Spiraling in on Speed-Ups of Genetic Algorithm Solvers for Coupled Non-Linear ODE System Parameterization and DNA Code Word Library Synthesis

Dan Burns[1], Kevin May[1,2], Thomas Renz1, and Virginia Ross[1]

[1] Air Force Research Laboratory
Information Technology Division
burnsd, renzt, rossv @rl.af.mil
315-330-2335, -3423, -4384

[2] Clarkson University
maykn@clarkson.edu

Abstract:
This paper reports interim results and lessons learned while pursuing speed-ups of a Genetic Algorithm (GA) solver for two hard optimization test case problems, Non-Linear Ordinary Differential Equation (ODE) Parameterization, and DNA Code Word Library Generation. We obtained speed-ups by choice of language on a single PC platform (100-1000x speed-up), transition from C on a PC to C/MPI on a cluster (30x speed-up), and transition from C on a PC to VHDL for FPGA (500x speed-up). Taken together, these and further speed-ups using a cluster with FPGA's enable solutions in minutes vs. months. We hand crafted VHDL code, and also evaluated the Impulse Accelerated Technologies Co_Developer tool for one test case. We encountered difficulties in making an "automated" transition from C to VHDL related to tool maturity, lack of support for double precision floating point math, and MPI communication among FPGA's. We also report the design of a hardware FPGA core, and a systolic array that accelerates the calculation of the Levenstein Matrix, which dominates the solution time of one of our test cases, by 500x compared to software.

Background:

One currently open question is whether Evolutionary Computing (EC) methods such as the Genetic Algorithm offer advantages over more classical methods for solving hard optimization problems, especially in the context of parallel and hardware implementations aimed at achieving extreme solution time speed-ups and problem size scaling. We hypothesize that the GA may indeed offer a speed advantage when implemented in hardware because of its relatively simple random number generation, cut, paste, and bit flip operations.

EC algorithms are being applied by an increasing number of researchers to hard, NP-complete combinatorial optimization problems in a number of diverse problem domains, e.g. dynamic reconfiguration of networks hosted with constrained resources, optimization of distributed database architectures and operations, assignment of routing & loading of vehicles, composing and evaluating adversary courses of action (Bayesian Belief Networks), generation of adaptable filters for hyper-spectral imaging and compression, and biological process modeling to support development of bio-hazard sensors and bio-molecular computing paradigms. We chose two test case problems that are of current interest to workers at AFLR/IF to study the speed-up advantage question.

The first test case evaluated the performance of a GA guided method that composes and refines the parameters in sets of coupled, non-linear ODE's to match both simulated and experimental data. We used a biological process ODE model associated with bio-hazard sensing[1].

Both classical[2] and GA methods[3] for solving this problem involved simulating and comparing a large number of double precision ODE data sets for candidate parameterizations, and this motivates attempts at extreme speed-ups.

The second test case evaluated the performance of a GA guided method for designing highly constrained sets of DNA Code Words[4]. This problem involves composing large libraries of pairs of short DNA strands that hybridize perfectly within each pair, but poorly if strands are taken from different pairs in the library. Such libraries are useful for applications in micro-arrays, bio-molecular data storage and computing, and DNA assisted schemes for nano-device self-assembly. The problem involves extensive constraint checking of strands in new candidate code word pairs with strands in existing pairs in library. The primary constraint metric is the string edit distance, measured by calculating the Levenshtein matrix. This calculation consumes about 98% of the application's compute time. Exhaustive search is impractical for this problem because of the size of the search space, and there are no known deterministic algorithms for composing optimum libraries.

Although much work has been done on hardware GAs[4-10], we found no turn-key FPGA cores for GA in VHDL for a single chip solution, so we developed one.

Speed-up by choice of language on single PC platform:

We implemented a simple GA that uses uniform random crossover, a low level of single bit mutations, and elitism, and applied it to the first test case problem, Non-Linear ODE Parameterization. We developed versions in three languages, LabView, MatLab, and C, all for the single PC platform, and we observed between 100-1000x speed-ups by moving to C, as shown in Figure 1. The ODE model consists of a number of equations that contain parameters that must be adjusted to make model data match known data. The fitness function that guides the GA in the fitting process is the normalized sum of squared differences between data predicted with the model and known data. This model data is double precision floating point, and the model parameters are floating point numbers that are scaled to integers over various ranges for manipulation by the GA.

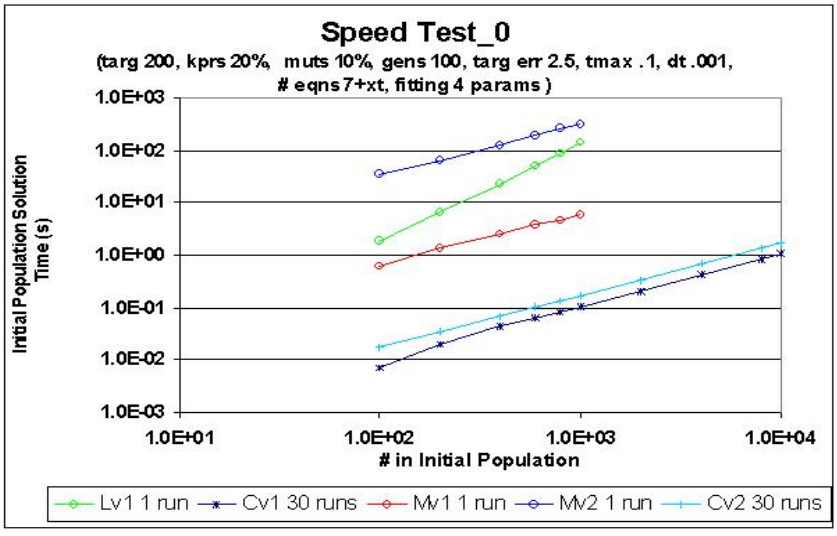

Figure 1. Speed Test_0, solution time of the initial population as a function of the number of individuals in the initial population for different language implementations.

It is interesting to note that the classical methods for solving this problem are not time competitive with the GA method. Generally, they involve reducing the order of the model (an effort may take days), and then using a sequence of steps that do partial parameter fits[2]. The GA converges using the full, unreduced model, and also with sparse or noisy data.

Speed-up on cluster platform:

We implemented a distributed Island Mode GA to solve the Non-Linear ODE Parameterization Problem. It was written in C, with MPI communication between processor nodes. We observed approximately linear speed-up up to 25 processors, as shown in Figure 2. The distributed GA uses a group of processors connected in a ring topology, and each processor breeds a separate population, or "deme" of individuals. However, after every epoch of a certain number of generations, each processor passes a few of its most fit individuals to the processor after it in the ring, and receives a few from the processor before it in the ring. Each processor runs an identical program, although the 0^{th} processor is the master and does certain bookkeeping tasks. Communication between the processors takes place only at the end of epochs when individuals are passed, and when a solution has been found and the run terminates.

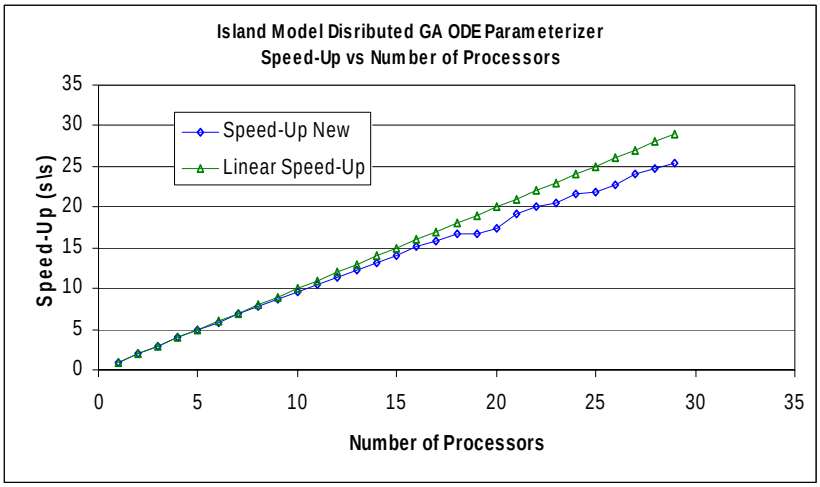

Figure 2. Speed-up curve for distributed Island Model GA ODE Parameterizer.

We also applied the distributed Island model GA to the DNA Code Word Library test case problem, with similar results, as shown in Figure 3. In this problem, a DNA strand is represented as an array of genes assigned to be an integer number representing A, C, G, or T base pair types. The GA starts with an empty library, and breeds new pairs of words that meet certain constraints that relate to the quality of binding between the strands in the new pair and the strands in all other pairs in the library. As word pairs are added to the library, this becomes more difficult. The fitness function that guides the GA is based on a set of measurements of the string edit distance between strands, which is calculated by the Levenstein matrix.

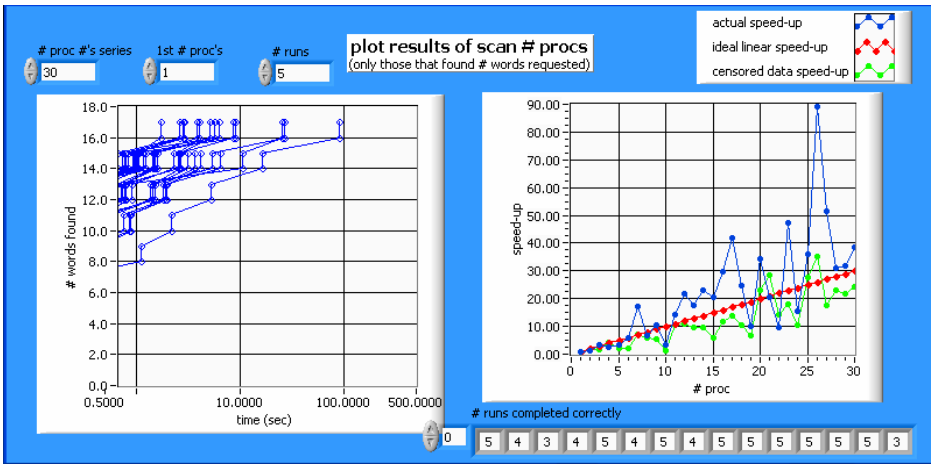

Figure 3. Multiple run data analysis tool front panel display. Data shown indicates near linear speed-up.

In this problem, communication occurs at the end of each generation to check whether any processor has found new words, at the end of epochs when good individuals are passed around the ring, and at the end of a run if a termination criteria has been met (time, # generations, # words found).

We tested the speed of this algorithm against the best known approaches found in the literature[11], and it was faster, as shown in Figure 4.

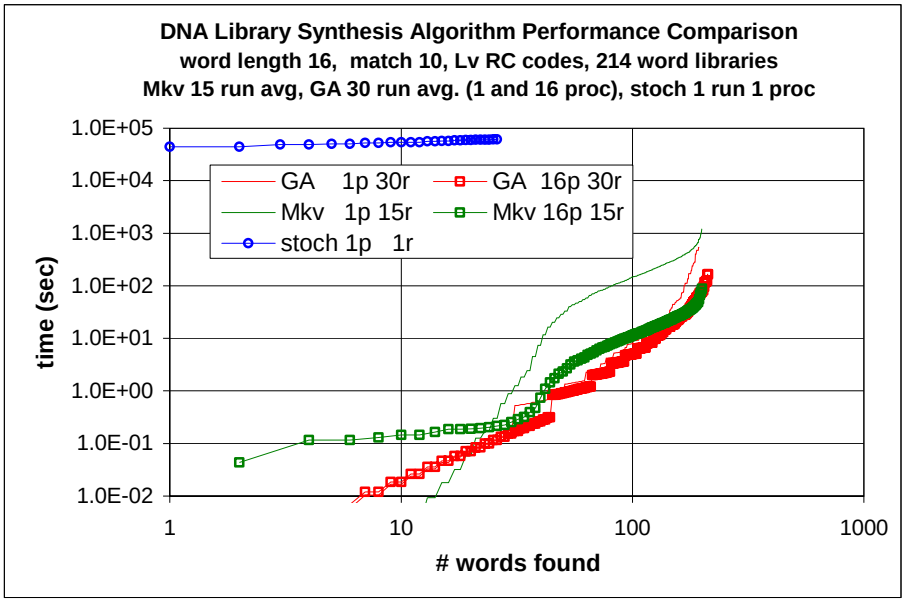

Figure 4. Comparison of Markov, GA, and stochastic DNA Code Word Library Generation methods. GA (red) finds words generally faster than Markov (green) for both 1 and 16 processor cases, although Markov eventually found more words for 1 processor case. Stochastic (blue) is very slow.

Speed-Up on FPGA platform:

We did not pursue hardware speed-up for the ODE Parameterization problem because of its heavy use of double precision floating point calculations, and the perception that floating point problems are not appropriate for FPGA hardware acceleration, although fixed point methods[12,13] might be applicable. In the present work we chose to focus on the second problem.

We used GNU gprof to profile the total time for subroutines in the second problem, the distributed GA DNA Code Word Library Generation Problem, as shown in Table I. The do_matrix subroutine dominates the time, and it calculates the Levenshtein matrix. This is an all integer calculation, and is well suited to hardware acceleration.

Table I. Time profiling of GA/DNA Code Word Library Generation application shows time consumed by subroutine (produced with GNU gprof).

%	Cum	self	self	total		
Time	sec	sec	calls	s/call	s/call	name
98.13	46.77	46.77	13115396	0.00	0.00	do_matrix_v6
0.65	47.08	0.31	6603415	0.00	0.00	i2s
0.44	47.29	0.21	90836	0.00	0.00	do_checks
0.38	47.47	0.18	3	0.06	0.07	clean_up_pop
0.23	47.58	0.11	272685	0.00	0.00	s2i
0.08	47.62	0.04	90895	0.00	0.00	compliment_x_str
0.06	47.65	0.03	91835	0.00	0.00	are_you_in_there
0.02	47.66	0.01	93753	0.00	0.00	pop_to_word
0.00	47.66	0.00	90895	0.00	0.00	compliment_reverse_x_str
0.00	47.66	0.00	960	0.00	0.02	smart_flip_2
0.00	47.66	0.00	63	0.00	0.50	find_fitnesses

Figure 5 shows the basic Levenstein Matrix calculation that is the main work of the do_matrix_v6 subroutine that dominates the calculation time. Word strands are presented along the top and left edges of the matrix. Each cell along the top row (and left column) is initialized to 0, and then set to 1 if the bases at the top and left edges aligned with that cell are the same, or if the cell to the left (or top) is already a 1. In the inner cells, the cells are initialized to 0, and are set to the maximum of 3 values, the value of the upper, left, and upper left adjacent cells, except that 1 is added to the upper left adjacent cell value if the edge bases aligned to the cell are the same.

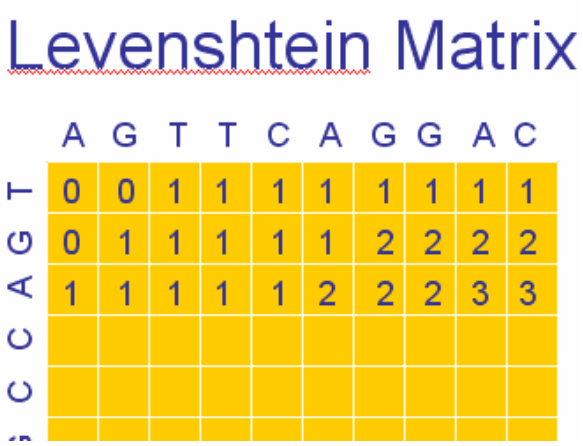

Figure 5. Levenstein Matrix Calculation

We developed two versions of synthesizable VHDL Levenstein matrix hardware accelerators, a ripple through version, and a time multiplexed systolic array[14] version. The ripple through version was a good first step, but was slow (10MHz). Only the systolic array version of this array will be discussed here.

The tool path we used includes the ModelSim Xilinx Edition II design environment from Mentor Graphics, the ISE 6.1i Service Pack3 synthesis tool from Xilinx, and we targeted the chip used in the boards in our cluster with hardware, the XC2V6000 from Xilinx.

Figure 6 shows the systolic array version, which consists of an array of cells with register arrays at the top and left edges. The resister arrays sequence word pairs into the edge cells. It also shows a breakout of one cell in the format of an entity defined in VHDL. The U, L, and UL inputs are simple signals carried in on wires, but the A and B inputs are signals latched into registers in the cell. The ans output is also latched in the cell by a register. Actually, the output of the A and B registers have outputs as well that connect to the A and B inputs of adjacent cells below and to the right, respectively.

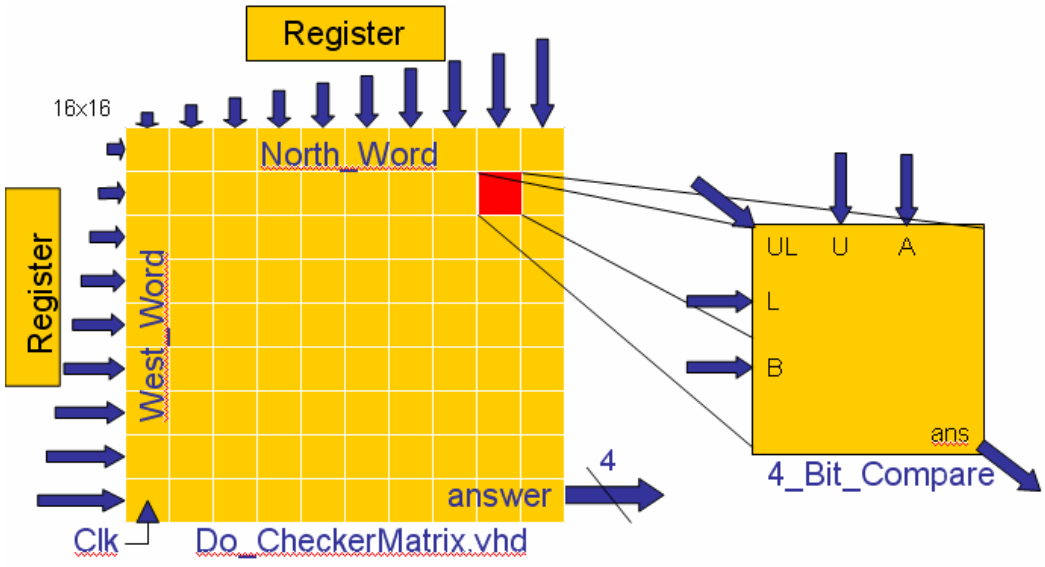

Figure 6. Levenstein matrix calculation array implementation.

The systolic array computes 16 word pair comparisons simultaneously, with calculations flowing along diagonal lines down through the array from upper left to lower right. The shift registers at the edges of the array delay the presentation of the upper bases of the words to the edge cells as needed to synchronize with the waves of calculations. The base pair tokens are shifted down columns and across rows inside the array. Also, the matrix is operated in a checkerboard fashion, with one half of the cells loading inputs on any clock, and with the other half of the cells calculating outputs. On the following clock, the opposite happens. The "latency" of the array is 16 words, i.e. the answer for the first word pair flow out the bottom right corner after the 16th word pair is processed. After that, answers flow out of the lower right cell every two clock cycles.

Figure 7 shows a higher level functional block diagram that includes entities for on-chip SelectBlock memories to hold the GA population, the fitness values of the population, and the words of the DNA Library. It also shows a hardware entity called MemBlock.vhd that sequences

the fitness evaluation of all the individuals in the population against all the words in the library, and stores the results A simple software test bench is shown in place of a GA for testing purposes at this point. The design of Figure 7 has been synthesized, and software simulations show that it operates correctly.

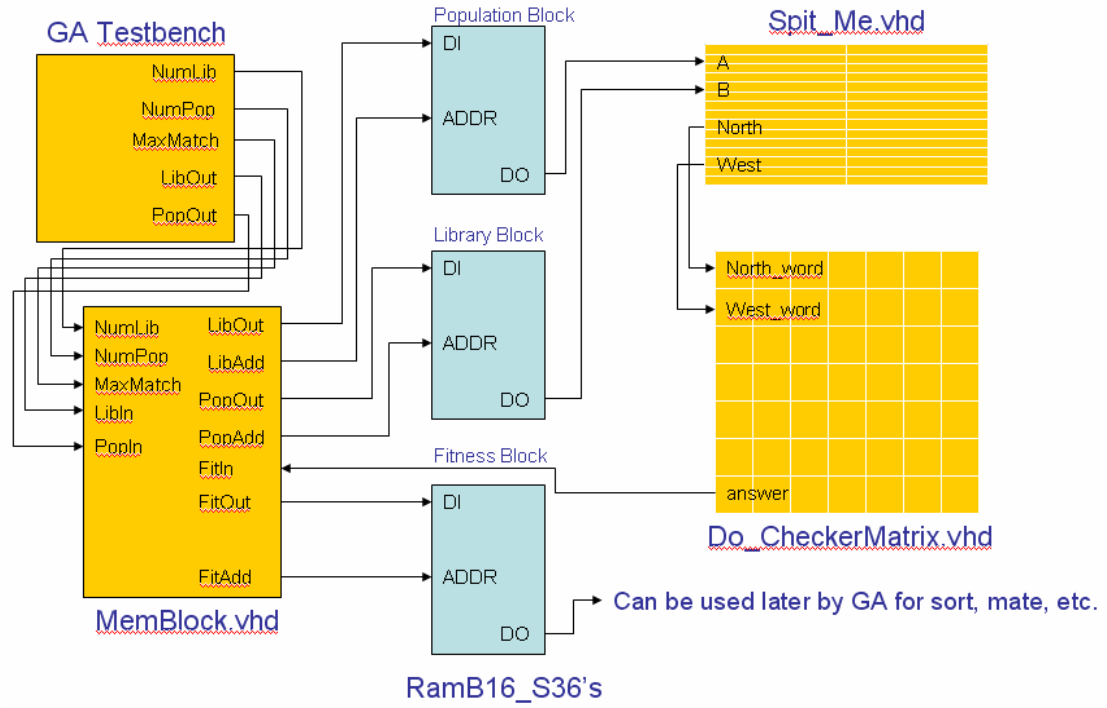

Figure 7. Upper level functional block diagram of fitness evaluator.

Table II. shows the synthesis run report with resource utilization and expected speed. This synthesis was for the case of 512 member population and word libraries, which are adequate for the runs we have made thus far. It is clear that we could use much larger populations and libraries, since only 3 of the 144 SelectBlock RAMS were used for this case. Over 80% of the chip resources are available for the GA and DNA Code Word Library application. This is encouraging, and it might even be possible to support multiple fitness evaluators, or multiple populations, on one FPGA chip. We are also close to our target speed of 100MHz, which represents a 1000x speed-up over the software version (10us per word pair).

Table II. Synthesis report showing resource utilization and expected speed. Minimum period: 12.37ns (80.8MHz)

Number of Slices:	4283 out of 33792	12%
Number of Slice Flip Flops:	2544 out of 67584	3%
Number of 4 input LUTs:	7532 out of 67584	11%
Number of bonded IOBs:	98 out of 1104	8%
Number of BRAMs:	3 out of 144	2%

We know that the resource utilization shown here could be cut about in half by downsizing data path width in many of the cells. The synthesized design used 4 bit data paths in all cells, i.e. the cell can calculate values up to 15 (the last cell is a special case). However, the maximum values that will ever be computed by the cells varies through the array as shown in

Figure 8. Therefore, the data path only needs to be 1 bit wide in the cells in the top row and left column, 2 bits wide in the cells in the next 2 rows and columns, and so on. When we added this feature to the ripple through version that we designed before this systolic array version it did reduce the size to about half, and we intend to carry this feature into the systolic array design.

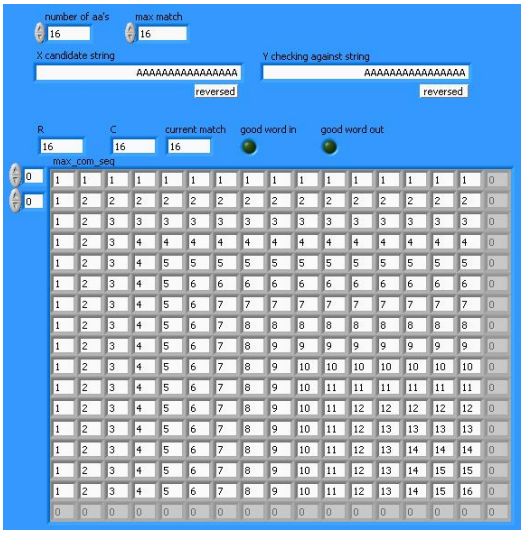

Figure 8. Maximum numbers that can be calculated in Levenstein matrix cells.

A higher level functional block diagram of the application is shown in Figure 9. The portions in yellow are those described above. This draft design was composed by Larry Merkle, Rose Hulman Institute of Technology, during a Visiting Summer Faculty Research assignment at IFTC. He produced synthesizable VHDL for most of the blocks which will serve as a starting point for us to produce a one FPGA solution. That work will be reported in the future.

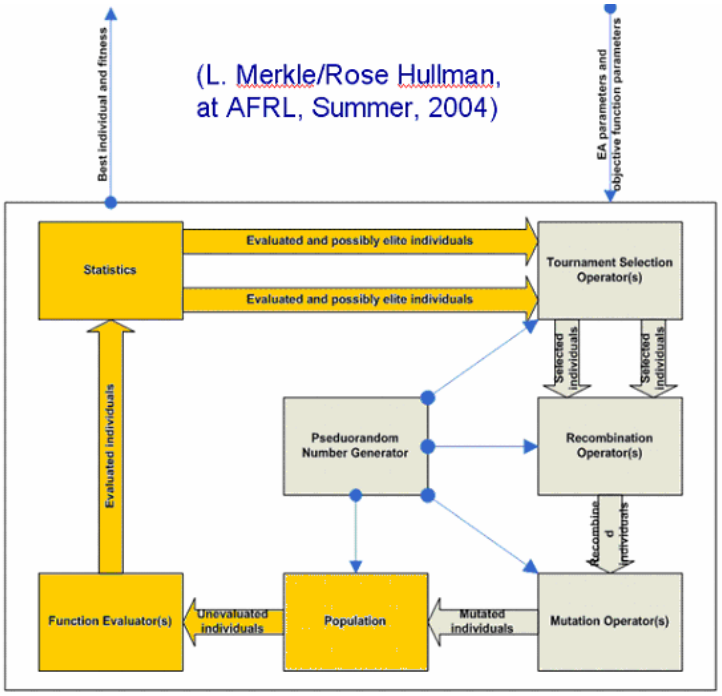

Figure 9. Higher level function block diagram of GA optimization FPGA hardware core.

Impulse_C Version:

We researched some of the Higher Order Language (HOL) VHLD development tools on the market today, and chose to evaluate Co_Developer from Impulse Accelerated Techologies, Inc. It requires the user to augment the original C source code with constructs that fashion a data path representation with queued streams passing data between blocks of combinational logic and registers implementing calculations. We produced an Impulse_C version of the complete GA/DNA Code Word Library Generation application that simulates correctly in the Co_Developer Application Monitor. However, to date we have not been able to successfully compile to a VHLD version that will load and execute in ModelSim. At this point progress is still being made, and we will continue to work with Impulse on the GA algorithm part of the problem.

Finally, there is presently no built in support for floating point or MPI communications in the Impulse tools set, although that may follow.

Conclusions:

In this paper we discussed the application of a Genetic Algorithm to two hard optimization test case problems, Non-Linear ODE Parameterization, and DNA Code Word Library Generation. We have also reported on speed-ups obtained through choice of language on a single PC platform, by moving to distribute C/MPI version running on a cluster, and by moving to a synthesizable VHDL version targeted to run on a hardware accelerated FPGA core. The GA exhibited faster speed solving both test case problems than other methods found in the literature. We also described a systolic array design for accelerating the calculation of the Levenstein matrix in the Code Word problem, as we have synthesized a version that clocks at 9.8ns, giving a 500X speed-up over software. We evaluated a particular commercial C to VHDL tool, and experienced problems related to tool maturity, support for floating point, and support for MPI for cluster applications. This work leads us to conclude that FPGA cores for hardware GAs will become an important method for solving a variety of integer optimization problems.

Acknowledgements:

Dr. Larry Merkle, Rose Hulman Institute of Technology, contributed to the GA Core for VHDL as a Visiting Summer Faculty Research program participant in the summer of 2004. Dr. Tony Macula, SUNY Geneseo, and Morgan Bishop, JEANSEE Corp., Geneseo, NY, contributed the Markov DNA Code Word generation results. Ann Rundell, Purdue University, provided MatLab codes for the Ag-Ab binding model. David Pelliren/Impulse Accelerated Technologies, provided assistance in evaluating Co_Developer software. Dr. Qinru Qiu contributed the work on the Sensor Network Energy Management Problem. The DARPA SIMBIOSYS (Dr. Anantha Krisnan), BIOCOMP (Dr. Sri Kumar/DARPA), Bob Kaminski/IFGA) programs provided partial support for this work through agent fees. Dr. John Gallagher/Wright State University, and Prof. Gary Lamont provided references and valuable discussions about compact/mini-pop GAs, and about multi-objective optimization using GA, respectively.

References:

1. Y. Zheng and A. Rundell, "Biosensor Immuno-surface Engineering Inspired by B-cell Membrane Bound Antibodies: Modeling and Analysis of Multivalent Antigen Capture by Immobilized Antibodies, IEEE Transactions on NanoBioscience, 2(1):14-25, 2003.
2. A. Rundell, R. DeCarlo, P. Doerschuk, and H. HogenEsch,, "Parameter Identification for an Autonomous 11th Order Nonlinear Model of a Physiological Process", Proceedings of the 1998 American Control Conference, 6: 3585-3589, 1998.
3. D.J. Burns and K.N. May, "On Parameterizing Models of Antigen-Antibody Binding Dynamics on Surfaces – a Genetic Algorithm Approach and the Need for Speed", Proceedings of the Genetic and Evolutionary Computing Conference – GECCO 2004, Seattle, WA, June, 2004, Vol. 1., pp 497-498.
4. A. Brenneman and A E. Condon, "Strand Design for Bio-Molecular Computers" (Survey Paper), Theoretical Computer Science, Vol. 287:1, 2002, pages 39-58:
5. S. Scott, A. Samal, and S. Seth, "HGA: A Hardware Based Genetic Algorithm", Proceedings of the 1995 ACM Third International Symposium on Field-programmable Gate Arrays, Monterey, CA, pp. 53-59, Feb. 1995.
6. P. Graham and B. Nelson, "Genetic algorithms in Software and in Hardware - a Performance Analysis of Workstation and Custom Computing Machine Implementations", 1996 Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines, Napa Valley, CA, USA, April 1996, pp. 216 – 225.
7. C. Aporntewan, and P. Chongstitvatana, "A hardware implementation of the Compact Genetic Algorithm", Proceedings of the 2001 Congress on Evolutionary Computation, 2001, Volume 1, May 2001, pp. 624 - 629 vol. 1.
8. B.E. Wells, C. Patrick, L. Trevino, J. Weir, and J. Steincamp, " Applying a Genetic Algorithm to Reconfigurable Hardware – a Case Study", 2004 MAPLD, paper 169.
9. M.K. Pakhira, R.K. De, "A Hardware Pipeline for Function Optimization Using Genetic Algorithms", Proceedings of the 2005 Conference on Genetic and Evolutionary Computation, Washington, DC, June 2005, pp. 949-956.
10. G.M. Megson and I.M. Bland, "The Systolic Array Genetic Algorithm, An Example of a Systolic Arrays as a Reconfigurable Design Methodology", Proceedings of the 1998 IEEE Symposium on FPGA's for Custom Computing Machines, Apr. 1998, pp. 260-261.
11. M. Bishop, A. Macula, W. Pogozelski, and T. Renz, "DNA Codeword and Library Design", 2nd Annual Conference on FOUNDATIONS OF NANOSCIENCE: SELF-ASSEMBLED ARCHITECTURES AND DEVICES (FNANO 05), Snowbird, UT, Apr. 2005. (Poster paper).
12. Eric Cigan and Robert Anderson "An Automated System for Floating- to Fixed-Point Conversion of High Performance of MATLAB Algorithms in FPGAs and ASICs", MAPLD 2004, Paper 228
13. Xiaojun Wang, Miriam Leeser, Haiqian Yu, "A Parameterized Floating-Point Library Applied to Multispectral Image Clustering", MAPLD 2004, Paper 166
14. H. T. Kung. Why Systolic Architectures? IEEE Computer, 15(1):37-46, January 1982.