purpose of the project is to provide a framework for the creation of Open Source accelerated versions of major bioinformatics applications. Presently, work is being done to accelerate the BLASTP algorithm for protein comparisons. Depending on the priorities of the developers' community, other algorithms considered are Translated BLAST, Smith-Waterman and Hidden Markov Models.*

KEYWORDS: FPGA, BLAST, bioinformatics, software acceleration, Mitrion

1. Introduction

The National Center for Biotechnology Information (NCBI) Basic Local Alignment Search Tool (BLAST) [1] is one of the most widely used bioinformatics applications used to compare gene or protein sequences to sequence databases.

As the explosive growth of genomic data continues to outpace the performance increases for traditional CPU-based server technology, there is an expressed interest to develop alternative solutions to accelerate these searches. The use of Field Programmable Gate Arrays (FPGAs) as accelerators for this purpose has a number of significant advantages:

- First of all, the flexibility of the FPGA architecture allows highly efficient operations on the narrow bit-width data used to represent gene and protein sequences, without the overhead of the fixed processor register sizes used by traditional processors.
- FPGAs are significantly less power-hungry than high performance CPUs, typically using less than ¼ of the power. This, combined with an attained algorithm acceleration of 10x – 20x, reduces the required power to a small fraction.
- In contrast to other acceleration technologies, FPGA-accelerated systems have been around for several years, and are readily available off the shelf from major system vendors.

Beyond proof-of-concept

In order for FPGA acceleration technologies to gain acceptance it must be proven that the technology has passed the point were it is only meaningful for experimental and proof-of-concept use. For this reason, the critical parts of the NCBI BLASTN application were accelerated by porting them to run on the Mitrion Virtual Processor (MVP) and then fully integrated with the standard distribution of NCBI BLASTN. This gives acceleration of a full production quality application, rather than just an indication of what could be possible with the technology. For the user, it is simply their familiar application, only much faster.

About the Mitrion Platform

The Mitrion Platform enables software developers to target FPGA-based computers without needing any of the hardware design skills required by traditional FPGA development.

The core of the Mitrion Platform is the Mitrion Virtual Processor, a configurable processor design for a fine-grain massively parallel, soft-core processor. You program this processor using standard software development methods. In order to make efficient use of the resources available on the FPGA, the processor is then automatically adapted to the program it is going to run. The result is a configuration file for the FPGA, which will turn it into a co-processor running your software algorithm 10-30 times faster than the latest generation high-performance CPUs, like AMD Opteron or Intel Itanium.

To access the massive parallelism afforded and required by the Mitrion Virtual Processor, a fully parallel programming language is needed. It is simply not sufficient to rely on vector parallel extensions or parallel instructions. The Mitrion-C programming language is designed to make it easy for programmers to write parallel software that makes the best use of the Mitrion Virtual Processor. Mitrion-C has a C-family syntax, but the focus is on describing data dependencies rather than order of execution. The Mitrion-C compiler is able to extract all the parallelism of the algorithm being developed.

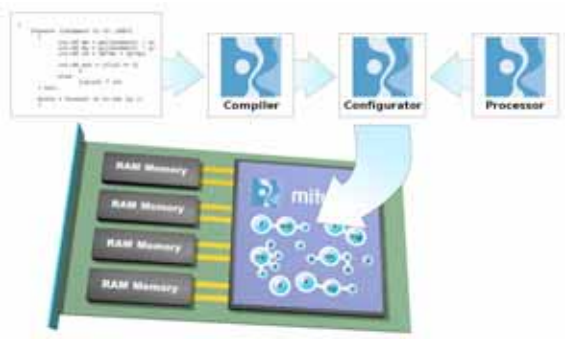

Fig. 1, The Mitrion Platform

The Mitrion Software Development Kit (SDK) is used to develop applications for the Mitrion Virtual Processor. The Mitrion SDK consists of:

- A Mitrion-C compiler for the Mitrion Virtual Processor,
- A graphical simulator and debugger that allows Mitrion-C applications to be tested and evaluated without the need to run them in actual FPGA hardware
- A processor configuration unit, which adapts a Mitrion Virtual Processor to the compiled Mitrion-C code.

2. Mitrion-Accelerated NCBI BLAST

The application that has been accelerated is the BLASTN nucleotide search algorithm of the NCBI `blastall` program. All other functionality of the `blastall` program remains untouched, giving the user an identical interface as well as input and output options. The `blastall` program runs on the server, communicating with and controlling the Mitrion Virtual Processor running on the attached FPGA accelerator hardware.

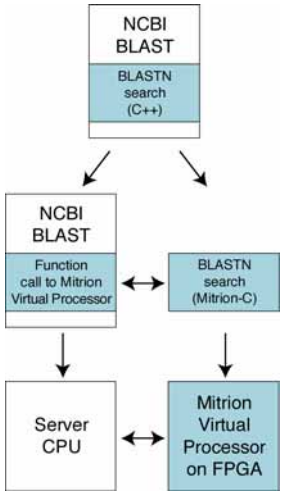

Fig. 2, Overview – Mitrion-accelerated BLAST

The FPGA Accelerator Hardware

Typically, FPGA accelerator hardware consists of a large FPGA connected to high-speed bus interface for communication with the host system, and a to number of independent RAM memory banks used to store the data the FPGA is operating on.

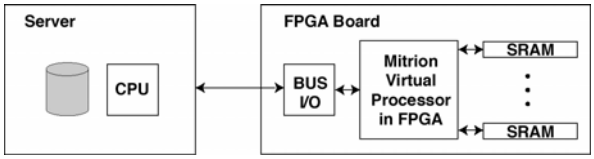

Fig. 3, A typical FPGA accelerator set-up.

The fact that the attached memory banks can be accessed independently and simultaneously gives the FPGA an important performance benefit with a sustained memory bandwidth of 10-20 Gbytes/s. Communication with the host system typically has a bandwidth of 3-6 Gbytes/s. Even more memory bandwidth can be obtained from the large number of internal memory banks in the FPGA. With the current Xilinx Virtex-4 FPGA generation, the bandwidth to the internal memory banks is about 0.5TB/s. The draw-back is that this very high bandwidth memory is limited to around 750 kB of storage in total.

Accelerating the BLASTN Algorithm

The BLAST algorithm uses three stages of processing to filter the contents of the database being searched against the query. Each processing stage drops the majority of the data, passing on only the possibly matching parts of the database to the subsequent stages. In this sense, the BLAST algorithm can be seen as a sequence of filters. The output from the final stage are the BLAST alignments.

According to research done at Washington University [2], the bulk of the data is being processed by the first two stages, and these are expected to require most of the compute time. We have therefore focused our effort on the acceleration of the first two stages and left the third stage to run on traditional CPUs. See figure 4.

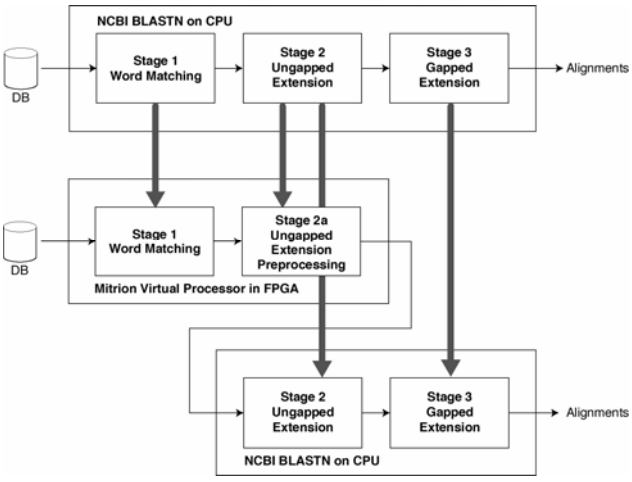

Fig. 4, Three-stage deployment of BLAST

The FPGA is loaded with a Mitrion Virtual Processor programmed and adapted for the first two stages of the NCBI BLASTN algorithm, and the nucleotide database as it is streamed through. Resulting matches are fed back to the original BLASTN program, which performs the third processing stage and presents the results.

Algorithm Details

In stage one, the BLASTN algorithm looks for exact patterns of a certain length (11 bases by default). It is divided into two sub-stages (see Figure 4). First a fast probabilistic filter returns a list of probable exact matches. Then any false positives are filtered out.

The first sub-stage is implemented using bloom filters, a space-efficient probabilistic data structure that is used to test whether an element in the database is a member of the set of query words. Each bloom filter consists of a number of hash functions that check whether a specific bit is set or not in memory. One can think of bloom filters as a hash-table that does no collision detection. This filter may produce false positives, but it will not produce any false negatives.

```
bloomfilter(wmer_list) {
 /* Create bloom filter buffers */
 ht0_1 = _memcreate(mem bits:1[524288] ht0_back);
 ht1_1 = _memcreate(mem bits:1[524288] ht1_back);
 ht2_1 = _memcreate(mem bits:1[524288] ht2_back);
 ht3_1 = _memcreate(mem bits:1[524288] ht3_back);

  /* Go through all wmers from the input shifter*/
  out = foreach(codeword in wmer_list by address) {
    /* Find addresses in the bloom buffers */

    addr0 = h3hash_22x19_0(codeword);
    addr1 = h3hash_22x19_1(codeword);
    addr2 = h3hash_22x19_2(codeword);
    addr3 = h3hash_22x19_3(codeword);

    t0 = _memread(ht0_1,addr0);
    t1 = _memread(ht1_1,addr1);
    t2 = _memread(ht2_1,addr2);
    t3 = _memread(ht3_1,addr3);
    uint:1 ans = ((t0 & t1) & (t2 & t3));
    out = _tup(codeword,address);
    /* If we found a match, pass it on to the next stage */
    res = if (ans == 1 ) (. out .) else (.-.);
 } res;

 ht0_back = ht0_1;
 ht1_back = ht1_1;
 ht2_back = ht2_1;
 ht3_back = ht3_1;

 result = reshape(out,(..));
} result;
```

Listing 1, Bloom filter stage in Mitrion C

Due to the fact that FPGAs have multiple distributed internal memories that can be accessed simultaneously by the Mitrion Virtual Processor, it is possible to run a number of bloom filters in parallel, depending on the query size. In this way, we take advantage of the very high bandwidth to the internal memory banks of the FPGA. On the SGI® RASC™ RC100, where the algorithm has first been implemented, the Mitrion Virtual

Processor provides a sustained lookup rate of 16 memory loads per clock cycle for a 100k query and 64 memory loads per cycle for a 10k query. The throughput of the first stage is 400 megabases per second for a 100k query and 1.6 gigabases per second for a 10k query.

The second sub-stage used to filter out the false positives generated by the bloom filters uses a traditional hash table look-up. It also looks up the position of the word within the query. The hash-table look-up is usually the first stage of the BLAST algorithm. However, through the use of the bloom filters, most hits have already been filtered away at this stage. Thus, the performance requirement of the hash-table look-up are significantly reduced. The hash table can sustain one lookup per clock cycle to one of the SRAM banks directly attached to the FPGA, while the database is simultaneously being read from another local SRAM memory bank.

The matches that pass the hash table look-up are passed on to stage 2.

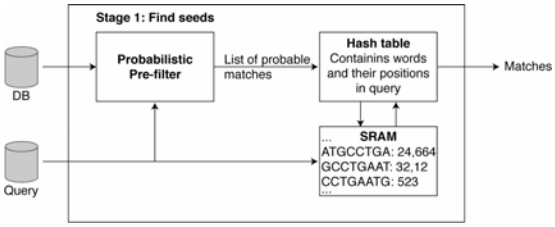

Fig. 5, Stage 1 processing

The first part of stage two (see Figure 6 and Listing 2) extends the seed within a fixed window, returning matches that pass the threshold score and discarding the rest. It is implemented as an unrolled systolic array that is easily expressed as a Mitrion-C loop, which finds the best start position, stop position and score of two 64 character words in a single clock cycle.

```
ungapped_extension(hit_list) {
  out = foreach(elem in hit_list) {
   (db,query,qidx,dbidx) = _untup(elem);
   (res_score,res_beginning,res_end) = for(de,qe in db,query) {
     s = if (de == qe) MATCH_REWARD else MISMATCH_PENALTY;
     cur_score = score + s;

     (max_score,max_beginning,max_end,beginning,score) =
     if (cur_score > 0) {
           (n_max_score,
            n_max_beginning,
            n_max_end) = if (cur_score > max_score && i >= WMER_END ) {
              /* If we have a positive score and we are
                 after the wmer ended, set a new best end
                 position. */
              n_max_score = cur_score;
              n_max_beginning = beginning;
              n_max_end = i;
            } (n_max_score,n_max_beginning,n_max_end)
            else {
              /* keep the old best end position */
            } (max_score,max_beginning,max_end);
            } (n_max_score,
           n_max_beginning,
           n_max_end,
           beginning,
           cur_score)
     else {
        (n_beginning,n_score) = if (i <= WMER_START) {
           /* score has dropped below zero, so set a new
              starting position at the current position
              and hope it yields better results.*/
           n_beginning = i;
           n_score = 0;
        } (n_beginning,n_score)
        else {
          /* we have passed the start of the wmer, so we
             can't change the beginning any more */
        } (beginning,cur_score);
      }
(max_score,max_beginning,max_end,n_beginning,n_score);
      i = i + 1;
    } (max_score,max_beginning,max_end);

     /* Pass the result on to the next stage if the resulting
        score is larger than the threshold */
     r = if (res_score > ungapped_threshold) {
       e = _tup(res_score,res_beginning,res_end,qidx,dbidx);
       r = (. e .) ;
     } r else {
       r = (.-.);
    } r;
  } r;
  final = reshape(out,(..));
} final;
```

Listing 2, Un-gapped extension implemented as an unrolled systolic array.

All alignments that pass the threshold score are passed to a filter that discards matches that are so close to each other that they could not be extended without being joined. In the standard NCBI implementation, this is done as part of stage 1, but for the Mitrion implementation, we have found that the performance is improved by doing this filtering after the un-gapped extension. This is due to the fact that the MVP implementation of the un-gapped extension is significantly faster than the implementation on traditional CPUs.

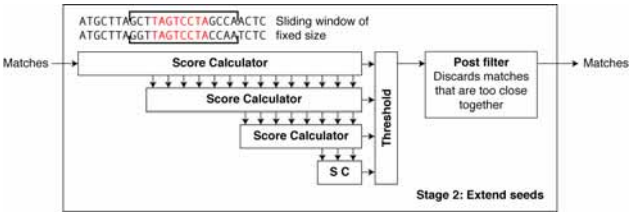

Fig. 6, Stage 2 processing

Matches that pass the final filer are returned from the MVP to the host. Here, the un-accelerated parts of NCBI BLAST may extend them even further through gapped extension. The final alignments and scores are then generated and the results formatted and printed.

3. Results

Presently the Mitrion-accelerated NCBI BLAST application runs on SGI® RASC™ RC100 hardware, which has been used to develop and evaluate the application. The acceleration that has been obtained with

the Mitrion Virtual Processor on this system provides an order of magnitude leap in performance over a traditional CPU-only solution.

We have found that the speedup varies with the queries used as well as the databases being searched. An important factor is the impact of Amdahl's Law from the amount of data being forwarded to the un-accelerated gapped extension stage. Though it is usually very low, it becomes significant in some queries. Speedups of 10x – 20x are observed on the current implementation.

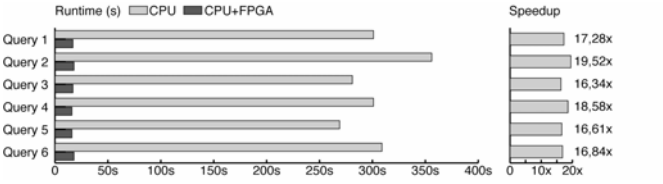

Fig. 7, Speedup searching the EST Mouse database.

4. Future Directions

The Mitrion-accelerated NCBI BLAST application has been released as open source under the GNU general public license as part of the Mitrion-C Open Bio Project at SourceForge (http://mitc-openbio.sourceforge.net). The purpose of the Mitrion-C Open Bio Project is to support the public in accelerating key bioinformatics applications by porting them to the Mitrion Virtual Processor.

For the BLASTN algorithm, work is being done to further optimize the use of the available hardware. A possible future enhancement would be to use the Mitrion Virtual Processor to accelerate the gapped extension parts of the algorithm, which typically consumes half of the remaining execution time in the accelerated BLAST. Also, as new CRAY FPGA computing platforms become available, the porting of the accelerated application to these is a priority.

Within the Mitrion-C Open Bio Project, progress is being made to enhance the usefulness of the accelerated version of NCBI BLAST by accelerating the BLASTP protein search algorithm. Possible future developments being discussed are the translated BLAST versions, the Smith-Waterman algorithm and Hidden Markov Models.

5. Conclusion

The use of the Mitrion Virtual Processor to run software in an FPGA makes it possible to accelerate real applications, without resorting to circuit design such as done by earlier proof-of-concept accelerated algorithms.

With Mitrion-accelerated NCBI BLAST, users can continue using their familiar BLAST application, while at the same time getting their BLASTN searches completed 10 to 20 times faster.

Since the source code is released as open source, users who own Mitrion Virtual Processor licenses can run the code for free, as well as tweak and adapt the code for their own purposes.

Acknowledgments

The authors would like to thank colleagues and SGI's RASC team for their valuable input.

About the Authors

Henrik Abelsson is senior software engineer at Mitrionics AB and the lead developer of the accelerated BLAST, E-Mail: henrik.abelsson@mitrionics.com, Göran Sandberg is VP of Product Management, Mitrionics AB, E-Mail: goran.sandberg@mitrionics.com, Stefan Möhl is CTO and co-founder of Mitrionics AB and, along with Pontus Borg, inventor of the Mitrion Virtual Processor architecture. E-Mail: stefan.mohl@mitrionics.com. All can be reached at Mitrionics AB, Ideon Science Park, SE-223 70 Lund, Sweden.

References

1. Altschul, S. F., Gish, W., Miller, W., Myers, E. W. & Lipman, D. J. Basic local alignment search tool. *J Mol Biol* **215**, 403-410 (1990).

2. Krishnamurthy, P. et al. Biosequence Similarity Search on the Mercury System. *Proc. of the IEEE 15th International Conference on Application-specific Systems, Architectures and Processors* 365-375 (2004).