Smith-Waterman Local Sequence Alignment Application for Novo-G Platform

Carlo Pascoe, Sunny Gogar, and Matthew Gudwin, *Electrical Engineering, University of Florida*

Abstract – Bioinformatics is the application of information technology to the field of Biology. One use of Bioinformatics is the efficient comparison of DNA, RNA or protein sequences to determine structural or evolutionary similarities. A common method for quantifying these similarities is the Smith-Waterman algorithm which compares sequence segments of all possible lengths and optimizes their similarity measure. The Smith-Waterman algorithm is based on a dynamic programming algorithm and has shown substantial performance increases over purely software implementations when implemented on FPGA platforms such as the Cray XD1. Following the path of previous research, this paper chronicles a design created for University of Florida's Novo-G platform which performs the Smith-Waterman algorithm in a parallelized fashion to provide sizable speedup when compared to software or previous hardware implementations; observed speedups of up to 71 times when compared to the Cray XD1 and up to 702 times when compared to ssearch34 software implementation.

TABLE OF CONTENTS

I. INTRODUCTION 1
II. RELATED RESEARCH 2
III. IMPLEMENTATION APPROACH 3
IV. EXPERIMENT 5
V. RESULTS AND ANALYSIS 6
VI. CONCLUSIONS AND FUTURE WORK 7
VII. REFERENCES 8

I. INTRODUCTION

"In bioinformatics, a sequence alignment is a way of arranging the sequences of DNA, RNA, or protein to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between the sequences [7]." Sequence alignment techniques are used to help decode the Human Genome as well as identifying similarities in viruses and bacterium to help find cures for diseases and ailments. A popular method for performing local sequence alignment is the well-known Smith-Waterman algorithm. The Smith-Waterman algorithm is based on a dynamic programming algorithm that breaks down the process of sequence alignment into a set of simpler subproblems. Given database and query sequences, with cardinality m and n respectively, m*n scores are calculated from the max value of a set of deterministic functions with dependence on some of the previously calculated scores. These scores are used to populate a (m+1)x(n+1) scoring matrix (a scoring matrix example is included in **Figure 3**). The maximum score in the matrix represents the degree to which the two sequences match, allowing for quantitative comparisons between the calculated scores of different query sequences. The calculation of (m+1)x(n+1) scores leads to $O(mn)$ time and space complexity which is not so bad until you consider the exponential growth in the size of genetic databases over the past decade. Past solutions have been to run software implementations on faster and faster processors or to supplement the software with hardware accelerators that compute the m*n score values partially in parallel. There has been much research in the area of Smith-Waterman application acceleration, specifically in the area of FPGA acceleration. Continuing this research, a Smith-Waterman sequence alignment application, running on the Novo-G platform (**Figure 1)**, was designed based on the "Smith-Waterman Accelerator" Cray XD1 implementation [1, 2]. The requirements for implementing the algorithm are well defined and the real problem comes from discovering ways to calculate results more rapidly and efficiently.

Figure 1: Novo-G

s

Figure 2: ProcStarIII Board

The Novo-G platform, designed by the NSF Center for High-Performance Reconfigurable Computing (CHREC) at the University of Florida, is the most powerful reconfigurable supercomputer ever fielded in academia. Novo-G is a server cluster featuring 96 high-end StratixIIIE260 accelerator FPGAs from Altera housed in 24 quad-FPGA ProcStarIII boards (**Figure 2**) from GiDEL and mounted in SuperMicro servers connected with non-blocking DDR InfiniBand or 20Gb/s Gigabit Ethernet. More information about Novo-G can be found in [9].

II. RELATED RESEARCH

There are many implementations of the Smith Waterman algorithm on FPGA's. For research, multiple papers were found and read documenting others success. Of all the designs researched, the Cray XD1 approach showed the most promise. The XD1 design consists of a linear unidirectional systolic array of basic processing elements (PEs) responsible for the Smith-Waterman scoring matrix functionality. Each PE compares a query character to a stream of database characters along with other parameters to provide a single column of the score matrix based on the set of deterministic equations in **Figure 3**. The PEs are individually loaded with one query character per PE and the database is then streamed into the first PE, and shifted, one PE at a time, until the entire

$$S(y,x) = \max\begin{Bmatrix} 0, \\ S(y-1,x-1)+Sub(cy,cx), \\ S(y,x-1)-eog, \\ S(y-1,x)-eog, \\ H(y,x-1)-e, \\ V(y-1,x)-e \end{Bmatrix} \text{ for } 1 \le x \le s1, 1 \le y \le s2$$

where:

$$H(y,x) = \max\begin{Bmatrix} H(y,x-1)-e \\ S(y,x-1)-eog \end{Bmatrix} \text{ for a given } y$$

$$V(y,x) = \max\begin{Bmatrix} V(y-1,x)-e \\ S(y-1,x)-eog \end{Bmatrix} \text{ for a given } x$$

and:

$$S(0,x) = S(y,0) = 0$$

$$H(0) = V(0) = -eog$$

Figure 3: Smith-Waterman equations used by the Cray XD1

database had been processed. Using an array of PEs allows for the diagonals of the score matrix to be calculated simultaneously as indicated by the arrows in **Figure 4**. If the two characters (database and query) match, a specific score is given depending on previous scores. When two characters do not match, a "gap" is either opened, or extended in the sequence, leading to a reduced score.

A general Block diagram for the XD1 implementation on a single FPGA is found in **Figure 5.** The single FPGA design utilizes a feedback structure that allows for the extension of queries past the physical limit of the number of PEs that can fit on a particular FPGA (N). Since each column directly depends on the previous column, in order to extend the query length past N, the data from the N^{th} column must be buffered to be used as the initial conditions for the $(N+1)^{th}$ column, which is the first column for the second round of calculations. This buffering adds a significant amount of latency to the design, requiring nearly twice as long to process an (N+1) character query as would be required for an N character query. Removing this latency was a major design goal for the Novo-G implementation.

With this design, the Cray XD1 was able to produce a speedup of 10.15 times for a single XilinxXC2VP50 FPGA when compared to a purely software implementation (ssearch34) run on a single 2.2GHz AMD Opteron CPU [2]. The data used in this benchmark is a micro-RNA case; 3685 query sequences compared with all 24 human genome chromosomes, ranging from 50 to 240MBs of characters in length.

Other recent research in the area of Smith-Waterman acceleration is the effort of another group working on the Novo-G project, Team 3. Their approach is very similar,

	0	A	C	G	T	A	T	G	C
0									
A		2	0	0	0	2	0	0	0
C		0	4	2	1	0	1	0	2
G		0	2	6	4	3	2	3	1
A		2	1	4	5	6	4	3	2
A		2	1	3	3	7	5	4	3
C		2	4	2	2	5	6	4	6
C		0	2	3	1	4	4	5	6
C		0	2	1	2	3	3	3	7
T		0	0	1	3	2	5	3	5
T		0	0	0	3	2	4	4	4
G		0	0	2	1	2	2	6	4
C		0	2	0	1	0	1	4	8

Figure 4: The string of PEs allows the XD1 to process the scores in a diagonal fashion, maximizing data calculated per clock cycle.

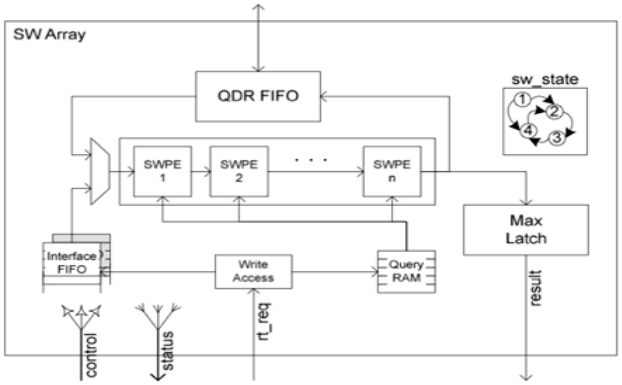

Figure 5: Simple Block Diagram of Cray XD1 Implementation

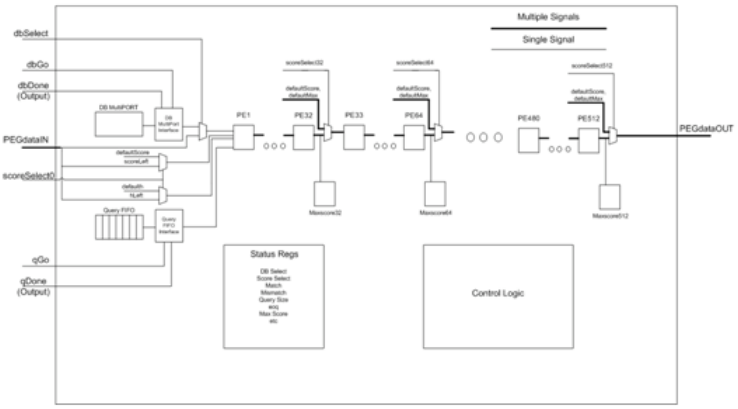

Figure 6: Simple Block Diagram of Novo-G Implementation

except they use Impulse C instead of VHDL to design the application. The benefit of using Impulse C is that it is a higher-level language than VHDL, the use of which should reduce development time, possible at the cost of performance.

As our effort was partially based on the design from Cray, Team 3 based their approach on a Smith-Waterman design from Impulse C. The basic method used is to store up to 32KB of query elements in processors, similar to a group of PEs, and stream the database through. Two independent streams are used to transfer data; the initialization data and score matrices are sent in one stream while the query and database data are sent in the other. Processing blocks consist of 32 concurrent hardware processors connected in cascade, of which the streams pass through. Each processor stores 1/32nd of the query sequence and computes 1/32nd of the columns in the scoring matrix as the database is streamed through them. Once the calculations for that query have finished, the next query is streamed in and the process repeats. This is a more course grained approach than our VHDL effort. Team 3 goes one step further by also implementing a traceback feature in software, which graphically shows how the query and database sequences match; this is not implemented in the VHDL design.

III. IMPLEMENTATION APPROACH

In this research the initial approach taken was to recycle as much code from the Cray XD1 Implementation discussed in [1] and [2] as possible while making Novo-G platform specific improvements (with the consent of Cray). After careful consideration and deliberation it was decided that the platform differences were so severe that any advantages of reusing Cray's existing code were outweighed by the disadvantages of modifying the code to run efficiently on Novo-G. With the Cray's implementation algorithm as an initial starting point, an efficient design was conceived to, at the very least, match the XD1 performance and in most situations out perform it with measured speedups as high as 71.

Figure 6 is a simple block diagram of the final Novo-G Smith-Waterman application hardware for a single FPGA. This Figure shows a general structure similar to Cray's implementation structure (**Figure 5**) with key differences introduced to improve performance. Similarities include the use of a large unidirectional linear systolic array of processing elements to perform the scoring functionality of the Smith-Waterman algorithm, the use of a large FIFO to stream in a sequence database one character at a time, the loading of each processing element with a single query character used to calculate a single column of the scoring matrix, etc… Key differences include the number of available PE's that can fit on a single FPGA, the elimination of a feedback loop used to extend a query past the number of available PE's in exchange for the ability to extend a query across multiple FPGAs, the method used to load each PE with the appropriate query character, the addition of simple hardware that allows for maximum PE utilization during each database stream by allowing additional queries to be loaded into unused PE's, etc… These will be discussed in more detail in the following paragraphs.

The Bulk of hardware resources are dedicated to the instantiation of a large systolic array of processing elements that when controlled correctly perform the Smith-Waterman algorithm. The PE's are the single most important element to both designs because they are responsible for calculating the needed scores. The PE's used for the Novo-G platform implement the same set of scoring equations used for the XD1 platform (**Figure 3**) but do so in a way which is more efficient for the Novo-G hardware. Due to the substantial difference in the number of available logic elements in the StratixIIIE260 compared to the XilinxXC2VP50, it is possible to fit more PE's per FPGA in the Novo-G implementation than the XD1 implementation; 512 PE's per FPGA on Novo-G as apposed to 128 PE's per FPGA on the XD1.

During program execution, each processing element contains a single query character to use for comparison against the entire database string that is streamed through each PE at a rate of one character per clock cycle. The XD1 implementation loads each PE with a query character by reading them from a single Query Ram that is routed to each PE and asserting the appropriate load signals at the correct time. In the Novo-G implementation, the query characters are popped off the Query FIFO and forwarded to the correct PE by shifting through the systolic array the correct number hops.

This is facilitated by the passing of a time-to-live value along side each character that is passed to the array; when a PE receives a character, the time-to-live is compared to zero and the PE either forwards the character with a decremented time-to-live to the next PE if the value is greater than zero or keeps the character for itself.

One limitation of the XD1 implementation is that if a short query less than the number of available PEs needs to be computed, the computing power of all unused PEs will be wasted. The Novo-G implementation improves upon this by adding customized register and mux hardware between each set of 32 PEs which allows for the start of a new query or the extension of a previous query; this is possible because the only thing that differentiate the start of a new query from the extension of a previous query is the passing of default score values (S(y,0) and H(0) from **Figure 3**) or previous PE score values to the next PE. By correctly orchestrating numerous select lines and the logging of an appropriate subset of available max scores (subset of all 16 reported max scores), it is possible to run multiple queries at the same time with the streaming of a single database. This allows the Novo-G implementation to process up to 16 times more queries in the same number of runs as the XD1 implementation under optimum conditions.

Databases are loaded into FPGA memory through the use of an asynchronous DMA transfer from the host program. When in memory, databases are streamed to the array of PE through the use of a sequential ProcMultiPort IP core provided with the ProcWizzard software; the multiport places sequential blocks of memory from a starting address into a simple FIFO which can be popped once per clock cycle after it has been filled with enough initial data. Using ProcMultiPort has a great advantage over using a FIFO only because it has the ability to be reset to the starting address, causing all data in the FIFO to be flushed and loaded with the data from the starting address. Using this allows a single database, loaded from the host only once, to be streamed against multiple queries; if this reset functionality were not possible (like with the MegaFIFO IP core) it would be necessary to transfer MBs of data (depending on the size of the database) to load the FIFO on every query run greatly increasing the execution time of the program. This partially explains the massive speedup observed when running the same case 1 benchmark (explained later) on Novo-G and XD1.

Since the characters compared will only consist of characters from the English alphabet, only 26 values need to be mapped to the characters meaning only 5bits are needed per character rather than the 8bits required for an ASCII character. One small benefit to removing these extra bits allows for the volume of transmitted data from the host to the FPGA to be greatly reduced. The greater benefit comes from the elimination of several signals per PE that would otherwise need to be routed which carry no useful data; removal of these useless signals allows more PEs to fit on a single FPGA. There are other situations where the number of characters can be reduced to as little as 4 (like with human chromosomes where the only characters needed are A, C, G, and T), requiring only 2 bits, and greatly increasing performance but it was decided that having the capability to run the general case was more important in this situation. A possible solution would be to have a separate application that only compares with a 4 characters alphabet rather than the 26 character alphabet, a solution that would be very easy to implement with minimal modifications to the current hardware.

There are interfaces between the array of PEs and each database and query memory module (MultiPort and MegaFIFO) that are responsible for retrieving needed character data at the correct times by interpreting command signals from the host and following a rigid protocol implemented with 20 state, state machines.

The XD1 design utilizes a feedback structure that allows for the extension of queries past the physical limit of the number of PEs that can fit on a single FPGA by buffering all of the last PEs score data to be used as the input to the first PE on the next run. The time required to utilize this feedback functionality is orders of magnitude greater than the time that would be required if there were enough PEs to process all query characters in parallel. This is a good solution for the XD1 because there is no way to increase the number of PEs a single FPGA has access to; this is not the case for Novo-G. Because each ProcStarIII board has the rather rare quality of possessing 4 large FPGAs that are connected with an “adjacent FPGA bus” that connects each FPGA to the FPGA’s directly adjacent to the left and right, it is possible to extend a query across multiple FPGAs effectively increasing the number of PEs available to a single FPGA if the situation requires it. This is facilitated by adding bus connections (PEGDataIn input from the left FPGA and PEGDataOut output to the right FPGA in **Figure 6**) with registers and muxing hardware in the same way similar hardware is added between every 32PEs hat selects between continuing a query across multiple FPGAs or starting a new set of queries. Extending the query across multiple FPGAs implies the streaming of the same database across multiple FPGAs and the ability to mux between continuing a database or starting a new database.

Because all of this hardware is the same regardless of which FPGA used it is possible to make a single FPGA design and load it on all ICs, the manipulation of control signals being the only thing that differentiates each FPGA. There are several possibilities of how to control each FPGA individually and how to control a group of FPGAs to work collaboratively. Each FPGA can be loaded with its own database file or a subset can be loaded with database files while the other FPGAs continue the database from the previous FPGA; the only restriction is that the first FPGA on a board must be loaded with a database file (this leads to the possibility of 8 different database configurations on a board). The most efficient configuration of database files depends on the number of databases as well as the batch of query strings that need processing. For example, if all query strings are less than 512 characters and there are 4 different database files then it would be beneficial to schedule each FPGA with its own database and a local copy of the batch of query strings so all 4 databases are processed at the same time; this situation changes if there is even one query string greater than 512 characters causing the need for multiple FPGAs and making it more efficient to divide the batch of query strings across the same number of FPGAs. Another example would be if there were only 2

database files in the example above rather than 4; with only 2 database files it would be best to divide the query and database files across 2 FPGAs which would utilize all of the available hardware and finish close to twice as fast as if the database files were only scheduled on 2 FPGAs leaving the remaining 2 FPGAs on the board inactive. Another example would of course be if there were only single database files; the best course of action would be to extend the database and divide the query file across all 4 FPGAs.

With this many scheduling options, the best performance will be observed by analyzing the input files and writing custom code to schedule resources. With the knowledge that this is not always desirable, an attempt was made to design code that analyzes the query file and number of database files to make the following scheduling decisions:

- If there are any queries longer than 2048, those queries are removed from the list and a note is placed in the output file saying that the query is too long to process.
- If there are any queries longer than 1024 and less than 2048, the queries are divided amongst 4 FPGAs and only 1 database is scheduled per board.
- If there are queries longer than 512 and less than 1024, the queries are divided amongst 2 FPGAs and 2 databases are scheduled per board (IC1&2 and IC3&4).
- If all queries are less than 512 then the number of database files begins to effect scheduling:
    - A single database file leads to the query file divided across all 4 FPGAs.
    - A multiple of 2 but not a multiple of 4 database files leads to the query file divided across 2 FPGAs and 2 database files per board.
    - All other situations (odd number or a multiple of 4) cause the databases and queries to be scheduled to a single FPGA.

It is easy to see how this algorithm is modified to schedule on multiple boards. Any scheduling that would cause multiple rounds of database loading from host to FPGA can be broken down into a single load to multiple boards. The only thing that distinguishes one board from another would be the names of the files it must load. Because of this it is easy to modify the single board code to run on multiple boards by adding MPI functionality to pass the names of the scheduled files to the appropriate ProcStarIII board.

IV. EXPERIMENT

The experiments performed were separated into two separate categories; experiments which test functionality and experiments which test performance. While designing the application, the functionality was tested in incremental steps to aid in debugging. The performance tests were used in benchmarking to provide an absolute processing time and also to providing a direct comparison to a purely software implementation (FASTA ssearch34), the Cray XD1 implementation, and Team 3's implementations.

The functionality tests performed were as follows, in chronological order:

- Redesigned PEs tested to verify proper operation of all scoring calculations and shifting properties.
- Small group of PEs were coupled to validate proper PE to PE communication, specifically score passing and proper shifting.
- Query character loading and database streaming functionality tested for a group of PEs.
- Gidel's simple FIFO and MultiPort tested for expected operation and interface requirements.
- Database interface and query interface tested with simple FIFO and MultiPort for proper communication and serializing of parallel inputs.
- Memory and interface components connected to a group of 32 PEs, were tested with up to 32 character queries and relatively small database streams for correct scoring and general operation.
- Number of PEs increased to 512 (max PEs per FPGA)
- Mid array mux functionality added and tested for correct function with up to 16 queries per FPGA.
- Size of database stream increased up to 32MB, amount allowed by Single DMA transfer.
- Simple test performed to verify expected inter-FPGA communication; a large shift register spanning all 4 FPGAs on a single ProcStarIII board.
- Smith-Waterman design instantiated on multiple FPGAs (up to 4) verifying query extension across FPGA interconnects.
- Above design tested for varying lengths of queries, up to 2048, and a single streaming database.
- Above design tested for independent FPGA operation using individual databases and different queries of lengths up to 512 characters.
- Multiple instantiations of board design tested using MPI to pass a list databases and queries files to each board.

All of the above tests were performed solely for validating the design and debugging. The importance of these tests for debugging purposes was critical. There were several deviations from expected operation discovered during this process. One example is a problem discovered with design of the PEs. After several hundred test query runs it was noticed that one of the reported scores was 1 less than the actual score. It was later discovered that this was because one 'and' gate in the PE vhdl file was supposed to be an 'or' gate. Without performing these tests several times, the error would never have been discovered. There are were many other situations similar to this that have allowed us to say with great confidence that most if not all of the errors in the hardware design have been discovered and fixed; of coarse the design is error free only until the next error is discovered.

With a properly operating and verified design, benchmarking was performed to quantify performance. Since the Novo-G implementation was based on the Cray XD1 implementation, comparison to the XD1 is natural. The same case 1 benchmark used in [2] was used to compare the designs. The case 1 benchmark consists of 3685 queries, all less than 128 characters long (this means that the feedback functionality of the XD1 will never be used), and all 24 human genome sequences ranging from 50 to 240MB of characters, 2.9GB in its entirety. Since the XD1 implementation does not require

the use of its feedback functionality for this benchmark, XD1 application is running under favorable conditions. A less favorable benchmark for the XD1 implementation but a more favorable benchmark for the Novo-G application would be a long list of queries ranging in size from 128 to 512 characters; this benchmark would cause the XD1 application to use its feedback system and still allow for multiple scheduling of queries on the Novo-G application. Since timing data for such a benchmark is currently unavailable for the XD1, no such comparisons will be made.

While the above test illustrates how well the Novo-G application accelerates versus another FPGA-based Smith-Waterman hardware accelerator, comparison to a purely software implementation is also valuable. For this comparison, the software application used for comparison by Cray in the XD1 benchmarking was obtained and the same sets of tests were run. The benchmarking Cray used was the FASTA ssearch34 application on a single 2.2GHz AMD Opteron processor. As it was expected that the Novo-G implementation would show speedup over the XD1 implementation and the XD1 implementation showed speedup over the software implementation, it is only natural to expect the Novo-G implementation to outperform the software as well. A more fair comparison to a software implementation would be to compare the FPGA application to a parallel implementation of the software; since a parallel implementation of the software is currently unavailable, no such comparisons will be made.

V. RESULTS AND ANALYSIS

Several tests were performed to analyze the performance of the Smith-Waterman application on Novo-G. Table 1 shows the execution times and speedups for the case 1 benchmark discussed in [2] with a 2.2GHz AMD Opteron processor, the XD1, and Novo-G. The case 1 benchmark required 3685 query sequences searching across all 24 human genome chromosomes but certain timing issues caused the benchmark to be changed; because running the ssearch34 software implementation against just the first chromosome (a 250MB database sequence) took 75 hours, the benchmark was modified to just use the first chromosome as a test. The benchmark was run for 1, 2, 3, and 4 FPGAs.

Though accurate, technically the times shown in Table 1 for the Novo-G implementation are just simulated because there is an issue with the correct DMA transfer of data greater than 32MBs. Under Linux, when trying to transfer a contiguous block of host memory greater than 32MBs to FPGA memory on a ProcStarIII board, multiple DMA transfers are required [10]. As of the time of this paper, multiple DMA transfers are possible but the data does not arrive as a contiguous chunk on the ProcStarIII board (an email has been sent to Gidel on the issue but a response has not been received at the time of this paper). The Novo-G Table 1 times were calculated by running the Smith-Waterman application with multiple DMA transfers under the assumption that the database file is transferred correctly even though it is not. This approach will ultimately produce incorrect scores

	Number of (CPUs/FPGAs)			
	1	2	3	4
AMD Opteron	75	x	x	x
XD1	7.39	3.75	2.48	1.91
Novo-G	0.1069	0.0528	0.0352	0.0274
Speedup vs. CPU	701.59	x	x	x
Speedup vs. XD1	69.13	71.02	70.45	69.70

Table 1: Hours to Perform Case 1 Benchmark. Times Provided for Novo-G are Average/Simulated Values.

but will execute in the same amount of time it would take to calculate the correct answers. The times shown are averaged execution times of the benchmark run several times.

The table clearly shows significant speedups over both the software and XD1 implementations. The objective of our analysis is to identify and quantify the factors that contribute to the speedup achieved on the Novo-G implementation over the other tested implementations.

It is easy to see why the Novo-G implementation beats the software implementation when you consider the ssearch34 code profile from [2] provided in **Figure 7**. The figure shows 98.61% of the entire program execution is spent in the function FLOCAL_ALIGN() which is the function responsible for calculating the score matrix of the Smith-Waterman local sequence alignment algorithm. This function directly maps to the portion of the algorithm that is executed on FPGAs in the Novo-G implementation meaning that this function has the benefit FPGA acceleration when comparing the implementations. Since this function requires tens of clock cycles to calculate a single score in software and each FPGA can calculate 512 scores per clock cycle, even at a clock rate difference of 2.2GHz to 125MHz (approximately 17 times faster), the software function can not compete.

The speedup obtained against the XD1 implementation is a little harder to quantify; some of the speedup can be explained by the intended improvements in the Novo-G implementations design while some has an unknown origin. The first source of speedup comes from the physical size difference between the Altera StratixIIIE260 and the Xilinx XC2VP50; because the StratixIIIE260 is larger, 512 PEs can fit on a single FPGA rather than the 128 PEs in the XD1 implementation. Another source of speedup comes from the optimization that allows multiple queries to be processed in a single run, allowing up to 16 times more queries to be processed in the same amount of runs as the XD1 implementation. Yet a third source comes from the clock rates

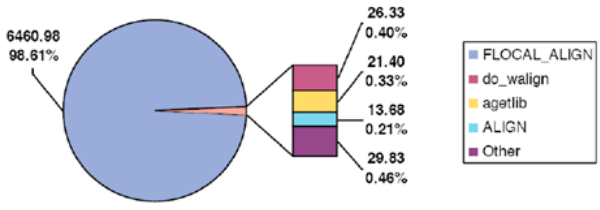

Figure 7: Code Profile of ssearch34 Application

of the two designs; the XD1 implementation runs at 200MHz but needs 2 clock cycles to calculate a score while the Novo-G implementation runs at 125MHz and only needs 1 clock cycle to calculate a score. These sources explain some of the speedup but do not explain the observed 70 speedup for a single FPGA.

Where does the rest of the speedup come from? The exact source is unknown. An email was sent to Dave Strenski at Cray to determine possible sources but at the time of this paper a reply has not been received. In this email a few suggestions were offered such as the time required to stream a database (it is unknown if the XD1 implementation must stream the database into the FPGA every run or if it strams the database from FPGA memory like the Novo-G implementation) or the time to stream a query (XD1 uses a query ram to load PEs but Novo-G streams query through PEs from a FIFO). When a reply is received a more accurate analysis can be made.

In order to produce unsimulated execution times, test performance of an MPI implementation, and produce actual scores that can be checked for accuracy, an additional test was conceived. 96 database files of about 32MB each were run with the 3685 query sequences from case 1. Table 2 shows the execution times for this test case run on 1, 2, 4, 8, and 16 FPGAs. As of 8/5/09 there are still issues with MPI working on Novo-G. Tests on this date determined that the scheduler is not working properly; blocks of 8 processes are scheduled to the same node resulting in 7 processes not completing because there is only one ProcStarIII board available per node (Rafael Garcia knows about the issue and is looking into it). We were able to trick the scheduler into running on two boards by setting the number of threads to 9 but this of course left 7/9th of the required work undone. When this issue is fixed the experiment can be completed.

Running this test allows for the reporting of real timing data rather than the simulated times reported in Table 1; since each database is less than 32MB, a single DMA transfer is all that is needed, allowing for the database to be placed into FPGA memory in a contiguous chunk. With a contiguous database the application can be run correctly and should produce correct scores. As expected the output files were checked for accuracy and scores were reported correctly.

One final experiment to test the performance of our Novo-G implementation was to compare performance against Team 3's Impulse C implementation. The experiment consisted of one 512 length query ran against a single 16MB database; 16MB because both designs are limited to database lengths of 32MB due to the previously discussed DMA issue. The results of this run were 0.5 seconds of useful work (total execution time minus constructor time) versus Team 3's 12 seconds; a 24 times speedup versus Team 3's. Each design has tradeoffs. Our VHDL implementation may outperform Team 3's implementation in this situation, but if the query length exceeds 2048 characters, Team 3 would dominate because, unlike the VHDL implementation, the Impulse C implementation can handle up to 32k length queries. On the other hand, if multiple queries were to be ran against the single database, our design would show an even greater speedup due to the efficient database reset functionality, not implemented in Team 3's design.

MPI Execution Times	
Boards	Time (seconds)
1	4329
2	2330
4	1094
8	x
16	x

Table 2: Execution Times of MPI implementation

VI. CONCLUSIONS AND FUTURE WORK

The effort of this project produced a product with respectable increases in speed and flexibility over previous implementations. The data in the results section illustrates Novo-G's power, and the significant speed up which can be achieved by parallelizing the Smith-Waterman algorithm on such large FPGAs. The Novo-G implementation was partially based on the Cray XD1 implementation and without the efforts of Cray, would not be as successful.

The Novo-G platform is very expensive, but provides more computing power than any other reconfigurable platform in the known world. Since the Nov-G platform is a cutting-edge technology and still being developed, there are understandably a few issues. The MPI, DMA, driver, etc… issues will undoubtedly be resolved with future research.

By extending development on this project, it could be improved in many ways; more optimization to produce faster run times, variable scoring and traceback functionality, query length extension, etc... Given the time available to develop this application, the results are very respectable.

One of the main sources of slowdown results from the software manipulation of the data prior to the FPGA processing it. Experiments have shown that the time required for manipulation does not have a linear relationship with the length of the data; the time required to manipulate 3685 queries was nearly ten seconds whereas the time required to manipulate 14740 queries was nearly ninety seconds. One possible solution is to assume that all input data would be received in a specified format requiring no manipulation. This solution would essentially eliminate this required time in the Smith-Waterman application, adding to the achieved speedups.

The comparison of each character is currently performed as a binary operation, ie only two possible outcomes, match or mismatch. The added functionality of a match matrix would provide more flexibility. The match matrix is a matrix of the match scores between all possible combinations of the characters used in both the query and the database; though no mismatches are desirable, some character mismatches are less desirable than others. Implementing this functionality would allow the user to specify what score each character comparison produces.

Another feature, which can be added, is the traceback feature; traceback pieces the matching characters from the query and database together for a graphical representation of

the matching. Traceback can be performed in software, which can be implemented in this application, but there is the possibility of hardware acceleration for the traceback. Adding this functionality would drastically decrease logic resources available for the PEs, ultimately reducing the maximum length of processable quires.

Even though the designed application produces significant speedup numbers, it can only process queries of lengths up to 2048 characters. In future development, this limit could be significantly increased either by implementing a feedback system similar to that of Cray's XD1, or by extending the query beyond a single ProcStarIII board. Neither approach is straightforward but each adds flexibility. It is also possible to combine the software implementation with the hardware implementation, allowing all queries greater than 2048 to be handled in software while the others are handled in hardware; with the quad core Xeon E5520 processors available to each board it is perfectly plausible to run the hardware on a single thread and run multiple software threads processing queries larger than 2048.

VII. REFERENCES

[1] Margerm, S. (n.d.). Cray XD1 Smith Waterman Accelerator (SWA) FPGA Design.

[2] Strasli, O., Weikuan, Y., Strenki, D., & Malby, J. (May 2007). *Performance Evaluations of FPGA-Based Biological Applications.* Seattle WA: Cray Users Group Preceedings.

[3] H.Y, L., M.L, Y., & Y, C. (2004). A parallel implementation of Smith Waterman algorithm for massive sequences searching. *26th Annual International Conference of the Engineering in Medicine Biology Society* (EMBC., vol 2), 2817-2820.

[4] Waterman, T., & Smith, M. (1982). Identification of Common Molecular Subsequences. *Journal of Melecular Biology* , 147(1), 195-197.

[5] Yamaguchi, Y., & Maruyama, T. (2002). High Speed Homology Search with FPGAs. *IPSJ Transactions on High Performance Computing Systems, 43*, 196-205.

[6] Zang, F., Xiang-Zhen, Q., & Zhi-Yong, L. (2002). A Parallel Smith-Waterman Algorithm Based on Divide and Conquer. *IEEE Computer Society* .

[7] *BioInformatics*. Retrieved from Wikipedia: http://en.wikipedia.org/wiki/Bioinformatics

[8] *Sequence Alignment.* Retrieved from Wikipedia: http://en.wikipedia.org/wiki/Sequence_alignment

[9] *Novo-G Info.* Retrieved from Novo-G Wiki: https*://novog-wiki.hcs.ufl.edu*

[10] *ProcstarIII Data Book Version 1.* Retrieved from Novo-G Wiki: https*://novog-wiki.hcs.ufl.edu*