Um Algoritmo Híbrido Paralelo Cooperativo Baseado em DE, PSO e AG

Uma Avaliação em Computadores Multicore

Omar Andres Carmona Cortes, Bruno Alberth Silva Barros, Rafael Fernandes Lopes, Josenildo Costa da Silva
Departamento Acadêmico de Informática
Instituto Federal de Educação, Ciência e Tecnologia do Maranhão (IFMA)
São Luis, MA, Brasil
omar@ifma.edu.br, b.alberthsb@gmail.com, rafaelf@ifma.edu.br, jcsilva@ifma.edu.br

Pedro Felipe do Prado
Instituto de Ciências Matemáticas e de Computação (ICMC)
Universidade de São Paulo (USP)
São Carlos, SP. Brasil
pfprado@icmc.usp.br

***Abstract*—This paper presents a new parallel hybrid algorithm combining Particle Swarm Optimization (PSO), Differential Evolution (ED) and Genetic Algorithms (GA) for optimizing unconstrained numerical functions. Basically, PSO and ED evolves independently in our proposal, then they cooperate between them exchanging their best individual, which undergo GA operators locally in order to search different areas in the search space. The results are evaluated concerning the quality of the solution and speedup against six benchmarks functions. The outcomes are compared with GA, PSO and ED in the serial version and with the classical island model of PSO and DE, considering two and four threads. A comparison with a hybrid algorithm is done as well.**

Keywords—optimization; particle swarm; genetic algorithms; constraints; benchmarks

I. INTRODUÇÃO

Os métodos de busca baseados nos princípios da teoria da evolução, proposta por Charles Darwin, tem recebido crescente interesse por parte da comunidade científica, devido principalmente à capacidade de resolver problemas complexos e de obter soluções satisfatórias onde métodos convencionais são ineficientes. Embora esses algoritmos gerem boas soluções para problemas de otimização contínua ou numérica, eles nem sempre são suficientes, ou seja, a qualidade das soluções nem sempre é satisfatória. Uma alternativa para solucionar esse problema é a utilização de algoritmos híbridos, onde características de diferentes heurísticas podem ser utilizadas para melhorar a qualidade das soluções encontradas, dado que cada heurística tem uma forma diferente de explorar o espaço de busca.

Muitos trabalhos baseados em algoritmos híbridos na otimização numérica têm sido desenvolvidos. Por exemplo, o trabalho de Thangaraj [1] mistura evolução diferencial e programação evolutiva. Uma mistura entre evolução diferencial, simulated annealing e teoria do caos é feita no trabalho de Gao & Jia [2]. PSO e algumas variações baseadas na teoria de autômatos são aplicados na otimização de cinco funções de benchmarks no trabalho de Hasehimi & Meybodi [3]. No trabalho de Gong & Jiao [4] são usados operadores genéticos em um algoritmo de seleção clonal de modo a melhorar as afinidades dos antígenos. Cortes & da Costa [5], misturam AGs, seleção clonal e Hillclimbing. Alizade et al. [6] fazem uso cooperativo entre AGs e PSO, onde o resultado do AG e passado para o PSO.

Dependendo de como ocorre a hibridização, o processo de evolução pode se tornar computacionalmente intenso quando se trata de um problema com muitas variáveis, ainda mais quando são utilizadas funções multimodais, ou seja, que apresentam muitos ótimos locais. A computação paralela se apresenta atraente para esse tipo de cenário, dado que há um paralelismo natural dos algoritmos evolutivos, permitindo assim uma maior exploração do espaço de busca. Aliado a este fato esta a popularização dos computadores multicore (vários núcleos), o que disponibiliza para usuários e programadores uma plataforma com vários processadores e um espaço de memória compartilhado [7], o que antes só era possível em grandes computadores, como as máquinas fabricadas, por exemplo, pela Cray e SGI. Nesse âmbito, os algoritmos evolutivos paralelos também tem recebido atenção. O trabalho de Singh et. al. [8] utiliza um algoritmo genético (AG) paralelo para descobrir estruturas de proteínas. Deep et. al. [9] usa uma nuvem de partículas (PSO) paralelas em algumas funções de benchmarks, melhorando a qualidade das soluções. Outros trabalhos nessa linha são: [10],[11] ,[12] e [13]. Assim, pode-se identificar que o paralelismo trás benefícios tanto na qualidade da solução quanto na redução do custo computacional.

Quando se trata de algoritmos híbridos paralelos, alguns trabalhos podem ser citados. O de Luong [14], por exemplo, mescla AGs com uma busca local (Busca Tabu) para solucionar problemas de QAP (*Quadratic Assigment Problem*). Said e Nakamura [15] usam AGs e Algoritmos de Estimação de Distribuição na otimização de algumas funções de benchmark. Li e Yang [16] executam o modelo de ilhas com diferentes heurísticas em cada uma delas, ou seja, as diferentes heurísticas cooperam entre si para encontrar a solução de algumas funções de

benchmark. Outros exemplos de trabalhos nessa linha são: [17],[18] e [19] .

Neste artigo é proposto um novo algoritmo híbrido paralelo cooperativo, cujo modelo foi inspirado no modelo apresentado por Li e Yang [16] onde diferentes heurísticas cooperam entre si. No entanto, neste trabalho utilizam-se as heurísticas evolução diferencial (ED), PSO e AG. Outra diferença esta no fato que as ilhas executam ED e PSO, sendo que o AG é aplicado quando as populações são migradas, objetivando aumentar ainda mais a capacidade de exploração do espaço de busca. Os algoritmos são testados em seis funções de benchmarks, sendo cinco multimodais (possuem vários ótimos locais) e uma monomodal (possui um único ótimo). Nesse contexto, este trabalho esta dividido da seguinte forma: a Seção II apresenta as heurísticas utilizadas; a Seção III mostra o algoritmo paralelo proposto; a Seção IV discute os experimentos e os resultados da avaliação de desempenho em termos de qualidade da solução e speedup; finalmente a Seção V sumariza as conclusões deste trabalho.

II. ALGORITMOS EVOLUTIVOS

A. Evolução Diferencial

A Evolução Diferencial (ED) é uma metaheurística desenvolvida por Rainer Storn e Kenneth Price [20], em 1995. Seu funcionamento se dá basicamente pelo algoritmo mostrado na Figura 1, sendo que a estratégia mais popular é chamada "DE/rand/1/bin" [21].

```
pop <- Inicializa(x)
fit <- Avalia (pop)
Enquanto critério de parada não atingido
   pop_m <- mutação(pop)
   pop_c <- cruzamento(pop_m)
   pop <- Seleção(Pop_m, Pop_c)
Fim-Enquanto
```

Fig 1. Algoritmo de evolução diferencial

O algoritmo e iniciado criando uma população inicial escolhida aleatoriamente com distribuição uniforme. Em seguida, enquanto o critério de parada não for atingido o algoritmo passa pelos seguintes passos. A mutação é feita através do vetor de diferença da Equação 1. Os vetores X são escolhidos aleatoriamente ($k \neq l \neq m$); i varia de 1 até D (dimensões) e F é um fator escolhido pelo usuário. O cruzamento é efetuado com a intensão de manter a diversidade no processo de busca de acordo com a Equação 2, onde um gene é escolhido do vetor V_i caso $\eta_j \in [0,1]$ seja menor do que $CR \in [0,1]$ (parâmetro fornecido pelo usuário), ou escolhido de X_i (vetor a ser substituído, chamado de *trial*) caso contrário. Será escolhido para a próxima geração aquele vetor que apresentar a melhor aptidão entre o conjunto $\{U_i, X_i\}$

$$V_i = X_k + F(X_l - X_m) \tag{1}$$

$$U_{ij} = \begin{cases} V_{ij}, se\ (\eta_j \leq CR) \\ X_{ij}, caso\ contrário \end{cases} \tag{2}$$

B. Algoritmos Genéticos(AGs)

Os AGs [22] são algoritmos robustos, genéricos e facilmente adaptáveis, consistem de uma técnica amplamente estudada e utilizada em diversas áreas. Basicamente, o que um algoritmo genético faz é criar uma população de possíveis respostas para o problema a ser tratado (inicialização) para depois submetê-la ao processo de evolução como mostra a Figura 2.

```
pop <- Inicializa(x)
fit <- Avalia (Pop)
Enquanto critério de parada não atingido
   pop <- Seleção(pop)
   pop_c <- cruzamento(pop)
   pop_m <- mutação(pop_c)
   pop <- atualização(pop_m)
Fim-Enquanto
```

Fig 2. Algoritmo de algoritmo genético

Assim como na evolução diferencial, a população pode ser criada aleatoriamente com distribuição uniforme. Em seguida faz-se a seleção dos indivíduos para o cruzamento. A probabilidade de uma dada solução i ser selecionada é proporcional à sua aptidão (método da roleta), torneio (indivíduos competem para ser selecionado). Detalhes podem ser visto no trabalho de Michalewicz [22] e Herrera [23]. O cruzamento mais comumente utilizado é o cruzamento simples, onde um ponto de corte é selecionado e os filhos são gerados com partes de cada pai. Neste trabalho adotou-se o cruzamento BLX-α, no qual um vetor $V = (v_1, v_2, ..., v_d)$ é gerado onde cada elemento estará no intervalo mostrado na Equação 3, os genes c_1 e c_2 correspondem aos genes dos pais que irão gerar o descendente. Detalhes de como os demais cruzamentos funcionam podem ser vistos no trabalho de Herrera [23]. Na mutação as características dos indivíduos resultantes do processo de reprodução são alteradas, acrescentando assim uma pequena variedade à população. Os genes a serem mutados são escolhidos dada uma probabilidade baixa. Dentre as mutações disponíveis as mais comum é a aleatória, também chamada de uniforme, que gera um novo gene no intervalo permitido pela dimensão $[a_i, b_i]$. Por fim, os novos indivíduos substituem os pais para a próxima geração. No entanto, é possível utilizar-se de elitismo onde o melhor indivíduo é sempre mantido mesmo que ele esteja na população de pais

$$\begin{gathered} v_i = [cmin - I\alpha,\ cmax + I\alpha] \\ c_{min} = \min\{c_i^1, c_i^2\} \\ c_{max} = \max\{c_i^1, c_i^2\} \\ I = c_{max} - c_{min} \end{gathered} \tag{3}$$

C. Otimização por Núvem de Partículas (PSO)

O PSO (Particle Swarm Optimization ou Otimização por Nuvem de Partículas) é um algoritmo baseado no conceito de movimentação de enxames, que é como diversos animais, como peixes, pássaros e insetos se comportam no meio-ambiente. Foi desenvolvido pelo psicólogo social James Kennedy e o engenheiro eletricista Russel Eberhart em 1995 para simulação de comportamento social [24]. O PSO clássico funciona como mostrado na Figura 3.

A Inicialização prepara a população inicial (chamada de nuvem ou enxame) de candidatos a solução (chamadas de

partículas). As partículas possuem uma posição (x_{i1}, x_{i2}, …, x_{id}), uma melhor posição prévia (ou *pbest*, p_{i1}, p_{i2},…,p_{id}), velocidades (v_{i1}, v_{i2},…, v_{id}) e uma aptidão atual. Em seguida compara-se a aptidão e a aptidão *pbest* da partícula. Se o valor atual for melhor que o de *pbest* então o valor da aptidão *pbest* é substituído pelo valor da aptidão encontrada, em seguida a posição (p_{i1},p_{i2},...,p_{id}) é substituída pela posição atual no espaço d-dimensional. Depois o valor de todas as aptidões são comparadas com a aptidão da partícula *gbest*. Se alguma for melhor que este, o *gbest* é substituído por essa partícula da mesma forma.

```
Inicializa(P)
Inicializa(V)
fit <- Avalia_Particulas(P)
Enquanto critério de parada não atingido
   Se melhor(fit) < melhor(global)
      gbest <- melhor(fit)
   Se melhor(fit) < melhor(pbest)
      Pbest <- melhor(fit)
   V <- Atualiza_velocidades()
   P <- Atualiza_posições(P,V)
   Fit <- Avalia(P)
Fim-Enquanto
```

Fig 3. Algoritmo de otimização por núvem de partículas (PSO)

Na atualização é feita a otimização das partículas, que depende dos parâmetros fator de sociabilidade, que determina a atração das partículas para a melhor posição descoberta por qualquer elemento do enxame. Esta também é feita utilizando o fator de individualidade e a velocidade máxima, que determinam a atração da partícula com sua melhor posição e a delimitação do movimento, respectivamente, como pode ser visto na Equação 4, onde c_1 e c_2 são os componentes cognitivo e social, respectivamente; r_1 e r_2 são números aleatórios no intervalo [0, 1]; e w é o peso ou fator inercial, que determina a diversificação ou intensificação de partículas. A posição final da partícula é calculada pela Equação 5.

$$v_{ij} = wv_{ij} + c_1r_1(p_{id} - x_{ij}) + c_2r_2(g - x_{ij}) \quad (4)$$

$$x_{ij} = x_{ij} + v_{ij} \quad (5)$$

III. O ALGORITMO PARALELO HÍBRIDO (DEPSO-GA)

A ideia do algoritmo paralelo híbrido é executar EDs e PSOs independentes em cada *thread* e trocar informações entre as populações a cada certo número de iterações (migração), como ilustrado no algoritmo da Figura 4. O modelo de computação paralela utilizado é o modelo de ilhas, onde as populações evoluem de forma independente.

```
Inicializar threads/ilhas
Enquanto critério de parada não atingido
   Executar a heurística(DE ou PSO)
   Se (Todas as Threads estiverem na barreira)
     Migrar indivíduos entre populações;
     Receber indivíduos
     Aplicar AG;
Fim-Enquanto
Selecionar indivíduo mais apto
```

Fig 4. Algoritmo paralelo híbrido

Basicamente, o algoritmo começa inicializando as populações de cada heurística em cada *trhead*. Como são considerados até quatro núcleos são duas as possibilidades de execução. Usando dois núcleos, executam-se um PSO e uma ED. Com quatro núcleos executam-se dois PSOs e duas EDs. Quando a época de migração chega migra-se o melhor indivíduo para as demais ilhas. Ao recebê-los cada thread executa um AG juntamente com elementos aleatórios da população para tentar melhorar o indivíduo recebido, assim, fazendo com que cada thread possa explorar diferentes regiões no espaço de busca. A barreira é utilizada para que no momento da migração todos os threads estejam prontos para receber os indivíduos. Ao final é retornado o melhor indivíduo entre os threads executados.

IV. EXPERIMENTOS

Os experimentos foram realizados em um Quad-Core de 2.33GHz da Intel, 4GB de RAM, 1TB de HD e no ambiente de desenvolvimento Eclipse, utilizando o JAVA JDK 1.6.0_23. As funções de benchmarks utilizadas são bem conhecidas na literatura, sendo elas, seus ótimos e domínio as seguintes: Ackley ($f_1 = 0$, [*-32,768, 32,768*]), Griewank ($f_2 = 0$, [*-600, 600*]), Rastrigin ($f_3 = 0$, [*-5,12, 5,12*]), Schwefel ($f_4 = 12569.5$ [*-500, 500*]), Sphere ($f_5 = 0$, [*-5,12, 5,12*]) e Rosenbrock ($f_6 = 0$, [*-2.048, 2.048*]), sendo que todas elas foram otimizadas em uma dimensão igual a 30. Duas avaliações são consideradas: a qualidade da solução encontrada e o ganho com o paralelismo.

Neste trabalho adotou-se a migração a cada 50 iterações, onde apena o melhor indivíduo é migrado. Na ED a estratégia usada é a DE/best/1/bin, onde as operações são aplicadas ao melhor indivíduo e não a um indivíduo escolhido aleatoriamente. Dada a natureza estocástica do algoritmo o mesmo foi executado 31 vezes, utilizando 3000 gerações/iterações. O PSO executado é o canônico, com os seguintes parâmetros: c_1 e c_2 = 3.0, w_{max} = 0.9 e w_{min} = 0.5.

Tanto a ED quanto o PSO utilizam uma população de 160 indivíduos/partículas. Sendo que a população é dividida na execução paralela garantindo que a mesma quantidade de chamadas à função de avaliação sejam utilizadas. Em outras palavras, na versão com dois threads cada ilha terá uma população de oitenta indivíduos/partículas. Enquanto que na versão com quatro *threads* cada ilha terá uma população com quarenta indivíduos/partículas. O peso w é definido como mostrado na Equação 6, onde, w_{max} é o peso inicial, w_{min} o peso final, *iter* corresponde a iteração atual e *maxIter* o número máximo de iterações do PSO. Na ED o peso F é igual a 0.5. No AG utilizou-se o cruzamento BLX-α e adotou-se um α = 0,5 comum na literatura. O método de seleção é o de torneio com 6 indivíduos (os da migração mais os indivíduos da população local). A probabilidade de cruzamento é de 0.8 e de mutação de 0.07, sendo realizada a mutação uniforme.

$$w = w_{max}-[(w_{max}-w_{min}) \text{ x iter}]/maxIter \quad (6)$$

A. Qualidade da Solução

A Tabela I apresenta os resultados relativos às execuções seriais, onde EDPSO-AG2 representa a execução serial com 2 populações e EDPSO-AG4 é a execução serial com 4 populações. Nas versões seriais tudo acontece da mesma forma, porém as populações compartilham o mesmo *thread* e não é necessária a sincronização, enquanto que na versão paralela os *threads* executam separadamente. É importante notar que esse

execução também será utilizada posteriormente no cálculo do *speedup* e da eficiência como sugerido por Alba & Luqe [24]. Considerando os resultados, a versão cooperativa EDPSO-AG4 apresenta o melhor resultado na maioria das funções, perdendo para o AG em f_4 e para ED em f_6, provavelmente porque a divisão das populações as vezes tende a diminuir a qualidade da solução, pois menos indivíduos/partículas são espalhados pelo espaço de busca. Destaca-se também que a versão híbrida apresenta um baixo desvio padrão em todas as funções, exceto em f_4, o que mostra a estabilidade do algoritmo. Em outras palavras, o algoritmo vai encontrar soluções semelhantes em todas as execuções.

TABELA I. QUALIDADE DAS SOLUÇÕES DOS ALGORITMOS SERIAIS

	AG	**PSO**	**ED**	**EDPSO-AG2**	**EDPSO-AG4**
f_1	1,53426E-14 *(5,22296E-15)*	0,00232278 *(0,001074029)*	7,06961E-14 *(2,24639E-14)*	7,0911E-15 *(1,21068E-15)*	**6,17427E-15** *(2,99742E-15)*
f_2	0,001439739 *(0,003350902)*	0,000820933 *(0,002424819)*	**0** ***(0)***	9,80301E-05 *(0,000392564)*	**0** **(0)**
f_3	4,108067332 *(12,29418456)*	1,61 *(6,214887536)*	159,7956144 *(6,980344316)*	3,8772021 *(15,89925336)*	**0,00018352** *(0,001021797)*
f_4	**-12331,6956** *(388,2897955)*	-9564,626773 *(1849,534358)*	-6343,256971 *(275,7003739)*	-11821,88317 *(296,6773969)*	-11892,76795 *(528,5654312)*
f_5	8,19682E-92 *(4,5638E-91)*	4,10959E-20 *(1,34264E-19)*	**0** ***(0)***	**0** ***(0)***	**0** ***(0)***
f_6	25,68007698 *(2,405610984)*	7,012312445 *(10,88155831)*	**0,122020365** ***(0,066829595)***	0,221393585 *(0,421376218)*	0,280658694 *(1,086105849)*

TABELA II. QUALIDADE DAS SOLUÇÕES DOS ALGORITMOS PARALELOS

	EDPSO-AG2	**EDPSO-AG4**	**PSOIlha2**	**PSOIlha4**	**EDIlha2**	**EDIlha4**
f_1	**5,83046E-15** *(1,80478E-15)*	6,05967E-15 ***(1,78208E-15)***	0,002820994 *(0,001215359)*	0,229859003 *(1,214946009)*	2,728136279 *(6,165162575)*	0,804955171 *(3,662902327)*
f_2	2,65058E-05 *(0,000147578)*	**0** ***(0)***	0,000546529 *(0,000557596)*	0,016770324 *(0,046896811)*	5,69437E-16 *(4,91301E-16)*	0,000238582 *(0,001328368)*
f_3	0,154390945 *(0,605141185)*	**3,22E-05** ***(0,000178679)***	0,008184109 *(0,038801619)*	0,10881824 *(0,32836847)*	27,52182708 *(15,44421696)*	18,60981436 *(5,448651038)*
f_4	-11848,9232 *(436,1591358)*	**-11899,18244** ***(324,6900507)***	-9789,220036 *(1590,708832)*	-9458,38842 *(1412,714103)*	-10494,15773 *(394,3419212)*	-10023,11625 *(522,4513225)*
f_5	**0** ***(0)***	**0** ***(0)***	6,62172E-19 *(2,31645E-18)*	1,48013E-17 *(4,81004E-17)*	**0** ***0***	**0** ***(0)***
f_6	0,225972388 *(0,568109427)*	0,274998801 *(0,997126557)*	8,014301443 *(11,5502447)*	18,08544947 *(11,88825446)*	0,774982133 ***(0,324757931)***	**0,140782464** *(0,35869049)*

B. Qualidade da Solução

A Tabela I apresenta os resultados relativos às execuções seriais, onde EDPSO-GA2 representa a execução serial com 2 populações e EDPSO-GA4 é a execução serial com 4 populações. Nas versões seriais tudo acontece da mesma forma, porém as populações compartilham o mesmo *thread* e não é necessária a sincronização, enquanto que na versão paralela os *threads* executam separadamente. É importante notar que esse execução também será utilizada posteriormente no cálculo do *speedup* e da eficiência como sugerido por Alba & Luqe [24]. Considerando os resultados, a versão cooperativa EDPSO-GA4 apresenta o melhor resultado na maioria das funções, perdendo para o AG em f_4 e para ED em f_6, provavelmente porque a divisão das populações as vezes tende a diminuir a qualidade da solução, pois menos indivíduos/partículas são espalhados pelo espaço de busca. Destaca-se também que a versão híbrida apresenta um baixo desvio padrão em todas as funções, exceto em f_4, o que mostra a estabilidade do algoritmo. Em outras palavras, o algoritmo vai encontrar soluções semelhantes em todas as execuções.

A Tabela II mostra os resultados da comparação entre a abordagem cooperativa e os algoritmos padrão no modelo de ilhas. A versão cooperativa apresentou melhores resultados em quase todas as funções exceto em f_6, cujo melhor resultado foi obtido pelo modelo de ilhas utilizando evolução diferencial com 4 ilhas. Nas versões de ilhas pode-se observar que o PSO aparentemente fica preso em ótimos locais, em especial em f_1, f_2, f_3, f_4 e em especial em f_6. Algo parecido ocorreu no modelo de ilhas com evolução diferencial em f_3. Destaca-se que a função f_6 (Rosenbrock) é consideravelmente complexa de resolver, pois é quadrática e multimodal a partir de 3 dimensões. Nesse caso, tanto o algoritmo híbrido paralelo proposto quanto as

ilhas usando ED se mostraram eficientes em chegar próximo ao mínimo local dessa função.

As Tabelas III e IV apresentam a taxa de acertos para as versões seriais e paralelas, respectivamente, ou seja, se uma solução foi encontrada com um erro máximo de $1x10^{-6}$. O valor zero indica que nenhuma solução foi encontrada nas 31 execuções do algoritmo. Por outro lado, cem por cento indica que a solução foi encontrada em todas as execuções. Nas versões seriais destacam-se os algoritmos híbridos EDPSO-AG4 e a ED com 100% em três das funções de benchmark, sendo que o EDPSO-AG4 ainda chegou próximo de encontrar sempre a solução para a função f_3. O AG teve um desempenho razoável na maioria das funções, porém não conseguiu chegar no erro esperado nem sequer uma única vez na função f_6. Isso se deve ao fato da função ser quadrática, o que dificulta significativamente a busca pela solução. O PSO teve um desempenho pobre o que indica que o algoritmo fica preso em mínimos locais, prova disso é que a única função em que todos os algoritmos conseguem chegar a um solução, inclusive o PSO, é a função f_5, que é monomodal .

TABELA III. TAXA DE ACERTOS DAS VERSÕES SERIAIS

	GA	**PSO**	**DE**	**DEPSO-AG2**	**DEPSO-AG4**
f_1	100%	0	100%	100%	100%
f_2	80%	0	100%	93%	100%
f_3	58%	0	0	0	96,7%
f_4	61%	0	0	0	0
f_5	100%	100%	100%	100%	100%
f_6	0%	22,5%	0	6,4%	0

Nas versões paralelas o comportamento pode tender a piorar porque a população é dividida entre as ilhas, ou seja, a qualidade das soluções pode cair. Porém, isso não acontece com a versão paralela híbrida, pois se tenta manter a independência na evolução de cada ilha. Por outro lado, nota-se uma melhora no desempenho do PSO e uma piora no desempenho da ED. Mais do que isso, o EDPSO-AG2 conseguiu uma melhora significativa na busca pelo ótimo na função quadrática f_6.

TABELA IV. TAXA DE ACERTOS DAS VERSÕES PARALELAS

	PSO Ilha2	**PSO Ilha4**	**DE Ilha2**	**DE Ilha4**	**DEPSO-AG2**	**DEPSO-AG4**
f_1	77,4%	93,5%	0	0	100%	100%
f_2	100%	96,7%	0	0	96,7%	100%
f_3	0	0	0	0	0	93,5%
f_4	0	0	0	0	0	0
f_5	100%	100%	100%	100%	100%	100%
f_6	0	42%	19,3%	0	19,3%	0

C. Avaliação do Paralelismo

A Tabela 4 mostra o *speedup* e a eficiência alcançada pelos algoritmos paralelos com dois e quatro threads, respectivamente, onde a parte superior representa o *speedup* e a inferior a eficiência em porcentagem. Obviamente para efeitos de cálculo utilizaram-se as respectivas versões seriais com dois e quatro ilhas usando uma única *thread*, fazendo com que a comparação seja justa [25].

TABELA V. SPEEDUP E EFICIENCIA

	DEPSO-AG2		**DEPSO-AG4**	
	Sp	*Ef*	*Sp*	*Ef*
f_1	1,48	*74%*	4,12	***102%***
f_2	1,62	*81%*	4,15	***103%***
f_3	1,35	*67%*	3,02	*75%*
f_4	1,48	*74%*	2,58	*64%*
f_5	1,97	*98%*	4,17	***104%***
f_6	1,25	*62%*	2,24	*56%*

Na versão com 2 ilhas o melhor *speedup* foi encontrado na função f_5, pois chega-se próximo ao que seria o ideal. Esse comportamento deve-se ao fato da função f_5 ser fácil de ser calculada por ser monomodal. Nas demais há um ganho razoável especialmente em f_2.

Na versão com 4 ilhas obteve-se um *speedup superlinear* nas funções f_1, f_2 e f_5, ou seja, acima do ideal. Isso acontece quando efeitos da memória cache aceleram a execução do programa, dado que o tamanho da população diminui em cada *thread* e a comunicação é feita por memória compartilhada. Nas demais funções há um ganho razoável, especialmente em f_3.

D. Comparação com a Literatura

Nesta seção será comparado o EDPSO-AG com uma metaheurística chamada HPSO-SA, desenvolvida por Idoumghar [25]. São usando os parâmetros descritos no artigo, que são: 2000 iterações, 30 partículas, $c_1 = c_2 = 2$ e w = [*0.9, 0.4*]. Os resultados são apresentados na Tabela VI, onde se pode observar que o algoritmo HPSO-AS conseguiu um melhor resultado apenas em f_4, enquanto que o EDPSO-AG apresentou melhores resultados nas demais funções.

TABELA VI. COMPARAÇÃO ENTRE EDPSO-AG E HPSO-SA

	Média		**Melhor**		**Pior**	
	HPSO	EDPSO	HPSO	EDPSO	HPSO	EDPSO
f_3	5,234E-05	**1,279E-09**	4,35E-09	**0**	0,0008	**6,39E-08**
f_4	**206,6955**	297,8615	**0**	39,7177	**355,318**	640,0839
f_2	0,0012	**0**	1,223E-15	**0**	0,0172263	**0**
f_6	0,5835	**0,0273**	0,0443	**1,89E-06**	1,36171	**1,3344**

V. CONCLUSÕES

Este artigo apresentou o desenvolvimento de um algoritmo híbrido paralelo, onde ilhas evoluem de forma independente utilizando ED e PSO. A migração ocorre a cada 50 iterações para que os algoritmos possam cooperar entre si. Para que as ilhas não sejam forçadas a seguirem todas na mesma direção, os indivíduos recebidos passam por mais uma evolução utilizando AGs, assim, permite-se que as ilhas explorem direções ainda melhores do que as fornecidas pelos membros imigran-

tes. Os resultados se mostraram bastante promissores em seis funções de benchmark. Contudo, outras configurações se fazem necessárias para tentar identificar qual a melhor combinação entre metaheurísticas, como por exemplo, executar uma heurística diferente em cada ilha, considerando PSO, ED e AG. Além disso, é necessário uma melhor avaliação em mais funções de benchmarks como as propostas pelo CEC2013 [26]. Outras possibilidades também advêm do controle na migração dos indivíduos, onde mais indivíduos podem ser migrados e a decisão sobre a migração ou não poderia ser feita automaticamente através da utilização de lógica nebulosa ou de uma rede neural. Outros modelos de algoritmos evolutivos paralelos também podem ser testados e comparados.

REFERENCIAS

[1] R. Thangaraj, M. Pant, A. Abraham, Y. Badr, Hybrid Evolutionary Algorithm for Solving Global Optimization Problems, Hybrid Artificial Intelligence Systems, LNCS, v. 5572, 2009, pp, 310-318, 2009.

[2] Y. Gao, S. Jia, Hybrid Differential Evolution Algorithm with Annealing and Chaos, in 5th International Conference on Natural Computation: IEEE Computer Society, pp. 270-274, 2009.

[3] A. B. Hashemi, M. R. Meybodi, Adaptive Parameter Selection Scheme for PSO: A Learning Automata Approach, In Proceedings of the 14th International CSI Computer Conference: IEEE Press, 2009.

[4] M. Gong, L. Jiao, W. Ma and R. Shang. "Hybrid Immune Algorithm with Intelligent Recombination". In IEEE Congress on Evolutionary Computation, CEC, pp. 1807 –1814, 2009.

[5] O. A. C. Cortes, J. S. da Costa, A Local Search Algorithm Based on Clonal Selection and Genetic Mutation for Global Optimization, Brazilian Symposium of Neural Networks (SBRN), 2010.

[6] G. Alizadeh, M. Baradarannia, P. Yazdizadeh, Serial Configuration of Genetic Algorithm and Particle Swarm Optimization to Increase de Convergence Speed and Accuracy, In: Conference on Intelligent Systems Design and Applications, IEEE Press, p. 272-277, 2010.

[7] T. Rauber, G. Rünger, Parallel Programming for Multicore and Cluster Computing, Berlin: Springer, 2010.

[8] S. K. Singh, K. Aggarwal, A. Gupta, Performance of Parallel Genetic Algorithms on Distributed Memory Architectures, Journal of Engineering Research and Studies, v. 2, p. 10-17, 2010.

[9] K. Depp, S. Sharma, M. Pant, Modified parallel particle swarm optimization for global optimization using Message Passing Interface, In: Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA), 2010.

[10] A. A. Eroftiev, N. E. Timofeeva, A; N. Savin, Parallel computing in application to global optimization problem solving, MIPRO, Proceedings of the 34th International Convention , p. 23-27, 2011.

[11] A. Lancinskas, J. Žilinskas, P. M. Ortigosa, Investigation of parallel particle swarm optimization algorithm with reduction of the search area, IEEE International Conference on Cluster Computing Workshops and Posters, p. 20-24, 2010.

[12] X. Lai; G. Tan, Studies on migration strategies of multiple population parallel particle swarm optimization, Eighth International Conference on Natural Computation, p. 29-31, 2012.

[13] Luong, T. V.; MELAB, N.; TALBI, E.-G., Parallel hybrid evolutionary algorithms on GPU, In: IEEE World Congress on Computational Intelligence, p. 1-9, 2010.

[14] S. M. Said, M. Nakamura, Parallel Enhanced Hybrid Evolutionary Algorithm for Continuous Function Optimization, Seventh International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, p. 125-130, 2012.

[15] C. Li, S. Yang, An Island Based Hybrid Evolutionary Algorithm for Optimization, Simulated Evolution and Learning, Lecture Notes in Computer Science Volume 5361, p. 180-189, 2008.

[16] J. Zhang; T-S. Pan, J-S Pan, A Parallel Hybrid Evolutionary Particle Filter for Nonlinear State Estimation," First International Conference on Robot, Vision and Signal Processing (RVSP), p. 308-312, 2011

[17] A. Nakib, B. Daachi, P. Siarry, Hybrid Differential Evolution Using Low-Discrepancy Sequences for Image Segmentation, IEEE International Symposium on Parallel and Distributed Processing, p.634-640, 2012.

[18] Van Luong, T.; Melab, N.; Talbi, E., "Parallel hybrid evolutionary algorithms on GPU," Evolutionary Computation (CEC), 2010 IEEE Congress on , vol., no., pp.1,8, 18-23 July 2010

[19] R. Storn, K. Price, Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. Journal of Global Optimization, 1997. v. 11, p. 341-359

[20] C. Sun, H. Zhou, L. Chen, Improved differential evolution algorithms, IEEE International Conference on Computer Science and Automation Engineering , vol. 3, p.142-145, 2012.

[21] Michalewicz, Z. Genetic Algorithms + Data Structures = Evolution Programs, 2 ed. Berlim: Springer Verlag, 1996.

[22] F. Herrera, M. Lozano, J. L. Verdegay, Tackling Real-Coded Genetic Algorithms: Operators and Tools for Behavioural Analysis. Artificial Intelligenc, v. 4, p. 265-319, 1998.

[23] J. Kennedy, R. C. Eberhart, Particle swarm optimization, In Proceedings of IEEE International Conference on Neural Networks, p. 1942-1948, 1995.

[24] E. Alba, G. Luque, Evaluation of Parallel Metaheuristics. In: PPSN-EMAA'06. Groelandia, 2006. P. 9-14.

[25] L. Idoumghar, M. Mahmoud, R. Schott, M. I. Aouad. Hybrid PSO-SA Type Algorithms for Multimodal Function Optimization and Reducing Energy Consumption in Embedded Systems. In: Applied Computational Intelligence and Soft Computing. Nova York: Hindawi Publishing Corp, 2011

[26] X. Li, A. Engelbrecht, M. G. Epitropakis Benchmark Functions for CEC'2013 Special Session and Competition on Niching Methods for Multimodal Function Optimization, Unpublished Report, Encontrado em http://goanna.cs.rmit.edu.au/~xiaodong/cec13-niching/competition/cec2013-niching-benchmark-tech-report.pdf, visitado em 06/08/2013.