# PARTICIONAMENTO HARDWARE/SOFTWARE : PROPOSTA DE SOLUÇÃO COM REDES NEURAIS ARTIFICIAIS

**Maurício Acconcia Dias**[1], **Wilian Soares Lacerda**[1]

[1]Departamento de Ciência da Computação – Universidade Federal de Lavras (UFLA)
Caixa Postal 37 – 37200-000 – Lavras (MG) – Brasil

mdias@comp.ufla.br, lacerda@ufla.br

**Resumo.** O problema do particionamento hardware/software é um dos problemas mais importantes em projetos de desenvolvimento que possuem um hardware/software codesign. Os problemas de particionamento estão no grupo de problemas NP-Completos e, caso seja adotado um caráter de otimização ao problema e passa ao âmbito dos problemas NP-Difíceis. Este fato justifica a utilização de heurísticas para a resolução em tempo aceitável. As heurísticas utilizadas na resolução do problema são algoritmos genéticos para a geração do banco de dados e redes neurais artificias para atacar o problema. O resultado deste projeto de pesquisa é um rede neural artificial que resolve o problema do particionamento para um dada instância apresentando um bom resultado em um curto tempo de execução.

**Palavras-chave.** Hardware/software co-design, Hardware/Software Partitioning, Redes Neurais Artificiais, NP-Difici, Heurísticas

## *HARDWARE SOFTWARE PARTITIONING: SOLVING WITH ARTIFICIAL NEURAL NETWORKS*

***Abstract.*** *The hardware/software partitioning problem is one of the most important step on project development when it has a hardware/software co-design. Partitioning problems are include in the set of NP-Complete problems and when they have a optimizing feature associated the are moved to the set of NP-Hard problems. This fact justifies the usage of heuristics on solving this problem. The choosen heuristics are genetic algorithms for database generation and artificial neural networks for problem solving. The result of this work is a neural network that can solve the problem for an especific instance with good results in a short comptational execution time.*

***Keywords:*** *Hardware Software Codesign, Hardware Software Partitioning, Artificial Neural Networks, NP-Hard, Heuristics*

(Received October 31, 2007 / Accepted November 21, 2007)

## 1. Introdução

Segundo [10], um sistema embarcado é um sistema cuja principal função não é computacional porém é controlado por um computador embarcado nele. Através desta definição é possível perceber que estamos rodeados por sistemas embarcados como celulares, sistemas de carros, eletrodomésticos que vem se desenvolvendo ao longo dos últimos anos.

Os sistemas embarcados modernos necessitam de uma otimização de design priorizando flexibilidade, variabilidade, aplicabilidade e utilizando microprocessa-

dores, *FPGA's, SoC's,* dentre outras plataformas de hardware.

A questão de design e desenvolvimento de um sistema que possui partes em hardware e partes em software é conhecida como hardware/software co-design. Para [3], hardware software co-design significa reunir objetivos a nível de sistema explorando o sinergismo entre hardware e software em seus próprios designs.

Existem hoje várias metodologias disponíveis para construção de um modelo para o hardware/software co-design. Em geral, os modelos apresentados possuem os mesmo problemas. Os problemas apresentados são peculiares ao tipo de aplicação que esta no foco do desenvolvimento e apresentam-se em todas as fases do projeto, desde a modelagem até a fase de implementação do modelo.

O principal problema encontrado no desenvolvimento de projetos para sistemas embarcados é a escolha de qual parte do sistema será implementada em hardware e qual parte do sistema será implementada em software. Este problema é conhecido como Particionamento Hardware/Software.

O problema do particionamento hardware/software é um ponto chave na questão do desenvolvimento tendo em vista que as decisões tomadas nesta etapa do projeto irão influenciar no projeto até o final de sua execução impactando diretamente na performance, na dificuldade de implementação e no custo final do projeto [7].

Segundo [8], os problemas de particionamento em geral são NP-Completos. Os problemas NP-Completos que possuem caráter de otimização são considerados NP-Difíceis, e o problema do particionamento hardware software enquadra-se nesta classificação. Em [1] encontra-se a redução do problema ao problema da mochila, e sendo o problema da mochila NP-Completo, esta redução é a prova de que o problema do particionamento hardware/software é NP-Difícil.

A classe de problemas NP não possui algoritmos de resolução em tempo polinomial conhecidos que apresentem o resultado exato para o problema, sendo então necessária a utilização de heurísticas como forma de encontrar soluções aceitáveis em tempo hábil.

Existem várias heurísticas sendo utilizadas para resolução do problema na área de inteligência computacional como algoritmos genéticos, *simulated annealing* e busca TABU. Alguns destes métodos utilizados são métodos aproximativos, ou seja, garantem uma margem de erro da resposta que encontram para o problema.

A heurística escolhida para solucionar o problema do particionamento hardware software neste trabalho é a Rede Neural Artificial. Segundo [2] as redes neurais artificiais são uma forma de computação não-algorítmica que caracteriza-se por sistemas que em algum nível relembram a estrutura do cérebro humano .

As redes neurais tem se mostrado uma solução viável e com resultados satisfatórios na solução de problemas de classificação, previsão e categorização. O problema do particionamento hardware software enquadra-se na parte de classificação onde as entradas serão generalizadas e então um modelo de desenvolvimento será proposto como saída da rede.

As redes neurais embora não garantam aproximação da resposta exata, quando bem treinadas, podem apresentar respostas muito próximas às desejadas e o único tempo gasto no caso das redes neurais é o tempo de treinamento pois a resposta a um problema por uma rede treinada é praticamente instantânea.

## 2. Hardware/Software Co-Design

Hardware/Software co-design significa unir todos os objetivos do sistema explorando a interação existente entre os componentes de hardware e de software durante o seu desenvolvimento [9].

Existem cinco fases básicas do hardware/software co-design. Inicialmente tem-se a especificação do sistema a ser desenvolvido, passando por transformações de projeto esta especificação passa pelo particionamento e a síntese de hardware, de interface e a geração de código são feitas e por fim uma verificação do projeto [6].

### 2.1. Design de Sistemas de hardware e software

O design de sistemas de hardware/software envolvem modelagem que é o processo de conceitualização, refinamento de requisitos e produção de um modelo, validação que é o processo de se atingir um nível razoável de segurança de que o modelo vai funcionar da forma que foi planejado e a implementação que é a construção física do modelo do hardware e do software executável [3].

A modelagem de sistemas embarcados pode ser homogênea, que no caso é quando uma linguagem de programação ou formalismo gráfico é utilizado para representação do particionamento hardware/software, ou heterogênea que consiste no particionamento a partir do próprio modelo , que em suas fases iniciais possui componentes que podem ser expandidos e modificados para atingir o objetivo do projeto.

A medida que os sistemas ficam mais complexos a validação torna-se essencial para assegurar o correto funcionamento do mesmo alcançando assim os níveis de performance desejados. Esta validação é feita com a simulação do hardware e do software.

A implementação do sistema envolve várias subtarefas começando com a síntese do hardware e compilação do software, que são igualmente complexas. Os principais problemas consistem no particionamento hardware/software que envolve a arquitetura utilizada, os objetivos do projeto e várias técnicas de particionamento e o problema do escalonamento que consiste em ajustar o tempo de início com cada tarefa em um conjunto onde as tarefas estão interligadas por alguma relação.

### 2.2. Particionamento Hardware/Software

O particionamento hardware/software trata a implementação de partes de um projeto de sistema em formas heterogêneas, ou seja, em hardware e em software. Esta questão é extremamente importante em um projeto de sistema pois as decisões tomadas nesta etapa vão guiar o projeto até sua implementação final.

O particionamento hardware software é uma parte de um projeto de hardware/software co-design cujo objetivo principal é encontrar um design de implementação que preencha todos os requisitos especificados otimizando seu funcionamento e diminuindo ao máximo os custos do projeto.

Geralmente ao propor um design específico a um projeto, o projetista decide quais partes do projeto serão implementadas em hardware e quais partes do projeto serão implementadas em software através de seu conhecimento adquirido. Afim de automatizar este árduo processo, vários algoritmos e e técnicas vem sendo desenvolvidas e propostas em vários projetos de co-design. Estes modelos e abordagens funcionam muito bem nos ambientes de desenvolvimento par aos quais foram projetados, porém há tantas diferenças entre os modelos que não é possível comparar os resultados destas soluções.

Alguns aspectos são comuns e geralmente levados em conta na maioria dos ambientes de projeto que necessitem deste particionamento.

A especificação inicial do projeto é muito importante pois fixa o nível de abstração do projeto, o tipo de aplicação para a qual será voltado o projeto , o modelo computacional escolhido como representação e a granularidade escolhida.

O particionamento hardware/software é um problema NP-Difícil [1].

## 3. Materiais e Métodos

### 3.1. Ambiente de desenvolvimento

O ambiente utilizado para desenvolvimento do trabalho foi o sistema operacional *Microsoft Windows XP* com *service pack 3,* o ambiente de programação utilizado foi o *Microsoft Visual C++ 2008 express edition* com *service pack* 1. As execuções foram feitas em um processador *Intel Celeron M 1,50 GHz,* com *1.24 Gb* de memória *ram.*

### 3.2. Modelagem do Problema

O problema do particionamento hardware/software pode ser abordado de várias maneiras diferentes. Existem enumeras variáveis envolvidas em uma questão de projeto de sistemas embarcados como por exemplo tempo de resposta de hardware, custo do hardware, tempo de resposta de software, custo computacional, tamanho da placa onde será implementado o circuito, o tempo necessário para desenvolvimento de cada módulo em hardware e em software dentre outros.- No caso da geração de dados o algoritmo genético utilizado aborda o problema do particionamento da seguinte forma : partindo de um circuito real foi feito um grafo acíclico direcionado semelhante ao da Figura 1.

Este grafo acíclico direcionado possui as seguintes informações do problema : tempo de hardware, custo de hardware, tempo de software, custo de software e os custos de comunicação.

Limitante a todos estes custos relacionados está o tempo de resposta limite que é um dado que o projetista insere juntamente com os dados do grafo para que este particionamento possa ser calculado.

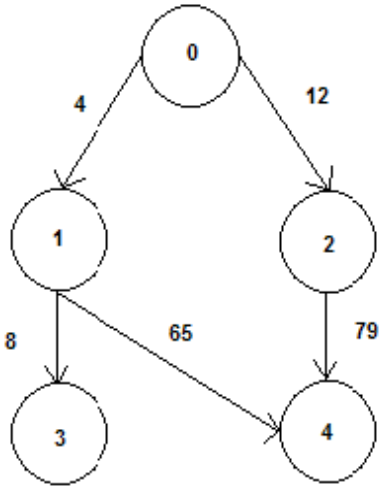


**Figura 1: Exemplo de grafo acíclico direcionado**

### 3.3. Geração do banco de dados

Segundo [5], em comparação com métodos *Simulated Annealing, Fiduccia* – Matheyses *e Modified Fiduccia- Matheyses* o algoritmo genético simples apresentou melhores resultados de particionamento em períodos de tempo de execução mais curtos. Devido a ampla utilização de algoritmos genéticos para resolução deste problema esta técnica foi escolhida para geração de dados como proposto por [5] devido aos resultados satisfatórios que apresentou.

$$f1 = \begin{cases} k1 \cdot (t - tr) + k2 \cdot c & if \quad t \geq tr \\ c & if \quad t < tr \end{cases}$$

**Figura 2: Função *fitness***

A função *fitness* utilizada no algoritmo foi a função da Figura 2.

- $t = \Sigma\, t\_hw * gene + \Sigma\, t\_sw * \neg\, gene$
- $c = \Sigma\, c\_hw * gene + \Sigma\, c\_sw * \neg\, gene + Com.Costs$

**Figura 3: Cálculo de parâmetros**

A função objetivo funciona da seguinte maneira : caso o tempo(t) for menor que o tempo limite(tr) estipulado pelo projetista, a função assume o valor dos custos, e caso for maior a função penaliza o particionamento com a constante K1 com valor 1000 e K2 com valor 1, inviabilizando a solução.

O tempo (t) é calculado como o somatório dos tempos de hardware dos nós que forem implementados em hardware somado ao somatório dos tempos de software dos nós implementados em software.

```
5
(custos) 1 4 2 12 0    //refernete a nó 0
(custos) 3 8 4 65 0    //refernete a nó 1
(custos) 4 79          //refernete a nó 2
(custos) 0             //refernete a nó 3
(custos) 0             //refernete a nó 4
35                     //Tempo limite
```

**Figura 4: Arquivo de entrada referente ao grafo da Figura 1.**

O custo (c) é calculado com o somatório dos custos de hardware dos nós implementados em hardware somados aos custos de software dos nós implementados em software e aos custos de comunicação. Os custos de comunicação são levados em conta apenas quando um dos nós é implementado de uma forma e o outro nó da comunicação é implementado de outra, não havendo custo caso os nós sejam implementados de mesma forma.O algoritmo genético funciona de seguinte maneira: o algoritmo recebe um arquivo com o grafo similar ao da Figura 4. Após armazenar o grafo, é criada uma população inicial. A população inicial é um conjunto de vetores binários de tamanho n, onde n é o numero de nós do grafo de entrada. Cada indivíduo sugere um particionamento onde o gene que possui valor 0 será implementado em software e o gene que possui valor 1 será implementado em hardware.

O algoritmo genético utilizado para geração dos dados funciona da seguinte forma: a população inicial de 13 indivíduos é gerada aleatoriamente, ou seja, são criados 13 vetores de bits e preenchidos com 0 e 1 aleatoriamente. Para que seja realizado o *crossover* dois indivíduos são selecionados por *cluster*. Os indivíduos da população são organizados em uma árvore ternária de acordo com o valor da função de aptidão. Cada população possui quatro clusters de quatro indivíduos cada que são organizados em dois níveis um *cluster* no primeiro e três no segundo. A situação é ilustrada pela Figura 5.

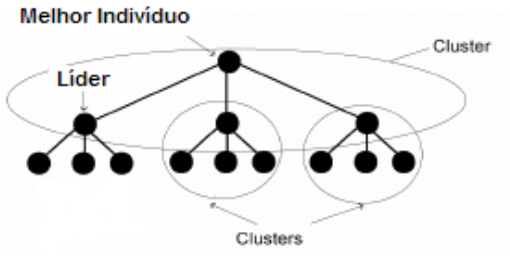


**Figura 5: Seleção por cluster**

A seleção dos pais para *crossover* começa com a seleção de um *cluster* aleatoriamente onde os pais serão o líder do *cluster* juntamente com um dos seguidores de forma aleatória. O ponto de corte do vetor que irá determinar qual parte dos indivíduos será cruzada é calculado aleatoriamente. Após o *crossover* o novo indivíduo gerado é inserido no *cluster* se ele for melhor que um de seus pais. A estrutura da população deve ser reorganizada a cada nova inserção da mesma forma que é feito o balanceamento de uma árvore.

A taxa de *crossover* utilizada é de 0.7, ou seja, de uma população inicial são gerados 70% de novos indivíduos fruto do *crossover*. A mutação é uniforme e ocorre percorrendo-se o vetor, e de acordo com a taxa, os bits são invertidos. A taxa de mutação utilizada é de 0,1 ou seja, 10% de chance de ocorrer mutação.

A geração dos grafos pode ser feita por qualquer programa que gere grafos de tarefas acíclicos orientados que possam ser limitados com relação aos parâmetros.

Após gerados os grafos os pares entrada/saída da rede, ou seja grafo/particionamento, são salvos em um arquivo no formato utilizado pela rede neural.

### 3.4. Rede Neural

Após várias pesquisas e análises a rede neural escolhida para ser utilizada foi a rede implementa pela biblioteca FANN (*Fast Artificial Neural Network)* .

A FANN é uma biblioteca livre e de código aberto para redes neurais artificiais que implementa redes de múltiplas camadas em linguagem de programação C e que permite a utilização de redes densas e esparsas. Plataformas de execução cruzadas com valores inteiros e de ponto flutuante também são implementadas. Em [11] pode ser encontrada a documentação referentes às enumeras funções para criação, execução, controle de parâmetros, treinamento dentre outras que a biblioteca possui.

Para criar a rede foi utilizada a função *fann_create_standard* .A função de ativação das camadas escondidas e da camada de saída foram configuradas respectivamente com as funções *fann_set_activation_function_hidden* e *fann_set_activation_function_output.* A rede foi treinada com um arquivo de entrada no formato aceito pela rede e utiliza a função *fann_train_on_file* .

O treinamento e a execução da rede foram feitos em programas separados como recomendado pelo manual de referência da biblioteca. Após o algoritmo de treinamento com os dados ser executado para um dado arquivo de entrada, a configuração da rede neural é armazenada em um arquivo com a função *fann_save*. O programa que executa a rede cria a rede a partir deste arquivo com a função *fann_create_from_file* e utiliza com a função *fann_run*. As informações de como utilizar estas funções e outras funções existentes na biblioteca estão no manual de referencia que pode ser encontrado em [12], e também nos tutoriais que estão no mesmo endereço.

A rede neural foi modelada da seguinte maneira: como cada nó possui quatro informações associadas além dos custos de comunicação, portanto levando em conta os vários tipos de grafos possíveis com sete nós, várias formas de associação destes nós e vários tempos limite para os projetos, a rede terá 71 entradas. Destas 71 entradas cada grupo de 10 entradas serão equivalentes a um nó. O primeiro neurônio será o tempo limite do projeto. Cada nó será representado por uma seqüência de tempo de hardware, custo de hardware, tempo de software, custo de software. Os outros seis neurônios representarão o custo de comunicação do nó em questão com os nós restantes sendo este numero suficiente devido a característica acíclica direcionada do grafo. Os custos de comunicação inexistentes irão ser colocados como 0.

Para a solução do problema do particionamento foi escolhida a rede densa de perceptrons de múltiplas camadas, no caso duas, *feedforward* e com a função de ativação de neurônios, tanto das camadas escondidas como da camada de saída, sigmoidal simétrica. O algoritmo de treinamento utilizado é o Rprop, sem momento com tamanho do passo 0.5, o número máximo de épocas

de treinamento como 15 mil. As camadas escondidas possuem 30 neurônios cada.

## 4. Resultados e Discussão

O algoritmo genético utilizado para geração do banco de dados apresentou resultados de particionamento melhores que os resultados apresentados em [5], para o arquivo de entrada referente ao grafo da Figura 6, como pode ser visto nas tabelas 1 e 2 e na Figura 7. O arquivo de entrada possuí a primeira linha com o número de nós do grafo seguido de linhas em seqüência de acordo com os nós ( linha 1 nó 0, linha 2 nó 1) com informação de tempo de hardware, custo de hardware, tempo de software e as comunicações representadas pelo primeiro número como o nó que recebe a comunicação e pelo segundo número como o custo da comunicação. O custo de software neste arquivo foi omitido pois segundo [5], a implementação do circuito representado por este grafo considera que sempre haverá memória disponível para a execução dos algoritmos.

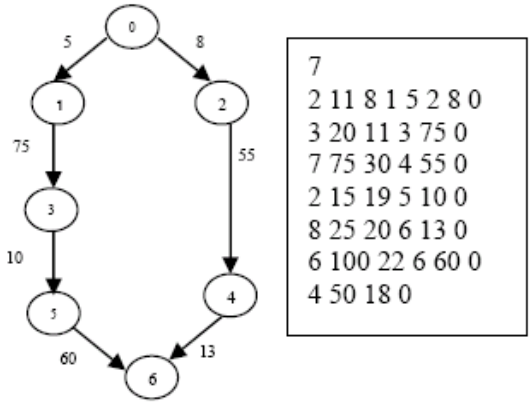


**Figura 6: GAO e Arquivo de entrada**

Na Tabela 1 a primeira coluna mostra o tempo limite de projeto que foi utilizado, seguido pelos custos do particionamento encontrado pelo algoritmo *Fidducia-Mtheyses* (Cfm)*, custos dos particionamentos encontrado pelo algoritmo simulated annealing* (Csa)*, c*ustos dos particionamentos encontrado pelo algoritmo *Modified Fidducia-Matheyses* (Cmf) *e* por fim os custos dos particionamentos encontrados pelo algoritmo genético (Cag) utilizado por [5] seguido dos indivíduos encontrados pelo algoritmo genético em cada caso.

Após gerado o banco de dados, o arquivo foi utilizado para treinamento da rede neural.

Tabela 1: Resultados do AG [5]

| Tr | Cfm | Csa | Cmf | Cag | Individual |
|---|---|---|---|---|---|
| **35** | 296 | 296 | 296 | 296 | 1111111 |
| **40** | 296 | 296 | 296 | 296 | 1111111 |
| **45** | 296 | 296 | 296 | 296 | 1111111 |
| **50** | 296 | 296 | 296 | 266 | 1111101 |
| **55** | 285 | 285 | 285 | 266 | 1111101 |
| **60** | 285 | 285 | 285 | 256 | 1111101 |
| **65** | 285 | 285 | 285 | 169 | 1111100 |
| **70** | 285 | 285 | 285 | 169 | 1111100 |
| **75** | 261 | 261 | 261 | 169 | 1111100 |
| **80** | 261 | 261 | 261 | 169 | 1111100 |
| **85** | 250 | 261 | 146 | 157 | 1101100 |
| **90** | 196 | 250 | 146 | 129 | 1010100 |
| **95** | 146 | 146 | 146 | 121 | 0010100 |
| **100** | 146 | 146 | 146 | 64 | 1101000 |
| **105** | 111 | 135 | 111 | 50 | 0101000 |
| **110** | 111 | 135 | 111 | 50 | 0101000 |
| **115** | 100 | 100 | 46 | 50 | 0101000 |
| **120** | 100 | 100 | 35 | 50 | 0101000 |
| **125** | 100 | 150 | 35 | 24 | 1000000 |
| **130** | 0 | 150 | 0 | 0 | 0000000 |

A evolução do erro quadrático com relação ao número de épocas no treinamento da rede que apresentou melhores resultados iniciou-se com 0.87 e depois oscilou ente 0,20 e 1,8 atingindo seu mínimo em 0,158.. Os parâmetros utilizados por esta rede são : 30 neurônios nas duas camadas escondidas, passo de treinamento 0.5, momento 0, função de ativação sigmoidal simétrica nas entradas e na saída da rede. O treinamento da rede foi feito em aproximadamente duas horas e o resultado obtido para o grafo de entrada da Figura 6 gerou os resultados da Figura 8. O espaço do particionamento gerado pela rede neural não pôde ser desenhado devido ao número elevado de dimensões.

As figuras 7 e 8 mostram comparações de resultados. O algoritmo genético apresentou melhores resultados com relação aos custos do particionamento do que os resultados apresentados pela rede neural. Os dados de entrada não estavam no conjunto de treinamento, porém a rede conseguiu atingir o objetivo de apresentar particionamentos com custos interessantes.

A algoritmo genético utilizado para geração do banco de dados apresentou melhores resultados em com-

paração ao algoritmo de [5] como mostrado nas tabelas no início do capítulo. Os resultados garantiram a geração

Tabela 2: Comparação entre AG's

| TR | Custo AGH | Custo AGBD |
|---|---|---|
| 35 | 296 | 296 |
| 40 | 296 | 291 |
| 45 | 296 | 291 |
| 50 | 266 | 266 |
| 55 | 266 | 266 |
| 60 | 266 | 261 |
| 65 | 169 | 169 |
| 70 | 169 | 164 |
| 75 | 169 | 164 |
| 80 | 169 | 154 |
| 85 | 157 | 154 |
| 90 | 129 | 129 |
| 95 | 121 | 121 |
| 100 | 64 | 64 |
| 105 | 50 | 50 |
| 110 | 50 | 50 |
| 115 | 50 | 35 |
| 120 | 50 | 35 |
| 125 | 24 | 24 |
| 130 | 0 | 0 |

de um banco de dados com particionamentos melhores em comparação ao banco de dados que seria gerado caso o algoritmo de geração fosse o algoritmo genético simples.

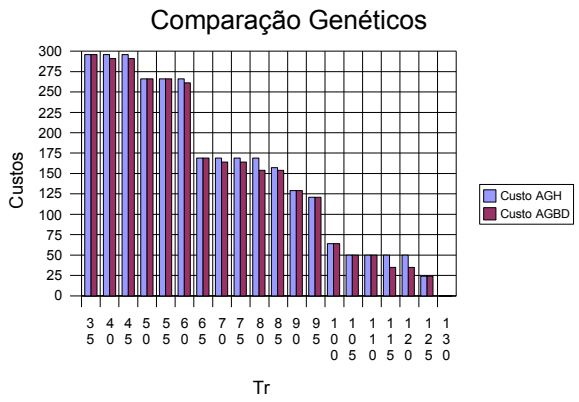


**Figura 7: Comparação de custos entre AG's**

A técnica utilizada para a resolução do particionamento é justificável pelo fato da rede neural poder apresentar resultados melhores do que algoritmos que levam em conta uma função para avaliação pois percorrem uma parte maior do espaço de busca. O treinamento da rede, mesmo com um banco de dados cujos pares entrada/saída sejam gerados por um algoritmo genético que leve em conta uma função de ativação, não resulta em uma rede que siga perfeitamente a função de ativação.

Uma rede neural pode ser treinada várias vezes fazendo com que sua aplicabilidade seja interessante para projetos onde o particionamento possua características próprias da empresa ou instituição que os desenvolve, pois a rede pode adaptar-se aos padrões de projeto com um novo treinamento onde este projetos estejam presentes no conjunto de dados de treinamento.

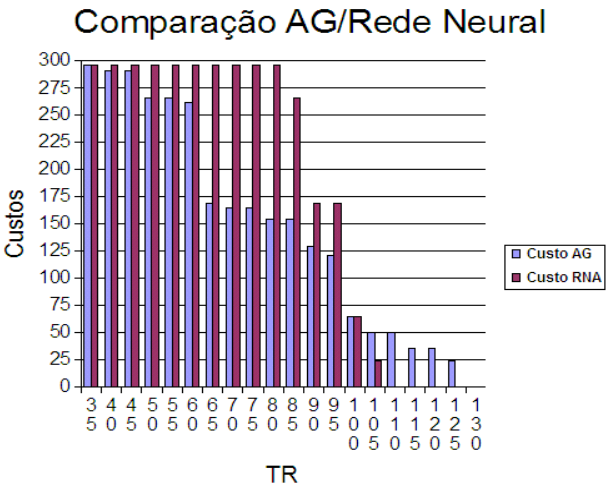


**Figura 8: Comparação AG e rede neural**

Este problema tem sido amplamente atacado com técnicas heurísticas conhecidas como busca TABU, algoritmos genéticos, *simulated annealing* ou até algoritmos desenvolvidos exclusivamente para a resolução do problema, porém devido à falta de um banco de dados ou de uma metodologia para criação de bancos de dados consistentes, as redes neurais não têm sido uma opção constante nas tentativas de resolução dos problemas.

A rede neural não apresentou melhoras com relação ao erro quadrático mínimo quando momento foi adicionado ao treinamento. O conjunto dados gerado possuí 10 mil entradas, porém melhores resultados foram obtidos com uma margem de 5 mil dados.

Os resultados obtidos pela rede neural geralmente não são exatamente iguais aos resultados esperados ou contidos no banco de dados utilizados para treinamento, porém este fato indica que ela poderá apresentar particionamentos com custos aceitáveis para as entradas esco-

lhidas para a modelagem que também não estejam presentes no treinamento.

O banco de dados gerado apresentou alguma peculiaridades que devem ser ressaltadas para que os resultados da rede neural sejam melhor interpretados.

Quando submetido a análise o banco de dados apresentou uma grande quantidade de particionamentos cujo resultado apresentou todos os nós do grafo em hardware ou todos os nós do grafo em software. Como esta quantidade de dados era muito significativa no conjunto de treinamento o resultado apresentado da rede estava tendencioso a este tipo de particionamento o que justifica o alto custo das soluções encontradas.

A rede neural apresenta seus resultados em números com notação de ponto flutuante que necessitam de um arredondamento para serem analisados de forma mais consistente. Ao analisar os dados sem o arredondamento foi notado que , apesar dos resultados da rede apresentarem-se em maior quantidade particionamentos completamente em hardware ou software, os bits que representavam nós que deveriam estar implementados em outra forma segundo resultado do algoritmo genético sofriam realmente uma modificação que não apresentava-se significativa para o arredondamento porém era numéricamente perceptível. O significado desta observação é o seguinte : arredondando-se o particionamento para um TR de 40 ou 45 por exemplo, ao invés de apresentar uma saída toda em hardware ( 1 1 1 1 1 1 1), o problema deveria apresentar uma saída com o segundo nó em software (1 0 1 1 1 1 1), porém quando analisados os valores reais da rede neural, é notável que o segundo bit deste conjunto de bits que deveria estar em 0 apresentou a maior queda de valores entre todos os outros bits, ou seja, o bit que deveria estar em software mudou seu valor de 0.99678 para 0.84560 enquanto os outros bits permaneceram no patamar de 0.99 – 0.95.

Esta análise mostra que a rede apresenta sim uma boa capacidade de aprendizado e generalização e teve as respostas prejudicas pelo fato do banco de dados apresentar exemplos de dois tipos em quantidade notadamente maior.

Este trabalho apresenta um grande número de possibilidades de trabalhos futuros. Estes trabalhos devem levar em conta dois pontos básicos : melhorias na geração do banco de dados e melhorias na arquitetura e topologia da rede neural escolhida para o particionamento.

Com relação à geração do banco de dados, a função de avaliação do algoritmo genético utilizado para geração pode ser remodelada para que os cálculos dos parâmetros apresentem melhores resultados e também para que possa levar em conta um número maior de variáveis de entrada.

Outro ponto importante na geração do banco de dados é a utilização do máximo possível de grafos de entrada que representem grafos de projetos reais já implementados. Caso não seja possível reunir este banco de dados, a utilização de um programa de geração de grafos como desenvolvido por [4] pode ser avaliado.

A rede neural também pode ser modificada para que grafos de tamanhos maiores possam ser utilizados como entrada da rede. Para isso o número de entradas da rede deve ser alterado juntamente com a modelagem utilizada para a rede tanto no caso da entrada de dados, como no caso da saída de dados.

Um estudo sobre topologias e algoritmos de treinamento para auxiliar a escolha da rede que apresente melhores resultados para um dado conjunto de entradas também é uma sugestão de trabalhos futuros.

## 5. Conclusão

Durante o desenvolvimento do trabalho, foi possível perceber que a rede neural realmente possui uma característica de generalização que pode ser útil no desenvolvimento do problema.

Os grafos gerados pelo algoritmo de geração são realmente grafos válidos influenciando de forma correta no treinamento da rede neural proporcionando resultados de particionamento que condizem com os dados de entrada apresentados.

O resultado apresentado pelo algoritmo genético na geração do banco de dados mostrou que a técnica utilizada para seleção realmente acelerou o processo de busca de melhor solução no caso do problema do particionamento hardware/software. Este algoritmo pode ser utilizado para resolução de outros problemas de particionamento que apresentarem características de modelagem semelhantes.

Algumas dificuldades foram encontradas principalmente na geração do banco de dados. As informações de circuitos e de projetos com os dados necessários são escassas dificultando assim a utilização de heurísticas como as redes neurais. O banco de dados se mostrou

uma peça chave para o desenvolvimento do trabalho, necessitando de uma atenção especial em sua geração.

O resultado do trabalho mostra também que todos os tipos de projetos de sistemas embarcados que seguem um padrão podem ser utilizados no treinamento da rede neural para que o resultado da rede possua as características inerentes a estes projetos.

A resposta da rede neural, para uma dada entrada no formato escolhido para modelagem do problema, se dá de forma instantânea  o que representa um diferencial com relação a outros tipos de heurísticas utilizadas que necessitam de  um tempo de resposta mínimo para apresentarem a melhor solução. A rede neural utilizada apresenta características simples, diminuindo de forma considerável o tempo de treinamento. Estima-se que a complexidade da rede deve ser aumentada de acordo com o número de nós que forem utilizados para a modelagem do problema. Redes com complexidade maior não obtiveram resultados satisfatórios para a solução do problema em questão.

## 6. Referências Bibliográficas


[1] ARATÓ, P.; JUHÀSZ, S.;MANN, Z. Á.; ÓRBAN, A.; PAPP, D.. Budapest University oc Technology and economics. **Hardware-software partitioning in embedded system design**, , v. 1, n. 1, p. 1 - 6,  2004.

[2]BRAGA, A. P.;CARVALHO, A.P.L.;LUDERMIR,T.B.. **Redes Neurais Artificiais Teoria e Aplicação**. Segunda Edição. : LTC, 2007. 226 p.

[3]DE MICHELI, G.; GUPTA, R. K.. Hadrware/Software Co-Design. **Proceedings of the IEEE**, , v. 85, n. 3, p. 349 - 365,  1997.

[4]DICK, R.P.; RHODES, D.L.; WOLF, W.. TGFF: task graphs for free. **Proceedings of the Sixth International Workshop on Hardware/Software Codesign, 1998.** , , v. 15, n. , p. 97 - 101,  1998.

[5]HIDALGO, J.I.; LANCHARES, J.. Functional partitioning for hardware-software codesign using genetic algorithms. **Proceedings of the 23rd EUROMICRO Conference**, , v. 1, n. , p. 631 - 638,  1997.**tion to Neural Networks**. Oitava Edição. : The University of Amsterdan, 1996. 134 p.

[6]LÓPEZ-VALLEJO, M. L.;LÓPEZ,J.C.;IGLESIAS, C. A.. Hardware-Software Partitioning at the Knowledge Level. **Applied Intelligence**, , v. 10, n. 1, p. 173-184,  1999.

[7]LÓPEZ-VALLEJO, M.; LÓPEZ, J. C.. On the Hardware-Software Partitioning Problem: System Modeling and Partitioning Techniques. **ACM Transactions on Design Automation of Electronic Systems**, , v. 8, n. 3, p. 269-297,  2003.

[8]SIPSER, M.. **Introduction to The Theory of Computation**. Segunda Edição. : Thomson, 2006. 453 p.

[9]STRAUNSTRUP, J.; WOLF, W.. **Hardware Software Codesign - Principles and Practice**. Primeira Edição. : Springer, 1997. 416 p.

[10]WILMSHURST,T.. **Designing Embedded Systems with PIC Microcontrolers**. Primeira Edição. : Elsevier, 2007. 583 p.

[11]Fast Artificial Neural Network Library Reference Manual. DCS, University of Copenhagen. Desenvolvido por: Nissen, S., 2005. . Disponível em: <http://leenissen.dk/fann/html/files/fann-h.html>. Acesso em: 27/10/2008.

[12]Fast Artificial Neural Network Library. DCS, University of Copenhagen. Desenvolvido por: Nissen, S., 2005. . Disponível em: <http://leenissen.dk/fann/>. Acesso em: 27/10/2008.