PROPOSTA DE IMPLEMENTAÇÃO PARALELA DE ALGORITMO GENÉTICO EM FPGA

Matheus F. Torquato*, Marcelo A. C. Fernandes*

*Departamento de Engenharia da Computação e Automação - DCA
Centro de Tecnologia - CT
Universidade Federal do Rio Grande do Norte - UFRN
Natal, Brasil

Emails: matheusft@dca.ufrn.br, mfernandes@dca.ufrn.br

Abstract— This paper proposes a parallel implementation of a genetic algorithm (GA) on field-programmable gate array (FPGA). Results associated with the processing time and area occupancy (in FPGA) for various population size are analyzed. Studies concerning the accuracy of the GA response for the optimization problem using the square function, were also analyzed for the hardware implementation. The project was developed using the System Generator software (Xilinx development platform) and the Virtex-6 xc6vcx240t 1ff1156 FPGA.

Keywords— FPGA, Genetic Algorithm, Hardware, Embedded system.

Resumo— Este artigo possui como objetivo o desenvolvimento de um protótipo associado a uma implementação paralela de um algoritmo genético em FPGA (*Field-programmable gate array*). Resultados associados com o tempo de processamento e a área ocupada para vários tamanhos de população foram analisados. Estudos relativos à precisão da resposta do algoritmo genético para o problema de otimização de uma função quadrática também foram analisados para implementação em hardware. Todo projeto foi desenvolvido utilizando a plataforma de desenvolvimento *System Generator* da Xilinx tendo como FPGA alvo um Virtex 6 xc6vcx240t-1ff1156.

Palavras-chave— FPGA, Algoritmo Genético, Hardware, Sistemas Embarcados.

1 Introdução

Nos últimos anos, a exigência por velocidades de processamento cada vez maiores é impulsionada pela demanda de aplicações que necessitam fornecer soluções com respostas mais rápidas que milissegundos. Uma forma de atacar essas demandas de desempenho é utilizando técnicas de paralelização de algoritmos. Ao aliar a implementação em hardware à paralelização de algoritmos, as duas formas de aceleração de um algoritmo genético segundo (Mengxu and Bin, 2015), tem- se uma solução satisfatória para aplicações de alto desempenho que demandam respostas em tempo real. Atualmente, estruturas de hardware reconfigurável e programável como *Field-programmable Gate Arrays* (FPGAs) apresentam uma solução viável para essa situação. Como afirmado em (de Souza and Fernandes, 2014), essa arquitetura de hardware é considerada uma das mais utilizadas na atualidade pois pode fornecer desempenho semelhante aos circuitos integrados de aplicação específica (*Application-Specific Integrated Circuit* - ASIC) com a vantagem da utilização de prototipagem rápida. Além disso os FPGAs ainda apresentam bons índices quando se analisa o consumo energético e ocupação em área (Hauck and DeHon, 2010)

Algoritmos genéticos são técnicas de busca direcionadas que procuram soluções tomando como inspiração o método da seleção natural (Deliparaschos et al., 2008). Esses algoritmos têm se mostrado um eficiente mecanismo de busca e são utilizados para explorar múltiplas regiões da solução de um problema com um significativo ganho em tempo de execução (Fernando et al., 2008). Implementações em hardware de AGs já foram descritos por vários autores com a principal finalidade de reduzir o tempo de execução. Como apresentado em (Zhu et al., 2007), as implementações em hardware podem reduzir o tempo de processamento por um fator de aproximadamente 50 vezes quando comparadas às implementações em software (*speedup*).

Nos trabalhos apresentados em (Thirer, 2012) e (Mengxu and Bin, 2015) uma implementação de AG em FPGA é desenvolvida. Em (Thirer, 2012) a implementação é direcionada para a resolução do jogo Sudoku e em (Mengxu and Bin, 2015) é desenvolvida uma implementação modular do AG. Todavia, estes trabalhos não apresentam detalhes de análise de ocupação e velocidade de processamento associada a construção do algoritmo em FPGA. Nos trabalhos apresentados em (Fernando et al., 2008) e (Chen et al., 2008), a implementação do AG em FPGA é mais genérica nas quais, os autores geram um módulo de propriedade intelectual (IP Core) de um AG de propósito geral com ênfase na facilidade para modificação de parâmetros, arquitetura e suporte à múltiplas funções de avaliação. Por outro lado, as arquiteturas apresentadas em (Fernando et al., 2008) e (Chen et al., 2008) comprometem bastante o *speedup* (apenas em torno de 5 ×) devido a serialização da implementação do AG. Já em (Chen et al., 2008) foi desenvolvido um soft IP core em C++ na construção do AG. É de conhecimento na literatura que existe um *tradeoff* en-

tre desempenho×flexibilidade e os autores em (Fernando et al., 2008) e (Chen et al., 2008) optaram na flexibilização da implementação reduzindo o desempenho do IP core em termos de *speedup*.

Assim, este trabalho tem como objetivo apresentar o desenvolvimento de uma implementação paralela de um AG em hardware. Todos os detalhes associados a implementação e uma análise de desempenho em hardware para vários tamanhos de população são apresentados. É importante observar que a arquitetura proposta neste trabalho busca otimizar o desempenho e reduzir o tempo de processamento do AG objetivando sua viabilidade em aplicações de tempo real como robótica, sistemas com tactile internet, sistemas críticos, entre outros.

2 Algoritmos Genéticos

Os AGs são utilizados para resolver problemas de busca e otimização nos quais, uma solução ótima pode ser encontrada utilizando um processo iterativo cuja a busca se dá a partir de uma população inicial, que combinando os melhores representantes, obtém-se uma nova população, que substitui a anterior (Holland, 1975).

O AG é um algoritmo iterativo que é iniciado a partir de uma população de N indivíduos, criados aleatoriamente onde N é par para facilitar a implementação. Em cada k-ésima iteração, também chamada de geração, os N indivíduos são avaliados, selecionados, recombinados e mutados para formar uma nova população também de N indivíduos. A nova população então é utilizada como entrada para a próxima iteração (geração) do algoritmo, e esse procedimento de atualização dos valores da geração anterior é repetido K vezes, onde K é número de gerações do algoritmo.

O Algoritmo 1 apresenta o pseudo-código do funcionamento do AG. Esse código detalha todas as variáveis e procedimentos que serão utilizados na implementação a ser apresentada na Seção 3. A variável $x_j[m](k)$ representa o j-ésimo indivíduo de m bits na k-ésima geração e $\mathbf{X}[m](k)$ é um vetor que armazena todos os N indivíduos, ou seja,

$$\mathbf{X}[m](k) = \begin{bmatrix} x_1[m](k) \\ \vdots \\ x_N[m](k) \end{bmatrix}. \quad (1)$$

Após o processo de inicialização, a função de avaliação, chamada de *FA* (Linha 4 do Algoritmo 1), calcula a aptidão dos N indivíduos $x_j[m](k)$ da população. Essa operação é aplicada para todos os indivíduos e resulta em um respectivo valor $y_j[b](k)$ para cada j-ésimo indivíduo, onde b é o número de bits que representa o valor de aptidão. Quanto melhor o valor $y_j[b](k)$ do indivíduo $x_j[m](k)$, maior a probabilidade do mesmo prosseguir nas novas gerações. Os valores de aptidão de todos os N indivíduos são armazenados em

$$\mathbf{Y}[b](k) = \begin{bmatrix} y_1[b](k) \\ \vdots \\ y_N[b](k) \end{bmatrix}. \quad (2)$$

Algoritmo 1 Algoritmo Genético

1: Inicializa $\mathbf{X}[m](k)$ com indivíduos aleatórios
2: **para** $k \leftarrow 1$ **até** K **faça**
3: **para** $j \leftarrow 1$ **até** N **faça**
4: $y_j[a](k) \leftarrow FA\left(x_j[m](k)\right)$
5: **fim para**
6: **para** $j \leftarrow 1$ **até** N **faça**
7: $w_j[m](k) \leftarrow FS\left(\mathbf{Y}[b](k), \mathbf{X}[m](k)\right)$
8: **fim para**
9: **para** $i \leftarrow 1$ **até** $N/2$ **faça**
10: $\begin{bmatrix} z_{2i-1}[m](k) \\ z_{2i}[m](k) \end{bmatrix} \leftarrow FC\left(\begin{bmatrix} w_{2i-1}[m](k) \\ w_{2i}[m](k) \end{bmatrix}\right)$
11: **fim para**
12: **para** $v \leftarrow 1$ **até** P **faça**
13: $x_v[m](k) \leftarrow FM\left(z_v[m](k)\right)$
14: **fim para**
15: **fim para**

Após o cálculo da aptidão de cada j-ésimo indivíduo da k-ésima geração é realizada a operação de seleção. No AG o propósito da seleção é destacar os indivíduos $x_j[m](k)$ com seus respectivos valores de função de avaliação, $y_j[b](k)$, com o intuito de gerar populações futuras mais aptas. Existe na literatura uma grande variedade de métodos de seleção e entre eles podem ser citados: método da seleção por ranking, torneio, roleta e elitismo. O método da seleção por torneio que foi utilizado nessa implementação, é um dos mais utilizados (Noraini and Geraghty, 2011) e faz uma competição entre dois ou mais indivíduos escolhidos aleatoriamente a partir da população armazenada em $\mathbf{X}[m](k)$. Essa competição consiste em comparar as aptidões, $y_j[b](k)$, dos indivíduos participantes e o indivíduo que possui o melhor valor na função de avaliação prossegue no algoritmo para passar o seus genes adiante. O função de seleção, chamada aqui de *FS* (Linha 7 do Algoritmo 1), possui como entrada os vetores $\mathbf{Y}[b](k)$ e $\mathbf{X}[m](k)$ da k-ésima geração e como saída a variável $w_j[m](k)$ que pode assumir o valor de qualquer um dos N indivíduos armazenados em $\mathbf{X}[m](k)$. Todos valores de $w_j[m](k)$ são agrupados em

$$\mathbf{W}[m](k) = \begin{bmatrix} w_1[m](k) \\ \vdots \\ w_N[m](k) \end{bmatrix} \quad (3)$$

para serem utilizados na etapa de cruzamento.

O processo de cruzamento da k-ésima geração ocorre após a seleção dos indivíduos mais aptos da população (armazenados em $\mathbf{W}[m](k)$) pela função de seleção e tem como objetivo originar novos indivíduos que após a etapa de mutação irão

compor a próxima geração do AG (atualização do vetor $\mathbf{X}[m](k)$). Existem várias técnicas de cruzamento apresentadas na literatura e a estratégia adotada nesta implementação foi o cruzamento com um ponto de corte. A operação de cruzamento, chamada aqui de *FC* (Linha 10 do Algoritmo 1), possui como entrada pares de elementos do vetor $\mathbf{W}[m](k)$ da k-ésima geração e como saída pares do vetor

$$\mathbf{Z}[m](k) = \begin{bmatrix} z_1[m](k) \\ \vdots \\ z_N[m](k) \end{bmatrix} \tag{4}$$

que armazenam os indivíduos após o cruzamento, ou seja, os filhos da k-ésima geração.

A última etapa do AG é caracterizada pela operação de mutação que altera o valor de um grupo P de indivíduos, a fim de permitir uma maior diversidade na população evitando a estabilização da solução em mínimos locais. A taxa de mutação, TM, é o parâmetro que controla a quantidade de indivíduos a sofrerem mutação. Normalmente, TM varia em torno de $0,1\%$ e o valor de P pode ser facilmente calculado pela expressão

$$P = \lceil N \times TM \rceil. \tag{5}$$

A operação de mutação, chamada aqui de *FM*, é apresentada na Linha 13 do Algoritmo 1.

3 Descrição do Projeto

A Figura 1 apresenta uma arquitetura geral da implementação em hardware do AG proposto. Todo o algoritmo foi desenvolvido utilizando uma arquitetura paralela com o intuito de acelerar a velocidade de processamento, aproveitando os recursos disponíveis em hardware, semelhantemente a (Nedjah and de Macedo Mourelle, 2007). A estrutura, ilustrada na Figura 1, permite a visualização de uma população de N indivíduos de m bits no qual $x_j[m](k)$ representa o j-ésimo indivíduo da população na k-ésima geração, segundo o Algoritmo 1. Cada j-ésimo individuo $x_j[m](k)$ é armazenado em um registrador de m bits, chamado aqui de RXj onde seu valor é atualizado a cada k-ésima geração pela nova população de N indivíduos gerada nos processos de seleção, cruzamento e mutação.

Todos os valores aleatórios foram criados por geradores de números pseudo-aleatórios baseados em registradores de deslocamento do tipo Linear Feedback Shift Register (LFSR) (Deliparaschos et al., 2008) e (Goresky and Klapper, 2006). Foram utilizados LFSR's independentes de 32 bits baseados no polinômio $r^{32} + r^{22} + r^2 + 1$ (Goresky and Klapper, 2006). Cada gerador é caracterizado como LFSRCClj onde CC, l e j são rótulos para sua posição no circuito. A cada k-ésima geração do AG uma variável aleatória de 32 bits, chamada aqui de $\mathrm{CC}r_{lj}[32](k)$, é produzida. Para evitar uma mesma sequencia de valores, cada gerador LFSRCClj possui um valor inicial de 32 bits diferente, chamado $\mathrm{CCseed}_{lj}[32]$.

A proposta é formada por quatro módulos principais chamados de aqui Módulo da Função de Avaliação (MFA), Módulo de Seleção (MS), Módulo de Cruzamento (MC) e Módulo de Mutação (MM). Cada módulo possui implementações específicas que serão detalhadas nas seções seguintes.

3.1 Módulo da Função de Avaliação - MFA

Este módulo tem como objetivo calcular o valor da aptidão de cada j-ésimo individuo a partir de uma função de avaliação $f(\cdot)$ que é específica para cada tipo de problema. A estrutura proposta possui N módulos do tipo MFA e cada j-ésimo módulo chamado, aqui de MFAj, é associado a um individuo $x_j[m](k)$ e gera como saída em cada k-ésima geração um valor de aptidão em ponto fixo expresso por

$$y_j[a.b](k) = \mathrm{MFA}j(x_j[m](k)), \tag{6}$$

onde a representa o número total de bits e b o tamanho da parte fracionaria (equivalente a linha 4 do Algoritmo 1). Cada j-ésimo MFAj é implementado através de uma RAM, chamada aqui

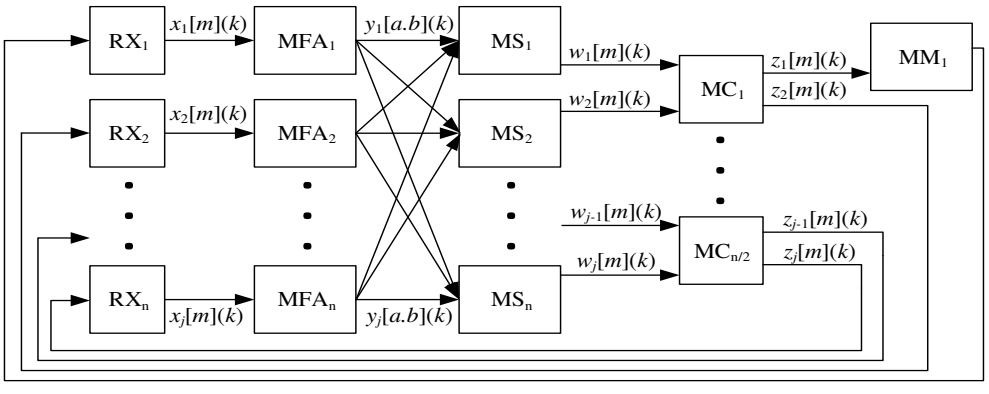

Figura 1: Arquitetura geral da proposta de implementação paralela de algoritmo.

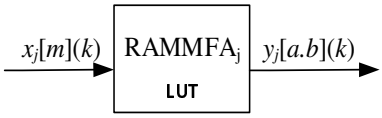

Figura 2: Arquitetura do j-ésimo módulo de Função de Avaliação (MFAj).

de RAMMFAj, funcionando como uma *lookup table* (LUT), de tamanho $m \times a$ bits, onde m é a profundidade e a é o tamanho de cada palavra armazenada. A Figura 2 detalha o funcionamento do j-ésimo MFA.

3.2 Módulo de Seleção - MS

O módulo de seleção (MS) implementa o método de seleção por torneio, detalhado na Seção 2, com competição entre dois indivíduos. Semelhante ao MFA existem N MS responsáveis por selecionar um grupo de N indivíduos. Como detalhado na Figura 3, cada j-ésimo MS, chamado aqui de MSj, possui como entrada os N valores de aptidã, $y_j[a.b](k)$, e os N indivíduos, $x_j[m](k)$, da k-ésima geração (equivalente a linha 7 do Algoritmo 1).

Cada j-ésimo MSj possui dois geradores aleatórios chamados de LFSRMS1j e LFSRMS2j. Além dos geradores aleatórios este módulo é formado por três multiplexadores de N entradas chamados aqui de MUXMS1j, MUXMS2j e MUXMS3j, um comparador de m bits, chamado de COMPMSj e três multiplexadores de duas entradas, chamados de MUXMS4j, MUXMS5j e MUXMS6j. Os multiplexadores MUXMS1j e MUXMS2j trabalham na seleção da melhor aptidão a partir do sinal de saída dos geradores LFSRMS1j (MS$r_{1j}[32](k)$) e LFSRMS2j (MS$r_{2j}[32](k)$), respectivamente. Como apresentado na Figura 3 o sinal de saída de cada gerador (MS$r_{1j}[32](k)$ e MS$r_{2j}[32](k)$) é truncado nos $\lceil log_2(N) \rceil$ bits mais significativos para se ajustar ao MUX. Finalmente, o MUXMS3j seleciona o indivíduo associado a melhor função a partir da saída do MUXMS6j que seleciona se o objetivo é maximizar ou minimizar a função de avaliação através da variável MAXMINMSj.

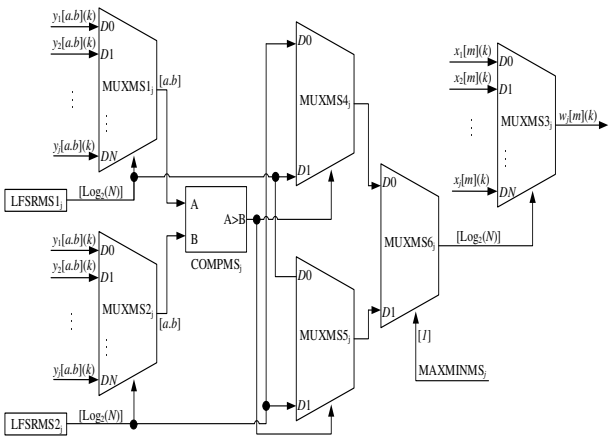

Figura 3: Arquitetura do j-ésimo módulo de seleção (MSj).

3.3 Módulo de Cruzamento - MC

A Figura 4 apresenta o circuito responsável pela etapa de cruzamento, chamado aqui de MCi (equivalente a linha 10 do Algoritmo 1). Existem $N/2$ MC's e cada i-ésimo circuito é formado por um MUX de $m+1$ entradas, chamado aqui de MUXMC1i, cujo objetivo é selecionar de forma aleatória um dos m possíveis pontos para o corte do cruzamento. A seleção de cada MUXMC1i é controlada pelo gerador aleatório LFSRMC1i onde seu sinal de saída, MC$r_{1i}[32](k)$, é truncado nos $\lceil log_2(m+1) \rceil$ bits mais significativos antes de entrar no seletor do MUX.

Em cada k-ésima geração, as duas entradas do módulo MCi, os pais $w_{2i-1}[m](k)$ e $w_{2i}[m](k)$, são divididas em cabeça

$$hw_{2i-1}[m](k) = \neg s_i[m](k) \wedge w_{2i-1}[m](k) \quad (7)$$

$$hw_{2i}[m](k) = s_i[m](k) \wedge w_{2i-1}[m](k) \quad (8)$$

e calda

$$tw_{2i-1}[m](k) = \neg s_i[m](k) \wedge w_{2i}[m](k), \quad (9)$$

$$tw_{2i}[m](k) = s_i[m](k) \wedge w_{2i}[m](k), \quad (10)$$

onde $s_i[m](k)$ é a saída do MUXMC1i. Após esta etapa, o cruzamento será realizado concatenando a cabeça do pai 1, $hw_{2i-1}[m](k)$, com a calda do pai 2, $tw_{2i}[m](k)$, e a cabeça do pai 2, $hw_{2i}[m](k)$, com a calda do pai 1,$tw_{2i-1}[m](k)$, dando origem assim, a dois novos filhos da nova população, ou seja,

$$z_{2i-1}[m](k) = hw_{2i-1}[m](k) \vee tw_{2i}[m](k) \quad (11)$$

e

$$z_{2i}[m](k) = hw_{2i}[m](k) \vee tw_{2i-1}[m](k). \quad (12)$$

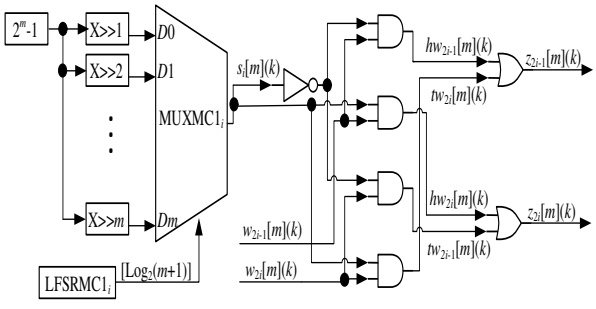

Figura 4: Arquitetura do i-ésimo módulo de cruzamento (MCi).

3.4 Módulo de Mutação - MM

Semelhante ao Algoritmo 1 na Linha 13 a operação de mutação será realizada em um grupo de P indivíduos (ver Equação 5), ou seja, existem P módulos de mutação e cada v-ésimo módulo, MMv, altera o indivíduo a ser mutado através de uma operação XOR com um número criado aleatoriamente por um gerador associado chamado de LFSRMM1v (Figura 5). A saída do v-ésimo, MMv, módulo em cada k-ésima geração pode ser expressa por

$$x_v[m](k) = (\neg z_v[m](k) \wedge \text{MM}r_v[m](k)) \vee (z_v[m](k) \wedge \neg\text{MM}r_v[m](k)) \quad (13)$$

onde MM$r_{1v}[m](k)$ representa os m bits mais significativos do número gerado por LFSRMM1v (MM$r_{1v}[32](k)$).

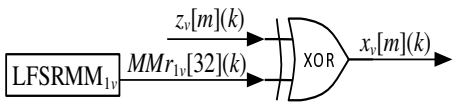

Figura 5: Arquitetura do v-ésimo módulo de seleção (MMv).

4 Resultados

Objetivando validar a proposta de implementação do AG em FPGA foram realizadas análises de síntese e funcionamento do algoritmo na otimização da função de avaliação x^2 para vários tamanhos de população. Todos os resultados foram obtidos utilizando a plataforma de desenvolvimento *Xlinx - System Generator* (*Xilinx System Generator User's Guide*, n.d.) e um FPGA Virtex 6 xc6vcx240t 1ff1156. O FPGA Virtex 6 utilizado possui 37.680 *slices* que agrupam 301.440 registradores (*flip-flops*), 150.720 células lógicas que podem ser utilizados para implementar funções lógicas ou memórias e 768 células de DSP com multiplicadores e acumuladores (*Xilinx System Generator User's Guide*, n.d.). Em todos os testes realizados as operações do AG são realizadas a uma taxa de amostragem, $R_g = \frac{1}{T_g}$ onde T_g é o tempo entre cada k-ésima geração.

A tabela 1 apresenta os resultados de síntese no FPGA alvo para vários tamanhos de população. Observa-se claramente, em todos os casos, que a taxa de amostragem, R_g, e área ocupada são parâmetros muito sensíveis ao tamanho da população. A área ocupada gasta em registradores, apresentada na segunda coluna, é devido ao armazenamento dos valores da população e dos geradores de número pseudo-aleatório, principalmente. Já a ocupação de células lógicas (LUTs), apresentada na terceira coluna, foi crescente e não linear com N. Este crescimento não linear é causado pelo pelo módulo de seleção (Subseção 3.2) que para cada j-ésimo módulo, MSj, possui três multiplexadores de N entradas (MUXMS1j, MUXMS2j e MUXMS3j). Baseado em (Chapman, 2014), cada célula lógica da Virtex 6 pode construir um MUX de 4 entradas de 1 bit, assim para montar um MUX de N entradas, são necessárias aproximadamente $\frac{N}{4}$ células lógicas totalizando um valor de aproximadamente $\frac{3N}{4}$ células para cada MSj (MUXMS4j, MUXMS5j e MUXMS6j não foram contabilizados). Como existem N módulos do tipo MS tem-se aproximadamente $\frac{3N^2}{4}$ células logicas para para cada bit do barramento de entrada do MUX. É importante observar que a implementação com 64 indivíduos não atinge a metade das células do FPGA (em torno de 45% do Virtex 6). Isso é um fato positivo para implementações com populações maiores.

Finalmente, a quarta coluna mostra as taxas de amostragem para cada valor de N e verifica-se uma redução da taxa, T_g, com o crescimento da população. Teoricamente se todos os módulos fossem independentes (específicos para cada indivíduo) essa redução não deveria acontecer, porem, observa-se que nos módulos de seleção, MSj, existe uma dependência entre o N indivíduos (devido ao compartilhamento de informação) causando uma junção no circuito (*join*) e com isto um aumento no tempo de processamento. Por outro lado, também observa-se que a taxa de redução não é linear o que favorece o circuito. Outro ponto importante a ser verificado é que mesmo com a redução, cada geração do AG de 64 indivíduos pode trabalhar a uma velocidade de $Tg \approx 42$ ns, em outra palavras, 24.000 gerações a cada 1 ms. Este resultado trás um impacto bastante significativo e viabiliza o AG em várias aplicações embarcadas de tempo real como em robótica, telecomunicações e outras.

Tabela 1: Resultados da síntese do algoritmo genetico no FPGA

N	Registradores (*flip-flops*)	Células Lógicas (LUTs)	R_g (MHz)
4	154	856(1%)	112.25
8	292	2.045(1%)	86.85
16	624	5.726(3%)	61.34
32	1.260	19.885(13%)	39.29
64	2.893	68.224(45%)	23.63

As Figuras 6 e 7 apresentam o funcionamento do circuito proposto para os casos com $N = 32$ e $N = 64$, respectivamente. Em ambas as figuras são mostrados os melhores valores de aptidão da população para cada k-ésima geração. Observa-se que em ambos os casos o algoritmo genético convergiu como esperado, validando desta forma a implementação em hardware. É importante enfatizar que a implementação proposta pode trabalhar

com diferentes funções de avaliação, alterando os valores armazenados em cada RAMMFAj associada a cada j-ésimo módulo de função de avaliação, MFAj.

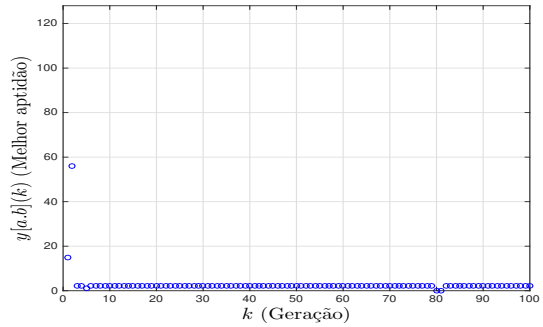

Figura 6: Melhor indivíduo de cada geração, $N = 32$.

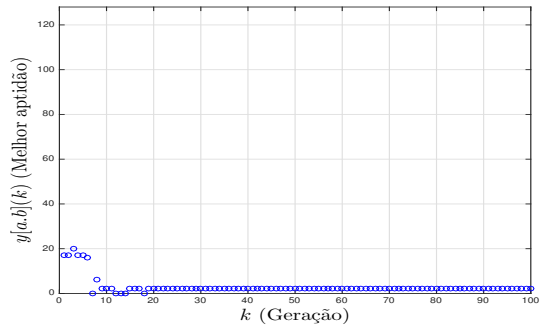

Figura 7: Melhor indivíduo de cada geração, $N = 64$.

5 Conclusões

Este trabalho apresentou uma proposta de implementação paralela para FPGA de algoritmo genético. Todos detalhes de implementação da proposta foram apresentados e analisados em termos de área ocupada e tempo de processamento. A estrutura proposta foi submetida a testes com dois tamanhos de população na otimização de uma função de avaliação quadrática. Os resultados obtidos são bastante significativos e apontam para novas possibilidades de utilização de algoritmos genéticos embarcados em hardware para aplicações de tempo real.

Referências

Chapman, K. (2014). Multiplexer design techniques for datapath performance with minimized routing resources, Application Note: Spartan-6 Family, Virtex-6 Family, 7 Series FPGAs.

Chen, P.-Y., Chen, R.-D., Chang, Y.-P., Shieh, L.-S. and Malki, H. A. (2008). Hardware implementation for a genetic algorithm, *Instrumentation and Measurement, IEEE Transactions on* **57**(4): 699–705.

de Souza, A. C. and Fernandes, M. A. (2014). Parallel fixed point implementation of a radial basis function network in an fpga, *Sensors* **14**(10): 18223–18243.

Deliparaschos, K., Doyamis, G. and Tzafestas, S. (2008). A parameterised genetic algorithm ip core: Fpga design, implementation and performance evaluation, *International Journal of Electronics* **95**(11): 1149–1166.

Fernando, P., Sankaran, H., Katkoori, S., Keymeulen, D., Stoica, A., Zebulum, R. and Rajeshuni, R. (2008). A customizable fpga ip core implementation of a general purpose genetic algorithm engine, *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, IEEE, pp. 1–8.

Goresky, M. and Klapper, A. (2006). Pseudonoise sequences based on algebraic feedback shift registers, *IEEE Transactions on Information Theory* **52**(4): 1649–1662.

Hauck, S. and DeHon, A. (2010). *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation (Systems on Silicon)*, Morgan Kaufmann.

Holland, J. H. (1975). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.*, U Michigan Press.

Mengxu, F. and Bin, T. (2015). Fpga implementation of an adaptive genetic algorithm, *Service Systems and Service Management (ICSSSM), 2015 12th International Conference on*, IEEE, pp. 1–5.

Nedjah, N. and de Macedo Mourelle, L. (2007). An efficient problem-independent hardware implementation of genetic algorithms, *Neurocomputing* **71**(1): 88–94.

Noraini, M. R. and Geraghty, J. (2011). Genetic algorithm performance with different selection strategies in solving tsp.

Thirer, N. (2012). About the fpga implementation of a genetic algorithm for solving sudoku puzzles, *Electrical & Electronics Engineers in Israel (IEEEI), 2012 IEEE 27th Convention of*, IEEE, pp. 1–3.

Xilinx System Generator User's Guide (n.d.). `www.xilinx.com`.

Zhu, Z., Mulvaney, D. J. and Chouliaras, V. A. (2007). Hardware implementation of a novel genetic algorithm, *Neurocomputing* **71**(1): 95–106.