MAURÍCIO ACCONCIA DIAS

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

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

LAVRAS
MINAS GERAIS – BRASIL
2008

MAURÍCIO ACCONCIA DIAS

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

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

Área de Concentração:

Inteligência Computacional

Orientador(a) :

Prof. Dr. Wilian Soares Lacerda

LAVRAS
MINAS GERAIS – BRASIL
2008

Ficha Catalográfica

Dias, Maurício Acconcia

PARTICIONAMENTO HARDWARE/SOFTWARE: PROPOSTA DE SOLUÇÃO COM REDES NEURAIS ARTIFICIAIS / Maurício Acconcia Dias. Lavras – Minas Gerais, 2008. 71p : il.

Monografia de Graduação – Universidade Federal de Lavras. Departamento de Ciência da Computação.

1.Introdução. 2.Computação Evolutiva. 3.Redes Neurais Artificiais. 4.Teoria da Complexidade Computacional. 5.Hardware/Software Co-design. 6.Particionamento Hardware/Software. 7.Materiais e Métodos. 8.Resultados e Discussão. 9.Conclusão I. DIAS,M.A. II. Universidade Federal de Lavras. III. Título.

MAURÍCIO ACCONCIA DIAS

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

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

Aprovada em 21/11/2008

_____
Prof. Dr. André Vital Saúde

_____
Prof. Dr. João Carlos Giacomin

_____
Prof. Dr. Wilian Soares Lacerda
(Orientador)

LAVRAS
MINAS GERAIS – BRASIL
2008

Dedico este trabalho de pesquisa a todas as pessoas que estiveram envolvidas a ele, ou seja, a todos os meus amigos, colegas de república, familiares e principalmente a meus pais.

Agradecimentos

Agradeço a meus pais pelo apoio emocional nos dias que seguiram este trabalho, pela compreensão com relação à falta de tempo e de atenção,

aos meus amigos pela paciência com relação às atitudes e às dificuldades que foram superadas com a ajuda deles, principalmente ao Bruno pelo apoio incondicional, ao Lucas pela extrema paciência dedicada à ajuda ao meu trabalho e ao Daniel que nunca se queixou de escutar as reclamações e conselhos que nem sempre eram bem vindos,

ao meu orientador, Wilian, por todos os conselhos, advertências e cobranças que ajudaram a me formar como pessoa e não só como profissional,

também aos professores que de uma forma ou de outra foram responsáveis por toda a base de conhecimento adquirida até este ponto,

e , por fim, a todos os funcionários dos departamentos utilizados por mim na graduação, principalmente no DCC , a Ângela, ao Deivson e ao Clayton que foram sempre muito prestativos e eficientes sabendo compreender os problemas e ajudar da melhor forma possível.

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

RESUMO

O particionamento hardware/software é um dos problemas mais importantes em projetos de desenvolvimento que possuem um hardware/software co-design. Os problemas de particionamento estão no grupo de problemas NP-Completos, caso seja adotado um caráter de otimização ao problema 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 artificiais para atacar o problema. O resultado deste projeto de pesquisa é uma rede neural artificial que resolve o problema do particionamento para uma dada instância apresentando um bom resultado em um curto tempo de execução.

Palavras-chave: Hardware/Software Co-design, particionamento Hardware/Software, Redes Neurais Artificiais.

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 Co-design, Hardware/Software Partitioning, Artificial Neural Networks.*

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1 INTRODUÇÃO

1.1 Contextualização e Motivação

Segundo Wilmshurst(2007), 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 design de sistemas embarcados modernos deve priorizar flexibilidade, variabilidade, aplicabilidade e utilizando microprocessadores, dispositivos de hardware reprogramável como *FPGA's*, sistemas desenvolvidos em chips (*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 De Micheli(1997), hardware/software co-design significa reunir objetivos como um sistema explorando o sinergismo entre hardware e software em seus próprios designs.

O recente aumento do interesse na área de hardware/software co-design se deve ao aparecimento de várias ferramentas (CAD) para auxiliar no desenvolvimento, além da alta demanda por sistemas embarcados no mercado atual, que tem crescido a uma taxa de 34%a.a.. O desenvolvimento da tecnologia na área de circuitos integrados, diminuindo o tamanho dos circuitos e aumentando sua capacidade computacional, tem atraído muitos profissionais para a área, devido ao campo de atuação que está sendo proporcionado. O investimento na área está fragmentado devido aos problemas encontrados no design de sistemas.

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 está 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 influenciá-lo até o final de sua execução impactando diretamente na performance, na dificuldade de implementação e no custo final do projeto (López-Vallejo & Lopez, 2003).

Segundo Sipser(2006), 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 Arató et. al. (2004) 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 Braga, Carvalho & Ludermir(2007) as redes neurais artificiais são formas de computação não-algorítmica que caracteriza-se por sistemas que em algum nível relembram a estrutura do cérebro humano .

As redes neurais têm se mostrado soluções viáveis 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 em sua utilização é o tempo de treinamento pois a resposta a um problema por uma rede treinada é praticamente instantânea.

1.2 Objetivos do Trabalho

Como objetivo principal deste trabalho tem-se o desenvolvimento e treinamento de uma rede neural artificial para resolução do problema do particionamento hardware/software. Os objetivos secundários são o estudo do problema do particionamento, criação de uma rede neural em linguagem de programação C++ com algoritmo de treinamento ou estudo e utilização de uma biblioteca já existente, estudo detalhado de redes neurais para que a escolha certa da rede seja feita. Também como objetivo secundário há a criação de um banco de dados para treinamento da rede caso o mesmo não seja encontrado em alguma das referencias utilizadas neste trabalho.

O problema do particionamento hardware/software pode ser definido de várias maneiras de acordo com cada projeto de hardware/software co-design. O particionamento ótimo visa tanto economia de custo de implementação e energético quanto o desempenho do produto final do projeto.

O enfoque principal deste trabalho é o desenvolvimento de sistemas embarcados e principalmente na questão de fixação de uma modelagem que seja abrangente e que apresente resultados rápidos e concisos.

1.3 Organização do Documento

Os capítulos deste documento científico estão organizados da seguinte forma: o capítulo dois contém a teoria sobre computação evolutiva básica seguida dos algoritmos genéticos, programação genética , estratégias evolutivas e por fim a programação evolutiva.

O capitulo três apresenta o histórico de redes neurais artificiais, uma definição, vários modelos de neurônios artificiais, arquiteturas de redes neurais, algoritmos de treinamento e por fim algumas aplicações de redes neurais artificiais.

O capítulo quatro apresenta conceitos de teoria da complexidade computacional, definindo esta teoria, explorando complexidade de tempo, a classe de problemas P, a classe de problemas NP, a questão da redutibilidade, NP-Completude e problemas NP-Difíceis. Este capítulo da suporte à definição do problema apresentado no capítulo seis.

O capítulo cinco apresenta o hardware/software co-design iniciando com uma pequena introdução, seguida da definição, da apresentação das questões envolvidas em qualquer tipo de

design em especial o co-design, introdução ao co-design de sistemas embarcados, métodos propostos para o hardware/software co-design e por fim aprofundamento em classificação e técnicas para o co-design de sistemas embarcados.

O capítulo seis apresenta o problema do particionamento hardware software, definido, em seguida é apresentado um histórico das tentativas de resolução do problema, um modelo de particionamento em sistemas embarcados, seguido da definição formal e classificação do problema.

A parte de materiais e métodos, sétimo capítulo, apresenta o tipo de pesquisa em que o trabalho se encaixa segundo Jung(2004), os procedimentos metodológicos que foram seguidos no trabalho, juntamente com as ferramentas utilizadas para estes procedimentos encerrando-se com o ambiente em que foi desenvolvido o trabalho.

O capítulo oito apresenta os resultados atingidos durante o desenvolvimento do projeto em ordem cronológica, seguidos da discussão sobre os mesmos e propostas de trabalhos futuros, baseados no trabalho desenvolvido.

O capitulo nove contém as conclusões do trabalho.

2 COMPUTAÇÃO EVOLUTIVA

2.1 Introdução

Os princípios da evolução natural têm incentivado um grande número de pesquisadores a desenvolverem esta área de pesquisa. O princípio da "sobrevivência do mais forte" proposto por Charles Darwin instiga a mente dos pesquisadores.

Esta teoria da seleção natural propõe que os organismos que existem hoje em dia são resultado de milhões de anos de adaptações às demandas do ambiente em que viviam. Vários tipos de organismos viviam no mesmo ecossistema competindo por alimentos, locais adequados para que pudessem procriar e perpetuar a espécie. Os organismos de espécies que não eram capazes de se sobressair nestas condições eram extintos. Os organismos que conseguiram sobreviver foram considerados mais aptos por possuírem características que se sobressaíram às características de outros indivíduos sendo portanto selecionados pelo meio.

As técnicas de computação evolutiva abstraem os princípios evolucionários para algoritmos que são utilizados para encontrar soluções para problemas, substituindo a busca exaustiva, que torna-se inviável, para um espaço de busca muito grande. Segundo Hussain(1998), a diferença do algoritmo em questão para outros algoritmos heurísticos é a eficiência da busca direcionada.

2.2 Computação Evolutiva Básica

Para um algoritmo evolutivo, é escolhido um esquema de representação pelo pesquisador para definir o conjunto de soluções que irão formar o espaço de busca à ser explorado pelo algoritmo. Um número inicial de indivíduos é gerado aleatoriamente e chamado de população inicial. Após os parâmetros iniciais, a seqüência seguinte de passos é executada até que alguma condição de término seja satisfeita. Cada indivíduo é avaliado por uma função de avaliação que é específica do problema que está sendo resolvido. Após a avaliação alguns indivíduos são escolhidos para serem pais que irão gerar os novos indivíduos. A nova geração de descendentes é produzida por operadores de reprodução aplicados aos indivíduos escolhidos como pais. Os melhores indivíduos da geração descendente são escolhidos e todo o processo é repetido. A Figura 2.1 ilustra, segundo Back & Schwefel(1996) um pseudocódigo que resume os principais componentes de um algoritmo evolutivo.

```
t = 0;
inicializa P(t);
avalia P(t);
enquanto não terminar faça:
   P'(t) = crossover P(t);
   P''(t) = mutação P'(t);
   avalia P''(t);
   P(t+1) = seleção (P''(t) U Q);
   t = t+1;
fim
```

Figura 2.1: Algoritmo evolutivo básico

Neste algoritmo o tempo é iniciado e uma população inicial *(P(t))* é criada no tempo *t*. Esta população, depois de avaliada, entra em um laço que termina de acordo com o programador, podendo ser através do tempo de execução ou pelo número de iterações, por exemplo. O laço é responsável por executar o cruzamento e a mutação na população seguido da avaliação. Os indivíduos são selecionados e a população seguinte *(P(t+1))* é composta pelos indivíduos novos *(P''(t))* somados aos indivíduos antigos que foram selecionados para permanecerem por apresentarem características relevantes *Q*.

Os mecanismos que especificam como e quais indivíduos serão selecionados, quando descendentes serão gerados e quantos irão sobreviver para a próxima geração, são chamados de métodos de seleção. Existem vários tipo de métodos de seleção de diferentes tipos de complexidade, que geralmente garantem uma população homogênea em tamanho em todas as gerações.

2.3 Algoritmos Genéticos

A técnica mais popular em computação evolutiva são os algoritmos genéticos. O algoritmo genético tradicional utiliza como representação um vetor de tamanho fixo de bits. Cada posição do vetor é um gene e representa uma característica individual e o valor armazenado nesta posição representa como esta característica influi na solução.

Geralmente este vetor é avaliado como uma coleção de caraterísticas estruturais de uma solução que possuem pouca ou nenhuma interação.

O operador principal de reprodução é o cruzamento entre bits, onde dois vetores são utilizados como pais e novos indivíduos são formados pela troca de sub-estruturas entre eles. A

Figura 3.1 mostra um exemplo onde o final do vetor de bits que representa os indivíduos é trocado.

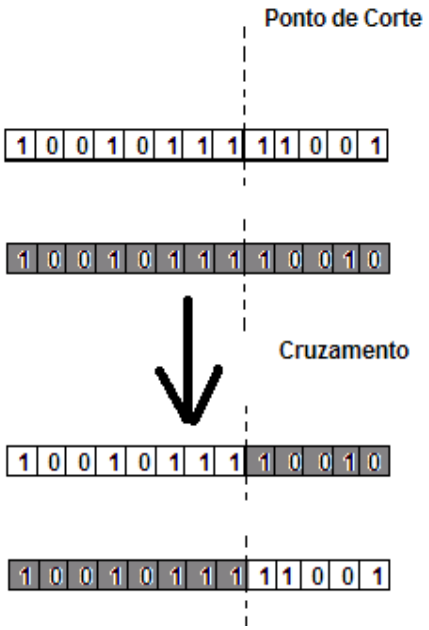

Figura 2.2: Exemplo de cruzamento

Um operador também muito utilizado é o operador de mutação. Este operador faz com que um ou mais bits do vetor que representa o indivíduo, que são escolhidos por sorteio ou por alguma técnica que envolva probabilidades por exemplo, sejam invertidos gerando assim um novo indivíduo. A Figura 2.3 ilustra a situação.

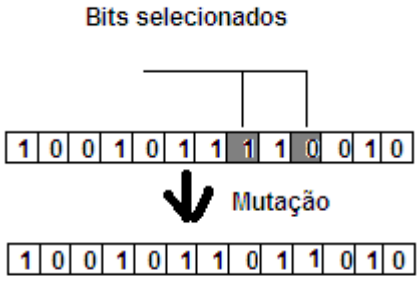

Figura 2.3: Exemplo de mutação

Existem outros operadores que foram desenvolvidos, porém utilizados com menos freqüência. Uma importante consideração sobre os operadores é a necessidade de verificação da existência de uma mudança em sua utilização. Além disso o operador também deve manipular o indivíduo de maneira consistente preocupando-se em manter o novo indivíduo no espaço de busca.

Tradicionalmente os indivíduos que são escolhidos para serem os pais são escolhidos probabilisticamente com base no resultado da função de avaliação para o dado indivíduo.

2.4 Programação Genética

A programação genética é uma técnica de programação evolutiva. Na programação genética a representação utilizada é uma árvore de tamanho variável de funções e valores. As folhas da árvore são valores escolhidos dentre vários dentro de um conjunto. Os nós internos da árvore são funções escolhidas também em um conjunto de funções válidas. A árvore toda representa uma função que será avaliada. As folhas são avaliadas pelos valores e as funções são avaliadas utilizando como argumento o resultado da avaliação dos filhos.

Programação genética e algoritmos genéticos são similares em muitos aspectos exceto pelo fato de que os operadores de reprodução são convertidos a uma representação em árvore no caso da programação genética. O cruzamento utilizado com mais freqüência é o que faz a troca de uma sub-árvore entre os pais.

Na programação genética as funções geralmente retornam o mesmo tipo e diferem no número de argumentos, isto permite que seja feita a troca de sub-árvores e garante que o indivíduo formado pelo cruzamento é válido.

2.5 Estratégias Evolutivas

No caso de estratégias evolutivas, a representação é feita através de um vetor de números reais com tamanho fixo onde cada posição corresponde a uma característica do indivíduo.

O principal operador neste caso é a mutação gaussiana na qual um valor randômico de uma distribuição gaussiana é adicionado a cada elemento do vetor que representa o indivíduo para criar um descendente. Outro operador é utilizado para recombinação intermediária onde é feita a média entre os valores dos vetores para a geração do indivíduo. Os dois métodos podem ser vistos respetivamente em a) e b) da Figura 2.4.

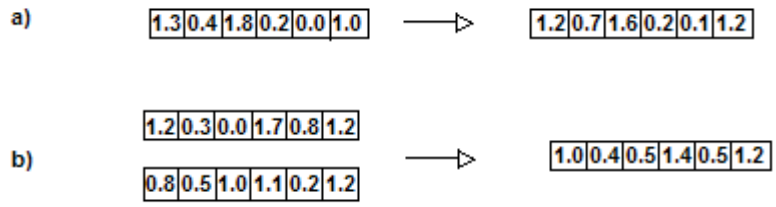

Figura 2.4: Exemplos de a)mutação gaussiana e b) recombinação intermediária

Em uma estratégia evolutiva típica *N*, pais são selecionado uniformemente e aleatoriamente. Um número maior que *N* de descendentes é gerado através da recombinação sendo selecionados os sobreviventes deterministicamente.

2.6 Programação Evolutiva

A representação utilizada em programação evolutiva é adaptada ao tipo de problema em questão sendo normalmente um vetor de tamanho fixo com valores reais.

A diferença básica entre este método e os métodos apresentados acima é o fato da utilização de mutação como único operador, não havendo troca de material genético entre indivíduos. Este método é similar às estratégias evolutivas porém com ausência de recombinação.

Um método comum de seleção é utilizado considerando-se todos os indivíduos da população como pais. Aplica-se a mutação em todos os indivíduos gerando o mesmo número de descendentes e, após este procedimento, seleciona-se dentre todos os indivíduos metade deles cuja função de avaliação retornou os valores mais altos.

Neste capítulo foram apresentadas as técnicas mais comuns de programação evolutiva e suas principais características auxiliando no entendimento da escolha pelos algoritmos genéticos no desenvolvimento deste trabalho de pesquisa.

3 REDES NEURAIS ARTIFICIAIS

3.1 Histórico

As redes neurais artificiais tiveram seu início com o trabalho do psicólogo e neurologista Warren McCulloch juntamente com o matemático Walter Pitts. O trabalho publicado por eles chamado "A Logical Calculus of The Ideas Immament in Nervous Activity" concentrou-se na descrição de um neurônio artificial e apresentação de suas capacidades computacionais além de uma discussão sobre redes lógicas, neurônios artificiais, idéias sobre máquina de estados finitos, elementos de decisão limiar lineares e representação lógica de várias formas de comportamento de memória. O neurônio apresentado é um neurônio simplificado, chamado de MCP (devido as iniciais de seus nomes), resultado da modelagem computacional do neurônio biológico juntamente com a idéia de um componente conceitual para circuitos que realizam tarefas computacionais.

Como no cérebro humano, os neurônios artificiais formam redes neurais que precisam de um aprendizado. Este assunto foi tratado em 1949 por Donald Hebb cujo trabalho mostrou que a aprendizagem de uma rede neural é conseguida através da variação dos pesos dados às entradas dos neurônios. A regra de Hebb , teoria proposta para explicar o aprendizado dos neurônios biológicos, foi interpretada e é utilizada até hoje para o treinamento de redes neurais artificiais. Em 1960, Window e Hoff propuseram a regra delta para treinamento que também é utilizada até hoje.

Frank Rosenblatt, em 1958, propôs um novo modelo de neurônio, o perceptron. Neste modelo o autor demonstra que, se forem adicionadas sinapses ajustáveis aos neurônios MCP de uma rede de vários neurônios, esta rede poderia ser treinada e utilizadas para reconhecimento de padrões.

Em 1969 Minsky e Papert publicaram o livro Perceptrons, e nele mostraram as deficiências do neurônio proposto por Rosenblatt , ou seja, que ele apenas classificava dados linearmente separáveis, não podendo assim identificar paridade, conectividade, simetria, considerados problemas difíceis de aprender por serem não-linearmente separáveis.

Devido à repercussão deste trabalho, as pesquisas na área de redes neurais artificiais declinou, pois as verbas destinadas aos projetos foram desviadas a outros projetos de pesquisa

em outras áreas. Apenas alguns pesquisadores continuaram seus trabalhos na área, dentre eles Tuevo Kohonen, Steven Grossberg, Kunihiko Fukushima, Igor Aleksander.

O interesse em redes neurais artificiais ressurgiu por volta de 1980 quando Hopfield publicou um artigo ressaltando a capacidade associativa das redes. Este trabalho mostrou a relação entre redes recorrentes auto-associativas e sistemas físicos. Aliado a este fato o algoritmo de treinamento *backpropagation* apresentado em 1986 mostrou que que Minsky e Papert estavam subestimando a capacidade do modelo de Rosenblatt.

O desenvolvimento da tecnologia na área de hardware juntamente com a dificuldade na resolução de problemas com as técnicas inteligência computacional conhecida até este momento histórico ,fizeram com que a área das redes neurais alavancasse novamente.

Atualmente a área concentra-se no desenvolvimento de algoritmos de treinamento mais rápidos baseados no algoritmo *backpropagation* juntamente com o avanço do hardware. As pesquisas da área de hardware neural também tem avançado devido ao interesse recente pelos modelos neurais com maior apelo fisiológico. Os novos neurônios apresentados como os *spiking neurons* tem esbarrado nos mesmos problemas encontrados nos neurônios MCP, fechando assim o ciclo de desenvolvimento das redes neurais artificiais.

3.2 Definição

Segundo Haykin(1999) redes neurais artificiais são sistemas paralelos distribuídos compostos por unidades de processamento simples (neurônios artificiais) que calculam determinadas funções matemáticas dispostas em uma ou mais camadas interligadas por um grande número de conexões geralmente unidirecionais.

3.3 Neurônios Artificiais

Existem vários modelos de neurônios artificiais apresentados atualmente, alguns deles são amplamente utilizados como o perceptron e o adaline.

O primeiro modelo de neurônio proposto foi o MCP e foi uma simplificação do que era conhecido sobre os neurônios biológicos na época. O MCP consiste em um conjunto de entradas e apenas uma saída. As entradas estão associadas a pesos que podem ser positivos ou negativos de acordo com sua função. Os pesos indicam qual a importância de cada entrada para o

neurônio. O resultado do neurônio em questão é a soma das entradas multiplicadas por seus pesos e colocados em sua função de ativação.

A função de ativação é responsável por gerar a saída do neurônio a partir dos valores de peso e das entradas. Existem várias funções de ativação, dentre elas as funções radial, gaussiana, degrau e linear.

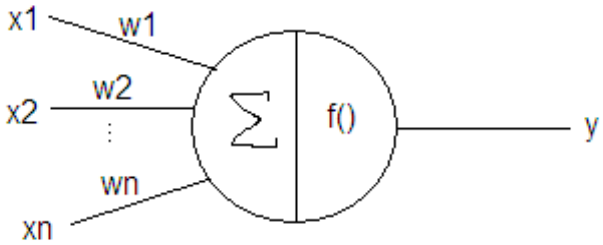

Figura 3.1: Neurônio de McCulloch e Pitts

No neurônio apresentado na Figura 3.1, as entradas são representadas por *(x1...xn)*, os pesos por *(w1...wn)*, a função de ativação por *f( )*, a saída por *y* e sigma representa a soma ponderada das entradas. O neurônio dispara quando as somas dos impulsos que recebe ultrapassa o valor de limiar.

O modelo do perceptron proposto por Rosenblatt , segundo Braga, Carvalho & Ludermir(2007), segue a mesma idéia do neurônio MCP que possui função de ativação do tipo limiar, ou seja, as soma ponderada das entradas é comparada com um valor de limiar e caso o valor da soma seja maior ou igual ao valor de limiar o neurônio é ativado, caso contrário a saída fica desativada. As mudanças básicas entre os modelos são o fato do perceptron possuir um algoritmo de treinamento sendo introduzido assim o aprendizado em redes neurais artificiais e a organização dos neurônios em uma rede com a camada de entrada, um nível intermediário formado pelas associações e um nível de saída. Embora essa topologia possua três níveis ela é conhecida como perceptron de uma camada. O perceptron constitui essencialmente um separador linear.

Rosenblatt demonstrou o teorema de convergência do perceptron, ou seja, ele provou que um neurônio MCP treinado com o algoritmo de treinamento do perceptron sempre converge caso o problema em questão seja linearmente separável.

O perceptron possui um termo de polarização associado ao primeiro peso que determina onde ocorrerá a separação linear do espaço de entrada. Este termo é chamado de bias e é ajustado durante o treinamento da rede para auxílio no treinamento.

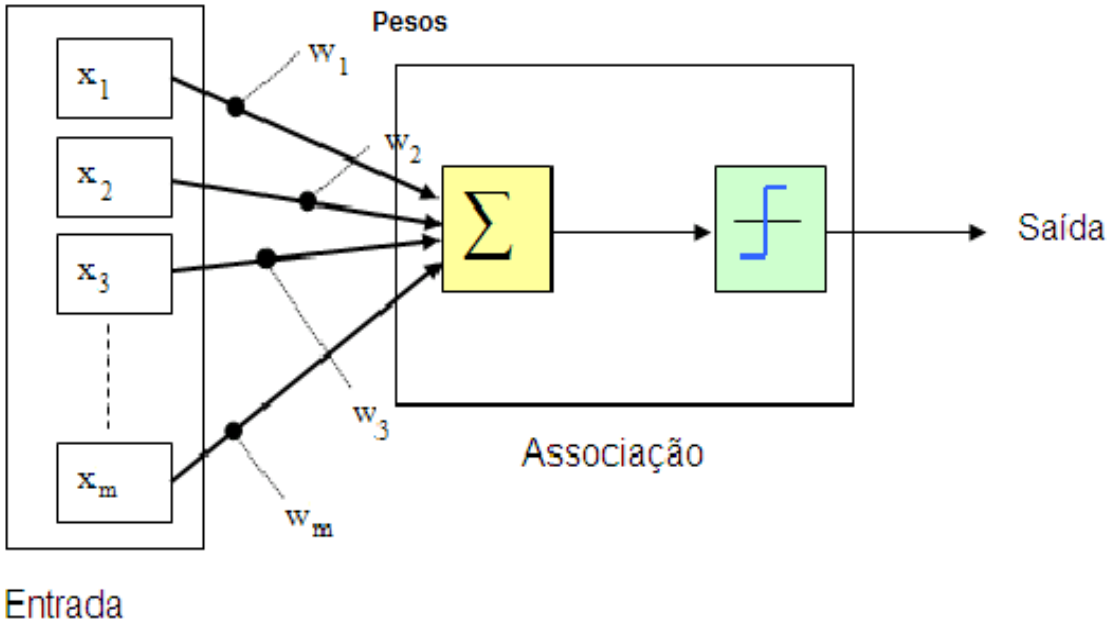

Figura 3.2: Perceptron simples com saída única

No caso da Figura 3.2, a camada de associação possui um neurônio apenas que gera a saída desejada.

Segundo Kröse & Van der Smagt(1996) ,o modelo adaline (*adaptive linear element*) foi proposto por Bernard Wildrow e Marcian Hoff. Este modelo é muito similar ao perceptron porém possui função de ativação exclusivamente linear. Este modelo caracteriza um aproximador linear de funções.

No caso do adaline, existe um termo de polarização, representando um grau de liberdade maior para o modelo, que acaba por deslocar a função de ativação em relação à origem. A saída deste modelo corresponde a uma combinação linear das entradas em que os pesos associados são obtidos através de treinamento.

A Figura 3.3 representa o neurônio adaline com duas entradas *x0* e *x1*, a saída *y*, pesos *w0* e *w1*. Associado ao peso w0 o termo de polarização ou *bias* indicado pela letra grega θ.

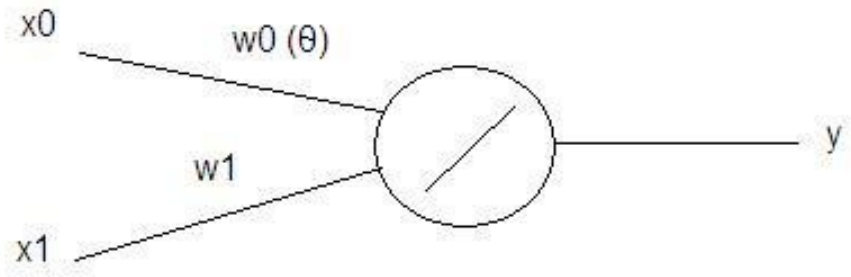

Figura 3.3: Neurônio Adaline

Atualmente os neurônios que estão sendo desenvolvidos são chamados de *spiking neurons*. Segundo Gerstner & Kistler(2002), *spinking neurons* são neurônios que não são ativados em todos os ciclos de propagação como acontece com os perceptrons por exemplo. Estes neurônios possuem uma ativação apenas quando um potencial de membrana (uma qualidade do neurônio relacionada carga elétrica da membrana) atinge um certo valor. Quando este neurônio é acionado ele propaga um sinal por toda a rede que irá integrar o potencial dos outros neurônios e ativar os neurônios devidos.

O potencial de ativação é calculado por uma equação diferencial e depende do estado do neurônio. Este tipo de neurônio possui também a informação do tempo em seu treinamento, sendo assim importante o seu estado atual no momento em que está sendo treinado.

3.4 Arquitetura de Redes Neurais Artificiais

A capacidade de processamento de um neurônio artificial é baixa, porém quando associados com outros neurônios, em camadas, em conjunto com os padrões de conexão entre camadas formam o que é chamado de rede neural artificial(FAUSSETT, 1994).

Existem várias formas de associar neurônios em redes neurais. Algumas destas arquiteturas são amplamente utilizadas e serão apresentadas a seguir.

A determinação do número de neurônios de cada camada da rede neural é o problema mais fundamental em aprendizado de redes neurais e motiva muitas pesquisas na área. A complexidade da rede neural deve acompanhar a complexidade do problema a ser resolvido. O

número de neurônios em cada camada é conseguido por testes de várias arquiteturas diferentes até ser encontrada a que apresente a diminuição mais rápida do erro quadrático de treinamento.

Quanto ao número de camadas, as redes podem ser classificadas como *single-layer* (camada única) ou *multi-layer* (camada múltipla). As unidades de entrada não são consideradas camadas por não fazerem nenhum tipo de computação.

No caso das redes *multi-layer,* as camadas se comportam da seguinte maneira :na primeira camada intermediária cada neurônio contribui com retas para a formação a superfície no espaço de entrada, a segunda camada recombina as retas descritas pelos neurônios da camada anterior formando regiões convexas onde o número de lados é definido pelo número de unidades conectadas e na camada de saída cada neurônio forma regiões que são combinações das regiões convexas definidas pelos neurônios conectados a ele da camada anterior.

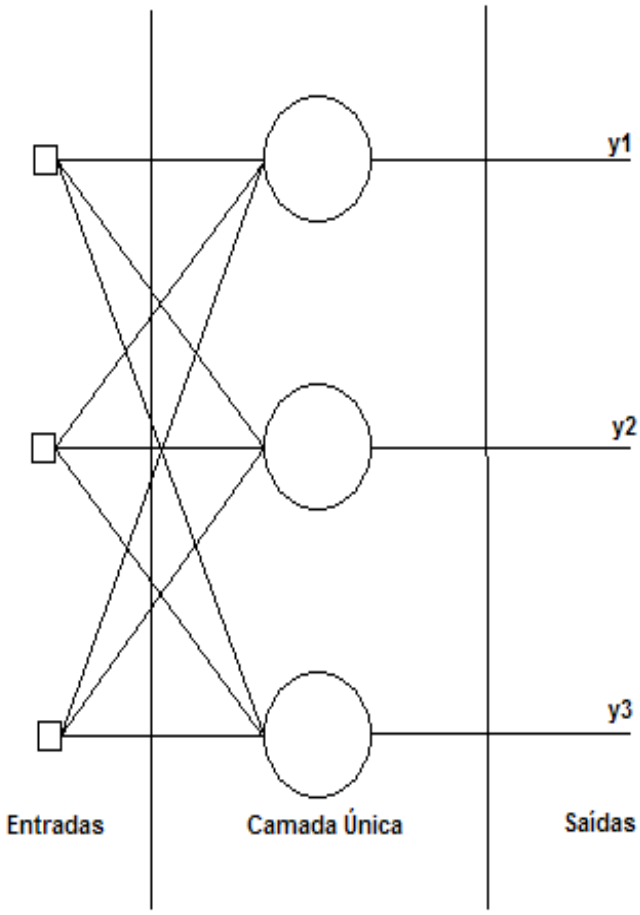

Figura 3.4: Rede de camada única

Quanto ao sentido das redes, as mesmas podem ser do tipo *feedforward* onde os sinais fluem da entrada para a camada de saída da rede, sendo assim uma rede que é sempre alimen-

tada para frente ou do tipo recorrente onde as saídas são utilizadas como entrada da própria rede sendo que o sinal faz um *loop* na rede.

Existem também as redes Hopfield. Este tipo de rede neural artificial caracteriza-se por não possuir entradas externas e por sua operação ser feita através da dinâmica de mudança de estados dos neurônios operando de forma auto-associativa.

A Figura 3.4 representa uma rede *feedforward* de camada única de neurônios. Este tipo de rede consegue resolver problemas multivariáveis de múltiplas funções acopladas com algumas restrições por possuírem apenas uma camada. Ao aumentarmos o número de camadas para duas, a capacidade computacional da rede neural é ampliada além de proporcionar uma universalidade na aproximação de funções porém a dificuldade de treinamento desta rede aumenta juntamente com a possibilidade de sucesso do treinamento . Estas redes são consideradas redes estáticas pois não há recorrência dos dados.

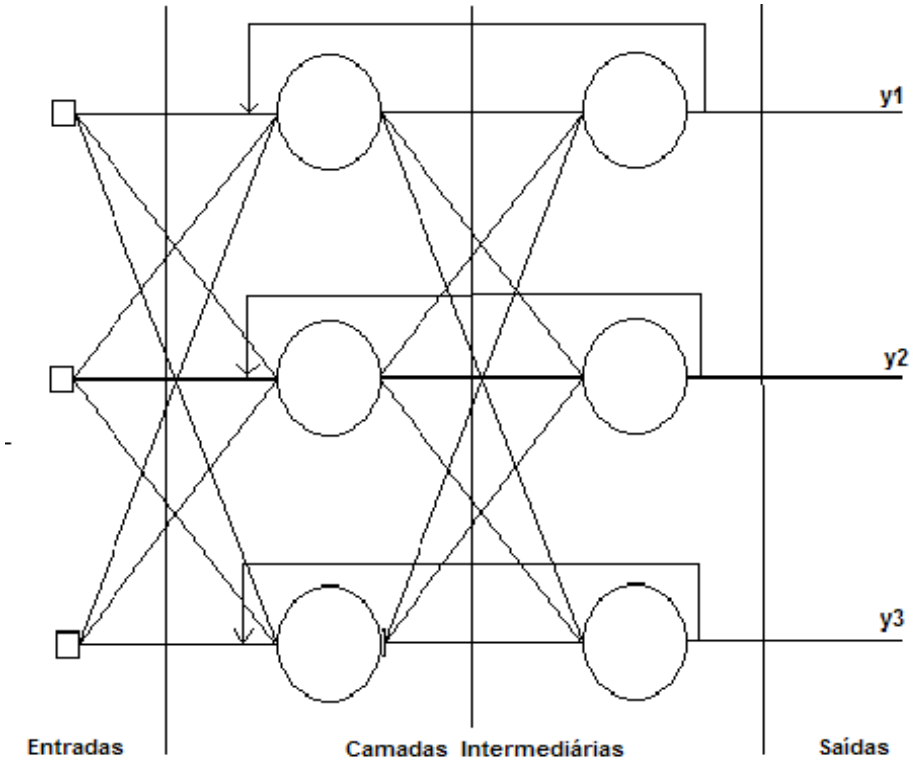

Figura 3.5: Rede com recorrências e camada intermediária

As redes com recorrência, semelhantes a rede da Figura 3.5, são consideradas redes dinâmicas e possuem um algoritmo de treinamento consideravelmente complexo comparado às redes *feedforward.* Estas redes são utilizadas para resolução de problemas que envolvem processamento temporal como os casos onde há previsão de eventos futuros.

3.5 Treinamento de Redes Neurais Artificiais

Dentre as principais características das redes neurais artificiais, a capacidade de aprender por meio de exemplos é uma das mais importantes. Este aprendizado em uma rede neural consiste no ajuste dos pesos e do *bias* pelos algoritmos de treinamento de acordo com o conjunto de entradas através de vários passos. O treinamento de uma rede neural consiste na diminuição do erro quadrático médio da resposta da rede neural para os dados de treinamento.

Segundo Braga, Carvalho & Ludermir(2007), existem dois tipos básicos de aprendizado para redes neurais, o aprendizado supervisionado e o aprendizado não-supervisionado.

O aprendizado supervisionado consiste na presença de uma entidade que é responsável por estimular as entradas da rede neural por meio de padrões de entrada e comparar a saída calculada pela rede neural com a saía desejada. A saída é ajustada através da modificação do valor do conjunto de pesos. Este tipo de aprendizado é utilizado em problemas que se deseja obter o mapeamento de padrões entre entrada e saída.

O aprendizado supervisionado pode ser *on-line* onde os dados de treinamento da rede mudam continuamente e a rede deve estar sempre se atualizando, e pode ser *off-line* onde o conjunto de treinamento não muda e a solução obtida pela rede é fixa.

A forma mais comum de implementar este tipo de aprendizado é através de correção de erros, onde a saída da rede é comparada com a saída desejada e procura-se minimizar o erro entre estes dois valores. Os exemplos mais conhecidos de algoritmos deste tipo de aprendizado são a regra delta e o algoritmo *backpropagation.*

No caso do aprendizado não-supervisionado , apenas os dados de entrada estão disponíveis para o treinamento ao contrário do aprendizado supervisionado onde temos pares de entrada e saída. Os conjunto de entradas é apresentado exaustivamente a rede e a presença de padrões neste conjunto faz com que o aprendizado seja possível. Este tipo de aprendizado é utilizado quando a finalidade é encontrar características estatisticamente relevantes nos dados de entrada.

Como exemplos de aprendizados não supervisionados tem-se o aprendizado hebbiano e o aprendizado por competição. Segundo Haykin(1999), o aprendizado hebbiano consiste basicamente em incrementar as sinapses entre neurônios que são ativados simultaneamente e decrementar ou eliminar as sinapses de neurônios que são ativados de modo assíncrono . No

aprendizado por competição os neurônios da camada de saída da rede neural competem entre si para serem ativados. A unidade vencedora tem seus pesos atualizados, e como esta unidade tem uma chance maior de vencer outras disputas ela ficará muito mais forte que as outras restando sozinha no final do treinamento. Este procedimento é chamado de *Winners Takes All (WTA)*. Este aprendizado é baso para os modelos de Grossberg e mapas de Kohonen, dois algoritmos de treinamento de redes neurais artificiais.

3.5.1 O Algoritmo *Backpropagation*

O algoritmo de treinamento de redes MLP mais popular é o *backpropagation* que, por ser supervisionado, utiliza pares entrada e saída para possibilitar o ajuste de pesos da rede através de um mecanismo de correção de erros.

O algoritmo percorre a rede em dois sentidos possuindo portanto duas fases a fase *forward* e a fase *backward*.

A fase forward envolve os seguintes passos :

1. O vetor de entrada é apresentado às entradas da rede e as saídas dos neurônios da primeira camada escondida são calculadas.

2. As saídas da primeira camada escondida proverão as entradas da camada seguinte. As saídas da camada seguinte são então calculadas e o processo é repetido até que atinja-se a camada de saída.

3. As saídas produzidas pelos neurônios da camada de saída são então comparadas às saídas desejadas para o dado vetor de entrada e o erro correspondente é calculados

Como pode ser visto, o objetivo principal desta fase é obter o erro de saída após a propagação do sinal por todas as camadas da rede. Em seguida tem inicio a fase backward que envolve as etapas a seguir:

1. O erro da camada de saída é utilizado para ajustar diretamente os pesos utilizando-se assim o gradiente descendente do erro.

2. Os erros dos neurônios da camada de saída são propagados para a camada anterior , utilizando-se para isso os pesos das conexões entre as camadas, que serão multiplicados pelos erros correspondentes. Assim tem-se um valor de erro estimado para

cada neurônio da camada escondida que representa uma medida da influência de cada neurônio da camada em questão no erro de saída da camada posterior.

3. Os erros calculados são utilizados para ajustar seus pesos pelo gradiente descendente , analogamente ao procedimento utilizado para a cama posterior.

4. O processo se repete até que os pesos da camada de entrada sejam ajustados concluindo-se assim o ajuste dos pesos de toda a rede para o vetor de entrada X e sua saída deseja Y.

O ajuste dos pesos feito pelo algoritmo é baseado na regra delta para o treinamento de redes adaline. A generalização para redes de múltiplas camadas é conhecida como regra delta generalizada. Em Braga, Carvalho & Ludermir(2007) podem ser encontradas as deduções das equações de ajuste.

3.5.2 O Algoritmo Rprop

O algoritmo *backpropagation* padrão é muito lento para várias aplicações e seu desempenho piora sensivelmente para problemas maiores e mais complexos. O algoritmo em questão requer que todos os padrões de treinamento sejam apresentados centenas ou até milhares de vezes. Este fato limita a utilização prática deste algoritmo permitindo apenas o treinamento de pequenas redes com poucos milhares de pesos ajustáveis.

Segundo Braga, Carvalho & Ludermir(2007), o algoritmo Rprop (*Resilient backpropagation*) é um algoritmo de adaptação global, derivado do *backpropagation*, que realiza um treinamento supervisionado *batch* em redes do tipo MLP. Este algoritmo procura eliminar a influência negativa do valor da derivada parcial na definição do ajuste de pesos. Essa influência ocorre devido ao fato de que quando a saída do neurônio for próxima de 0 (ou 1) e a saída desejada for 1 (ou 0) a derivada irá apresentar valores próximos de 0 fazendo com que os neurônios recebam um ajuste mínimo.

Este algoritmo elimina este problema utilizando apenas o sinal da derivada e não seu valor. O sinal indica a direção do ajuste dos pesos. O tamanho do ajuste é dado por um "valor de atualização" conforme indicado pela equação presente na Figura 3.6.

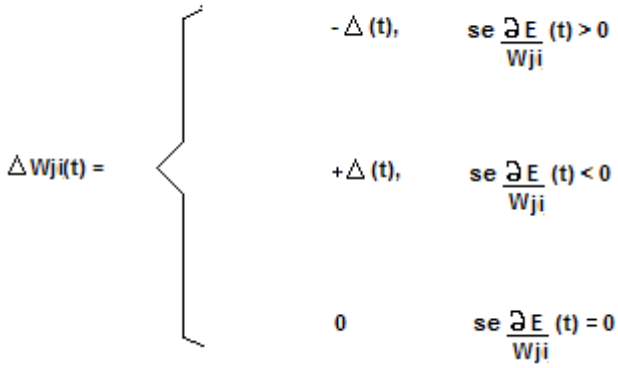

Figura 3.6: Função do valor de atualização

O valor de atualização, Figura 3.7, é definido por um processo de adaptação que depende do sinal da derivada do erro com relação ao peso a ser ajustado.

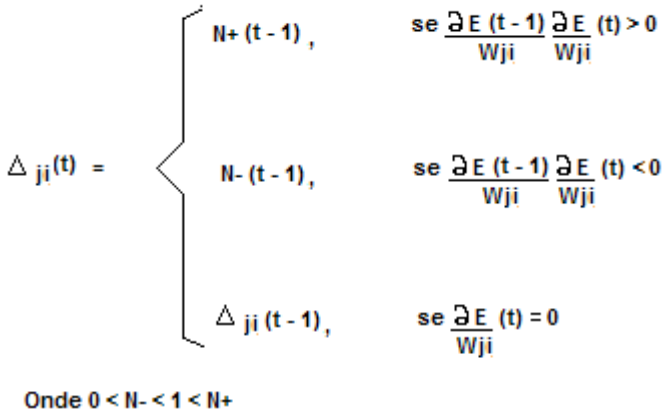

Figura 3.7: Adaptação do valor de atualização

Segundo a regra de adaptação do algoritmo, quando a derivada parcial do erro em relação a um peso mantem seu sinal, o que indica que o o ultimo ajuste reduziu o erro cometido, o valor de atualização é aumento pelo fator N+, acelerando a convergência do treinamento. Caso o sinal seja trocado, o valor é reduzido pelo fato N-, mudando a direção do ajuste.

3.6 Utilização de Redes Neurais Artificiais

As redes neurais artificiais podem ser utilizadas para executar 3 tarefas básicas : classificação, previsão e categorização.

Algumas aplicações na área de classificação são reconhecimento de caracteres, reconhecimento de imagens, diagnóstico, análise de risco de crédito. No caso das previsões as redes neurais podem ser utilizadas em previsão do tempo, previsões financeiras, modelagem de sistemas dinâmicos e previsões de seqüências de DNA . Mineração de dados, análise de expressão gênica, agrupamento de clientes são exemplos de categorização (BRAGA,CARVALHO & LUDERMIR, 2007).

4 TEORIA DA COMPLEXIDADE COMPUTACIONAL

A teoria da complexidade computacional é uma investigação de tempo, memória ou outros recursos necessários para a resolução de problemas computacionais. A abordagem que será discutida neste trabalho trata a complexidade de tempo.

4.1 Definição

Segundo Sipser(2006), o tempo de execução ou complexidade de tempo é o número máximo de passos que que uma máquina de Turing *M* usa sobre entradas de comprimento *n*. Normalmente utiliza-se *n* para indicar o comprimento da entrada. Seja *f(n)* a função que descreve o tempo de execução de *M*, dizemos portanto que *M* roda em tempo *f(n)* e que *M* é uma máquina de Turing de tempo *f(n)*.

4.2 Classe P

Segundo Lewis & Papadimitriou(1998), os problemas de computação podem ser classificados de duas formas possíveis : os que podem ser resolvidos com algoritmos e os que não podem.

Mesmo com o desenvolvimento da tecnologia alguns problemas computacionais não podem ser resolvidos em uma quantidade de tempo aceitável.

A classe P é a classe das linguagens que podem ser decididas em tempo polinomial, ou seja, existe uma máquina de Turing *M* determinística de uma única fita que após *p(n)* passos, onde *p(n)* é um polinômio, decide estas linguagem, ou seja, é a classe de problemas que possui um algoritmo de resolução que é executado em tempo polinomial. Os problemas computacionais podem ser descritos em forma de linguagens.

Os algoritmos que rodam em tempo polinomial possuem duas características básicas: eles possuem um limitante superior polinomial para o número de estágios que o algoritmo executa quando tem-se uma entrada de tamanho *n* e também cada estágio da execução do algoritmo é examinado para assegurar que possam ser implementados em tempo polinomial.

Alguns exemplos de problemas da classe P são : o problema da existência de um caminho entre dois nós de um grafo, testar se dois números são primos entre si.

4.3 Classe NP

A classe P nos mostrou que pode-se evitar a força bruta em alguns problemas e encontrar um algoritmo que resolva os problemas em tempo polinomial, porém existem alguns problemas que não possuem algoritmos de tempo polinomial conhecidos até hoje.

Um verificador polinomial é um algoritmo que possui como entrada uma instância do problema e uma possível resposta para o problema e faz a verificação da resposta em tempo polinomial. Os problemas da classe NP são problemas que possuem um verificador polinomial, são problemas que possuem um algoritmo que os resolve em um tempo não-determinístico polinomial(SIPSER,2006).

Alguns exemplos de problemas da classe NP são : o problema do caminho hamiltoniano, o problema da clique em um grafo, o problema da soma de subconjuntos.

4.4 Redutibilidade

A redução de problemas para facilitar sua resolução é uma prática muito comum tanto na matemática quanto na vida.

Segundo Atallah(1998), para resolver o novo problema temos duas alternativas, ou resolvemos o problema para o qual o problema inicial foi reduzido e temos a resposta em termos do problema, ou então utilizamos uma subrotina que resolve o problema principal. Por exemplo podemos resolver um problema de otimização cuja solução é viável e maximize o valor de uma função objetivo *g* chamando repetidamente a subrotina que resolve o problema de decisão correspondente onde existe uma solução viável *x* que satisfaça *g(x) >= k*. O segundo método apresentado é chamado de redutibilidade de Turing.

Um problema *A* é redutível a um problema *B* por mapeamento em tempo polinomial, ou simplesmente em tempo polinomial, se existe uma função computável em tempo polinomial em que todas as instâncias de *A* são convertidas em instâncias de *B*. Esta função é denominada redução de tempo polinomial de *A* para *B* (SIPSER,2006).

4.5 NP-Completude

Stephen Cook e Leonid Levin no inicio dos anos 70 contribuíram de forma muito significativa para a questão P x NP. Eles descobriram que alguns problemas NP possuem complexi-

dade relacionada com todos os problemas da classe, ou seja, se for descoberto um algoritmo que resolva algum destes problemas em tempo polinomial, todos os problemas da classe podem ser resolvidos com este algoritmo. Estes problemas ficaram conhecidos como NP-Completos.

A importância desta descoberta na prática é a seguinte : o fato de existirem problemas NP-Completos faz com que não seja necessária a busca de um algoritmo para resolução de problemas que não podem ser resolvidos em tempo polinomial provando que são NP-Completos.

Para provarmos que um problema é NP-Completo são necessários dois passos básicos, o primeiro é encontrar um verificador polinomial para o problema, o segundo passo é reduzir o problema a um problema NP-Completo qualquer em tempo polinomial.

A Figura 4.1 ilustra em conjuntos os tipos de problemas.

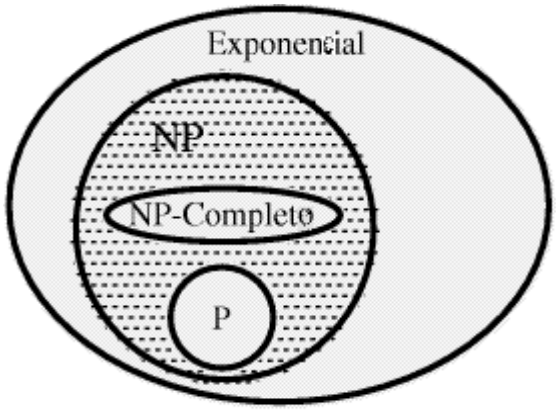

Figura 4.1: Classificação dos problemas

4.6 Problemas NP-Difíceis

Os problemas NP-Difíceis são problemas que possuem um problema NP-Completo que é redutível a ele. Geralmente estes problemas são de otimização, decisão e busca. Se um problema de otimização possui uma versão de problema de decisão NP-Completo, então ele é NP-Difícil. Os problemas de particionamento são geralmente NP-Difíceis devido à sua característica de otimização.

5 HARDWARE/SOFTWARE CO-DESIGN

5.1 Introdução

Segundo De Micheli & Gupta(1997), os sistemas de engenharia podem ser vistos como um grupo de componentes com operações combinadas afim de desenvolver alguma tarefa.

A maioria dos sistemas projetados atualmente são eletrônicos ou possuem subsistemas eletrônicos para monitoramento e controle. Esses sistemas possuem interface programável e com isto possuem componentes que podem ser implementados em hardware e componentes que podem ser implementados em software.

O projeto do sistema depende da área de aplicação onde está inserido para definição de prioridades que podem ser o desempenho, o custo total do produto, o grau de dificuldade de programação.

5.2 Definição

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 (STRAUNSTRUP & WOLF,1997).

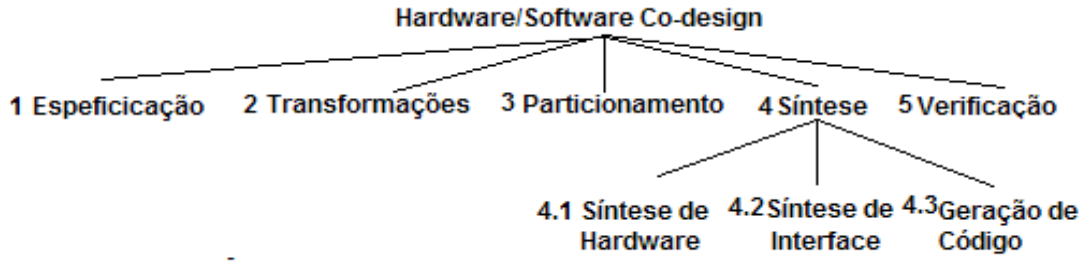

Figura 5.1: Fases do Hardware/Software Co-design .

Adaptado de Lòpes-Vallejo,Lòpez & Iglesias,1999.

A Figura 5.1 apresenta as fases básicas do hardware/software co-design. Inicialmente tem-se a especificação do sistema a ser desenvolvido seguida pelas transformações de projeto . Um particionamento é proposto e a síntese de hardware, de interface e a geração de código são

desenvolvidas e por fim uma verificação do projeto encerra o processo (LÓPEZ-VALLEJO,LÓPEZ & IGLESIAS,1999).

5.3 Questões Envolvidas em Design

Devido à existência de inúmeras configurações de hardware e aplicações possíveis para os sistemas, existem várias questões de interesse para o projeto.

O domínio da aplicação é uma questão importante para o design de sistemas. Os sistemas podem ser classificados por sua área de atuação como sistemas não embarcados e embarcados. Os sistemas não embarcados são utilizados em processamento de informação por exemplo e sistemas embarcados tem uma vasta área de atuação nas industrias, produtos para consumidores como celulares, veículos e sistemas de telecomunicação dentre outros. Estes sistemas podem ser distribuídos geograficamente como sistema de telefonia, distribuídos localmente como sistemas de controle de aviões ou fixos em estações de trabalho.

O grau de programabilidade do sistema também é muito importante no desenvolvimento, pois o hardware pode ser totalmente programável ou pode não ser programável. Um computador pessoal, por exemplo, possui alto grau de programabilidade enquanto em um sistema embarcado não é acessível um alto nível de programação. Este fato ocorre pois o software é disponibilizado pelo fabricante do produto sendo apenas algumas funções programáveis. Os vários níveis de programação em um sistema são a programação de hardware, de instruções ou de aplicativos. O nível mais alto é o nível de aplicação onde o sistema já possui programas dedicados sendo executados e o usuário só programa para executar as tarefas. O nível de instruções está presente na maioria do sistemas eletrônicos que possuem um conjunto específico de instruções. Exemplo desses sistemas são os microprocessadores e os microcontroladores, onde se pode criar o software para controle. A programação de hardware permite que este seja modificado da forma que o usuário desejar para atender suas necessidades. Como exemplos destes tipo de programação de hardware tem-se os dispositivos de hardware reconfigurável como as *FPGA's*.

Com relação à implementação desses sistemas podemos citar o estilo de desenvolvimento de circuitos, o nível de integração e a tecnologia de produção. Um sistema deve possuir componentes de diferentes escalas de integração e vários tipos de tecnologia de fabricação. A escolha dos componentes afeta diretamente a desempenho final do sistema, custo e sua impor-

tância. O nível de programabilidade pode ser atingido com a utilização de memórias de leitura e escrita além das interconexões programáveis.

5.4 Co-design de Sistemas Embarcados

Segundo De Micheli & Gupta(1997), geralmente sistemas embarcados regulam componentes mecânicos via atuadores e recebem informações do meio externo através de sensores. Sistemas de controle são geralmente sistemas de reação pois reagem a estímulos gerados pelo ambiente. Sistemas de tempo real implementam funções que devem ser executadas em uma certa janela de tempo.

As funções requeridas e o tamanho dos sistemas embarcados variam de acordo com sua aplicação. A utilização de microcontroladores faz com que o custo do projeto diminua consideravelmente e possibilita soluções flexíveis, ou seja, que se adaptam a várias situações e que resolvem muitos problemas. Entretanto sistemas de controle são complexos e exigem uma alta vazão de dados e processamento necessitando assim de projetos específicos com componentes que possuam grande capacidade de processamento.

O critério mais importante no design de um sistema embarcado que possua processamento de informações é performance , enquanto para sistemas de controle tem-se como prioridades a segurança, confiabilidade e disponibilidade.

As funções podem ser implementadas tanto em hardware quanto em software, porém neste caso devem ser seguidas regras de design para garantir as características principais aos sistemas desenvolvidos. As técnicas de testes para desenvolvimento devem ser utilizadas para testar se a opção escolhida realmente foi a melhor opção.

Os problemas no design de sistemas embarcados incluem modelagem, validação e implementação. Estas operações são críticas devido a heterogeneidade dos componentes que estão interagindo no sistema e porque a implementação que otimiza os objetivos do projeto deve possuir um particionamento hardware/software específico.

5.5 Métodos para Hardware/Software Co-design

Existem vários modelos propostos para o design conjunto de hardware e software em sistemas. A seguir são apresentados dois deles.

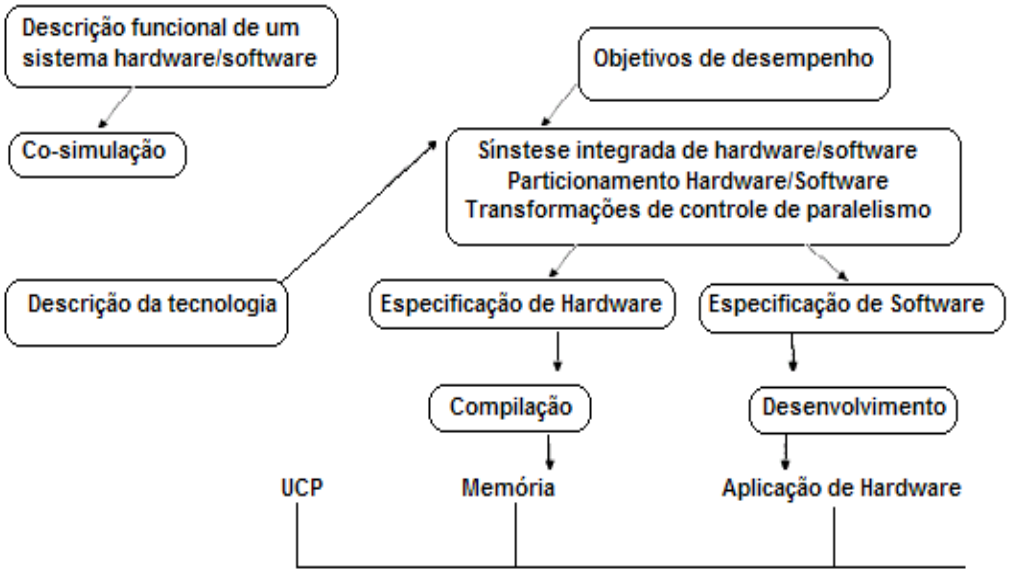

Figura 5.2: Metodologia para Hardware/Software Co-design.

Adaptado de Thomas, Adams & Schmit(1993).

A Figura 5.2 propõe uma metodologia de hardware/software co-design . Segundo Thomas, Adams & Schmit(1993), esta metodologia apresenta o particionamento hardware/software sendo influenciado pelos objetivos de desempenho e projeto, pela descrição de um modelo e sua simulação além da tecnologia envolvida. Esta metodologia gerou um modelo de co-design como o mostrado na Figura 5.3. O modelo da Figura 5.2 foi criado com objetivo de omitir a parte do sistema operacional, ser explícito com relação ao nível de concorrência entre hardware e software e para ser facilmente modificado caso algum componente migrasse da parte do hardware para parte do software.

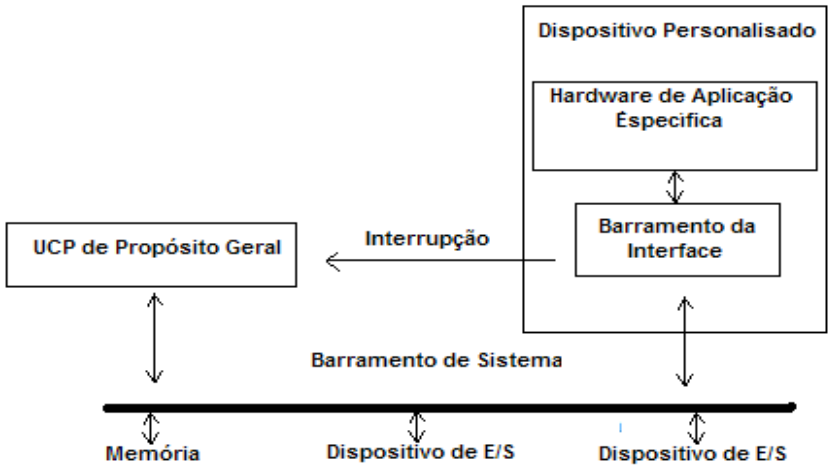

Figura 5.3: Modelo de co-design

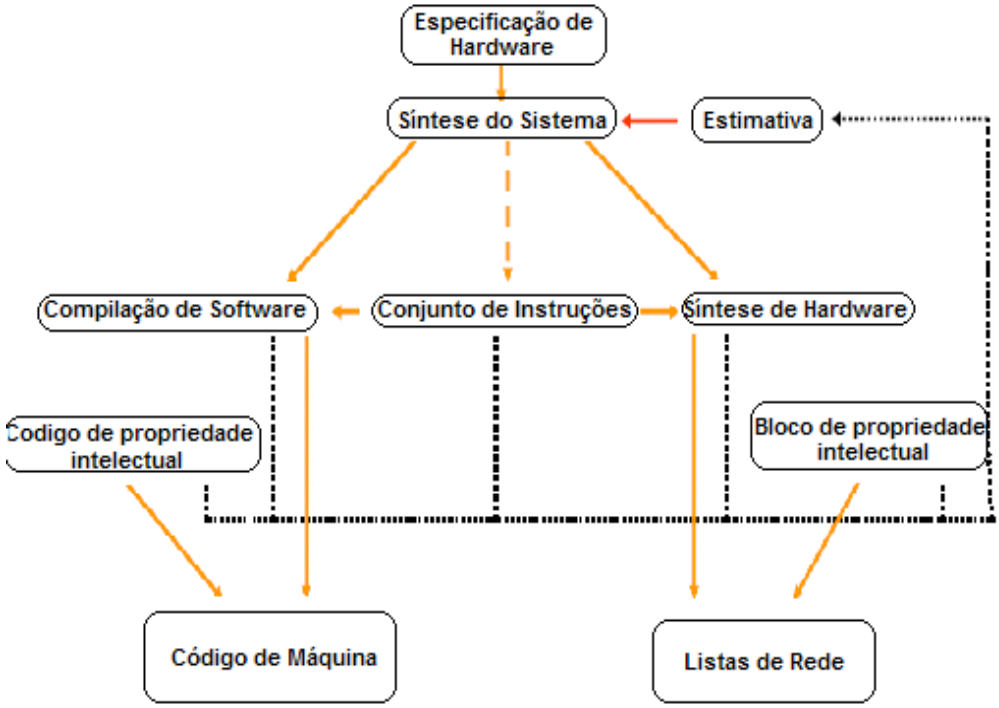

Figura 5.4: Outra metodologia para hardware/software co-design

Na Figura 5.4, é apresentada outra possível metodologia para o co-design. Iniciando-se com a especificação do hardware , tem-se em seguida uma síntese do sistema que divide o hardware, o software e o conjunto de instruções que são inseridos em uma estimativa e modificados até que o objetivo do projeto seja alcançado.

5.6 Projeto de Sistemas de Hardware e Software

O design de sistemas de hardware/software envolvem modelagem validação e implementação. Modelagem é o processo de conceitualização, refinamento de requisitos e produção de um modelo. Validação é o processo de se atingir um nível razoável de segurança de que o modelo vai funcionar da forma que foi planejado. Implementação que é a construção física do modelo do hardware e do software executável (DE MICHELI & GUPTA,1997).

A modelagem de sistemas embarcados pode ser homogênea, 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 desempenho desejados. Esta validação é feita com a simulação do hardware e do software.

A implementação do sistema envolve várias sub-tarefas, 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.

6 PARTICIONAMENTO HARDWARE/SOFTWARE

6.1 Introdução

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 co-design cujo objetivo principal é encontrar um particionamento que represente uma implementação preenchendo 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 técnicas vêm sendo desenvolvidas e propostas em vários projetos de co-design. Esses modelos e abordagens funcionam muito bem nos ambientes de desenvolvimento para os quais foram projetados, porém há tantas diferenças entre os modelos que não é possível comparar os resultados destas soluções.

No particionamento hardware/software tanto a parte de hardware quanto a parte de software podem ter diferentes níveis de granularidade. Um alto nível de granularidade significa que as tarefas ou os módulos de hardware possuem um grande número de especificações de funcionalidade ou comportamento enquanto um nível baixo de granularidade significa que estas especificações estão em menor número. Implementar sistemas em hardware traz um ganho em tempo de execução e gasto de energia porém consome espaço físico e gera alto custo de implementação. Em contrapartida implementar sistemas em software diminui o custo financeiro de implementação poréme possui um custo de manutenção maior, tempo de execução é alto e os processadores consomem muita energia(ARATÓ,MANN & ÓRBAN,2005).

6.2 Histórico

Os dois primeiros algoritmos que resolviam o particionamento hardware/software, *Cosyma* e *Vulcan,* foram apresentados em 1993. O algoritmo *Cosyma* utiliza a técnica de *simulated anealling* com boa granularidade e o algoritmo *Vulca*, através de processos interativos, blocos de software são extraídos da versão totalmente de hardware como solução considerando a questão de tempo de execução (LÓPEZ-VALLEJO & LÓPEZ,2003).

Em seguida foi apresentado algoritmo por Ptolemyem (1997), que possuía duas características importantes : a primeira é que ele é capaz de adaptar sua função objetivo para ser global ou para priorizar operações críticas, a segunda é que diferentes implementações de hardware são consideradas.

A introdução das técnicas de inteligência computacional se deu em 1998 por grupos de pesquisa que utilizaram algoritmos genéticos para solucionar o problema do particionamento. Em 2001 foi proposto um algoritmo que leva em conta granularidade variável. Uma comparação dos algoritmos de particionamento propostos foi feita em Wiangtong, Cheung & Luk(2002), e pode ser visualizada na Figura 6.1. O algoritmo genético em todos os casos apresentou resultados satisfatórios com relação ao *simulated anealling* e a busca TABU.

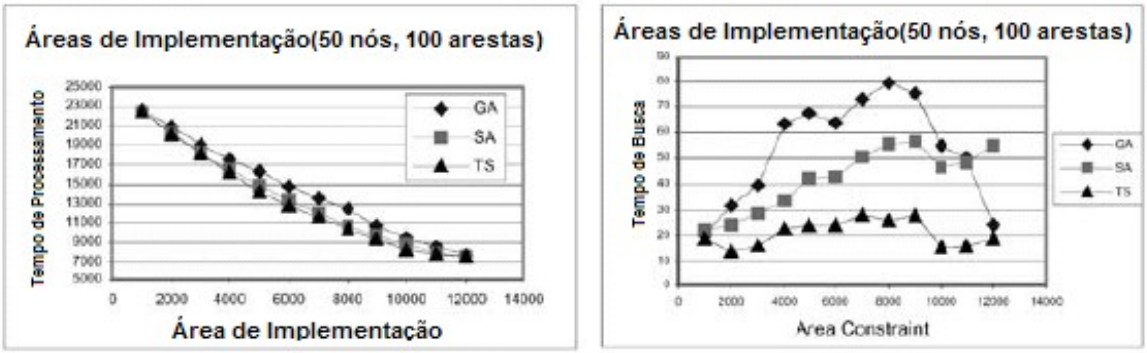

Figura 6.1: Comparação de algoritmos (Algoritmo Genético (GA), Simulated Anealling(SA), Busca TABU(TS) para particionamento.

Adaptado de Wiangtong, Cheung & Luk(2002).

6.3 Aspectos Básicos de um Ambiente de Particionamento

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, o tipo de aplicação para a qual será voltado o projeto , o modelo computacional escolhido como representação e a granularidade escolhida.

6.4 Modelo de Particionamento em Sistemas Embarcados

As definições encontradas na literatura sobre o problema possuem duas falhas básicas : não se comportam bem com uma entrada de dados muito grande devido ao fato do custo para resolução do problema crescer exponencialmente em relação ao número de tarefas e módulos de hardware , e as heurísticas utilizadas dão um resultado que possui um erro muito grande com relação ao resultado esperado. Entretanto a complexidade deste problema é difícil de se determinar.

Um modelo simplificado de particionamento deve definir formalmente vários tipos de particionamentos, analisar a complexidade destes particionamentos formalmente e obter como resultado algoritmos que possam ser utilizado para um grande numero de projetos.

O modelo proposto por Arató et.al. (2004) possui as seguintes características : apenas um contexto de hardware e um contexto de software são considerados sendo necessário que as partes do sistema sejam mapeadas para apenas um dos dois contextos, o custo associado à implementação em software de um componente é o tempo de execução deste software, o custo associado à implementação de hardware está relacionado a fatores do tipo, dissipação de calor, área necessária para implementar o circuito, consumo de energia. Sendo o tempo de execução em hardware consideravelmente mais rápido que o tempo de execução em software este é tomado como 0 e por fim se dois componentes que se comunicam foram implementados de forma diferente é associado um custo de comunicação entre eles e caso sejam implementados da mesma forma, é como se o custo da comunicação não existisse.

Segundo Quan,Hu & Greenwood,(1998), a abordagem hierárquica tem atingido bons resultados em problemas complexos pois os sistemas podem ser hierarquicamente divididos em sistemas mais simples que são mais fáceis de trabalhar.

Hierarchical task graphs (HTG's) são grafos que possuem três tipos de nós. O nó simples que representa uma tarefa que não possui sub-tarefas, um nó complexo pode ser dividido em inúmeras sub-tarefas e os *dummie nodes* que existem em sub-grafos deste tipo representando apenas a relação de entrada e saída com outros grafos e não são associadas a nenhum tipo de tarefas.

O modelo hierárquico proposto considera que os módulos de hardware são conjuntos de objetos como SoC's, componentes funcionais, microcontroladores. Primeiramente é dado uma série de HTG's que descrevem o comportamento do sistema embarcado, depois são desenvolvidos os módulos hierárquicos de hardware(que são tratados igualmente a este modelo hierárquico). As definições de todos os pontos que devem ter comunicação são feitas considerando-se custo, energia, tempo de execução, etc e por fim a preferencia de design por alguns atributos é considerada e um modelo em grafo HTG é feito para os módulos de hardware na forma que otimiza todos os atributos de design ao mesmo tempo que satisfaçam os objetivos do projeto. Após feita a modelagem, uma forma de resolução do particionamento é utilizada para resolver o problema.

6.5 Definição Formal e Classificação do Problema

Para provarmos que o problema do particionamento é NP-Difícil é necessário que ele seja reduzido a um problema NP-Completo.

Para que seja feita a redução é necessário que seja feita uma definição formal do problema do particionamento hardware/software. A definição do problema e a redução proposta podem ser encontradas também em (ARATÓ et. al.,2004).

Dado um grafo não-orientado $G(V, A)$ onde V é um conjunto de vértices e A o conjunto de arestas. G representa o grafo de tarefas do sistema, os nós do grafo são os componentes do sistema que devem ser particionados e as arestas representam a comunicação entre estas partes do sistema. $s(vi)$ e $h(vi)$, onde i é índice, representam o custo de software e o custo de hardware de cada nó do grafo, enquanto $c(vi,vj)$, onde j também é índice, representa o custo de comunicação entre os nós vi e vj se eles estiverem em diferentes contextos.

P é uma instância de particionamento hardware/software se é uma bipartição de *V : P = (Vh, Vs)*, enquanto *Vh U Vs = V e Vh ∩ Vs = {}*. As arestas de *P* seguem a seguinte regra : *Ap = {(vi,vj) : vi Є Vs, vj Є Vh ou vi Є Vh, vj Є Vs}*. O custo total da parte de hardware é o somatório dos *h(vi)* para todo *vi Є Vh*, e o custo total da parte de software é a soma do somatório dos *s(vi)* para todo *vi Є Vs* com o somatório de *c(vi,vj) Є Ap*. O custo de software engloba custo de comunicação entre componentes pois são grandezas de tempo e juntas formam o tempo de execução total do software.

A partir desta definição formal, três problemas podem ser formulados :

1. Dados um conjunto inicial de hardware *H0* e um conjunto inicial de software *S0*. Existe algum *P* (particionamento hardware/software) onde *Hp<=H0 e Sp <= S0*?

2. Dado um conjunto de hardware *H0,* encontre um *P onde Hp <= H0 e Sp* é mínimo (Sistemas onde o custo total de produção é fator determinante).

3. Dado um conjunto de software *S0*, encontre um *P* onde *Sp <= S0 e Hp* é mínimo (Sistemas de tempo real).

O item 1 é NP-Completo se apenas grafos não-direcionados forem considerados. O item 1 esta em NP , desde que o particionamento seja possível com os limites estabelecidos, o próprio particionamento é uma prova valida. A redução proposta para este problema é feita para o problema da mochila.

Seja a instância seguinte do problema da mochila *:* dados *n* objetos com peso associado *wi* e valor associado *pi*, o peso é limitado a *W* e o valor mínimo limitado a *K*. A questão é verificar se existe um subconjunto de objetos *X* onde a soma dos pesos dos objetos não seja maior que *W* e que o valor dos objetos selecionados não seja menor que *K*. Baseado nesta definição temos a definição do item 1 como : seja *V* um conjunto de vértices e *A* um conjunto vazio de arestas. Sejam os valores de *h(vi)* associados com o *p(vi)* e os valores de *s(vi)* associados com os pesos *w(vi)*. Sendo *A* o somatório de todos os *p(vi)*, temos que *S0 = W* e *H0 = A – K*.

Esta instância do problema possui solução caso o problema da mochila possua solução. Admitindo uma solução para o item 1, *P = (Vh, Vs)* onde *Vh U Vs = V* e *Vh ∩ Vs = {}*, isto significa que *Sp* é o somatório de todos os *w(vi)* para todo *vi* pertencente a *V* e é menor ou igual a *W* e que *Hp* é o somatório de todos os *p(vi)* para todo *vi* pertencente a *V* e é menor ou igual a *A – K*. Estas duas definições provam que *X* é igual a *Vs* e é uma solução do problema da

mochila. Assumindo que *X* seja realmente uma resposta, *S0* é realmente a soma de todos os *s(vi)* para todo *vi* pertencente a *V* que é igual ao somatório de todos os *w(vi)* para todo *vi* pertencente a *V* e *H0* é maior ou igual ao somatório de todos os *h(vi)* para todo *vi* pertencente a *V* e também ao somatório de todos o *p(vi)* para todo *vi* pertencente a *V*. Isto verifica que *P = (V/X, X)* resolvendo o item 1.

A prova correspondente ao item 1 pode ser reduzida para o item 2. No caso da parte 2 temos uma solução onde *Hp* é menor ou igual *H0* e *Sp* é minimo. A parte 1 é claramente resolvível se *Sp* < *S0*. Da mesma forma resolvemos o problema do item 3. Com esta redução provamos que o particionamento hardware software é um problema NP-Difícil.

Apesar deste problema se tornar muito complexo para um número de entradas muito grande, existem casos particulares cuja resolução torna-se mais fácil. Por exemplo quando a parte das arestas do grafo, ou seja, a comunicação entre componentes é nula temos então puramente o problema da mochila. No caso da comunicação ser o fator crucial para o projeto, a parte de hardware é nula e a solução trivial é colocar todos os componentes em software.

O fato do problema pertencer a classe de problemas NP-Difíceis justifica a utilização de uma heurística para solucioná-lo já que não se conhece um algoritmo que resolva estes problemas em tempo de execução polinomial.

7 MATERIAIS E MÉTODOS

Este capítulo apresenta, inicialmente, o tipo de pesquisa em que se enquadra o presente trabalho. Em seguida, são apresentados os procedimentos metodológicos e ferramentas utilizadas para a realização do trabalho.

7.1 Tipo de Pesquisa

Segundo Jung(2004), o tipo de pesquisa deste trabalho pode ser classificado da seguinte forma :quanto à natureza é uma pesquisa aplicada, pois utiliza a aplicação dos conhecimentos básicos obtidos por meio de estudo para geração de um produto como objetivo; quanto aos objetivos esta pesquisa enquadra-se como exploratória, já que visa a descoberta de uma prática que pode vir a modificar a abordagem existente de um problema; quanto aos procedimentos é uma pesquisa experimental pois visa a descoberta de uma nova técnica e produto de software. A pesquisa é de laboratório pois é possível controlar as variáveis que podem intervir no experimento.

O tempo de aplicação do estudo, segundo Jung(2004), pode ser classificado como longitudinal pois os dados foram obtidos ao longo do tempo e a obtenção dos resultados foi lenta e sistemática.

O estudo e o referencial documental foram feitos com base principalmente em documentos e fontes primárias de informação, ou seja, em livros, artigos que constam em periódicos e anais de congresso.

7.2 Definição do Problema

O problema do particionamento hardware/software pode ser abordado de várias maneiras diferentes. Existem inúmeras 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.

A técnica de treinamento escolhida para a resolução do problema necessita de um banco de dados sobre o tema para que seja treinada a rede neural de forma consistente. A inexistência

do banco de dados, devido ao fato de o problema não possuir abordagem de resolução por métodos que utilizem banco de dados, levou à questão de geração de dados para treinamento da rede.

Para geração dos dados foi utilizada a técnica de algoritmos genéticos que será descrita a seguir, e o algoritmo genético utilizado aborda o problema do particionamento da seguinte forma : partindo de um circuito real foi feito um grafo acíclico direcionado. 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. Na Figura 7.1 é possível visualizar um grafo do tipo utilizado.

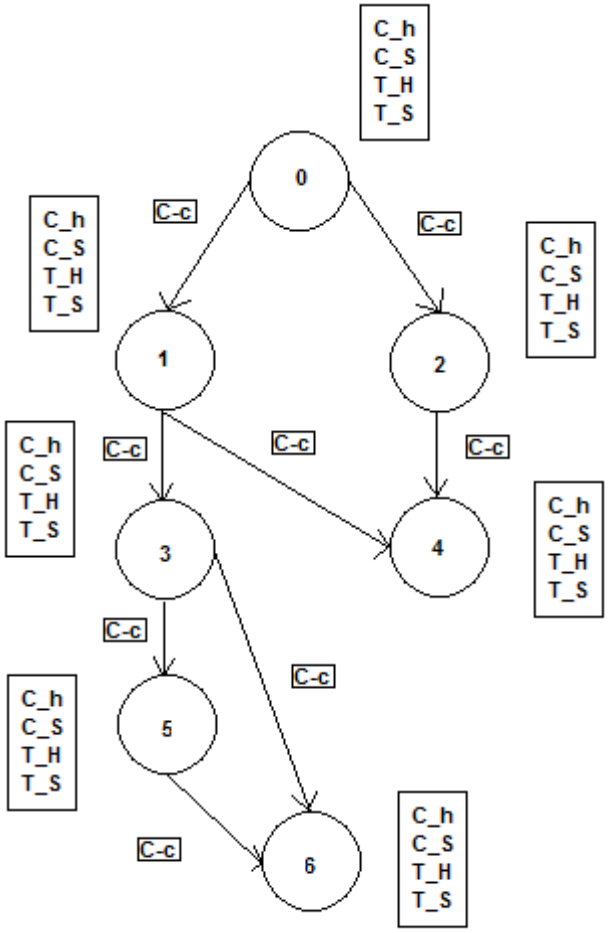

Figura 7.1: Exemplo de Grafo Acíclico Direcionado

No grafo da Figura 7.1 cada nó possui associado um custo de hardware *(C_h)*, um custo de software *(C_s)*, tempo de hardware *(T_H)* e tempo de software *(T_s)*. Cada nó do grafo representa um dos módulos do sistema a ser implementado e as arestas representam os custos

de comunicação *(C_c)* entre dois módulos. Os custos são medidos, no caso de hardware o custo de implementação e de software, o custo computacional, e os tempos são medidos como tempo de resposta em milissegundos.

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.

Este grafo pode ser escrito de várias formas para ser utilizado por um programa, porém foi escolhida a forma de arquivo texto como na Figura 7.2. Na primeira linha tem-se o número de nós e em seqüência cada linha é um nó em ordem crescente com informação dos custos associados ao nó primeiramente, seguidos dos custos de comunicação do nó para os outros nós precedidos pelo número do nó para o qual o custo se refere. Após o último custo existe um 0 para indiciar o final dos custos de comunicação. Tal notação é possível devido ao fato de que custos 0 não possuem arestas no grafo. Por fim é colocado o tempo limite de resposta.

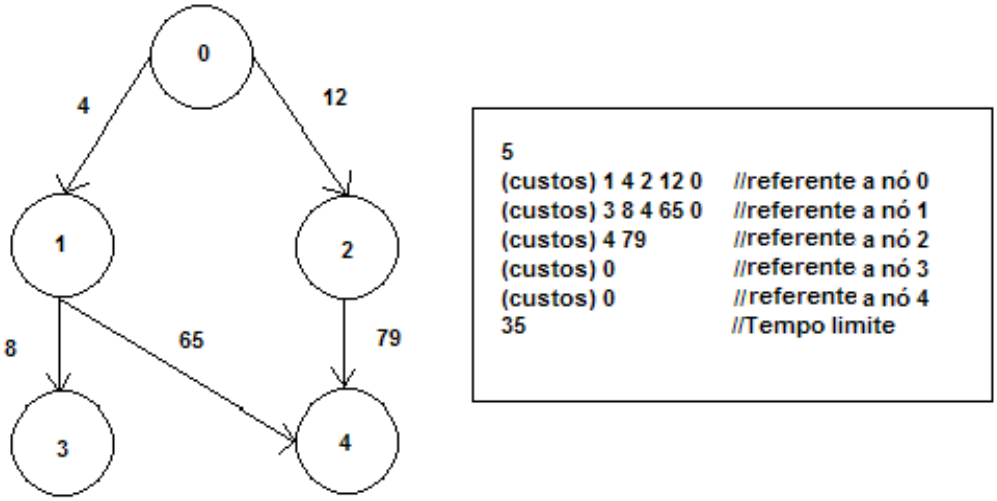

Figura 7.2: Exemplo de grafo acíclico direcionado juntamente com arquivo referente.

Após a geração do arquivo com o banco de dados, a rede neural o recebe como entrada para treinamento seguindo a metodologia proposta na Figura 7.3.

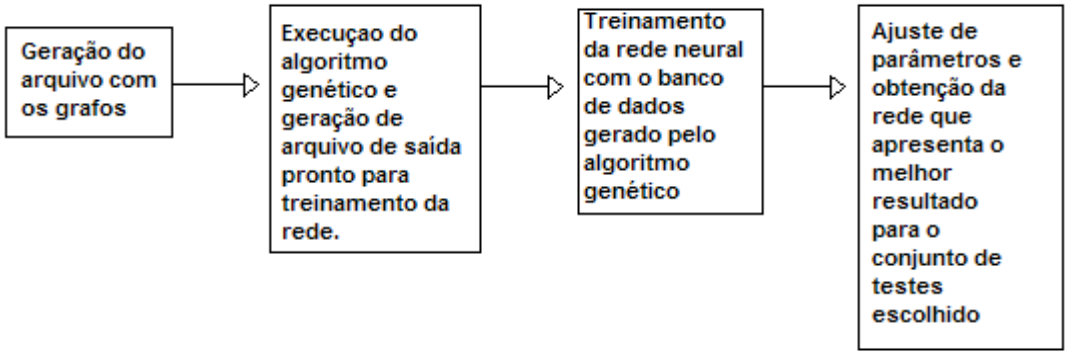

Figura 7.3: Metodologia de desenvolvimento do trabalho

7.3 Geração de Dados

A heurística escolhida para resolução do problema do particionamento hardware/software foi uma rede neural artificial. Tal heurística necessita de um treinamento, pois como explicado no capítulo dois, a rede funciona como uma abstração de um cérebro humano e em conseqüência deste fato necessita de informações iniciais para aprender e depois poder ser utilizada para resolver o problema em questão.

O treinamento da rede é feito com um banco de dados de pares entrada/saída para o algoritmo de treinamento escolhido. Portanto para que fosse possível o treinamento seria necessário que houvesse um banco de dados disponível que contivesse um estudo de vários tipos de grafos de entrada e seu particionamento encontrado por algum outro tipo de heurística.

O banco de dados procurado não foi encontrado em nenhum dos meios utilizados para pesquisa. Este fato tornou necessária a criação de um banco de dados da forma desejada.

Para a criação deste banco de dados foi necessário um estudo de qual método já implementado para resolução do problema apresentava melhores resultados e em um tempo de execução aceitável.

Segundo Hidalgo & Lanchares (1997), 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 para o grafo de entrada utilizado para testes, que é um grafo real e esta representado na Figura 8.1. Devido a ampla utilização de algoritmos genéticos para resolução deste problema

esta técnica foi escolhida para geração de dados como proposto por Hidalgo & Lanchares (1997) devido aos resultados satisfatórios que apresentou.

O algoritmo utiliza a função objetivo mostrado na Figura 7.4 e o cálculo das variáveis como mostrado na Figura 7.5. Algumas observações relevantes sobre o cálculo de custo foram encontradas em Jagadeeswari & Bhuvaneswari (2008) e o cálculo dos custos encontra-se, da forma correta, apresentado na Figura 7.5.

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

Figura 7.4: Função objetivo

Fonte: Hidalgo e Lanchares (1997)

- $t = \Sigma\ t\_hw * gene + \Sigma\ t\_sw * \neg\ gene$
- $c = \Sigma\ c\_hw * gene + \Sigma\ c\_sw * \neg\ gene$ + Communication Costs

Figura 7.5: Calculo e parâmetros da função objetivo

Fonte: Hidalgo e Lanchares (1997)

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. Os valores de *K1* e *K2* foram atribuídos experimentalmente segundo Hidalgo e Lanchares(1997).

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.

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.

A abordagem feita por Hidalgo & Lanchares (1997) não levava em conta os custos de software levando em conta que existisse sempre memória suficiente para a execução das aplicações. Porém como sugerido por Jagadeeswari & Bhuvaneswari (2008), foi feita uma modificação no calculo dos custos (*c*) para que este parâmetro pudesse ser levado em consideração.

A multiplicação pelos genes acontece devido à decisão de modelagem o que faz com que os tempos sejam calculados de forma correta. O algoritmo genético funciona de seguinte maneira: o algoritmo recebe um arquivo com o grafo similar ao da Figura 7.2. 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 cruzamento 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 Figura 7.6 ilustra a situação :

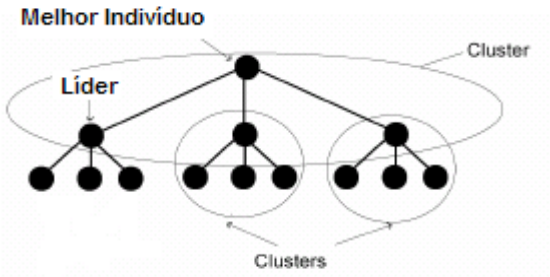

Figura 7.6: Organização da população em arvore ternária

A seleção dos pais para cruzamento 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 cruzamento 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 como mostra a Figura 7.7:

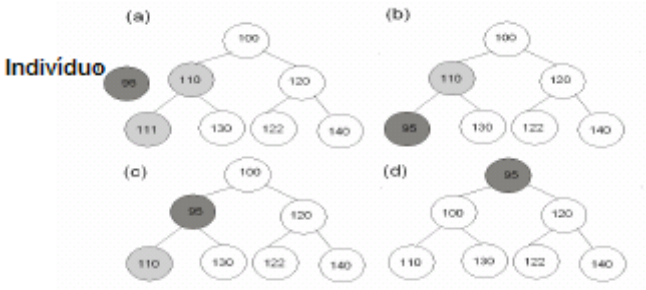

Figura 7.7: Inserção de indivíduo

Na figura acima o novo indivíduo de valor 95 é inserido na árvore. Como seu valor é menor que todos os outros valores da árvore, ele é inserido em seu *cluster* e toda a árvore é reorganizada levando o indivíduo ao nó raiz. Após não haverem mais inserções, ou seja quando a população convergir, a árvore reorganizada é extinta sobrando apenas o melhor indivíduo e as outras posições são preenchidas novamente com indivíduos aleatórios e o processo recomeça.

A taxa de cruzamento utilizada é de 0,7, ou seja, de uma população inicial são gerados 70% de novos indivíduos fruto do cruzamento. 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 chances de ocorrer mutação.

O critério de parada utilizado para o algoritmo genético é o tempo. Um tempo de cinco segundos foi definido para a busca das soluções.

A partir destes parâmetros o algoritmo é executado e os indivíduos são selecionados pela função objetivo apresentada até que o tempo limite estabelecido seja alcançado e o indivíduo com custo menor, dentro do tempo limite de projeto, é apresentado como solução do particionamento.

O banco de dados foi criado da seguinte maneira: valores de custos aleatórios limitados serão atribuídos aos nós do grafo e novas comunicações, também aleatórias porém limitadas, serão criadas, gerando assim grafos similares ao grafo de entrada proposto por Hidalgo & Lanchares (1997), que trata-se de um circuito real chamado pelos autores de *JAM1*. As limita-

ções dos valores aleatórios, que surgiram de análise de grafos de tarefas contidos em vários artigos, asseguraram que serão criados apenas grafos passiveis de implementação. Após criados os grafos segundo algoritmo presente na Figura 7.8, eles serão lidos pelo algoritmo genético e os vetores binários dos indivíduos serão armazenados juntamente com os grafos de entrada. Estes pares, grafos de entrada e vetores de saída, serão utilizados para treinamento da rede. As limitações para os valores aleatórios foram feitas a partir da análise de grafos encontrados na literatura juntamente com o conhecimento empírico do pesquisador.

```
14 srand (time (0));
15
16 for (int k = 0; k < numeroGrafos; k++){
17
18   for (int i = 0; i < 71; i++) grafo[i] = 0;
19
20   grafo[0] = rand()%100 + 35;// calculo do tr
21
22   for (int i = 1; i < 71; i = i + 10) grafo[i]=(rand()%15) + 1; // tempo de hardware
23
24   for (int i = 2; i < 71; i = i + 10) grafo[i]=(rand()%191) + 10; // custo de hardware
25
26   for (int i = 3; i < 71; i = i + 10) grafo[i]=(rand()%15) + 8; // tempo de software
27
28   for (int i = 4; i < 71; i = i + 10) grafo[i]=(rand()%91) + 10; // custo de software
29
30   //modelagem de custos
31   //6 posições onde cada posição é o custo da comunicação para um nó da seguinte forma:
32   //custo nó 1 2 3 4 5 6
33
34   for (int i = 0; i < 6; i ++){
35
36       numeroComunicacoes = (rand()%(6 - i)) + 1;
37
38       for (int j = 0; j < numeroComunicacoes; j++){
39
40           posicaoRandomica = (rand()%(6 - i))+ (11*i + 5);
41           grafo[posicaoRandomica] = (rand()%100) + 1;
42
43
44       }
45   }
```

Figura 7.8: Geração aleatória de grafos

7.4 A Rede Neural

Após criado o banco de dados a etapa de desenvolvimento da rede neural pode ser iniciada. O desenvolvimento da rede neural deveria ser em uma linguagem de programação que permitisse uma implementação rápida e eficiente. Após várias pesquisas e análises a biblioteca

de implementação de redes neurais artificiais escolhida foi a 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 Nissen(2005) pode ser encontrada a documentação referente às inúmeras 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 semelhante ao da Figura 7.9, e utiliza a função *fann_train_on_file* . Os três primeiros números indicam quantos pares de treinamento há no arquivo, ou seja, quantos pares entrada saída desejada o arquivo possuí, seguido do número de entradas por par e pelo número de saídas por par, todos separados por espaços. Da segunda linha em diante são colocadas as entradas em uma linha e as saídas desejadas para as entradas na linha seguinte sendo os dados também separados por espaços.

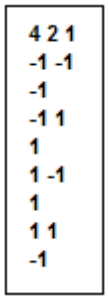

Figura 7.9: Exemplo de arquivo de entrada para a rede neural

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 armaze-

nada 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 Nissen (2005), 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. Um exemplo de arquivo de entrada pode ser visto na Figura 7.10 e um esboço da rede na Figura 7.11.

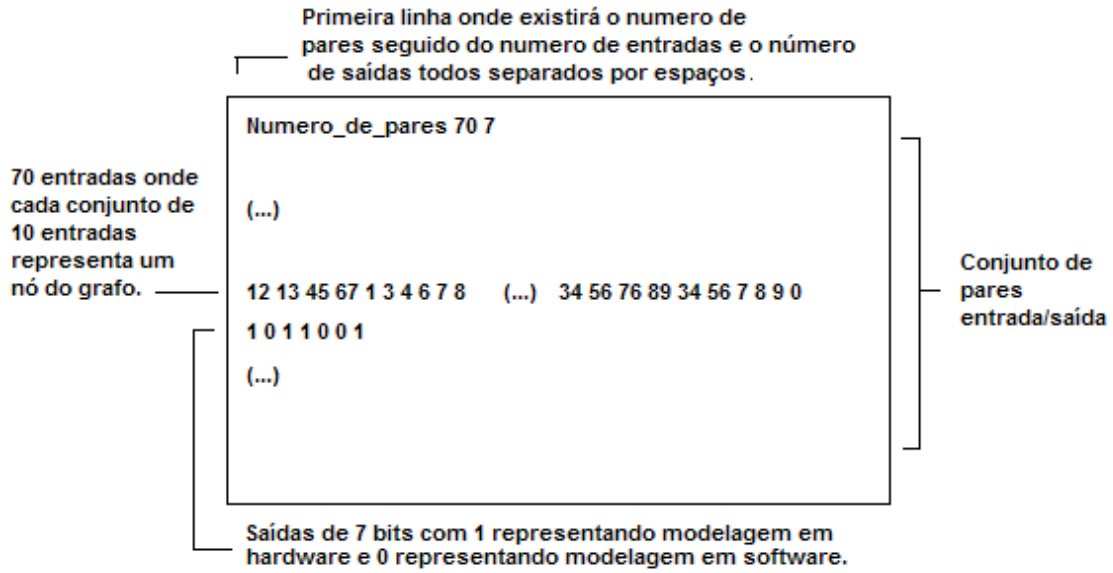

Figura 7.10: Exemplo de arquivo de entrada

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 (*Resilient backpropagation*), sem momento com

tamanho do passo 0.5, o número máximo de épocas de treinamento 15 mil. Estes parâmetros são escolhidos pelo pesquisador através de seu conhecimento.

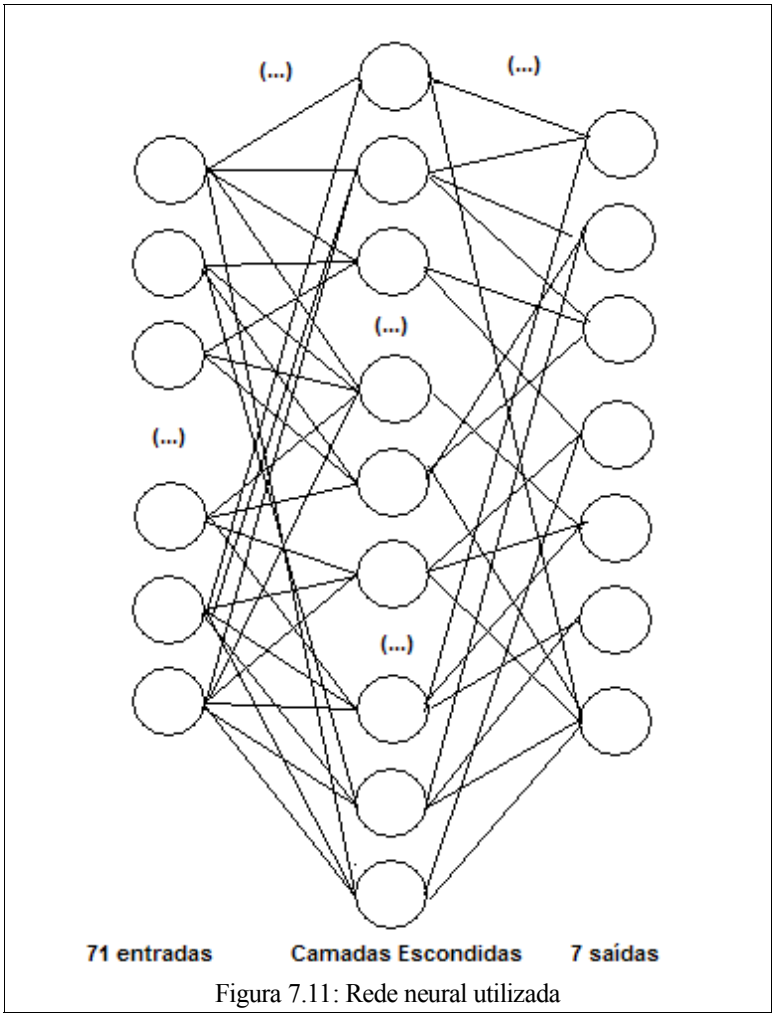

Figura 7.11: Rede neural utilizada

7.5 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.*

8 RESULTADOS E DISCUSSÃO

8.1 Resultados

O algoritmo genético utilizado para geração do banco de dados apresentou resultados de particionamento semelhantes aos resultados apresentados em Hidalgo & Lanchares (1997), para o arquivo de entrada referente ao grafo da Figura 8.1, como pode ser visto na tabela 8.1 e na Figura 8.2. 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 Hidalgo & Lanchares (1997), a implementação do circuito representado por este grafo considera que sempre haverá memória disponível para a execução dos algoritmos.

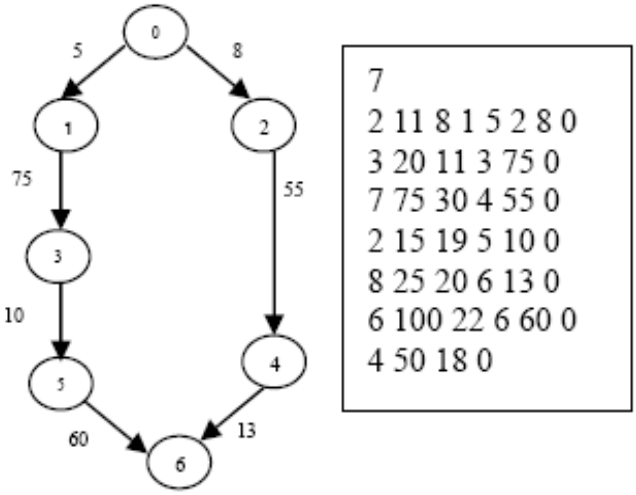

Figura 8.1: Grafo acíclico orientado e arquivo de entrada de dados

Fonte: Hidalgo & Lanchares (1997)

Na Tabela 8.1 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*), custos dos particionamentos encontrado pelo algoritmo *Modified Fidducia-Matheyses* (*Cmf*) e por fim os

custos dos particionamentos encontrados pelo algoritmo genético (*Cag*) utilizado por Hidalgo & Lanchares (1997) seguido dos indivíduos encontrados pelo algoritmo genético em cada caso.

Tabela 8.1: Resultados do algoritmo genético utilizado por Hidalgo & Lanchares (1997)

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

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

O gráfico da Figura 8.4 mostra a evolução do erro quadrático com relação ao número de épocas no treinamento da rede que apresentou melhores resultados, cujos parâmetros 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 8.1 gerou os resultados da Figura 8.3. O espaço do particionamento gerado pela rede neural não pôde ser desenhado devido ao número elevado de dimensões.

A figura 8.2 compara os resultados entre algoritmos genéticos e a figura 8.3 compara os resultados da rede neural implementada e o algoritmo genético. 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, ou seja, custos de implementação válidos para o tempo de resposta limite válido.

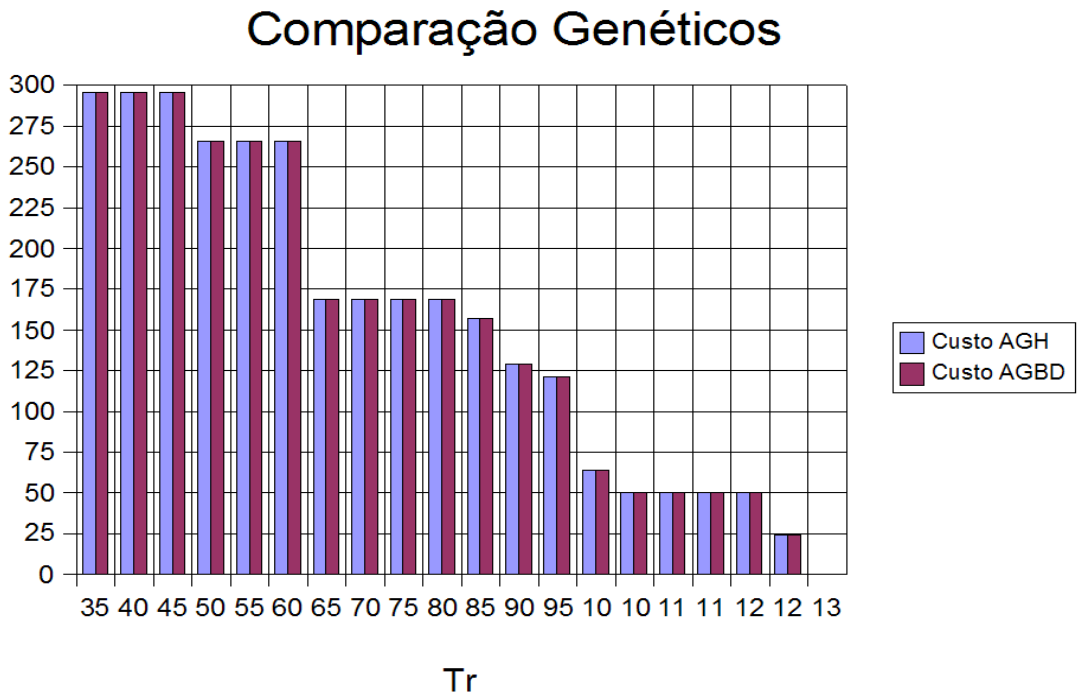

Figura 8.2: Gráfico da comparação entre os algoritmos genéticos

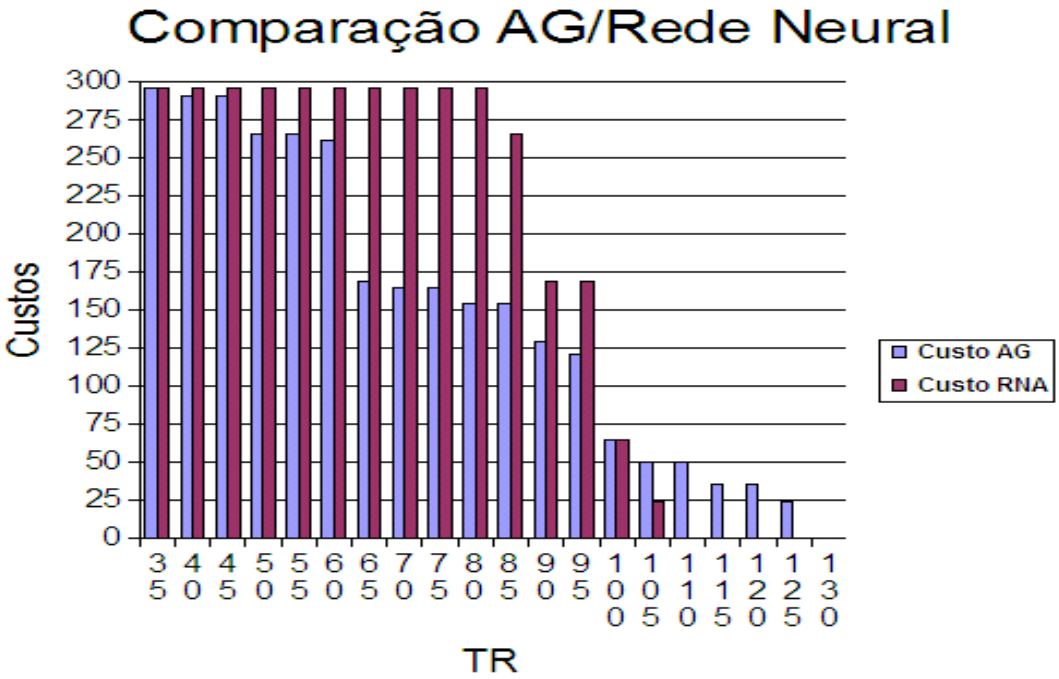

Figura 8.3: Comparação de custos entre algoritmo genético e RNA

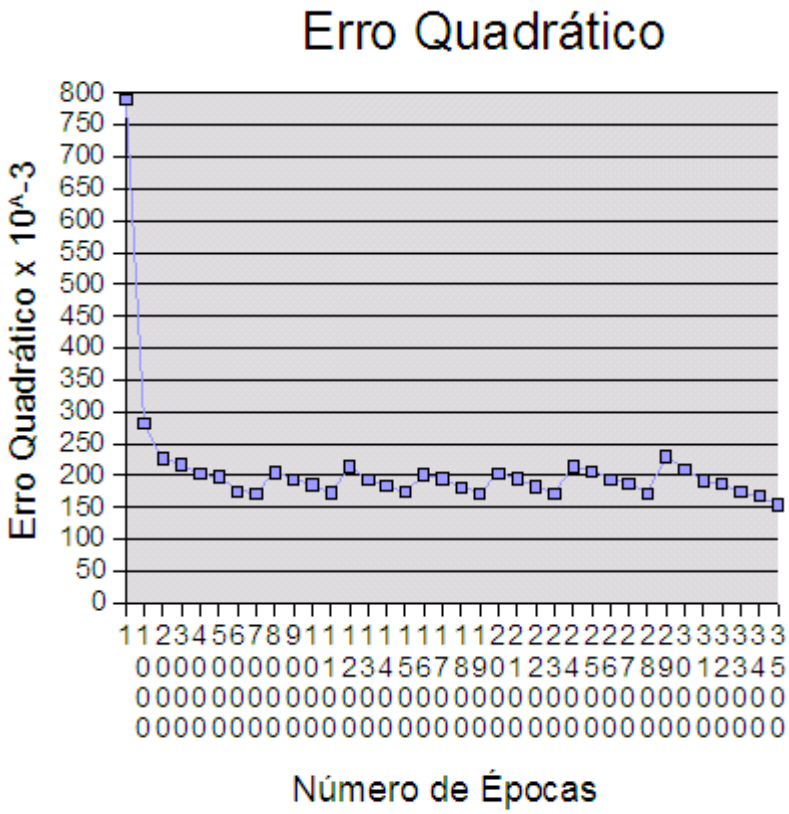

Figura 8.4: Gráfico do erro quadrático de treinamento

8.2 Discussão

A algoritmo genético utilizado para geração do banco de dados apresentou resultados semelhantes ao algoritmo de Hidalgo & Lanchares (1997) como mostrado no início do capítulo. Os resultados garantiram tempo de geração de um banco de dados melhor, com particionamentos semelhantes, em comparação ao banco de dados que seria gerado caso o algoritmo de geração fosse o algoritmo genético simples.

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.

O problema do particionamento hardware/software tem sido amplamente atacado com técnicas heurísticas conhecidas como busca T.A.B.U., 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 a 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 possui 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 escolhidas para a modelagem que também não estejam presentes no treinamento.

8.2.1 Análise Detalhada do Banco de Dados

O banco de dados gerado apresentou algumas 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 serem 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 numericamente 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 prejudicadas pelo fato do banco de dados apresentar exemplos de dois tipos em quantidade notadamente maior.

8.3 Trabalhos futuros

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 Dick, Rhodes & Wolf (1998) 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.

9 CONCLUSÃO

Durante o desenvolvimento do trabalho, foi possível perceber que a rede neural possui uma característica de generalização que pode ser útil no desenvolvimento do problema. Esta caraterística tornou-se evidente devido à tendencia das respostas da rede aos particionamentos que estão presentes em grandes quantidades no banco de dados.

A representação por grafos acíclicos direcionados mostrou-se eficiente por simplificar a modelagem do sistema e assemelhar-se à definição formal do problema.

O resultado apresentado pelo algoritmo genético na geração do banco de dados mostrou que a técnica utilizada para seleção 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 no sentido de influenciar de forma direta o treinamento e necessita de balanceamento em sua geração, ou seja, é necessário que existam exemplos em quantidades semelhantes de cada particionamento.

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. O tempo de resposta é fator crítico já que os projetos de sistemas embarcados atualmente priorizam o tempo de desenvolvimento.

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 forneceram resultados satisfatórios para a solução do problema em questão, ou seja, para o tamanho de grafo adotado, redes com um número maior de neurônios ou de camadas não apresentou melhores resultados..

A utilização da biblioteca FANN proporciona uma redução de até 150 vezes no tempo de treinamento comparado a treinamentos que utilizam toolbox ou IDE's. Este tempo reduzido permitiu que um número maior de testes fossem realizados. O trabalho também mostrou que a rede *multi-layer* perceptron é recomendável para a resolução deste tipo de problema.

REFERÊNCIAS BIBLIOGRÁFICAS

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.

ARATÓ,P.; MANN, Z. Á.;ÓRBAN,A.. Algorithmic Aspects of Hardware/Software Partitioning. **ACM Transactions on Design Automation of Electronic Systems**, , v. 10, n. 1, p. 136-156, 2005.

ATALLAH, M.J.. **Algorithms and Theory of Computation Handbook**. Primeira Edição. : Purdue University, 1998. 1265 p.

BACK, T.; SCHWEFEL, H.. Evolutionary computation: An overview. **Proceedings of IEEE International Conference on Evolutionary Computation**, , v. 1, n. , p. 20--29, 1996.

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

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

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.

FAUSETT, L.. **Fundamentals of Neural Networks - archtectures, algorithms and applications**. Primeira Edição. : Prentice Hall, 1994. 461 p.

GERSTNER, W; KISTLER, W. M.. **Spiking Neuron Models - Single Neurons, Population, Plasticity**. Primeira Edição. : Cambridge, 2002. p.

HAYKIN, S.. **Neural Networks - A Comprehensive Foundation**. Segunda Edição. : Pearson, 1999. 823 p.

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.

HOUSSAIN, T. S.. An introduction to Evolutionary Computation. **Department of Computing and Information Science**, , v. , n. , p. , 1998.

JAGADEESWARI, M.;BHUVANESWARI, M.C. A Fast Multi-Objective Genetic Algorithm for Hardware-Software PartitioningIn Embedded System Design. **ICGST-AIML Journal**, , v. 8, n. 2, p. 1 - 7, 2008.

JUNG, C. F.. **Metodologia para Pesquisa & Desenvolvimento : aplicada a novas tecnologias, produtos e processos**. Primeira Edição. : Axcel Books do Brasil, 2004. p.

KRÖSE, B; VAN der SMARQT, P.;. **An Introduction to Neural Networks**. Oitava Edição. : The University of Amsterdan, 1996. 134 p.

LEWIS, H. R.;PAPADIMITRIOU , C. H.. **Elements of The Theory of Computation**. Segunda Edição. : Prentice Hall, 1998. 375 p.

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.

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.

QUAN,G.;HU, X.;GREENWOOD,G.;. Preference-Driven Hierarchical Hardware/Software Partitioning. , , v. , n. , p. , .

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

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

THOMAS, D. E.; ADAMS J.K.; SCHMIT, H.. A Model and Methodology for Hardware-Software Codesign. **IEEE design ans test of computers**, , v. 1, n. , p. 6 - 15, 1993.

WIANGTONG, T.; CHEUNG, P.Y.K.; LUK, W.;. Comparing Three Heuristic Search Methods for Functional Partitioning in Hardware-Software Codesgin. **Design Autmation for Embedded Systems**, , v. 10, n. 1, p. 425-449, 2002.

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

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.

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.