Estratégias Evolutivas Paralelas em Otimização Multimodal

OMAR ANDRES CARMONA CORTES*
OSVALDO RONALD SAAVEDRA+

* USP - Universidade de São Paulo
ICMC - Inst. de Ciências Matemáticas e de Computação
Secretaria de Pós Graduação
Caixa Postal 668
13560-970 São Carlos - SP
ocortes@icmc.sc.usp.br

+ UFMA – Universidade Federal do Maranhão
DEE – Departamento de Engenharia de Eletricidade
Av. dos Portugueses s/n Campus Universitário
65080-040 - São Luis – MA
osvaldo@dee.ufma.br

Resumo: Este artigo apresenta um estudo de otimização multimodal através de estratégias evolutivas em ambiente distribuído. O paradigma implementado é chamado de migração onde várias populações evoluem em processadores separados e em determinadas épocas se produz um intercâmbio entre elas. Como base para os testes são utilizadas funções conhecidas como funções de *benchmarks* que permitem estabelecer algumas conclusões quanto à eficiência da computação paralela em algoritmos evolutivos assim como o efeito das migrações nos mesmos.

Palavras Chaves: Estratégias Evolutivas, Computação Paralela, Avaliação de Desempenho, Otimização Multimodal.

1. Introdução

A computação evolutiva tem sido utilizada com sucesso para resolução de complexos problemas de otimização. Esta abordagem tem como principal obstáculo à precisão da solução a ser encontrada, quanto mais próxima da solução ótima se quer chegar mais poder computacional e tempo de processamento são exigidos.

Este trabalho está concentrado no estudo das EE´s em ambiente distribuído. Cada indivíduo gerado é uma solução em potencial do problema. Os operadores evolutivos de recombinação e mutação são aplicados a cada indivíduo com o objetivo de se obter uma evolução consistente de melhora das soluções candidatas. Todo processo é repetido sucessivamente até alcançar uma solução aceitável ou atingir algum critério de parada.

Em termos computacionais esse processo pode ser demorado, ainda mais quando são utilizadas funções multimodais, ou seja, que apresentam muitos mínimos locais. Para lidar com este obstáculo, a computação paralela se apresenta extremamente atraente, haja visto o paralelismo natural das estratégias evolutivas. Outro ponto importante que deve ser destacado é que a utilização da computação paralela oferece um potencial de busca muito maior no espaço de soluções do problema, onde populações que evoluem mais rápido ajudam de tempos em tempos as populações mais lentas, abrindo caminho para percorrer uma infinidade de trajetórias, que dificilmente são visitadas por um processo seqüencial clássico. Neste trabalho é mostrado o impacto da computação paralela nesses dois pontos de vistas.

Este trabalho está dividido da seguinte maneira: na seção dois é dada uma introdução às estratégias evolutivas, a sua notação e é mostrado um algoritmo evolutivo genérico. A seções três mostra como o algoritmo foi paralelizado. E finalmente, as duas ultimas seções mostram como foi feita a implementação e os resultados dos testes.

2. Estratégias Evolutivas (EE's)

As EE's foram desenvolvidas inicialmente na Alemanha, na década de 60 [2], focalizando a resolução de problemas contínuos de otimização paramétrica, sendo estendidas recentemente para tratamento de problemas

discretos. A sub-seção a seguir dá uma pequena introdução aos conceitos básicos de EE's.

2.1 Conceitos Básicos

A computação evolutiva é dividida em três grandes áreas: algoritmos genéticos (AG's), programação evolutiva (PE's) e estratégias evolutivas (EE's) [1]. Em todos estes três paradigmas, é simulada uma evolução neo-Darwiniana, onde uma população de soluções candidatas de um problema é submetida a um processo de recombinação (cruzamento), mutação e competição pela sobrevivência. As EE's e PE's são semelhantes sendo que nestas últimas os indivíduos não são submetidos a cruzamentos. Os AG's enfatizam o aspecto cromossômico de cada indivíduo.

Nas EE's, um indivíduo é representado por um par de vetores reais da forma v=(x,σ), onde x representa o ponto de busca no espaço e σ o vetor de desvio padrão associado. Nas versões atuais, a descendência é obtida submetendo-se os indivíduos da geração a dois operadores: cruzamento e mutação. O cruzamento é feito de forma aleatória e a mutação é feita tipicamente através de uma perturbação Gaussiana de média nula e desvio padrão unitário, porém outros tipos de mutação são possíveis [1][2][3]. Observa-se que o parâmetro σ - que determina a mutação de x - também está sujeito ao processo de evolução também através de mutação. Esta é uma característica fundamental das EE's, que permite o auto-ajuste de seus parâmetros. Assumindo algumas hipóteses, é possível provar que as EE's convergem ao ótimo global com probabilidade 1, considerando um tempo de busca suficientemente longo.

Uma primeira versão de EE's focaliza um processo de busca no esquema 1 genitor - 1 descendente. Isto foi denominado (1+1)-EE , onde um único filho é criado a partir de um único genitor e ambos são confrontados numa competição por sobrevivência, onde a seleção elimina a solução mais pobre. Um aspecto negativo observado é a convergência lenta, além da busca ponto a ponto ser susceptível a estagnar em mínimos locais.

Outras versões foram desenvolvidas com o objetivo de resolver tais problemas. Estas estratégias são denominadas multi-indivíduos, onde o tamanho da população é maior que 1.

As EE's multi-membros foram aperfeiçoadas tendo-se atualmente dois principais tipos (μ+λ)–EE e (μ,λ)–EE. Na primeira, μ indivíduos produzem λ descendentes, gerando-se uma população temporária de (μ+λ) indivíduos, de onde são escolhidos μ indivíduos para a próxima geração.

Na versão (μ,λ)–EE, μ indivíduos produzem λ descendentes, com $\mu < \lambda$, sendo que a nova população de μ indivíduos é formada por apenas indivíduos selecionados do conjunto de λ descendentes. Assim, o período de vida de cada indivíduo é limitado a apenas uma geração. Este tipo de estratégia tem bom desempenho em problemas onde o ponto ótimo é em função do tempo, ou onde a função é afetada por ruído.

2.2 EE's: Algoritmo Padrão

Considerando cada elemento composto por um par de vetores na forma v = (x,σ), o algoritmo para o modelo (μ + λ) – EE é mostrado a seguir:

I. Inicializa-se uma população de μ indivíduos com variância 1 para cada posição de x.

II. Faz-se uma recombinação dos μ pais até gerar λ descendentes.

III. Faz-se a mutação dos descendentes. e das variâncias seguindo as seguintes expressões:

$$x_i^j = x_i^j + N(0,\sigma)$$

$$\sigma_i^j = \sigma_i^j . \exp(\tau' N(0,1) + \tau N_j(0,1))$$

onde i = 1,...,λ; j = 1,...,n. N(0,1) representa um número Gaussiano com média zero e variância 1, nota-se que esse número é o mesmo para todos os indivíduos quando multiplicado pelo fator τ'. O numero Gaussiano que multiplica a τ deve ser obtido independentemente para valor de j. Os parâmetros τ' e τ foram sugeridos por Back [4][5][6] como:

$$\tau' = (\sqrt{2n})^{-1} \quad \text{e} \quad \tau = (\sqrt[4]{2n})^{-1}$$

Note-se que na mutação dos descendentes a variância será diferente a cada geração (auto-adaptação)

IV. Avalia-se o *fitness*[1] de genitores e descendentes, onde serão escolhidos os μ indivíduos com os melhores valores de *fitness* (competição), os quais serão os pais na próxima geração.

V. Repete-se o processo a partir do passo II até ser atingido o critério de parada especificado.

No passo I a população inicial é gerada aleatoriamente, porém atendendo as restrições impostas pelo problema.

O passo II é iniciado o processo da geração da população de descendentes, que só estará finalizada após o término do passo III, ou seja, após a mutação.

No passo IV ocorre a competição entre todos os indivíduos, tanto genitores como descendentes. O critério de competição é a *fitness*, que neste problema

[1] Entende-se por *fitness* a medida do desempenho do individuo na geração atual. Neste caso, a fitness corresponde ao valor da função objetivo do problema.

corresponde à função objetivo a ser otimizada. Este processo é determinístico, levando em consideração que apenas μ indivíduos sobrevivam e passem para a próxima geração na qualidade de genitores. O critério de parada especificado no passo V é geralmente uma certa quantidade de gerações definidas pelo usuário.

A próxima seção discute como o algoritmo foi paralelizado.

3. Paralelização

Os algoritmos evolutivos, além de serem capazes de tratar funções não lineares e descontínuas, têm o potencial de poder utilizar processamento paralelo, sendo este aspecto extremamente importante, devido que o aspecto crítico da Evolução Simulada é o tempo consumido na resolução. Os paradigmas para paralelizar estes algoritmos tem como fonte de inspiração os princípios de evolução orgânica [12]. Estes modelos podem ser classificados em duas categorias:

- Modelo de Migração;
- Modelo de Difusão.

No Modelo de Migração, a idéia é executar versões seqüenciais em cada processador de uma máquina paralela e trocar algumas informações entre processadores durante o processo de busca. Supondo que há indivíduos integrando uma população em cada processador, o intercâmbio de informação pode ser considerado com a migração dos melhores indivíduos. Devido às características estocásticas dos algoritmos, é possível obter eficiências acima de 100% (desempenho super-linear). Em [13] é explicado que o melhor desempenho de uma versão paralela é obtido quando o ótimo local tem parâmetros em comum com o ótimo global. O modelo de Difusão representa uma versão de granulação fina onde cada indivíduo é colocado num processador e seleciona outro indivíduo com o qual trocará informações com objetivo de gerar um descendente. A eficiência deste tipo de paralelização depende da topologia onde seja implementada, sendo mais adequada as máquinas SIMD (única instrução - múltiplos dados). Topologias tipo malha e cúbicas são adequadas, pois cada indivíduo tem vários vizinhos (quatro, no caso de topologia tipo malha e no mínimo três, no caso de topologia cúbica) [15].

O multiprocessamento também pode ser viabilizado através das chamadas “plataformas baratas”, que consistem de uma rede de computadores e de um sistema de computação paralela, muitos destes já disponíveis sem custo em alguns sites da Internet.

A paralelização do algoritmo evolutivo com migração é relativamente simples. A abordagem de programação utilizada é a SPMD (*Simple Program Multiple Data* – Programa Único Dados Múltiplos), onde o mesmo programa é executado em cada processador. Neste caso o que diminui é a quantidade de gerações. Por exemplo, se seqüencialmente eram feitas nove mil gerações, ao se utilizarem três processadores, cada um dos processadores fará de forma independente três mil gerações.

O fato de apenas usar programas independentes em cada processador sem nenhuma forma de interação entre eles só irá influenciar na velocidade de processamento e não tem muita utilidade. Dessa forma, faz-se uma interação entre as populações através da migração. Este processo de interação entre as populações irá influenciar positivamente o processo de busca de uma melhor solução. Ela pode ser feita quando um certo número de gerações é atingido, neste ponto um determinado número de melhores indivíduos (ou apenas o melhor) de cada processador são enviados para os demais e a população mais forte que foi gerada por cada processador compete pela sobrevivência com as populações que estão chegando. A seguir é mostrado o algoritmo simplificado do processo de migração:

```
Programa Evolutivo;
Gerar população de μ indivíduos;
Ger ← G / Q_processdores;
Mig ← M;
Q_Ger ← 1;
Faça{
   Algoritmo seqüencial;
   Se (Ger % Mig = 0 e Ger ≥ Mig)
     Envia indivíduos mais fortes;
     Recebe população migrada;
     Competir;
     Estruturar novos pais;
   Fim-Se
   Q_Ger ← Q_Ger + 1;
Até (Q_Ger = Ger);
Fim-Programa Evolutivo;
```

onde, G é a quantidade de total de gerações. M representa de quantas em quantas gerações a migração deve ser feita. Q_processadores é a quantidade de processadores utilizados. Q_Ger é uma variável que controla a quantidade de gerações criadas. O símbolo % representa o resto da divisão de duas variáveis. (Ger % Mig = 0 e Ger ≥ Mig) verifica se um múltiplo de Mig foi alcançado para executar a migração. A próxima seção mostra como foi feita a implementação das EE’s.

4. Implementação

Para implementação do algoritmo foi utilizado o MPI (*Message Passing Interface*), que nada mais é do que uma extensão paralela para linguagens seriais. No caso

deste artigo foi utilizada a linguagem C, mas também existem versões do MPI para Fortran e C++. Maiores detalhes sobre MPI podem ser vistos em [10][8].

Para o processo de otimização, foram utilizadas as duas funções abaixo, também conhecidas como funções de *benchmark* [1].

$$\min f_1 = \sum_{i=1}^{n-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + \left(x_i - 1\right)^2\right]$$

$$\min f_2 = \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right]$$

A primeira equação (*f1*) é uma função unimodal. Para este tipo de equação existem métodos mais eficientes que os de EE's, mas serve para verificar o comportamento das EE's com esse tipo de função e para avaliar o desempenho da versão paralela.

A segunda equação (*f2*) é multimodal, ou seja, apresenta muitos mínimos locais. Ambas as funções foram escolhidas por se assemelharem às funções encontradas em sistemas de energia elétrica.

As duas funções acima possuem mínimos matematicamente calculados, apresentam limites para geração dos números aleatórios o que por conseqüência limita a mutação aos mesmos valores que são mostrados na Tabela 1.

Tabela 1 – Funções com seus mínimos e limites.

Função	Mínimo	Limites
f1	0	[-30,30]
f2	0	[-5.12,5.12]

O tamanho da população inicial foi de $\mu = 20$. A população de descendentes é de $\lambda = 100$. A número de gerações é de 3.000 por processador perfazendo um total de 9.000 gerações ao total.

Para se realizar a migração é feita uma passagem de mensagem a partir de cada processador para os demais. Para verificar a sensibilidade da estratégia com relação ao numero de migrações, foram realizados testes considerando o intercâmbio a cada 1000, 500, 200, 100, 50, 25 e 10 gerações, em outras palavras, 3, 6, 15, 30, 60, 120 e 300 migrações respectivamente.

5. Resultados Dos Testes

As simulações foram realizadas num computador multiprocessado IBM-SP2 cujas características são apresentadas na Tabela 2 seguindo a seguinte legenda:

I. Tipo de processador.

II. Quantidade de processadores disponíveis.

III. Velocidade do processador em Mhz.

IV. Memória de cada tipo de processador em MBytes.

V. Disco usado por cada processador em Gbytes.

VI. Barramento utilizado pelo processador em bits.

Tabela 2 - Características físicas do IBM-SP2.

I	II	III	IV	V	VI
wide	1	66.7	256	9	256
thin	2	66.7	256	9	128

5.1 Análise de Desempenho

A análise de desempenho foi feita através da utilização das métricas *speedup* e eficiência conforme[8][16][17].

As Tabelas 3 e 4 apresentam os resultados dos tempos de execução das equações *f1* e *f2* em três processadores. A primeira coluna representa a quantidade de migrações. A segunda coluna mostra o tempo total em segundos da execução em paralelo. A terceira e quarta coluna representam o *speedup* e a eficiência conforme as referências acima. O tempo de execução seqüencial é de 260.291 segundos. A quantidade total de gerações é de 9000, ou seja, 3000 gerações para cada processador para ambas as tabelas.

Tabela 3 – Resultado da Análise de Desempenho da *f1*.

Qtd. de Mig.	Tempo (s)	*Speedup*	Eficiência
3	86.9404	2.9939	0.9979
6	86.9453	2.9937	0.9978
15	87.0237	2.9910	0.9970
30	87.2930	2.9818	0.9939
60	87.7048	2.9678	0.9892
120	88.3979	2.9445	0.9815
300	90.4104	2.8789	0.9596

O tempo de execução seqüencial da *f2* é de 235.55 segundos. A Tabela 4 a seguir apresenta o mesmo padrão da Tabela 3.

Tabela 4 – Resultado da Análise de Desempenho para *f2*.

Qtd. de Mig.	Tempo (s)	*Speedup*	Eficiência
3	80.0307	2.9432	0.9810
6	80.2192	2.9363	0.9787
15	80.2826	2.9340	0.9780
30	80.5914	2.9227	0.9742
60	81.1815	2.9015	0.9671
120	81.2635	2.8985	0.9661
300	82.7020	2.8481	0.9493

Nota-se em ambas as tabelas, que apesar das migrações aumentarem consideravelmente, o tempo de execução não sofre um aumento drástico. Isto o corre porque o IBM-SP2 apresenta um sistema de comunicação entre seus processadores de alto desempenho chamado HPS (*High Performance Switch*). Desta forma o custo do processamento não é dominado pela comunicação, como conseqüência, o *speedup* alcançado é próximo do ideal. Como era de se esperar, o mesmo comportamento ocorre com a eficiência.

5.2 Análise da Sensibilidade do Resultado

Nas próximas tabelas é apresentado o efeito da migração na qualidade da solução obtida (função objetivo). A Tabela 5 mostra os mínimos que foram alcançados por Yao e Liu em [1], a terceira coluna da tabela representa o mínimo encontrado na versão seqüencial do algoritmo com 9000 gerações, e a quarta coluna representa os mínimos encontrados na versão paralela implementada neste trabalho. Os resultados reportados por Yao e Liu em [1] foram obtidos utilizando estratégia sem recombinação. Nota-se dessa forma que a recombinação afeta diretamente na precisão da solução.

Tabela 5 – Funções com os mínimos alcançados por Yao e os mínimos alcançados neste trabalho.

Função	Mínimo Yao&Liu	Mínimo Seqüencial	Mínimo Paralelo
f1	33.28	29.554	0
f2	70.82	28.554	0.003

A Tabelas 6 e 7 mostram os menores valores alcançados de acordo com a quantidade de migrações para *f1* e *f2*. A primeira coluna representa a quantidade de migrações utilizadas. A segunda coluna exibe o menor valor alcançado pela função objetivo. A última coluna mostra a quantidade de processadores que conseguiu atingir a menor solução.

Tabela 6– Sensibilidade da solução à migração de populações para f1.

Qtd. de Migrações	Mínimo *f(x)*	Qtd. de Proc.	Tempo (s)
3	44.7347	1	86.9453
6	33.3329	1	86.9453
15	28.9929	1	87.0237
30	28.9770	1	87.2930
60	28.9740	1	87.7048
120	28.9731	3	88.3979
300	28.9730	3	90.4104

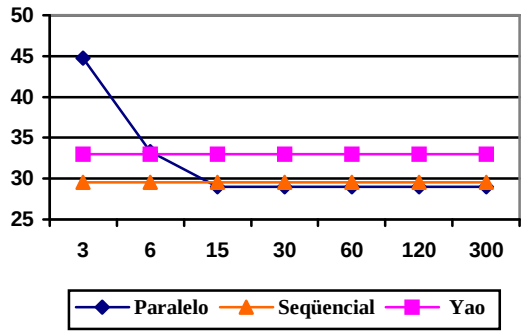

Figura 1 - Gráfico de sensibilidade da *f1*.

Na Tabela 6 os processadores não chegam exatamente ao mesmo mínimo, mas sim a valores bem próximos um do outro. Durante a execução dos testes conseguiu-se chegar uma vez no mínimo matemático da função, o que é considerado extremamente raro, pois todo o processo é aleatório e depende diretamente da solução inicial. A Figura 1 mostra o gráfico de sensibilidade baseado na Tabela 6 e pode-se notar que a partir de 15 migrações a evolução não é significativa e tende a se estabilizar.

Tabela 7 – Sensibilidade da solução à migração de populações para f2.

Qtd. de Migrações	Mínimo *f(x)*	Qtd. de Proc.	Tempo (s)
3	5.008	1	80.0307
6	4.4941	1	80.2192
15	0	1	80.2826
30	0	1	80.5914
60	0	1 - 2	81.1815
120	0	2	81.2635
300	0	3	82.7020

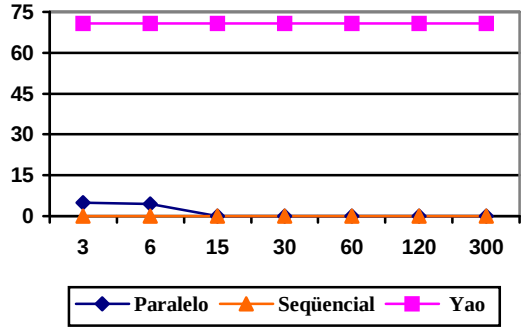

Figura 2 - Gráfico de sensibilidade da *f2*.

Na Figura 2 que é construída a partir da Tabela 7, observa-se que a partir de 15 migrações um dos

processadores encontra a solução ótima da função. Quando são utilizadas 60 migrações em determinadas execuções dois processadores conseguem chegar a solução ótima. Pode-se constatar também que a partir de 300 migrações quase sempre os três processadores chegaram a solução ótima. Dessa forma, vê-se que a migração de populações torna a competição mais intensa entre os indivíduos fazendo-os evoluir mais rapidamente na busca de melhores soluções.

Apesar da versão seqüencial chegar bem próximo da solução ótima, o tempo de processamento exigido é quase três vezes a mais que a versão com maior quantidade de migrações.

Observando-se o comportamento da *f2* através da Figura 2 e da Tabela 7, nota-se também que as EE's são adequadas para otimização de funções multimodais.

6. Conclusões

Neste trabalho foi apresentada uma proposta para utilização da computação paralela na resolução de problemas de otimização através de estratégias evolutivas. Os resultados mostraram que a computação paralela, quando aplicada a algoritmos evolutivos, é benéfica em dois aspectos:

- Diminuição do tempo de processamento; e
- Obtenção de soluções de mais qualidade.

Duas funções *benchmark* amplamente utilizadas nas referencia foram utilizadas como base de testes. O algoritmo foi testado numa máquina IBM-SP2 multiprocessada com excelente ganho de desempenho e de qualidade de soluções, atingindo nos casos testados o ótimo global.

7. Referências

[1] Yao, X.; Liu, Y. "**Fast Evolution Strategies**", in Proc. 5th Annual Conf. Evolutionary Programming, Cambrige, MA: MIT Press, pp. 451-460, 1993.

[2] Gomes, J. R.; Saavedra, O. R., "**Optimal Reactive Power Dispatch using Evolutionary Computation: Extend Algorithms**", IEE Proc.-Gene. Tranmition. Distrib., Vol. 146, No. 6, 1999.

[3] Nogueira, M.L.; O. R. Saavedra, "**Estratégias Evolutivas Aplicadas à Resolução de Otimização Multimodal**", Simpósio Brasileiro de Automação Inteligente, 1999.

[4] Bäck, T; Schwefel, H. P., "**An Overview of Evolutionary Algorithms for Parameters Optimization**", Evolutionary Computation, pp. 1-27, 1993.

[5] Bäck, T; Hammel, U.; Schwefel, H. P., "**Evolutionary Computation: An Overview**", in Proc. 3rd IEEE Conf. on Evolutionary Computation, Piscataway, NJ: IEEE Press, pp. 20-29, 1996.

[6] Bäck, T; Rudolph, G.; Schwefel, H. P., "**Evolutionary Programming and Evolution Strategies: Similarities and Differnces**", in Proc. od European Conf. on Alife, Granda, Spain, 1995.

[7] Hockney, R. W., "**The Science of Computer Benchmarking**", Philadelphia: society for Industrial and Applied Mathematics, 1996.

[8] Carmona Cortes, O. A., "**Desenvolvimento e Avaliação de Algoritmos Numéricos Paralelos**", Dissertação de Mestrado, ICMC, Abril, 1999.

[9] Potter, M. A.; Jong, D. A. De, "**Cooperative Coevolution: An Architecture for Evolving Coadapted Subcomponents**", Evolutionary Computation, vol. 8, n. 2, MIT Press, 2000.

[11] MacDonald, N., Minty, E., Harding, T., Browa, S., "**Writing Message-Passing Parallel Programs with MPI – Course Notes**". The University of Edinburgh, 1994.

[12] F. Hoffmeister, "**Scalable Parallelism by Evolutionary Algorithms"**, Applied Parallel and Distributed Optimization, Lecture Notes in Mathematical Systems and Economics, Springer, 1991.

[13] P. Spiessens, B. Manderick, "**A Massively Parallel Genetic Algorithm: Implementation and First Analysis**", Proc. of IV Conference of Genetic Algorithms, pp. 279 - 286, Morgan Kaufman, San Mateo , 1991.

[14] J.P. Cohoom, et al., "**Punctual Equilibria: A parallel Genetic Algorithm**", Proceedings of Second International Conference on Genetic Algorithms, 1987.

[15] R. Tenese, "**Parallel Genetic Algorithm for a Hypercube**", Proceedings of Second International Conference on Genetic Algorithms, 1987.

[16] Navaux, P. O. A., "**Introdução ao Processamento Paralelo**". RBC – Revista Brasileira de Computação, v. 5, nº 2, pp.31-43. Outubro, 1989.

[17] Misra, C. "**Parallel Program Design: A Foundation**", Texas: Addison-Wesley Publishing Company, 1989. 516p.

Agradecimentos

Os autores agradecem o apoio financeiro das instituições de fomento à pesquisa CNPq e CAPES.