FPGA IMPLEMENTATION OF PARTICLE SWARM OPTIMIZATION FOR INVERSION OF LARGE NEURAL NETWORKS

Paul D. Reynolds
Department of Engineering
Baylor University
Waco, TX 76798
Paul_Reynolds@baylor.edu

Russell W. Duren
Department of Engineering
Baylor University
Waco, TX 76798
Russell_Duren@baylor.edu

Matthew L. Trumbo
Department of Engineering
Baylor University
Waco, TX 76798
Matt_Trumbo@baylor.edu

Robert J. Marks II
Department of Engineering
Baylor University
Waco, TX 76798
Robert_Marks@baylor.edu

ABSTRACT

Particle swarm inversion of large neural networks is a computationally intensive process. By the implementing a modified particle swarm optimizer and neural network in reconfigurable hardware, many of the computations can be preformed simultaneously, significantly reducing computation time compared to a conventional computer.

1. INTRODUCTION

Reconfigurable hardware is capable of performing computations in parallel, allowing for the computation time of algorithms to be significantly decreased. However, the number of computations performed in parallel, and hence the speedup, is limited by hardware constraints. In order to maximize speedup, algorithms can be modified to use minimal hardware for computations such as multiplications and additions. Portions of an algorithm can also be changed in order to decrease the number of computations required. In this paper we demonstrate with a particle swarm used for inversion of a neural network.

2. THE INVERSION PROBLEM

Given a set of sonar system parameters and environmental parameters, the ensonification of an underwater environment can be predicted by computationally intensive computer models. It is desirable to be able to perform an inversion on the system: given the state of the underwater environment (bottom type, bottom depth, wind speed, sound speed profile, and others) determine other, controllable input parameters (frequency, depth of transmitter, and depth of receiver) to achieve optimal sonar performance.

Inversion was originally performed by comparison with previously known response results. The controllable parameters from sets with similar conditions were run through the computational model with the environment and uncontrollable parameters fixed. The "inverse" was selected by choosing the set which mostly closely matched the desired sonar performance.

This method is faulty in two aspects. First, is that it does not find the actual optimum parameters for the scenario, but is limited by the template of previously known optimal parameters. The second is that the method is too slow for real time due to the time required to compute the acoustic model.

In order to perform a more accurate inversion in real time, a better method for testing unknown parameters must be implemented and the computation time of the model must be significantly decreased.

3. NEURAL NETWORK SOLUTION

In order to decrease the time of the acoustic model computation, an artificial neural network was trained with data calculated by the model [1-3]. The network has a 27-40-50-70-1200 architecture, with 27 inputs corresponding to the sonar system and environmental parameters and 1200 outputs corresponding to the signal to interference ratio in decibels at locations in an 80 by 15 pixel grid [4]. The forward computation time run on a dedicated 1.3 GHz PC of the neural network is approximately 1 millisecond, much faster than the tens of seconds used by the acoustic model.

Pixel output is typically in the range of -80 to -500 dB. The average error per pixel of the neural network outputs from the acoustic model outputs is 4.58 dB. This gives a typical error between 1% and 5%.

In order to perform a more accurate inversion, a particle swarm optimization is used to generate the input sets tested. The particle swarm uses multiple "agents" to search through the input space. The standard velocity and position update equations are used [5].

$$X[k+1] = X[k] + V[k]$$
$$V[k+1] = V[k] + C_1 R_1 (P - X[k]) + C_2 R_2 (G - X[k])$$

where X is the position vector, V is the velocity vector, P is the personal best location, G is the global best location, C_1 and C_2 are personal and global bias constants, and R_1 and R_2 are uniform random variables between zero and one.

The swarm was tested on its ability to find a set of inputs that produces a specified SIR on an 80 by 15 output grid. The objective function used for verifying the particle swarm's accuracy is calculated by taking the average of absolute difference between the desired outputs and the found outputs. A lower average value is considered to be

0-7803-8916-6/05/$20.00 ©2005 IEEE

better. Particle swarm, in such cases, keeps track of the local and global 'smallest' values. A second inversion problem is finding the inputs that maximize the SIR over a specified region on the grid. The fitness function used to find input sets for maximum output SIR is calculated by taking the sum of outputs in the specified location. In this case, a higher number is considered to be better.

Ten agents were found to work well with these fitness functions. The swarm was allowed to run for one hundred thousand fitness evaluations for each test.

4. TEST RESULTS

For one hundred tests, the average error per pixel between the best input set found by the particle swarm and the desired output set was approximately 1.94 dB. This was deemed to be an acceptable level of error in the inversion, with typical error between 0.5% and 2.5%. The neural network error from 1% to 5% is more significant than that introduced by the particle swarm. The low error suggests that a particle swarm inversion is a good method for the maximization problem.

However, performing one hundred thousand fitness evaluations and position and velocity updates takes approximately two minutes to complete on a dedicated 1.2 GHz PC. Though an improvement over the more cumbersome template matching method of inversion, this is too slow to be considered real time. In order to achieve results closer to real time, the inversion method has to be modified and implemented into hardware.

5. HARDWARE LIMITATIONS

The main goal for a hardware implementation is to decrease computation time by performing as many calculations simultaneously as possible. For many complex algorithms this is limited by hardware space.

To solve the sonar inversion problem using a trained neural network, the SRC-6e reconfigurable computer was used. It contains two Xilinx XC2V6000 field programmable gate arrays (FPGAs). These FPGAs each contain 144 multipliers and 144 eighteen-kilobit random access memory blocks as well the equivalent of 6 million logic gates. The XC2V6000s run at 100 megahertz. One FPGA is used for fitness function calculations while the other is used for the particle swarm update equations.

Having only 144 multipliers on an FPGA greatly limits the neural network implementation speed, which requires approximately 92,000 multiplications. The data representation and sigmoid function of the neural network also had to be modified to better fit the hardware. The neural network modifications and implementation structure are the topic of another paper [7].

Knowledge of some of the neural network structure is needed for the particle swarm implementation. The network implementation uses a sixteen-bit fixed-point representation for both the inputs and outputs. The input data has a two's complement representation with the fixed point after the most significant bit. The output data also has a two's complement representation with the fixed point after the eleventh most significant bit. Also, one forward evaluation of the modified network can be calculated in 1465 clock cycles. At one hundred megahertz, this is just under fifteen microseconds. This allows over sixty thousand fitness evaluations to occur every second, about sixty times faster than the conventional computer implementation.

6. PARTICLE SWARM MODIFICATIONS

In order to save area on the FPGA and avoid using predefined components, some particle swarm modifications are necessitated. .

The simplest modification of the update equations is to use powers of two for the bias constants. This way, instead of using a slower and larger multiplier, a simple right shift is used. This saves one clock cycle in the update equations. For this particle swarm implementation the personal best bias constant is set to one eighth and the global best bias constant is set to one sixteenth. This is a right shift of three and four bits respectively.

Generating high-quality random numbers in hardware can be difficult. For one hundred thousand fitness evaluations in the swarm algorithm, 5,400,000 random numbers are needed.

The first option examined was simply removing the random components from the update equations. Randomness was previously removed to prove the stability of the particle swarm algorithm [6]. In the computer simulation, a particle swarm was run thirty times both with and without random contributions. The same desired output was used for all runs. Figure 1 shows the results from the simulations.

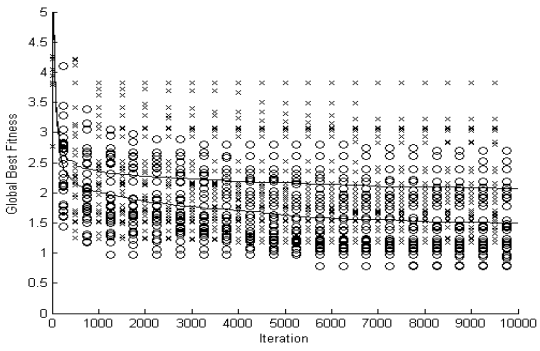

Figure 1. The global fitness over time for particle swarms with and without randomness.

The x's show the global best fitness over time for the swarms without randomness. The top line is the average fitness value for these deterministic swarms. The o's show the global best fitness over time for the stochastic swarms. The lower line is the average fitness of these swarms.

The stochastic swarms out perform the deterministic swarms by about 0.5 dB per pixel on average. However, there is much overlap between the results of the two methods. Many of the deterministic swarms perform better than several of the stochastic swarms. It was decided that exploring implementation of stochastic updates in hardware would be useful, though actual implementation of stochastic updates might not be necessary.

7. RANDOMNESS

The particle swarm was implemented in hardware using three different methods, one deterministic, and two containing different implementations of random number generators.

The two methods of generating pseudorandom numbers used were a linear feedback shift register (LFSR) and a simple squaring technique. The LFSR is typically used for generating data to test digital logic. It is made of XOR gates and a shift register. For the squaring technique, a simple equation is implemented where the fractional portion of a square is used as the random value. The equation is

$$R[k+1] = dec[(C + R[k])^2]$$

where C is a constant greater than one with a random fractional portion. For this implementation, the binary value of C is set to a 1 followed by 17 randomly selected fractional bits.

Both methods are modeled in computer simulations. Figure 2 and Figure 3 show comparisons between these methods and a computer generated random variable. From the figures, the LFSR has a histogram more similar to a uniform random variable, while the squared fractional implementation is closer in frequency spectrum.

In the hardware implementation, the average pixel error for the deterministic swarm over one hundred trials is 2.36dB. The particle swarm with a LFSR generating random numbers has an average pixel error of 2.35dB. The particle swarm using the squared fractional implementation had an average pixel error of 2.37dB.

When searching for known achievable sets, all three implementations produce approximately the same level of output error. Due to its simplicity, this makes the deterministic method most desirable. The deterministic method introduces a small amount of randomness due to truncation caused by the fixed-point calculations.

8. RESULTS

None of the hardware implementations are as accurate as the conventional computer average error of 1.94 dB per pixel. In order to account for this increase, note that the hardware implementation uses fixed-point math, while the conventional computer uses floating-point math. An

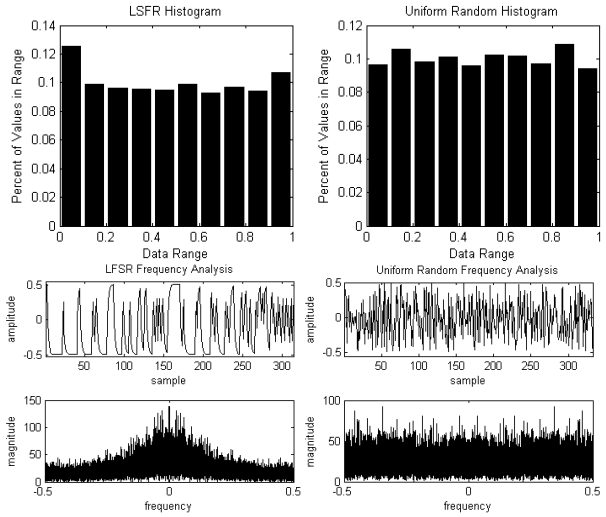

Figure 2. A comparison between the LFSR randomness implementation and a uniform random variable.

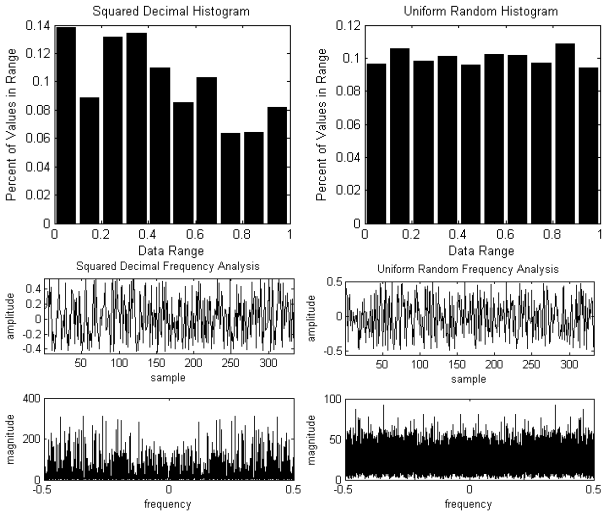

Figure 3. A comparison between the squared fractional randomness implementation and a uniform random variable.

analysis was performed to determine the components of the error in the hardware implementation. The error component from the deterministic hardware implementation of the swarm calculation was 1.81dB, which is actually better than the conventional swarm. The error component resulting from the hardware implementation of the neural network was 1.42dB. These two components combined in an RMS manner to produce an overall error of 2.30dB which agrees with the experimental results.

The average pixel error between the computer network outputs and the acoustic model outputs it mimics is 4.58dB. With this level of error already in the system, the RMS error added by the network and particle swarm translation to hardware is less than 0.22dB, which is insignificant.

9. CONCLUSIONS

The output from the hardware particle swarm inversion has an average pixel difference of 2.54dB from a known achievable desired output or an average difference of 1.53%. This low error implies that the particle swarm inversion will be able to find a set of inputs that produces outputs closest or near the closest to a desired output set.

Figure 6 show two sets of outputs from inputs found for the maximization of a specific area as well as the specified area. All other areas were ignored for calculation of fitness. Localized maximization is equivalent to attempting to find infinite signal to interference ratio, which, of course, is outside the achievable set. It is evident from the figures that the particle swarm optimization found a set of inputs which maximizes the local area and ignores the rest of the figure.

The time to complete the same one hundred thousand iteration particle swarm optimization on a conventional computer is nearly two minutes. At one hundred megahertz, the two-chip hardware implementation takes under 1.8 seconds to complete, approximately sixty-five times faster.

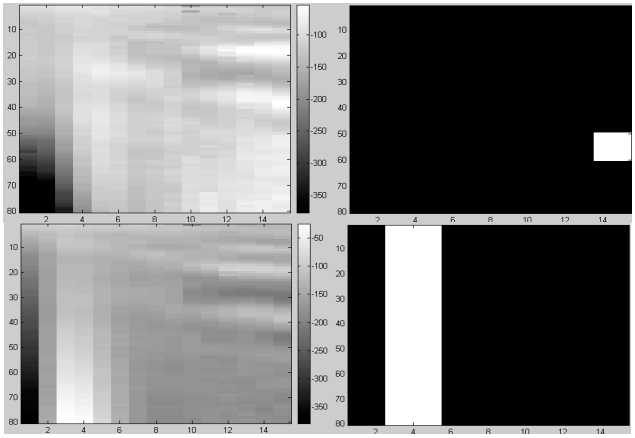

Figure 6. Particle swarm inversion results maximizing the specified area. The white areas in the images on the right show the desired maximization areas. The images on the left show the outputs from the solution found by the particle swarm, where lighter areas represent higher signal to interference ratios.

10. FUTURE IMPLEMENTATIONS

The calculation time of evaluating the fitness function, as with this problem, is typically much longer than the position and velocity updates for one agent. Therefore, inversion time is determined solely by the fitness function implementation.

The current particle swarm implementation uses two identical FPGAs operating at one hundred megahertz and with 144 multipliers. For specially designed inversion hardware, chips of different sizes could be used. The particle swarm implementation would be implemented in a much smaller chip without multipliers. The fitness function would use the newer generation of Xilinx Virtex 4 FPGAs operating at a five hundred megahertz and containing 512 multipliers. The speed increase alone would allow the inversion time to decrease from 1.8 seconds to .36 seconds. The additional multipliers could be used to perform fitness evaluations of several agents at the same time or to improve the speed of a single fitness evaluation. Predicted speedup based on the increase in multipliers is about seven. This combined with the faster chip speed would allow nearly twenty network inversions to be performed every second.

11. REFERENCES

[1] W. Fox, R. Marks II, M. Hazen, C. Eggen, and M. El-Sharkawi, "Environmentally Adaptive Sonar Control in a Tactical Setting.", in Impact of Environmental Variability on Acoustic Predictions and Sonar Performance, pp. 595-602, Sept. 2002.

[2] M. Hazen, R. Marks II, W. Fox, M. El-Sharkawi, and C. Eggen, "Sonar Sensitivity Analysis Using a Neural Network Acoustic Model Emulator," *Oceans '02 MTS/IEEE*, vol. 3, pp. 1430-1433, Oct. 2002.

[3] C. Jensen, R. Reed, R. Marks II, M. El-Sharkawi, Jae-Byung Jung, R. Miyamoto, G. Anderson, and C. Eggen, "Inversion of Feedforward Neural Networks: Algorithms and Applications," *Proceedings of the IEEE*, vol. 87, pp. 1536 -1549, Sept. 1999.

[4] B. Thompson, R. Marks II, M. El-Sharkawi, W. Fox, and R. Miyamoto, "Inversion of Neural Network Underwater Acoustic Model for Estimation of Bottom Parameters Using Modified Particle Swarm Optimizers," *2003 International Joint Conference on Neural Networks*, pp. 1301-1306, July 2003.

[5] R. Eberhart, and Y. Shi, "A Modified Particle Swarm Optimizer," *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence*, pp. 69 -73, May 1998.

[6] M. Clerc and J. Kennedy, "The Particle Swarm – Explosion, Stability and Convergence in a Multidimensional Complex Space," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 58-73, Feb. 2002.

[7] P. Reynolds, "Algorithm Implementation in FPGAs Demonstrated through Neural Network Inversion on the SRC-6e," M.S. thesis, Baylor University, Waco, TX, 2005.