Simulação de Redes Neuronais em Ambientes Heterogêneos e Paralelos

Wagner Meira Junior * Márcio Luiz Bunte de Carvalho†

Departamento de Ciência da Computação
Instituto de Ciências Exatas
Universidade Federal de Minas Gerais
Caixa Postal 702 – Belo Horizonte – MG – CEP 30.161-970

Resumo

Um dos maiores problemas na simulação de redes neuronais é o custo computacional associado à execução desses algoritmos. A paralelização dessas simulações tem sido uma solução bastante adotada para contornar esse problema, muito em virtude do inerente paralelismo das redes neuronais. Arquiteturas paralelas do tipo SIMD e MIMD são freqüentemente utilizadas, de forma isolada, para essa finalidade. Esse trabalho investiga a adequação e as restrições de utilização de heterogeneidade e paralelismo conjugando arquiteturas SIMD e MIMD para a simulação de redes neuronais. Essa estratégia se mostrou boa, entretanto sua eficiência é extremamente dependente das dimensões da rede a ser simulada e do ambiente computacional utilizado.

Abstract

One of the greatest problems simulating neural networks is its high computational cost. The parallelization of these simulations has frequently been used to minimize this problem, mainly because the neural networks intrinsics parallelism. To achieve this goal, parallel architectures such SIMD and MIMD are used isolated. This work studies the suitability and the restrictions of utilization of heterogeneity based upon SIMD and MIMD architectures to implement neural networks. As it will be presented in this work, this strategy presents a good performance in spite of being dependent of the dimension of the simulated network and the architecture used.

Este trabalho foi desenvolvido com o apoio do CNPq e da FAPEMIG (Tec 1130/90)

*Mestrando em Ciência da Computação - UFMG; Redes neuronais, processamento paralelo e distribuído; E-mail: meira@dcc.ufmg.br

†Doutorando em Pesquisa Operacional (UC Berkeley); Algoritmos paralelos, programação matemática, algebra linear numérica, redes neuronais; Professor do Departamento de Ciência da Computação – UFMG; E-mail: mlbc@dcc.ufmg.br

1 Introdução

Redes neuronais artificiais (RNs) são modelos matemáticos baseados no que se conhece do funcionamento do cérebro e da mente. RNs são também conhecidas por sistemas conexionistas, sistemas adaptativos, neurocomputadores e processadores paralelos e distribuídos. Elas exploram o processamento local massivamente paralelo e as propriedades de representação distribuída que acredita-se haver no cérebro. O seu primeiro objetivo é investigar e reproduzir tarefas humanas de processamento de informação como fala, visão, olfato, tato, processamento do conhecimento e controle motor. Além dessas, as RNs também são utilizadas para compressão de dados, resolução de problemas de otimização combinatória, classificação de padrões, modelagem de sistemas e aproximação de funções [Simpson, 1990].

Uma definição geral, embora não rigorosa, desses modelos pode ser encontrada em [Hecht-Nielsen, 1988]:

> *Uma rede neuronal é uma estrutura de processamento de informação paralela e distribuída que consiste de elementos de processamento (que podem possuir memória local e efetuar processamento de informações locais) interconectados via canais unidirecionais chamados conexões. Cada elemento de processamento tem uma única conexão de saída que pode conduzir um mesmo sinal a tantos outros elementos de processamento quanto se deseje. Essa saída pode ser de qualquer tipo matemático desejado. Toda a computação que ocorre em cada elemento de processamento deve ser completamente local; dependendo apenas dos valores de entrada correntes e dos valores armazenados na memória local do elemento.*

Uma grande variedade de simulações de redes neuronais têm sido desenvolvidas no intuito de aproveitar computacionalmente o paralelismo inerente a esse tipo de modelo. As características principais de uma simulação de RNs são:

Computacionalmente intensiva: a saída de cada elemento de processamento de uma RN é normalmente uma soma de vários produtos; assim, cada conexão da rede requer uma multiplicação seguida de uma soma para computar o valor do sinal a ser transmitido.

Massivamente paralela: as RNs tipicamente tem processamento local completo; desta forma, cada elemento de processamento (o qual será chamado de nodo) pode ser tratado como um processador individual operando em paralelo. Isso significa que, assumindo que cada nodo tenha um processador dedicado, a velocidade de processamento da RN depende apenas da conectividade de cada nodo e não do número deles. Por exemplo, desde que cada nodo tenha o mesmo número de conexões, uma rede com 10 nodos pode operar na mesma velocidade que uma com 10.000 nodos.

Grande memória requerida: as RNs têm geralmente muitas conexões. Cada conexão tem um peso associado que deve ser armazenado. Isso cria um sério problema à medida que o tamanho da rede cresce. Por exemplo, uma rede totalmente conectada de dois níveis com 10 nodos em cada nível tem 100 conexões. Dobrando-se o número de nodos, tem-se 400 conexões. Tendo em vista que as redes neuronais utilizáveis na prática têm grandes dimensões, a memória ocupada por elas pode representar uma séria restrição à sua simulação.

Sem dúvida, a quantidade de dados a serem tratados são o componente principal do custo computacional desses modelos. Entretanto, a possibilidade de efetuar-se várias computações em paralelo mostrou uma maior adequação dessas simulações aos paradigmas do processamento paralelo e distribuído. Nesse intuito, muitos trabalhos já foram desenvolvidos (vide Seção2) explorando a relação modelo neuronal x arquitetura não convencional ou paralela. Essa exploração introduz um nível a mais de complexidade na simulação, representado pela busca do ferramental adequado. A não observância disso pode trazer conseqüências desastrosas como, por exemplo, o fenômeno conhecido como *speed-down*, onde a simulação paralela é menos eficiente que a seqüencial.

Muitas vezes, uma boa solução é combinar diferentes paradigmas de forma que partes diferentes do problema sejam abordadas de forma diversa. A utilização de componentes de processamento com diversidade funcional é denominada computação heterogênea e tem sido muito difundida. Assim, o presente trabalho traz uma proposta de simulação de redes neuronais utilizando heterogeneidade. Esse trabalho está dividido em seis seções: a primeira, essa introdução; na seção 2 será feito um levantamento seguido de avaliação

das diversas simulações paralelas existentes na literatura; na terceira seção, será apresentado o ambiente computacional utilizado; a seção 4 descreve a funcionalidade e utilização do simulador HENNS; por fim, as duas últimas seções descrevem os resultados e conclusões do trabalho respectivamente.

2 Simulações Paralelas

O inerente paralelismo das redes neuronais é apenas o ponto de partida para a exploração de arquiteturas que se aproximem da neuronal como a SIMD e a MIMD. Entretanto, cabe lembrar que essas são classes de arquiteturas de computadores com diferentes particularidades e potencialidades. Na verdade, tem sido uma constante por parte dos pesquisadores a busca de arquiteturas mais adequadas e eficientes para a simulação de redes neuronais. Parte desse esforço vai ser descrito a seguir, onde se pode perceber essa diversidade e vários dos nuances desse tipo de investigação.

2.1 Simulações em Máquinas MIMD

Dada a própria semelhança estrutural, as simulações de redes neuronais em máquinas MIMD são as mais populares. Esses trabalhos enfocam, além da arquitetura alvo, outros aspectos como mapeamento e sincronismo, além de simulações dedicadas a certos modelos. Essas diferentes abordagens e características serão explicitadas nas seções seguites:

2.1.1 Arquiteturas Utilizadas

Os *transputers* foram, sem dúvida, a arquitetura mais utilizada [Forrest et al., 1987, Fortuna et al., 1990, Meira Jr. and Carvalho, 1992, Tollenaere and Orban, 1991, Barbosa and Lima, 1990, Perez, 1990]. Isso se deve a basicamente dois fatores:

1. Baixo custo por unidade de processamento;
2. Comunicação eficiente entre processadores.

Como desvantagem pode-se apontar o baixo desempenho se comparado com máquinas da mesma "categoria" [Nordström and Svensson, 1992]. Assim, outras arquiteturas como o Hypercube Intel iPSC 2 [Chu and Wah, 1992], o SBC [Heileman et al., 1992] e BBN Butterfly [Feldman et al., 1988] também foram utilizadas obtendo bons resultados.

2.1.2 Assinalamento de elementos de processamento a processadores

Um problema que surge quando da simulação de RNs em máquinas paralelas é a escolha de quais tarefas serão executadas por quais processadores. Isso é importante para que haja um balanceamento de carga. Se não acontece, tem-se uma simulação mais lenta e com ociosidade por parte de alguns e sobrecarga por parte de outros processadores.

Há várias propostas, que consideram basicamente dois aspectos:

1. Processamento a ser efetuado localmente;
2. Dados a serem transmitidos de um computador para outro.

O primeiro aspecto é mais fácil de ser resolvido pois o que está em questão é o montante de processamento local, pode ser previsto e devidamente balanceado. Já no segundo caso, a topologia tem importância fundamental, pois é ela que vai reger o custo de comunicação. Mais importante do que isso, a prática tem mostrado que um desbalanceamento de carga de processamento local causa menos danos que uma comunicação densa entre processadores.

Seguindo essa idéia, Fortuna [Fortuna et al., 1990] propõe um esquema de particionamento. Essa proposta é adequada a redes que tenham um padrão de conexão regular mas não sejam totalmente conectadas. Quando da utilização de redes com essa característica, o mapeamento se mostrou eficiente.

Já Feldman [Feldman et al., 1988] não fez qualquer esquema de particionamento, apesar de relatar que isso causa sérios desbalancementos de carga na máquina alvo (BBN Butterfly). Essa estratégia contrasta com [Chu and Wah, 1992] que trata o problema do assinalamento como NP-difícil, resolvendo-o com uma heurística de sua autoria. Esse trabalho é direcionado para mapeamentos estáticos (constantes durante o processamento) e dinâmicos (cujas arquiteturas permitem relocação de processos). Os mapeamentos estáticos foram testados com bons resultados no iPSC 2, enquanto os dinâmicos foram avaliados via simulação, também com bons resultados.

Uma abordagem semelhante é adotada por [Tollenaere and Orban, 1991], que, de posse do grafo de interconexões da rede, faz uma decomposição geométrica em tantas partes quanto o número de processadores, minimizando o número de conexões entre as partes. A importância do mapeamento também é ressaltada em [Ghosh and Hwang, 1989], onde são tratadas redes com padrão de interconexão irregular. Novamente priorizou-se a diminuição da comunicação como fator de aceleração da simulação.

2.1.3 Sincronismo

A simulação de redes neuronais pode ser enquadrada como uma simulação orientada a eventos. Assim, devido à topologia de certas redes como a de Hopfield, onde há conexões bidirecionais que devem conduzir a uma ativação simultânea de nodos; a simulação pode estar sujeita a situações como *deadlock* e inconsistência do funcionamento da rede em relação ao modelo original.

Esse problema foi tratado em [Barbosa and Lima, 1990]. Nesse trabalho foi proposto um protocolo que evita essas situações ditas "perigosas" pela utilização da seguinte política: num dado instante da simulação, o sentido das comunicações de toda a rede deve formar um grafo acíclico.

Esse problema foi generalizado para qualquer tipo de topologia e conexões [Meira Jr. and Carvalho, 1992] e sua solução foi utilizada na implementação do HENNS(vide Seção 4).

2.1.4 Modelo Neuronal Abordado

Sem dúvida o modelo neuronal mais utilizado é o *back-propagation* [D. E. Rumelhart and Williams, 1986]. Há também algumas simulações de Hopfield e ART [Zurada, 1992]. Deve-se também ressaltar que muitos trabalhos não se destinam a um modelo específico de rede neuronal, mas a toda uma classe de arquiteturas conexionistas, como por exemplo [Meira Jr. and Carvalho, 1992, Nordström and Svensson, 1992, Ghosh and Hwang, 1989, Tollenaere and Orban, 1991], entre outros.

2.1.5 Análise das Simulações MIMD

Verificando os trabalhos supra citados, pode-se concluir que o custo da comunicação é o fator decisivo quando se usa uma rede de processadores:

1. Com redes neuronais total ou randomicamente conectadas, a topologia formada pelos processadores pode influenciar no sentido de uma pequena redução no custo de comunicação;
2. Com redes neuronais conectadas randomicamente, a limitação do número de conexões inerentes a um dado nodo não provoca um decréscimo significativo do custo de comunicação;
3. Uma regularidade global nos padrões de interconexão, como as encontradas em redes modulares [Murre, 1993], pode ser usada para implementar esquemas mais eficientes de processamento.

2.2 Simulações em Máquinas SIMD

As simulações em máquinas SIMD são menos freqüentes que as em arquitetura MIMD. Isso se deve a dois fatores básicos:

1. O mapeamento de redes neuronais em arquiteturas SIMD não é tão trivial quanto o MIMD, pois, embora o processamento síncrono dos elementos de processamento as torne semelhantes às RNs, a implementação da comunicação entre nodos pode ser uma tarefa muito difícil;
2. O custo desse maquinário é em geral maior que de máquinas MIMD, se considerarmos, por exemplo, os transputers.

A seguir será feita uma descrição das simulações desse gênero encontradas na literatura levando em conta a arquitetura de simulação e a forma de mapeamento.

2.2.1 Arquiteturas Utilizadas

A eficiência da simulação em máquinas SIMD é, até certo ponto, condicionada pelas facilidades arquiteturais que a máquina oferece, principalmente no que tange à comunicação entre processadores. Topologias pouco regulares podem ser de difícil simulação [W. M. Lin and Przytula, 1991] e a conectividade dos nodos pode ser fator determinante da velocidade de simulação [Salles et al., 1992].

Várias máquinas SIMD já foram utilizadas, entre elas a Zephyr-Wavetracer [Salles et al., 1992], DAP [Forrest et al., 1987, Nordström and Svensson, 1992], CM-2 [Nordström and Svensson, 1992] e Hughes SCAP [W. M. Lin and Przytula, 1991].

Por fim, Nordstron [Nordström and Svensson, 1992] faz um apanhado geral das máquinas usadas em simulações paralelas em arquiteturas SIMD, sendo uma boa referência.

2.2.2 Mapeamento do modelo na arquitetura

Conforme já foi dito, o mapeamento de redes neuronais em arquiteturas SIMD é muito dependente das máquinas alvo. As redes *back-propagation* foram mapeadas para a arquitetura Zephyr-Wavetracer em [Salles et al., 1992]. Já [W. M. Lin and Przytula, 1991] propõe um esquema geral de mapeamento de redes com topologia arbitrária e, embora o *back-propagation* tenha sido o algoritmo usado para a metodologia, ela se aplica a uma grande gama de modelos que se baseiam em operações com vetores e matrizes.

2.3 Outras Propostas

Após uma ampla verificação de todos os modelos e arquiteturas para a simulação de redes neuronais, Nordstrom [Nordström and Svensson, 1992] conclui que a arquitetura MIMSIMD (*Multiple Instruction streams for Multiple SIMD arrays*) é a mais adequada para a simulação de redes neuronais.

Essa afirmação se baseia no fato de que o grande desafio das simulações de redes neuronais num futuro proximo são as simulações de sistemas neuronais artificiais, ou seja, sistemas compostos de um grande número de módulos de redes neuronais cooperantes. Cada módulo deveria simular uma estrutura diferente de rede e os módulos devem estar habilitados a interagir de formas diferentes a uma alta velocidade. Isso implica que sistemas heterogêneos compostos de arranjos de processadores homogêneos devem ser desenvolvidos, e especial atenção deve ser dada ao problema da interação entre módulos e entre módulos periféricos e o ambiente onde está se processando a simulação.

2.4 Análise das Simulações Paralelas

As arquiteturas paralelas tradicionais não se mostram muito adequadas para uma simulação de modelos variados de redes neuronais, assim:

Arquiteturas MIMD: têm como maior problema o custo de comunicação, mas outras questões como sincronismo e mapeamento de nodos a processadores podem também ter de grande influência;

Arquiteturas SIMD: a adequação da grande diversidade de algoritmos de redes às várias arquiteturas é a grande barreira, não se conseguiu um mapeamento transparente da conectividade desses modelos de uma forma eficiente.

A análise desses trabalhos de simulação paralela e a proposta de [Nordström and Svensson, 1992] endossam a necessidade de investigações sobre a aplicabilidade de arquiteturas heterogêneas às simulações de redes neuronais, objeto desse trabalho.

3 Ambiente Computacional

Para verificar a adequação da simulação de RNs a ambientes de processamento distribuído e heterogêneo utilizou-se dois conjuntos de recursos do DCC- UFMG: (i) a rede de *workstations Sun* em conjunto com o pacote PVM que permite usá-la como uma máquina MIMD; (ii) a máquina SIMD Zephyr-Wavetracer. Esses recursos são descritos a seguir.

3.1 Arquitetura PVM

O PVM (*Parallel Virtual Machine*) [Geist et al., 1993] é um pacote de programas e bibliotecas que permite que uma rede de computadores heterogêneos UNIX seja usada como um único e grande computador paralelo. Assim, grandes problemas computacionais podem ser resolvidos usando o poder agregado de vários computadores.

Sob o PVM, o usuário define uma coleção de computadores seriais, paralelos e vetoriais que se comportarão como um único computador paralelo com memória distribuída. Desta forma, denomina-se máquina virtual esses computadores lógicos com memória distribuída e hospedeiro cada um dos computadores reais. O PVM provê funções para disparar tarefas automaticamente nas máquinas virtuais e permitir que as tarefas se comuniquem e sincronizem entre si. Uma tarefa é definida como sendo a unidade de computação do PVM de forma análoga a processos UNIX. Freqüentemente ela é um processo UNIX, mas não necessariamente. Aplicações, que podem ser escritas em C ou FORTRAN 77, podem ser paralelizadas usando construções para trocas de mensagens comuns a muitos computadores de memória distribuída. Um bom exemplo de utilização do PVM é na implementação de aplicações do tipo "cliente – servidor".

PVM suporta heterogeneidade nos níveis de aplicação, de máquina e de rede. Em outras palavras, ele permite que tarefas de aplicação explorem a arquitetura mais adequada à solução do problema enfocado. Todas as conversões de dados necessárias são automaticamente efetuadas para viabilizar a comunicação entre computadores que usem diferentes representações de inteiros e reais.

O sistema PVM é composto de duas partes. A primeira parte é um *daemon*, chamado *pvmd3* que reside em todos os computadores formando a máquina virtual. (Um exemplo de um *daemon* é o programa *sendmail* que manipula todas as mensagens eletrônicas - *mails* que chegam e saem em um sistema UNIX.) *pvmd3* foi projetado de tal forma que qualquer usuário com um *login* válido possa instalá-lo em uma máquina. Ao rodar uma aplicação PVM, ele executa *pvmd3* em um dos computadores, disparando o *daemon* em cada um dos outros computadores e criando a máquina virtual definida pelo usuário. A aplicação PVM pode então ser disparada de qualquer dessas máquinas como uma tarefa UNIX normal. Usuários podem configurar máquinas virtuais com sobreposição, e cada um pode executar várias aplicações PVM simultaneamente.

A segunda parte do sistema é uma biblioteca de rotinas. Essa biblioteca contem rotinas que podem ser chamadas pelo usuário para troca de mensagens, disparo de processos, coordenação de tarefas e modificação da máquina virtual. Os programas de aplicação devem incluir essa biblioteca para usar o PVM.

3.2 A Zephyr-Wavetracer

3.2.1 Arquitetura

A Zephyr-Wavetracer (WT) é uma máquina SIMD (Single Instruction Multiple Data) com 8192 processadores de 1 bit cada. Esses 8K processadores são dispostos em duas placas, cada uma com 4K processadores. Essa máquina é ligada a uma *workstation* hospedeira por uma interface SCSI.

A memória total da máquina (da ordem de 256 megabytes) é igualmente dividida entre os processadores. Cada processador possui dois tipos de memória, a primeira é da ordem de 2 Kbits e é interna ao processador, já a segunda é externa a cada processador possui 32 Kbytes.

Os processadores podem ser organizados bi ou tridimensionalmente via *software*, no caso, para uma máquina com 8k processadores, o arranjo bidimensional tem dimensões 64 × 128 e o tridimensional 16 × 32 × 16.

Às dimensões do arranjo de processadores denomina-se espaço de solução, assim, num arranjo de 64 × 128 pode-se manipular simultânea e identicamente 8192 elementos.

3.2.2 Processamento Virtual

Para suportar um espaço de soluções com dimensões maiores que as dos arranjos de processadores, o controlador particiona a memória de cada processador em n partes iguais e associa cada uma delas a um nodo no espaço de soluções. Assim, sempre que uma instrução for gerada para o arranjo de processadores, o controlador instrui a sua execução para cada uma das partições via alterações de endereçamento, ou seja, o controlador trata cada processador real como n processadores virtuais.

3.2.3 Linguagem MultiC

Um programa de aplicação para a WT é escrito como um programa para a *workstation* hospedeira em MultiC, uma extensão da linguagem Ansi C. Tendo em vista que o programa executa na *workstation* e não no controlador da WT, o programa de aplicação compilado tem acesso às facilidades da *workstation*, como sistema de arquivos, rede e ambientes gráficos. As dimensões do arranjo de processamento virtual são especificadas em tempo de execução, sem o prévio conhecimento da configuração do arranjo físico. Como resultado, um programa pode rodar em máquinas com quaisquer quantidades de processadores sem necessidade de recompilação.

As principais extensões ao Ansi C providas pelo MultiC são as seguintes:

1. O especificador de tipos multi, que declara uma variável que tem um valor numérico independente em cada processador virtual. O especificador de tipo uni também foi acrescentado à linguagem com a finalidade de declarar variáveis que tenham apenas um valor que é armazenado na *workstation*.
2. Um operador de comunicação interprocessador que permite o trânsito e intercâmbio de dados entre processadores. Deve ser ressaltado que o fluxo de dados é idêntico para todo o espaço de soluções virtual.
3. Operadores de reduçã que aplicam a operação indicada a todos os valores da expressão multi alvo nos processadores virtuais ativos, condensando o resultado em um único tipo uni. As operações de redução são: soma, subtração, multiplicação, divisão, operações lógicas (e, ou , ou-exclusivo) e operadores de máximo e mínimo.

4 O simulador HENNS

HENNS[Meira Jr, 1993], desenvolvido no DCC - UFMG, é um simulador de redes neuronais para ambientes paralelos e heterogêneos. Nesse ambiente o usuário pode definir topologias e modelos de redes neuronais

arbitrários. Uma vez definido e especificado o modelo, a simulação pode ser feita de variadas formas: seqüencial, distribuída e vetorizada. Na forma seqüencial, todos os componentes da rede são simulados em um único programa executável. Por outro lado, nas formas distribuída e vetorizada, a rede corresponde a um processo PVM e cada conjunto que deva ser simulado em separado tem um programa executável particular. Os programas da forma vetorizada usam as facilidades da WT, enquanto os distribuídos são executados seqüencialmente; ambos são processos da máquina virtual PVM.

Para simular um modelo neuronal nesse ambiente são executados basicamente três passos:

1. Especificação da arquitetura neuronal e da sua simulação;
2. Geração do código-fonte referente à especificação;
3. Compilação e Execução dos códigos-fonte.

A seguir cada um desses passos será descrito com mais detalhes:

4.1 Especificação da Arquitetura Neuronal

Nessa fase, o usuário do ambiente especifica a sua simulação através de uma linguagem de descrição própria do HENNS. Essa especificação é feita em dois níveis:

Estrutural: no HENNS, uma rede neuronal é definida como uma estrutura hierárquica de três níveis:

- Rede: Tem por principal finalidade representar a rede neuronal, provendo a abstração necessária ante o mundo externo e servindo como base para especificação do funcionamento dos diversos conjuntos.
- Conjunto: É uma abstração que condensa nodos semelhantes, os quais têm as mesmas conexões de entrada e saída e funcionam de forma semelhante. No *back-propagation*, por exemplo, cada nível da rede corresponderia a um conjunto.
- Nodo: É a menor unidade funcional de uma rede neuronal.

Além da hierarquia de componentes, a definição da estrutura contem as conexões entre esses. No caso do HENNS, essas conexões podem ser entre a rede e algum conjunto ou entre conjuntos. Não há conexões intra-conjuntos por motivos de eficiência e paralelização.

Funcional: a definição do funcionamento da simulação é feita de forma procedimental na própria linguagem de definição da rede. A rede neuronal (e conseqüentemente os outros componentes da hierarquia) pode possuir vários modos de operação, cada um definido como uma seqüência de comandos que pode incluir os seguintes:

- Protocolo: são os comandos da linguagem que implementam a comunicação transparente entre componentes da hierarquia, independente da arquitetura e configuração de máquinas utilizada;
- Funções de Usuário: todas as funções relativas ao funcionamento interno de cada um dos componentes nos diversos modos de operação são definidos pelo usuário. Essa definição é feita na linguagem da máquina destino usando, para acesso às estruturas de dados dos componentes da hierarquia, funções da biblioteca do HENNS, como por exemplo a função getinput, que retorna o valor de uma dada entrada.
- Fluxo de Controle: também são providas duas formas de fluxo de controle, while e if, tornando a linguagem procedimental mais poderosa e flexível.

4.2 Geração de Código Fonte

Nessa fase, com base no arquivo de especificação, são geradas várias informações:

Definições: são as constantes e outras definições oriundas da especificação da rede. Essas definições serão usadas na compilação da aplicação (vide Secção 4.3);

Inicialização: são funções destinadas a inicializar as estruturas de dados referentes ao modelo neuronal e ao protocolo de comunicação do HENNS;

Operação: são as funções que implementam o fuincionamento da rede;

Compilação: são as especificações e dependências de compilação da aplicação e dos seus componentes, na forma de um arquivo tipo *makefile*.

Simulação: é um *script* dos passos a serem seguidos para a execução da aplicação, passando pela compilação, instanciação de *daemons* do PVM e do WT e execução da aplicação propriamente dita.

4.3 Compilação e Execução do Código Fonte

Para compilar a aplicação gerada, o arquivo *makefile* é invocado e todo o processo é feito automaticamente: os códigos fonte de inicialização e operação gerados são compilados e acoplados à biblioteca do HENNS, resultando nos executáveis da aplicação. Desta forma, se a simulação for executada seqüencialmente, apenas um executável é criado; por outro lado, se uma rede com três conjuntos for toda ela executada distribuidamente, serão gerados quatro executáveis, um para a rede e um para cada conjunto. No caso do processamento vetorizado, é utilizado o compilador do multiC e gerado um executável como na forma distribuída.

A execução da simulação se faz através do programa de rede. Quando a simulação é distribuída, esse dispara os demais programas, fazendo todo o controle da inicialização e finalização dos processos "filhos".

5 Resultados

Os resultados a seguir são fruto da investigação sobre a arquitetura mais adequada para a simulação de redes neuronais dentre as disponíveis no DCC. Para que isso fosse avaliado, foi especificada uma rede *back-propagation* [D. E. Rumelhart and Williams, 1986] sobre a qual foram feitos os testes. Assim, foram alteradas as quantidades de nodos em cada nível para avaliar os diferentes fatores que serão discutidos a seguir.

5.1 Ambiente de Execução

Os testes apresentados a seguir foram executados na rede de *Suns* do DCC - UFMG, a máquina SIMD *Zephyr* - *Wavetracer* estava acoplada a uma *Sparc station 2* também conectada à rede do DCC.

Assim, foram efetuadas simulações com configurações de aplicação variadas:

Seqüencial: O código da simulação era executado em uma única *workstation*. No caso, foram utilizados os modelos *Sparc station 2* e *Sparc station SLC* ambas da marca *Sun*;

Distribuído: Os conjuntos ou partes de conjuntos foram executados em máquinas *SLC*, um executável por máquina;

Vetorizado: Nesse caso, instanciou-se dois processos na máquina hospedeira da WT: (i) o correspondente ao conjunto a ser simulado vetorizadamente; (ii) o correspondente ao restante da rede. Essa separação se fez necessária tendo em vista a linguagem utilizada para o vetorizado (multiC) ser diferente da utilizada para o seqüencial (C).

Heterogêneo (Distribuído e Vetorizado): Nessas simulações, há três tipos de pares executável-máquina: (i) o programa da rede é executado em uma *Sparc 2*; (ii) o programa do conjunto vetorizado executa na hospedeira do WT; (iii) os demais conjuntos executam, cada um, em máquinas *SLC*, um por máquina.

Inicialmente será avaliada a forma mais adequada de se simular um conjunto isoladamente, a seguir considerar-se-á a rede como um todo usando as medidas próprias de arquiteturas neuronais.

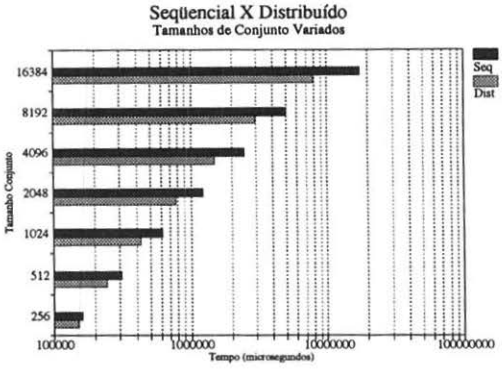

Figura 1: Seqüencial × Distribuído

5.2 Distribuído × Seqüencial

Nesse teste verificou-se a adequação do processamento distribuído para a simulação de conjuntos com um número variável de nodos. Desta forma, foram criadas RNs com número de neurônios na camada de entrada e de saída fixos, enquanto era variado o tamanho do conjunto referente ao nível intermediário. Tanto as simulações seqüenciais quanto as distribuídas foram executadas em máquinas *Sun SLC*, de forma que o teste fosse fidedigno.

Pelo gráfico da Figura 1 pode-se observar os resultados para conjuntos de tamanho variando de 256 até 16384 nodos. As execuções distribuídas foram claramente superiores às seqüenciais, sendo o tempo dessas últimas 10 % a 70 % maior que o das primeiras. O ganho foi menor para os conjuntos de menor tamanho, o que pode ser explicado pelo *overhead* decorrente da comunicação entre os processos.

5.3 Particionamento de Conjunto

Para verificar o peso do *overhead* de comunicação entre processos, foram simuladas redes neuronais cujos conjuntos intermediários estavam particionados em 2 e 4 processos. Cada um desses processos foi colocado em uma máquina separada e os níveis de entrada e saída permaneceram inalterados. Os resultados podem ser vistos no gráfigo da Figura 2, onde onde verifica-se claramente que o particionamento é mais adequado para conjuntos maiores, tornando-se desvantajoso à medida que os conjuntos diminuem de tamanho. No caso do ambiente da presente simulação, isso ocorreu com conjuntos de tamanho 128, que ao contrário dos de tamanho 256, já mostram ser a partição desvantajosa.

5.4 Vetorizado × Seqüencial

Aqui foi comparado o desempenho do algoritmo seqüencial com o que utiliza um conjunto vetorizado. Isso foi feito no intuito de verificar se realmente o uso de processamento vetorial seria interessante para a aplicação em questão, no ambiente computacional utilizado. Todas as simulações foram feitas na *Sun Sparc 2* hospedeira do WT, que usava ou não a máquina SIMD conforme fosse o caso.

Observando o gráfico da Figura 3, verifica-se que o vetorizado foi até 50% mais rápido do que o seqüencial. Entretanto, esse ganho foi diminuindo gradativamente, chegando a 15% para o conjunto intermediário com 256 nodos. Essa variação pode ser explicada pelo custo de comunicação entre a máquina hospedeira e a WT.

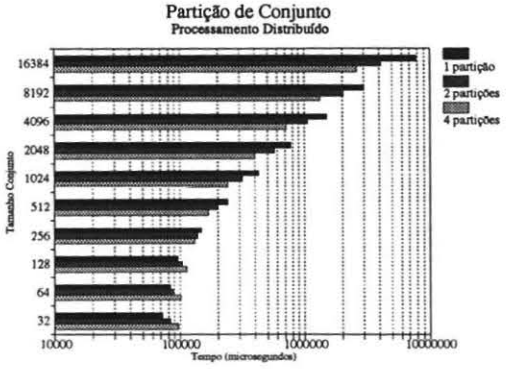

Figura 2: Particionamento de conjuntos

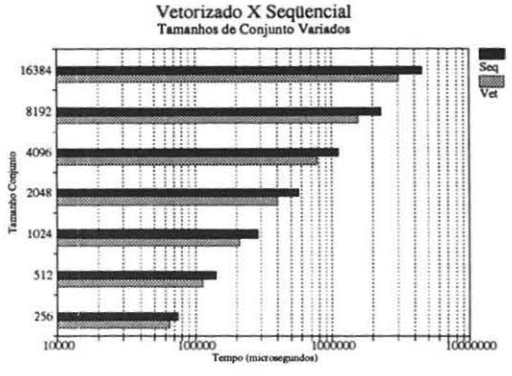

Figura 3: Seqüencial × Vetorizado

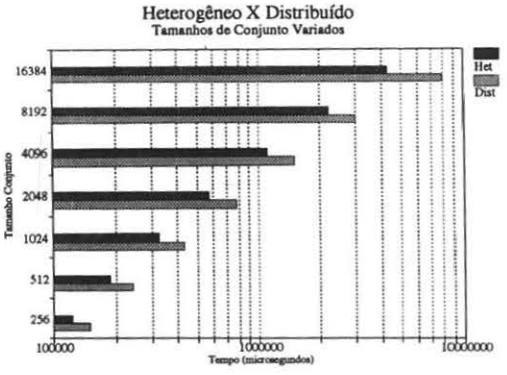

Figura 4: Heterogêneo × Distribuído

5.5 Distribuído × Heterogêneo

Após ter sido verificada a supremacia das simulações distribuída e vetorizada ante os seus pares seqüenciais, avaliou-se qual seria o ganho decorrente da utilização de ambas as soluções simultaneamente. Desta forma, a hospedeira da máquina WT se tornaria um nodo de processamento distribuído configurando um ambiente de processamento heterogêneo.

No gráfigo da Figura 4 observa-se o desempenho da simulação heterogênea ante a distribuída, onde verifica-se ser a primeira melhor. Nesse caso, o ganho variou de 20 a 80 % na medida que o conjunto utilizado era maior.

5.6 Outras Medidas de Performance

De forma a comparar diferentes simulações, alguns tipos de procedimentos padrão de medida devem ser providos. Medidas como velocidade de processamento aritmético, por exemplo, embora reflitam a essência dos algoritmos neuronais, podem ser bastante influenciados por diferentes modelos neuronais e por particularidades do problema abordado.

A seguir, serão apresentados resultados utilizando medidas de performance encontradas na literatura [Nordström and Svensson, 1992, M. H. Garzon and Dickerson, 1992]. Para fazer esses testes foi necessário variar a conectividade média dos nodos da rede, desta forma, optou-se por manter o conjunto intermediário fixo e variar os de entrada e saída.

5.6.1 CPS *Connections per Second*

A cada ocasião em 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 medida de performance.

O gráfico da Figura 5 mostra a comparação entre as simulações heterogênea e distribuída tomando por base essa medida de performance. As curvas resultantes mostram que a simulação heterogênea processa 50% mais CPS do que a distribuída. Isso permite dizer que as variações individuais de cada curva dependem mais do *overhead* de comunicação entre os processos do que do processamento interno propriamente dito.

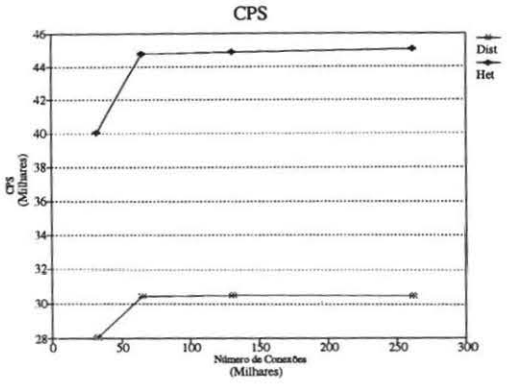

Figura 5: *Connections per Second*

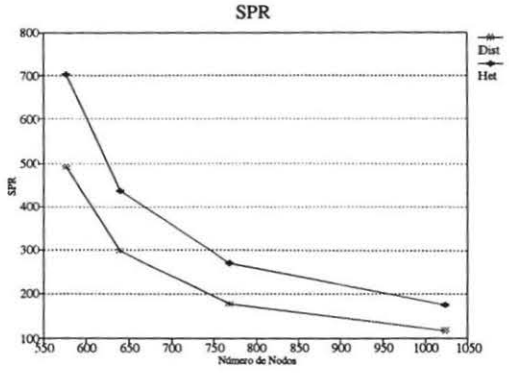

Figura 6: *Synaptic Processing Rate*

5.6.2 SPR *Synaptic Processing Rate*

Essa medida indica o balanceamento entre poder de processamento e tamanho da rede. Ela se baseia no argumento de que um neurônio biológico dispara aproximadamente 100 vezes por segundo. Isso implica que cada sinapse processa sinais a uma taxa de 100 por segundo; assim o SPR humano é aproximadamente 100. Uma vez que o cérebro humano, é, sem dúvida, a rede neuronal mais perfeita de que se tem notícia, esse número serve como parâmetro de comparação do poder de processamento da simulação.

No gráfico da Figura 6 verifica-se que o heterogêneo também é superior ao distribuído tendo um SPR aproximadamente 50 % maior. Entretanto, ocorre também um decrescimento da taxa de SPR à medida que o número de nodos é aumentado, o que pode ser explicado pelo aumento do número de conexões a serem processadas.

6 Conclusões

No presente trabalho foi avaliada a possibilidade da heterogeneidade como estratégia para atenuar o custo computacional das simulações de redes neuronais.

Utilizando a rede de *Suns* do DCC - UFMG e a máquina SIMD Zephyr-Wavetracer, foi verificada inicialmente a viabilidade das simulações distribuída e vetorizada de redes neuronais, seguida de uma avaliação da estratégia proposta. Nessa avaliação pode-se constatar que a estratégia proposta permite ganhos diretamente proporcionais ao tamanho da rede a ser simulada, o que pode ser explicado pelo custo de comunicação associado às implementações. Quantitativamente, o tempo de simulação distribuída foi até 80 % maior que o utilizando a solução heterogênea, sem considerarmos a simulação seqüencial, que foi maior por uma ordem de magnitude.

Cabe ressaltar que essa estratégia é geral desde que se use máquinas com especificações arquiteturais semelhantes, o que não indica *a priori* que os resultados serão os mesmos.

Como perspectivas futuras desse trabalho pode-se considerar a simulação de outros tipos de redes como, por exemplo, as auto-organizativas, onde o dinamismo da topologia da rede traria à tona problemas como partição e relocação de agrupamentos. Outra questão é investigação da eficiência estratégia em outros ambientes computacionais.

Referências

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

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

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

[Feldman et al., 1988] Feldman, J. A., Fanty, M. A., Goddard, N. H., and Lynne, K. J. (1988). Computing with structured connectionist networks. *Communications of the ACM*, 31(2):170 –187.

[Forrest et al., 1987] Forrest, B. M., Roweth, D., Stroud, N., Wallace, D. J., and Wilson, G. (1987). Implementing neural network models on parallel computers. *The Computer Journal*, 30(5):413 – 419.

[Fortuna et al., 1990] Fortuna, L., G.Muscato, G.Nunnari, and Sicurella, G. (1990). A transputer-based multilayer perceptron network simulator.

[Geist et al., 1993] Geist, A., Benguelim, A., Dongarra, J., Jiang, W., Manchek, R., and Sunderam, V. (1993). *PVM 3.0 User's Guide and Reference Manual*. Oak Ridge National Laboratory.

[Ghosh and Hwang, 1989] Ghosh, J. and Hwang, K. (1989). Mapping neural networks onto message-passing multicomputers. *Journal of Parallel and Distributed Computing*, (6):291 – 330.

[Hecht-Nielsen, 1988] Hecht-Nielsen, R. (1988). Applications of counterpropagation neural networks. *Neural Networks*, (1):131 - 140.

[Heileman et al., 1992] Heileman, G. L., Georgiopoulos, M., and Roome, W. D. (1992). A general framework fo concurrent simulation of neural network models. *IEEE Transactions on Software Engineering*, 18(7):551 – 562.

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

[Meira Jr, 1993] Meira Jr, W. (1993). Implementação de redes neuronais em ambientes paralelos. Master's thesis, Departamento de Ciência da Computação - UFMG, Belo Horizonte, MG.

[Meira Jr. and Carvalho, 1992] Meira Jr., W. and Carvalho, M. L. B. (1992). *SIRNEM: A Parallel Neural Network Simulation System.* In XVIII *Conferencia Latinoamericana de Informatica*, pages 757 – 761. Centro Latinoamericano de Estudios em Informatica CLEI.

[Murre, 1993] Murre, J. M. J. (1993). Transputers and neural networks: An analysis of implementation constraints and performance. *IEEE Transactions on Neural Networks*, 4(2):284 – 292.

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

[Perez, 1990] Perez, M. J. (1990). Simulação paralela de redes neurais numa rede de transputers. Manuscrito.

[Salles et al., 1992] Salles, J., Vieira, M., Meira Jr., W., and Carvalho, M. L. B. (1992). Implementação de redes neuronais em máquinas simd (*Implementation of Neural Networks in SIMD Machines*). In *IV Simpósio Brasileiro de Arquiteturas de Computadores e Processamento de Alto Desempenho*, pages 347 – 361.

[Simpson, 1990] Simpson, P. K. (1990). *Artificial Neural Systems - Foundations, Paradigms, Applications and Implementations.* Neural Networks: Research and Applications. Pergamon Press, 1st edition edition.

[Tollenaere and Orban, 1991] Tollenaere, T. and Orban, G. A. (1991). Simulating modular neural networks on message-passing multiprocessors. *Parallel Computing*, 17(17):361–379.

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

[Zurada, 1992] Zurada, J. M. (1992). *Introduction to Artificial Neural Systems.* St Paul, USA.