Exceeding 100X Speedup/FPGA
Cray XD1 Timing Analysis Yields Further Gains

Olaf O. Storaasli, *Oak Ridge National Laboratory*
and **Dave Strenski**, *Cray*

ABSTRACT: *Field Programmable Gate Arrays (FPGAs), which promise to accelerate calculations by one or more orders of magnitude, are of increased interest in speeding HPC applications. Our CUG 2008 paper evaluated performance of two Cray XD1 systems using FASTA, a computational biological human genome comparison program. Results indicated typical Cray XD1 FPGA speedups of 50x (Virtex-II Pro 50) and 100x (Virtex-4 LX160) compared to a 2.2 GHz Opteron. These results were shown to be scalable to 150 FPGAs. A data transfer bottleneck was identified causing the FPGA computations to stop while I/O was performed by the Opteron. Testing showed this I/O time dominated processing time to such an extent that actual FPGA computation time was almost negligible compared to Opteron I/O computations. This paper proposes two I/O streaming procedures to overcome this bottleneck, to significantly reduce I/O, yielding up to 10X additional speedup (above the 100X already achieved) for human genome sequencing.*

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

1 Introduction

Computer technology innovations[1] are fulfilling long-term projections[2] for faster science and engineering computations. Heterogeneous computers[3] with hardware accelerators, show the potential to speed High-Performance Computing (HPC) applications by one or more orders of magnitude over traditional processors. In particular, low-power FPGAs, invented in 1984 by Ross Freeman, Xilinx co-founder, are extremely flexible devices (**Fig. 1**. left) containing thousands of functions (Computational Logic Blocks) and optional "on-board" processors (**Fig. 1**. center). These functions (**Fig. 1**. right) include adders, multipliers, memory, LookUp Tables (LUTs), Digital Signal Processors (DSPs) and high-speed communication. Unlike "fixed" processors, FPGA hardware gates are reconfigurable (changeable "on the fly") by users in the "field" (thus, field programmable).

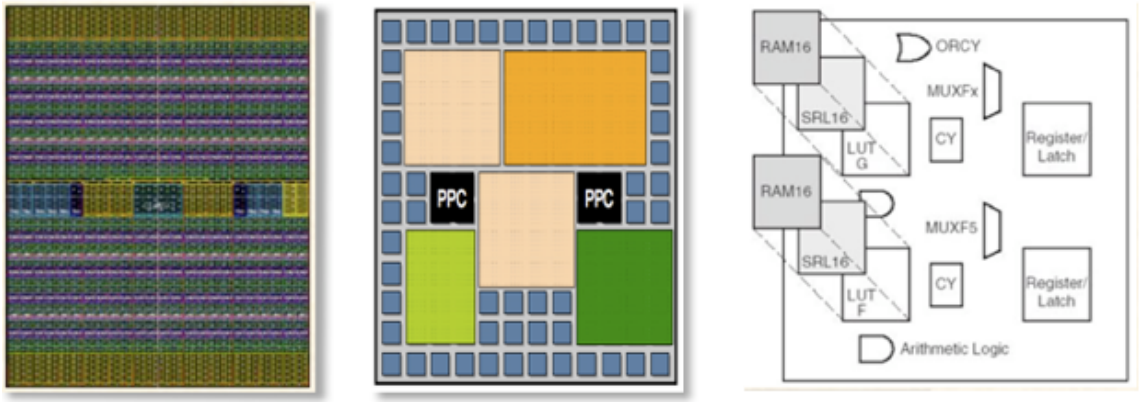

Figure 1. Virtex-4 FPGA (left), PPC processors, memory, I/O (center) and logic slice (right)

FPGA Characteristics: Compared to processors, FPGA layout is extremely regular, simplifying fabrication, and allowing FPGAs to be among the first to reduce feature sizes (90nm => 65nm => 45nm). This regularity with redundant algorithms, limits radiation damage (i.e. NASA Mars Rovers), making

low-powered FPGAs popular for space, flight and military environments. Thousands of operations per clock cycle are possible for FPGA codes to maximize silicon use (>90%), compared to processors whose 1-2 operations/cycle (< 2% of silicon/cycle) draw 10x FPGA power. Fueled by the growing, high-volume, telecommunications market, FPGAs have already experienced 3 generations of "spin-off" to HPC.
First generation FPGAs, communicated via PCI, replaced discrete logic devices, digital signal processing and, with the help of DARPA's ANS Program, became attractive for High-Performance Embedded Computing (HPEC). Their substantial speed increases over traditional ASICS were realized for FPGA-centric applications with limited I/O using the "slow" PCI bus.

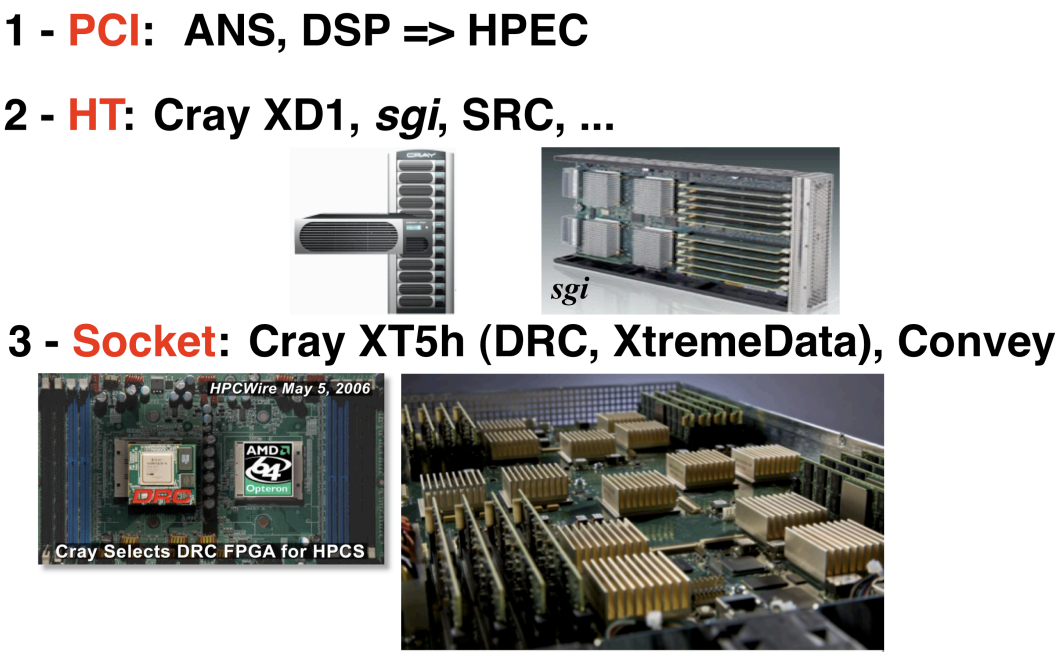

Figure 2. Three HPC-related Generations of FPGAs (PCI, HyperTransport and socket interfaces)

Second generation FPGAs (**Fig. 2**. center) offered higher communication via Hypertransport and RocketIO interfaces, considerably increasing the scope and size of potential applications. FPGAs made serious inroads into the large DSP and HPEC markets and more serious HPC applications. However, HPC sales (< 1%) were a spinoff rather than a driver for FPGA designs, and programming in VHDL/Verilog was a serious impediment, as was lengthy (overnight) compile (place/route) times.

Third generation FPGAs (**Fig 2**. bottom), characterized by socket compatibility with processors, use the same high-speed I/O communication: AMD's HyperTransport/AMD or Intel's PCI-Express or QPI. Convey (**Fig 2**. bottom right) extended this socket innovation to connect their companion accelerator board (4 computation FPGAs and 10 service FPGAs) to a dual-socket X86 board.

FPGA Coding: FPGAs were developed by logic designers, so for efficiency they are programmed using VHDL/Verilog circuit design languages. 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 longer 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. A number of "C-to-gate" languages have emerged, some including VHDL libraries. This software is generally FPGA-specific, expensive, slow (compared to VHDL/Verilog), but still has attracted advocates.

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. Floating point and double-precision applications rapidly exhausted the slices available on early FPGAs, so they were often avoided. This situation has changed for current FPGAs, which now have sufficient logic to fit about 80 parallel 64-bit multiplies[4].

2 FASTA DNA and Protein Search/Alignment

FASTA[5] (*fasta.bioch.virginia.edu*) contains programs for protein:protein, DNA:DNA, protein:translated DNA (with frameshifts), and ordered/unordered peptide searches using a unique search heuristic with the optimal Smith-Waterman algorithm[6-8]. 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 the same used for other sequence alignment programs and database search tools (i.e. BLAST[9]). FASTA's speed is attributed to its 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 word size selected 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[10].

1. Identify regions of highest density in each sequence comparison
2. Re-score using PAM scoring matrix, keeping top scoring segments.
3. Use joining threshold to remove segments unlikely to contain the highest score segment.
4. Optimize alignment in a narrow band of top scoring segments via dynamic programming.

Ssearch34 in FASTA uses the Smith-Waterman algorithm[6-17], whose essentials are summarized next.

3 Smith-Waterman Algorithm

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 scoring table (**Fig. 3**) is generated with query sequence characters written across the top and database sequence characters 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 a row or column have begun, calculations for adjoining rows or columns may begin in parallel.

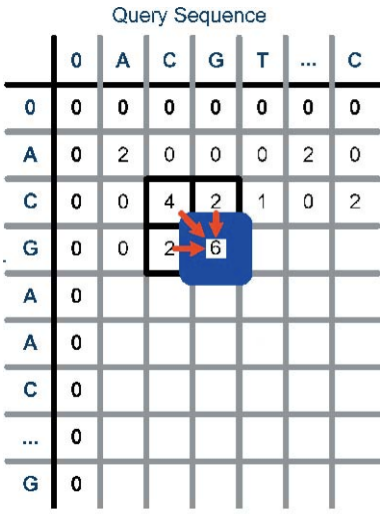

Figure 3. Smith-Waterman Algorithm Scoring

Fig. 3 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. **Fig. 3** 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.

The Smith-Waterman algorithm can be divided into sub-problems and solved in parallel on FPGAs[6-17].

4 Algorithm Acceleration

The Smith-Waterman algorithm is an excellent candidate for FPGA acceleration as it is a small code kernel (**Fig**. 4.) which calculates the maximum alignment score of two sequences that consumes 98.61% of all FASTA calculations. Numerous copies of this kernel were placed in parallel as a linear systolic array of processing elements (PEs) in a pipeline (**Fig. 5**.) on each VirtexII and Virtex4 FPGA. Twice as many PEs fit on Virtex-4 FPGAs, so their solution speed was double that of VirtexII FPGAs.

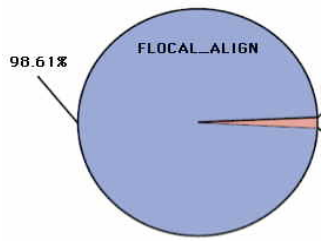

Figure 4. Ssearch34 Timing Profile

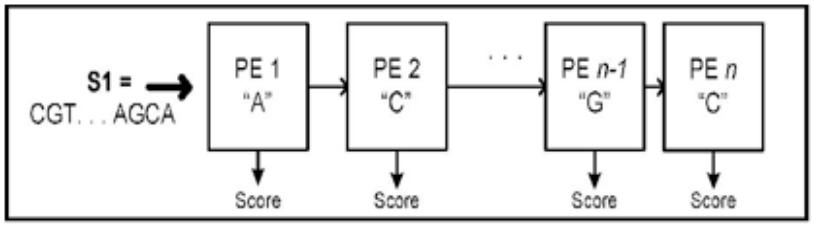

Figure 5. Smith-Waterman Pipeline

One query character is preloaded into each PE which then calculates a score in the column for that query character. The database string (S1) is shifted through the pipeline until each database character is compared to each query character, resulting table of scores, **Fig. 6**., filled from top to bottom over time. Building the pipeline of PEs comprises most of the accelerator design. However, additional logic (**Fig. 7)** is required to feed the PEs, interface the array logic to the processor, 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.

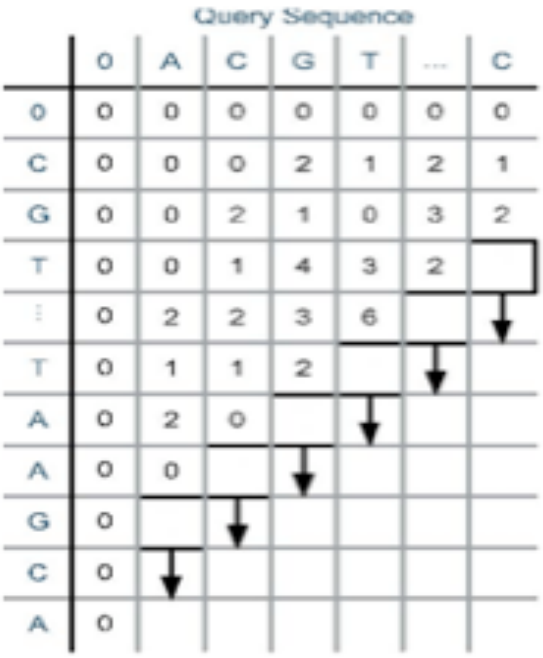

Figure 6. Smith-Waterman Score Calculation

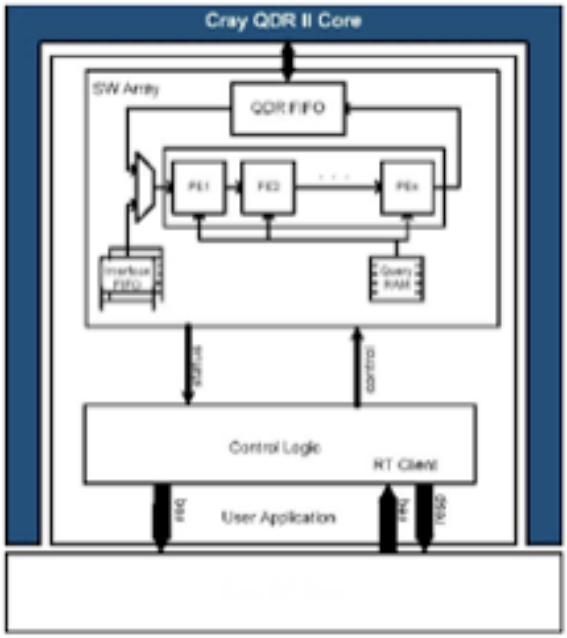

Figure 7. Overall Smith-Waterman Design

External QDR II SRAM stores intermediate results generated when query sequences exceed the number of PEs. Four external QDR II SRAMs are accessed via the Cray QDR II Core and internal block RAM is 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 bursts rather than one character at a time. The Control Logic block (**Fig. 7)** provides status and control registers for the Opteron, and writes the final scores back to the Opteron's local DRAM memory. It does this by interfacing with the Cray RT Core, which processes read/write requests to/from the Opteron. The status and control registers allow the processor to set up the logic for a given alignment and detect errors that may have occurred during its operation. When the alignment is complete, the maximum score generated is sent to the Opteron.

5 Results: Bascillus_anthracis DNA comparison

FPGA speedups for the Bacillus_anthracis (Human DNA comparison) were obtained using the FASTA[5] application ssearch34, programmed to call an FPGA version of the Smith-Waterman algorithm[6]. Two sets output options were used, the first (verbose) prints all alignment sequence details and a second (minimal) output with only the scores from the searches.

Verbose 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`

The data are 18 DNA query sequence files (AE017024-41) and a large database file (AE016879). Each query sequence contains ~300 thousand characters and the database exceeds 5 million characters. The FPGA version of the Smith-Waterman algorithm limits query sizes to 16k characters and the database size to 512k characters. To overcome these limits and maintain 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 to both Opteron and FPGA versions of ssearch34. Runs were compared for both 16k and 8k character sequence sizes (both verbose and minimal output) using Virtex-4 LX160 FPGAs.

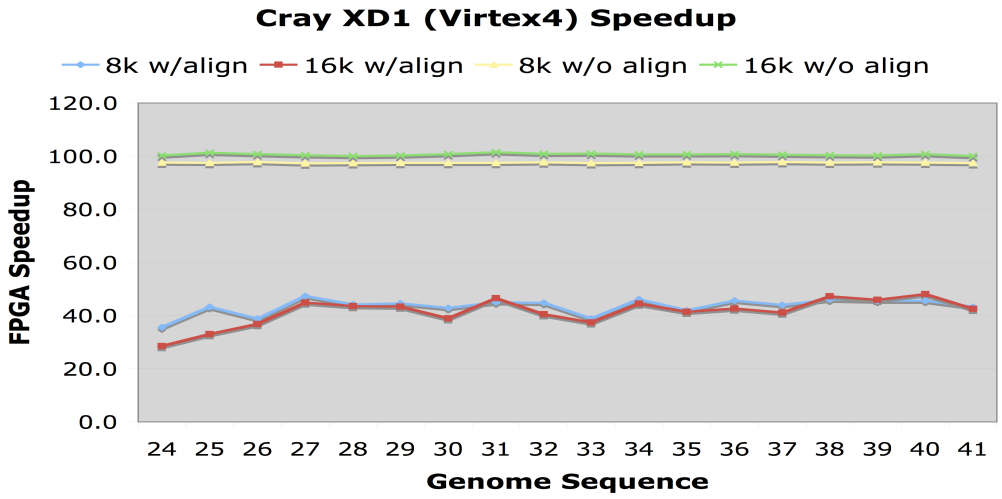

Figure 8. FPGA speedup for 8k and 16k sequence lengths (verbose & minimal output)

Fig. 8 shows that using minimal output of aligned sequence data (green-yellow) significantly improves performance (and reduces variability) to 100X speedup compared to verbose output (blue-red curves) with ~40X speedup. The output code took such a small part of the overall execution time for the Opteron version of the code that it was not optimized, so it slows down the faster FPGA version, and actually halts FPGA computations while the Opteron is processing output. 8k and 16k query sequence sizes exhibit similar performance, with 16k query sequences perfroming only slightly faster.
With 100x speedup, searches that took 100 days (ie: 14 weeks) now take one day. To see if this speedup is scalable, a large FASTA application comparing Human and Mouse DNA was tested on the Naval Research Laboratories 150 FPGA Cray XD1 (**Fig**. 9).

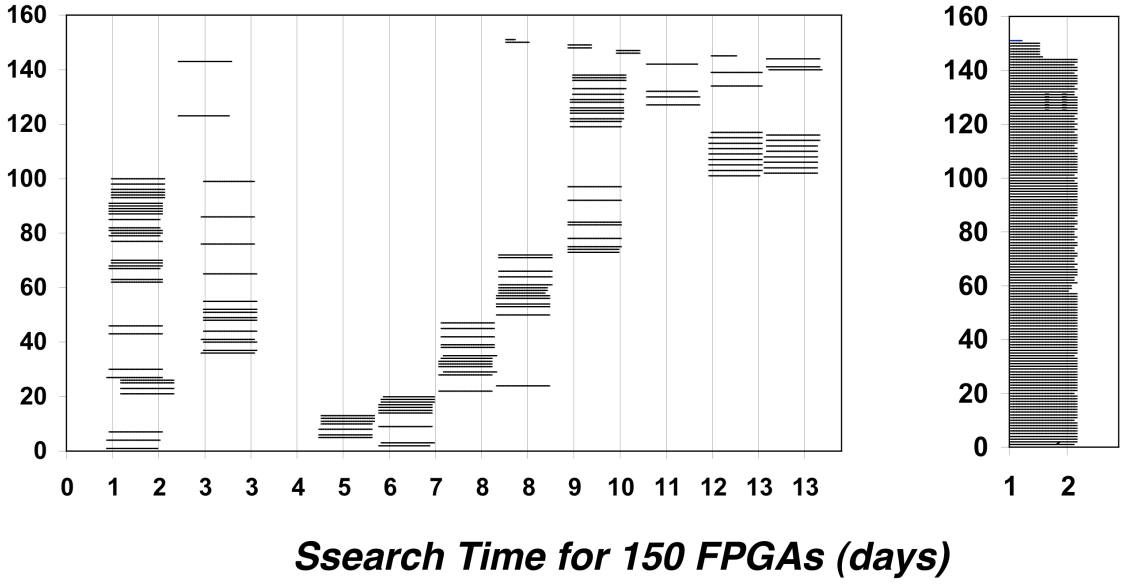

Figure 9. DNA Sequencing Time on 150 FPGAs: actual (left) and dedicated (right) time.

Fig. 9 shows Ssearch time along the x-axis for each of the 150 jobs submitted (shown vertically over a 2 week period. When run in a non-dedicated mode (over two weeks) each of the 150 jobs took approximately 1 day to complete except for six run on Virtex4s (top center) which took half a day. All 150 jobs could finish in 1 day (**Fig. 9 right**) if run in a dedicated mode on the 150 FPGAs.
The calculation speed unit often reported for genome comparisons is billions (Giga) cell updates/second, Or GCUPS. The Human-Mouse Genome comparison (**Fig. 9**) took 12.9 days to complete on 150 FPGAs, For a solution rate of 46 TCUPS, or 605 TCUPS for dedicated mode (**Fig. 10**).

❖ DNA Characters: **Human = 155 million, Mouse = 165 million**

Total Compares = 155M x 165M x 106^2 x 2 = $51x10^{15}$ Cell Updates

❖Sequential FPGA ==> 138 days (11,923,200 secs) **==> 4.3 TCUPS**
($51x10^{15}$/11,923,200)

❖Parallel (actual) ==> 12.9 days (1,114,560 secs) **==> 46 TCUPS**

❖Parallel (dedicated) ==> 1 day (86,400 secs) **==> 605 TCUPS**

Figure 10. Performance of Human-Mouse DNA comparison for Sequential & Parallel FPGAs

However, as the Opteron I/O is the bottleneck, causing the FPGA to halt computations, the athors recognized that replacing small I/O packets using "Fortran writes" with streamed I/O (**Fig. 11**) offers up to 10X speed improvement, based on previous experience with streamed I/O. This streamed I/O approach avoids sending many small write packets, (continually halting the FPGA) with large strings.

```
Change:   do 100 i=1,n
              write(6,110) x(i),y(i),z(i)
          100 continue
          110 format (1pe13.5, 1pe13.5, 1pe13.5)

To:           write(format_string,200) '(',n,'(1pe13.5,1pe13.5,1pe13.5\))'
          200 format (a1,i3,a20)
              write(6,201) (x(i),y(i),z(i),i=1,n)
          201 format (format_string)
```

Figure 11. Technique to string I/O to avoid slower I/O with small formatted data packets.

Alternatively, a similar speedup can be realized by writing the data to a large character buffer in parallel, and then transferring the buffer to disk using one binary write.nIn either case, achieving the additional 10X speedup via streamed IO to reflect performance speedup (**Fig. 12)** of 6088X (20 years on 1 Opteron to 1.2 days on 150 Virtex4 FPGAs) or 980X speedup over 150 Opterons.

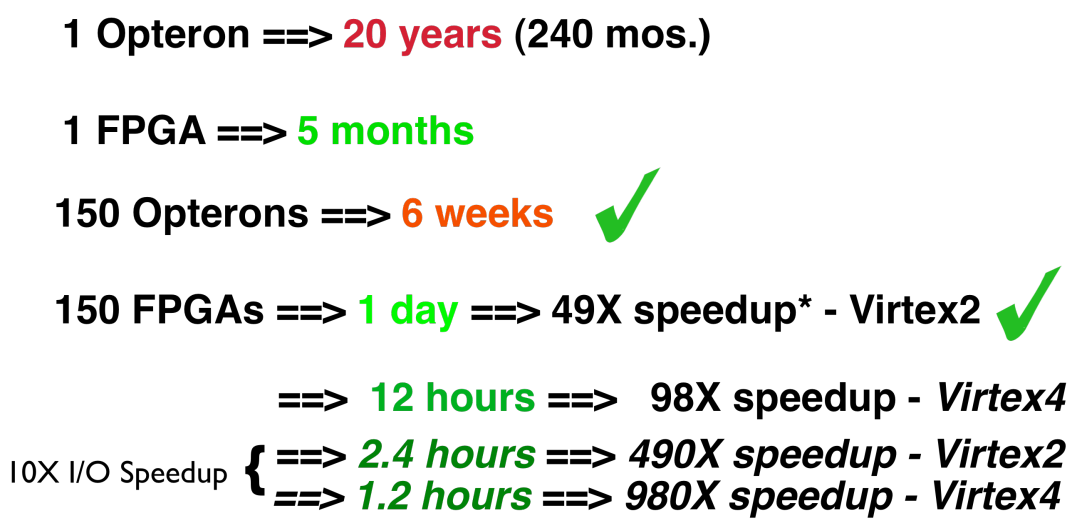

Figure 12. Effect of 10X I/O speedup in reducing Human-Mouse genome sequencing time.

6 Concluding Remarks

A description of three generations of FPGAs and their use as HPC accelerators is given together with results of their performance to speed Human-Mouse DNA genome sequencing on Cray XD1 computers. In addition to 100x speedup using Virtex4 FPGAs observed (over a 2.2GHz AMD Opteron), a technique to stream I/O by replacing N Fortran writes with one binary write illustrates how up to 10X further speedup is possible, based on previous experience with similar I/O methods on Cray computers. As HPC moves toward heterogeneous computers with accelerators, whether using GPUs or FPGAs, such streamed I/O techniques to achieve maximum performance should be of increasing importance.

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.Brodtkorb A, Dyken C, Hagen T, Hjelmervik J, and Storaasli, O., State-of-the-Art in Heterogeneous Computing, *Scientific Programming*, Ed. Ronald Perrott, IOS Press, Amsterdam (in review), 2009.
4.Strenski, Dave, FPGA Floating Point Performance, *HPCWire* - Jan 12 2007.
5.FASTA Sequence Comparison Program, University of Virginia
6.Margerm, Steve, Reconfigurable Computing in Real-World Applications, *FPGA Journal*, Feb 7 2006.
7.Storaasli O, Yu W, Strenski D & Maltby J, Performance Evaluation of FPGA-Based Biological Applications, Cray Users Group Proceedings, Seattle WA May 7-10, 2007. http://ft.ornl.gov/~olaf/pubs/CUG07Olaf17M07.pdf
8.Margerm, Steve, and Maltby, Jim; Accelerating the Smith-Waterman Algorithm on the Cray XD1 (*Cray White Paper* WP-0060406) 2006.
9.MitrionC/BLAST, http://www.hpcwire.com/hpc/1274236.html
10.Sternberg, M. (Ed.), *Protein Structure Prediction: A Practical Approach*, Chapter by Geoffrey Barton: Protein Sequence Alignment and Database Scanning, Oxford University Press ISBN 0199634963.
11.Smith T.F. and Waterman M.S. "Comparison of Biosequences." *Adv Applied Math.* 2: 482-489. (1981)
12.Smith T.F. and Waterman M.S. "Identification of common molecular subsequences." *J. Mol. Biol*. 147: 195-197 (1981)
13.Mount, D.W. In *Bioinformatics – Sequence and Genome Analysis*, Cold Spring Harbor Laboratory Press, New York (2001)
14.Pearson W.R. and Miller W., "Dynamic programming algorithms for biological sequence comparison." *Methods Enzymol*. 210: 575-601 (1992)

15. 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)
16. Guccione S.A. and Keller E. "Gene Matching using Jbits" in *Proc. 12th Field-Programmable Logic and Applications* Springer, Berlin (2002)
17. Yamaguchi Y. and Maruyama T. "High Speed Homology Search with FPGAs" in *Pacific Symposium on Biocomputing* 2002 pp. 271-282 (2002)

Acknowledgments

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. The authors also thank the US Naval Research Laboratory for access to the 150 FPGA Cray XD1.

About the authors

Olaf Storaasli joined the Future Technologies Group, Computer Science and Mathematics Division at Oak Ridge National Laboratory in 2005 as a Distinguished Research Scientist after his career as a Senior Research Scientist at NASA Langley Research Center. Olaf may be contacted at: 865-574-0494 (Office), 757-553-0333 (iPhone), Olaf@ornl.gov (Email) , http://ft.ornl.gov/~olaf (Web) or Postal Address: ORNL, PO Box 2008, MS-6173, 1 Bethel Valley Road, Oak Ridge TN, 37830-6173.

Dave Strenski has been an Applications Analyst at Cray since 1993. He has significant technical and practical knowledge and experience in both HPC software and hardware. Dave may be contacted at: 734-383-9077 (Office) or Stren@cray.com (Email). In addition to his HPC expertise, Dave has considerable knowledge and first-hand experience in solar power generation.