Designing Evolvable Hardware for Military Systems (Extended Abstract)

Gary Lamont, Yong Kim, Rusty Baldwin, Guna Seetharanan
Department of Electrical And Computer Engineering
Air Force Institute of Technology

Dan Burns and Robert Ewing
Information Directorate
Air Force Research Laboratory

1 Introduction

Evolutionary Algorithms (EAs) and in particular Genetic Algorithms (GAs) are stochastic optimization heuristics in which searches in solution space are carried out by imitating the population genetics stated in Darwin's theory of evolution; "survival of the fittest." Selection, crossover and mutation operators, derived directly from natural evolution mechanisms are applied to a population of solutions, thus favoring the birth and survival of the best solutions. One needs to create an initial population, to define an objective function to measure the fitness of each solution. and to design the genetic operators that produce a new population of solutions from a previous one. By iteratively applying the genetic operators to the current generation population. It is hoped that the fitness of the best individuals in the population converges to at least a local optima, if not the global optimum.

Thus, the evolvable nature of a genetic algorithm (GA) provides a robust problem-solving method based on natural selection. Hardware's speed advantage and its ability to parallelize offer more efficient genetic algorithms computation(engines), possible meeting time-critical requirements. Speedups of 1-3 orders of magnitude have been observed when frequently used software routines were implemented in hardware by way of reprogrammable field-programmable gate arrays (FPGAs). Also, this research direction is part of a larger meta-level effort revolving around the system-on-a-chip concept. Reprogrammability is essential in a general-purpose GA engine because certain GA modules require changeability (e.g. the function to be optimized by the GA). Thus a hardware-based GA known as evolvable hardware is both feasible and desirable. Fully functional hardware-based genetic algorithms are discussed as a proof-of-concept system. They are generally designed using VHDL to allow for easy scalability. In general, they are designed to act as a co-processor with the CPU of a PC. The user programs the FPGAs which implement the function to be optimized. Other GA parameters may also be specified by the user. Simulation results and performance analysis are required in order to validate each of the integrated designs. Various GA-FPGA prototypes are described. In general, the advantage of hardware implementations is that they are more efficiency as compared to similar GA software implementations. Our current work [2] reflects a comprehensive discussion of contemporary GA-FPGA research and system-on-a-chip efforts along with an extensive bibliography. The efforts of individuals indicated in the following text are reflected in this bibliography.

1.1 Application

The major objective of this paper is to consider various generic FPGA designs employing genetic algorithm types that reflect upon various AF applications. In particular, those appropriate applications are in the Air Force Research Laboratory, the Information Directorate and the Sensors Directorate. The following is a incomplete list of known GA-FPGA activity:

- Reconfigurable Computing
  - Robot/UAV motion planning & path correction
  - Image processing and recognition
  - Fault tolerant computing
- Homeland Defense and Security
  - Internet monitoring & intruder detection
  - Use of GA-FPGA for recognizing intrusions
  - Chemical spectra analysis
- High Performance Time-Critical Algorithms

- Data and Video Processing
- Pattern classification and profiling
- Wavelet transformations and coefficients
- Target recognition and prediction
- DSP algorithms
- Nonlinear modeling

In designing specific GAs, appropriate representations are studied and efficient operators (crossover, mutation, selection) developed. Along with designing GA-FPGA implementations, our intent is to include GAs in the design process itself in order to move towards ”optimal” GA-FPGA designs.

2 Field-Programmable Devices

Prompted by the development of new types of sophisticated *Field-Programmable Devices* (FPDs), such as *Field Programmable Gate Arrays* (FPGAs), the process of designing digital hardware has changed dramatically over the past few years. Unlike previous generations of technology, in which board-level designs included large numbers of SSI chips containing basic gates, virtually every digital design produced today consists mostly of high-density devices. This applies not only to custom devices like processors and memory, but also for logic circuits such as state machine controllers, counters, registers, and decoders. When such circuits are destined for high-volume systems they have been integrated into high-density gate arrays. However, gate array Nonrecurring-Engineering (NRE) costs often are too expensive and gate arrays take too long to manufacture to be viable for prototyping or other low-volume scenarios. For these reasons, most prototypes, and also many production designs are now built using FPGAs. The most compelling advantages of FPGAs are instant manufacturing turnaround, low start-up costs, low financial risk, easy of design changes, and performance near equal to that of *Application Specific Integrated Circuits* (ASICs).

The market for FPDs has grown dramatically over the past decade to the point where there is now a wide assortment of devices to choose from. A designer today faces a daunting task to research the different types of chips, understand what they can best be used for, choose a particular manufacturer’s product, learn the intricacies of vendor-specific software and then design the hardware. Confusion for designers is exacerbated by not only the sheer number of FPDs available, but also by the complexity of the more sophisticated devices. Moreover, there are complex embedded engineering problems associated with input/output, memory overhead, specialized device interfacing for the variety of applications.

Currently there are many different types of FPDs, such as *Programmable Logic Array*, *Programmable Array Logic*, *Complex Programmable Logic Device), Field Programmable Gate Array*, and *High Capacity PLDs*. Among all these programmable devices, FPGA is the mostly commonly used and considered for implementing GAs for its versitility and high performance.

There are two basic categories of FPGAs on the market today, SRAM-based FPGAs and antifuse-based FPGAs. In the first category, Xilinx and Altera are the leading manufacturers in terms of number of users. For antifuse-based products, Actel, Quicklogic, Cypress, and Xilinx offer competing products. FPGAs are produced by many vendors, but current state-of-the-art FPGAs are produced by Altera and Xilinx. The current flagship FPGAs from Xilinx and Altera were introduced on first half of 2003 and both features an 1.5v system, 130nm copper wiring, 100k logic cells, and 10Mb RAM. The basic structure of xilinx FPGAs is array-based, meaning that each chip comprises a two-dimensional array of logic blocks that can be interconnected via horizontal and vertical routing channels. Altera’s FLEX 8000 series consists of a three-level hierarchy.

3 GA-FPGA Research

Various researchers have and are continuing to develop GA inspired FPGA implementations [1]. Some of these are discussed here. The intent is to map various GA algorithmic elements to FPGA structures and implement in various architectures. The various GA types include the simple GA, the compact GA, and the extended compact GA and possibly a multi-objective GA. Also, single objective and multiobjective applications are addressed. It is desired that hierarchical testing of such systems compare software implementations with various FPGA designs and implementations. Of course since GAs are stochastic techniques, realtime operation depends upon the application and desired results.

3.1 GA-FPGA Design

The desire is to create an *implementable* VHDL representation of a general genetic algorithm similar to that in Figure 2 which would allow the user to choose several GA parameters. The user-controlled parameters are the initial population’s size and its members, the number of generations, the initial seed for the pseudorandom number generator, and the mutation and crossover probabilities. Values for these parameters would be selected by the user in software which would send the appropriate signals to initialize and start a hardware GA-FPGA.

3.1.1 The Modules and Their Functions

The modules in Figure 1 are patterned after the GA operators defined in Goldberg’s simple genetic algorithm (SGA). The

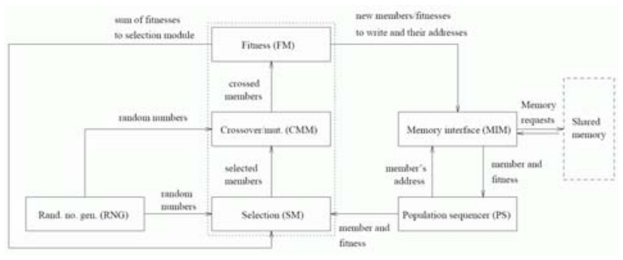

Figure 1. Box-level schematic of the overall HGA system.

Hardware GA (HGA) of S. Scott (1998) has modules that operate concurrently with each other and together form a coarse-grained pipeline. All modules are written in VHDL and are independent of the operating environment and implementation technology (e.g. Xilinx FPGAs or fabricated chips) except for the memory interface module. The functionality of this module varies according to the physical memory attached to it and the desired interface between the HGA and its user.

The basic functionality of the HGA design of Figure 1 is as follows: After all the parameters have been loaded into the shared memory, the memory interface module (MIM) receives a "Go" signal from the front end. The MIM acts as the main control unit of the HGA and is the HGA's sole interface to the outside world. The MIM notifies the fitness module (FM), crossover/mutation module (CMM), the pseudorandom number generator (RNG) and the population sequencer (PS) that the HGA is to begin execution. Each of these modules requests its required parameters from the MIM, which fetches them from the appropriate places of the shared memory. The population sequencer starts the pipeline by requesting population members from the MIM and passing them along to the selection module. The task of the selection module (SM) is to receive new members from the PS and judge them until a pair of sufficiently fit members is found based on a random number. At that time it passes the pair to the crossover/mutation module (CMM), resets itself, and restarts the selection process. When the crossover/mutation module receives a selected pair of members from the SM, it decides whether to perform crossover and mutation based on random values sent from the RNG. When done, the new members are sent to the fitness module for evaluation.

The fitness module evaluates the two new members from the CMM and writes the new members to memory through the MIM. The FM also maintains some records concerning the current state of the HGA that are used by the SM to select new members and by the FM to determine when the HGA is finished. The above steps continue until the FM determines that the current HGA run is finished. It then notifies the MIM of completion which in turn shuts down the HGA modules and sends the "Done" signal to the front end.

Since the modules of the HGA system are written entirely in VHDL, specific aspects of the design such as I/O bus size, storage facility size, etc. can be specified in terms of parameters which can be easily changed when the need arises. The interesting parameters of the HGA are *n*, the maximum width in bits of the population members, the maximum width in bits of the fitness values, the maximum size of the population, and the maximum number of generations. Many parameters are specified at VHDL compile time and are different than the HGA run time parameters.

Design Pipelining and Parallelization: The design in Figure 1 is a coarse-grained pipeline. This is evident by noting that when a module completes a task, it immediately awaits more input to repeat processing. Because of this pipelining, GA operations do not have to be suspended while other GA operations run, which happens in a sequential software implementation. Thus a significant speedup over software is realized. Parallelization of HGA modules is also possible.

3.1.2 VHDL Implementation

A genetic algorithm is usually written in VHDL and intended for parallel FPGA hardware implementation. Some contemporary developers include S. Scott(1998), C. Aporntewan (2001), P. Martin (2002). Due to pipelining, parallelization, and no function call overhead, a hardware GA yields a significant speedup over a software GA, which can be especially useful when the GA is used for real-time applications, e.g. disk scheduling and image registration. Since a general-purpose GA requires that the fitness function be easily changed, the hardware implementation must exploit the reprogrammability of certain types of field-programmable gate arrays (FPGAs), which are programmed via a bit pattern stored in a static RAM and are thus somewhat easily reconfigured. Also course-grained and fine-grained parallel pipelining need to be considered. Automated GA-FPGA detailed design and implementation synthesis is a desired development environment. Other technologies might also be considered such as use of systolic array libraries. In this section, we briefly discuss some of work in hardware-based GAs.

The Compact GA-FPGA: J. I. Hadalgo (2001, 2002) has investigated the design of a compact genetic algorithm (cGA)to solve Multi-FPGA Partitioning problems. Nowadays Multi-FPGA systems are used for a great variety of applications such as dynamically re-configurable hardware applications, digital circuit emulation, and numerical computation. Both a sequential and a parallel version of a compact genetic algorithm have been designed and implemented on a cluster of workstations. The peculiarities of the cGA permits one to save memory in order to address large Multi-FPGA Partitioning problems, while the exploitation of parallelism allows to reduce execution times. The good results achieved on several experiments conduced on different Multi-

FPGA Partitioning instances show that their solution is viable to solve Multi-FPGA Partitioning problems.

The Extended Compact GA-FPGA: The eCGA by Kumara Sasty (2000) is based on the idea that the choice of a good probability distribution is equivalent to linkage learning. The measure of a good distribution is quantified based on minimum description length (MDL) models. The key concept behind MDL models is that given all things are equal, simpler distributions are better than the complex ones. The MDL restriction penalizes both inaccurate and complex models, thereby leading to an optimal probability distribution. Thus, MDL restriction reformulates the problem of finding a good distribution as an optimization problem that minimizes both the probability model as well as population representation. The probability distribution used in ECGA is a class of probability models known as marginal product models (MPMs). MPMs are formed as a product of marginal distributions on a partition of the genes and are similar to those of compact genetic algorithm and others. Unlike the models used in CGA, MPMs can represent probability distributions for more than one gene at a time. MPMs also facilitate a direct linkage map with each partition separating tightly linked genes. Hence, in the current study each gene partition would refer to a building block (BB). The identification of MPM using MDL and the creation of a new population based on MPM need to be explained. The identification of MPM in every generation is formulated as a constrained optimization problem.

Multi-Objective Genetic Algorithms: A multi-objective optimization problem (MOP) consists of decision variables, two or more objective functions, and constraints. These three components of an MOP are defined as follows: 1) decision variables: variables whose numerical values are controlled by the decision maker. 2) objective function: a function that maps decision variable values to values reflecting a performance level; optimizing this performance level entails either maximization or minimization. 3) constraints: restrictions imposed by the particular problem that must be satisfied to render a solution acceptable. They describe dependencies among decision variables and constants (or parameters) involved in the problem. Our intent is to continue designing and implementing MOGAs for selected Air Force problems in a multiple parallel FPGA environment.

4 Generic Design Validation and Analysis

In previous sections, we presented several hardware-based GAs for FPGAs. Each is a variation of the genetic algorithm potentially implementable in hardware and each can provides significant speed increase over the same algorithms executed on a general-purpose computer.

The various GA-FPGAs should be simulated in order to validate correct functionality and to analyze the performance of the design. Various FPGA technologies are and will be considered. Initially, pedagogical test problems are to be utilized with real-world higher-dimensional examples to follow. The performance efficiency analysis includes analyzing the pipelines to identify bottlenecks, understanding the memory access processes and identifying the I/O traffic, as well as others: Verification of Correct Functionality,Performance (efficiency and effectiveness) Analysis,Design Improvements,Prototype System Comparisons with a Software-Based GA. Note that associated efforts are also being supported by the NSF, DARPA, and the other military services.

Reconfigurable hardware by itself, however, cannot provide the GA solution for all possible situations. One of the major problems is that not all functions or VHDL codes are synthesizable/implementable using FPGAs. When a highly abstract function is required, it may be difficult or even impossible to realize such function using real hardware. This is true for not only FPGA-based implementation, but also true for more versatile ASIC-based implementation. The appropriate way to approach such problem is to consider and find a system-wide solution.

A flexible, future-proof platform must contain supporting hardware and interface as a whole system, such as multiple microprocessors, ASICs and memories to support the core FPGAs to maximize the effectiveness of genetic algorithm based applications. In addition, a specific operating system and matched hardware to reconfigure and monitor current GA operation are essential to accommodate varying nature of GAs. A new GA high-level language standard and library, such as VHDL, are required and desirable to facilitate direct implementation and to optimize GA applications. Most importantly, a verifiable and/or fault-tolerant implementation are required to validate, to monitor current process, and to provide reliable service especially for various military applications.

References

[1] John C. Gallagher, Saranyan Vigraham, and Gregory Kramer. A family of compact genetic algorithms for instrinsic evolvable hardware. *IEEE Trans. on Evolutionary Computation*, vol. 8, no. 2:pp. 111–126, April 2004.

[2] Gary Lamont, Yong Kim, Rusty Baldwin, and Guna Seetharanan. *Designing FPGA Based Genetic Algorithm Systems*. Report, Department of Electrical and Computer Engineeing, Graduate School of Engineering and Management, Air Force Institute of Technology, March, 2004.