Proceedings of the IV Brazilian Conference on Neural Networks - IV Congresso Brasileiro de Redes Neurais
pp. 142-147, July 20-22, 1999 - ITA, São José dos Campos - SP - Brazil

Co-Processador Reconfigurável para a Memória Esparsamente Distribuída de Kanerva

Marcus Tadeu Pinheiro Silva
Coord. de Eletrônica - Centro Federal de Educação Tecnológica de Minas Gerais
Av. Amazonas, 5253, CEP 30480-000, Belo Horizonte, MG

Antônio de Pádua Braga
Depto. de Eng. Eletrônica - Universidade Federal de Minas Gerais
Caixa Postal 209, CEP 30.161-970, Belo Horizonte, MG

E-mails: silva@novell.cpdee.ufmg.br, apbraga@cisne.cpdee.ufmg.br

Abstract

The implementation on hardware of the first layer of Kanerva's Sparse Distributed Memory (SDM) is presented in this work. The hardware consists on a co-processor board for connection on ISA standard Bus of an IBM-PC AT compatible computer. The board, named CR-SDM (Reconfigurable Co-processor for SDM), comprises of Xilinx FPGAs. 524 Kbytes of RAM and bus interface circuits. Based on in-system reconfiguration capacity of FPGAs, CR-SDM alows easily change of the characteristics of SDM topology implemented. First results show a speed-up of four times of CR-SDM in relation to a software implementation of the algorithm.

1. Introdução

No projeto de um microprocessador de propósito geral existe um compromisso entre a área de silício disponível e o repertório de instruções. Assim, em função da qualificação *"propósito geral"*, selecionam-se as instruções do repertório com base em duas premissas. Primeiro, o repertório deve ser *completo*, no sentido de que seja possível construir programas capazes de implementar *qualquer* algoritmo. Segundo, tal repertório deve ser *eficiente* para permitir que os algoritmos mais frequentemente requeridos possam ser executados rapidamente usando uma pequena quantidade de instruções [1]. Dadas estas características do repertório, os sistemas construídos com tais microprocessadores são capazes de executar com eficiência programas em uma ampla faixa.

Porém, o desenvolvimento da computação nas últimas décadas tem continuamente gerado novos algoritmos, sendo que para muitos destes a programação utilizando um conjunto fixo de instruções resulta em tempos de execução incompatíveis com sua aplicação prática. Em geral, tais algoritmos operam sobre grandes quantidades de dados e muitas vezes também requerem operações para as quais os microprocessadores não estão adaptados, ou seja, operações que demandam longas seqüências de instruções de máquina para serem implementadas. Algoritmos com esta característica ocorrem com frequência na área de redes neurais artificiais (RNA's), sendo que o processamento da memória esparsamente distribuída (SDM) de Kanerva [2] propicia um exemplo ilustrativo disto.

Desde o fim da década de 80 vários ASIC's (*Application Specific Integrated Circuits*) desenvolvidos para o processamento de RNA's tem sido apresentados[3]. Contudo, tais componentes são muitas vezes de difícil obtenção, e de alto custo. E há ainda modelos neurais para os quais nenhum circuito integrado (CI) específico encontra-se disponível, sendo este o caso do modelo de Kanerva. Assim, em casos de aplicação prática ou de pesquisa na área de RNA's, onde a quantidade de unidades do processador dedicado é baixa, muitas vezes a abordagem ASIC para acelerar o processamento pode ser inviável.

Porém, recentemente surgiu uma segunda opção para acelerar o processamento de RNA's, ou de outros algoritmos computacionalmente intensivos. Esta nova opção são os processadores baseados em FPGA's (*Field Programmable Gate Arrays*)[4].

Este artigo descreve o desenvolvimento do CR-SDM (Co-processador Reconfigurável para SDM), um processador baseado em FPGA's especificamente projetado para a SDM. Apesar de este projeto ser orientado para o modelo de Kanerva, o CR-SDM pode ser usado para implementar em *hardware* outros algoritmos, de modo a acelerar seu processamento. Nas seções que se seguem é apresentada a SDM de Kanerva e a análise do modelo que norteou o projeto do CR-SDM. A seguir,

descreve-se a arquitetura do CR-SDM e é apresentado o resultado obtido com o mesmo no processamento da SDM.

2. Descrição da SDM

A SDM é uma memória associativa adequada para armazenar informações codificadas em longas cadeias de bits, ou seja, vetores binários com centenas ou milhares de coordenadas; ao desenvolve-la Kanerva se baseou nas propriedades do espaço binário de alta dimensão. A SDM pode ser analisada de duas formas: ou como uma rede neural *feed-forward* de duas camadas, ou como uma generalização da RAM convencional de computador. Neste texto a discussão da SDM baseia-se principalmente na segunda abordagem.

A Fig. 1 apresenta a visão esquemática da SDM. Tal visualização tem uma correspondência direta com a representação da SDM em forma de rede [5]. Assim, os blocos **S**, **d** e **y** equivalem a primeira camada da rede SDM, e os blocos **H**, **v** e **O** equivalem a segunda camada. Os blocos componentes da SDM e sua operação são descritos a seguir.

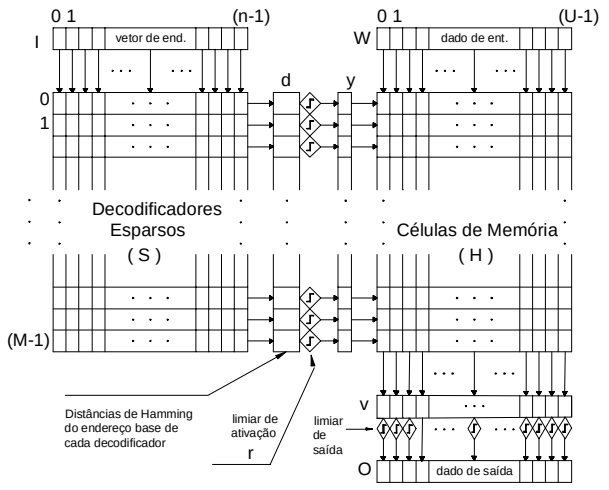

Fig. 1 – Visão Esquemática da SDM

i) Matriz de Decodificadores Esparsos - **S**

Em cada linha desta matriz temos um decodificador que é capaz de ativar uma única locação de memória (a correspondente linha na matriz **H**). Mas, enquanto em uma RAM convencional temos que cada decodificador ativa sua saída para um único endereço (vetor de entrada **I**), no caso da SDM o decodificador ativa sua saída dado que o endereço de entrada **I**, esteja até a uma certa distância de Hamming máxima ("r"), do endereço fixado no decodificador. Logo, na SDM há um conjunto de endereços em torno do endereço base do decodificador tal que a saída do mesmo é ativada; não é único o vetor **I**, para o qual a saída do decodificador é ativada. De acordo com a descrição podemos obter que um decodificador ideal para a SDM é um neurônio com função de ativação degrau [6].

A dimensão de **I**, *n*, deve ser alta, de modo que as propriedades do espaço binário de alta dimensão tornem-se válidas. Tendo vetores binários de *n* posições, define-se um espaço de vetores com 2^n elementos. Supondo, por exemplo, $n = 1000$, 2^n é uma quantidade astronômica. Claramente, é impossível ter-se uma RAM com 2^{1000} locações. Mas, se no lugar de ter 2^n, forem usados M decodificadores e locações (onde $M << 2^n$), usando um limiar adequado pode-se fazer com que cada decodificador englobe uma área razoável do espaço binário. Assim, no total estes M decodificadores poderão incluir a maior parte do espaço 2^n.

ii) Matriz de Locações de Memória - **H**

Em **H**, cada linha corresponde aproximadamente aos registradores de uma célula de memória . Mas, ao invés de um flip-flop para cada posição de bit, na SDM temos um contador binário para cada posição. Se, na escrita da SDM, o bit em armazenamento na posição é 1, o contador é incrementado, se é 0, o contador é decrementado. Mas, há ainda outra diferença fundamental entre a SDM e a RAM convencional. Na RAM em cada acesso uma única locação de memória é ativada. Na SDM em cada acesso várias locações de memória são ativadas, ou seja, a informação quando é armazenada na SDM é distribuída em várias locações. Mais ainda, uma locação em geral será ativada várias vezes quando muitos dados estiverem sendo armazenados.

iii) Obtenção do Vetor de Saída da SDM

Para obtenção do vetor de saída, **O**, temos para cada uma de suas posições de bit um somador. Cada somador tem como valores a serem somados aqueles obtidos dos contadores das locações ativadas na leitura. Podemos ter uma soma igual a 0, ou uma soma positiva, ou uma soma negativa. O valor negativo ocorre desde que nas locações ativadas na posição de bit em questão, tenham sido armazenados mais vezes 0's (contador decrementado) do que 1's (contador incrementado). Análise análoga aplica-se a obtenção de valor positivo.

Após o somador temos a aplicação de uma função degrau, tal que, se a saída do somador for maior ou igual a 0, o correspondente bit será 1, caso contrário será 0.

iv) Recuperação de informação na SDM

A recuperação do dado originalmente escrito na SDM é possível com base na análise a seguir.

Quando da escrita de um dado ζ ele é armazenado de forma distribuída naquelas j locações ativadas através de um correspondente vetor de endereço ξ. Como já indicado antes, estas j locações não são exclusivas quanto a alteração a partir do endereço ξ, ou seja, muitas destas j locações poderão ser atingidas nas escritas de outros dados em outros endereços. Mas de

FASE DE LEITURA

***v** = 0;*
Para Lin de 0 à M-1
a) *Se **y**(Lin) =1* *Para Col de 0 a U-1* ***v**(Col) = **H**(Lin, Col) + **v**(Col);*

*(Para cada locação (linha) de **H** acessada, acrescente em **v** o conteúdo da locação)*

Para Col de 0 à U-1 *(Obtém o resultado de leitura da SDM (**O**)*
b) *Se **v**(Col) ≥ 0* ***O**(Col) = 1;* *aplicando função degrau com limiar zero*
*Senão **O**(Col) = 0;* *a cada coordenada de **v**)*

FASE DE ARMAZENAMENTO

Para Lin de 0 à M-1
c) *Se **y**(Lin) =1* *Para Col de 0 a U-1* ***H**(Lin, Col)= **H**(Lin, Col + **W**(Col);*

*(Para cada locação (linha) de **H** acessada, atualize o conteúdo da locação de acordo com vetor de dado **W**)*

---

*Onde: **H** é uma matriz de M linhas por U colunas, **v** e **O** são vetores de U colunas, **y** é um vetor de M linhas (**H** e **v** operam com um inteiro em cada posição; **y** e **O** operam com um bit em cada posição). Para o algoritmo de armazenamento o vetor **W** deve conter +1 ou -1. Usando **W** com valores 0 e 1, a operação de atualização deve ser codificada usando: Se **W**(col) = 1 Incremente **H**(Lin, Col);*
*Senão decremente **H**(Lin,Col);*

Fig. 2 – Algoritmos da Segunda Camada da SDM

cada vez que outro endereço de escrita atingir o armazenamento distribuído de ζ, ele de fato atingirá apenas uma pequena fração das j locações que ξ atingiu. Supondo uma probabilidade idêntica de "1's"e "0's", quando novamente as j locações forem ativadas na leitura com endereço ξ, as contribuições das escritas de interferência nas locações de ζ se anularam mutuamente, e na leitura se obterá o vetor de dado ζ. Uma condição adicional deve ser imposta para a validade da análise acima: é necessário que não tenham sido armazenados tantos dados na SDM a ponto de que interferência acumulada nas locações de ζ, atinja uma grande proporção das j locações. Esta última condição carrega a noção de que, como em outros tipos de memórias associativas, na SDM a capacidade de armazenamento é uma proporção reduzida de M, o número de locações de memória.

Com o armazenamento ocorrendo de forma distribuída, o valor lido da SDM é um resultado estatístico, onde a "regra da maioria" quanto ao que foi escrito nas locações ativadas constitui o valor efetivamente lido.

3. Algoritmos da SDM

A partir da descrição da SDM, pode-se expressar em forma de algoritmo o processamento em cada uma de suas camadas. São exatamente estes os algoritmos que devem ser codificados em uma linguagem qualquer de forma a realizar uma implementação em *software* da SDM.

3.1 Segunda Camada

No caso da 2ª camada, a SDM possui dois algoritmos: um para a fase de leitura e outro para a fase de armazenamento (Fig. 2). Para estes algoritmos existe um paralelismo inerente no âmbito das colunas de **H** (operações *"a"* e *"c"* da Fig. 2). Contudo, deve ser observado que tais operações só ocorrem para as locações de memória acessadas na matriz **H** (condição *Se **y**(Lin)= 1*).

3.2 Primeira Camada

Nesta camada, existe um único algoritmo (Fig. 3), ou seja, independente do acesso a SDM ser para leitura ou escrita, a decodificação de endereços da camada 1 ocorre da mesma forma.

Para Lin de 0 à M-1 {
a) *Dif = **S**(Lin)⊗**I**;*
b) *d = cont_uns(Dif);*
c) *Se d ≤ r* ***y**(Lin) = 1;*
*senão **y**(Lin) =0*
}

---

*Onde: **S** é uma matriz de M linhas por n colunas, **I** é um vetor de n colunas, e **y** é um vetor de M linhas (todos operam com um bit em cada posição).*

Fig. 3 – Algoritmo da Primeira Camada da SDM

Podem ser feitas as seguintes observações quanto ao algoritmo apresentado:

- Existe possibilidade de paralelização do algoritmo, mas diferentemente da 2ª camada, agora o paralelismo ocorre no âmbito das linhas de **S**.
- Como não existe no repertório de instruções de microprocessadores de propósito geral instrução para obter diretamente a distância de Hamming entre dois vetores binários, isto foi ressaltado na Fig. 3 codificando a operação em duas etapas (*"a"* e *"b"*).
- A operação *"b"* deve corresponder a um outro algoritmo especifico, porque microprocessadores não estão equipados com instrução para a obtenção direta da quantidade de "1's" em um operando.

3.3 Escolha do Algoritmo para Implementação em *Hardware*

Após a análise dos algoritmos acima decidiu-se implementar no CR-SDM o algoritmo da primeira camada da SDM. A justificativa fundamenta-se nos seguintes argumentos:

a) O requisito de memória da matriz **S** é muito menor que o da matriz **H**. Usando um exemplo que foi implementado no CR-SDM, com *n*= 256, U= 256, M=16384, obtém-se que o requisito de memória da matriz **S** é igual a 16384 X 256 X 1 bit. Um bit porque a matriz **S** contem apenas valores binários. Ou seja, têm-se na matriz **S** 4 Mbits. Já para a matriz **H**, temos que cada posição deve armazenar um inteiro, considerando que inteiros de 8 bits sejam suficientes o cálculo de memória para **H** é 16384 X 256 X 8 bits. Ou seja, a matriz **H** do exemplo corresponde a 32 Mbits. Para acelerar ao máximo o processamento o CR-SDM utiliza SRAM's de 15 ns de tempo de acesso. Tais memórias são as mesmas usadas para construção de bancos de memória *cache*, e embora rápidas tem pequena capacidade; atualmente o tipo com maior capacidade e disponível em encapsulamento adequado à construção de um protótipo tem capacidade de 1 Mbit. Assim, torna-se fisicamente inviável instalar CI's para implementar 32 Mbits de SRAM em um protótipo de padrão ISA, como é o caso do CR-SDM.

b) O algoritmo da primeira camada é o que maior fatia de tempo ocupa em cada acesso a SDM [7]. Assim, a aceleração do processamento da 1ª camada é aquele que maior impacto ocasiona no aumento da velocidade de processamento de toda SDM. A justificativa para esta maior fatia de tempo da 1ª camada resulta diretamente do fato de que em cada acesso toda matriz **S** deve ser processada, enquanto na segunda camada pode-se usar algoritmos que realizam processamento apenas em uma pequena porção da matriz **H**.

3.4 *Hardware* de processamento da 1ª camada.

Para o processamento em *hardware* do algoritmo da 1ª camada foi desenvolvido um elemento de processamento (EP) do tipo bit-serial (Fig. 4). Pode-se mostrar [8] que, no caso das FPGA's usadas no CR-SDM, este EP é mais adequado para a aceleração do processamento da 1ª camada do que EP's do tipo bit-paralelo.

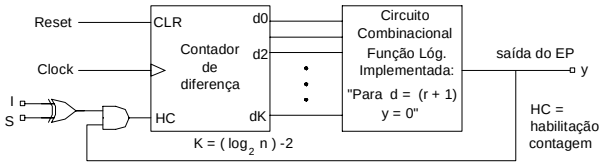

Fig. 4 – EP bit-serial para a 1ª camada da SDM.

4. *Hardware* X *Software* na implementação de RNA's

Pode-se visualizar (Fig. 5) a implementação de RNA's na forma de uma linha, onde em uma ponta tem-se o *hardware* dedicado, na outra ponta o *software* simulador rodando em um computador de uso geral, e no meio uma gradação contínua onde alguma quantidade de *hardware* especializado libera o *software* de partes cada vez mais significativas do processamento. Dentro desta ótica, uma SDM construída usando apenas CI's dedicados constitui a ponta do *hardware*, ou seja, toda SDM sendo implementada como um sistema de *hardware* dedicado, tipo uma caixa preta, onde no armazenamento ela recebe entradas **I** e **W**, e na leitura recebe a entrada **I** e fornece a saída **O**.

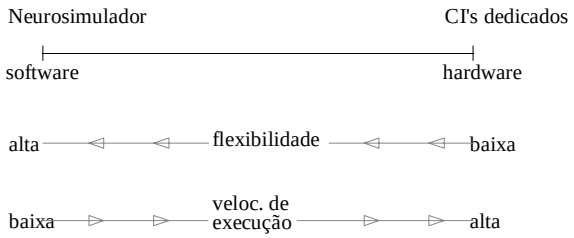

Fig. 5 – *Hardware* X *Software* na Implementação de RNA's

Contudo, uma implementação em *software*, a outra ponta da linha, constitui a alternativa mais flexível, ou seja, com uma SDM implementada em um simulador pode-se variar com facilidade todos seus parâmetros (*n*, M, U, r). No entanto, esta ponta resulta sempre em uma

velocidade de execução menor, do que utilizando qualquer quantidade de *hardware* que processe de forma dedicada pelo menos parte da rede.

O objetivo básico do CR-SDM foi criar um *hardware* de processamento dedicado para executar o algoritmo da 1ª camada da SDM, mas com uma flexibilidade similar à presente nas implementações em *software*. Para atingir tal objetivo o CR-SDM adota o processamento baseado em FPGA's, que se insere dentro do conceito de computação reconfigurável [4].

Computação reconfigurável (ou adaptativa) é a utilização de *hardware* programável para criar estruturas digitais dedicadas à execução de instruções específicas de um problema computacional. O termo instrução aqui deve ser entendido da mesma forma que uma instrução qualquer de um microprocessador. Por exemplo, não existe qualquer instrução em um microprocessador para obter a distância de Hamming entre dois vetores binários. Porém, usando computação adaptativa pode-se implementar a instrução no *hardware* de um co-processador com FPGA's e executá-la muito mais rapidamente do que usando um algoritmo executado em uma CPU. Além disso, os co-processadores com FPGA's permitem implementar um paralelismo na execução de instruções, onde a mesma instrução é executada ao mesmo tempo sobre diferentes dados, utilizando vários elementos de processamento.

A flexibilidade desta abordagem de processamento em *hardware* se aproxima da presente em simuladores, pois as FPGA's são CI's programáveis *"in-system*", sendo que seus tempos de programação são da ordem de milissegundos. Ou seja, em frações de segundo pode-se alterar a instrução implementada no co-processador, bastando para isto carregar nas FPGA's uma nova seqüência de bits de configuração.

5. Arquitetura do CR-SDM

A Fig. 6 apresenta o diagrama de blocos do CR-SDM, e Fig. 7 a foto do protótipo construído. Segundo a classificação apresentada por Hartenstein [9], para os sistemas de computação reconfigurável, o CR-SDM se enquadra na classe das máquinas de computação reconfigurável de conjunto de instruções ampliado. A característica básica desta classe é o estabelecimento da plataforma reconfigurável através da conexão de uma máquina de arquitetura von Neumann (hospedeiro) a um co-processador baseado em FPGA's. Assim, a computação adaptativa ocorre da seguinte forma. O *software* da aplicação em execução no hospedeiro configura as FPGA's e suas interconexões no co-processador de forma a implementar em *hardware* o(s) algoritmo(s) que ocupa(m) a maior fatia do tempo de execução da aplicação. A medida em que o *software* no hospedeiro é executado são transferidos dados para processamento no co-processador e são lidos os resultados de volta. Deve-se notar que quando um algoritmo é implementado no *hardware* do processador ele passa a ser referido como uma instrução do repertório do mesmo, daí a denominação desta classe.

Tem-se a seguir uma descrição dos blocos do CR-SDM, sendo verificado o direcionamento da arquitetura para a aceleração do processamento da SDM.

Interface de Barramento e configuração das FPGA's

Este bloco controla o acesso do hospedeiro ao co-processador. Este acesso pode ser de dois tipos: a) para carregamento dos bits de configuração das FPGA's; b) envio de comando para a FPGA de controle.

FPGA de Controle

Nesta FPGA são programadas as estruturas de *hardware* para controle do processamento realizado nos elementos de processamento da outra FPGA. Além disso, esta FPGA identifica os comandos enviados pelo hospedeiro. No caso particular do processamento da primeira camada da SDM os seguintes comandos foram implementados: a) armazenamento e leitura de dado na RAM da matriz **S**; b) armazenamento e leitura de dado na RAM do vetor **I**; c) armazenamento e leitura de dado na RAM do vetor **Y**; d) reinicialização dos controladores de acesso às memórias do CR-SDM; e) disparo do processamento do algoritmo; f) leitura do estado de processamento do algoritmo. Tanto a FPGA1 quanto a FPGA2 do CR-SDM são da série XC4000E da Xilinx.

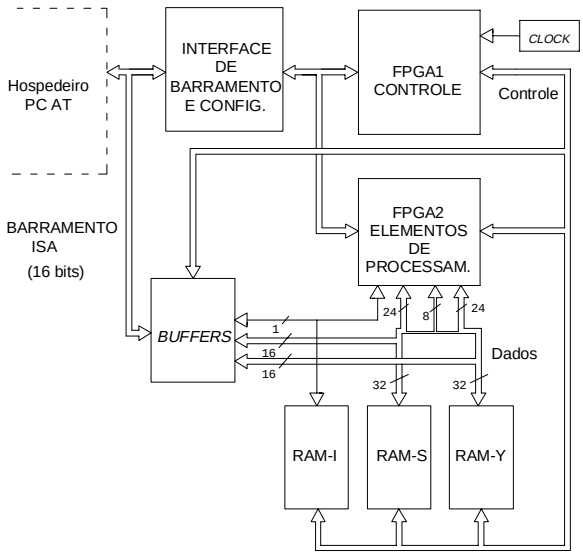

Fig. 6 – Diagrama de Blocos do CR-SDM

Memórias

O CR-SDM está equipado com 524 Kbytes de RAM do tipo usado em bancos de memória *cache*. São três blocos de memória com uma correspondência direta com os blocos **I**, **S** e **y** da Fig. 1. A RAM-S tem 512 Kbytes, a RAM-I 4 Kbytes e a RAM-Y 8 Kbytes. As RAM-S e RAM-Y estão organizadas em barramentos de 32 bits e a RAM-I possui um único pino para dados.

FPGA de Elementos de Processamento (EP's)

Nesta FPGA são programados os EP's que realizam o processamento do algoritmo implementado no CR-SDM. No caso particular do algoritmo da 1ª camada da SDM são implementados 32 EP's do tipo apresentado na Fig. 4, os quais operam em paralelo. A cada ciclo de *clock* cada EP recebe duas entradas: um bit do vetor **I** e um outro bit relativo a uma linha da matriz **S**, sendo que o bit do vetor **I** é igual para todos os EP's.

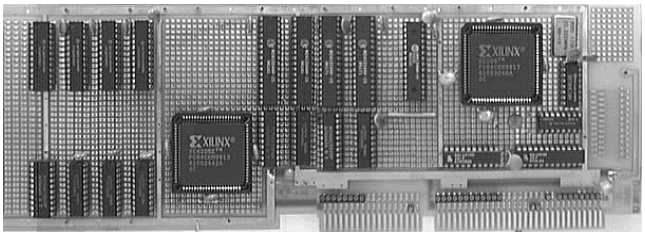

Fig. 7 – Protótipo do CR-SDM

6. Resultados

Para avaliar o desempenho do CR-SDM foram elaborados dois programas de processamento da 1ª camada de uma SDM com $n = 256$ e $M = 16384$.

Um destes programas foi usado como parâmetro de comparação para o tempo de processamento da placa, constituindo-se de uma implementação do algoritmo em *software*. A máquina escolhida para rodar este simulador foi exatamente a mesma que serviu de hospedeiro para o CR-SDM; um microcomputador compatível com IBM-PC AT equipado com CPU Pentium Celeron de 300 MHz. O programa simulador foi codificado em linguagem C, com uma rotina do mesmo sendo codificada em *assembly*. A parte em C tratou dos aspectos de E/S saída para disco e tela. A rotina *assembly* tratou do processamento do algoritmo SDM, de modo a executá-lo no menor tempo possível.

O segundo programa constitui-se de uma modificação do primeiro, onde o processamento na rotina *assembly* foi substituído pelo processamento na placa.

O tempo de processamento[1] da rotina *assembly* foi de 49,8 ms e o tempo de processamento no CR-SDM foi de 13,06 ms. Ou seja, o CR-SDM propiciou uma aceleração de aproximadamente 4 vezes em relação a versão em *software*.

7. Conclusão

Foi apresentado o desenvolvimento do CR-SDM, um co-processador baseado em FPGA's orientado ao processamento dos algoritmos da SDM. Foram também apresentados primeiros resultados obtidos com o CR-SDM no processamento da primeira camada de uma SDM com 16384 locações, de endereços de 256 bits.

Estes resultados mostrando uma performance 4 vezes superior a uma implementação em *software* mostram a validade do processamento baseado em FPGA's, para a obtenção de *hardware* dedicado para RNA's.

A flexibilidade possibilitada pelo reduzido tempo de configuração *in-system* das FPGA's também foi comprovada durante o teste indicado acima. Neste teste, as diversas versões da implementação em *hardware* do algoritmo eram testadas em poucos minutos, sem qualquer modificação física no CR-SDM.

8. Agradecimentos

Os autores agradecem o suporte do Programa Universitário da Xilinx, Inc., e em particular ao Eng. Rogério Moreira. Agradecem também ao CNPq e a FINEP.

Referências

[1] Hayes, J. P. - "Computer Architecture and Organization." 2nd Edition, McGraw-Hill Book Company, New York, USA, pp. 209-210, 1988.

[2] Kanerva, P. – "Sparse Distributed Memory.", MIT Press Cambridge, USA, 1988.

[3] Lindsey, C. S. & Lindblad, T. - "Survey of Neural Network Hardware," Symposium on Aerospace/Defense Sensing and Control and Dual Use Photonics, Proc. of Applications and Science of Artificial Neural Networks Conference, SPIE VOL. 2492, part Two, pp.1194-1205, 1995.

[4] Villasenor, J. & Mangione-Smith, W. H. - "Configurable Computing", Scientific American, June, pp 81-89, 1997.

[5] Kanerva, P. - "Associative-memory models of the cerebellum", in: Artificial Neural Networks 2, I. Aleksander and J. Taylor (Editors), Elsevier Science Publishers B. V., 1992.

[6] Wassermann, P. D. - "Advanced Concepts of the Neural Networks." Van Nostrand Reinhold, Reading, 1993.

[7] Nordström, T. – "Sparse distributed memory simulation on REMAP3," Research Report. TULEA 1991:16, Lulea University of Technology, Sweden, 1991.

[8] Silva, M. T. P. - "CR-SDM – Co-processador reconfigurável para a memória esparsamente distribuída de Kanerva: Estudo e Implementação." dissertação de mestrado nº 225, Programa de Pós-Graduação em Eng. Elétrica, Universidade Federal de Minas Gerais, Maio, 1999.

[9] Hartenstein, R. W. et al. - "Custom Computing Machines vs. Hardware/Software Co-Design: from a globalized point of view," 6th International Workshop on Field Programmable Logic And Applications, Darmstadt, Germany, September 23-25, 1996.

[1] Ambos os tempos são medidas relativas apenas ao processamento do algoritmo, tendo sido realizadas através de um analisador lógico.