Performance Evaluation of FPGA-Based Biological Applications

Olaf Storaasli and **Weikuan Yu,** ORNL
Dave Strenski, Cray, **Jim Maltby**, Mitrionics

Abstract

On the forefront of recent HPC innovations are Field Programmable Gate Arrays (FPGA), which promise to accelerate calculations by one or more orders of magnitude. The performance of two Cray XD1 systems with Virtex-II Pro 50 and Virtex-4 LX160 FPGAs, were evaluated using a computational biological human genome comparisons program. This paper describes scalable, parallel, FPGA-accelerated results for the FASTA application ssearch34, using the Smith-Waterman algorithm for DNA, RNA and protein sequencing contained in the OpenFPGA benchmark suite. Results indicate typical Cray XD1 FPGA speedups of 50x (Virtex-II Pro 50) and 100x (Virtex-4 LX160) compared to a 2.2 GHz Opteron. Similar speedups are expected for the DRC RPU110-L200 modules (Virtex-4 LX200), which fit in an Opteron socket, and selected by Cray for its XT Supercomputers. The FPGA programming challenges, human genome benchmarking, and data verification of results, are discussed.

Keywords: FPGA, reconfigurable, DNA, RNA, Smith-Waterman, Cray, FASTA, XD1, Virtex, OpenFPGA

1 Introduction

Remarkable innovations in computer technology[1] are fulfilling NASA long-term projections[2] for faster science and engineering computations. One forefront innovation is harnessing Field Programmable Gate Arrays (FPGA) to accelerate High-Performance Computing (HPC) applications by one or more orders of magnitude over traditional microprocessors. A completely new form of programmable logic, the FPGA, was invented in 1984 by Ross Freeman co-founder of Xilinx Corporation. FPGA architectures are extremely flexible and are dominated by interconnections to thousands of embedded functions (**Fig. 1**. left) like adders, multipliers, memory, and logic slices (**Fig. 1**. right) for digital signal processing and high-speed communication (Hypertransport, PCI-EXPRESS). Unlike programming conventional "fixed" microprocessors, FPGA hardware gates are themselves reconfigurable (changeable "on the fly") by users in the "field" (thus, field programmable). Some FPGAs may be partially reconfigured, even while other portions of the same FPGA are still running.

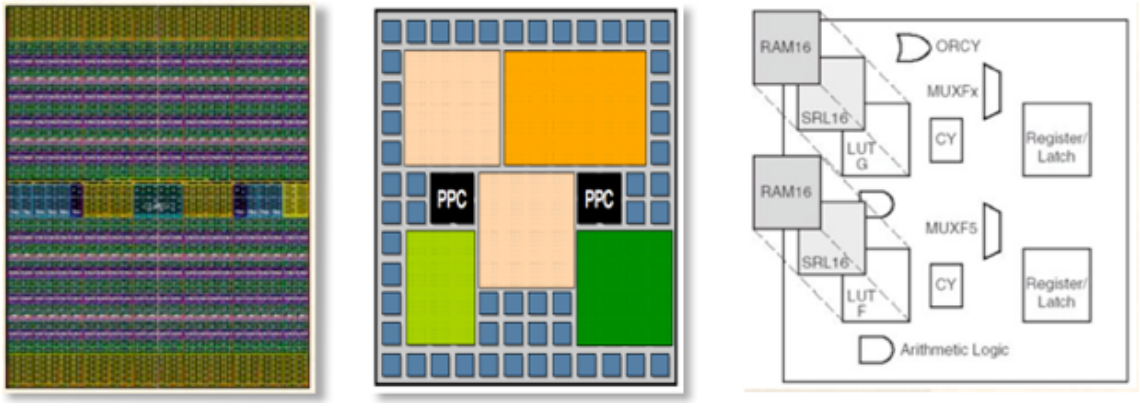

Figure 1. Virtex-4 FPGA (left), PPC processors, memory, I/O (center) and logic slice (right)

The Virtex-4 is available with one or more PowerPC (PPC) processors "on-board" the chip (Fig 1. center). The rapidly growing (15-20%/year) $2B FPGA market (focused on high-volume communications) is dominated by Xilinx and Altera. However, smaller, aerospace and High-Performance Embedded Computing (HPEC) applications are rapidly expanding FPGA use. HPC sales (< 1%) is a spinoff rather than a driver for future FPGA designs. Recently, however, FPGA designers have begun to consider HPC requirements in their next generation designs.

FPGA Characteristics: FPGA layout is extremely regular compared to microprocessors, simplifying fabrication, and allowing FPGAs to be among the first to reduce feature sizes (90nm => 65nm => 45nm). For space and flight use, this regularity and redundant algorithms, limits radiation damage (i.e. NASA Mars Rovers). At each clock cycle, FPGA algorithms (when coded to maximize the number of parallel operations) use nearly 100% of their silicon, compared to less efficient microprocessors which use less than 2% of their silicon (while drawing 10x FPGA power to perform only one or two operations).

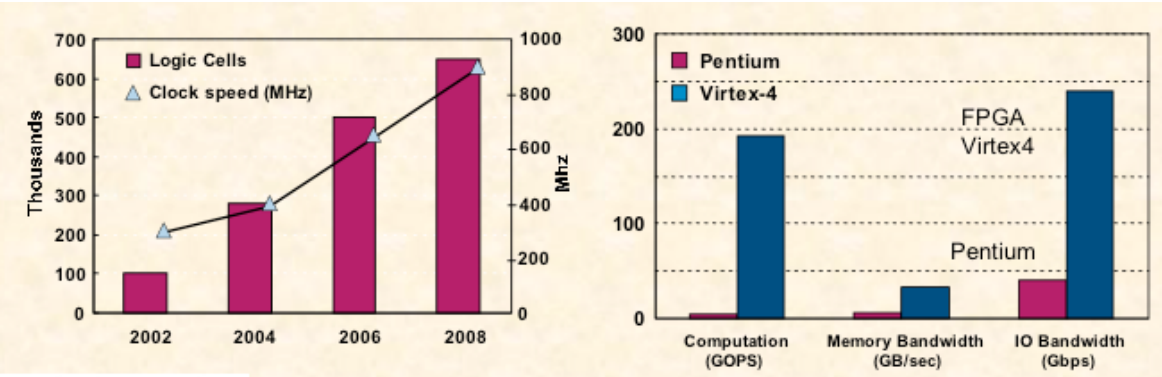

Figure 2. FPGA Characteristics: Logic Cell and clock speed growth, Computation and bandwidth speeds.

Figure 2 shows several key FPGA characteristics. FPGAs, unlike microprocessors, continue to advance at Moore's Law rate and still have far to go before reaching logic cell and speed limits (**Fig. 2**., left). FPGA clock speeds (frequently at 100-200 MHz for many applications) have far to go before facing heating issues that drove microprocessors to multi-core chips running at reduced clock speeds. When FPGA applications are programmed with a high degree of parallelism, their computation speed far exceeds that of microprocessors (**Fig. 2**., right). As expected for high-speed communications devices, the memory and IO bandwidths also significantly exceed those of microprocessors (**Fig. 2**).

FPGA Coding: As FPGAs were developed by logic designers, they are traditionally programmed using circuit design languages such as VHDL and Verilog. These languages require the knowledge and training of a logic designer, take months to learn and far longer to code efficiently. Even once this skill is acquired, VHDL or Verilog coding is extremely arduous, taking months to develop early prototypes and often years to perfect and optimize. FPGA code development, unlike HPC compilers, is greatly slowed by the additional lengthy steps required to synthesize, place and route the circuit.

Once the time is taken to code specific applications in VHDL, their FPGA performance is hard to beat. In particular, applications using basic integer or logic operations (compare, add, multiply) such as DNA sequence comparisons, cryptography or chess logic, run extremely fast on FPGAs. As floating point and double-precision applications rapidly exhausted the number of slices available on early FPGAs, they were often avoided for high-precision calculations. However, this situation has changed for current Xilinx FPGAs (Virtex-4 and Virtex-5) which have sufficient logic to fit about 80 64-bit multiply units[3].

The capability of early FPGAs was well suited for special-purpose High-Performance Embeded Computing (HPEC). However, their use for general-purpose HPC was initially restricted to first-generation, low-end reconfigurable supercomputers. (i.e. Starbridge Systems, SRC, Cray XD1). The lack of high-speed IO and infrastructure (compilers, libraries) to support general-purpose supercomputer applications, including legacy codes are typical of this early generation. However, this situation is rapidly changing with the latest generation of reconfigurable supercomputers and the FPGAs they use. DRC Computer, Xtreme Data and Xilinx in collaboration with Intel now provide modules containing the latest FPGAs which fit in the same socket used by microprocessors and using the same high-speed communications link. Cray selected DRC **Fig. 3**, to accelerate its XT line of supercomputers.

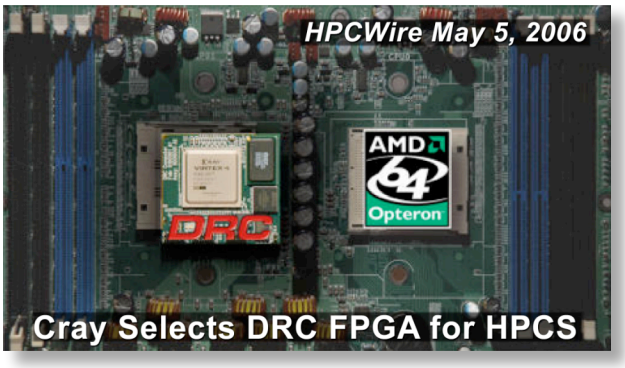

Figure 3. DRC module with Xilinx Virtex 4 FPGA (left) fits in AMD Opteron socket.

FPGA HPC Competition (alternatives): Accelerating HPC applications is so critical that many alternatives have entered the marketplace. Even though many legacy physics-based codes are written in sequential Fortran over 30 years ago, they have remarkably survived several HPC generations: vector (via compilers), parallel (via MPI, OpenMP) and now the first stages of multi-core microprocessors. Some surmise they may suffer severe performance degradation or even require significant rewrites to fully exploit 8 or more cores/chip. Major chip vendors (Intel and AMD) have vigorous efforts to accommodate accelerators, with their primary focus on FPGAs as a way to regain performance. As multi-core microprocessors face looming power, cooling, size and IO challenges, FPGAs appear more attractive.

Three other accelerator options are available to HPC architects: Cell (IBM), Array (ClearSpeed) and Graphical Processors (GPUs). Like FPGAs, Cell and Graphical processors have vast commercial markets (video games and graphics) driving down costs, promoting competition and stimulating advances making them increasingly attractive to HPC. However, array processors are custom devices which need to be amortized over relatively few users. GPUs require tremendous power/cooling and have complex programming and data precision issues to solve before they can enter the HPC market. Coding the 8+1 Cell processors is likely to be considerably more difficult than programming FPGAs in VHDL or Verilog, which already has a large user base. As FPGA hardware advances, FPGA software/tools simplify their use for HPC including MitrionC, Viva, DSPlogic, ImpulseC, Celoxica, Aldec, and Xilinx's CHiMPS, specifically aimed at the HPC market. Two of the authors are testing CHiMPS for HPC applications.

2 FASTA DNA and Protein Search/Alignment

The FASTA code suite[4], available from *fasta.bioch.virginia.edu*, contains programs for protein:protein, DNA:DNA, protein:translated DNA (with frameshifts), and ordered or unordered peptide searches using a unique search heuristic and an implementation of the optimal Smith-Waterman algorithm[5-7]. FASTA's major focus is to accurately calculate similarity statistics for biologists to determine whether alignments are random or homotopic. The FASTA input file format is widely used for other sequence database

search tools (i.e. BLAST[8]) and sequence alignment programs. FASTA's speed is attributed to the heuristic method of observing the pattern of word hits, word-to-word matches of a given length and marking potential matches prior to the time-consuming Smith-Waterman search. The selected word size controls the sensitivity and speed of the program. The word hits returned are examined for segments, containing clusters of nearby hits, which are investigated for a possible match. This is accomplished in four steps described in detail[9].

1. Identify regions of highest density in each sequence comparison
2. Re-score using PAM scoring matrix, keeping top scoring segments.
3. Apply joining threshold to eliminate segments unlikely to be part of the alignment containing the highest score segment.
4. Use dynamic programming to optimize the alignment in a narrow band encompassing the top scoring segments.

FASTA relies on ssearch34 routines, which in turn use the Smith-Waterman algorithm[5-16], the essentials of which are summarized next.

3 Smith-Waterman

Similarities between known database and query sequences are frequently used to detect functional similarities, whether for RNA, DNA or proteins. The Smith-Waterman dynamic programming algorithm is used in bioinformatics for sequence matching to detect such similarities by breaking down the sequence alignment problem into a set of simpler sub-problems. A table (e.g. **Fig. 4**) is generated with the query sequence characters written across the top and database sequence characters written down its side. The table is then filled with score values that reflect the quality of an alignment at a specific offset. The highest score in the table indicates the best potential to solve these sub-problems in parallel.

The score in a given table cell depends on the quality of the match between the query and database characters found at the head of that cell's row and column. It also depends on the adjoining scores above, above left, and directly left. The overall problem of calculating the total alignment is broken down into the simpler sub-problem of simultaneously calculating the many table score values in parallel. Once scores for one row or column have begun, calculations for adjoining rows or columns may begin in parallel.

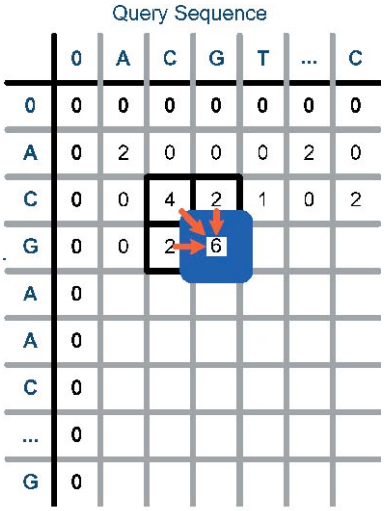

Figure 4 Example of Smith-Waterman Algorithm

Figure 4 shows a query sequence "ACGT…C" and a larger database sequence "ACGAAC…G". The first row and column of the Smith Waterman table are initialized to zero. The scores are then calculated starting in the upper left corner and proceeding outward. **Figure 4** illustrates how a score of '6' is calculated from its adjacent neighbor scores above and to the left, as well as from the fitness of the match between the 'G' query character and the 'G' database character found at the head of its row and column.

Although the mechanics of the cell calculations is interesting, of key importance is that the algorithm can be broken into many smaller sub-problems and solved in parallel[4]**.** Further details on implementing the Smith-Waterman algorithm on FPGAs is available[5-16].

3.1 Algorithm Acceleration

An algorithm's suitability for FPGA acceleration may be assessed prior to writing new code by using the following five criteria. These criteria illustrate that the Smith-Waterman algorithm is an excellent candidate for FPGA acceleration.

3.1.1 Code Profile

Often only a small algorithm segment can or need be placed onto a reconfigurable device to achieve significant acceleration. Applications that spend a large percentage of their time repeating this segment (or "kernel") benefit the most from acceleration. A profile of FASTA's ssearch34 application, **Figure 5**, shows 98.6% of its time is spent in the FLOCAL_ALIGN function. This function, an optimized Smith-Waterman algorithm that calculates the maximum alignment score for two sequences, is an excellent candidate for acceleration, provided it can be efficiently offloaded to FPGAs. The Smith-Waterman's inherent parallelism is an advantage when porting to an FPGA. Parallelism not only increases the potential FPGA performance, but it also allows the design to scale well. It may be difficult to initially assess how many Smith-Waterman score values can be calculated in parallel, but the design can be scaled to fit as many as possible.

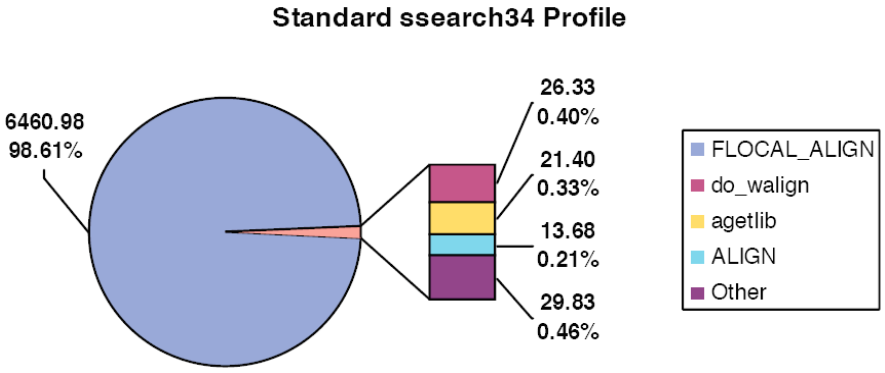

Figure 5 Profile of ssearch34 Application

3.1.2 Parallelism

Due to their flexible and generic nature, FPGA logic resources are often much slower than the dedicated logic used to construct modern microprocessors. To effectively compete with faster, serial microprocessors, reconfigurable logic must be able to perform many operations in parallel. As discussed above, the Smith-Waterman algorithm contains a great deal of parallelism. Many of the alignment scores

can be calculated in parallel. Additionally, some of the computations required to calculate single alignment scores can also be performed in parallel.

3.1.3 Instruction Efficiency

Modern 64-bit microprocessors support powerful, general-purpose instruction sets. However, for application calculations (e.g. compares) that are simple, using 64-bit microprocessors is extreme overkill and wasteful. FPGAs use only the minimum logic required for given calculations, freeing up other resources to exploit parallelism. The basic data types of the Smith-Waterman algorithm are sequences of characters. Each character can be represented by as little as two bits drastically reducing the logic required to perform each calculation.

3.1.4 Bandwidth and Data Localization

The performance of many algorithms is limited by bandwidth rather than computational power. Such bandwidth limitations can be between the microprocessor and the reconfigurable device, or between the processing device and its own memory. The bandwidth between the microprocessor and reconfigurable device tends to be fixed and likely to be the most efficient when large amounts of data are being transferred. Memory bandwidth tends to increase the closer the memory is to the microprocessor. For example, the internal cache memory of modern microprocessors is substantially faster than the external cache memory, and faster again than external SDRAM. The same is also true for the memory subsystems of reconfigurable devices. Algorithms that are currently limited by a microprocessor's SDRAM bandwidth can be similarly limited on a reconfigurable device.

Next, to take advantage of parallelism, we determine if there is sufficient bandwidth between the microprocessor and reconfigurable device, as well as between the reconfigurable device and its memory resources. To calculate the maximum alignment score, the microprocessor sends the query sequence, the database sequence and several scoring parameters to the FPGA. The number of scores the FPGAs must calculate to find the maximum is the length of the query sequence multiplied by the length of the database sequence. For every database sequence character sent to it, the FPGA must calculate an entire row of scores. Calculating each score requires many computations for every character sent to the FPGAmaking it unlikely that the bandwidth available to send the sequences to the FPGA will be a bottleneck. Likewise, since the only data returned from the FPGA is the maximum score, the bandwidth from the FPGA to the microprocessor will likely not be a limitation either. Thus, the only limitation is how quickly the FPGA can calculate scores.

3.1.5 Sufficient Memory:

The above analysis is accurate only if the FPGA can calculate and store the entire table of scores in one pass. This seems unlikely as it would require the scores in an entire row of the table to be calculated and stored in parallel. This would severely limit the size of the query sequence. For the FPGA to calculate the table of scores in sections, it must hold intermediate data in local memory. To break the table of scores into vertical segments, it must store the last column of a segment to use to calculate the first column of the next segment. Since only one column must be stored, the memory bandwidth required will not likely be the limiting factor. However, the size of the memory available will restrict the maximum length of the query and database sequences.

4 Accelerator Design

The Smith-Waterman algorithm was implemented on Xilinx Virtex-II Pro 50 and Virtex-4 LX160 FPGAs on Cray XD1 systems as a linear systolic array of processing elements. The processing elements are chained in a pipeline as shown in **Figure 6**.

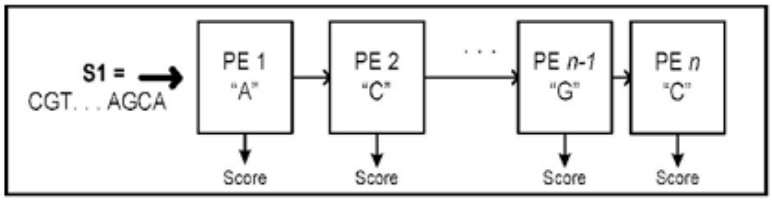

Figure 6. Smith-Waterman Pipeline

One query character is preloaded into each processing element. Each processing element then calculates a score in the column for that query character. The database string (S1) is shifted through the pipeline so that each database character can be compared to each query character. The resulting table of scores is filled out from top to bottom over time as shown in **Figure 7**.

Building the pipeline of processing elements comprises most of the accelerator design. However, additional logic is required to feed the processing elements, interface the array logic to the microprocessor, and to access the external QDR II SRAMs surrounding the FPGA. **Figure 8** shows the overall design including the Smith-Waterman Array block and additional control logic.

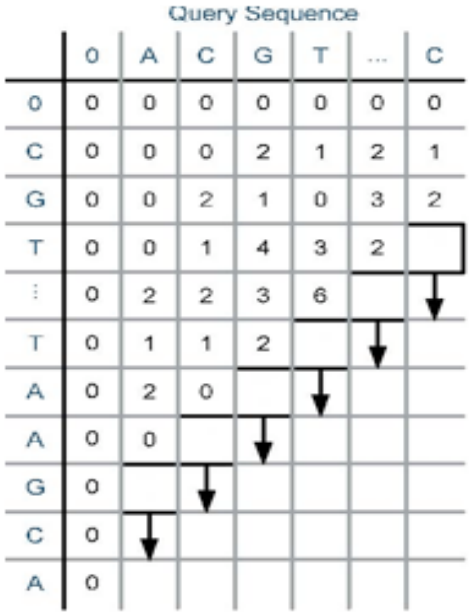

Figure 7. Smith-Waterman Score Calculation

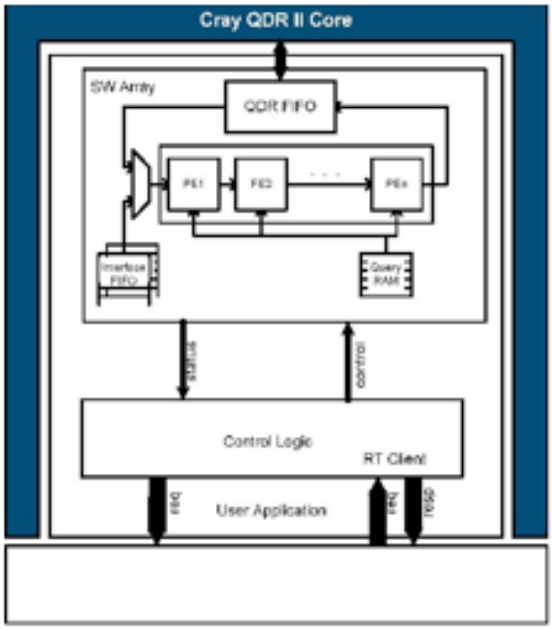

Figure 8. Overall Smith-Waterman Design

In addition to processing elements, the design uses the internal FPGA block RAM to store the complete sequence of query characters. It uses the external QDR II SRAM to store intermediate results generated when query sequences exceed the number of processing elements. The four external QDR II SRAMs are accessed via the Cray QDR II Core. Internal block RAM is also used as an interface FIFO to buffer part

of the incoming database sequence. The FIFO buffering allows the Opteron to write the database characters to the pipeline in efficient bursts rather than one character at a time.

The control logic block shown in **Figure 8** provides status and control registers for the Opteron, and writes the final scores back to the Opteron's local DRAM memory. The control logic block does this by interfacing with the Cray RT Core, which processes read and write requests to and from the Opteron. The status and control registers allow the microprocessor to set up the logic for a given alignment as well as detect any errors that may have occurred during its operation. When the alignment is complete, the maximum score generated is written back to the Opteron.

5 Results-OpenFPGA Benchmark (3 Cases):

FPGA speedups were obtained for the FASTA[4] application ssearch34, which is programmed to call a FPGA version of the Smith-Waterman algorithm[5]. Based on successful ssearch34 results on smaller data, the comprehensive 4GB human genome sequencing application comprising the three test cases of the OpenFPGA benchmark were attempted. These test cases and results are available for downloading and comparison both on OpenFPGA (openfpga.org) and ORNL (fpga.ftp.ornl.gov). Readers are encouraged to run comparative testing and communicate their performance results with the authors for potential posting on the ORNL site. The results of the three test cases follow.

5.1 Case 1: Micro-RNA (DNA Short Reference Sequences)

Case 1 required 3685 query sequences searching across all 24 human genome chromosomes. Using default options, a run was made with all the query sequences against the first chromosomes on a single Opteron to establish a baseline time. Unfortunately, this baseline took 3 days to complete. The same calculation, performed using the FPGA version, took only 7.4 hours for a speedup of 10x. The FPGA runs were then done in parallel using MPI (**Fig. 9)**, and showed excellent scaling, which was expected.

	1	2	3	4	5
CPU 2.2GHz	75	-	-	-	-
FPGA(s) .2GHz	7.39	3.75	2.48	1.91	1.56
FPGA Speedup vs 1 CPU	10.15	20.0	30.2	39.3	48.1

Figure 9. Cray XD1 hours to perform ssearch34 on chromosome one of the human genome (FPGA speedup in Red).

This was promising, but the authors observed that the output from the Opteron version was exceedingly large with slight differences in the output from the Opteron and the FPGA. Consulting with other OpenFPGA members, a new set of ssearch34 options were selected to minimize the amount of output generated from the program. Actually two sets of options were used, one to print out a reasonable amount of alignment sequences and a second with minimal output with only the scores from the searches.

Reasonable output: `-Q -H -f -10 -g -3 -d 10 -b 10 -s OpenFPGA.mat -E 0.0001`
Minimal output: `-Q -H -f -10 -g -3 -d 0 -b 10 -s OpenFPGA.mat -E 0.0001`

However, the Opteron code with either of these options, went into an infinite loop and did not complete, while the FPGA version performed well. The authors struggled to figure out why the Opteron version of the code was not working correctly, but decided to move on to the more interesting Case 2 due to the

lengthy run times (and paper deadline). Case 1 was specifically created, to have very short query sequences (approx. 20 characters), as some FPGA implementations are restricted to very short query sequences. It is believed that had the MPI Opteron code worked the performance results would be similar to those obtained for case 2 and 3 below.

5.2 Case 2-Bacillus_anthracis (DNA comparison)

Case 2 involves 18 DNA query sequence files named AE017024 through AE017041 and a large database file named AE016879. Each query sequence contains about 300 thousand characters and the database is over 5 million characters. The first limitation to overcome are the known restrictions of the FPGA version of the Smith-Waterman algorithm which limits the query sizes to 16k characters and the database size to 512k characters. To get around these limits and still have valid Opteron to FPGA comparisons, a program was written to split the input query and database files into smaller sequences. These sequences were then fed through both the Opteron and FPGA versions of the ssearch34 program. Multiple runs were performed using the maximum query size allowed, 16k characters, and a smaller 8k character sequence length for comparison. The options used for these runs were the two suggested by OpenFPGA.

Figure 10 shows speedup results on ORNL's Cray XD1 system, *Tiger*, with Virtex-II Pro 50 FPGAs for:
1. 8k query sizes with sequence alignment given in the output
2. 16k query sizes with sequence alignment given in the output
3. 8k query sizes with no sequence alignment given in the output
4. 16k query sizes with no sequence alignment given in the output.

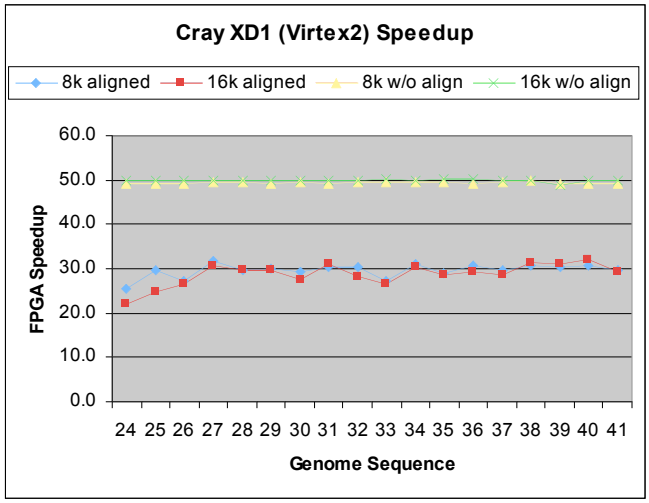

Figure 10. Virtex-II Pro 50 FPGA speedup for 8k and 16k sequence lengths (w, w/o alignment output)

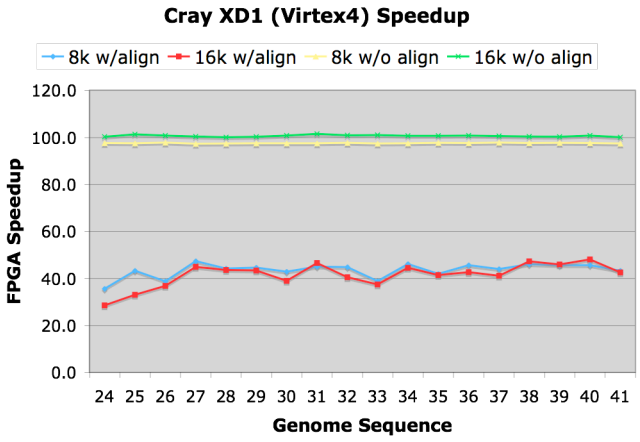

Figure 11. Virtex-4 LX160 speedup for 8k and 16k sequence lengths (w, w/o alignment output)

Figure 11 shows results for identical data on Cray's XD1 system, *Pacific,* with Virtex-4 LX160 FPGAs.

The first thing to notice in **Figs. 10 and 11** is that outputting the aligned sequences significantly reduces the performance (blue and red curves). This part of the code is performed on the Opteron and was not optimized, since it was such a small part of the execution time in the original Opteron version of the code. For the FPGA accelerated version, creating this additional output greatly slows down the code. Another observation is that query sequence sizes of 8k or 16k exhibit similar performance, with 16k query

sequences giving only slightly better performance. One can also notice greater variability in the runs with the additional output. This is due to more interaction with the operating system when performing I/O, and possibly disk contention during the writes.

The speedups (over the 2.2 GHz Opteron) obtained (**Figs. 10**) are very consistent: 30x with alignment output and 50x w/o alignment output for *Tiger* (Virtex-II Pro 50) with a standard deviation from the mean of 0.16.

Even more exciting are the speedups (**Figs. 11**) of 43x with alignment output and 100x w/o alignment output for *Pacific* (Virtex 4 LX160) with a standard deviation from the mean of 0.13. With 100x speedup, searches that took 100 days (ie: 14 weeks) are now possible in one day.

The Virtex-4 LX160 has about 3x more logic available than the Virtex-II Pro 50, so it can process more copies of the algorithm in parallel. This additional logic space means it runs faster even though it runs at a slightly slower clock speed (125MHz) compared to the Virtex-II Pro 50 (140 MHz). The Virtex-4 (LX160) design has 128 SWPEs, compared to 48 for the Virtex-II Pro 50. As the timings indicate, it does more work, but it's clock speed is slightly less as it has more silicon area taking signals longer to propagate from one side of the FPGA to the other. Also, more code optimization is possible which could speed up the Smith-Waterman algorithm by another factor of two promising potential speedups of 200X. FPGA designs generally run at different clock frequencies, however, there is a maximum clock speed for a given FPGA which a given design will run slower than unless it is very simple or extremely well written. The original SWA design frequency started out at less than 100 MHz on the Virtex-II Pro 50 and was increased to 140 MHz by optimizing the design. Similar improvements are also possible on the Virtex-4 LX160 version, given more time.

Since there is very little difference in the speedup between the 8k and 16k query sequence lengths, the authors decided to make additional runs with the first query sequence to better ascertain whether the speedup is affected by query sequence size and database sequence size. The first query sequence and database set was run an additional 30 times splitting the query sizes up into sequences of length 1k, 2k, 4k, 8k, and 16k. Additionally, the database was split up into sequences of sizes 16k, 32k, 64k, 128k, 256k, and 512k characters. These jobs were then both run on the Opteron and Virtex-II Pro 50 to calculate FPGA speedups, shown in **Figure 12**, for each combination of query and database sequence size.

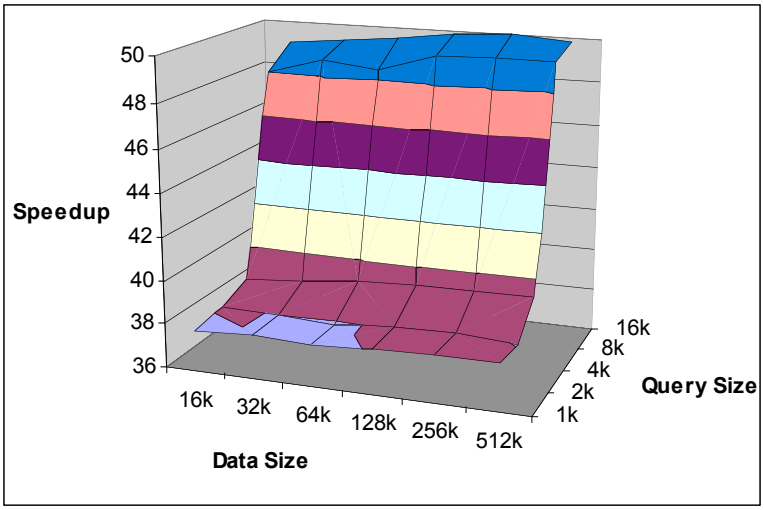

Figure 12. Speedup for Virtex-II Pro 50 FPGA for different query and database sequence lengths

The first observation shown in **Fig. 12** is that the size of the database has very little effect on the algorithm speedup. Similarly, query sequence sizes of 8k to 16k result in excellent speedups near 50x while even smaller query sequence sizes of 1k still give speedups of about 37x faster. The speedup of about 50X for larger query sizes coincides almost closely with speedups shown in **Fig. 10**. Similarly, one can well expect speedups of 100X for longer query sizes on the Virtex4.

5.3 Case 3 – Amino Acid Search

OpenFPGA Cases 1 and 2 are both nucleic acid data, which is represented by only four characters. FPGA programmers can take great advantage knowing there are only 4 characters and can perform all calculations using 2-bit integers. Amino acid data has many more characters and will not work for 2-bit integers. In an effort to evaluate FPGA performance on real applications, the OpenFPGA.org created a third test case involving amino acids.

Case 3 has three query sequences called myc, ras, and sec of lengths 60, 189 and 351 characters, respectively. These query sequences were then compared with the 24 human chromosomes, translated into amino acids. This translation was done for three different reading frames, with a total of 9 queries for 24 chromosomes. The three reading frame queries were performed back-to-back yielding one timing for each query sequence giving the Virtex2 speedup results shown in **Figure 13**.

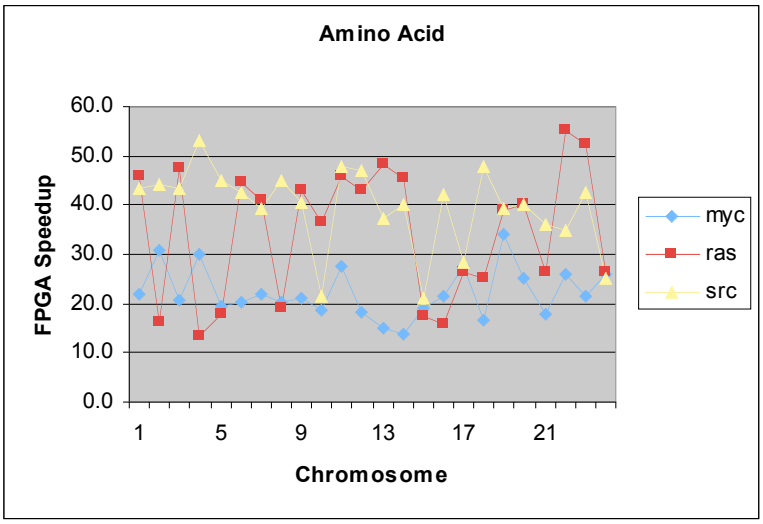

Figure 13. Virtex-II Pro 50 FPGA speedup for myc, ras and src sequences

Figure 13 shows considerably more speedup variability, attributed to very short FPGA run times, causing less accurate timing. **Figure 13** also shows that the shorter query sequence, myc, does not perform as well as the longer query sequence lengths in src. Despite this, the FPGA speedup obtained, even using amino acid input, is similar to that obtained previously for nucleic acid input.

6 Concluding Remarks

A description of FPGAs used in reconfigurable computing and lately used as supercomputer accelerators is given together with examples of their performance for comprehensive biological RNA, DNA and amino acid sequencing on Cray XD1 computers with both Virtex-II Pro 50 and Virtex-4 LX160 FPGAs. Significant speedups of up to 100x over traditional Opteron microprocessors were observed. Such

speedups are indicators of what may be expected on the Cray XT supercomputers outfitted with DRC's modules that contain Virtex-4 LX200 FPGAs and fit into an Opteron socket. Although development is still underway, the authors' feel theses advances will soon result in FPGA accelerators enabling supercomputer performance far beyond that possible solely with multi-core microprocessors.

References

1. Asanovic et al, The Landscape of Parallel Computing Research: A View from Berkeley, Tech. Report No. UCB/EECS-2006-183, http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html Dec 18 2006.
2. Sobieski, J. & Storaasli, O. Computing at the Speed of Thought, *Aerospace America*, Oct. 2004 p 35-38.
3. Strenski, Dave, FPGA Floating Point Performance, *HPCWire* - Jan 12 2007.
4. FASTA Sequence Comparison Program, University of Virginia
5. Margerm, Steve, Reconfigurable Computing in Real-World Applications, *FPGA Journal*, Feb 7 2006.
6. Yim, Michael, Jacobs, Adam and George, Alan, Performance evaluation of the Cray Bioscience Applications Package on the XD1 (*White Paper).*
7. Margerm, Steve, and Maltby, Jim; Accelerating the Smith-Waterman Algorithm on the Cray XD1 (*Cray White Paper* WP-0060406) 2006.
8. MitrionC/BLAST, http://www.hpcwire.com/hpc/1274236.html
9. Sternberg, M. (Ed.), *Protein Structure Prediction: A Practical Approach*, Chapter by Geoffrey Barton: Protein Sequence Alignment and Database Scanning, Oxford University Press ISBN 0199634963.
10. Smith T.F. and Waterman M.S. "Comparison of Biosequences." *Adv Applied Math.* 2: 482-489. (1981)
11. Smith T.F. and Waterman M.S. "Identification of common molecular subsequences." *J. Mol. Biol*. 147: 195-197 (1981)
12. Mount, D.W. In *Bioinformatics – Sequence and Genome Analysis*, Cold Spring Harbor Laboratory Press, New York (2001)
13. Pearson W.R. and Miller W., "Dynamic programming algorithms for biological sequence comparison." *Methods Enzymol*. 210: 575-601 (1992)
14. Yu C., Kwong K.H., Lee K.H., and Leong P.H.W. "A Smith-Waterman Systolic Cell" in *Proc. 13th Field-Programmable Logic and Applications* Springer, Berlin (2003)
15. Guccione S.A. and Keller E. "Gene Matching using Jbits" in *Proc. 12th Field-Programmable Logic and Applications* Springer, Berlin (2002)
16. Yamaguchi Y. and Maruyama T. "High Speed Homology Search with FPGAs" in *Pacific Symposium on Biocomputing* 2002 pp. 271-282 (2002)

Acknowledgment

This research was sponsored by the Laboratory Directed Research & Development Program of Oak Ridge National Laboratory managed by UT-Battelle for the U. S. Department of Energy under Contract No. DE-AC05-00OR22725. The U.S. Government retains a non-exclusive, royalty-free license to publish or copy the published form of this contribution, or allow others to do so, for U.S. Government purposes.

Appendix A

OpenFPGA.org Benchmark Test Data

There are three parts to this data set, designed to test different aspects of the benchmark. In each case, either the files or a tar of a group of files is available for downloading from the web page.

1. Micro-RNA (DNA Short Reference Sequences).

The reference sequences for this data set is micro_rna.fa and was taken from:
http://microrna.sanger.ac.uk/sequences/ftp/mature.fa.gz

An example of one entry from this file is:

```
>cel-let-7 MIMAT0000001 Caenorhabditis elegans let-7
UGAGGUAGUAGGUUGUAUAGUU
```

These are relatively short sequences (22 bases long), and originally used U rather than T. The version above have been edited to use T. There are 3685 sequences in this file, which is in fasta format. That is, each sequence is on two lines. The first line starts with a ">", and has a description of the micro-rna. The second line contains the dna sequence.

The test sequence is HsMar2006.tar.gz (the latest Human genome, and was taken from:
http://hgdownload.cse.ucsc.edu/goldenPath/hg18/bigZips/
File:chromFa.zip
This tar file contains both the original chromosomes, and the reverse-compliment versions of the chromosomes.

The alphabet for these databases are (upper and lower case) a, c, g, t, n (originally U in the reference sequence).

The match matrix is

	a	c	g	t	u	n
a	1	-1	-1	-1	-1	0
c	-1	1	-1	-1	-1	0
g	-1	-1	1	-1	-1	0
t	-1	-1	-1	1	1	0
u	-1	-1	-1	1	1	0
n	0	0	0	0	0	0

Gap enter and gap extend penalties = -10, -3

Threshold = 16. (If you use a match matrix of 5,-4, the corresponding threshold would be 83.)

2. Bacillus_anthracis (DNA comparison)

The test sequence is AE016879.fa and AE016879rc.fa (the reverse compliment of the above sequence) and each are approximately 5.23Mb in length.

The 18 reference sequences are in the file AE0170.tar.gzand all are approximately 290Kb in length.

The alphabet for these databases are A, C, G, T (lower case for the rc file).

Gap enter and gap extend penalties = -10, -3

The match matrix is again:

	A	C	G	T
A	1	-1	-1	-1
C	-1	1	-1	-1
G	-1	-1	1	-1
T	-1	-1	-1	1

Threshold = ?

3. Amino Acid Search

The test sequence is HsMar2006A.tar.gz
Each chromosome, in each direction has been converted to three ameno acid sequences (one for each reading frame), a total of six versions of each chromosome. In addition to the standard 20 ameno acid abreviations, * is used to represent a stop codon, and X to represent an unknown codon (usually one resulting from a sequence with N).

There are three reference sequences:

- c-myc
- c-ras
- c-src

The alphbet is the 20 leters representing the ameno acids but also include X and *.
A R N D C Q E G H I L K M F P S T W Y V X *

The match matrix is blosm-62, but has been expanded to also include X and *.

Gap enter and gap extend penalties = -9, -2

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	X	*
A	4	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-3	-2	0	-9	-4
R	-1	5	0	-2	-3	1	0	-2	0	-3	-2	2	-1	-3	-2	-1	-1	-3	-2	-3	-9	-4
N	-2	0	6	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2	1	0	-4	-2	-3	-9	-4
D	-2	-2	1	6	-3	0	2	-1	-1	-3	-4	-1	-3	-3	-1	0	-1	-4	-3	-3	-9	-4
C	0	-3	-3	-3	9	-3	-4	-3	-3	-1	-1	-3	-1	-2	-3	-1	-1	-2	-2	-1	-9	-4
Q	-1	1	0	0	-3	5	2	-2	0	-3	-2	1	0	-3	-1	0	-1	-2	-1	-2	-9	-4
E	-1	0	0	2	-4	2	5	-2	0	-3	-3	1	-2	-3	-1	0	-1	-3	-2	-2	-9	-4
G	0	-2	0	-1	-3	-2	-2	6	-2	-4	-4	-2	-3	-3	-2	0	-2	-2	-3	-3	-9	-4
H	-2	0	1	-1	-3	0	0	-2	8	-3	-3	-1	-2	-1	-2	-1	-2	-2	2	-3	-9	-4
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4	2	-3	1	0	-3	-2	-1	-3	-1	3	-9	-4
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	-2	2	0	-3	-2	-1	-2	-1	1	-9	-4
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5	-1	-3	-1	0	-1	-3	-2	-2	-9	-4
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5	0	-2	-1	-1	-1	-1	1	-9	-4
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6	-4	-2	-2	1	3	-1	-9	-4
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7	-1	-1	-4	-3	-2	-9	-4
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4	1	-3	-2	-2	-9	-4
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5	-2	-2	0	-9	-4
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11	2	-3	-9	-4
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	-1	-9	-4
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4	-9	-4
X	-9	-9	-9	-9	-9	-9	-9	-9	-9	-9	-9	-9	-9	-9	-9	-9	-9	-9	-9	-9	-9	-9
*	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-9	6

Threshold = ?