

# Exploring Accelerating Science Applications with FPGAs

Olaf O. Storaasli  
Oak Ridge National Laboratory  
[Olaf@ornl.gov](mailto:Olaf@ornl.gov)

Dave Strenski  
Cray Inc.  
[Stren@cray.com](mailto:Stren@cray.com)

## Abstract

FPGA hardware and tools (VHDL, Viva, MitrionC and CHiMPS) are described. FPGA performance is evaluated on two Cray XD1 systems (Virtex-II Pro 50 and Virtex-4 LX160) for human genome (DNA and protein) sequence comparisons for a computational biology code (FASTA). Scalable FPGA speedups of 50X (Virtex-II) and 100X (Virtex-4) over a 2.2 GHz Opteron were achieved. Coding and IO issues faced for human genome data are described.

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

## Introduction

This paper describes Field-Programmable Gate Arrays (FPGAs), several tools used to program them and how 100X speedup was achieved for a science application.

Remarkable innovations in computer technology [1-2] are fulfilling NASA future projections [3] for faster science and engineering computations. One innovation in the forefront is to harness FPGAs to accelerate High-Performance Computing (HPC) applications by one or more orders of magnitude over traditional microprocessors. FPGAs were invented in 1984 by Ross Freeman, co-founder of Xilinx Inc. FPGAs are extremely flexible and dominated by interconnections to thousands of embedded functions (Fig. 1. left) like adders, multipliers, memory, and logic slices (Fig. 2). They perform high-speed computations and communications (e.g., Hypertransport) in parallel via digital logic: LookUp Tables (LUTs), Registers, RAM, etc. Unlike “fixed” microprocessors, FPGA’s are reconfigurable “on the fly” by users in the “field”, thus, “field programmable”.

The Virtex-4 is available with one or more PowerPC (PPC) processors on the chip (Fig 1. right). The rapidly growing (15-20%/year) \$2B FPGA market (focused on the high-volume communications) is dominated by Xilinx and Altera. Aerospace and High-Performance Embedded Computing (HPEC) users are rapidly expanding their FPGA use. Although HPC sales (< 1%) are small, FPGA designers are open to HPC requirements for their next generation designs.

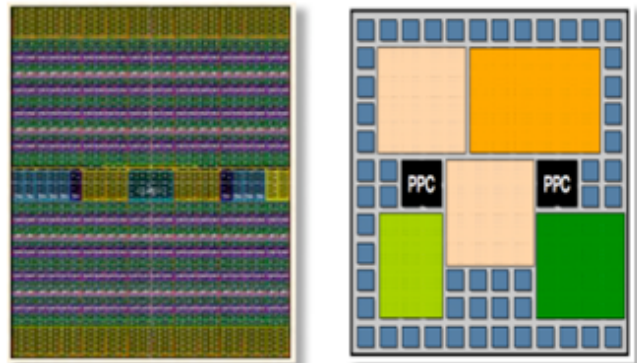


Figure 1. Virtex-4 FPGA with PPCs, memory, I/O (rt)

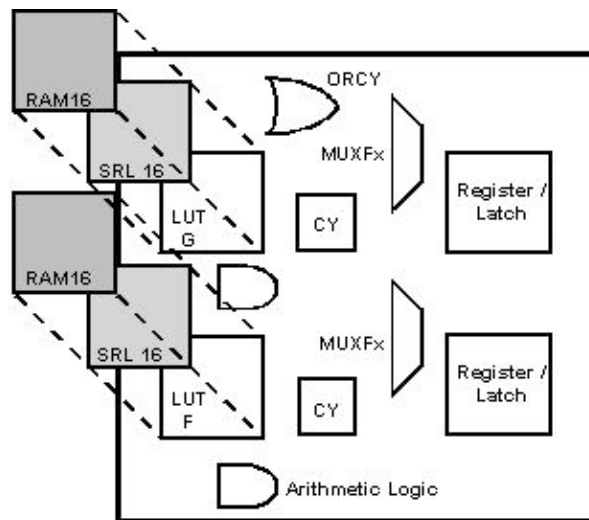


Figure 2. Virtex-4 FPGA logic slice: LUTs, RAM...

**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 triply-redundant code, 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 < 2% of their silicon (while drawing 10x FPGA power to perform only one or two operations).

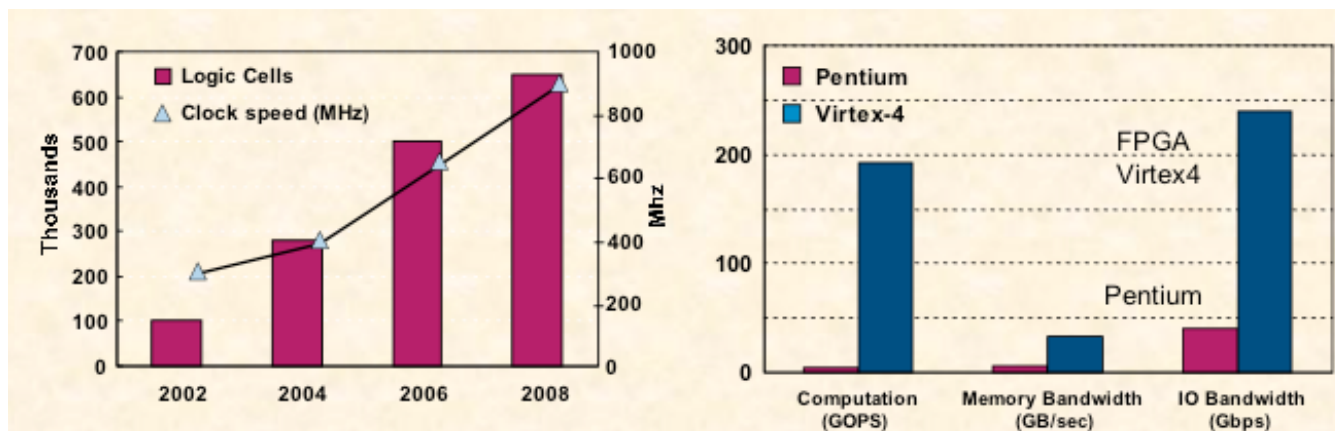


Figure 3. FPGA Characteristics: Logic Cell and clock speed growth, Computation and bandwidth speeds.

Figure 3 shows several key FPGA characteristics. Unlike microprocessors, FPGAs continue to advance at Moore's Law rate and have far to go before reaching logic cell and speed limits (Fig. 3., left). FPGA clock speeds (often 100-200 MHz) have far to go before facing heating issues that drove microprocessors to multi-core chips with reduced clock speeds. When FPGA applications are programmed to maximize parallelism, their computation speed far exceeds that of microprocessors (Fig. 3., right). Being high-speed communications devices, the memory and IO bandwidths also significantly exceed those of microprocessors (Fig. 3).

**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 applications in VHDL, its FPGA performance is excellent. In particular, applications using basic integer or logic operations (compare, add, multiply) such as DNA sequence comparisons, cryptography or chess logic, run extremely well 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 [2].

While early FPGAs had sufficient capability to be well suited for special-purpose HPEC, their use for general-purpose HPC was initially restricted to a first-generation of 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 provide modules with the latest FPGAs which fit in microprocessor sockets and use the same high-speed communications link. Cray selected DRC's module, Fig. 4, to accelerate its XT line of supercomputers.

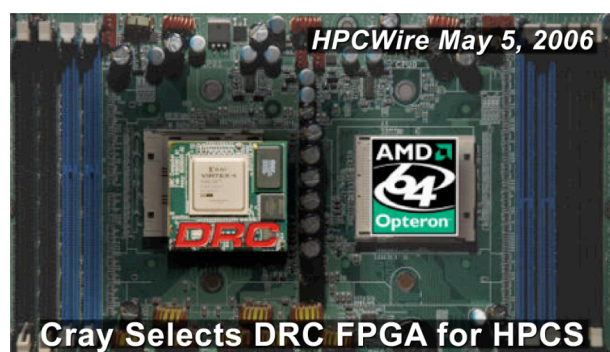


Figure 4. DRC Virtex-4 module in Opteron socket.

**FPGA and Multi-core for HPC:** 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 are increasingly attractive.

**Accelerator Options:** Three other accelerator options are available to HPC architects: Cell (IBM), Graphics (GPUs) and Array (ClearSpeed) processors. Like FPGAs, Cell and GPUs have vast commercial markets (video games and graphics) driving down costs, promoting competition and stimulating advances making them increasingly attractive to HPC. Array processors, however, are custom devices

requiring amortization over relatively few users. GPUs require significant 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, tools/software are simplifying their use for HPC: Viva, MitrionC, & Xilinx's CHiMPS (all discussed later) as well as DSPlogic, ImpulseC, Celoxica, Aldec, and, others. The authors are testing CHiMPS for HPC applications.

## Several FPGA Programming tools:

**1. VHDL/Verilog:** Although difficult and time-consuming for HPC programmers to learn and use due to coding requirements for esoteric details including timing signals and pinouts, VHDL should not be cast aside, but rather, should be used as the performance "gold standard" for other tools to be compared with. VHDL/Verilog algorithms may require months or even years to code and optimize, and then extensive rewriting when the next generation FPGA chip appears. This is because it is often programmed and tweaked by the circuit designer to be so close to the specific FPGA hardware. Since FPGA origins are with circuit designers, it is no accident that the vast majority of FPGA applications are written by circuit designers in VHDL or Verilog and/or using VHDL libraries or cores. Cores is the terminology used for the concept of a function written in VHDL or Verilog which is of such wide-scale interest and application that is made available to directly load on the "core" of fabric of an FPGA. Such cores are available for a myriad of functions, including Floating Point arithmetic, Digital Signal Processing, FFTs, Hypertransport communication etc.

**2. Viva (Starbridge Systems):** A VHDL/Verilog alternative to program FPGAs is Viva [4], a graphical icon-based tool shown on the left of Fig. 5 and used at NASA.

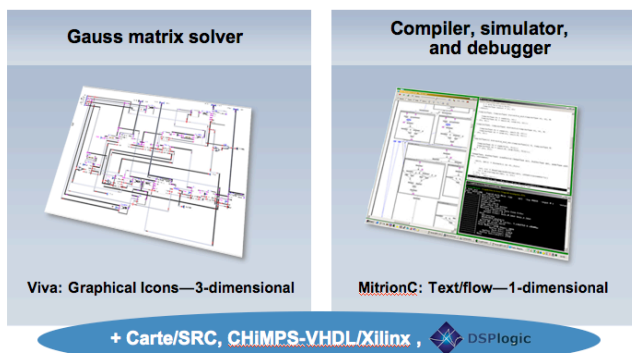


Figure 5. 3D Graphical Viva & 1D MitrionC 1D

Similar to Labview, Viva allows FPGA users to write/run scientific applications by avoiding the esoteric digital logic timing and pinout issues that require much attention in VHDL/Verilog. A free trial copy may be downloaded and run on a Windows or Unix PC. Programming FPGAs in Viva is a two-step process, the first step is creating your program graphically on a PC. This allows debugging and

algorithm checkout to be performed on a PC before beginning step 2, the place and route synthesis on the FPGA system. With no knowledge of VHDL/Verilog, all the science and engineering algorithms shown in Fig 6. were developed and run on the Starbridge Systems HC-36 with up to 10 VirtexII-6000 FPGAs. Viva's logical, intuitive options made it easy to learn and use, even for college and high school students working with the author. MPI and Viva were used by the author to teach a graduate parallel programming course. The students became proficient in both MPI and Viva (with similar learning times) and used both to complete a series of identical linear algebra/matrix assignments. Viva was marginally preferred over MPI by most graduate students. The most challenging algorithms developed (Fig. 6) were the cordic

### Algorithms Developed

- **Matrix Algebra:**  $\{V\}$ ,  $[M]$ ,  $\{V\}^T\{V\}$ ,  $[M]x[M]$ ,  $GCD, \dots$
- **$n!$   $\Rightarrow$  Probability:** Combinations/Permutations
- **Cordic  $\Rightarrow$  Transcendentals:** sin, log, exp, cosh...
- **$\partial y/\partial x$  &  $\int f(x)dx$   $\Rightarrow$  Runge-Kutta:** CFD, Newmark Beta: CSM
- **Matrix Equation Solvers:**  $[A]\{x\} = \{b\}$ , Gauss & Jacobi
- **Dynamic Analysis:**  $[M]\{\ddot{u}\} + [C]\{\dot{u}\} + [K]\{u\} + NL = \{P(t)\}$
- **Nonlinear Analysis:** reduces NL time
- **Analog Computing:** digital accuracy
- **Structural Design/Optimization**
- **Unsolved App:** Traveling Salesman

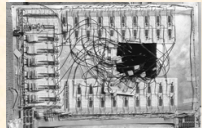


Figure 6. Viva algorithms developed/run.

algorithm (for transcendental functions), matrix equation solvers, structural design/optimization and solution of the traveling salesman problem. The typical speedup observed for these applications on Starbridge's HC-36 was 5-7X over the HC-36 microprocessor. Although the Viva experience was favorable, Starbridge's HC-36 slow PCIX bus linking FPGAs to user data restricted the size and class of applications (minimal I/O) that could achieve high performance. This issue is being addressed by Starbridge and other 2<sup>nd</sup> generation reconfigurable supercomputer developers. A Viva strength is that step 2 (place & route) is "hands off" so HPC users can focus on developing/debugging algorithms. The algorithms shown in Fig. 6 were performed on single FPGAs assigned to different users (on the same network). Viva's ability to run on multiple FPGAs was also demonstrated.

**3. MitrionC:** Another innovative approach for HPC users to program FPGAs without VHDL/Verilog (or the circuit design skills they require), is provided by Mitronics. Users program the so-called "Mittrion Virtual Processor" in MitrionC [5], with C additions for FPGA memory and data access. MitrionC, like Viva is free to download. Step 1 (major part used to develop algorithms) is run on a PC/Mac (Windows/Linux/ MacOSX). Step 2, initially was complex/esoteric on Cray XD1 systems, but is now improved. The author's recent experience with Mitrion unrolled BLAST [5] genome matching code (1200 lines of

Mitron C calling a VHDL hash code function) on SGI's RASC Virtex 4 FPGAs indicates significant FPGA

application performance. Mitron BLAST code achieves performance speedups of 10-16X on the Virtex 4.

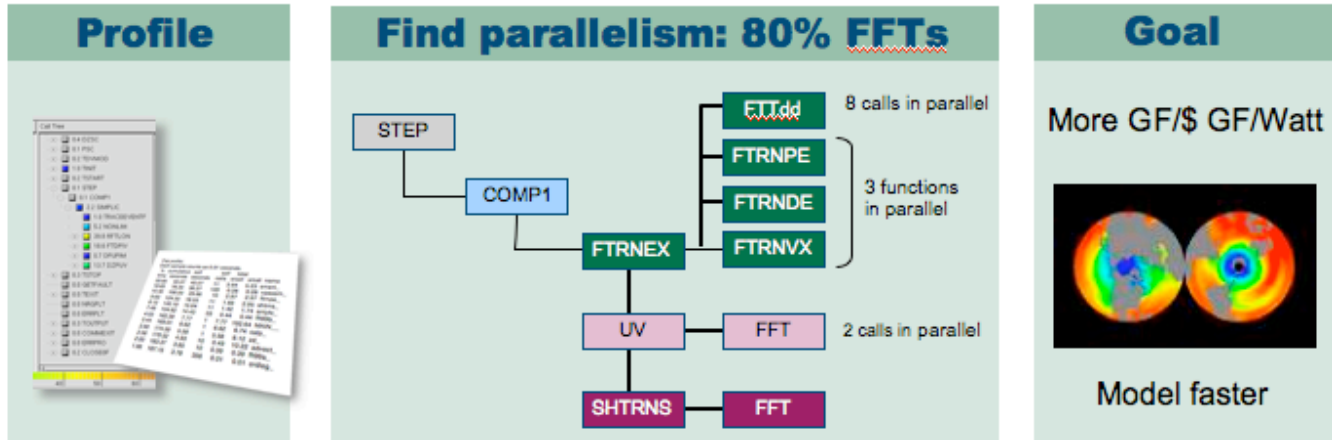


Figure 7. Weather/climate code port to CHiMPS.

**4. CHiMPS (Xilinx):** Rounding out FPGA tools is a joint effort Xilinx (major FPGA vendor) is conducting with the authors and others to evaluate the Xilinx's CHiMPS [6] to simplify porting major science/engineering applications to FPGAs. ORNL and Xilinx scientists profiled a popular weather/climate code (STSWM) for early CHiMPS testing (Fig. 7). Following this first successful test, a workshop was held this spring at Xilinx for initial CHiMPS users aimed at simplifying porting HPC codes run on FPGA-enabled supercomputers. A molecular dynamics open-source test code was ported to CHiMPS in 20 minutes and optimized in another 20 minutes. The authors are helping Xilinx's HPC team make CHiMPS a practical method to accelerate HPC applications via FPGAs.

### FASTA Application

FASTA [7] is used for protein: protein, DNA:DNA, protein: translated DNA and ordered or unordered peptide searches. It calculates similarity statistics for biologists to determine if alignments are random or homotopic. 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 [8].

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 eliminating segments unlikely to be alignments containing the highest score segment.
4. Use dynamic programming to optimize alignment in narrow band of top scoring segments.

FASTA's input format is widely used by other search tools including BLAST [5]. The FASTA component used here was ssearch34 [5], which calls the Smith-Waterman algorithm [9-10], the essentials of which follow.

### 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 (Fig. 8) is generated with the query sequence characters across the top and database sequence characters down its side. The table is then filled with score

values reflecting the quality of an alignment at a specific offset. The high score in the table indicates the best potential to solve these sub-problems in parallel.

The score in each 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 total alignment is calculated by solving simpler sub-problems simultaneously for the many table score values in parallel. Once scores for one row or column has begun, adjoining rows or columns can be computed in parallel.

		Query Sequence						
	0	A	C	G	T	...	C	
0	0	0	0	0	0	0	0	
A	0	2	0	0	0	2	0	
C	0	0	4	2	1	0	2	
G	0	0	2	6				
A	0							
A	0							
C	0							
...	0							
G	0							

Figure 8. Example of Smith-Waterman Algorithm.

Fig. 8 shows a query sequence “ACGT...C” (top) and a larger database sequence “ACGAAC...G” (side). The first row and column of the Smith Waterman table are initialized to zero. Scores are then calculated starting in the upper left corner and moving outward. Fig. 8 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.

More important than cell calculation mechanics is how the algorithm is broken into smaller sub-problems and solved in parallel [8-10].

### 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.

**Code Profile:** Often only a small algorithm kernel can or need be placed on an FPGA to achieve significant acceleration. Applications with most computation in this kernel benefit the greatest from FPGA acceleration. The ssearch34 code profiles (Fig. 9) show 98.61% of its time is

in FLOCAL\_ALIGN (optimized Smith-Waterman code) to calculate the maximum alignment score for two sequences. This function is an ideal candidate for acceleration, provided it can be efficiently offloaded to FPGAs to capture the Smith-Waterman’s inherent parallelism. Parallelism is key to FPGA performance, and also allows designs to scale well. It is 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.

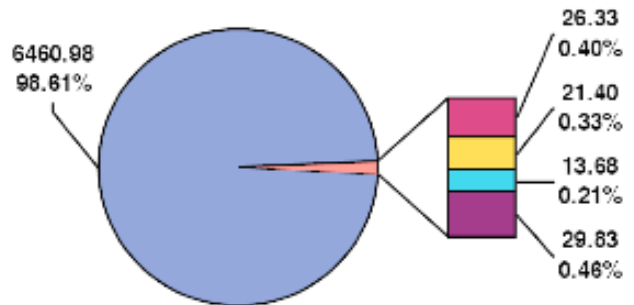


Figure 9. Profile of ssearch34 Application.

**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, FPGAs must perform many operations in parallel. For the Smith-Waterman algorithm, many alignment scores can be calculated in parallel. Additionally, some computations of single alignment scores can also be performed in parallel.

**Instruction Efficiency:** Modern 64-bit processors have powerful, general-purpose instruction sets. However, for many simple application calculations (e.g. compares), using 64-bit microprocessors is extreme overkill and wasteful. FPGAs use the minimal logic required for given calculations, freeing silicon to exploit parallelism. The basic Smith-Waterman data types are character sequences. Each character is represented by as few as two bits drastically reducing the logic required for each calculation.

**Bandwidth, Data Localization:** Bandwidth, rather than computation speed limits the performance of many algorithms. Bandwidth limitations can occur 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 (i.e., microprocessor internal cache is substantially faster than external cache, and faster again than external SDRAM). The same holds true for FPGA memory subsystems (i.e., code limited by microprocessor SDRAM bandwidth can be similarly limited on FPGAs).

Next, to harness parallelism, we determine if sufficient bandwidth exists between the microprocessor and FPGA, and between the FPGA and its memory. 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 FPGA making it unlikely that the bandwidth available to send the sequences to the FPGA is a bottleneck. Likewise, since the only data returned from the FPGA is the maximum score, the bandwidth from the FPGA to the microprocessor is also not likely a limitation. Thus, the only limitation is how quickly the FPGA can calculate scores.

**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 in calculating 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. The size of the memory available, however, will restrict the maximum length of the query and database sequences.

### 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 (PEs). The PEs are chained in a Smith-Waterman FPGA pipeline algorithm [7] (Fig. 10) to calculate the maximum alignment score for two sequences.

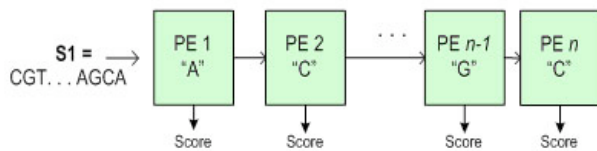


Figure 10. Smith-Waterman FPGA Pipeline.

One query character is preloaded into each processing element which then calculates scores in the column of that query character. The database string (S1) is shifted through the pipeline so each database character is compared to each query character in parallel, resulting in a table of scores such as is shown in Figure 11.

		Query Sequence						
		0	A	C	G	T	...	C
Database Sequence	0	0	0	0	0	0	0	0
	C	0	0	0	0	0	0	0
	G	0	0	0	0	0	0	0
	T	0	0	0	0	0	0	PE N
	⋮	0	0	0	0	0	PE ...	↓
	T	0	0	0	0	PE 4	↓	
	A	0	0	0	PE 3	↓		
	A	0	0	PE 2	↓			
	G	0	PE 1	↓				
	C	0	↓					
A	0							

Figure 11. Smith-Waterman Parallel Scoring.

Most of the accelerator design is building the PE pipeline. However, added logic is required to feed the PEs, interface the array logic to the microprocessor, and to access the external QDR II SRAMs surrounding the FPGA.

In addition to PEs, 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.

Control logic provides status and control registers for the Opteron to write the final scores back to the Opteron's local DRAM memory. This is done 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.

### OpenFPGA Benchmark Results:

FPGA speedups were obtained for the openfpga.org 4GB human genome benchmark using FASTA's ssearch34. All three OpenFPGA benchmark test cases were run and results made available to download/compare both on openfpga.org

and [ft.ornl.gov/~olaf/fpga](http://ft.ornl.gov/~olaf/fpga) to compare/share results with the authors. The results of the three cases follow.

### 1: Micro-RNA (DNA Short Sequence Search)

This case required 3685 query sequences searching across all 24 human genome chromosomes. To establish a baseline time, a run was made on a single Opteron (with default options} for all the query sequences against the first chromosomes. 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. FPGA runs were then made in parallel using MPI (Fig. 12), and showed, as expected, excellent FPGA scaled speedup (red),

	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 12. Cray XD1 hours to run ssearch34 on chromosome one of the human genome.

Although promising, the authors observed the Opteron output was extensive and varied slightly from the FPGA output. On advice of OpenFPGA experts, two ssearch34 options were selected and used in subsequent runs. Minimal output produced all scores for all searches:

Detailed: -Q -H -f -10 -g -3 -d 10 -b 10 -s  
 Minimal: -Q -H -f -10 -g -3 -d 0 -b 10 -s

**2. Bacillus anthracis DNA search:** Genome matching was performed on Virtex2, Virtex4 and Opetron Cray XD1 configurations for 18 DNA query sequences: AE017024-AE017041 on a large database, AE016879 for two outputs: Each query sequence (~300,000 characters) was compared with ~5 million character database. Each run took over 3 days on the 2.2 GHz Opteron. As the FPGA Smith-Waterman code was limited to a maximum query size of 16k characters (and maximum database size of 512k characters), code was written to split the input query and database into smaller sequences. Ssearch34 results were then obtained for 16k and 8k query sizes for two output options on two Cray XD1 systems (ORNL’s *Tiger-Virtex-II Pro 50* and Cray’s *Pacific-Virtex4 LX160*) and compared with Opteron to determine FPGA speedups (Figs. 13-14). **Figures 13-14** (blue/red) with detailed alignment sequence output show Virtex2/Virtex4 speedups to 29X/43X (1.8/3.9 standard deviation). Reduced output (yellow/ green) increased speedups to 50X/ 100X (0.16/0.13 standard deviation) with 16k queries slightly faster than 8k. Output IO (performed by the Opteron) was small for the Opteron’s 75-hour solution time, so it was not optimized. However, reducing the additional output gave significant speedups up to 100X, but only minor reductions on the Opteron.

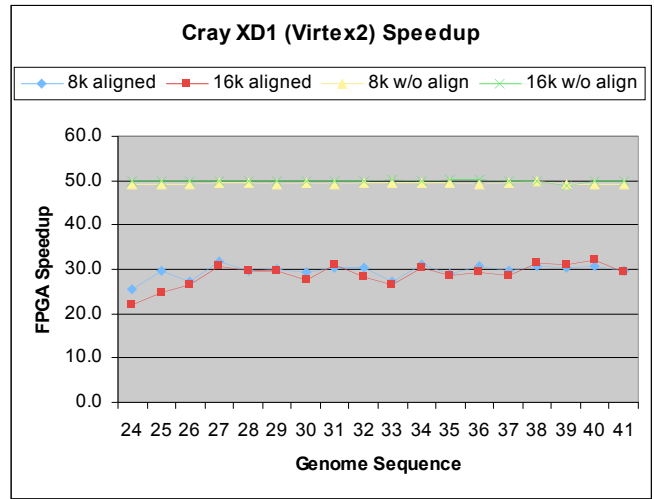


Figure 13. Virtex-II Pro 50 FPGA speedup.

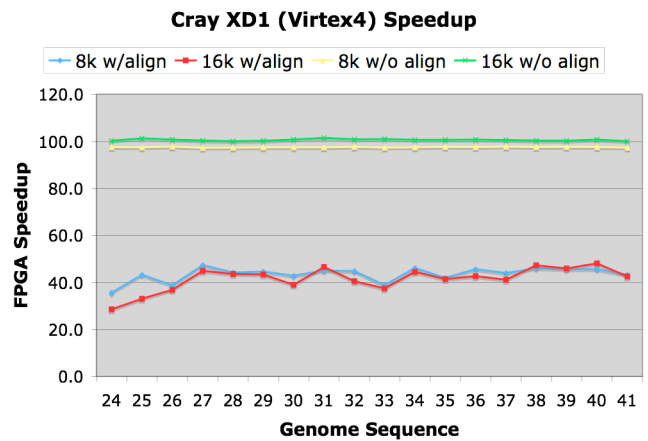


Figure 14. Virtex-4 LX160 speedup.

**Analysis:** 3X more gates (128 SWPEs) on the Virtex-4 LX160 can run 3X more algorithm copies in parallel than the Virtex-II Pro 50 with 48 SWPEs). With this added logic area, it runs faster despite its 125MHz clock (140MHz for Virtex-II Pro 50) required for signals to travel across the FPGA. Code optimization would likely double the FPGA performance to 200X. Just as the original 100 MHz Virtex II algorithm was increased to 140 MHz, similar Virtex4 optimization is also possible.

**Query and Database Sizes:** Speedup similarity for 8k and 16k queries, prompted additional study on the impact of query and database size on FPGA speedup. The same query sequence and database set was run 30 more times splitting the query sizes into sequences of length 1k, 2k, 4k, 8k, and 16k. The database was then split into sequences sizes: 16k, 32k, 64k, 128k, 256k, and 512k characters. Virtex II Pro FPGA speedups varied from 37-50X the Opteron (Fig. 15) for 1k-16k queries with minor variations in data size. Larger query sizes (8k and 16k) gave better (~50X) speedups as in Fig. 13 (100X expected for Virtex4).

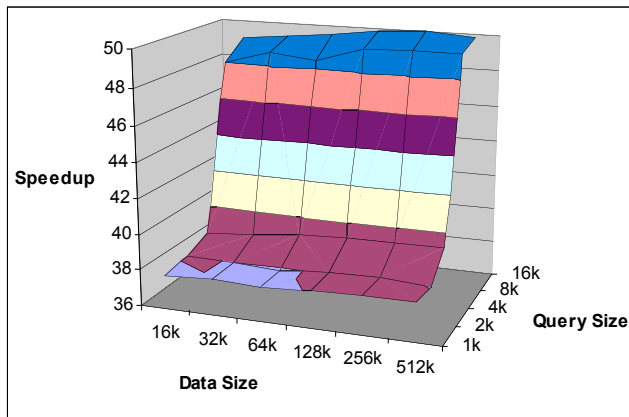


Figure 15. Speedup for Virtex-II Pro 50 FPGA.

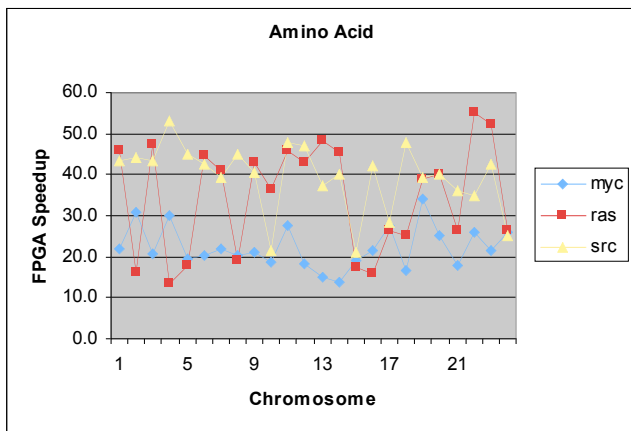


Figure 16. Virtex-II speedup for myc, ras and src.

**3: Amino Acid Search:** Unlike Cases 1-2 (4-character nucleic acid data representation), amino acids use more than 4 characters (2-bit integers). Query sequences of 60 (myc), 189 (ras) and 351 (src) characters are compared against the 24 human chromosomes, translated via 3 frames into amino acids (9 queries for 24 chromosomes). FPGA speedups (Fig. 16) for Amino acid queries (myc, ras, src) in the openfpga.org benchmark are similar, but exhibit a much wider variation among chromosomes (particularly src). This variation is attributed to their longer sequence size.

## Summary

An overview of FPGAs is given including experience using several FPGA programming tools: VHDL, Viva, MittrionC and CHiMPS. FPGA performance was evaluated using the FASTA (sssearch34) code for comprehensive biological 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 2.2 GHz Opteron processors were achieved, with better speedups for larger query sizes. The speedups are clearly scalable (shown on 5 Cray XD1 FPGAs) to achieve ~500X speedup over one Opteron. These results indicate similar speedups are likely for acceleration modules (DRC and Xtreme data) that fit in Opteron sockets, both on small embedded systems and Cray XT supercomputers.

## References

- [1] Asanovic et al, The Landscape of Parallel Computing Research A View from Berkeley, *Tech. Rpt # UCB/ECS-2006-183* Dec 18 2006. [Google](#) ECS-2006-183
- [2] Strenski, Dave, FPGA Floating Point Performance, *HPCWire* - Jan 12 2007. [Google](#) Strenski wire
- [3] Sobieski, J. & Storaasli, O. Computing at the Speed of Thought, *Aerospace America*, Oct. 2004 p 35-38. [Google](#) aiaa speed thought
- [4] Viva Software: [Google](#) Viva Starbridge
- [5] MittrionC/BLAST: [Google](#) Mittrion BLAST
- [6] Bennett, Dave: An FPGA-oriented target language for HLL compilation, RSSI '06, NCSA, UIUC. [Google](#) Bennett RSSI06
- [7] FASTA Sequence Comparison Code: [fasta.bioch.virginia.edu](http://fasta.bioch.virginia.edu)
- [8] Sternberg, M. (Ed.), Protein Structure Prediction: A Practical Approach, Chapter by Barton: Protein Sequence Alignment and Database Scanning, *Oxford University Press* ISBN 0199634963.
- [9] Margerm, Steve, and Malby, Jim; Accelerating the Smith-Waterman Algorithm on the Cray XD1, *Cray WP-0060406* 2006.
- [10] Storaasli, Olaf, Yu, Weikuan, Strenski, Dave, & Malby, Jim; Performance Evaluation of FPGA-Based Biological Applications, *Cray Users Group Proceedings*, Seattle WA, May 2007. <http://ft.ornl.gov/~olaf/pubs/CUG07Olaf17M07.pdf>

## Acknowledgment

This research was sponsored by the Laboratory Directed Research & Development Program of ORNL managed by UT-Battelle for the U. S. Department of Energy on Contract DEAC0500OR22725. 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.