Simulação de Redes Neuronais em Máquinas SIMD *

Juliana Pereira Salles
Maria Augusta Guimarães Vieira
Wagner Meira Junior
Marcio Luiz Bunte de Carvalho

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Caixa Postal 702 CEP 30.161-970 Belo Horizonte - MG
Telefone: (031) 443-4088 FAX: (031) 443-4352
E-Mail: meira@dcc.ufmg.br ou mlbc@dcc.ufmg.br

Resumo

Uma característica importante das redes neuronais é a flexibilidade que elas oferecem para a solução de problemas complexos quando comparadas às técnicas convencionais. Outra característica apresentada por estes modelos é a alta complexidade requerida para a sua simulação em arquiteturas tradicionais. O aumento da complexidade e das dimensões das redes neuronais a serem simuladas faz com que seja necessária a busca de implementações eficientes viabilizando a sua utilização.

Este trabalho apresenta uma proposta de implementação de redes neuronais em máquinas SIMD, que se baseia no grafo de dependência de dados da própria rede neuronal. Foi implementado um algoritmo de simulação de redes de grande porte em uma máquina SIMD com 4096 processadores de 1 bit. A implementação paralela mostrou ser mais adequada para redes de maior tamanho quando confrontada com a versão seqüencial.

Abstract

An important feature of neural networks is their flexibility in dealing with complex problems when compared to convetional techniques. Another characteristic is the high computational cost required by its simulation by traditional architectures. The increase in complexity and dimension of the neural networks which are to be simulated make it necessary to look for efficient implementations which make its use feasible.

In this work it is presented a proposal of implementation of neural networks in SIMD machines, which is based on the data dependency graph of the neural network itself. It has been implemented an algorithm for simulation of large networks using a SIMD machine with 4096 one-bit processors. The parallel implementation proved to be more suitable for larger networks in comparison to the sequential one.

*Este projeto foi financiado pela FAPEMIG Tec 1113/90

1 Introdução

Redes neuronais são modelos que se basearam no cérebro humano, onde o funcionamento se dá pela interação de um grande número de células simples, devidamente interconectadas. Uma rede neuronal armazena as informações nos pesos associados às conexões entre pares de neurônios e no nível de ativação de cada neurônio [13]. Utilizando algoritmos simples, os sistemas neuronais são robustos e capazes de integrar grandes quantidades de informação. Entretanto, sistemas de computação que proporcionem um alto desempenho são requeridos para simular redes neuronais de maiores dimensões capazes de manipular complexos sistemas de controle ou realizar uma emulação dos sentidos humanos. Assim, implementações eficientes de redes neuronais se tornaram imprescindíveis para a evolução da área, pois tornam praticáveis muitos dos modelos desenvolvidos.

Grande parte das implementações de redes neuronais foram feitas inteiramente por *software* que é executado em máquinas seriais de uso geral. Esta solução é absolutamente necessária para trabalhos experimentais, devido a flexibilidade que estes exigem. Entretanto, os computadores convencionais são extremamente lentos ao simularem redes neuronais de grandes dimensões, não sendo incomum certas simulações demandarem dias e até semanas. Nos últimos anos, foram realizadas várias implementações neuronais em *hardware*. Elas são caracterizadas por uma alta velocidade aliada a um baixo custo, mas com o incoveniente de uma flexibilidade muito limitada [16, 14], justificando a utilização de outros recursos não tão eficientes em uma fase de desenvolvimento.

Este trabalho mostra uma implementação de redes neuronais em uma arquitetura massivamente paralela. As técnicas usadas para mapeamento de modelos neuronais em arquiteturas paralelas podem ser divididas em duas categorias [20]: heurísticas e algorítmicas. Entre os mapeamentos heurísticos podemos citar os descritos em [5, 17, 3], que se caracterizam por uma abordagem empírica, baseada em particularidades do algoritmo e da arquitetura. Já os mapeamentos algorítmicos se baseiam em uma abordagem sistemática da implementação. Um exemplo pode ser encontrado em [18], onde a implementação de uma rede se baseia no grafo de dependência dados obtido a partir do modelo neuronal.

O método que será aqui apresentado tem como base a teoria dos grafos e é aplicável a modelos de redes neuronais cujas computações se baseiem em operações utilizando vetores e matrizes.

Este trabalho consiste de nove seções. Na segunda e terceira seções será descrito o algoritmo neuronal implementado. A seção seguinte fornece um descrição geral da arquitetura utilizada. A seção 5 traz a formulação do modelo neuronal com base na teoria dos grafos seguida da descrição do algoritmo neuronal adaptado à arquitetura SIMD na sexta seção. A seção sete especifica as medidas de desempenho utilizadas para comparar as simulações. Por fim, as duas últimas seções trazem os resultados, conclusões e perspectivas futuras.

2 Operações Básicas de Redes Neuronais

Uma rede neuronal pode ser definida por dois componentes primários: um conjunto de neurônios e um padrão de interconexão.

O neurônio é a unidade básica de processamento e é caracterizado pelo seu estado, por uma função de ativação e por uma função de saída. A função de ativação transforma os sinais de entrada, ponderados pelos seus pesos, e seu estado atual em um novo estado. A função de saída transforma o estado do neurônio em um sinal de saída.

Os neurônios podem ser classificados em três tipos: neurônios de entrada, neurônios intermediários e neurônios de saída. Neurônios de entrada recebem os sinais do mundo exterior; neurônios de saída enviam sinais àquele e os neurônios intermediários não são acessíveis externamente.

Nas redes neuronais do tipo *back-propagation*, os neurônios são normalmente agrupados em níveis. Dois níveis adjacentes são completamente conectados entre si, ou seja, cada um dos neurônios pertencentes a um destes níveis, possui obrigatoriamente uma conexão com cada neurônio do outro nível.

Os níveis por sua vez pode ser classificados de acordo com a função que desempenham na rede neuronal, sendo assim divididos em três grupos:

- Nível de Entrada: há apenas um em cada rede neuronal;
- Níveis Intermediários: pode haver um ou mais destes níveis em cada rede;
- Nível de Saída: há somente um em cada rede neuronal.

As redes neuronais deste modelo têm normalmente três níveis, pois o aumento do número de níveis, embora possível, não traz nenhuma melhora significativa [7] no que tange a função erro.

Durante a operação de uma rede neuronal pode-se distinguir duas fases: fase de produção e fase de aprendizagem. Na fase de produção é que se pode utilizar o conhecimento previamente adquirido pela rede. Para tanto, ela recebe entradas do mundo externo e produz sinais de saída para o mesmo. Já na fase de aprendizagem a rede "adquire" conhecimentos que são utilizados na fase de produção. O "conhecimento" é descrito no que se chama de padrão e consiste de um conjunto de entradas e as saídas esperadas quando da aplicação das primeiras. Ao conjunto de padrões conhecidos do problema a ser solucionado denomina-se de amostra de treinamento. A aprendizagem se faz por repetidas alterações dos pesos inerentes às conexões da rede com base na diferença entre as respostas geradas pela rede e as correspondentes do padrão.

Na verdade, a fase de aprendizagem se compõe de uma fase de produção seguida da fase de alteração de conexões. Assim, as entradas de cada padrão a ser treinado são inicialmente avaliadas e as saídas geradas pela rede são confrontadas com as do padrão para que as conexões sejam alteradas.

A regra de aprendizagem especifica o mecanismo pelo qual serão modificados os pesos das conexões. Uma rede neuronal pode aprender via alteração dos pesos das suas conexões. Em geral, a modificação do peso de uma conexão é uma função de quatro termos:

1. o estado do neurônio destino da conexão;
2. o valor de saída do neurônio origem da conexão;
3. o peso atual da conexão;
4. o par a ser treinado, ou seja, o valor esperado de saída do neurônio juntamente com a entrada correspondente.

Denomina-se ciclo à execução de uma fase de aprendizagem para cada um dos padrões da amostra.

3 Algoritmo

Nesta seção serão descritas a produção e a aprendizagem para redes neuronais *back-propagation*. Essas rotinas são a versão original do algoritmo descritas em [6, 11].

Inicialmente será definida a notação para identificar as diferentes partes da rede neuronal: os neurônios de saída são identificados por O_i, os intermediários por V_j e os de entrada por ξ_k. Cabe ressaltar que os algoritmos aqui apresentados são válidos para qualquer que seja o número de níveis intermediários. Neste trabalho serão consideradas redes com apenas um nível intermediário uma vez que isso facilita o entendimento não alterando a generalidade da exposição. As conexões entre o nível de entrada e o intermediário serão denotadas por w_{kj} e as existentes entre o nível intermediário e o de saída por W_{ji}. Deve-se notar que o índice k sempre se refere a um nodo de entrada, j a um nodo intermediário e i a um nodo de saída.

A função de ativação da saída(g) de cada neurônio é definida como:

$$g(x) = \frac{1}{1+e^{-2x}}$$

onde x é o valor de ativação do neurônio (a ser definido a seguir).

3.1 Produção

A cada padrão u a ser avaliado é associado um conjunto de entradas ξ^u.

Dado um padrão u, colocado como saída na camada de entrada, cada neurônio interno j recebe estas entradas ponderadas pelos respectivos pesos gerando o seu valor de ativação (h):

$$h_j^u = \sum_k w_{kj}\xi_k^u \tag{1}$$

e produz como saída:

$$V_j^u = g(\sum_k w_{kj}\xi_k^u) \tag{2}$$

Cada neurônio de saída i recebe:

$$h_i^u = \sum_j W_{ji} V_j^u \tag{3}$$

e gera a saída final:

$$O_i^u = g(\sum_j W_{ji} V_j^u) \tag{4}$$

3.2 Aprendizagem

Quando se efetua a aprendizagem, a cada padrão de entrada ξ^u temos um padrão de saída desejado ζ^u.

O objetivo primário da aprendizagem é minimizar a função erro, que pode ser definida como em função da diferença entre os valores esperados e calculados:

$$E = \frac{1}{2} \sum_u \sum_i [\zeta_i^u - O_i^u]^2$$

Será denotado $W_{ji}(t)$ o peso da conexão na iteração t e por $W_{ji}(t+1)$ o peso da conexão na iteração $t+1$, valendo o mesmo para w_{kj}. A atualização dos pesos das conexões é feita da seguinte forma:

Para conexões entre a camada intermediária e de saída:

$$W_{ji}(t+1) = \beta W_{ji}(t) + \Delta W_{ji}(t) \tag{5}$$

com

$$\Delta W_{ji} = \eta \sum_u \delta_i^u V_i^u \tag{6}$$

onde:

$$\delta_i^u = g'(h_i^u)[\zeta_i^u - O_i^u] \tag{7}$$

e para as conexões entre a camada de entrada e intermediária:

$$w_{kj}(t+1) = \beta w_{kj}(t) + \Delta w_{kj}(t) \tag{8}$$

$$\Delta w_{kj} = \eta \sum_u \delta_j^u \xi_k^u \tag{9}$$

onde:

$$\delta_j^u = g'(h_j^u) \sum_i W_{ji} \delta_i^u \tag{10}$$

Pode-se observar que há duas constantes inerentes ao treinamento:

β - Indica a importância do peso anterior para o cálculo do próximo (Fator de decaimento);

η - Indica o ganho a cada aprendizagem (a importância da alteração a ser feita na conexão).

Deve-se notar que para calcular (10) é necessário que se tenha calculado (7), de onde vem a denominação de *back-propagation* para o algoritmo.

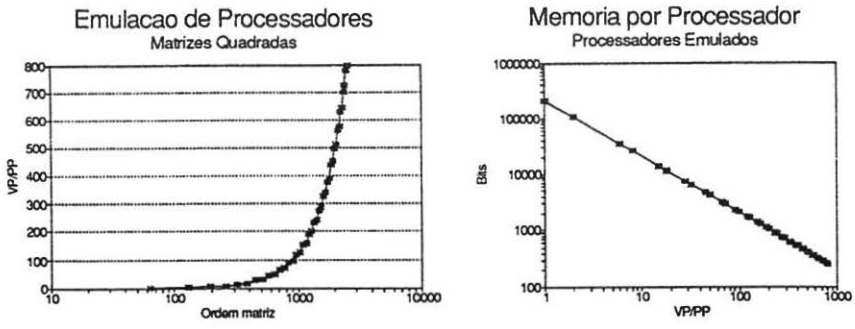

Figura 1: Emulação de Processadores

4 A Arquitetura Zephyr - Wavetracer

A Zephyr-Wavetracer[10] é uma máquina massivamente paralela formada por arranjos de processadores interconectados e ligada a uma estação de trabalho hospedeira via interface SCSI. Do ponto de vista da arquitetura, se enquadra no modelo SIMD [10]. Os seus processadores podem ser dispostos em arranjos de duas ou três dimensões de forma a atender às necessidades do problema a ser resolvido.

A utilização dos recursos que a máquina oferece se faz através da linguagem MultiC. Esta linguagem permite transparência para o usuário na utilização da máquina, combinando em um mesmo programa operações tradicionais da linguagem C [8] com aquelas providas pela máquina. Basicamente a Zephyr atua como um "co-processador" da máquina hospedeira, executando apenas as operações sobre os tipos privativos do Multi C.

A linguagem MultiC é uma extensão do padrão ANSI C contendo: aritmética de números complexos, operadores bit a bit e manipulação de estruturas de dados de até três dimensões, as quais são tratadas como uma única unidade. Cabe ressaltar outras características interessantes da linguagem como:

- Comunicação inter-processadores, ou seja, dados de um processador serem acessados por outro;
- Operações de redução, por exemplo, fazer o somatório de todos os elementos de um dado tipo do Multi C;
- O "formato" do arranjo de processadores é determinado automaticamente pelo programa em tempo de execução, permitindo que o mesmo programa execute em máquinas diferentes sem necessidade de recompilação.

A máquina existente no DCC-UFMG possui 4K processadores de 1 bit e memória de 128 Mb. Entretanto, o número de processadores que um programa pode usar não se limita a esta quantidade pois, sendo necessário, ela realiza automaticamente a emulação de processadores virtuais usando a sua memória interna. Assim, o tamanho das aplicações que utilizem a máquina é limitado pela memória disponível e não pelo número de processadores, como poder-se-ia supor.

Pode-se observar na Figura 1 o comportamento do recurso de emulação de processadores para uma aplicação que manipule matrizes quadradas. Assim, com o crescimento da ordem da matriz ocorre o aumento do número de processadores virtuais utilizados (*Virtual Processors per Physical Processors*). Em contrapartida, a memória que cada um desses pode utilizar diminui à medida que mais processadores são emulados.

Cabe ressaltar que, apesar das características expostas acima, esta arquitetura mostra uma redução drástica de desempenho em operações que envolvam indexação ou acesso de elementos isolados de suas estruturas de dados [12]. Esta restrição faz com que implementações como a descrita em [20] não possam ser eficientemente implementadas, obrigando a adoção de outras soluções mais adequadas à arquitetura.

5 O Modelo Neuronal

As operações de uma rede neuronal podem ser representadas por um grafo não direcionado, ao qual chamar-se-á grafo de configuração [4]. Um nodo em um grafo de configuração representa um agrupamento de

neurônios idênticos, que executam operações semelhantes nas fases de produção e aprendizagem. Uma aresta neste grafo representa o caminho de comunicação entre dois agrupamentos. No caso das redes neuronais *back-propagation*, um nodo do grafo de configuração corresponde a um nível (vide seção 2) e uma aresta às conexões entre dois níveis. A orientação de cada conexão, que representa o sentido de comunicação da ligação, depende da operação que a rede esteja executando em um dado momento.

Uma vantagem desta abordagem é permitir desconsiderar detalhes de operação da rede, devendo ser levados em conta apenas os agrupamentos (no caso níveis) e as relações entre eles.

Na fase de produção, as arestas são orientadas da entrada para a saída, representando a propagação das entradas pelos vários níveis da rede. Já a fase de aprendizagem pode ser representada por um grafo composto de dois subgrafos: o primeiro é o próprio grafo de produção visto acima, que representa a fase de avaliação de entradas, a qual precede toda alteração de pesos; já o segundo é obtido pela inversão dos sentidos das arestas do grafo de produção, representando a alteração de pesos via *back-propagation*. Esses dois subgrafos são conectados por uma aresta que vai do último nodo do primeiro subgrafo para o primeiro nodo do segundo subgrafo. Deve-se observar que os nodos do primeiro subgrafo se repetem no segundo, assim, por exemplo, o nodo relativo ao nível de saída tem uma aresta que aponta para ele mesmo, significando o início da fase de alteração de pesos. Esse grafo também representa as dependências de dados existentes em uma rede neuronal, ou seja, quando há uma aresta do nodo a para o nodo b tem-se que o nodo b utiliza as informações oriundas de a para o seu processamento.

Com base no grafo de configuração de uma rede *back-propagation* na fase de produção, pode-se afirmar que um nível só necessita das informações do nível que o precede imediatamente. As redes neuronais podem ser consideradas sistemas paralelos e distribuídos [6], onde cada neurônio depende somente das suas entradas para gerar a sua saída. Portanto, se um nível de neurônios já tiver calculado as saídas relativas às suas entradas atuais, ele pode computar novas entradas desde que seja mantida a integridade dos dados a serem propagados.

A exploração desta possibilidade certamente aumentaria sobremaneira o desempenho das implementações de redes neuronais, pois estas passariam a se comportar como um *pipeline* [9, 19], recebendo novas entradas a cada propagação efetuada e gerando as respectivas saídas num tempo igual ao do número de propagações necessárias para uma fase de produção. Nesse caso, o ganho vem do fato que uma produção pode ser iniciada sem que outra tenha terminado.

A grande dificuldade para que esta característica seja explorada em implementações paralelas é o sincronismo rigoroso que deve ser observado de forma a manter a integridade dos dados transmitidos entre níveis que estejam conectados [2, 21]. Entretanto, dadas as características da arquitetura SIMD aqui utilizada, esse sincronismo necessário é implícito, pois a mesma instrução é aplicada à massa de dados tratada simultanemente.

Já para o caso do treinamento, não é possível que várias fases sejam executadas simultaneamente, pois há dependências de dados mútuas entre os níveis, conforme pode ser visto no grafo de configuração correpondente. Isso pode ser explicado pela necessidade da manutenção do valor de saída de cada neurônio calculado na fase de produção para a fase de alteração de pesos (vide seção 3.2).

6 Implementação

Há na literatura, diversas propostas de implementação de algoritmos neuronais em máquinas paralelas [20, 5, 17, 18]. Algumas delas necessitam de manipular certas estruturas de dados de forma bastante particular, não podendo ser realizadas pela simples aplicação de um mesmo conjunto de operações fundamentais a cada um dos elementos da referida estrutura de dados e sendo, portanto, extremamente inadequadas à arquitetura em questão. (vide seção 4).

Conforme visto na seção 2, pode-se modelar todas as operações inerentes a redes neuronais como seqüências de manipulações de vetores e matrizes. Esta abordagem, embora tenha um alto custo em termos de alocação de memória, permite um alto nível de paralelismo em certas arquiteturas [1] como a Zephyr, o que mostra a viabilidade de uma implementação eficiente de simulações neuronais nessa máquina.

Para a realização de tal implementação foram utilizadas algumas operações especiais a serem realizadas sobre vetores ou matrizes, cuja definição será fornecida a seguir. Para efeito de coerência da notação, ao longo de todo o texto, os vetores serão denotados por letras minúsculas e as matrizes por letras maiúsculas.

Replicação Horizontal: (v^{*x}) fornece como resultado uma matriz onde cada uma das colunas é idêntica ao vetor operado;

Replicação Vertical: (v^{*y}) fornece como resultado uma matriz onde cada uma das linhas é idêntica ao vetor operado;

Expansão Horizontal: (v^{E_x}) fornece como resultado uma matriz onde a primeira coluna é igual ao vetor operado e o restante das posições é nula.

Expansão Vertical: (v^{E_y}) fornece como resultado uma matriz onde a primeira coluna é igual ao vetor operado e o restante das posições é nula.

Multiplicação Termo a Termo: ($M_1 \Theta M_2$) fornece como resultado uma matriz onde cada elemento é obtido pelo produto dos elementos presentes na posição correspondente à sua em cada uma das matrizes operadas;

Soma Termo a Termo: ($M_1 \Phi M_2$) fornece como resultado uma matriz (ou vetor) onde cada elemento é obtido pela soma dos elementos presentes na posição correspondente à sua em cada uma das matrizes operadas;

Subtração Termo a Termo: ($M_1 \Xi M_2$) fornece como resultado uma matriz onde cada elemento é obtido pela subtração dos elementos presentes na posição correspondente à sua em cada uma das matrizes operadas;

Condensação Horizontal: (M^{C_x}) fornece como resultado uma matriz onde o i-ésimo elemento da primeira coluna é igual ao somatório de todos os elementos pertencentes à i-ésima linha da matriz operada e as demais colunas são nulas;

Condensação Vertical: (M^{C_y}) fornece como resultado uma matriz onde o i-ésimo elemento da primeira linha é igual ao somatório de todos os elementos pertencentes à i-ésima coluna da matriz operada e as demais linhas são nulas;

Transposição: (M^T) fornece como resultado uma matriz onde cada elemento é igual ao elemento presente na posição simétrica à sua, em relação à diagonal principal, na matriz operada;

Redução Horizontal: (M^{R_x}) fornece como resultado um vetor que corresponde à primeira coluna da matriz operada;

Redução Vertical: (M^{R_y}) fornece como resultado um vetor que corresponde à primeira linha da matriz operada.

As oito primeiras operações são funções disponíveis na linguagem Multi C, que embora não ofereça diretamente as outras cinco, possui alguns recursos básicos a partir dos quais elas podem ser construídas sem qualquer prejuízo para o desempenho da implementação. Uma outra característica interessante do Multi C é que quando se realiza alguma operação aritmética cujos operandos sejam uma matriz e um escalar (por exemplo), este escalar é expandido automaticamente, ocupando todas as posições da matriz quando, então, é realizada a operação termo a termo.

6.1 Algoritmo de Produção

Conforme descrito na seção 3.1 e utilizando as operações e notação definidas acima, o algoritmo de produção pode ser reescrito da seguinte maneira:

- Inicialmente tem-se um vetor (v) cuja dimensão é o número total de neurônios da rede e que contém os valores de saída de cada um deles. Os pesos da rede neuronal são armazenados em uma matriz (w) de ordem do número total de neurônios da rede, de tal forma que o peso w_{ij} tem o seu valor definido na posição (i, j) (linha, coluna).
- Para cada grupo de conexões entre dois níveis, a geração dos valores de saída para os neurônios do nível superior compreende a seguinte seqüência de operações:
  1. O vetor v de valores de saídas a serem propagadas é replicado horizontalmente;
  2. A matriz de pesos w é multiplicada termo a termo pela obtida no passo anterior;
  3. A matriz resultante é condensada verticalmente;
  4. Cada um dos termos da primeira linha da matriz obtida é então operado pela função de ativação (g), definida na seção 3.1.
  5. Por fim, a matriz resultante é transposta de forma a conter os valores de saída necessários para a próxima propagação nas posições adequadas.

6. Se ainda houver novas propagações a serem realizadas, a matriz contendo as novas saídas pode ser simplesmente replicada e o algoritmo é executado a partir do segundo passo, caso contrário, a redução horizontal da matriz fornece o vetor de saídas v;

Em termos de operações especiais, cada propagação pode ser expressa por:

$$v = [[g([W\Theta v^{*_x}]^{C_y})]^T]^{R_x}$$

Para uma rede de três níveis, por exemplo, são necessárias duas propagações para que a rede produza a sua saída. Um aspecto interessante a ser notado é que a cada propagação são geradas saídas em todos os níveis, podendo ser válidas ou não.

6.2 Algoritmo de Aprendizagem

O algoritmo de aprendizagem descrito na seção 3.2 pode ser reescrito da seguinte forma:

- Além dos vetor de saída v e da matriz de pesos w descritas na seção 6.1, para o algoritmo de aprendizagem serão usados o vetor r que contém as saídas desejadas para as entradas contidas em v e uma matriz auxiliar p inicialmente nula que se destina a acumular os valores de δ (vide seção 3.2) para cada peso da rede;
- A primeira fase da aprendizagem consiste da geração das saídas relativas às entradas especificadas no vetor v, o que é efetuado por meio de uma produção, conforme descrito na seção 6.1.
- A seguir são produzidos os valores da matriz p que se referem aos neurônios do nível de saída, o que é feito seguindo os seguintes passos:
  1. É calculada a diferença entre os valores desejados e obtidos com a rede para cada neurônio de saída;
  2. Estes valores são multiplicados termo a termo pelo valor da derivada primeira da função de ativação de cada elemento de v que se refira a neurônio de saída, produzindo os valores de δ.
- p e v são replicados verticalmente e multiplicados pelo ganho de aprendizagem η elemento a elemento;
- A nova matriz de pesos é obtida pela soma termo a termo da matriz resultante do passo anterior com a matriz de pesos atual multiplicada termo a termo pelo escalar fator de inércia da aprendizagem(β).
- Para calcular os valores de p relativos aos neurônios intermediários, os valores de p replicados verticalmente são então multiplicados pelos respectivos novos pesos w, a matriz resultante é condensada horizontalmente e transposta, para ser usada na próxima retro-propagação do algoritmo;
- A matriz p é multiplicada termo a termo pelo valor da derivada primeira da função de ativação de cada elemento de v que se refira a neurônio intermediário, produzindo os valores de δ.
- A nova matriz de pesos é obtida pela soma termo a termo da matriz resultante do passo anterior com a matriz de pesos atual multiplicada termo a termo pelo escalar fator de inércia da aprendizagem(β).

O algoritmo de alteração de pesos pode ser expresso da seguinte forma usando as operações especiais definidas:

1. Os valores de p relativos aos neurônios de saída são dados por: (equação 7)

$$p = [(r^{E_x}\Xi v^{E_x})\Theta 2\Theta v^{E_x}\Theta(1\Xi v^{E_x})]$$

2. Já os valores de p relativos aos neurônios intermediários são calculados pela seguinte expressão: (equação 10)

$$p = [[W\Theta p^{*_y}]^{C_x}\Theta 2\Theta v^{E_x}\Theta(1\Xi v^{E_x})]$$

3. Para todos os casos a matriz de pesos é alterada da seguinte forma: (equações 8 e 5)

$$W = [W\Theta\beta\Phi\eta\Theta p^{*_y}\Theta v^{*_y}]$$

7 Avaliando o Desempenho de Simulações Neuronais

De forma a se comparar diferentes implementações, alguns padrões de medida devem ser providos. Medidas como velocidade de processamento aritmético, por exemplo, embora reflitam a essência dos algoritmos neuronais, podem ser bastante influenciadas por diferentes modelos neuronais e particularidades de cada problema.

Alguns dos indicadores mais comuns de velocidade [15, 14] são dados a seguir:

7.1 CPS (*Connections per Second*)

Cada ocasião que uma nova entrada é fornecida à rede neuronal, todas as suas conexões devem ser computadas. O número de conexões por segundo que uma rede executa é freqüentemente usado como uma medida de performance. Para que esse parâmetro possa ser usado para se avaliar o desempenho de redes é necessário considerar alguns fatores que influenciam nos resultados. Entre eles, o tamanho do problema em questão, a função de *thresholding* escolhida e a precisão com que se deseja obter os resultados. Quando se mede o desempenho em CPS, estamos avaliando a implementação da fase de produção da rede, conforme descrito na seção 2.

7.2 CUPS (*Connection Updates per Second*)

Quando se avalia o número de conexões por segundo que um sistema é capaz de processar, estamos medindo apenas a sua habiliadade em mapear entradas em saídas. Para indicar a performance do mesmo durante a fase de aprendizagem (vide seção 2), devemos medir a velocidade com a qual as conexões são atualizadas (CUPS). Em uma rede *back-propagation*, é possível obter facilmente esta medida, pois tanto a produção quanto a aprendizagem podem ser computados independentemente. Tipicamente o número de CUPS de um sistema é da ordem de 20% a 50% do CPS para a mesma rede.

7.3 SPR (*Synaptic Processing Rate*)

Esta é uma medida que indica o balanceamento entre o poder de processamento e o tamanho da rede, ou seja, o quão eficiente é um algoritmo para redes de tamanhos variados. O principal argumento para a sua adoção é que um neurônio biológico dispara sinais aproximadamente 100 vezes por segundo. Isso implica que cada uma das sinapses processa cerca de 100 sinais por segundo, que é o valor aproximado de SPR do cérebro humano. Tendo em vista ser o cérebro humano o sistema neuronal mais perfeito até hoje conhecido, este número pode ser um bom parâmetro para implementações neuronais. Algumas implementações paralelas têm SPRs ordens de magnitude maior do que 100, tendo, portanto, um alta taxa de processamento por conexão. Outros, baseados em computadores seqüenciais, têm um SPR próximo de 1, indicando um processamento sub-balanceado.

7.4 OPS (*Outputs per Second*)

Uma última medida de performance interessante é o número de saídas geradas pela rede neuronal por unidade de tempo. Essa medida reflete o paralelismo ou não do sistema na realização de produções, pois implementações que estejam habilitadas a realizar mais de uma propagação entre dois níveis simultaneamente têm um resultado melhor.

8 Resultados

Nesta seção vão ser apresentados os resultados obtidos na implementação do algoritmo descrito na seção 6. Todas as simulações foram executadas utilizando uma *SUN Sparc Station 2*. Os resultados seqüenciais para para todas as simulações são também mostrados para que se possa comparar os desempenhos.

As simulações realizadas avaliaram basicamente dois parâmetros:

Conectividade: é a quantidade de conexões que um neurônio do nível intermediário tem com os níveis de entrada e saída. Esta é uma medida de quão densa é a matriz de peso, pois quanto maior a conectividade, menor será o número de posições não usadas na matriz de pesos;

Número de Neurônios: é a quantidade de neurônios do nível intermediário. Esta medida expressa o número de neurônios que poderão ser avaliados simultaneamente em uma implementação paralela.

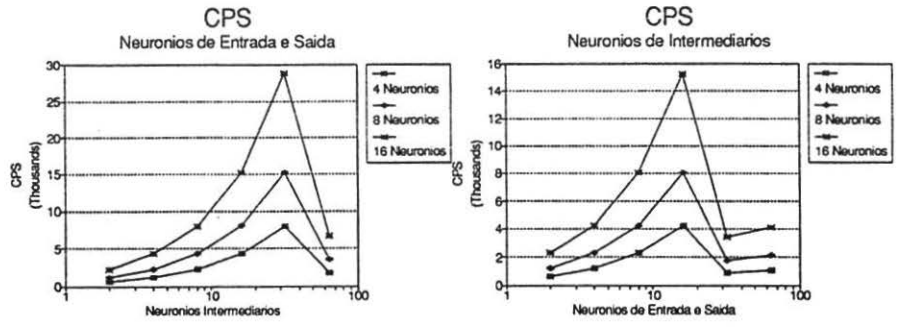

Figura 2: CPS:Implementação SIMD

Uma preocupação foi avaliar o desempenho do algoritmo para redes de tamanho variado. Por outro lado, a constância do número de neurônios das camadas de entrada e saída facilitava a avaliação da influência da variação da conectividade. Assim, os testes foram conduzidos ora variando-se o número de neurônios da camada intermediária, ora das camadas de entrada e saída.

Para cada uma das medidas descritas acima foram gerados dois gráficos, o primeiro variando o número de neurônios intermediários e mantendo, para cada curva, o número de neurônios de entrada e saída fixos e o segundo variando o número de neurônios de entrada e saída e mantendo fixa a quantidade de intermediários em cada curva. Em todas as figuras, o gráfico que corresponde à variação do número de neurônios intermediários foi colocado à esquerda e o que corresponde à variação do número de neurônios de entrada e saída foi colocado à direita.

8.1 Análise da Implementação Paralela

Serão apresentados a seguir os resultados obtidos com a implementação paralela do algoritmo. A principal finalidade dessas medidas é avaliar o desempenho dessa implementação ante a variações dos parâmetros descritos acima.

CPS: Pode-se observar pelo gráfico da Figura 2 que quanto maior é o número de neurônios, seja no nível de entrada, seja no nível intermediário, mais eficiente é a implementação paralela. Um aspecto interessante a ser observado neste gráfico e em alguns próximos é que, quando a máquina SIMD passa a emular processadores (como descrito na seção 4), o seu desempenho cai visivelmente. Mas, verificando o gráfico mais à direita, observamos que após a queda pelo início da emulação de processadores o número de CPS volta a crescer com o aumento do número de neurônios;

SPR: O gráfico da Figura 3, mostra que a variação número de neurônios de entrada e saída influencia mais a taxa de processamento sináptico do que o número de neurônios intermediários. Novamente pode-se observar que as redes maiores trazem resultados melhores e que o início do processo de emulação de processadores resulta em uma queda do desempenho.

OPS: Pelos gráficos da Figura 4, pode-se notar que a variação do número de neurônios intermediários não causa grandes alterações na taxa de produção de saída da rede neuronal. Já a variação do número de neurônios de entrada e saída gera mudanças significativas, similares às observadas pela análise dos gráficos relativos aos parâmetros CPS e SPR.

CUPS: Os gráficos de CUPS são vistos na Figura 5. O seu comportamento em relação às variações do número de neurônios são semelhantes ao do CPS. Um fato a se ressaltar é que o CUPS foi em torno de 50% do CPS, para cada caso, o que é o valor ótimo se considerarmos que um treinamento compreende duas "propagacões".

8.2 Seqüencial X Paralelo

Serão apresentados a seguir os gráficos onde são feitas comparações entre os resultados obtidos seqüencialmente e utilizando a arquitetura SIMD:

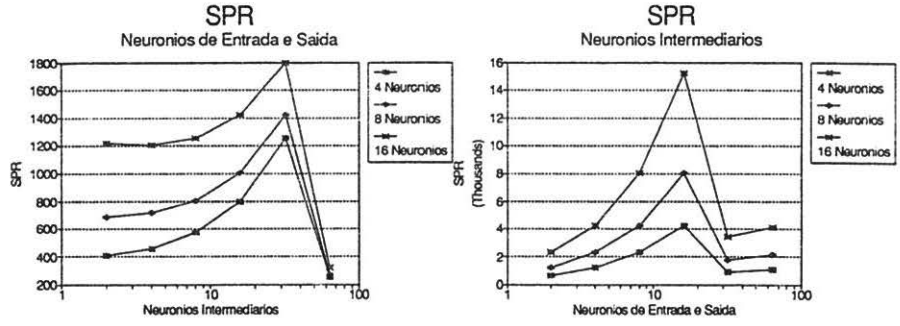

Figura 3: SPR:Implementação SIMD

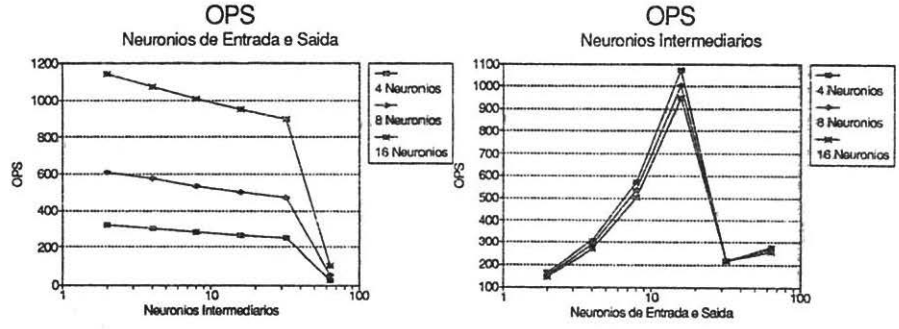

Figura 4: OPS:Implementação SIMD

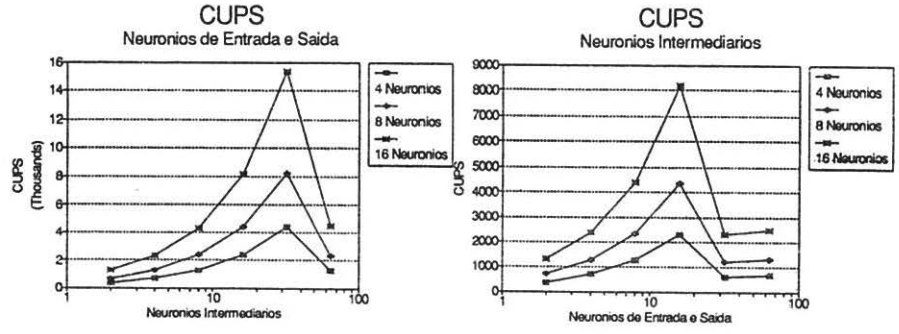

Figura 5: CUPS:Implementação SIMD

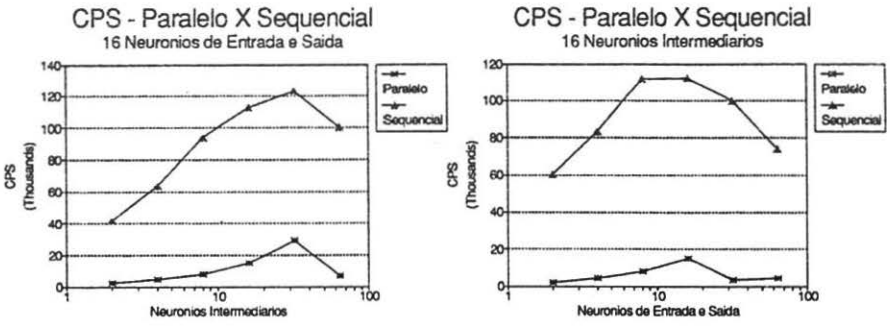

Figura 6: CPS:Paralelo X Sequencial

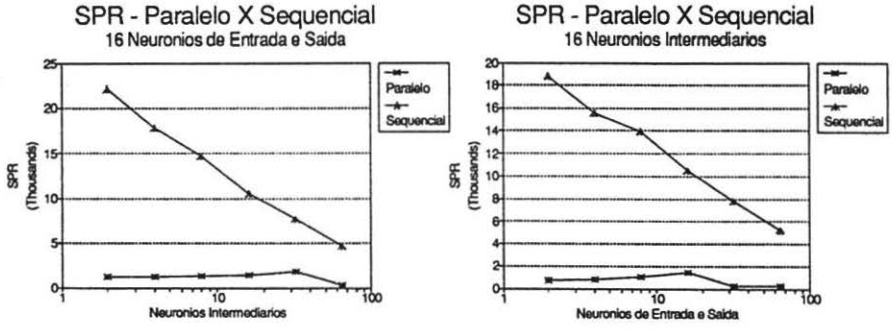

Figura 7: SPR:Paralelo X Sequencial

CPS: Observando os resultados da Figura 6, percebe-se que, para redes pequenas a versão seqüencial é claramente melhor que a paralela. Entretanto, quando a soma do número de neurônios nos níveis de entrada e saída se torna maior que o número de neurônios intermediários, o desempenho do algoritmo seqüencial começa a decrescer, enquanto o do paralelo se mantém estável.

SPR: Os gráficos da Figura 7 mostram claramente que esse parâmetro se comporta de forma semelhante ao CPS, ou seja, quanto maior a rede, melhor o resultado indicado por ele para a implementação paralela.

OPS: No que tange ao número de saídas geradas pela rede numa unidade de tempo (Figura 8), pode-se observar que a variação do número de neurônios intermediários não afeta o desempenho da implementação paralela, enquanto a seqüencial piora visivelmente. Já no caso da variação do número de neurônios de entrada e saída, percebe-se que o desempenho da implementação seqüencial piora a partir do momento em que o número de neurônios intermediários passa a ser menor que a soma do número de neurônios dos níveis de entrada e saída, à semelhança do observado com CPS.

CUPS: Comparando a performance das implementações na atualização de pesos na Figura 9, pode-se verificar que, quando o número de neurônios é aumentado, a implementação paralela melhora o seu resultado, enquanto a seqüencial apresenta um decréscimo em seu desempenho.

9 Conclusões

Este trabalho apresentou uma proposta de implementação de redes neuronais em máquinas SIMD. A implementação se baseia no fato de que o funcionamento da rede neuronal a ser modelada pode ser visto como uma série de operações sobre vetores e matrizes e ser modelado como um grafo de dependência de dados.

Entretanto, essa abordagem ocasiona um sub-aproveitamento das potencialidades da máquina SIMD utilizada. Para redes muito grandes, matrizes de um tamanho considerável são alocadas e apenas uma

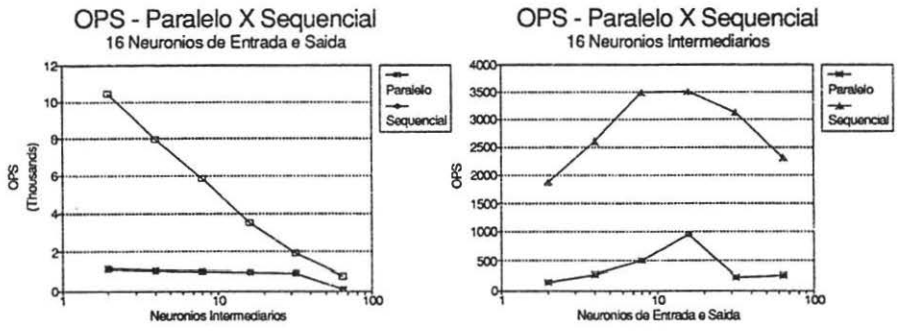

Figura 8: OPS:Paralelo X Sequencial

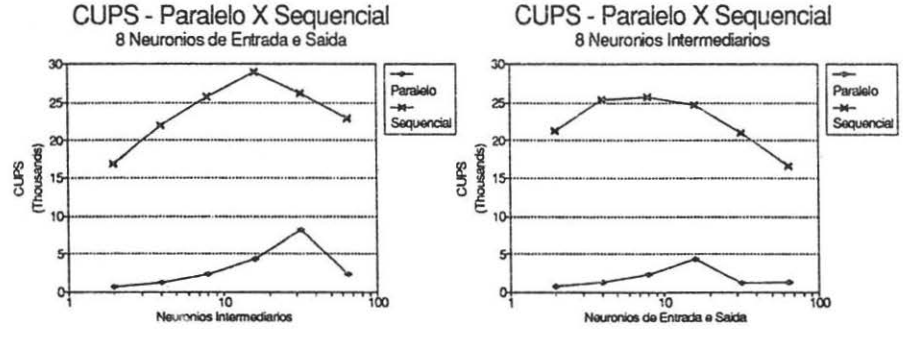

Figura 9: CUPS:Paralelo X Sequencial

pequena porção é realmente utilizada (em torno de 15% para os casos analisados). Uma forma de aproveitar as reais capacidades da máquina, seria através da reestruturação do algoritmo de forma que as matrizes envolvidas fossem mais densas. Um outro fator a ser considerado em implementações futuras é a queda de performance provocada pela emulação de processadores por parte da máquina. Apesar de ser um recurso que aumenta sua capacidade é necessário avaliar seu custo.

Embora não se tenha conseguido ganhos reais em relação à implementação seqüencial, a implementação paralela se mostrou bastante adequada para o problema quando este abrange redes com milhares de neurônios, pois esta última apresentou melhora de resultados quando do aumento do número de neurônios. Simulações com redes maiores não puderam ser realizadas, em virtude da limitação de memória das máquinas utilizadas.

Referências

[1] S. G. Akl. *The Design and Analysis of Parallel Algorithms.* Prentice-Hall, 1989.

[2] V. C. Barbosa and P. M. V. Lima. On the distribuited parallel simulation of hopfield's neural networks. *Software - Pratice and Experience*, 20(10):967 - 983, October 1990.

[3] G. Belloch and C. Rosenberg. Network learning on the connection machine. In *Proc. 10th Int. Joint Conf. Artif. Intell.*, Milan, Italy, 1987.

[4] L. C. Chu and B. W. Wah. Optimal mapping of neural-network learning on message-passing multicomputers. *Journal of Parallel and Distibuited Computing*, (14):319 - 339, 1992.

[5] D. S. Touretzky D. A. Pomerlau, G. L. Gusciora and H. T. Kung. Neural network simulation at warp speed: How we got 17 million connections per second. In *Proc. Int. Conf. Neural Networks*, San Diego, CA, June 1988.

[6] G. E. Hinton D. E. Rumelhart and R. J. Williams. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. MIT Press, Cambridge, MA, 1986.

[7] Márcio L. B. de Carvalho and Wagner Meira Jr. Um modelo de redes neurais para controle de sistemas robóticos. In *Anais do III Simpósio Brasileiro de Arquitetura de Computadores e Processamento Paralelo*, pages 22-37, Rio de Janeiro, Novembro 1990. Sociedade Brasileira de Computação.

[8] B. W. Kernighan e D. M. Ritchie. *C, a linguagem de programação: padrão ANSI.* Editora Campus, Rio de Janeiro, 1990.

[9] J. P. Hayes. *Computer Architecture and Organization.* McGraw-Hill, 2nd ed. edition, 1988.

[10] Wavetracer Inc. *The MultiC Programming Language.* 289 Great Road, Acton, Massachusetts, USA, September 1991.

[11] R. G. Palmer J. Hertz, A. Krogh. *Introduction to the Theory of Neural Computation*, volume 1 of *Computation and neural systems series.* Allan M. Wylde, 1991.

[12] W. Meira Jr. M. L. B. de Carvalho J. P. Salles, M. A. G. Vieira. Zephyr wavetracer: uma análise de suas operações elementares. Technical Report RT008/92, DCC - ICEx - UFMG, 1992.

[13] M. James and D. Hoang. Design of low-cost, real-time simulation systems for large neural networks. *Journal of Parallel and Distributed Computing*, (14):221 - 235, 1992.

[14] W. Baggett W. S. Boyd Jr. M. H. Garzon, S. P. Franklin and D. Dickerson. Design and testing of a general-purpouse neurocomputer. *Journal of Paralel and Distributed Computing*, (14):203 - 220, 1992.

[15] T. Nordström and B. Svensson. Using and designing massively parallel computers for artificial neural networks. *Journal of Parallel and Distributed Computing*, (14):260 - 285, 1992.

[16] L. M. Reyneri. An analysis on the performance of silicon implementations of backpropagation algorithms for artificial neural networks. *IEEE Transactions on Computers*, 40(12):1380 - 1389, December 1991.

[17] U. Schweigelsohn. A shortperiodic two-dimensional systolic sorting algorithm. In *Proc. Int. Conf. Systolic Arrays*, pages 257 - 264, 1988.

[18] S. Shams. In *Proc. IEEE int. Symp. Circuits Syst.*, New Orleans, LA, May.

[19] H. S. Stone. *High-Performance Computer Architecture*. Addison-Wesley Publishing Co., 2nd ed. edition, 1990.

[20] V. K. Prasanna W. M. Lin and K. W. Przytula. Algorithmic mapping of neural network models onto parallel simd machines. *IEEE Transactions on Computers*, 40(12):1390 – 1401, December 1991.

[21] M. L. B. Carvalho W. Meira Jr. Sirnem: A parallel neural network simulation system. In *XVIII Conferencia Latinoamericana de Informatica*. CLEI, Agosto 1992.