# Análise do Desempenho de uma Implementação Paralela da Rede Neural Perceptron Multicamadas Utilizando Variável Compartilhada

Luís F. W. Góes, Fabrício Roulin, Luiz E. S. Ramos, Luis E. Zárate, Carlos A. P. S. Martins
Departamento de Ciência da Computação, Pontifícia Universidade Católica de Minas Gerais
Belo Horizonte, MG, 30535-610, Brasil

## RESUMO

Neste trabalho apresentamos uma análise do desempenho de uma implementação da rede neural perceptron multicamadas onde o treinamento, baseado no algoritmo "retro-propagação do erro", é realizado de forma paralela, utilizando-se o modelo de variável compartilhada. Nossa **principal meta** é o desenvolvimento de uma implementação paralela da rede neural perceptron multicamadas utilizando o modelo de programação paralela de variável compartilhada. Nós analisaremos o desempenho da implementação paralela em relação à implementação seqüencial tradicional, variando a topologia da rede (número de entradas e número de neurônios). Identificaremos situações em que o uso de processamento paralelo possibilite a diminuição do tempo de treinamento. Também identificaremos o tamanho ideal do grão ou quantidade de processamento, que proporcione ganho maior de desempenho.

## 1. INTRODUÇÃO

A cada dia cresce o número de aplicações de Redes Neurais Artificiais (RNA) nas diversas áreas do conhecimento como economia, administração, robótica, engenharia e outras [2][5][9]. É conhecido o bom desempenho computacional quando a rede está em operação, o que não é verdade quando está ainda no processo de treinamento (a rede pode atingir tempos de minutos até dias, para associar os conjuntos de dados a ela fornecidos). Existem diversos tipos de redes neurais, cada uma mais apropriada a uma categoria de aplicações. A rede perceptron multicamadas *"feedforward"* totalmente conectada, aproxima-se das teorias conexionistas que procuram descrever o funcionamento do cérebro, esta se destaca em relação as demais, por possuir uma grande capacidade de aprendizado e generalização proporcionado pela multiplicidade de neurônios organizados hierarquicamente em camadas [9].

No aprendizado da rede neural é utilizada a estrutura de aprendizado supervisionado, onde para cada conjunto de entrada é associado um conjunto de saída. Durante o treinamento da rede, o ajuste dos pesos pode ser feito através do algoritmo "retro-propagação-do-erro".

O treinamento é um dos grandes problemas no uso de redes neurais, pois ele pode ser extremamente demorado. Diante de aplicações em tempo real ou que necessitem de um tempo de resposta muito pequeno, o treinamento da rede torna-se um ponto de contenção. Com o intuito de se resolver este problema, vêm sendo utilizadas diversas alternativas para a diminuição do tempo de treinamento da rede neural, dentre elas podemos citar o uso de circuitos integrados, computação reconfigurável, algoritmos genéticos e computação paralela [1][2][5][8].

A computação paralela possui dois principais modelos de programação: passagem de mensagens e variável compartilhada. No modelo de passagem de mensagens, cada máquina possui uma memória própria. Então é necessário que as máquinas troquem mensagens entre elas sobre uma rede, para requisitar dados remotos e também para sincronização dos processos. Os programas escritos para multicomputadores ou máquinas de memória distribuída são baseados neste modelo. Normalmente eles utilizam bibliotecas de passagem de mensagens como PVM e MPI [4][6].

Já no modelo de variável compartilhada, uma variável é colocada na memória local de um multiprocessador ou máquina de memória compartilhada, a qual todos os processadores têm acesso. Neste caso, toda a comunicação entre os processos é feita através da memória principal. Os programas baseados neste modelo, normalmente utilizam bibliotecas de suporte como OpenMP, POSIX, Win e Java threads [4][6].

### MOTIVAÇÃO

As redes neurais artificiais têm sido utilizadas para solucionar problemas de diversas áreas do conhecimento. Mas o treinamento destas redes pode ser extremamente demorado. Em aplicações de tempo real que necessitam do re-treinamento da rede, as RNAs não conseguem solucionar o problema em tempo hábil.

A computação paralela tem sido amplamente utilizada para aumentar a vazão ou diminuir o tempo de resposta de vários tipos de aplicações em geral [7]. Por este motivo, resolvemos utilizar a computação paralela para agilizar o treinamento de uma RNA perceptron multicamadas. Especificamente, iremos utilizar o modelo de programação de variável compartilhada, pois ele é mais adequado para este tipo de problema, onde a granularidade é pequena.

### METAS E OBJETIVOS

O principal objetivo deste trabalho é a análise do desempenho de uma implementação paralela da rede

neural perceptron multicamadas que utiliza o modelo de variável compartilhada. Nós analisaremos o desempenho da implementação paralela em relação à implementação seqüencial tradicional, variando a topologia da rede (número de entradas e número de neurônios). Como objetivos específicos podemos destacar: a identificação das situações em que o uso de processamento paralelo possibilita a diminuição do tempo de treinamento da rede; a identificação do tamanho ideal do grão ou quantidade de processamento ideal, que proporciona um ganho maior de desempenho.

A **principal meta** deste trabalho é o desenvolvimento de uma implementação paralela da rede neural perceptron multicamadas utilizando o modelo de programação paralela de variável compartilhada.

## 2. REDE NEURAL PERCEPTRON MULTICAMADAS

A topologia da rede multicamadas considerada neste trabalho corresponde a uma rede de duas camadas de neurônios, com N entradas na camada entrada; L neurônios na camada escondida e M neurônios na camada de saída. É também considerado que os neurônios possuem entradas de polarização cada um. A função não linear sigmóide foi escolhida neste trabalho como a função de transferência do axônio, por ser a mais consistente com a biofísica do neurônio biológico. A Figura 1 mostra a topologia da rede usada neste trabalho.

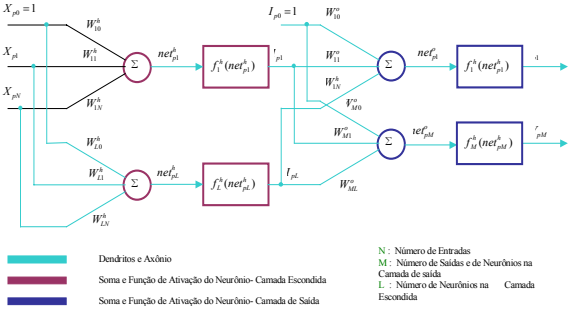


Fig. 1 – Topologia da rede neural multicamadas

Seja o conjunto de treinamento mostrado na Eq.(1), onde $X_i$, $i = 1,...,P$ são as entradas normalizadas e $Y_i$, $i = 1,...,P$ são as saídas desejadas da rede normalizadas pela função sigmóide, segue-se abaixo a descrição das etapas e equações matemáticas necessárias no algoritmo seqüencial para o treinamento da rede neural multicamadas.

$$(\underline{X}_1,\underline{Y}_1),...,(\underline{X}_p,\underline{Y}_p) \qquad \textbf{(equação 1)}$$

I. Aplicar o conjunto "p" representado pela Eq.(2), onde N é o número de entradas.

$$\underline{X}_p = (X_{p1},...,X_{pN})^T \qquad \textbf{(equação 2)}$$

II. Calcular a saída linear da camada escondida de acordo com a Eq.(3), onde $net^h_{pj}$ é produto dos pesos pelas entradas e $W^h_{ij}$ contém o peso do neurônio i e entrada j para a camada escondida.

$$net^h_{pj} = \sum_{i=0}^{N} W^h_{ji} X_{pi} \qquad j = 1,...,L \qquad \textbf{(equação 3)}$$

III. Calcular a saída da camada escondida através da Eq.(4), onde $I_{kj}$ contém as respostas da função sigmóide $f^h_j(net^h_{pj})$ para a camada escondida.

$$I_{pj} = f^h_j(net^h_{pj}) \quad j = 1,...,L \qquad \textbf{(equação 4)}$$

IV. Calcular a saída linear da camada de saída de acordo com a Eq.(5), onde $net^h_{pk}$ é produto dos pesos pelas entradas e $W^o_{kj}$ contém o peso do neurônio i e entrada j para a camada de saída:

$$net^o_{pk} = \sum_{j=0}^{L} W^o_{kj} I_{pj} \qquad k = 1,...,M \qquad \textbf{(equação 5)}$$

V. Calcular a saída da rede através da Eq.(6), onde $Y_{kj}$ contém as respostas da função sigmóide $f^o_k(net^o_{pk})$ para a camada de saída.

$$Y_{pj} = f^o_k(net^o_{pk}) \quad k = 1,...,M \qquad \textbf{(equação 6)}$$

VI. Calcular o erro da camada de saída através da Eq.(7), onde $\Gamma^o_{pk}$ contém os erros calculados da camada de saída.

$$\Gamma^o_{pk} = (\psi_{pk} - Y_{pk}) f^{o'}_k(net^o_{pk}) \quad k = 1,...,M$$

**(equação 7)**

VII. Calcular o erro na camada escondida através da Eq.(7), onde $\Gamma^h_{pk}$ contém os erros calculados da camada escondida.

$$\Gamma^h_{pk} = f^{h'}_j(net^h_{pj}) \sum_{k=1}^{M} \Gamma^o_{pk} W^o_{kj} \quad j = 1,...,L$$

**(equação 8)**

VIII. Atualizar os pesos da camada de saída de acordo com a Eq.(9), onde $\mu$ é o fator de ajuste.

$$W^o_{kj}(t+1) = W^o_{kj}(t) + \mu \Gamma^o_{pk} I_{pj}$$

$k = 1,...,M$ e $j = 0,...,L$ **(equação 9)**

IX. Atualizar os pesos da camada escondida de acordo com a Eq.(10).

$$W_{ji}^{h}(t+1) = W_{ji}^{h}(t) + \mu\Gamma_{pj}^{h}X_{pi}$$
$$j = 1,...,L \text{ e } i = 0,...,N \quad \textbf{(equação 10)}$$

X. Verificar a tolerância através da Eq.(11), onde $E_p$ é o erro global e $\delta$ é o erro médio.

$$E_p = \frac{1}{2}\sum_{k=1}^{M}\delta_{pk}^{2} < \text{tolerância} \quad \textbf{(equação 11)}$$

Neste trabalho foi escolhida a função sigmóide, que é representada no algoritmo de treinamento pelas equações Eq.(12) e Eq.(13).

$$f_k^o(net_{pk}^o) = \frac{1}{1+\exp^{-net_{pk}^o}} \quad \textbf{(equação 12)}$$

$$f_k^o{}'(net_{pk}^o) = Y_{pk}(1-Y_{pk}) \quad \textbf{(equação 13)}$$

## 3. IMPLEMENTAÇÃO PARALELA DA REDE NEURAL

A nossa implementação paralela da rede neural multicamadas é um algoritmo mestre-escravo que segue a metodologia proposta por Ian Foster [3] para o desenvolvimento de programas paralelos.

Segundo esta metodologia, o projeto de um algoritmo paralelo é consistido de quatro etapas: partição, comunicação, aglomeração e mapeamento. Primeiramente, nós observamos que os neurônios de uma mesma camada são independentes, ou seja, podem ser executados em paralelo, mas que o processamento de cada camada é obrigatoriamente seqüencial. Então determinamos que a partição seria entre os neurônios de uma mesma camada.

A comunicação e a sincronização entre os processos são realizadas na memória principal, por meio de variáveis compartilhadas e semáforos.

Nas etapas de aglomeração e mapeamento determinamos que a carga de trabalho seria distribuída entre dois processos, um mestre e um escravo, aonde o mestre se encarregaria de processar metade dos neurônios de cada camada e o escravo se encarregaria da outra metade. Cada um destes processos é alocado a processadores distintos do multiprocessador.

De acordo com a Tab. I, podemos perceber que nem todo o treinamento da rede é realizado em paralelo. Para evitar um grande número de sincronizações, somente o processo mestre realiza o cálculo do erro das camadas e também o cálculo da saída e do ajuste dos pesos da camada de saída, pois o número de neurônios da camada de saída é fixo e de tamanho desprezível no nosso experimento. Por estes motivos certas etapas só são processadas seqüencialmente pelo processo mestre.

| Etapa | Descrição da Etapa | Processamento |
|---|---|---|
| 1 | Escolha do conjunto de treinamento | Seqüencial |
| 1.1 | Sincronização (Semáforo 1) | Paralelo |
| 2 | Cálculo da saída linear e não linear da camada escondida | Paralelo |
| 2.1 | Sincronização (Semáforo 2) | Paralelo |
| 3 | Cálculo da saída linear e não linear da camada de saída | Seqüencial |
| 4 | Calcular erro da camada de saída | Seqüencial |
| 5 | Calcular erro da camada escondida | Seqüencial |
| 6 | Atualizar os pesos da camada de saída | Seqüencial |
| 6.1 | Sincronização (Semáforo 3) | Paralelo |
| 7 | Atualizar os pesos da camada escondida | Paralelo |
| 8 | Verificação da Tolerância do erro | Seqüencial |

Tabela I – Algoritmo Paralelo da Rede Neural Multicamadas

O processo mestre também é responsável por escolher a linha do conjunto de entradas a ser processada em cada iteração e verificar a tolerância do erro, informando ao escravo o término do treinamento da rede.

## 4. MÉTODO EXPERIMENTAL

Nosso método experimental é composto de três etapas: a implementação da versão seqüencial, a implementação da versão paralela e a análise de desempenho.

Na primeira etapa, uma versão seqüencial da rede neural perceptron multicamadas é desenvolvida e validada através da comparação do erro médio global encontrado após o treinamento, feito pela versão seqüencial e pela função do MATLAB que implementa a rede neural multicamadas. O conjunto de treinamento utilizado para validação foi o proposto em [9], onde ele é utilizado para o controle de laminadores. O erro médio global é calculado através da média da soma dos erros encontrados em cada saída da rede para cada linha do conjunto de entradas.

Na segunda etapa, a versão paralela da rede neural é implementada e validada através da comparação com os resultados obtidos pela versão seqüencial.

Na última etapa, a análise de desempenho é realizada através da variação do número de entradas e do número de neurônios da camada escondida. O tempo de resposta para cada iteração do treinamento da rede é coletado. São

gerados os gráficos do tempo de resposta, speedup e eficiência para facilitar a análise dos resultados obtidos durante o experimento.

### CONFIGURAÇÃO EXPERIMENTAL

Os testes foram realizados em um dual-processador, onde cada processador é um Pentium III 933 Mhz com a memória principal de tamanho 1 GB. A implementação paralela utiliza a biblioteca winthreads e foi desenvolvida no compilador C++ Borland Builder.

A configuração inicial da rede neural foi a seguinte: pesos iniciais iguais a -1 ou 1 consecutivamente, fator de ajuste igual a 0.01 e tolerância igual a $10^{-11}$.

O uso de um dual-processador nos levou a utilizar apenas 2 processos ou threads, para uma melhor utilização de cada processador. O número de neurônios e o número de entradas foram variados em potências de 2 (de 8 até 1024) e o tempo de resposta foi medido para cada 1000 iterações do treinamento da rede.

## 5. RESULTADOS EXPERIMENTAIS

Na primeira etapa, implementamos a versão seqüencial da rede neural perceptron multicamadas. O erro médio global encontrado pela versão seqüencial foi aproximadamente (diferença na oitava casa decimal) o mesmo encontrado pelo MATLAB. Com este resultado a versão seqüencial foi validada.

Na segunda etapa, comparamos a versão seqüencial e a paralela da mesma maneira, obtendo resultado aproximado ao da versão seqüencial, pois o mesmo resultado é difícil de ser alcançado devido a grande variação entre os treinamentos de uma mesma rede. Com isso, a versão paralela foi validada.

Na última etapa, o tempo de resposta gasto tanto pela versão seqüencial quanto pela paralela, para executar o treinamento por 1000 iterações, foi medido. O número de neurônios e o número de entradas foram variados em potências de 2 (de 8 até 1024).

Na figura 2, que mostra o tempo de resposta para a versão seqüencial, podemos observar que à medida que se aumenta o número de entradas e/ou o número de neurônios, o tempo de resposta também aumenta, pois o grão ou a quantidade de processamento vai se tornando maior. É importante notar que faz pouca diferença, em termos de quantidade de processamento, utilizar 1024 neurônios e 512 entradas ou 512 neurônios e 1024 entradas, pois a multiplicação entre esses termos resulta no número de pesos que a rede possui e a quantidade de processamento está diretamente relacionada ao número de pesos da rede. Existem também outros fatores como a comunicação, a sincronização e o overhead do sistema operacional, que estão relacionados com desempenho do programa.

Ainda na figura 2, o maior tempo de resposta encontrado foi 140,391 segundos com o número de neurônios e entradas iguais a 1024 (número de pesos igual a $2^{20}$). Já o menor tempo de resposta encontrado foi 0,015 segundos com o número de neurônios e entradas iguais a 8 (número de pesos igual a 64).

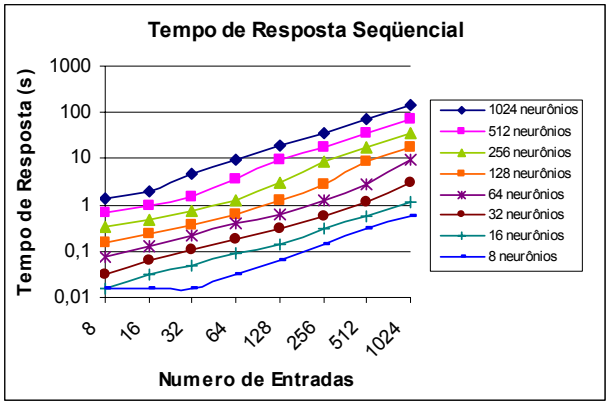


Figura 2 – Tempo de resposta gasto pela versão seqüencial da rede neural (1000 iterações)

Na figura 3, podemos observar que a versão paralela teve o mesmo comportamento da versão seqüencial, à medida que se aumenta o número de entradas e/ou o número de neurônios, o tempo de resposta também aumenta. Na versão paralela, o maior tempo de resposta encontrado foi 69,687 segundos, também com o número de neurônios e entradas iguais a 1024, ou seja, gastou a metade do tempo (speedup igual 2) da versão seqüencial. Já o menor tempo de resposta encontrado foi 0,125 segundos com o número de neurônios e entradas menores que 32. Neste caso não houve ganho com a versão paralela, chegando a um speedup igual a 0,12, ou seja, 10 vezes pior que a versão seqüencial.

Isso ocorre devido ao pequeno tamanho do grão de processamento e ao alto custo para a sincronização dos processos. O paralelismo só se torna interessante quando a quantidade de processamento prevalece sobre o custo para comunicação e sincronização dos processos.

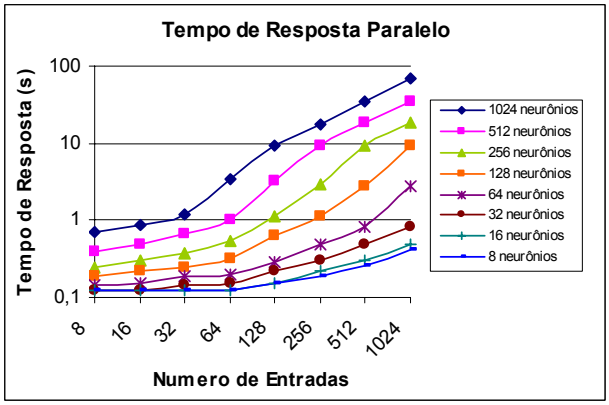


Figura 3 – Tempo de resposta gasto pela versão paralela da rede neural (1000 iterações)

Na figura 4, podemos observar o speedup, ou seja, o ganho da versão paralela sobre a seqüencial para todos os testes realizados. É importante notar que o paralelismo só foi vantajoso (speedup maior que 1) a partir de 2048 ou

$2^{11}$ pesos, como exemplo, podemos citar a situação onde a topologia da rede possui 128 neurônios e 32 entradas. O speedup igual a 2 é alcançado com aproximadamente 8192 ou $2^{13}$ pesos. O maior speedup alcançado foi 3,83 com o número de neurônios igual a 1024 e o número de entradas igual a 32. Apesar de fazer pouca diferença o uso de 32 neurônios e 16 entradas ou o contrário, um maior número de neurônios contribui um pouco mais para aumentar o speedup do treinamento da rede do que o número de entradas.

Um resultado interessante foi alcançado: mesmo utilizando apenas dois processadores, obtivemos valores de speedup maiores que dois (valor ideal de speedup, nesse caso). Isso acontece devido à utilização de caches internos em cada um dos processadores. Na versão seqüencial, na medida em que aumentamos o número de pesos, o cache do processador vai armazenando os dados necessários para o treinamento da rede (incluindo os pesos das entradas). Até se atingir um número de pesos igual a $2^{13}$, o speedup obtido é igual a 2. Isso indica que estão ocorrendo poucos (ou quase nenhum) *"cache miss"*. Quando o número de pesos passa de $2^{13}$, o cache da máquina seqüencial fica cheio, passando a utilizar políticas de substituição e gerando vários "*cache miss*". Notamos que existe um tamanho ideal do grão para o qual este experimento apresenta o melhor speedup (em torno de $2^{16}$ pesos).

Na versão paralela, que possui o dobro de cache (dois processadores possuem dois caches) do multiprocessador, ainda é possível manter todos os dados necessários para o treinamento da rede no cache. Isso evita *"cache miss"* ao se requisitar um dado. Ainda na versão paralela, após atingir o tamanho ideal do grão, os dois processadores se encontram com os caches cheios e começam a utilizar políticas de substituição. A partir de $2^{17}$ pesos os caches dos dois processadores ficam quase sempre cheios, ocorrendo vários "*cache miss*". Então o speedup volta a ter o valor igual a 2.

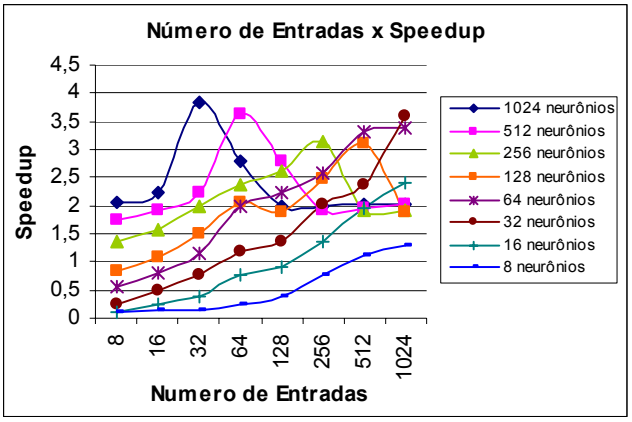


Figura 4 – Número de entradas x speedup para todas as variações do número de neurônios e entradas

Na figura 5, podemos observar a eficiência, ou seja, a utilização média de cada processador da versão paralela em relação a seqüencial, para todos os testes realizados.

É importante notar que a eficiência alcançou o valor 1,8 no melhor caso. A eficiência também foi muito baixa para um pequeno número de entradas e neurônios, chegando ao valor de 0,06 com número de neurônios e entradas iguais a 8.

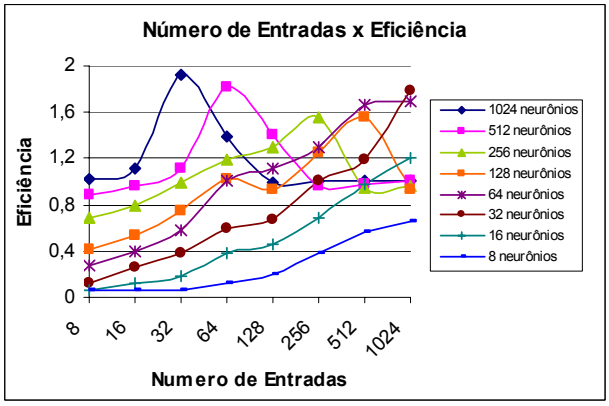


Figura 5 – Número de entradas x eficiência

## 6. CONCLUSÃO

A implementação paralela de uma rede neural perceptron multicamadas que utiliza o modelo de variável compartilhada, apresenta um alto desempenho em relação à implementação seqüencial.

Baseado nos resultados obtidos, nós concluímos que a nossa implementação paralela obteve um alto ganho de desempenho no treinamento da rede, ao utilizar-se uma topologia onde o número de pesos foi grande (maior que $2^{11}$). Caso contrário, ou seja, quando topologia possui poucos neurônios e poucas entradas, a implementação seqüencial ainda é mais vantajosa.

De acordo com os resultados obtidos, o tamanho ideal do grão está em torno de $2^{16}$, apresentando um ganho de quase 4 vezes no melhor caso, onde o número de neurônios é igual a 1024 e o número de entradas igual a 32.

O tamanho do cache do processador mostrou ser um ponto de contenção tanto para a implementação seqüencial quanto para a paralela. O custo para sincronização e comunicação entre os processos é muito alto, mas poderia ser reduzido com o uso de bibliotecas específicas e otimizadas para gerenciar a sincronização entre os processos.

A **principal contribuição** deste trabalho foi uma implementação paralela da rede neural perceptron multicamada que utiliza o modelo de variável compartilhada, onde foi demonstrada a sua capacidade de diminuição do tempo de treinamento, ou seja, o aumento do desempenho no treinamento da rede.

## 7. TRABALHOS FUTUROS

Como trabalhos futuros, nós pretendemos desenvolver uma implementação paralela da rede neural,

que utilize o modelo de passagem de mensagens. Também testaremos a implementação atual em multiprocessadores com mais de dois processadores. Seria interessante a análise do desempenho desta implementação em situações onde fossem variados o número de neurônios na camada de saída e também o número de camadas da rede neural.

## 8. AGRADECIMENTOS

Gostaríamos de agradecer ao Instituto de Informática, à Pró-Reitoria de Pesquisa e Pós-Graduação (PropPG), PIBIC, ao CNPq por nos conceder bolsas de pesquisa, aos amigos e companheiros de trabalho do LSDC e do LICAP.

## 9. REFERÊNCIAS BIBLIOGRÁFICAS

[1] Erdogan, S. S., Wahab, Abdul B. "Hierarchical decomposition model for reconfigurable architecture" High-Speed Computing, Digital Signal Processing and Filtering Using Reconfigurable Logic Conference, SPIE 96.

[2] Fathy, S.K., Syiam, M.M. "A parallel design and implementation for backpropagation neural network using MIMD architecture" Electrotechnical Conference MELECON '96, 8th Mediterranean , 1996.

[3] Foster., I. "Designing and Building Parallel Programs", On-line book, 1995. (http://www-unix.mcs.anl.gov/dbpp/text/book.html)

[4] Hwang, K.; Xu, Z. "Scalable Parallel Computing:Technology, Architecture, Programming", Mcgraw-Hill, 1998.

[5] Pinho, A. J. "Paralellizing the back-propagation algorithm using PVM", 7th Portuguese Conference on Pattern Recognition, RecPad 95, Portugal, 1995.

[6] Ramos, L. E. S., Góes, L. F. W., Martins, C. A. P. S. "Prober: Uma Ferramenta de Análise Funcional e de Desempenho de Programas Paralelos e Configuração de Cluster", *2º Workshop de Sistemas Computacionais de Alto Desempenho*, Brasil, 2001.

[7] Ramos, L. E. S., Góes, L. F. W., Martins, C. A. P. S. "Parallel Image Filtering Using WPVM in a Windows Multicomputer", CSITeA 2002.

[8] R.O. Rogers, D.B. Skillicorn. "Using the BSP cost model to optimize parallel neural network training" In Workshop of Biologically Inspired Solutions to Parallel Processing Problems (BioSP3), in conjunction with IPS/SPDP'98, March 1998.

[9] Zárate, L., Roulin, F. B. "Obtaining of Fuzzy Rules via Sensitivity Factors and its Application in Cold Rolling Process", SCI 2001.