PROPOSTA DE IMPLEMENTAÇÃO PARALELA DO ALGORITMO PSO PARA FPGA

Alexandre L. X. da Costa*, Caroline A. D. Silva*, Matheus F. Torquato*, Marcelo A. C. Fernandes*

*Departamento de Engenharia de Computação e Automação - DCA
Universidade Federal do Rio Grande do Norte (UFRN)
Caixa Postal 1524, 59078-970
Natal, RN, Brasil

Emails: alexluz321@gmail.com, carolads@gmail.com, matheusft@gmail.com, mfernandes@dca.ufrn.br

Abstract— This paper proposes a parallel implementation, on fixed point, of a Particle Swarm Optimization (PSO) algorithm on field-programmable gate array (FPGA). Results associated with the processing time and area occupancy (on FPGA) for various number of particles are analyzed. Studies concerning the accuracy of the PSO response for the optimization problem using the Rastrigin function, were also analyzed for the hardware implementation. The project was developed to the Virtex-6 xc6vcx240t 1ff1156 FPGA.

Keywords— Particle Swarm Optimization, FPGA

Resumo— Este artigo apresenta o desenvolvimento de um protótipo associado a uma implementação paralela, em ponto fixo, do algoritmo Otimização por Nuvem de Partículas (PSO, Particle Swarm Optimization) em um *Field-programmable gate array* (FPGA). Resultados associados com o tempo de processamento e a área de ocupação para vários tamanhos de número de partículas foram analisados. Estudos relativos ao funcionamento da PSO para o problema de otimização de uma função chamada de Rastrigin também foram analisados para implementação em hardware. Todo projeto foi desenvolvido para um FPGA Virtex 6 xc6vcx240t-1ff1156.

Palavras-chave— Otimização por Nuvem de Partículas, FPGA

1 Introdução

A resolução de problemas utilizando metaheurísticas tem sido objeto de estudo em diversas pesquisas na literatura e na indústria. Um dos aspectos mais importantes associados aos algoritmos metaheurísticos é a sua capacidade de fornecer uma solução aproximada, por vezes, a solução ótima, para um problema complexo dentro de um tempo de processamento razoável (Tsai et al., 2015; Rojas-Morales and Ureta, 2017; Soler-Dominguez and Kizys, 2017; Gogna and Tayal, 2013). Esta característica assegurou o uso difundido de metaheurísticas para resolver problemas em tempo real (Kamel et al., 2016; Kocadag et al., 2016; Pati et al., 2015; Samà et al., 2015; Souier et al., 2012; Fathi and Montazer, 2013).

A Otimização por Nuvem de Partículas, PSO, do Inglês *Particle Swarm Optimization*, é uma metaheurística inspirada na natureza relacionada à Inteligência de Enxame e à Computação Evolutiva. Ela requer apenas operadores matemáticos primitivos e é computacionalmente pouco custosa em termos de requisitos de memória e velocidade (Kennedy and Eberhart, 1995). Embora inicialmente projetada para resolver funções contínuas não-lineares, a PSO é usada atualmente para resolver vários outros tipos de problemas.

Paralelo às demandas associadas às metaheurísticas, existem também as demandas associadas ao processamento de grandes volumes de dados, criando novas áreas de conhecimento como *Big-Data* e *Mining of Massive Datasets*. Essa nova demanda mostra que algoritmos simples podem exigir um grande esforço computacional quando o volume de dados cresce exponencialmente (Leskovec and Ullman, 2014). Uma maneira de minimizar o problema é a paralelização de algoritmos, que surge como uma forma natural não só de reduzir o tempo de pesquisa, mas também de melhorar a qualidade das soluções fornecidas (Alba et al., 2012).

A melhoria de desempenho proporcionada pelas técnicas de paralelização pode ainda ser intensificada com uma implementação em plataformas de hardware de alto desempenho, como Processadores de Alto Desempenho, Circuitos Integrados para Aplicações Específicas, ASIC, do inglês *Application-specific Integrated Circuits*, e FPGAs, do inglês *Field Programmable Gate Array*. Entre essas, os FPGAs têm se destacado como uma alternativa que combina alto processamento, baixo consumo e baixo custo quando comparado às outras. Os FPGAs apresentam boas taxas quando analisam o consumo de energia e a eficiência de ocupação da área de silício (Hauck and DeHon, 2010). Além disso, como declarado em (de Souza and Fernandes, 2014), os FPGAs são atualmente a arquitetura mais utilizada para hardware reconfigurável, sendo capazes de fornecer desempenho similar a um ASIC, com a vantagem da utilização de prototipagem rápida.

Assim, o presente trabalho tem como objetivo apresentar uma implementação paralela do algo-

ritmo PSO em FPGA. É importante ressaltar que a arquitetura proposta busca otimizar o desempenho do PSO, reduzindo seu tempo de processamento, possibilitando seu uso em sistemas de tempo real com grande volume de dados ou severas restrições de tempo de processamento. Todos os detalhes associados à implementação são descritos na Seção 4, e as análises de desempenho do sistema proposto para diferentes dimensões e número de partículas são apresentados na Seção 5.

2 Trabalhos Relacionados

No artigo apresentado em (Calazan et al., 2014), os autores propõem melhorar o desempenho do algoritmo PSO implementando um coprocessador em hardware utilizando um FPGA Virtex 6 e obtiveram um tempo de processamento entre 1,93 ms e 2,08 ms para 4 e 10 partículas com a função Rastrigin em duas dimensões e 200 iterações. Isto equivale $\approx 10\,\mu s$ por iteração ou um *throughput* de $\approx$ 100 *Kilo-Sample per second, KBps*. Esta implementação ocupou 34% dos registradores, $\approx$ 102.000, e 94% das *LookUp Tables* (LUTs), $\approx$ 145.000, do FPGA Virtex 6. Já no trabalho apresentado em (Rathod and Thakker, 2014) é implementada uma proposta paralela com ponto flutuante em um FPGA Virtex V, onde os autores conseguiram um *throughput* de $\approx$ 7,8 MSps para 10 partículas com a função Rastrigin em 2 dimensões. Nesta implementação foi utilizado 1.534 dos registradores 25.355 LUTs da Virtex V.

O trabalho apresentado em (Arboleda et al., 2009) descreve uma implementação paralela em ponto flutuante com 32 bits da PSO em FPGA. Foram utilizadas 4 partículas e 2 dimensões para várias funções de otimização. Para o caso da função Rastrigin, a implementação proposta teve um *throughput* de $\approx$ 610KSps para um FPGA Virtex V. Também utilizando Virtex V, os autores do artigo (Muñoz et al., 2010) apresentaram uma proposta *full* paralela da PSO na qual obteve-se um *throughput* de $\approx$ 207KSps para um uma função Rastringin com 10 partículas e 6 dimensões. Nesta implementação foram consumidos 24.025 registradores, 73.881 LUTs e 100 multiplicadores, com DSP48E já embarcado no FPGA.

Finalmente, os trabalhos apresentados em (Li et al., 2011; Li et al., 2010) propõem uma implementação híbrida software/hardware utilizando o FPGA para acelerar certas partes da PSO para ser usado em sistemas embarcados. Através da técnica SOPC e pipeline, o hardware ficou responsável pela atualização da velocidade e posição da partícula, enquanto o software avalia a função de avaliação a ser usada. Com uma redução de 95% da quantidade de ciclos de CPU necessária e um *speedup* de aproximadamente 20 vezes, a abordagem proposta pelo autor se mostrou eficaz ao conseguir utilizar um chip de baixo custo e baixo consumo, Cyclone II, viabilizando seu uso em sistemas embarcados.

Diferentemente dos trabalhos apresentados na literatura a implementação proposta aqui foi desenvolvida em ponto fixo, reduzindo a ocupação e aumentando o *throughput*. Este esquema também permitiu aumentar a paralelização, possibilitando alcançar valores de *throughput* elevados quando comparados à literatura, como estão apresentados na Seção 5.

3 Otimização por Nuvem de Partículas

A Otimização por Nuvem de Partículas é uma metaheurística populacional estocástica que imita o comportamento social dos organismos naturais, como voos de pássaros e cardumes buscando um lugar com comida suficiente. Nesses enxames, o comportamento coordenado usando movimentos locais emerge sem controle central. De forma básica, uma nuvem consiste de um conjunto de N partículas voando em um espaço de busca D-dimensional. Cada partícula p é uma solução candidata ao problema em questão e possui seus próprios valores de posição e velocidade, que correspondem à direção do voo e ao passo da partícula.

O algoritmo PSO é iniciado gerando-se uma nuvem inicial de partículas, geralmente aleatórias. A cada iteração, todas as partículas são avaliadas de acordo com uma função de *fitness*, que deve indicar o quão próxima essa partícula está do valor ótimo, e têm alterados seus valores de velocidades e posição para que elas se aproximem ainda mais desse valor. A nova velocidade de cada partícula é determinada a partir de uma ponderação estocástica entre a velocidade atual, e as melhores posições encontradas pela própria partícula, *pbest* ou a melhor posição encontrada por toda a nuvem, *gbest*. A nova posição é determinada a partir da posição atual da partícula e sua nova velocidade. Desse modo, velocidade e posição de cada partícula são atualizadas de acordo com as equações

$$\begin{aligned} v_{i+1} &= w.v_i + c_1.r_1.(pbest - p_i) \\ &+ c_2.r_2.(gbest - p_i) \end{aligned} \quad (1)$$

e

$$x_{i+1} = x_i + v_{i+1} \quad (2)$$

em que p representa a partícula atual, v é a velocidade da partícula, r_1 e r_2 correspondem a números reais aleatórios no intervalo $[0, 1]$; w é o fator de inércia, isto é, o quanto a partícula confia em sua trajetória atual; c_1 é o fator de aprendizado cognitivo, representando o quanto ela confia em sua própria experiência; e c_2 o fator de aprendizado social, representando o quanto ela confia na experiência de seus vizinhos. De acordo com as novas posições, as melhores posições, *pbest* e *gbest*, são atualizadas. O processo continua até que o tempo

Tabela 1: Algoritmo do PSO

1: Inicialize $\mathbf{P}[m](1)$ com partículas aleatórias
2: **para** $k \leftarrow 1$ **até** K **faça**
3: **para** $j \leftarrow 1$ **até** N **faça**
4: $p_j F(k) \leftarrow Fitness\,(p_j[m](k))$
5: **se** $p_j F(k) \leq pbest_j F$ **então**
6: $pbest_j \leftarrow p_j[m](k)$
7: **fim se**
8: **fim para**
9: $gbest \leftarrow min\,(min\,(\mathbf{S}[m](k))\,, gbest)$
10: **para** $j \leftarrow 1$ **até** N **faça**
11: **para** $i \leftarrow 1$ **até** d **faça**
12: $v_{ji}(k+1) \leftarrow Velocidade(p_j[m](k), i)$
13: $p_j Pos_i(k+1) \leftarrow p_j Pos_i(k) + v_{ji}(k+1)$
14: **fim para**
15: **fim para**
16: **fim para**
17: Retorne *gbest*

limite seja atingido ou a solução seja encontrada.

O algoritmo apresentado na Tabela 1 resume a operação básica da PSO. A função *Fitness*, mostrada na linha 4, corresponde a função objetivo do problema a ser tratado; e a função *Velocidade*, mostrada na linha 12, corresponde às operações mostradas na Equação (1). As variáveis K, N e d correspondem, respectivamente, ao número de iterações do algoritmo, ao número de partículas da nuvem e à dimensão da partícula. A dimensão da partícula corresponde geralmente ao número de variáveis do problema. Cada partícula é representada por $p_j[m](k)$, em que j representa o índice dela na nuvem, m o número de bits de sua representação e k indica a iteração atual. A nuvem é representada pelo vetor $\mathbf{P}[m](k)$. $p_j F(k)$ indica o valor de *fitness* da partícula j na iteração k e $p_j Pos_i(k)$ indica o valor da i-ésima variável da partícula.

Assim como qualquer algoritmo de inteligência de enxame, os agentes trocam informações para compartilhar experiências sobre a busca realizada. O comportamento de todo o sistema emerge da interação desses agentes simples. Na PSO, a informação compartilhada através de seus agentes, as partículas, é composta da melhor solução global, *gbest*. Essas interações garantem a simplicidade de implementação e evitam que o algoritmo perca desempenho por sua característica estocástica.

4 Arquitetura de Hardware Proposta

O sistema proposto foi desenvolvido utilizando uma arquitetura paralela com o intuito de acelerar a velocidade de processamento, aproveitando os recursos disponíveis em hardware. Essa arquitetura realiza paralelamente a movimentação das partículas da nuvem, e depois as une em um mesmo processo para as operações em comum. Tal abordagem permite não só a redução do tempo de processamento, quanto possibilita que, pelos diversos processamentos simultâneos, o algoritmo atinja regiões diversificadas do espaço de busca.

A Figura 1 apresenta uma arquitetura geral da implementação em hardware do algoritmo PSO proposta. Quatro blocos principais estão destacados nesta estrutura: o módulo de partículas, MP, o módulo de comparação, MC, o módulo gBest, MgBest e, por último, o registrador MgBestFitness. Nessa implementação, um conjunto de N partículas de dimensão d é otimizado durante K gerações.

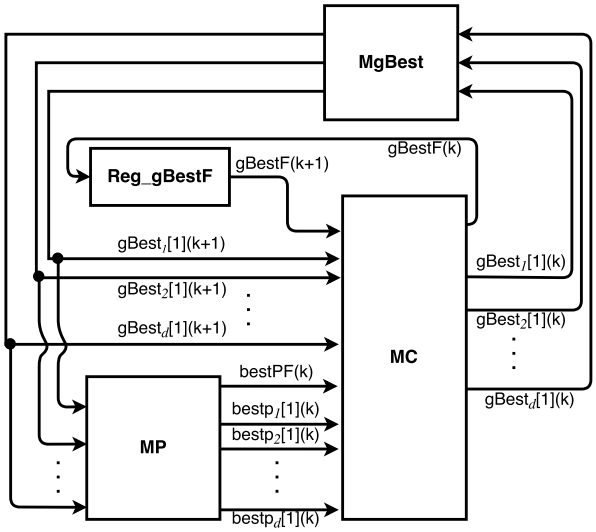

Figura 1: Arquitetura geral da proposta de implementação paralela de algoritmo.

O registrador MgBestFitness é utilizado para guardar o valor de *fitness* das coordenadas gBest, que é atualizado a cada geração, caso surja um melhor valor oriundo das partículas do MP, conforme descrito na Linha 9 do algoritmo descrito na Tabela 1. Os módulos mencionados anteriormente e exibidos na Figura 1 serão detalhados nas próximas seções.

4.1 Módulo de Comparação

A implementação do Módulo de Comparação, MC, é ilustrado em detalhes na figura 2, e tem como função comparar duas posições e seus respectivos valores de *fitness*. Logo, as entradas desse módulo são os dois valores de *fitness* a serem comparados, $bestpBestF$ e $gBestF$, mais os d pares de coordenadas, ou seja, um conjunto de coordenada para cada partícula a ser comparada, onde d é a dimensão da partícula.

Os $d+1$ multiplexadores de duas entradas são responsáveis por passar adiante as informações referentes à partícula com o menor valor de *fitness* e suas coordenadas. As saídas dos multiplexadores,

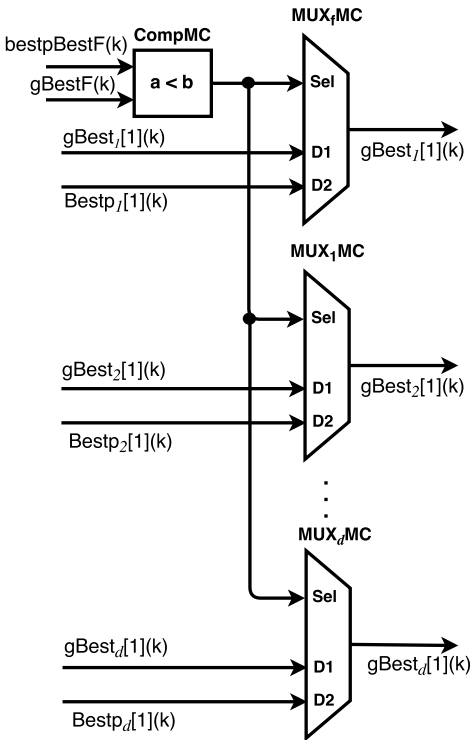

Figura 2: Módulo de Comparação - MC.

MUX$_f$MC e MUX$_1$MC ao MUX$_m$MC, são selecionados pelo comparador COMPMC que seleciona o menor valor de *fitness*. O MUX$_f$MC é o multiplexador responsável por selecionar qual valor de *fitness* vai ser passado adiante enquanto o MC e $_f$ são rótulos para sua posição do multiplexador no circuito. Essa notação será adotada durante todo o trabalho.

4.2 Módulo gBest

O módulo gBest, MgBest, é constituído por d registradores, conforme apresentado na Figura 3, em que cada um destes guarda uma das dimensões da posição da melhor partícula da nuvem, gBest. Os valores armazenados nesses registradores, por sua vez, serão dirigidos para o MC com o intuito de serem comparados, na próxima geração, com as novas posições encontradas na nuvem de soluções, e assim, detectar se foi encontrado um melhor valor para gBest.

4.3 Módulo de Partículas

O módulo de Partículas, MP, é composto por N submódulos PMP e N-1 submódulos MCMP. As entrada do MP são as coordenadas do gBest com profundidade d. Conforme ilustrado na Figura 4, as coordenadas são mostradas pelo barramento de tamanho m, que é a entrada do módulo e depois é distribuído para todos os P$_j$MP. Os MCMPs do MP são os Módulos de Comparação mostrados na Seção 4.1. Eles compararam qual das partículas

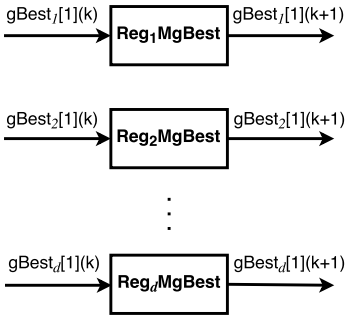

Figura 3: Módulo gBest - MgBest.

da nuvem obteve o melhor valor de *fitness*, e assim, passam as informações dessa partícula adiante.

O $j-$ésimo PMP (P$_j$MP) é mostrado em detalhes na Figura 5. Ele é formado por três submódulos: Calc$_j$MP, FitEvalMP e o CompFit$_j$MP. O P$_j$MP é responsável por calcular a nova posição de cada partícula da nuvem, seu respectivo valor de fitness e, se for o caso, atualizar o valor pBest da partícula. Dentro do P$_j$MP, o submódulo Calc$_j$MP é responsável por calcular a nova posição e velocidade da partícula através das Equações 1 e 2. As entradas desse submódulo são o gBest e o pBest. Esse último valor é selecionado pelos multiplexadores MUXP$_J$MP de acordo com o melhor valor entre o pBest e a nova posição recém calculada.

A implementação do $j-$ésimo $CalcPMP$, $CalcP_jMP$, pode ser vista na Figura 6. Esse bloco contém os valores necessário para o cálculo da Equação 1 incluindo os valores gBest e pBest de entrada, as constantes $w\_inercia$, $c_1 cognitivo$ e $c_2 social$ armazenadas em registradores e valores aleatórios (Rand$_1$CalcP$_j$PM e Rand$_2$CalcP$_j$PM). Como mencionado na Seção 3, a constante w é o fator de inércia, a constante c_1 é o fator de aprendizado cognitivo e c_2 o fator de aprendizado social da partícula.

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 com base no polinômio $r^{32}+r^{22}+r^2+1$ (Goresky and Klapper, 2006). A cada k-ésima geração do algoritmo, uma variável aleatória de 32 bits é produzida. Para evitar uma mesma sequência de valores, cada gerador $RandCalcP_jPM$ possui um valor inicial de 32 bits diferente.

Ainda sobre o submódulo $CalcP_jMP$, mostrado na Figura 6, após as operações matemáticas, existem limitadores de velocidades e de posição que possuem como função verificar se os novos respectivos valores calculados para cada variável estão dentro dos valores permitidos pelo algoritmo. Após essa verificação, os novos valores de velocidade e posição atualizam os registradores V_Atual

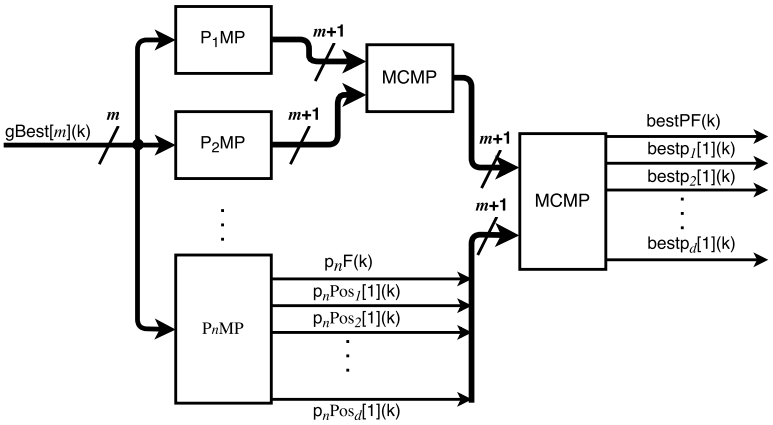

Figura 4: Implementação do MPs.

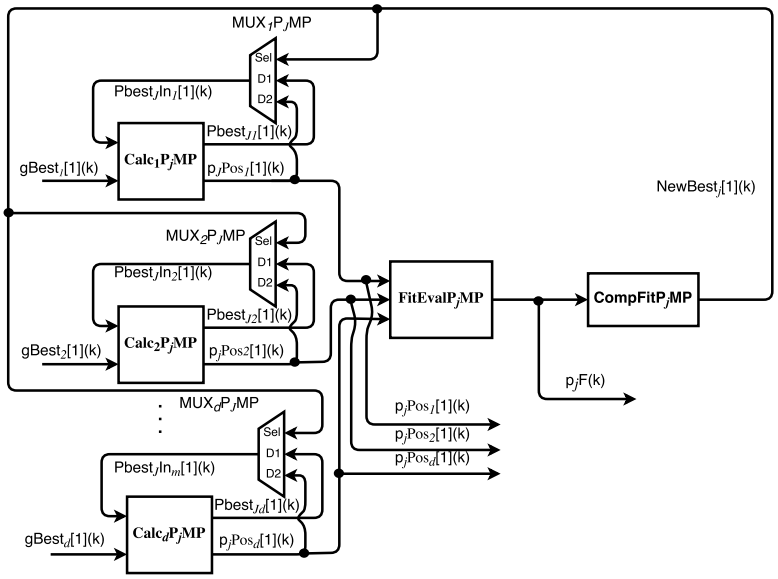

Figura 5: Submódulo P_jMP.

e *Posicao*, respectivamente.

Já o $j-$ésimo submódulo $CompFit_jP_jMP$, mostrado na Figura 7, tem como função validar se o melhor valor já alcançado pela partícula, pBest, foi superado ou não. Caso positivo, o registrador $BestLocal_dP_jMP$ é atualizado com o novo valor de *fitness*. A comparação é feita pelo $CompP_jMP$, que confronta o valor do registrador $BestLocal_dP_jMP$ com o valor de entrada $p_jF(k)$. Ele é responsável também por controlar a saída do multiplexador $MuxP_jMP$.

Por último, o bloco FitEvalP$_j$MP calcula o valor de *fitness* da j-ésima partícula, P$_j$MP, da nuvem de soluções. As entradas desse bloco são as componentes das coordenadas da partícula e a saída única é o valor de *fitness*, que por sua vez, é calculado através de uma LUT implementada em uma memória ROM permitindo, dessa forma, que a função de avaliação seja facilmente alterada, necessitando apenas a modificação dos valores armazenados na memória somente de leitura.

5 Resultados

Para validar a proposta de implementação da PSO em FPGA foram realizadas análises de síntese e funcionamento do algoritmo na otimização da função Rastrigin (Rastrigin, 1974; Mühlenbein et al., 1991), definida pela expressão

$$f(x) = AD + \sum_{i=1}^{D} \left(x_i^2 - A\ cos(2\pi x_i)\right), \qquad (3)$$

em que $A = 10$ é uma constante e D é o número de dimensões da função. Essa função é amplamente utilizada para validar técnicas de otimização por possuir um grau elevado de dificuldade devido aos seus inúmeros mínimos locais.

Este trabalho utiliza a função de Rastrigin com 3, 6 e 10 dimensões e, para cada uma, foram testadas implementações distintas da PSO, cujas nuvens foram compostas de 5, 10 ou 15 partículas.

Além disso, o tamanho m das partículas, em bits, também sofreu variações. Todos os resultados foram obtidos para um FPGA virtex 6

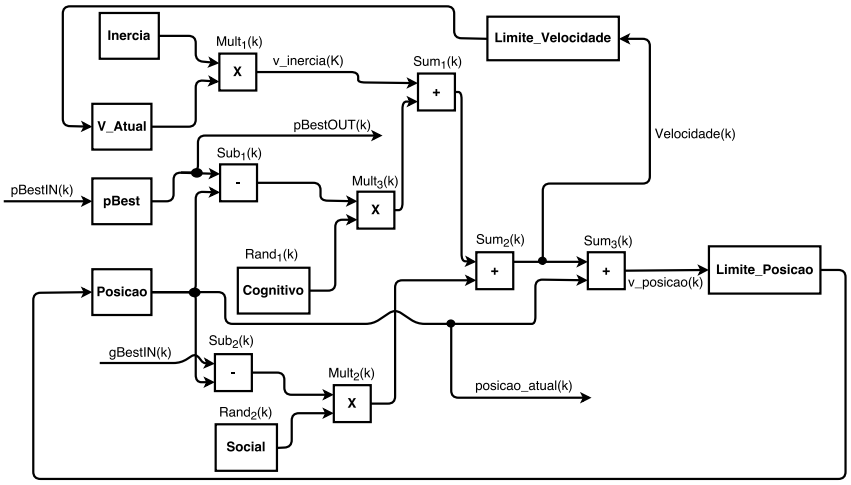

Figura 6: Submódulo $CalcP_jMP$.

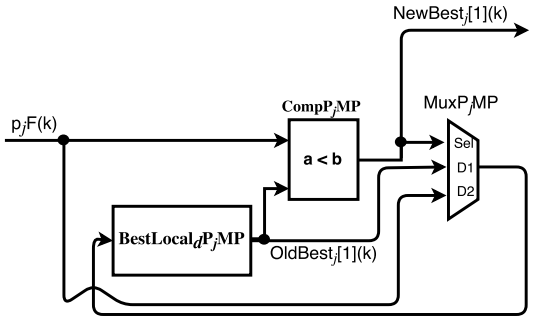

Figura 7: Submódulo $CompFit_jP_jMP$.

xc6vcx240t 1ff1156. O FPGA Virtex 6 utilizado possui 301.440 registradores; 150.720 células lógicas(LUT) que podem ser utilizadas para implementar funções lógicas ou memórias; e 768 células de DSP com multiplicadores e acumuladores. Em todos os testes realizados, as operações da PSO são realizadas a uma taxa de amostragem $R_s = \frac{1}{T_s}$ (em *Mega-Samples per Second*, MSPS), em que T_s, dado em segundos, é o tempo entre cada k-ésima iteração.

As Tabelas 2-4, 5-7 e 8-10 apresentam os resultados de síntese na FPGA alvo para as implementações da PSO com nuvens de $N = 5$, $N = 10$ e $N = 15$ indivíduos, respectivamente, cujas dimensões das partículas variam entre $D = 3$, $D = 6$ e $D = 10$. As colunas m, NR, NL, NM e R_s apresentam, respectivamente, o tamanho da partícula, em bits, o número de registradores, número de LUTs, o número de multiplicadores já embarcados em hardware e a taxa de amostragem, em MSPS.

Observa-se que, em geral, a taxa de amostragem R_s, em MSPS, e a ocupação de células lógicas são muito sensíveis tanto ao aumento do número de partículas na nuvem, quanto ao aumento do número m de bits que compõe as partículas. A ocupação de LUTs, colunas NL, foi crescente e aproximadamente linear em relação tanto ao aumento do número de partículas, quanto ao aumento do tamanho das partículas, quanto ao aumento da dimensão da função de otimização. Nas implementações mais críticas (15 indivíduos, 10 dimensões

Tabela 2: Resultado da síntese em FPGA da PSO com N=5 partículas para $D = 3$ dimensões.

m	NR	NL	NM	R_s
12	723	2.936	45	66,94
16	928	3.808	45	61,67
20	1.192	4.808	60	59,66
32	1.903	7.924	120	45,54

Tabela 3: Resultado da síntese em FPGA da PSO com N=5 partículas para $D = 6$ dimensões.

m	NR	NL	NM	R_s
12	1.366	5.774	90	61,56
16	1.790	7.380	90	55,99
20	2.259	9.350	120	51,83
32	3.609	15.134	240	42,29

Tabela 4: Resultado da síntese em FPGA da PSO com N=5 partículas para $D = 10$ dimensões.

m	NR	NL	NM	R_s
12	2.224	9.301	150	53,75
16	2.895	12.239	150	50,64
20	3.680	15.327	200	46,00
32	5.882	24.765	400	34,89

Tabela 5: Resultado da síntese em FPGA da PSO com 10 partículas para $D = 3$ dimensões.

m	NR	NL	NM	R_s
12	1.436	5.933	90	62,13
16	1.826	7.673	90	52,84
20	2.308	9.767	120	50,72
32	3.693	15.717	240	39,07

Tabela 6: Resultado da síntese em FPGA da PSO com 10 partículas para $D = 6$ dimensões.

m	NR	NL	NM	R_s
12	2.674	11.239	180	44,00
16	3.499	14.704	180	42,12
20	4.414	18.743	240	41,70
32	7.010	30.215	480	32,68

Tabela 7: Resultado da síntese em FPGA da PSO com 10 partículas para $D = 10$ dimensões.

m	NR	NL	NM	R_s
12	4.386	18.590	300	42,44
16	5.716	24.980	300	37,96
20	7.187	31.084	400	33,34
32	–	–	–	–

Tabela 8: Resultado da síntese em FPGA da PSO com N=15 partículas para $D = 3$ dimensões.

m	NR	NL	NM	R_s
12	2.128	8.634	135	53,83
16	2.710	11.179	135	49,50
20	3.452	14.609	180	41,52
32	5.469	23.357	360	33,36

Tabela 9: Resultado da síntese em FPGA da PSO com N=15 partículas para $D = 6$ dimensões.

m	NR	NL	NM	R_s
12	4.029	17.394	270	45,06
16	5.229	22.018	270	38,52
20	6.541	28.906	360	35,03
32	10.425	46.293	720	28,75

Tabela 10: Resultado da síntese em FPGA da PSO com N=15 partículas para $D = 10$ dimensões.

m	NR	NL	NM	R_s
12	6.496	28.007	450	39,52
16	8.462	36.574	450	33,52
20	10.706	46.683	600	33,60
32	–	–	–	–

e 20 bits, e 15 indivíduos, 6 dimensões e 32 bits), a taxa de ocupação de LUTs não passou de 30%. Esse fato é importante para implementações que demandem populações maiores ou partículas com mais informações, ou seja, maior número de bits para representá-las. A situação mais crítica em relação a ocupação foi em relação a utilização dos multiplicadores já embarcados (blocos de DSP), coluna NM, no FPGA que se chegou a utilizar 79% destes recursos para 15 partículas com $D = 10$ dimensões. Este problema pode ser resolvido implementando parte dos multiplicadores com células lógicas. Por conta da limitação de espaço disponível em um FPGA, a abordagem proposta não pôde ser testada em algumas configurações como as mostradas nas últimas linhas das Tabelas 7 e 10.

Os resultados mostram ganhos significativos com relação aos trabalhos na literatura apresentados em (Calazan et al., 2014), (Rathod and Thakker, 2014), (Arboleda et al., 2009), (Muñoz et al., 2010) e (Li et al., 2011) onde, para a função Rastrigin para 10 partículas e $D = 6$ dimensões os *throughput* obtidos foram entre 44 MSps e 32 MSps para 12 e 32 bits, respectivamente. Estes valores equivalem a um *speedup* de $\approx 212\times$ ($\frac{44\,\text{MSps}}{207\,\text{KSps}}$) para 12 bits e de $\approx 154\times$ ($\frac{32\,\text{MSps}}{207\,\text{KSps}}$), para 32 bits.

Em relação a ocupação em hardware, observa-se que a implementação proposta neste trabalho também obteve ganhos quando comparados com os dados da literatura utilizada. Em (Calazan et al., 2014) foram utilizados 145.000 das LUTs, de um FPGA Virtex 6, para 10 partículas e $D = 6$ enquanto que na proposta apresentada aqui foram utilizadas apenas 30.000, uma redução de 4,8×. É importante observar que em nenhum dos trabalhos estudados foi atingindo um hardware com 15 partículas e dimensão $D = 10$.

6 Conclusões

Este trabalho apresentou uma proposta de implementação paralela em ponto fixo para FPGA do algoritmo *Particle Swarm Optimization*. Todos detalhes de implementação da proposta foram apresentados, testados e analisados em termos de área de ocupação e tempo de processamento. Os resultados obtidos são bastante significativos com relação a outras abordagens propostas na literatura, obtendo melhorias tanto em termos de velocidade quanto de ocupação de espaço no hardware.

Apesar dos contrapontos em termos da limitação de espaço do FPGA, configurações semelhantes às consideradas nesse trabalho são suficientes para resolver um grande número de problemas de otimização. Adicionalmente, as altíssimas taxas de amostragem obtidas pela estrutura apresentada apontam a possibilidade de uso em aplicações de tempo real.

Referências

Alba, E., Luque, G. and Nesmachnow, S. (2012). Parallel metaheuristics: Recent advances and new trends, *International Transactions in Operational Research* **20**: 1–48.

Arboleda, D. M. M., Llanos, C. H., d. S., L. and Ayala-Rincón, M. (2009). Hardware architecture for particle swarm optimization using floating-point arithmetic, *2009 Ninth International Conference on Intelligent Systems Design and Applications*, pp. 243–248.

Calazan, R. M., Nedjah, N. and Mourelle, L. M. (2014). A hardware accelerator for particle

swarm optimization, *Applied Soft Computing* **14**: 347 – 356.

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.

Fathi, V. and Montazer, G. A. (2013). An improvement in rbf learning algorithm based on pso for real time applications, *Neurocomputing* **111**: 169 – 176.

Gogna, A. and Tayal, A. (2013). Metaheuristics: review and application, *Journal of Experimental & Theoretical Artificial Intelligence* **25**(4): 503–526.

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.

Kamel, M. A., Yu, X. and Zhang, Y. (2016). Real-time optimal formation reconfiguration of multiple wheeled mobile robots based on particle swarm optimization, *2016 12th World Congress on Intelligent Control and Automation (WCICA)*, pp. 703–708.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization, *Particle swarm optimization*, Vol. 4, pp. 1942–1948.

Kocadag, F., Cinar, R. F. and Demirkol, A. (2016). Kalman filter and particle swarm optimization on real time satellite tracking, *2016 CIE International Conference on Radar (RADAR)*, pp. 1–5.

Leskovec, J.; Rajaraman, A. and Ullman, J. D. (2014). Mining of massive datasets., *Cambridge university press* .

Li, S.-A., Hsu, C.-C., Wong, C.-C. and Yu, C.-J. (2011). Hardware/software co-design for particle swarm optimization algorithm, *Information Sciences* **181**(20): 4582 – 4596. Special Issue on Interpretable Fuzzy Systems.

Li, S.-A., Wong, C.-C., Yu, C.-J. and Hsu, C.-C. (2010). Hardware/software co-design for particle swarm optimization algorithm, *2010 IEEE International Conference on Systems, Man and Cybernetics*, pp. 3762–3767.

Mühlenbein, H., Schomisch, M. and Born, J. (1991). The parallel genetic algorithm as function optimizer, *Parallel Comput.* **17**(6-7): 619–632.

Muñoz, D. M., Llanos, C. H., d. S. Coelho, L. and Ayala-Rincón, M. (2010). Comparison between two fpga implementations of the particle swarm optimization algorithm for high-performance embedded applications, *2010 IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA)*, pp. 1637–1645.

Pati, A. B., Chile, R. H. and Wahane, V. R. (2015). Fuzzy-pso approach for control in nonlinear systems - real time application, *Michael Faraday IET International Summit 2015*, pp. 36–42.

Rastrigin, L. A. (1974). Extremal control systems, *Theoretical Foundations of Engineering Cybernetics Series*, Moscow.

Rathod, A. and Thakker, R. A. (2014). Fpga realization of particle swarm optimization algorithm using floating point arithmetic, *2014 International Conference on High Performance Computing and Applications (ICHPCA)*, pp. 1–6.

Rojas-Morales, N.; Rojas, M.-C. R. and Ureta, E. M. (2017). A survey and classification of opposition-based metaheuristics, *Computers & Industrial Engineering* .

Samà, M., DAriano, A., Toli, A., Pacciarelli, D. and Corman, F. (2015). Metaheuristics for real-time near-optimal train scheduling and routing, *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pp. 1678–1683.

Soler-Dominguez, A.; Juan, A. A. and Kizys, R. (2017). A survey on financial applications of metaheuristics, *ACM Comput. Surv.* .

Souier, M., Sari, Z. and Hassam, A. (2012). Real-time rescheduling metaheuristic algorithms applied to fms with routing flexibility, *The International Journal of Advanced Manufacturing Technology* **64**.

Tsai, C. W., Cho, H. H., Shih, T. K., Pan, J. S. and Rodrigues, J. J. P. C. (2015). Metaheuristics for the deployment of 5g, *IEEE Wireless Communications* **22**(6): 40–46.