Designing Energy-Efficient Approximate Adders using Parallel Genetic Algorithms

Adnan Aquib Naseer, Rizwan A. Ashraf, Damian Dechev and Ronald F. DeMara
Department of Electrical Engineering and Computer Science
University of Central Florida
Orlando, Florida - 32826
demara@mail.ucf.edu

***Abstract*—Approximate computing involves selectively reducing the number of transistors in a circuit to improve energy savings. Energy savings may be achieved at the cost of reduced accuracy for signal processing applications whereby constituent adder and multiplier circuits need not generate a precise output. Since the performance versus energy savings landscape is complex, we investigate the acceleration of the design of approximate adders using parallelized Genetic Algorithms (GAs). The fitness evaluation of each approximate adder is explored by the GA in a non-sequential fashion to automatically generate novel approximate designs within specified performance thresholds. This paper advances methods of parallelizing GAs and implements a new parallel GA approach for approximate multi-bit adder design. A speedup of approximately 1.6-fold is achieved using a quad-core Intel processor and results indicate that the proposed GA is able to find adders which consume 63.8% less energy than accurate adders.**

***Index Terms*—parallelism, genetic algorithms, parallel genetic algorithm, inexact arithmetic units, approximate computing, adder, variable accuracy, low power, adders, error distance, power consumption, process variation, delay, power reduction**

I. Introduction

Power consumption has become a significant concern in computing architectures ranging from mobile applications to high performance computing centers. The need for higher energy efficiency in recent times has encouraged industry and academia to explore various new avenues to decrease power consumption. Some of the common methods of reducing power consumption include Near Threshold Voltage operation [1], Runtime Datapath Adaptation [2] [3], and Algorithmic Truncation and Approximation [4]. Herein, we focus on approximate circuits to achieve energy-efficiency which can be achieved by trading off accuracy. Approximation in current scenarios could be developed using evolutionary methods such as *Evolutionary Algorithms*. Evolutionary algorithm based design approaches have proven to obtain creative design solutions in a wide range of optimization problems [5]. One of the advantages of an Evolutionary Algorithm as opposed to a random search is its ability to obtain a global optimum solution in shorter time for complex problems. In this paper, we look at an approach to reduce the time required by a *Genetic Algorithm* (GA) to determine a circuit design which occupies less area as compared to accurate design by parallelizing the algorithm used to evaluate the fitness of individuals in the population.

II. Genetic Algorithms

A. Fundamentals of Genetic Algorithms

This section reviews some of the fundamental terms of the Evolutionary-based Computing Approach known as Genetic Algorithms.

A GA, at the genesis of the procedure contains a number of random individuals generated by the system or pre-seeded by the user, depending on the extent of automation desired. The individuals together create a *population*. When this population is manipulated over time, it defines a sequence of *generations*, whereby the individuals of a population react with each other using Genetic Operators to form new individuals, these newly-created individual designs are referred to as *offspring*, and the population of offspring is a generation. Usually the original population is denoted as zeroth generation. Every individual in the population can be represented as a binary string, which is referred to as the *chromosome* of the individual. The chromosome is an important data structure which determines the design characteristics including device selection and interconnection of the individual circuit being represented. The performance of an individual can be evaluated by using a *fitness* function. Fitness can be specified as a cost function including mean error and resource usage according to a user-defined value called the *error criterion*.

As the GA runs, it performs Genetic Operators such as a *crossover*, which is a method by which interacting individuals exchange binary data, replacing the original data, to exchange subcircuits from high-performing parents to offspring. To make sure that the GA does not converge to a local solution, a *mutation* function is introduced, which randomly changes a value in the chromosome. The probability of mutation is kept to low levels in the range of 1 random modification for every 1000000 bits. The mutation operator is kept to such low levels to prevent the GA from performing a random search. In certain GAs, the fittest individual of each generation is always retained through successive generations without any modification, this process of selection is referred to as *elitism*. In cases where elitism is not used, individuals are selected based on probability, which is a function of its fitness. If the

fitness level of an individual is high, then it has a higher chance of being selected and vice-versa.

B. Parallelism in GAs

The power of Genetic Algorithms is given by Schema Theorem [6], the schema theorem is a set of symbols used in a string $\{1*00*1\}$ where * can either be 1 or 0. The order of the schema $o(H)$ is defined as the number of fixed positions in the schema, while $\delta(H)$ is the distance between the first and the last specific positions. The order of the string 1*00*1 is 4 and the defining length is 5. The schema theorem states that the fitness of a short schemata with higher fitness than the average population will increase exponentially in successive generations [7]. It is expressed by the equation:

$$E(m(H,t+1)) \geq \frac{m(H,t)f(H)}{a_t}[1-p] \tag{1}$$

where, $m(H,t)$ is the number of strings belonging to schema H, $f(H)$ is the average fitness of schema H, a_t is the average fitness at generation t and p is the probability that a mutation could destroy a schema. Here p is defined from the following equation:

$$p = \frac{\delta(H)}{l-1}p_c + o(H)p_m \tag{2}$$

where, $o(H)$ is the order of the schema, l is the length of the code, p_m is the probability of mutation and p_c is the probability of crossover [7]. From equation (1) and (2) it can be inferred that it is advantageous to keep the chromosomes as short as possible. Also, from the schema theorem it can be inferred that even in a sequential algorithm, it is possible for the algorithm to handle $o(N3)$ schema at a time for a population of N individuals. Thus the GAs can handle large number of schema at a time, which can be considered as inherent parallelism [6].

Most GAs can be subdivided into four parts based on amount of computation namely:

- Population Selection
- Crossover and Mutation
- Fitness Evaluation
- Offspring Generation

The list also illustrates where a GA could be parallelized with greater efficiency overall. Apart from this a GA can also be parallelized based on its detailed algorithmic flow [6]. Some of the methods mentioned in [6] are

Parallelism of Individual's fitness
: The fitness evaluation of population consumes the most amount of power and time, depending on the algorithm and fitness dependencies, the fitness calculation can be parallelized.

Parallelism of occurrence of offspring
: Processes by which offspring are produced could be parallelized, for example, when a crossover takes place only two parents are used in a single crossover, while the rest of the population lies idle, this can be rectified in parallel computation by performing crossover simultaneously over the entire population.

Parallelism based on population grouping
: In certain cases, the population of individuals could be subdivided into smaller sub-groups which could be assigned to different threads.

Apart from the areas where there is potential to parallelize, *Parallel Genetic Algorithms*, henceforth referred to as PGAs can be subdivided into three groups based on the resolution at which the PGA works, namely

Global PGA (GPGA)
: A GPGA model uses the master slave method to divide the workload among threads. The slave threads calculate the fitness of an individual and perform genetic operations in parallel, whereas the master thread operates serially, assigning individuals to the slave threads. It is one of the most simple and effective PGAs to implement, but in GPGAs there is a high change of uneven distribution of workload. [6]

Coarse-grained PGA (CPGA)
: In a coarse grained model the population is subdivided into groups and genetic operations are performed independently within a sub-population [6]. This type of PGA introduces a new function *Migration*. Migration defines the rate at which individuals from one group move from one sub-population to another sub-population.

Fine-grained PGA (FGPGA)
: FGPGA assigns each individual to a thread and this individual interacts with another individual within a hamming distance of 1. This type of PGA could maintain diversity based on the neighborhood chosen. [6]

III. Related Work

Approximate computing offers a promising approach for reduced energy operation for applications which can tolerate some imprecision. There are many types of approximate circuits, including those constructed by voltage over-scaling and over clocking [8] and others as mentioned in [9]–[12]. However, this paper uses approximate adders which utilize lesser number of transistors than original accurate design. The Adders used in this paper are obtained from [13]. As these adders are analyzed in a manner which is suited to the application presented in this paper. In this section we discuss and analyze the three different types of adders used and the accurate adder. The Accurate adder is the reference adder from which the other adders are derived.

A. Conventional Mirror Adder

A Conventional Mirror Adder (CMA) consists of a total of 24 transistors [13], as shown in Figure 1. The other approximate adders are obtained from the CMA.

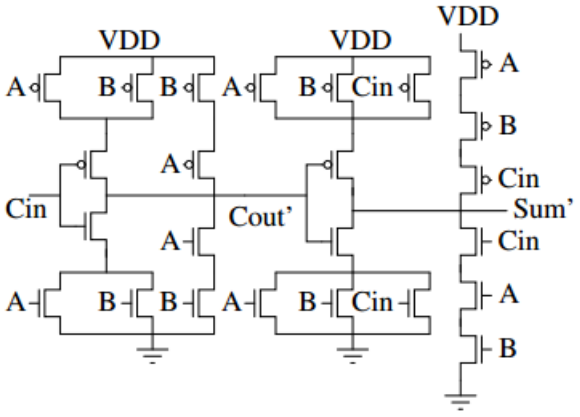

Fig. 1. Conventional Mirror Adder [13]

B. Approximation 1

The first approximate mirror adder is obtained by reducing the number of transistors in the schematic one by one until the defined error constraints are breached. Once the error constraints are breached the last modification is reverted to obtain the Approximation [13], this method is applied to the other approximate adders as well. AMA1 is shown in Figure 2.

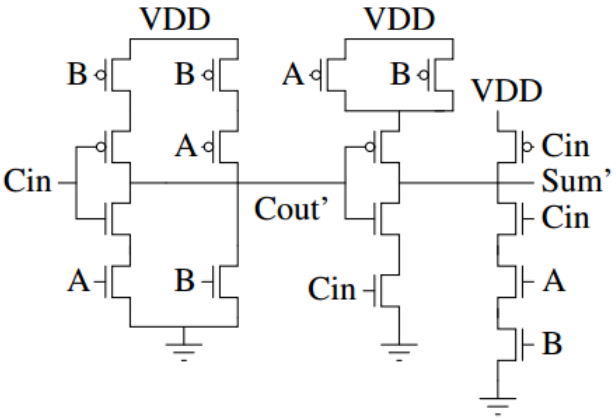

Fig. 2. Approximate Mirror Adder 1 [14]

C. Approximation 2

Approximate Mirror Adder 2 (AMA2) introduces 2 errors in C_{out} and 3 errors in Sum, as shown in Table I.

D. Approximation 3

Approximate Mirror Adder 3 (AMA3) C_{out} and Sum are accurate for 4 out of 8 outputs, the schematic for AMA3 is given in Figure 3.

The performance in terms of accuracy can be determined using the truth table shown in Table I.

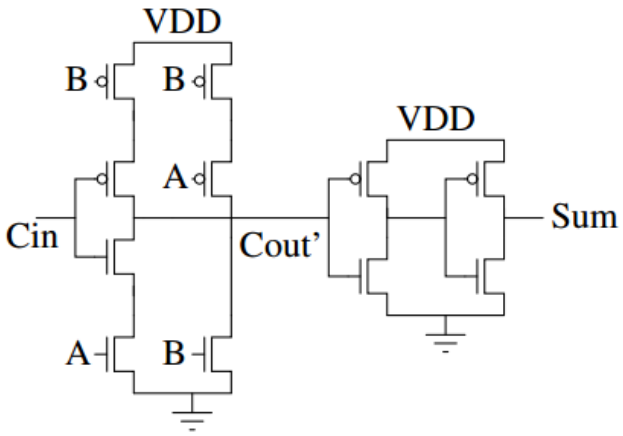

Fig. 3. Approximate Mirror Adder 3 [14]

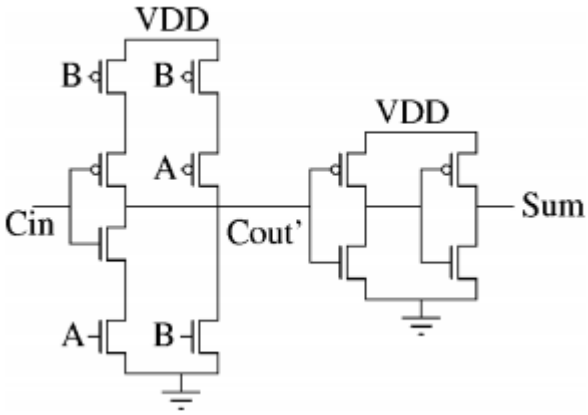

Fig. 4. Approximate Mirror Adder 4 [14]

IV. Fitness Evaluation

In this section we explore the various Error Metrics available in the literature to devise an Error Measurement Technique which will be utilized in the objective function of GA.

A. Error Distance

The measurement technique developed [15] measures the precision of approximate circuits. This method is termed as *Error Distance*. Error Distance is the arithmetic distance between two binary values i.e. the arithmetic distance between the predicted/required value and the obtained/output value. For example, let us consider a Full Adder, if the exact output of the Sum is 100101, but the circuit gives an output of 100100, then the error distance between the two values is said to be 1, similar to Hamming distance. But if the output was 101111" then the error distance would be 10. It can be seen that the value of error distance (ED) increases or decreases depending on the position of the incorrect bit. [15] defines ED with equation(3)

TABLE I
TRUTH TABLE FOR CONVENTIONAL FULL ADDER AND APPROXIMATIONS 1, 2 AND 3 [13]

Inputs			Accurate Outputs		Approximate Outputs					
A	B	C_{in}	Sum	C_{out}	Sum_1	C_{out1}	Sum_2	C_{out2}	Sum_3	C_{out3}
0	0	0	0	0	1	0	0	0	0	0
0	0	1	1	0	1	0	1	0	0	0
0	1	0	1	0	0	1	0	0	1	0
0	1	1	0	1	0	1	1	0	1	0
1	0	0	1	0	1	0	0	1	0	1
1	0	1	0	1	0	1	0	1	0	1
1	1	0	0	1	0	1	0	1	1	1
1	1	1	1	1	0	1	1	1	1	1

$$ED(a,b) = |a-b| = |\Sigma_j a[i] * 2^i - \Sigma_i b[j] * 2^j| \quad (3)$$

For a non-deterministic application, the output is probabilistic and usually follows a set distribution for a given input a_i. In this case ED is defined as the weighted average of all possible outputs to the nominal output. Assume that for a given input, the output has a nominal value b, but it can take any value given in a set of vectors b_j (1<j<r) [15].The ED of the output, in such cases, is given by equation(4) as shown in [15]

$$d_i = \Sigma_j ED(b_j, b) * p_j \quad (4)$$

Where, p_j is the output probability of b_j (1<j<r)

B. Mean Error Distance

Mean Error Distance is used in cases where the inputs are probabilistic and thus each input occurs only at a particular probability. The mean error distance (MED) of a circuit is defined as the mean value of the EDs of all possible outputs for each input [15] . Assume that the input is a set of vectors a_i and that each vector occurs with a probability of q_i [15]. Then, the MED, d_m of the circuit is given by

$$d_m = \Sigma d_i * q_i \quad (5)$$

Where, d_i is the ED of outputs for input a_i which can be computed by (3)

As an example assume a NOT gate, the probability of the input to the NOT gate being 1 is 0.7 and if the output of the NOT gate is 1 even if the input is 1. In this case the MED is given by ED multiplied by the probability (0.7) which is equal to 0.7 as the ED is 1 in this case. In case of multiple outputs, the ED is individually calculated and then multiplied with individual probabilities and after the values are calculated, all the values are summed with each other to obtain the final value.

C. Normalized Error Distance

MED increases with increase in the number of lower bits. It is therefore biased to use MED to compare between two adders with different lower bits as the maximum value of error that can be effectively reached has also changed [15]. To overcome this limitation *normalized error distance* (NED) is used. NED is defined as

$$d_n = d_m / D \quad (6)$$

Where, d_m is the MED and D is the maximum value of the error. The maximum value is usually 2n for n lower bits [15]. We use the fitness function used in [16]. It requires multiplication of the number of combinations possible in a truth table multiplied by the number of outputs to get the maximum fitness value. Hence, we can denote the formula as given in equation (4).

V. ENERGY EVALUATION

In this paper, we define error based on the accuracy of the output, we do not consider delay, although all the approximate adders here perform much better in terms of delay and power when compared to the fully exact adder. The Dynamic Power Dissipation of the system is given by equation (7) and the static power dissipation is given by equation (8)

$$P_D = C_{pd} * V_{CC}^2 * f_I * N_{SW} \quad (7)$$

Where,

P_D is the dynamic power consumption

V_{CC} is the supply voltage

f_I is the input signal frequency

N_{SW} is the number of bits switching

C_{pd} = dynamic power-dissipation capacitance The total static power consumption of this device can be given as:

$$P_S = V_{CC} * I_{leakage} \quad (8)$$

Where,

P_S is the static power consumption

V_{CC} is the supply voltage

$I_{leakage}$ is the current into a device (sum of leakage currents)

From, equation 7 and 8, we can derive equation 9,

$$P_T = \Sigma(P_D + P_S) \quad (9)$$

```
1: procedure GENETIC ALGORITHM(a, b)        ▷ population consisting of a and b
2:     while a&b > error constraint do
3:         population allocation
4:         tbb_evolve()
5:         tbb_postselect()
6:         fitness()
7:         elitism()
8:         while maxfit<desired fitness do
9:             migration() b/w populations
10:        end while
11:    end while
12:    return c & d        ▷ c & d are the latest generation
13: end procedure
```

Fig. 5. Parallel Master Slave GA

The next factor we consider is the average propagation delay of an adder which is given in equation 10. and for the last factor we used MED as described in equation (5).

$$t_p = \frac{t_{PHL} + t_{PLH}}{2} \tag{10}$$

Where,

t_{PHL} is the time delay from High to Low

t_{PHL} is the time delay from Low to High

VI. Implementation

We implement a Genetic Algorithm with three different modules: (1) Evolution of the circuit, (2) Selection of the Offspring, and (3) Maximum Fitness Calculation as shown in Figure 5. Each module consists of a for loop which loops through Look up Tables (LUT) assigning different functionalities to each LUT based on a random number between zero to four each corresponding to AMA 1-4. The three for loops were parallelized using Intel TBB [17] $parallel\_for$ loop, also vectors were used wherever possible to give flexibility to the program. We also used OpenMP in conjunction with the Intel TBB library to test out how it affects performance relative to only using TBB, the outcome of this is illustrated in the results section. The main program was already designed by [16], we modified it to change the operation of the Genetic Algorithm extensively. The original Genetic Algorithm in [16] could only design MUX decoders, we changed the functionality of this Algorithm by modifying the LUTs used in the original program to map the different Approximate Adders as single LUTs. The LUTs considered in this program can handle 3 inputs and 2 outputs, analogous to a full adder. The LUTs are lowest grain level the program could go. The Interconnection Mutation was switched off for these Genetic Operations as this resulted in the 2 outputs being mapped to the wrong adder occasionally. The implemented algorithm produced designs which outperform the other adders,for e.g. it obtains a fitness value of up to 40 for a maximum fitness value of 48.

The program used a hierarchical approach; first the Individuals containing CLBs were designed, each CLB has a vector of LUTs. The individuals, CLBs and LUTs are initialized as classes as shown [16]. The LUT class contains a function called $CalculateOutput$ which takes in the input vector to calculate the output based on the functionality chosen, this is done for each individual. The vector of individuals are initialized based on the initial population given in the input file.

VII. Experimental Setup & Results

Multi-bit approximate adder circuits are designed using the proposed methodology. Other benchmarks can also be designed if the multiple versions of approximate designs are available. For the benchmark utilized, the maximum fitness of the circuit is dependent on the number of inputs n and is equal to 2^{n+1}. The Genetic Algorithm was executed for a maximum of 5000 generations with a population size of 20 circuits consisting of randomly instantiated individuals. The GA selects a random set of individuals (parents) from the population after which tbb_evolve is invoked which evolves the circuit in parallel, which is followed by $tbb\_postselect$ and $maxfit$ which are the processes for selecting the offspring and calculating the maximum fitness respectively.

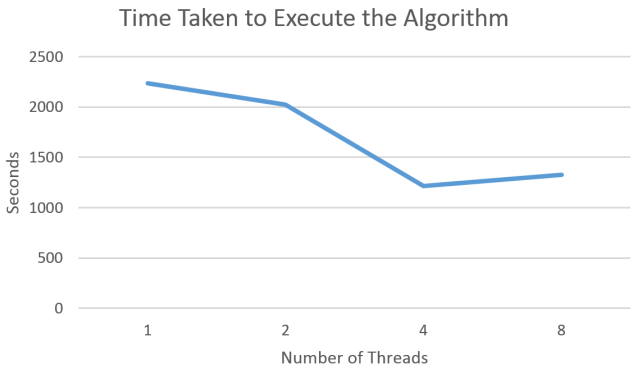

Fig. 6. Performance without OpenMP

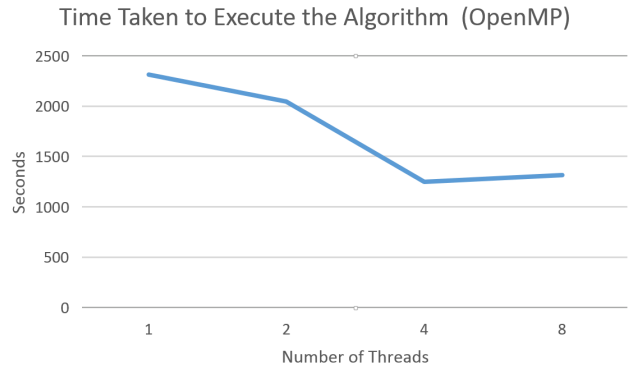

Fig. 7. Performance with OpenMP included

One of the important observations was that the GA took as much time to evolve as to calculate the Fitness, sometimes the time to evolve was greater than the time to calculate the Fitness. This is a context-dependent observation as the time to evolve or to calculate the fitness values depends on the complexity of the calculation within each function. The most-intriguing result obtained was for a 2-bit adder including the carry with a fitness value of 40 out of the maximum fitness value of 48, also the MSBs were the least affected by the combination of approximate adders. Additionally, if a redundant adder is added then the fitness could be improved to 45, this is because the third adder could be better in certain cases, this could be useful when the outputdoes not meet desired application criteria. Even when a redundant adder is used or swapped with the current adder component, the power consumption would be less than that required for an accurate adder. The power savings may extend up to 63.8 % when compared to cases when an Accurate Adder is used in ideal conditions. For example if we consider a 2 bit adder composed entirely of Accurate Adders the total power consumption would be 517.9 nW as shown in Table II. In the same scenario if two approximate adders are to be used for example the AMA3 adder, then the power consumption would be 187.552 nW and if we add another redundant AMA4 adder, then the total power consumption would be 284.434 nW, which is still much lesser than the combined power consumption of the 2-bit Accurate Adders.

Execution times for the design of approximate adders with and without OpenMP are shown in Figures 6 and 7 respectively. A speed-up of about 1.61-fold using 4 threads is achieved on a quad-core processor, which is far less than linear. A speedup of at least 2 on four threads would have been promising. We believe this could be possible with the introduction of larger Generation Gap in *Delayed Elitism*.When mutation was introduced in interconnections between LUTs, the developed circuit did not match the required configuration; the interconnections are important, as in a multiple bit adder, it is required for single bit adders to be in sequence so that the carry is propagated appropriately. The power and performance numbers for individual approximate adder designs are listed in Table II. These numbers are obtained by HSPICE simulations.

TABLE II
PROPAGATION DELAY AND POWER CONSUMPTION FOR APPROXIMATE ADDERS

Adder	Sum Worst(pS)	Carry Worst(pS)	Power (nW)
CMA	58.09	148	258.9419
AMA1	37.54	49.04	242.408
AMA2	56.41	49.15	122.37
AMA3	59.39	51.09	93.776
AMA4	58.78	52.12	96.882

From Figure 6, we can see that the difference between the GA run in a uniprocessor environment takes up to 700 seconds more than if it was run in a parallel environment. The speed up in such an environment is up to 1.6, this is the maximum we could achieve, the paper [16] achieves a speedup

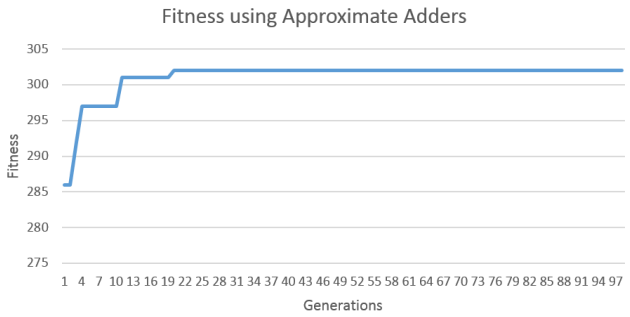

Fig. 8. Fitness Evolved when Approximate Adders are used to implement a 4-bit Adder

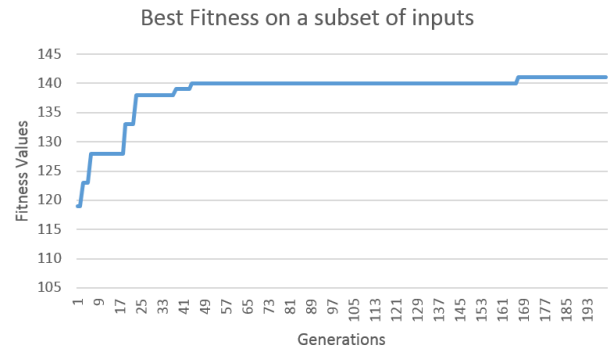

Fig. 9. Fitness Evolved when on a subset of inputs restricted to maximum fitness value of 220

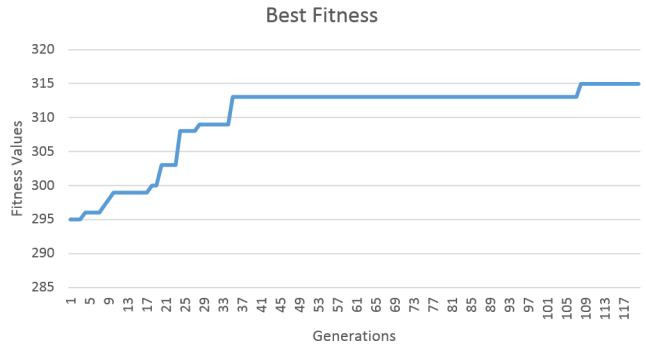

Fig. 10. Fitness Evolved when the number of gates was restricted to 16 for a 4 bit adder

of more than four for a 3:8 multiplexer, but this paper has been implemented using an Adder in mind which consists of larger sub-elements and the order in which connections take place are important, hence parallelism cannot be implemented as frequently. Additionally, the results obtained by including OpenMP are shown in Figure 7. It can be observed that there is only a minimal increase in performance by using OpenMP.

The fitness of evolved 4-bit Adder circuit is shown in Figure 8. It can be seen that the GA quickly converges to a fitness value of 303 out of a maximum fitness value of 424. In this case, the fitness evaluation takes up bulk of the execution time. As approximate designs are targeted in this paper, the adders

are also evolved using a subset of the input space as shown in Figure 9. In this case, the output is only calculated for 40% of the input space. Another attempt to restrict the design space of GA was made by restricting the total number of gates to 16. The results for this case are shown in Figure 10. It can be observed that the highest fitness value is obtained in this case. Thus, the solution space can be explored more extensively using the proposed techniques.

VIII. Conclusion and Future Work

This paper indicates benefit for parallelizing the GA exploration of creative cascaded circuits such as Adders where the current stage is heavily-dependent on the previous stage. Although we were able to achieve a modest speed-up of 1.61, in most cases, when the population size is increased then the effects of parallelization tends to be more pronounced.

Future work could include implementation of $DelayedElitism$ which could improve the avenues where parallelism could be improved, in case of $DelayedElitism$ the $parallel\_invoke$ method in TBB could be used to run fitness evaluation function and Evolution in parallel, this could improve the design-time significantly. Also, the GA could be modified to design larger circuits than the ones currently designed, these could then be implemented in chips where manual intervention for hardware faults is not possible, in such cases a GA could be remotely used to reconfigure the circuits when the systems aren't functioning appropriately.

References

[1] R. A. Ashraf, A. Alzahrani, and R. F. DeMara, "Extending modular redundancy to NTV: Costs and limits of resiliency at reduced supply voltage," *Second Workshop on Near-threshold Computing (WNTC), held in conjunction with ISCA 2014*, p. 6, 2014.

[2] N. Imran, R. A. Ashraf, J. Lee, and R. F. DeMara, "Activity-based resource allocation for motion estimation engines," *Journal of Circuits, Systems, and Computers*, vol. 24, no. 01, 2015.

[3] K. Zhang, R. DeMara, and C. Sharma, "Consensus-based evaluation for fault isolation and on-line evolutionary regeneration," in *Evolvable Systems: From Biology to Hardware* (J. Moreno, J. Madrenas, and J. Cosp, eds.), vol. 3637 of *Lecture Notes in Computer Science*, pp. 12–24, Springer Berlin Heidelberg, 2005.

[4] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *Test Symposium (ETS), 2013 18th IEEE European*, pp. 1–6, IEEE, 2013.

[5] J. F. Miller and P. Thomson, "Cartesian genetic programming," in *Genetic Programming*, pp. 121–132, Springer, 2000.

[6] X. Shengjun, G. Shaoyong, and B. Dongling, "The analysis and research of parallel genetic algorithm," in *Wireless Communications, Networking and Mobile Computing, 2008. WiCOM'08. 4th International Conference on*, pp. 1–4, IEEE, 2008.

[7] J. Levenick, "Showing the way: a review of the second edition of holland's adaptation in natural and artificial systems," 1998.

[8] Z. Vasicek and L. Sekanina, "Evolutionary design of approximate multipliers under different error metrics," in *Design and Diagnostics of Electronic Circuits & Systems, 17th International Symposium on*, pp. 135–140, IEEE, 2014.

[9] Z. Konfrst, "Parallel genetic algorithms: Advances, computing trends, applications and perspectives," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, p. 162, IEEE, 2004.

[10] K. Hasani, S. A. Kravchenko, and F. Werner, "Simulated annealing and genetic algorithms for the two-machine scheduling problem with a single server," *International Journal of Production Research*, no. ahead-of-print, pp. 1–15, 2014.

[11] H. Muhlenbein, "Evolution in time and space-the parallel genetic algorithm," in *Foundations of genetic algorithms*, Citeseer, 1991.

[12] V. S. Gordon and D. Whitley, "Serial and parallel genetic algorithms as function optimizers," in *ICGA*, pp. 177–183, 1993.

[13] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, "Impact: imprecise adders for low-power approximate computing," in *Proceedings of the 17th IEEE/ACM international symposium on Low-power electronics and design*, pp. 409–414, IEEE Press, 2011.

[14] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 32, no. 1, pp. 124–137, 2013.

[15] Z. Yang, A. Jain, J. Liang, J. Han, and F. Lombardi, "Approximate xor/xnor-based adders for inexact computing," in *submitted to IEEE Conf. on Nanotechnology*, 2013.

[16] R. A. Ashraf, F. Luna, D. Dechev, and R. F. DeMara, "Designing digital circuits for fpgas using parallel genetic algorithms (wip)," in *Proceedings of the 2012 Symposium on Theory of Modeling and Simulation-DEVS Integrative M&S Symposium*, p. 15, Society for Computer Simulation International, 2012.

[17] C. Pheatt, "Intel threading building blocks," *J. Comput. Sci. Coll.*, vol. 23, pp. 298–298, Apr. 2008.