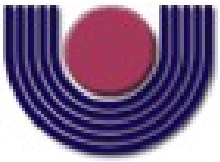

UNIOESTE – Universidade Estadual do Oeste do Paraná

CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS

Colegiado de Ciência da Computação

Curso de Bacharelado em Ciência da Computação

VIPEX: Um Ambiente Visual para Exploração e Otimização de Arquiteturas Multiprocessadas

Alexandre Specian Cardoso

CASCAVEL

2010

ALEXANDRE SPECIAN CARDOSO

VIPEX: UM AMBIENTE VISUAL PARA EXPLORAÇÃO E OTIMIZAÇÃO DE ARQUITETURAS MULTIPROCESSADAS

Monografia apresentada como requisito para obtenção do grau de Bacharel em Ciência da Computação, do Centro de Ciências Exatas e Tecnológicas da Universidade Estadual do Oeste do Paraná - Campus de Cascavel

Orientador: Prof. Dr. Marcio Seiji Oyamada

CASCAVEL

2010

ALEXANDRE SPECIAN CARDOSO

VIPEX: UM AMBITENTE VISUAL PARA EXPLORAÇÃO E OTIMIZAÇÃO DE ARQUITETURAS MULTIPROCESSADAS

Monografia apresentada como requisito para obtenção do Título de *Bacharel em Ciência da Computação*, pela Universidade Estadual do Oeste do Paraná, Campus de Cascavel, aprovada pela Comissão formada pelos professores:

Prof. Dr. Marcio Seiji Oyamada (Orientador)
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Dr. Clodis Boscariolli
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Dr. Adair Santa Catarina
Ciência da Computação,
UNIOESTE

Cascavel, 04 de Novembro de 2010.

Este trabalho é dedicado a minha família, em especial meus pais Saulo e Silvana, que me apoiou e aturou todos estes anos, meus colegas e amigo, que tornaram tempo difíceis em tempos de alegria e a Diana, que esteve sempre comigo mesmo quando achei que tudo estava perdido, ajudando em minhas decisões.

AGRADECIMENTOS

Agradeço primeiramente a Deus que me deu esta oportunidade e me guiou por este caminho, sempre me abençoando, e por várias vezes, me pondo à prova. Também à minha famìlia, em especial ao meu pai Saulo e a minha mãe Silvana, que esteve todo este tempo ao meu lado dando apoio e mudando algumas de minhas decisões, que por várias vezes, foram de desistir.

Agradeço também a Diana, que em muitos momentos me ajudou nesta caminhada, sempre compreensiva quando necessitei estar ausente por conta de obrigações da faculdade. Quero também deixar meus agradecimentos às amizades que cativei durante este período de graduação, amigos que irão permanecer mesmo depois que tudo acabar, em especial o Jhonata, Chuck e Adriano, que estiveram constantemente comigo, principalmente durante os últimos meses, que foram os mais difíceis. Assim como aos meus parceiros, com quem aprendi muito, sem citar nomes pois seria inevitável não esquecer de alguém mas entre eles em especial ao Andy, que sempre me apoiou.

Quero deixar um agradecimento especial ao meu orientador Márcio, que teve tanta paciência e compreensão, mesmo em vezes que não mereci, e mostrou a luz no fim do túnel em várias ocasiões que para mim pareciam não ter solução.

Enfim, agradeço a todos que cruzaram meu caminho, me fazendo crescer e aperfeiçoar tanto minhas habilidades acadêmicas quanto meu caráter pessoal.

Alexandre “Pardal” Specian Cardoso

Lista de Figuras

Lista de Abreviatura e Siglas

ABS	*Anti-lock Braking System*
AG	Algoritmos Genéticos
API	*Application Programming Interface*
CMOS	*Complementary metal-oxide-semiconductor*
DCT	*Discrete Cosine Transform*
DVD	*Digital Video Disc*
DVS	*Dinamic Voltage Scaling*
IA	Inteligência artificial
L1	*Level* 1
L2	*Level* 2
MLP	*Multi-Layer Perceptron*
MPSoC	*Multiprocessor System-on-a-Chip*
NoC	*Network-on-a-Chip*
OVP	*Open Virtual Plataform*
QoS	Q*uality of Service*
RAM	*Random Access Memory*
RNA	Redes Neurais Artificiais
SE	Sistemas Embarcados
SoC	*System-on-a-Chip*
UML	*Unified Modeling Language*
VHDL	*VHSI Hardware Description Language*
VHSI	*Very High Speed Integration Circuit*
VIPEX-VMP	*Virtual Platform Exploration - Vipro-MP*
VIPRO-MP	*Virtual Prototype for Multiprocessor Architecture*

Sumário

Resumo

A crescente complexidade no projeto de sistemas embarcados requer que novos tipos de ferramentas sejam desenvolvidos. Plataformas virtuais são desenvolvidas com o objetivo de facilitar a avaliação de arquiteturas, visando otimizar o desempenho, consumo de potência entre outros requisitos de um sistema embarcado. Para diminuir o tempo de exploração do espaço de projeto, métodos heurísticos podem ser utilizados par realizar otimizações em arquiteturas e diminuir o número de testes e simulações necessárias. Neste trabalho foi desenvolvida um interface gráfica para a plataforma VIPRO-MP, uma plataforma virtual para exploração de arquiteturas multiprocessadas. Adicionalmente, foram integrados Algoritmos genéticos e Redes Neurais Artificiais do modelo MLP para permitir uma rápida exploração do espaço de projeto e a otimização da configuração base definida pelo usuário. Os testes demonstraram que os Algoritmos Genéticos permitem minimizar o número de arquiteturas simuladas quando comparado com a simulação exaustiva, obtendo resultados similares. As Redes Neurais Artificiais não foram capazes de substituir o simulador, tornando necessário a realização de testes com outras configurações e modelos de Redes Neurais Artificiais.

Palavras-Chave: Sistemas Embarcados, Arquitetura Multiprocessada, Plataforma Virtual, Interface Gráfica, Metodos Heurísticos, Algoritmos Genéticos, Redes Neurais Artificiais.

Capítulo 1

Introdução

Nos dias atuais é comum se deparar com aparelhos eletrônicos com algum processador embutido, denominados Sistemas Embarcados (SE). Estes dispositivos operam de forma semelhante a um computador de propósito geral, porém desempenham tarefas específicas.

Segundo Marwedel (2006), um SE é um sistema de processamento de informações que é incorporado em um produto maior e, normalmente, não é diretamente visível pelo usuário. Temos como exemplos de sistemas embarcados celulares, aeronaves, eletrodomésticos entre outros. No princípio, o desenvolvimento de SE era voltado para aplicações menos complexas, pois sua estrutura não suportava rotinas com taxa de processamento elevado ou consumia potência demasiadamente.

Com a evolução da tecnologia e o cumprimento da lei de Moore (1965), foi possível desenvolver aparelhos menores e mais potentes. Um exemplo de avanço tecnológico e do poder de miniaturização é o protótipo lançado pela Intel, onde um único chip contém 80 processadores e alcança o desempenho de 1 *terraflop,* (Intel, 2010).

O desenvolvimento de *hardware* ou *software* para SE possui algumas diferenças em relação ao desenvolvimento de sistemas computacionais de propósito geral. Ao se desenvolver um SE, deve-se pensar na arquitetura que será utilizada, pois apresenta requisitos que tornam o sistema restrito, etapa que não se faz necessária no desenvolvimento de um sistema computacional de propósito geral. Abaixo, alguns requisitos de SE, que serão detalhados nas seções a seguir:

a) Desempenho;
b) Potência;
c) Custo de produção;
d) *Design*;
e) Tolerância as falhas e;
f) Tempo de projeto.

1.1 Potência e Desempenho

O desempenho tende a variar conforme a necessidade da aplicação que o SE comportará. Aplicações multimídia requerem mais desempenho, ao contrário de arquiteturas voltadas a processamento de texto que necessitam de um desempenho menor. O desempenho está diretamente relacionado ao requisito de potência, uma vez que processadores com maior desempenho tendem a consumir mais potência. Este é um atributo chave em um SE, pois a grande maioria desses dispositivos é alimentada por baterias; um grande consumo de energia resulta em um SE que necessita ser recarregado constantemente, fato incômodo em alguns aparelhos como celulares e *players* e se tornam um problema grave em sistemas mais críticos.

Nos aparelhos alimentados por baterias deve-se voltar a atenção para duas características limitantes ao se desenvolver uma aplicação para SE. Primeiramente a carga de uma bateria é limitada, fato que obriga o SE a reduzir a energia consumida. A segunda característica é relacionada à potência consumida, criada pelo limitador de potência consumida para que a bateria não queime, obrigando o SE a não utilizar muita potência de uma vez.

Entretanto, a tarefa de decidir entre consumir mais por menos tempo ou consumir menos por mais tempo não é trivial. Considerando que a energia consumida por um aparelho, E, é o resultante da integral da potência consumida, P, com relação ao tempo, t, como demonstra a Equação 1, pode-se aumentar a freqüência do processador para terminar a tarefa mais rapidamente, resultando em um alto consumo de potência por pouco tempo ou optar-se por diminuir a freqüência do processador, diminuindo assim o consumo de potência.

$$E = \int P \, dt \qquad (1)$$

Porém, prolongando o tempo de realização da tarefa, como demonstra a Figura 1.1, onde a linha P1 representa um processador com maior freqüência e a linha P2 representa um processador com menor freqüência.

A escolha de qual abordagem deve ser utilizada é definida na exploração do espaço de projeto; existem aplicações que necessitam ser concluídas rapidamente, outras exigem consumo reduzido e há aplicações que devem ser executadas com o melhor balanceamento entre esses dois requisitos.

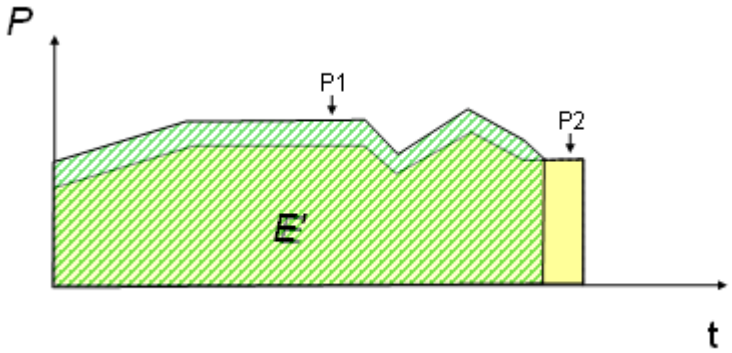

Figura 1.1: Consumo de Energia do SE.

Para melhorar a relação custo benefício entre o desempenho e o consumo de energia, empresas vêm desenvolvendo processadores que visam obter o máximo poder de processamento com o mínimo consumo de energia. Algumas técnicas utilizadas para reduzir o consumo de potência e ampliar o desempenho são: DVS (*Dinamic Voltage Scaling*), *Clock Gating* e *Power State*.

A técnica DVS, geralmente utilizada em arquiteturas alimentadas por baterias, consiste em simplesmente reduzir a freqüência do processador, conseqüentemente, reduzindo o desempenho do mesmo, e elevar o *clock* quando necessário. Como exemplo de DVS, pode-se citar o processador Intel PXA250 (2010), baseado na arquitetura ARM (2010), que opera normalmente em uma freqüência de 400MHz; porém, o escalonador de freqüências do Intel PXA250 pode alterá-lo para uma freqüência menor, diminuindo o consumo de potência. O Intel PXA250 possui um baixo consumo de potência, indo de 0.001 até 1,6 *W*.

O *clock gating* é utilizado em circuitos síncronos e tem como idéia base desativar partes do circuito que não estão sendo utilizadas. Ao desativar uma parte, os *flip-flops* que atuam no setor desativado têm seu consumo de energia diminuído para zero, gerando assim uma economia significativa no SE.

O *Power State*, também conhecido como *Operational State*, caracteriza-se por desligar os componentes quando não são utilizados, porém, diferencia-se do *clock gating* por estabelecer três estados para os quais os componentes podem ser mapeados, *Run*, *Idle* e *Sleep*. No estado *Run* o componente está sendo utilizado normalmente, já no estado *Idle*, o *clock* está habilitado na CPU, porém os componentes periféricos permanecem ativos. No estado *Sleep*, a CPU é quase totalmente desligada, deixando ativos apenas as instruções de entrada e saída e o manipulador de interrupções (BENINI, BOGLIOLO e DE MICHELI, 2000). A Figura 1.2

demonstra a energia consumida em cada estado, assim como o tempo de transição.

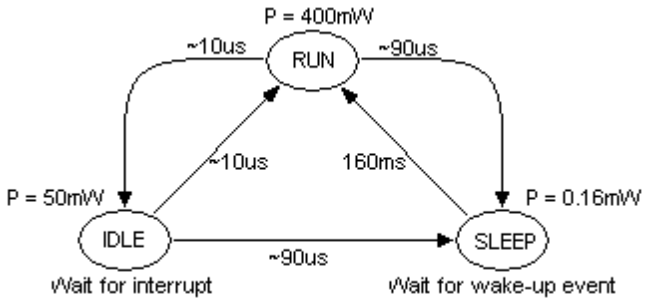

Figura 1.2: *Power State* em um processador StrongARM (OYAMADA, 2007)

Segundo Girvargis e Varid (2010), a definição de potência consumida por um circuito CMOS (*Complementary Metal-Oxidate-Semiconductor*) é obtida por meio da Equação 2.

$$P = (C \times A \times F \times V^2)/ 2 \qquad (2)$$

onde C é a capacitância do chaveamento, A é a atividade média do chaveamento, F é a freqüência aplicada no circuito e V é a tensão aplicada para que ocorra o chaveamento das portas lógicas. Nas arquiteturas CMOS a freqüência é linearmente proporcional a tensão; aliando-se esse fato à Equação 2, pode-se afirmar que alterações na freqüência têm impacto de proporções quadráticas na potência consumida pela arquitetura. Essa peculiaridade dos circuitos CMOS torna-os uma ótima opção no desenvolvimento de SE, pois proporcionam condições de cumprir as restrições de desempenho e potência consumida.

Devido a requisitos restritos, geralmente o projeto de um sistema embarcado envolve o desenvolvimento do *hardware* e do *software*. Sendo assim, é necessário esperar o *hardware* ser desenvolvido para só então desenvolver o *software*, realizar os testes e verificar seu desempenho. Existe também a possibilidade de que o protótipo desenvolvido não seja suficiente para acomodar o *software* para o qual foi projetado, resultando em um atraso significativo no projeto.

Desta forma, é necessário explorar o espaço de projeto visando obter a melhor configuração da arquitetura de *hardware* para uma determinada aplicação, principalmente para projetos com processadores de alto desempenho em SE. Tal fase auxilia os projetistas na

detecção e resolução de problemas decorrentes do projeto da arquitetura. Segundo Carro e Wagner (2003), a fase de exploração do espaço de projeto deve encontrar a solução para três questões: 1) Quantos e quais processadores e blocos dedicados serão necessários? 2) Qual o mapeamento ideal entre funções e componentes de *hardware*? 3) Qual a estrutura de comunicação ideal para conectar os componentes entre si?

Tendo em posse os requisitos necessários para o SE, pode-se fazer uso de estimadores que tenham condições de informar, com certo grau de precisão, o comportamento do sistema, estimando dados como desempenho e consumo de potência. O número de soluções arquiteturais possíveis para uma mesma funcionalidade é imensa, tornando o trabalho de varrer essas possibilidades em busca das melhores soluções demasiadamente custoso, razão pela qual esta etapa é geralmente substituída pela escolha de uma arquitetura já conhecida e que se aplique a funcionalidade do SE, composta por processadores e componentes já conhecidos e interligados por uma rede pré-definida.

No intuito de facilitar a exploração do espaço de projeto, plataformas virtuais são utilizadas para simular as arquiteturas em um estado inicial de projeto e, consequentemente, obter uma estimativa de desempenho antes que o *hardware* seja construído.

Um exemplo de plataforma virtual é o VIPRO-MP (GARCIA, SCHUCK e OYAMADA, 2009), que permite ao projetista criar uma arquitetura multiprocessada e realizar a simulação da mesma, gerando sua avaliação. Este ambiente virtual possui uma vasta gama de configurações de arquiteturas, permitindo variar características como número de processadores, as características dos processadores e o tamanho das memórias.

1.2 Custo de Produção, *Design*, Tolerância a Falhas e Tempo de Projeto

O desempenho e a potência consumida, assim como os componentes utilizados, têm grande influência no custo de produção. Um alto custo de produção, consequentemente, eleva o custo para o consumidor, fato que pode reduzir a aceitação do produto desenvolvido no mercado.

O *design* visa obter um produto com uma forma e tamanho que agrade o consumidor. Deve-se ter em mente a diferença entre as pessoas e o público-alvo que o produto desenvolvido deseja alcançar. Em algumas ocasiões, como celulares, esse é um fator que

influencia o usuário na escolha de qual aparelho irá adquirir.

A tolerância a falhas é um requisito de extrema importância, principalmente em sistemas mais críticos, como monitores de trem, controladores de freio ABS e de aeronaves, onde, ainda que um componente apresente falhas, o mesmo deve ser capaz de mascará-la evitando que o sistema pare de funcionar.

As restrições do SE devem ser bem pensadas e analisadas, garantindo assim um equilíbrio no produto final, para se obter uma aplicação que atenda a todos os requisitos com o máximo de desempenho da arquitetura. Porém, o projeto de um SE deve ser feito dentro do menor tempo possível, pois esse é um fator de fundamental importância ao se determinar a aceitação e o lucro obtidos com o produto. Um atraso no lançamento do produto no mercado pode ocasionar perdas no retorno financeiro como mostra a Figura 1.3, a qual exemplifica com clareza a curva mercadológica de um produto a partir de sua introdução no mercado e o lucro perdido pelo atraso.

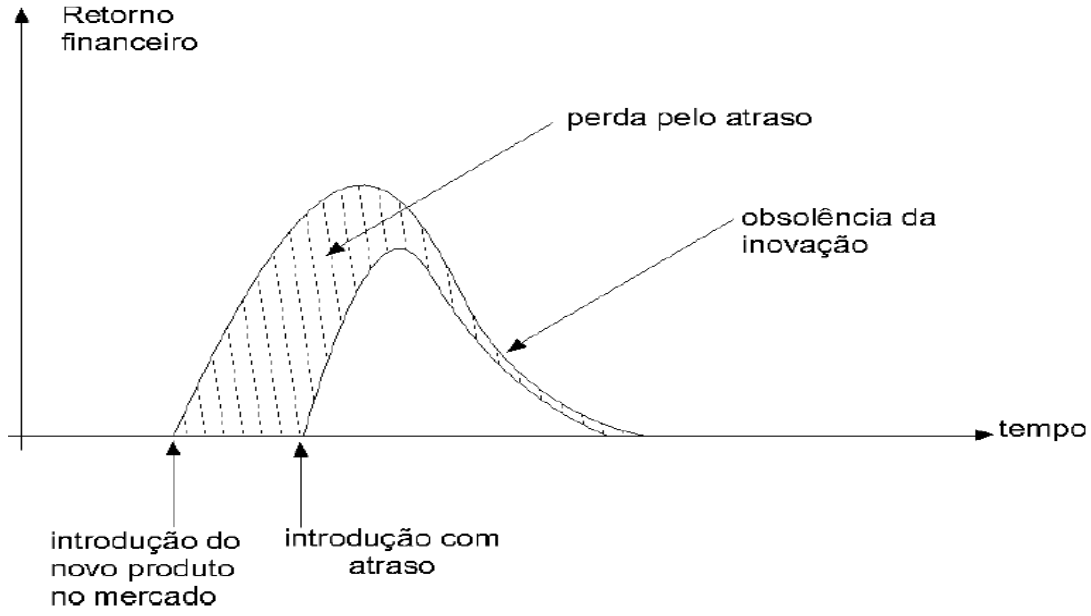

Figura 1.3: Lucro obtido a partir do lançamento de um produto com e sem atraso no mercado (CARRO e WAGNER, 2003)

1.3 Fluxo de Projeto de um SE

Os SE, em seu âmbito geral, possuem um elevado nível de complexidade, tornando necessário iniciar o projeto fazendo uma especificação em alto nível de abstração. Essas

especificações são feitas, geralmente, através de uma linguagem ou formalismo adequado, como *SystemC* e *StateCharts*; são elaboradas em um nível no qual nenhuma decisão quanto à implementação ou componentes de *hardware* e *software* foi tomada, e deve ser preferencialmente utilizadas para fins de validação (CARRO e WAGNER, 2003). Tais especificações auxiliam o projetista na detecção de erros que podem causar um atraso significativo no lançamento do produto.

É importante ressaltar que o projeto de um SE baseado em plataforma tem seu fluxo um pouco modificado, uma vez que se diferencia do projeto de um SE baseado em componentes pelo fato de não possuir o tempo de espera para que o *hardware* fique pronto, pois se baseia em plataformas já existentes.

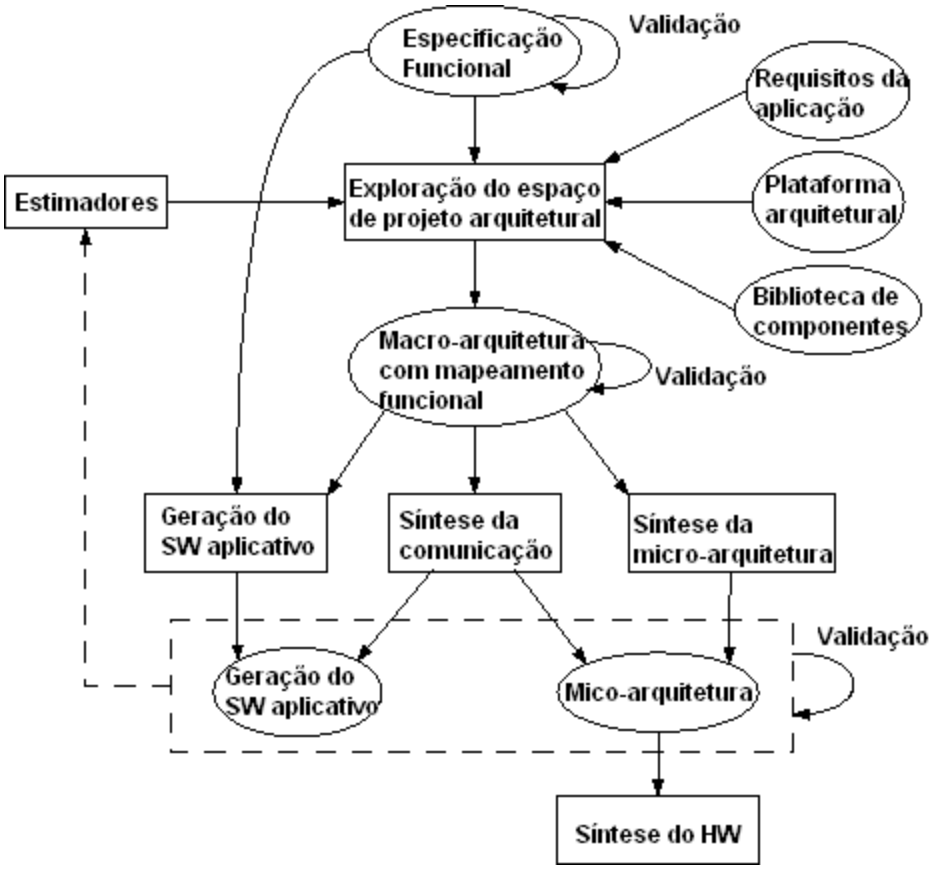

Figura 1.4: Fluxo de Projeto de um Sistema Embarcado (CARRO e WAGNER, 2003).

A Figura 1.4 demonstra o fluxo completo do projeto de um SE, sendo considerado o ideal, porém ainda não é implementada inteiramente por nenhum ambiente comercial de softwares. Esse fluxo será detalhado na Seção 2.3.

Uma alternativa que está facilitando o desenvolvimento de SEs são os chamados *System-on-a-Chip* (SoC), uma vez que suas estruturas podem conter processadores, memórias, interfaces para periféricos e blocos dedicados. Os componentes de um SoC são interligados através de uma estrutura de comunicação que pode ir desde um barramento até uma complexa rede NoC (*Network-on-a-Chip*). Uma classe em particular dos SoC, são os MPSoC (*Multi-Processor System-on-a-Chip*), que se destacam por encapsular em um único chip múltiplos processadores, podendo ser heterogêneos ou homogêneos.

Os MPSoC auxiliam os projetistas a alcançarem os requisitos necessários de um SE, pois o fato de toda esta arquitetura estar dentro do mesmo chip, elimina a necessidade de constantemente buscar informações em componentes que estão do lado de fora do chip. Uma vez que o barramento externo é menos acessado, a tendência é que o desempenho aumente e o consumo de potência diminua. A utilização de MPSoC também elimina elementos desnecessários e diminui o custo (JUNIOR e GARIBOTTI, 2007).

1.4 Objetivos

As plataformas virtuais para testes de arquiteturas multiprocessadas são de extrema relevância para o estudo e desenvolvimento de SEs. Entretanto, uma plataforma como o VIPRO-MP tem problemas na interface com o usuário, já que é operada totalmente em linhas de comando, dificultando o trabalho dos projetistas. Sendo assim, faz-se necessário uma interface gráfica que torne tal tarefa mais fácil.

Os ambientes baseados em simulações apresentam também o problema do tempo gasto com simulações; estas podem levar horas ou até mesmo dias. Visando reduzir o tempo de simulação, métodos heurísticos são amplamente utilizados para estimar a avaliação da arquitetura, dada uma aplicação, tornando o processo de exploração mais eficiente.

Esse trabalho tem como primeiro objetivo facilitar a exploração do espaço de projeto de um SE através de uma interface gráfica para a modelagem de arquiteturas multiprocessadas, que possibilitará a simulação, através da ferramenta VIPRO-MP, denominado VIPEX-VMP (*Virtual Platform Exploration - Vipro-MP*).

O segundo objetivo visa a implementação de métodos heurísticos, como Algoritmos Genéticos (AG) e Redes Neurais Artificiais (RNA), que serão acoplados à plataforma visual desenvolvida, possibilitando a otimização da arquitetura proposta pelo projetista com redução

do tempo de exploração do espaço de projeto.

1.5 Organização do Texto

Os temas abordados neste trabalho estão divididos da seguinte forma. O Capítulo 2 trata sobre o projeto de SE, explorando assuntos como fluxo de projeto, exploração do espaço de projeto e a utilização de métodos heurísticos no projeto de SE. Ainda no capítulo 2, descreve-se o que são plataformas virtuais, assim com a ferramenta VIPRO-MP e algumas plataformas visuais.

O Capítulo 3 explora a implementação da interface gráfica, dos Algorítmos Genéticos e das Redes Neurais Artificiais, bem como os problemas no desenvolvimento, testes e resultados obtidos. Por fim, o Capítulo 4 é destinado a relatar as conclusões obtidas com este trabalho e em que pontos o mesmo ainda pode ser aprimorado.

Capítulo 2

Projeto de Sistemas Embarcados

O projeto de SEs é extremamente complexo, por envolver conceitos até agora pouco analisados pela computação de propósitos gerais. Wolf (2001) cita que as questões da portabilidade, do limite de consumo de potência sem perda de desempenho, a baixa disponibilidade de memória, a necessidade de segurança e confiabilidade, a possibilidade de funcionamento em uma rede maior e o curto tempo de projeto tornam o desenvolvimento de sistemas computacionais embarcados uma área em si.

Considerando os vários requisitos, que tornam o projeto de SE uma tarefa complexa, a complexidade arquitetural e o grande número de arquiteturas a serem analisadas, é de extrema importância a utilização de alguma metodologia de projeto, objetivando a minimização de erros no produto final.

2.1 Metodologias de Projeto

As metodologias utilizadas nos dias atuais tem início com a especificação dos requisitos e funcionalidades do sistema em um alto nível de abstração, preferencialmente uma especificação funcional executável. Ao obter a descrição geral pode-se detalhar cada subnível, até obter uma descrição detalhada de todo o projeto. A utilização dessa fase permite que toda equipe tenha conhecimento do que está sendo desenvolvido em cada parte do projeto, tornando mais fácil e eficaz a comunicação entre as equipes de trabalho (WOLF, 2001).

Basicamente, as metodologias definem o fluxo de projeto que será utilizado para o desenvolvimento do produto, determinando os passos que devem ser seguidos, visando a maximização dos lucros, o menor tempo de desenvolvimento e a minimização de esforços necessários. A Figura 1.4 demonstra em detalhes um fluxo de projeto tido como ideal.

2.2 Especificação Funcional

É nesta fase que o objetivo é estudado, dando origem aos requisitos principais do SE. Ao analisar o mercado é definida a data de lançamento para o produto, visando ampliar os lucros

obtidos na venda do SE desenvolvido. São definidas também as ferramentas e técnicas que serão utilizadas no processo. Preferencialmente nesta fase, se desenvolve uma especificação funcional executável, que tem o objetivo de facilitar o processo de desenvolvimento do *hardware*. Entretanto, para se descrever os requisitos não funcionais, como desempenho e consumo de potência, faz-se necessário a elaboração de um documento extra.

Visando solucionar este problema, empresas tem se esforçado para padronizar essas especificações (CARRO e WAGNER, 2003) e, atualmente, tem-se estudado alternativas para facilitar tanto o projeto do *hardware* quanto o projeto do *software*. Dentre estas alternativas estão UML, *StateCharts*, Redes de Petri entre outros, que serão detalhadas nas seções subsequentes.

2.2.1 *StateCharts*

Foi introduzida por David Harel (1987) e é baseada no conceito de comunicação de memória compartilhada (MARWEDEL, 2006). Basicamente, são diagramas de transição de estado utilizados para modelar aspectos dinâmicos do sistema, mostrando o fluxo de um estado para outro. É utilizado por mostrar com clareza os estados atingidos por um determinado objeto ao se deparar com algum evento, assim como as respostas obtidas para tais eventos.

É composto por estados, eventos e transições, onde os estados são indicados pelos círculos, as transições pelas setas e os eventos pelas *labels* das setas. Caso um evento não seja previsto pelo projetista, o modelo irá simplesmente ignorá-lo. A Figura 2.1 traz um exemplo de diagrama de *StateChart* onde cada evento pode gerar uma saída, fato que não está demonstrado no diagrama.

A linguagem *StateChart* provê notações adicionais como *Timers*, utilizados para demonstrar atrasos no diagrama, e hierarquia que permite dividir o diagrama em partes.

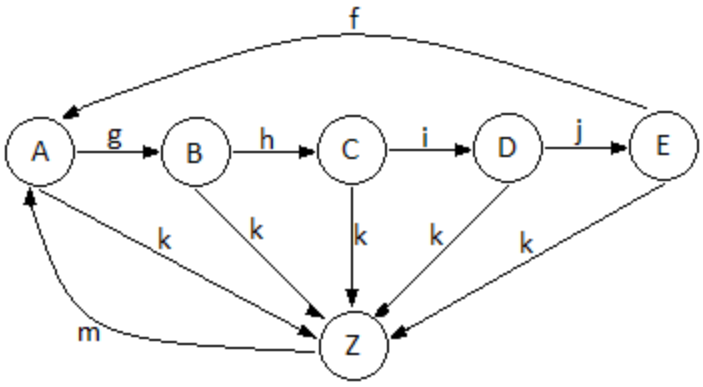

Figura 2.1: *State Diagram* (MARWEDEL, 2006).

2.2.2 Redes de Petri

Redes de Petri é outra maneira de realizar a especificação de um sistema. Foram definidas em 1962 por Carl Adam Petri e se baseiam em dependências casuais, (MARWEDEL, 2006). Há várias maneiras possíveis de se classificar uma Rede de Petri, uma da mais utilizadas é a classificação por grau de abstração.

Tem sua estrutura composta, basicamente, por três elementos: estados, ações e relações de fluxo. Os estados são utilizados para modelar componentes passivos do sistema, enquanto as ações são utilizadas para modelar componentes ativos do sistema e as relações de fluxo são utilizadas para demonstrar como se dá a transformação de um estado em outro pela ocorrência das ações no sistema (MARRAGHELLO, 2005).

As Redes de Petri podem ser representadas tanto algebricamente (abaixo), quanto graficamente, como demonstrado na Figura 2.2.

R = (E, A, F)

E = {e1, e2, e3, e4, e5}

A = {a1, a2, a3, a4}

F = {(e1, a2), (e2, a2), (e3, a1), (e5, a4), (e4, a3),
(a2, e3), (a3, e1), (a1, e2), (a4, e4), (a1, e5)}

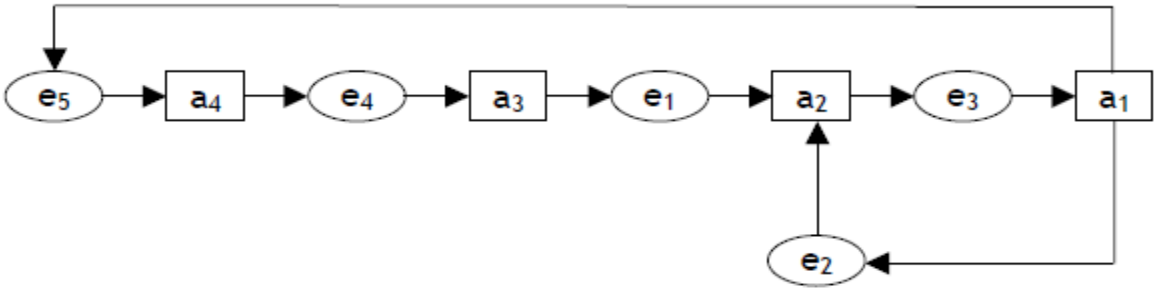

Figura 2.2: Redes de Petri representada graficamente (MARRAGHELLO, 2005).

2.2.3 UML - *Unified Modeling Language*

No desenvolvimento de *softwares* para computadores de propósito geral, a UML é uma das linguagens mais utilizadas e indicadas para modelar o sistema conforme as necessidades do cliente. Entretanto, no projeto de SE, essa ferramenta acabou não sendo utilizada, uma vez que existe tanto o desenvolvimento de *software* quanto de *hardware*. Mesmo assim; há pesquisas que visam introduzir o uso dessa linguagem no projeto de SE, para gerar uma padronização de fluxo de projeto, porém, ainda não foram obtidos resultados que pudessem ser implementados em ambientes corporativos (UML, 2010).

A linguagem UML é composta de vários diagramas, cada um sendo designado a modelar uma característica em particular do sistema. Para modelar as funcionalidades do sistema são utilizados os diagramas de Casos de Uso, Colaboração e Interação. O comportamento do sistema é definido pelos diagramas de transição de estado e diagrama de atividades, as restrições são denotadas pelos requisitos temporais e pelos requisitos de desempenho (Q*uality of Service* – QoS); a estrutura é modelada pelo diagrama de classes.

2.2.4 VHDL

A linguagem VHSI (*Very High Speed Integration Circuit*) teve sua origem no final dos anos 70, nos Estados Unidos. Foi criada visando a descrição técnica e projeto de uma nova linha de circuitos integrados. Esse programa tornou-se obsoleto com o passar do tempo. Em 1981, a linguagem VHSI foi aprimorada e acoplada a uma linguagem de descrição mais genérica e flexível, dando origem a VHDL (VHSI *Hardware Description Language)*, linguagem bem aceita pelos desenvolvedores de *hardware* da época. Posteriormente, no ano de 1993, foi lançada outra versão VHDL que provia mais recursos e flexibilidade, tornando-se

a mais utilizada até os dias atuais (HUSMANN, 2001).

Na VHDL cada unidade a ser modelada é denominada *design entity* (entidade de desenvovimento), ou VHDL *entity* (entidade VHDL), (MARWEDEL, 2006). Essas entidades são divididas em duas partes, sendo a declaração da entidade e uma, ou várias, arquiteturas como demonstra a Figura 2.3.

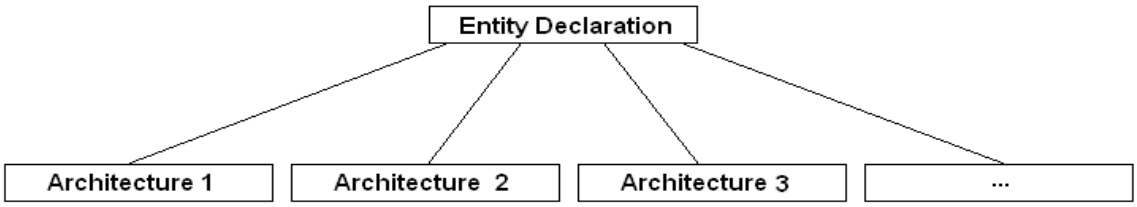

Figura 2.3: Declaração da Entidade e sua(s) Arquiteturas, (MARWEDEL, 2006).

A linguagem VHDL é muito semelhante as linguagens de programação, onde o projetista necessita instanciar os componentes, seguindo a sintaxe requirida pela mesma, e atribuir valores para poder observar o comportamento do sistema. Sendo assim, esta linguagem permite criar uma especificação funcional executável, facilitando o desenvolvimento do *hardware;* contudo, sua aprendizagem é mais difícil do que UML, Redes de Petri e *State Charts*.

2.2.5 SystemC

A linguagem SystemC (2010) segue o mesmo padrão da VHDL, pois é uma biblioteca de componentes que, uma vez adicionada a um código C/C++, pode simular arquiteturas, possibilitando modelar tanto o *hardware* quanto o *software*, pois permite ao usuário utilizar elementos de *hardware*, como processadores, portas e barramentos, como se fossem objetos da linguagem.

As arquiteturas descritas em SystemC são simuladas obedecendo a um relógio (*clock*) e podem ser implementadas em vários níveis de abstração, desde uma aplicação pura em C++ até um nível RTL (*Register Transfer Level*) (2010). O SystemC possui algumas limitações, pois não demonstrava resultados relevantes como área ocupada, frequência máxima do relógio e energia consumida. Atualmente, foram desenvolvidas ferramentas que fazem as transições da linguagem diretamente para silício (REGO, 2006), podendo-se assim automatizar a produção.

2.3 Exploração do espaço de projeto

A exploração do espaço de projeto é uma das etapas mais importante do fluxo de projeto de um SE, na qual são definidos requisitos como consumo de potência e desempenho que a arquitetura precisa atingir. Deve ser feita cuidadosamente, pois detalhes que venham a ser ignorados podem gerar um produto que não atinja as expectativas, principalmente com relação a consumo de potência e desempenho.

Como descrito na Seção 1.1, plataformas virtuais são utilizadas para simular as arquiteturas em um estado inicial de projeto e, consequentemente, obter uma estimativa de desempenho antes que o *hardware* seja construído, facilitando a exploração do espaço de projeto.

No entanto, mesmo com essas poderosas ferramentas de estimativa, o trabalho de simular e avaliar uma grande porção de arquiteturas afim de encontrar a que melhor satisfaça os requisitos impostos é muito custoso, tornando-se a tarefa que mais consome tempo no fluxo de projeto de um SE.

Objetivando minimizar o custo de avaliação de arquiteturas, métodos heurísticos são frequentemente utilizados. Métodos como os Algoritmos Genéticos (AG) são capazes de otimizar o número de arquiteturas simuladas pela plataforma. Como exemplo podemos citar o Platune (GIRVARGIS e VARID, 2002) que é uma plataforma de exploração de espaço de projeto que permite a variação de 26 parâmetros e utiliza AG para otimizar o número de arquiteturas simuladas e ainda sim obtêm resultados que satisfazem os requisitos. Ainda como exemplos da utilização AG na otimização de SE pode-se citar o algoritmo NSGAII (SILVA-FILHO *et al,* 2008) que visa otimizar a memória *cache* do protótipo através de AG.

Redes Neurais Artificiais (RNA) também podem estar sendo treinadas para que sejam capazes de substituir o simulador. Segundo Oyamada (2007), as RNA são utilizadas para estimativas de desempenho de SE uma vez que se pode generalizar seu comportamento mesmo quando o processo a ser modelado é altamente não-linear.

Como resultado da exploração do espaço de projeto, tem-se uma Macro-Arquitetura em um estado onde já estão definidos os componentes que serão utilizados e os processos já estão mapeados. A partir dessa Macro-Arquitetura já é possível desenvolver o *software.*

2.4 Geração do Software aplicativo, Síntese da Comunicação e Síntese da Micro-Arquitetura

A geração do *software* aplicativo e a síntese da micro-arquitetura são tarefas relativamente mais simples de serem executadas em uma metodologia de projeto. A geração do *software* consiste em desenvolver a aplicação que será utilizada no produto. Neste desenvolvimento alguns cuidados devem ser adotados na implementação para gerar o código mais otimizado possível, tendo em vista os requisitos restritos apresentados pelo *hardware*. A síntese da micro-arquitetura, como o próprio nome já diz, objetiva criar a descrição do *hardware;* a partir dela desenvolve-se o protótipo.

O resultado da geração do *software* aplicativo é o sistema e a síntese da micro-arquitetura gera o *hardware*. Para fazer a comunicação entre o *software* e o *hardware* obtidos faz-se necessário o uso de uma rede de comunicação. A síntese da comunicação é quem define como será feita a interligação entre os componentes da arquitetura.

Uma alternativa para solucionar o problema de comunicação entre os blocos de um SE é a utilização de redes NoC (*Network-on-a-Chip*) (CORRE, 2007). Essas tem se tornado amplamente utilizadas, pois mesmo resultando em um aumento do custo de produção e da latência do SE, possibilitam o reuso de componentes e apresentam escalabilidade e paralelismo. Porém, o aumento da latência provocado pelas redes NoC pode ser controlado, em alguns casos, através de ajustes na configuração, como topologia, arbritagem, mecanismos de controle de fluxo, políticas de roteamento e tamanho dos *buffers* (CORRE, 2007).

Os SE, de forma geral, apresentam cada vez mais requisitos rígidos com relação a QoS(*Quality of Service*) e tempo de projeto, aumentando assim a complexidade do projeto como um todo, fato que faz com que a exploração do espaço de projeto seja necessária para manter o produto final com um nível de qualidade aceitável e tempo de projeto viável.

2.5 Projeto Baseado em Plataformas

Os projetos baseados em plataforma visam otimizar o tempo de desenvolvimento e o custo, utilizando plataformas já conhecidas. Esses modelos de arquitetura possuem processadores, memórias, blocos dedicados e estruturas de comunicação. Estes componentes são, em

algumas vezes, reconfigurados para que se possa extrair o maior grau de desempenho possível da arquitetura (OYAMADA, 2007).

Quando se decide por um projeto baseado em plataformas tem-se um ganho de tempo na etapa de exploração do espaço de projeto, visto que a mesma já está pronta. Porém, o projetista fica preso às limitações oferecidas pela plataforma escolhida.

2.6 Projeto Baseado em Componentes

Nos projetos baseados em componentes toda a especificação e desenvolvimento do *hardware,* utilizado no SE a ser desenvolvido, depende do projetista, que produz o *hardware* integrando componentes comprados no mercado. Os componentes utilizados na arquitetura devem obedecer ao mesmo protocolo, para minimizar problemas de comunicação ao integrar os componentes. Um método fundamental que vem sendo utilizado, tendo como objetivo otimizar o processo é o reuso de componentes pré-testados (OYAMADA, 2007).

2.7 Processadores e Arquiteturas Embarcadas

A exploração do espaço de projeto é uma etapa onde várias decisões são tomadas, como definição dos processadores, rede de comunicação, mapeamento de processos entre outros. Porém, a variedade arquitetural que deve ser analisada na tentativa de determinar a que possua o melhor custo/benefício para o protótipo que irá ser desenvolvido é muito ampla.

O projetista deve prever o impacto de cada componente no SE, para poder certificar-se de que não está desenvolvendo um *hardware* sub ou super dimensionado para o *software* embarcado.

Nesta Seção serão apresentados o impacto de alguns componentes como memórias *cache* e a arquitetura do processador escolhida, assim como a influência da arquitetura em si no SE.

2.7.1 Caches

As memórias *cache* foram desenvolvidas pois as memórias convencionais já não acompanhavam mais a velocidade do processador, fazendo com que várias vezes o processador ficasse aguardando dados para continuar seu processamento, diminuindo significativamente seu desempenho. A memória *cache* é uma espécie de memória ultra rápida

utilizada para guardar os dados mais frequentemente utilizados pelo processador, evitando que o processador necessite acessar a memória RAM toda vez que necessitar um dado.

As memórias *caches* são divididas em dois tipo, a *cache* primária ou L1 (*level* 1) e a *cache* secundária ou L2 (*level* 2). A *cache* primária é embutida no próprio processador; sempre que se desenvolve um processador mais potente é necessário ampliar o desempenho da *cache*. A *cache* L1 é muito mais rápida que a memória RAM, porém, é muito mais cara que uma memória convencional, fato que faz com que se utilize apenas uma pequena quantidade de *cache* L1, complementado com a memória *cache* L2, que é um pouco mais lenta e mais barata, possibilitando ser utilizada em maior quantidade.

Pode-se concluir que a utilização de memórias *cache* aumenta a velocidade de resposta da arquitetura; porém são responsáveis pela maior parte do consumo de um processador, tornando necessário avaliar os impactos que o tamanho das memória *cache* trazem para o sistema não só em questão de velocidade mas também com relação a quantidade de ciclos e potência consumida (GARCIA, 2008), o que é demonstrado na Figura 2.4.

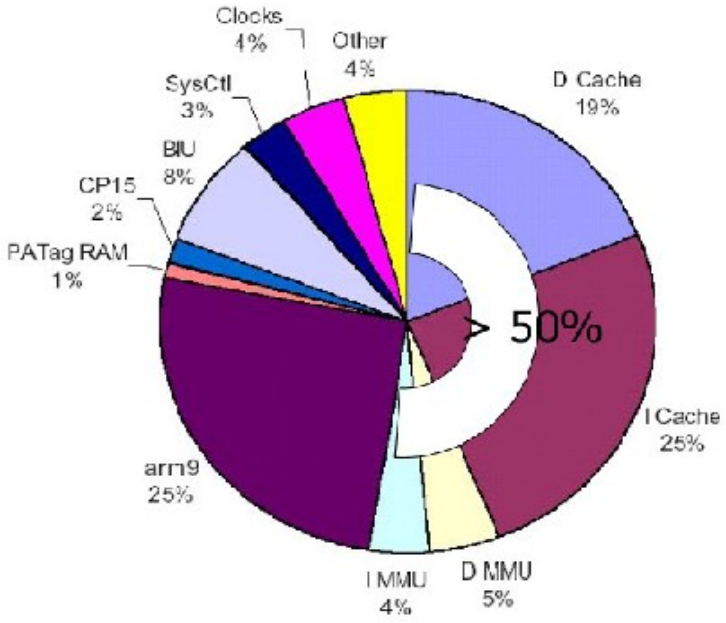

Figura 2.4: Consumo de potência em um sistema que utiliza Arm9 (ZHANG, VAHID e LYSECKY, 2004)

2.7.2 Arquitetura do Processador

Uma parte relevante da exploração do espaço de projeto é a definição dos processadores

que serão utilizados. Os SEs podem ser tanto homogêneos quanto heterogêneos. Chamamos de SEs homogêneos aqueles que possuem processadores do mesmo tipo, enquanto os heterogêneos possuem processadores de tipos variados. Esse conceito surgiu a partir do momento em que se percebeu que mesmo que um processador seja considerado melhor, o outro, considerado inferior, possui características que contribuiriam de uma maneira mais eficiente ao sistema como um todo.

Atualmente, a procura por processadores 32-bit no mercado tem se mostrado bastante diversificada, contendo vários tipos de processadores, entretanto, existe a dominância de utilização do processador ARM, que lidera com cerca de 57% dos processadores vendidos. Esta variedade gera aos projetistas opções com relação a desempenho, potência e custo (OYAMADA, 2007), mostrado na Figura 2.5.

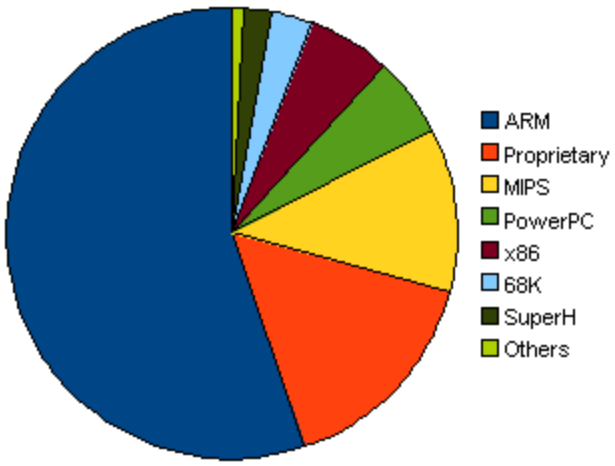

Figura 2.5: Procura por processadores embarcados no mercado (OYAMADA, 2007).

2.7.3 Arquitetura Multiprocessada vs Arquitetura Monoprocessada

Com o aumento do desempenho dos componentes de *hardware,* utilizados tanto em computadores de propósito geral quanto em SE, e o crescente poder de miniaturização, criam-se componentes cada vez menores e mais potentes.

Segundo a lei de Moore (1965), a cada 18 meses o número de transistores de um componente dobraria. Essa lei tem se concretizado, porém estudos deferiram que no ano de 2014, seguindo o ritmo da lei de Moore, os semicondutores atingirão seu limite físico, cerca

de 20 nanômetros. Tendo em vista esta possível problemática, começou-se a planejar e estudar alternativas para manter o crescimento de desempenho computacional, e uma das alternativas encontradas foram as arquiteturas multiprocessadas.

A aplicação da lei de Moore, possibilitou o desenvolvimento dos componentes denominados *Systems-on-a-Chip*(SoCs). A utilização de SoC vem se mostrando uma solução amplamente utilizada pela indústria para facilitar o desenvolvimento de sistemas embarcados, já que em suas arquiteturas podem conter um ou mais processadores, memórias, interfaces para periféricos e blocos dedicados. Os componentes de um SoC são interligados através de uma estrutura de comunicação que pode ir desde um barramento até uma complexa rede NoC (*Network-on-a-Chip*) (CARRO e WAGNER, 2003).

Utilizando-se da tecnologia dos SoC juntamente com a idéia de arquiteturas multiprocessadas, originou-se uma classe particular de SoC, denominada MPSoC que encapsula em um único chip múltiplos processadores, podendo estes serem homogêneos ou heterogêneos. Os MPSoC auxiliam os projetistas a alcançarem os requisitos necessários de um SE, pois atende a necessidade de desempenho, reduz o custo de projeto, elimina elementos desnecessários e reduz o consumo de potência (JUNIOR e GARIBOTTI, 2007).

Atualmente os processadores *multicore* e arquiteturas multiprocessadas são os principais produtos para a computação de propósito geral, pois além de aumentarem o desempenho, possibilitam um menor consumo de potência e paralelismo real. A arquitetura Normadik (2010) é um exemplo de arquitetura multiprocessada utilizada em telefones móveis. É composta por um processador ARM e um aceledador de áudio e vídeo (OYAMADA, 2007). Há também, como exemplo de arquitetura multirpocessada OMAP (2010), utilizado em aparelhos multimídia móveis em geral, composta por dois processadores, um ARM de propósito geral e um DSP para processamento multimidia.

Entretanto, a área de arquiteturas multiprocessadas está sendo muito estudada no meio acadêmico pelo fato de oferecer benefícios tanto para o projeto de SE quanto para computadores de propósito geral. Ao revermos a análise feita na Seção 1.1, embasada na Equação 2, é possível concluir que substituindo o processador por dois processadores com metade da frequência, o consumo cairia pela metade mantendo o mesmo desempenho. Este método de diminuição do consumo de potência, apesar de ser bastante usado atualmente, foi descrito por Gelsinger e colaboradores (1989). Gelsinger baseou sua previsão na possibilidade de existir um grande número de aplicações sendo executadas ao mesmo tempo em uma

arquitetura, melhor organização dos componentes computacionais e o limite da miniaturização (GELSINGER *et al.*, 1989). Esses processadores que possuem mais de um núcleo são denominados *multicore* e são bastante utilizados na computação de propósito geral.

2.8 Plataformas para simulação de arquiteturas

Ao se perceber que aumentando o foco na fase de exploração de projeto seria possível otimizar as arquiteturas, minimizar os erros no protótipo e, consequentemente, diminuir as chances de que o protótipo não atenda aos requisitos, começou-se a pesquisar e desenvolver plataformas que poderiam facilitar este processo. Essas plataformas tem o poder de simular as arquiteturas e mostrar estimativas de desempenho, potência consumida entre outras características que o *hardware* irá possuir.

Nesta Seção algumas plataformas serão discutidas, como o seu funcionamento, suas qualidades e suas peculiaridades.

2.8.1 ConvergenSC

O ConvergenSC (2010), lançado pela CoWare (2010), é uma ferramenta para projeto de SE baseada em plataformas em domínio genérico da aplicação. Sua biblioteca possui os modelos dos processadores ARP e MIPS e também modelos de barramento como o AMBA. Caso o projetista necessite de um componente que não consta na biblioteca é possível criá-lo, utilizando o LISATek (HOFFMAN, 2001) e adicioná-lo à biblioteca na forma de um componente.

A base do ConvergenSC é o SystemC que possui recursos de particionamento de *hardware*/*software*, modelagem de plataformas, simulações, depurações e análises. Sua estrutura permite a fácil realização de tarefas como otimização de arquiteturas SoC (LEI, YANHUI e SHAOJUN, 2010).

O ConvergenSC possui interface gráfica para modelagem de plataformas, facilitando a tarefa de montar, simular e avaliar arquiteturas. Entretanto, a exploração é realizada manualmente, resultando em um elevado tempo de exploração. A Figura 2.6 demonstra uma arquitetura modelada na plataforma ConvergenSC.

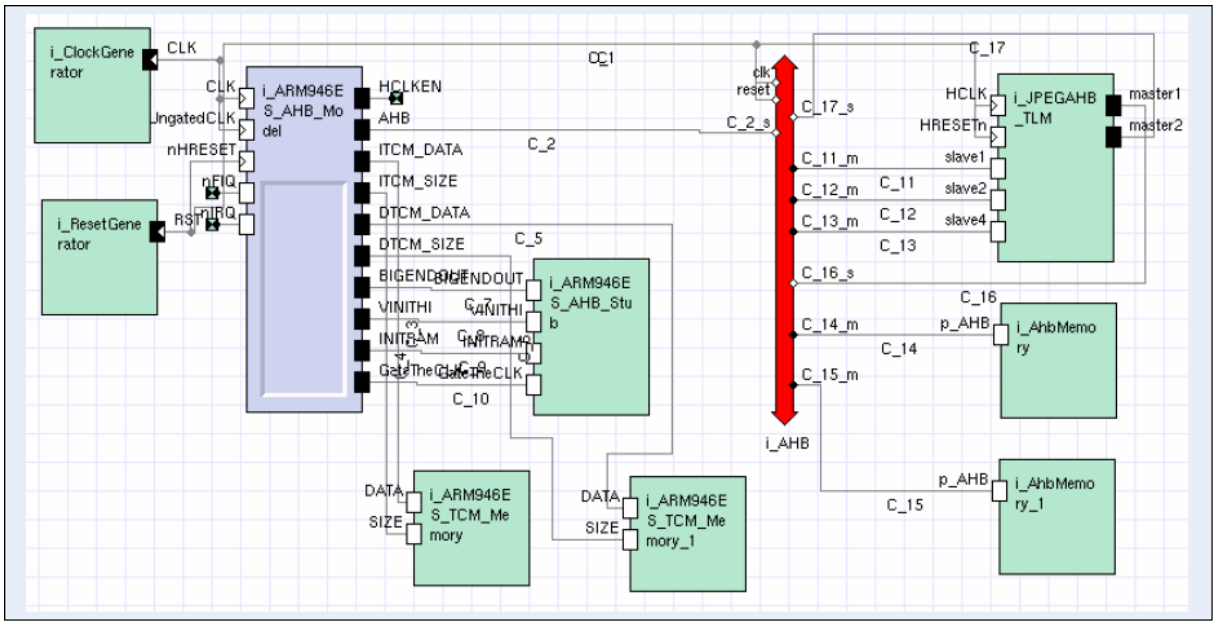

Figura 2.6: Arquitetura modelada na plataforma ConvegenSC

2.8.2 *Open Virtual Platform*

O OVP (*Open Virtual Platform)* foi lançado em março de 2008. É uma ferramenta de código aberto, flexível e gratuito. O OVP possui uma API que possibilita criar o modelo em linguagem C; também possui uma biblioteca, de fonte aberta, e modelos de processadores e periféricos.

Esta plataforma permite a criação do modelo de simulação de arquiteturas, podendo estas serem multiprocessadas ou monoprocessadas. Outra vantagem do OVP é a possibilidade de compilar esse modelo desenvolvido para um modelo executável, que pode, posteriormente, ser conectado ao depurador, proporcionando um ambiente de desenvolvimento de *software* embarcado.

O OVP é apoiado por um conjunto de empresas fabricantes de processadores embarcados que disponibiliza modelos de simulação para o ambiente. Desta forma é possível criar plataformas virtuais com modelos de processadores validados pelo fabricante. No entanto o ambiente não provê interface gráfica e muito menos métodos heurísticos para exploração do espaço de projeto.

2.8.3 VIPRO-MP

O VIPRO-MP (*Virtual Prototype for Multiprocessor Architectures*) é um ambiente desenvolvido para a simulação de plataformas multiprocessadas, que utiliza o SimpleScalar como simulador do processador e o *framework* para cálculo de potência Wattch (*Sim-Wattch*) e utiliza o SystemC para modelar os demais componentes de *hardware* (GARCIA, 2008).

O modelo arquitetural implementado no VIPRO-MP permite a execução de aplicações descritas em linguagem C e compiladas para conjunto de instrução PISA, semelhante à arquitetura MIPS (GARCIA, 2008). Segundo Garcia (2008), esta arquitetura foi escolhida por pertencer a uma empresa que vende processadores reconfiguráveis de acordo com as necessidades do cliente, e que está crescendo no mercado de SEs, alcançando produtos como DVD Recoders, BlueRay Decoders entre outros.

Os processadores possuem 2Gb de memória privada limitada e sua comunicação acontece apenas pela memória compartilhada, que por sua vez possui os endereços base e final como sendo, respectivamente, 0x80000000 e 0x8FFFFFFF (GARCIA, 2008). A princípio, os dados da memória compartilhada não eram armazenados na memória *cache*, fato que evitou a implementação de protocolos de coerência. Entretanto, na última versão, que ainda está em desenvolvimento, os dados da memória passaram a ser armazenados em memórias *cache* e os protocolos serão devidamente implementados. Os parâmetros que configuram cada processador mantém a sintaxe da SimpleScalar, fato que facilita a utilização do VIPRO-MP para usuários da SimpleScalar, uma vez que é uma linguagem muito utilizada na produção de SE. A Figura 2.7 demonstra como o VIPRO-MP está organizado, onde os componentes externos são possíveis módulos para estender a plataforma virtual.

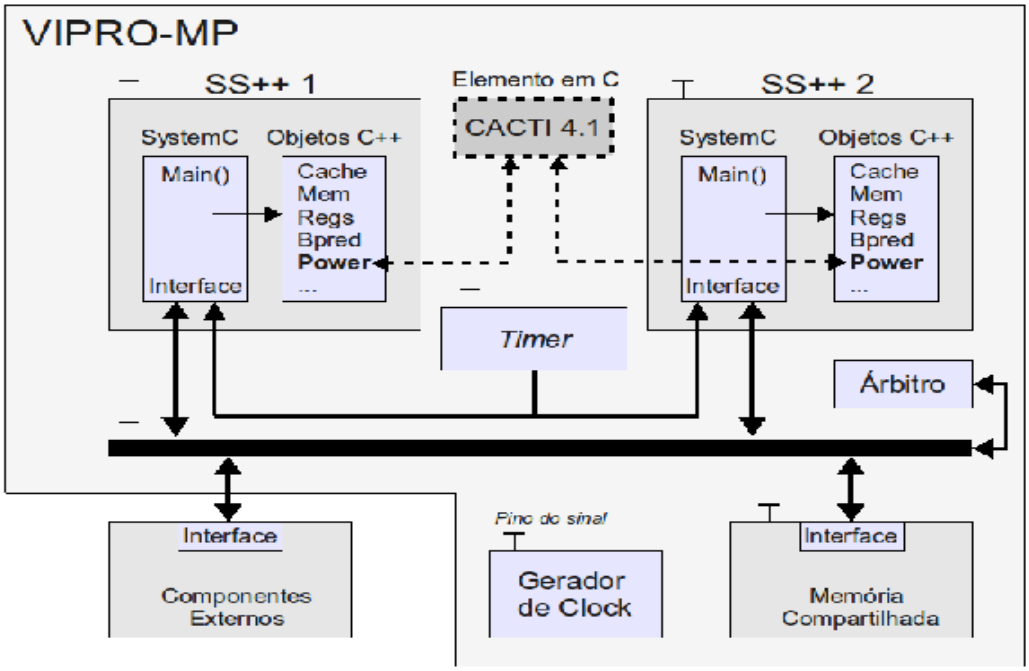

Figura 2.7: Estrutura da plataforma VIPRO-MP (GARCIA, 2008)

A grande problemática dessa plataforma é que todo o processo de montagem da arquitetura é feito a partir de códigos C++, dificultando a exploração do espaço de projeto. Um dos objetivos deste trabalho visa sanar esta problemática através de uma *interface* visual para com o usuário.

Capítulo 3

VIPEX-VMP

Como já dito este trabalho tem como objetivo o desenvolvimento de uma plataforma visual para exploração de espaço de projeto visando otimizar arquiteturas em termos de desempenho (ciclos para execução) e consumo de potência e energia denominada VIPEX-VMP (*Virtual Platform Exploration – Vipro-MP*), que utiliza métodos heurísticos para evitar simulações exaustivas. A interface com o usuário é demonstrada na Figura 3.1.

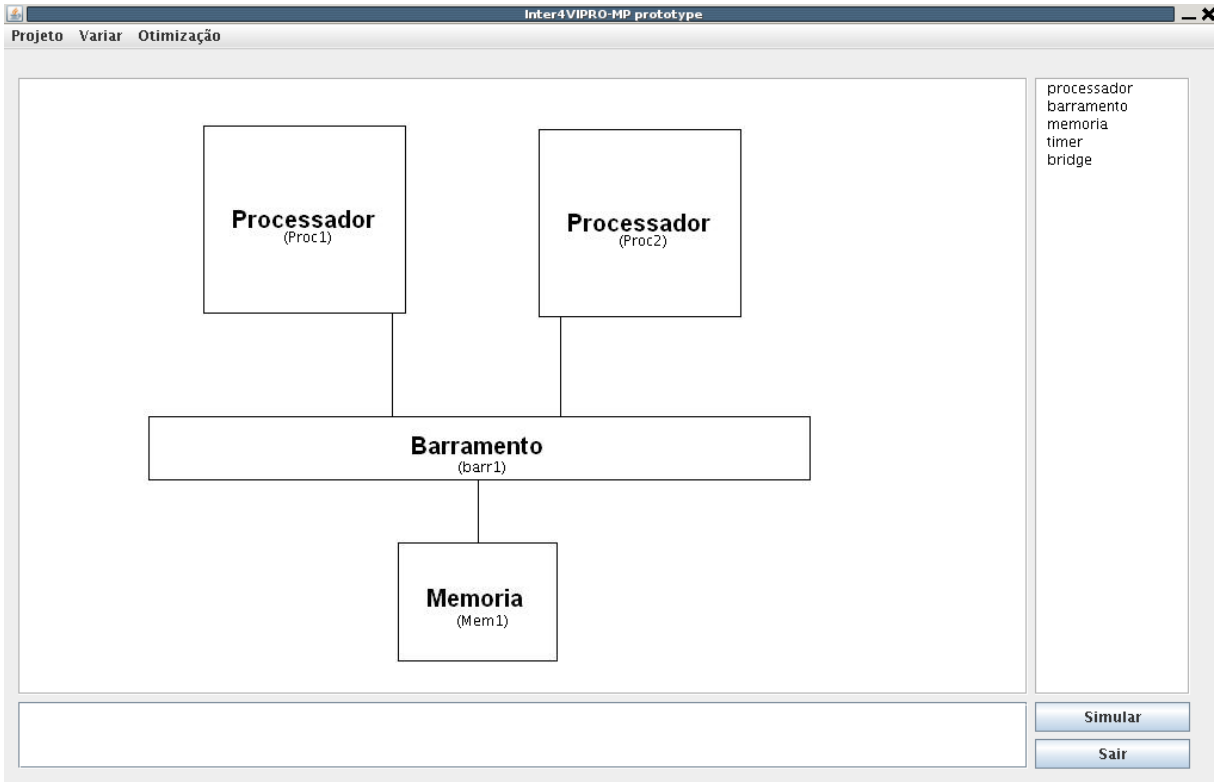

Figura 3.1: Interface desenvolvida

Para o desenvolvimento do ambiente visual, foi utilizada a IDE do NetBeans (2010). O protótipo desenvolvido visa tornar a tarefa de construção e simulação de arquiteturas mais interativa, tal que o ambiente foi implementado de modo que, para montar uma arquitetura,

basta que o projetista selecione os componentes e os coloque na tela em suas respectivas posições, podendo também conecta-los com facilidade. O VIPEX-VMP permite também que o projetista altere as propriedades dos componentes utilizados, como demonstra a Figura 3.2. A definição de quais processos irão executar em cada processador é feita na tela de propriedades do próprio processador, na guia "Arquivo".

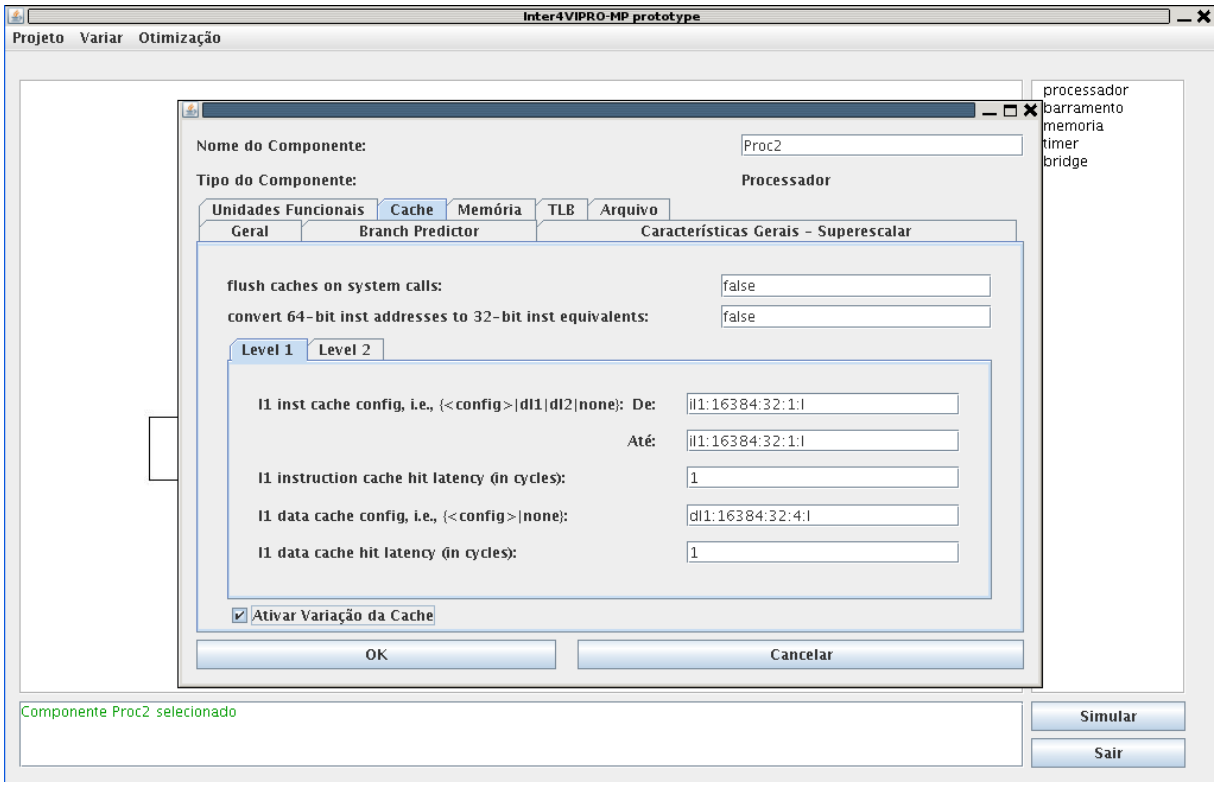

Figura 3.2: Propriedades do processador

Como a simulação das arquiteturas é feita pela plataforma VIPRO-MP, o VIPEX-VMP gera automaticamente, a partir da arquitetura modelada pelo projetista, os seguintes arquivos:

a) arquivo em linguagem C++ contendo a declaração, instanciação e conexão dos componentes da arquitetura;

b) arquivos de configuração dos processadores contendo os parâmetros configurados pelo usuário;

c) o arquivo *makeFile* para compilação e geração do executável;

d) e o *script* para execução da simulação.

O ambiente possibilita dois tipos de exploração da arquitetura: exaustiva e utilizando os métodos heurísticos AG e RNA MLP.

O fluxograma apresentado na Figura 3.3 representa o cenário inicial em que o explorador do espaço de projeto se encontra no começo da simulação de uma arquitetura. O VIPEX-VMP tem como entrada a configuração base da arquitetura, definida pelo projetista, e a aplicação que a mesma suportará. Ao receber os dados de entrada, o algoritmo genético cria a primeira população a partir da arquitetura de entrada e em seguida realiza a simulação de todos os indivíduos que foram criados.

Ao finalizar a simulação de toda a população, ocorre a transição 3, na qual os dados obtidos pela simulação são armazenados no banco de dados, alimentam o algoritmo genético, para que possa analisar os resultados e criar a próxima geração, e são utilizados para o treinamento da RNA. Após o treinamento, são apresentados casos conhecidos à RNA para verificar se a mesma já está treinada.

Para definir se a RNA está realmente treinada, ela deve respeitar 2 condições, sendo a primeira de que o número de gerações atual do AG seja maior que o número estipulado pelo projetista, restrição imposta para que não houvesse chance de utilizar-se uma RNA não treinada. Já a segunda condição diz respeito a saída proporcinada pela RNA, que deve ser menor que o erro definido pelo usuário.

Caso, após os testes, ambas as condições forem verdadeiras, o VIPEX-VMP passa a operar no cenário 2, representado pela Figura 3.3. Neste cenário o simulador foi eliminado e para estimar a avaliação da arquitetura está sendo utilizada a RNA, fazendo com que o objetivo deste trabalho esteja concluído.

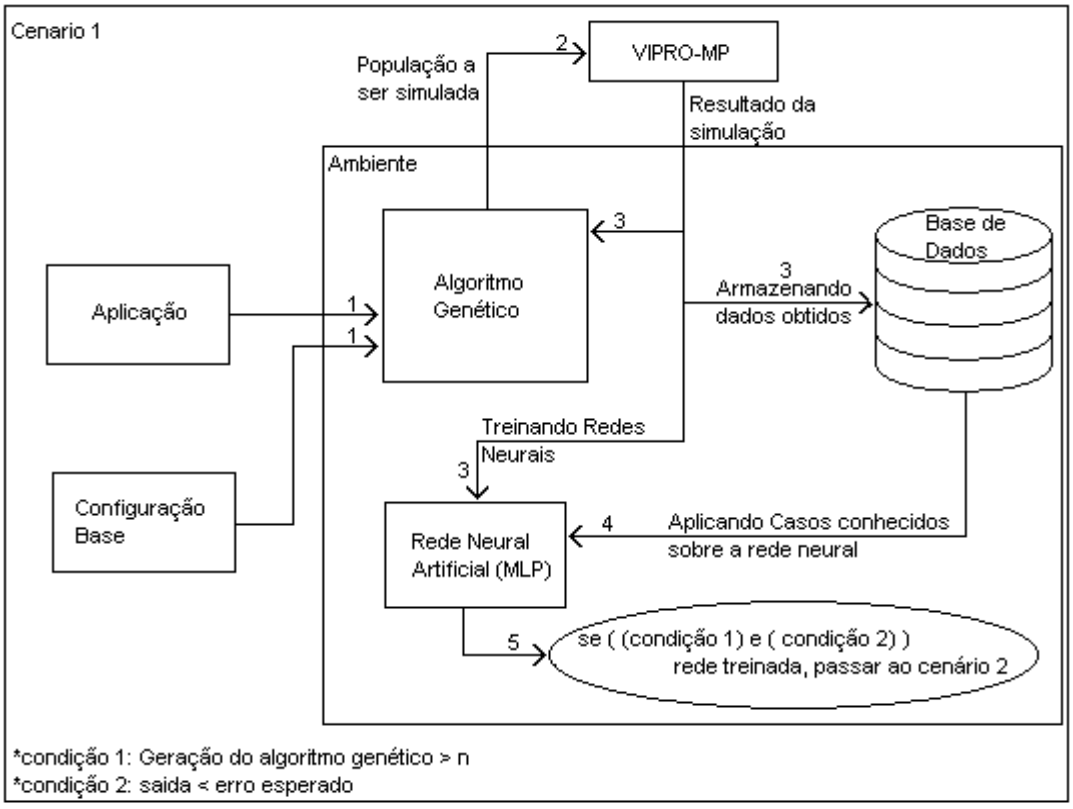

Figura 3.3: Fluxograma do cenário 1, treinamento das RNA

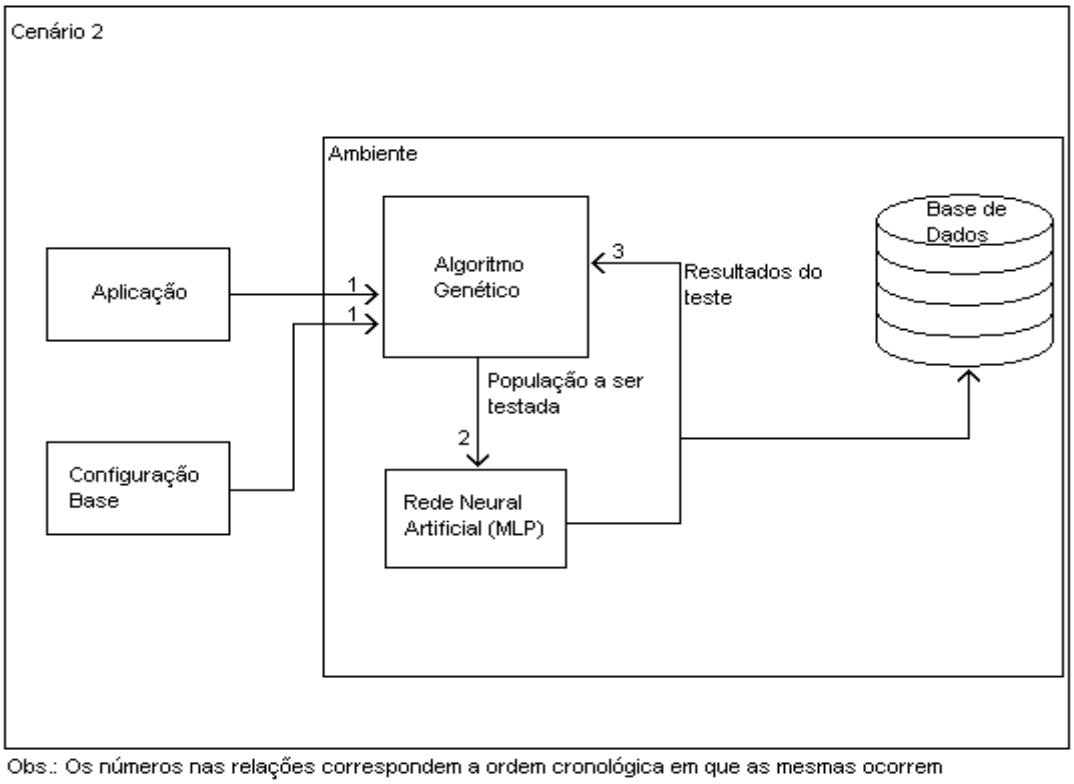

Figura 3.4: Fluxograma do cenário 2, utilização das RNA

3.1 Exploração do espaço de projeto – Simulação Exaustiva

Na simulação exaustiva, o VIPEX-VMP varia os parâmetros do processador de forma sequencial para determinar a melhor configuração. Atualmente, os parâmetros que podem ser avaliados são: configuração da memória, número de unidades funcionais e configuração da memória *cache* il e *cache* dl. O intervalo de variação das memórias *cache* il e *cache* dl é definido na tela de propriedades de cada processador, como demonstra a Figura 3.2; os demais componentes possuem intervalos fixos.

Após definida as configurações da arquitetura, para realizar a simulação exaustiva, basta clicar no menu “variar”, opção “variar”. O VIPEX-VMP simulará cada uma das arquiteturas e ao final gerará um arquivo na pasta de saida com o nome “dump”, contendo em cada linha o desempenho e a potência da arquitetura simulada.

Para avaliação do ambiente, um estudo de caso de um codificador JPEG paralelo foi utilizado (GARCIA, 2008). Uma arquitetura com dois processadores e uma memória, conectados entre si através de um barramento, foi utilizada como configuração base, conforme demonstrada na Figura 3.1. No estudo de caso foram variados 4 parâmetros, conforme apresentado na Tabela 3.1:

Tabela 3.1: Parâmetros variados pela simulação exaustiva.

Componente	Mínimo	Máximo	Incremento
Latência da memória compartilhada	4	16	2 ^ n
Cache da dados nível 1 (DL1)	64 Kb	65536 Kb	2 ^ n
Cache de instruções nível 1 (IL1)	256 Kb	262144 Kb	2 ^ n
Número de unidades funcionais	1 Unidade	8 Unidades	1

Considerando que a memória compartilhada, a memória *cache* il e *cache* dl são variadas em intervalos de 2^n e o número de unidade funcionais varia em incrementos de 1, o número total de combinações é de 2904 arquiteturas. Cada simulação executada pelo VIPRO-MP leva cerca de 90 segundos para codificar uma imagem de 16 x 16 pixels, resultando em um tempo total de simulação de cerca de 72 horas. Analisando os dados obtidos e fazendo uma relação

entre desempenho e potência consumida foi possível elaborar os gráficos apresentados nas Figuras 3.5 e 3.6. Ambos os gráficos possuem os mesmos dados, porém, no primeiro foram plotadas as arquiteturas que resultaram na menor portência para cada desempenho em particular, enquanto no segundo, ao invés de gerar o gráfico a partir de cada valor de desempenho, utilizou-se faixas de desempenho, com intervalos de 5000 ciclos, gerando assim um gráfico mais limpo.

O gráfico representado pela Figura 3.5 mostra a relação entre o número de ciclos e a potência consumida pela arquitetura, portanto quanto mais a direita o arquitetura estiver, maior o número ciclos utilizados, aos passo que quanto mair acima estiver, maior a potência consumida. Consideramos a arquitetura mais balanceada como sendo a que estiver mais próxima da origem do gráfico, pois apresentou um baixo consumo de potência em relação as outras e levou menos ciclos que as outras para finalizar a tarefa.

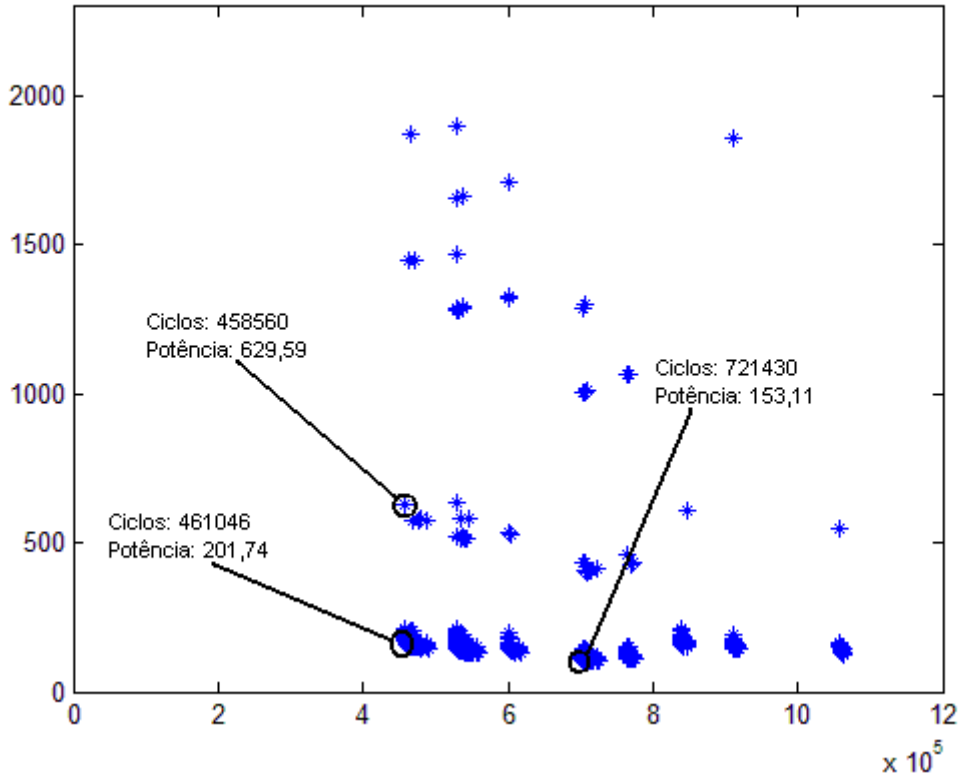

Figura 3.5: Arquiteturas avaliadas nos teste exaustivos

Em cada uma das simulações, ambos os processadores receberam a mesma configuração,

porém, para uma melhor otimização da arquitetura, os processadores deveriam ser testados com todas as combinações possíveis, o que, para nosso caso de teste, resultaria em 8433216 arquiteturas diferentes, elevando o tempo de otimização para cerca de 210830 horas. Estas estimativas demonstram a inviabilidade da realização de testes exaustivos.

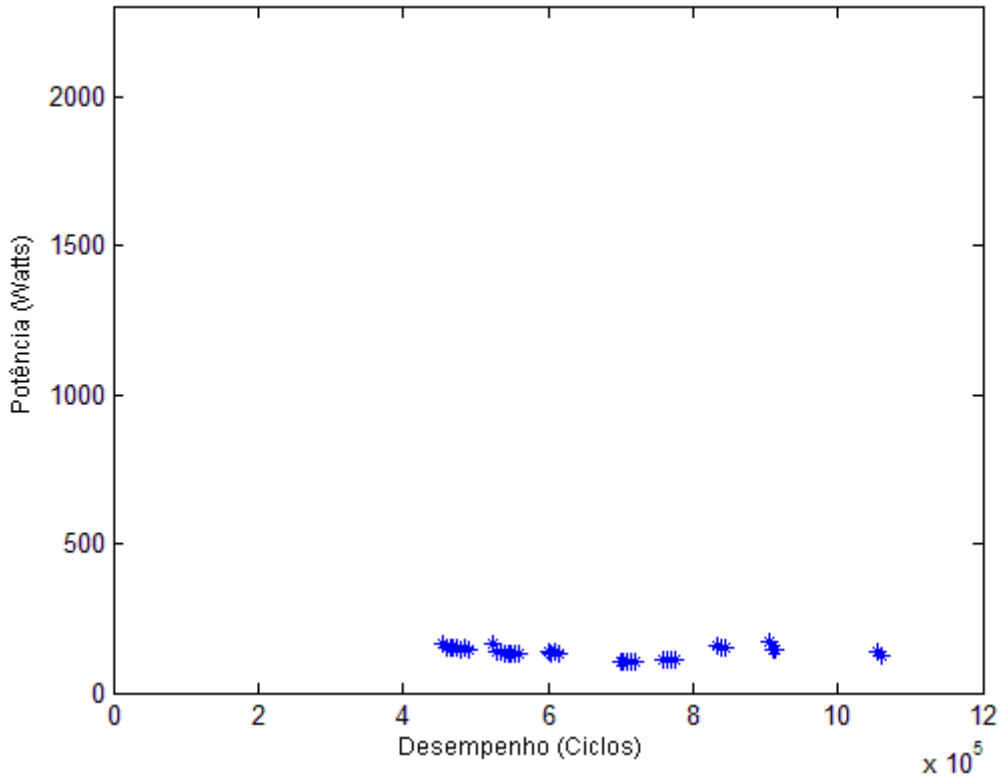

Figura 3.6: Arquiteturas avaliadas nos teste exaustivos demonstrada em intervalos de 5000 ciclos

3.2 Exploração do espaço de projeto - Algoritmos Genéticos

No VIPEX-VMP, os algoritmos genéticos (AG) foram utilizados para gerar as arquiteturas que serão simuladas. O AG foi modelado de maneira que cada um dos indivíduos seja uma arquitetura, sendo que seus cromossomos carregam parâmetros da configuração dos processadores. A primeira população é gerada a partir de mutações da arquitetura base fornecida pelo projetista e cada uma das arquiteturas é simulada pelo VIPRO-MP, que por sua vez gera um arquivo contendo o número de ciclos e a potência consumida pela arquitetura

simulada.

O método de seleção utilizado foi o método do torneio, que objetiva selecionar *n* indivíduos aleatoriamente e definir o melhor, que entra para um novo torneio, e assim sucessivamente até que um indivíduo seja escolhido como o melhor, em nossos casos o *n* é igual a três. Este método foi escolhido por não favorecer os melhores indivíduos da população. No entanto, a única vantagem que os melhores indivíduos tem é que, sendo selecionados, vencerão o torneio (LINDEN, 2008).

O método de mutação implementado foi o método da mutação aleatória, no qual é definida as chances que um cromossomo tem de sofrer a mutação. Caso a mutação ocorra, o cromossomo é substituído por um valor randômico entre o mínimo e o máximo permitido. A mutação aleatória, além de possuir uma fácil implementação, é muito eficiente para retirar o AG de um máximo ou mínimo local.

O método de cruzamento escolhido, foi o método do corte mediano. A idéia base do corte mediano prediz que cada "*casal*", dois indivíduos resultantes do método de seleção, irá gerar dois "*filhos*", um constituido da primeira metade de cromossomos do "*pai"* e da segunda metade de cromossomos da "*mãe*", e o outro "*filho*" é oposto ao "*irmão*", como demonstra a Figura 3.7. Este método permite que o número de cromossomos dos indivíduos permaneça o mesmo em todas as gerações, que neste caso é necessário, pois o cromossomo determinará qual a configuração de um determinado componente da arquitetura.

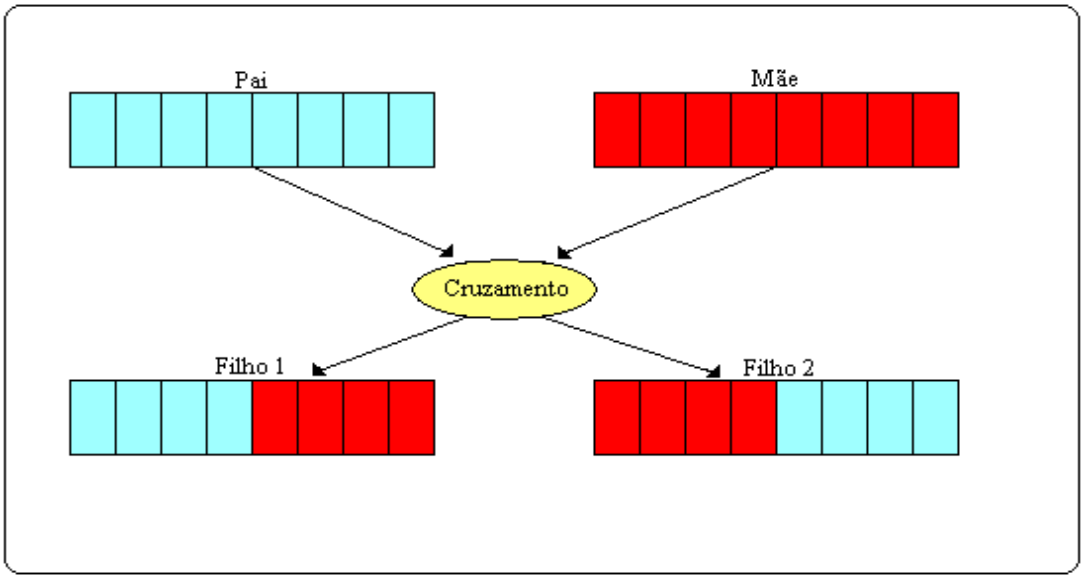

Figura 3.7: Método do corte mediano

Após todas as arquiteturas da geração serem simuladas, são aplicados os métodos de seleção, de cruzamento e de mutação sobre a população atual, criando assim a população da próxima geração. O critério de convergência adotado foi a repetição da melhor arquitetura por 15 gerações.

A função *fitness* utilizada para a avaliação dos indivíduos foi a soma da média da potência dos processadores com o desempenho da arquitetura. Como a diferença entre o número de ciclos e a potência consumida é muito grande, com o intuito de melhorar a avaliação, ambos os parâmetros foram normalizados para que tivessem valores mais próximos, para isso o desempenho foi dividido por 10^8 e a potência por 10^3.

A configuração base da arquitetura utilizada nos testes foi a mesma utilizada nos testes exaustivos. O AG foi implementado para variar 46 parâmetros, sendo 23 em cada processador. Em teste iniciais, foi identificado um problema ao perceber que cerca de 47% das arquiteturas geradas pelo AG já tinha sido simulada em alguma geração anterior. A estas arquiteturas era então atribuído o resultado já obtido, evitando que fossem novamente simuladas.

Isso reduziu o número de arquiteturas simuladas e avaliadas, facilitando a repetição da melhor arquitetura, levando à convergência precoce para um mínimo local. Para contornar essa incoveniência, a taxa de mutação foi aumentada de 30% para 75%, reduzindo o percentual de arquiteturas repetidas para 16%.

A aplicação utilizada foi a codificação JPEG paralela; a mesma utilizada nos testes exaustivos. O estudo de caso foi configurado para um número máximo de 300 gerações, com 10 indivíduos cada. Com essas configurações, a convergência foi obtida entre 165 e 270 gerações, resultando em uma otimização entre 25% e 55% do número de simulações realizadas, quando comparados aos testes exaustivos. O tempo de execução destes testes variou de, aproximadamente, 34 a 56 horas.

Analisando os resultados obtidos, pode-se gerar os gráficos representados pelas Figuras 3.8 e 3.9, no qual a arquitetura que está mais próxima da origem é a que possui a potência e o desempenho mais balanceados.

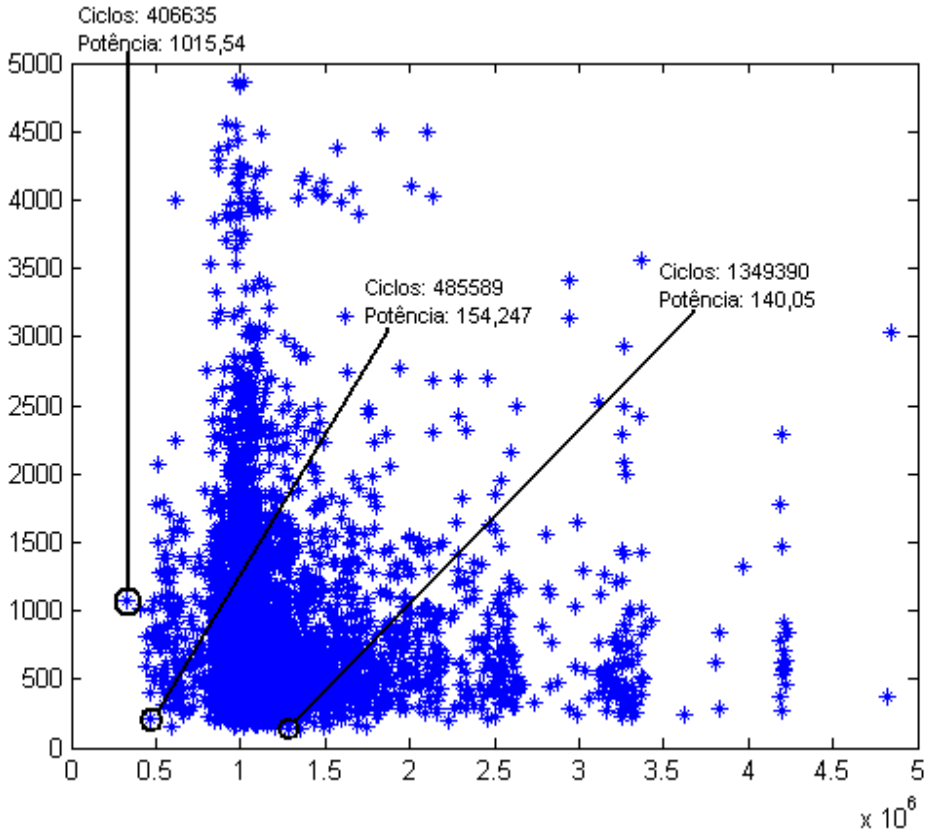

Figura 3.8: Resultado das arquiteturas obtidas pelos Algoritmos Genéticos

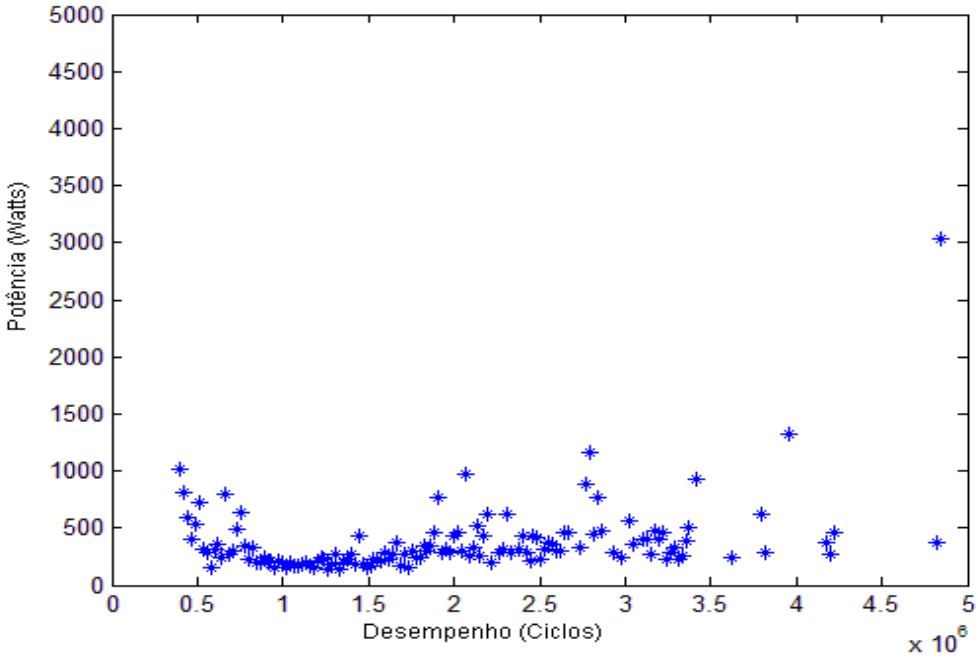

Figura 3.9: Resultado das arquiteturas obtidas pelos Algoritmos Genéticos amostrada em intervalos de 5000 ciclos

Para melhor avaliação dos AG, foram gerados gráficos das gerações 1, 75, 150, 225 e 300, representados, respectivamente, pelas Figuras 3.10 a 3.14. Devido a alta taxa de mutação utilizada, é comum a ocorrência de arquitetura atípicas, que obtém resultados muito diferentes dos outros indivíduos da geração. Para avaliar o comportamento dos AG perante a otimização de arquiteturas, o critério de convergência foi removido, para que o AG pudesse trabalhar até seu limite, chegando a geração de número 300. O tempo de execução desse teste foi de cerca de 62 horas, com 517 arquiteturas repetidas, representando, portanto, 17,23% do total de arquiteturas.

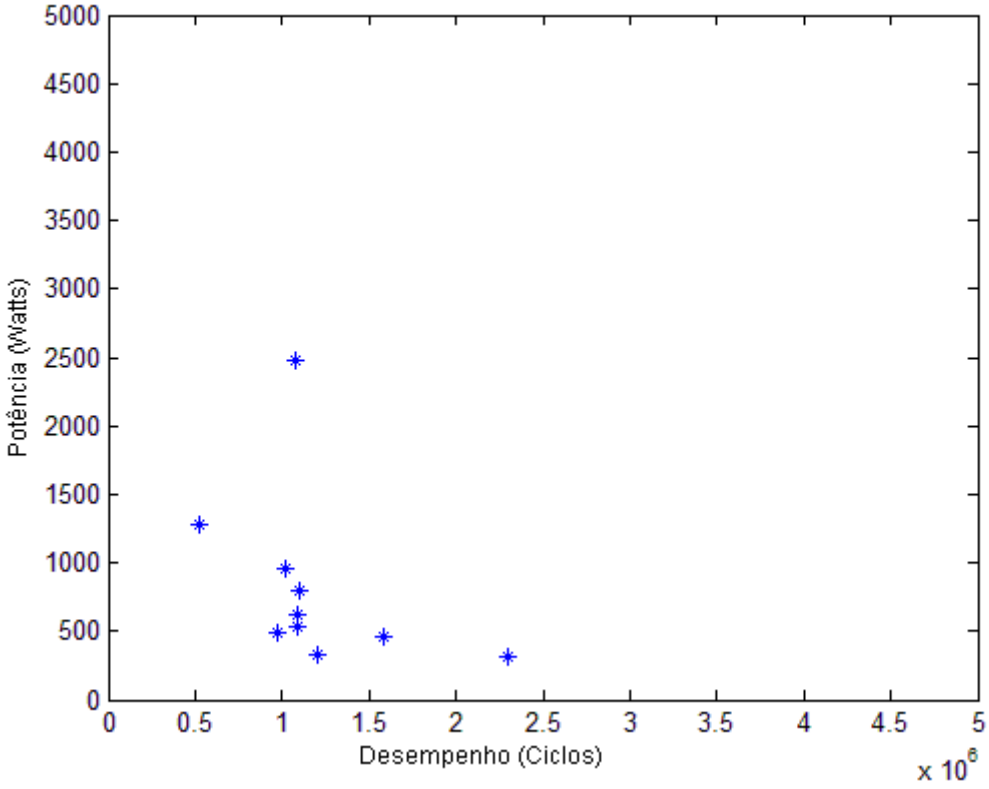

Figura 3.10: Geração 1, relação entre desempenho e potência

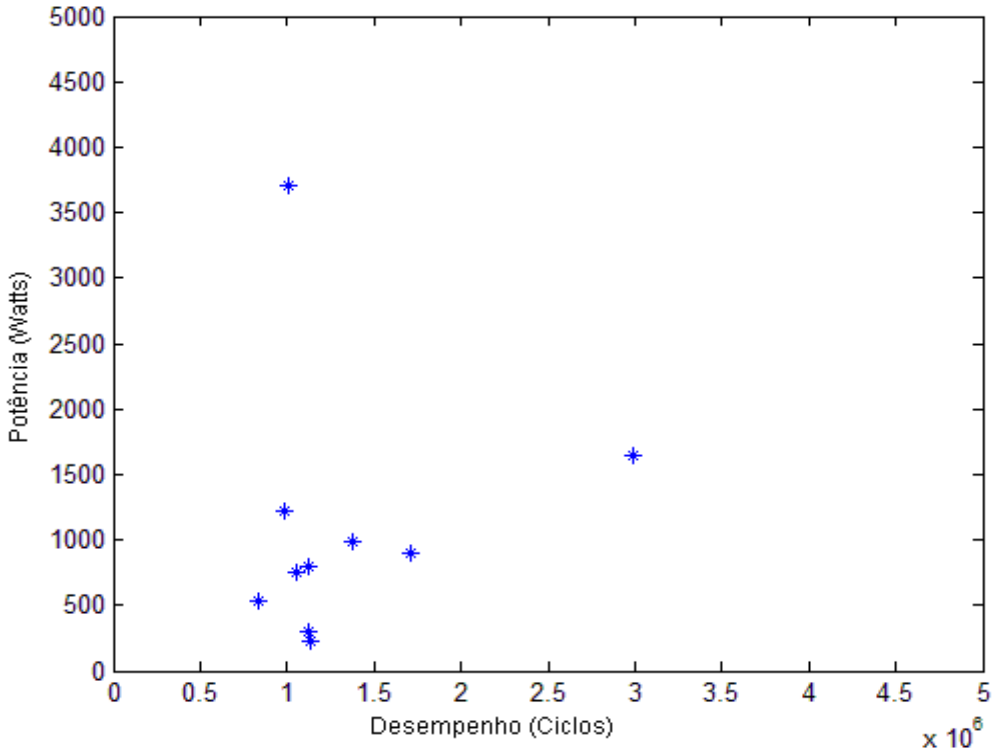

Figura 3.11: Geração 75, relação entre desempenho e potência

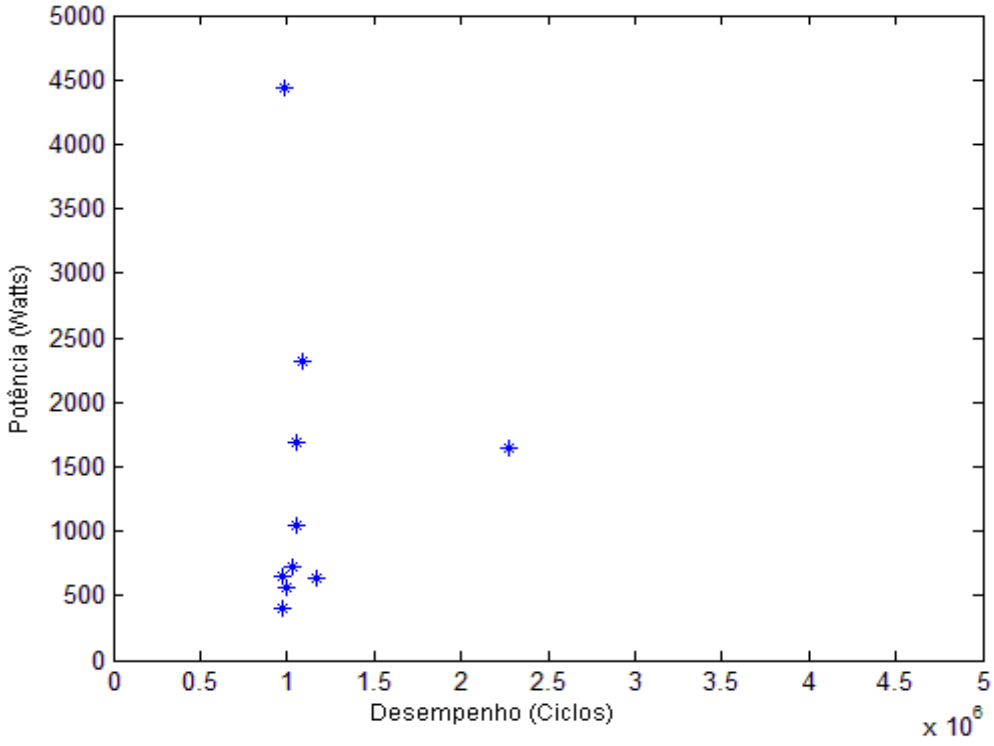

Figura 3.12: Geração 150, relação entre desempenho e potência

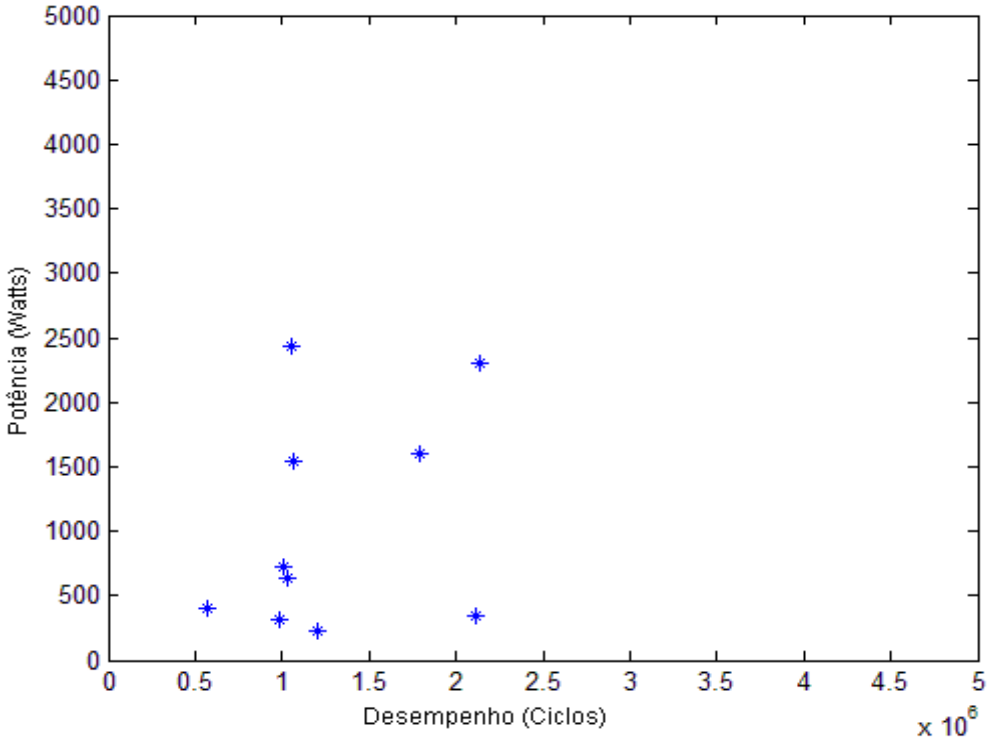

Figura 3.13: Geração 225, relação entre desempenho e potência

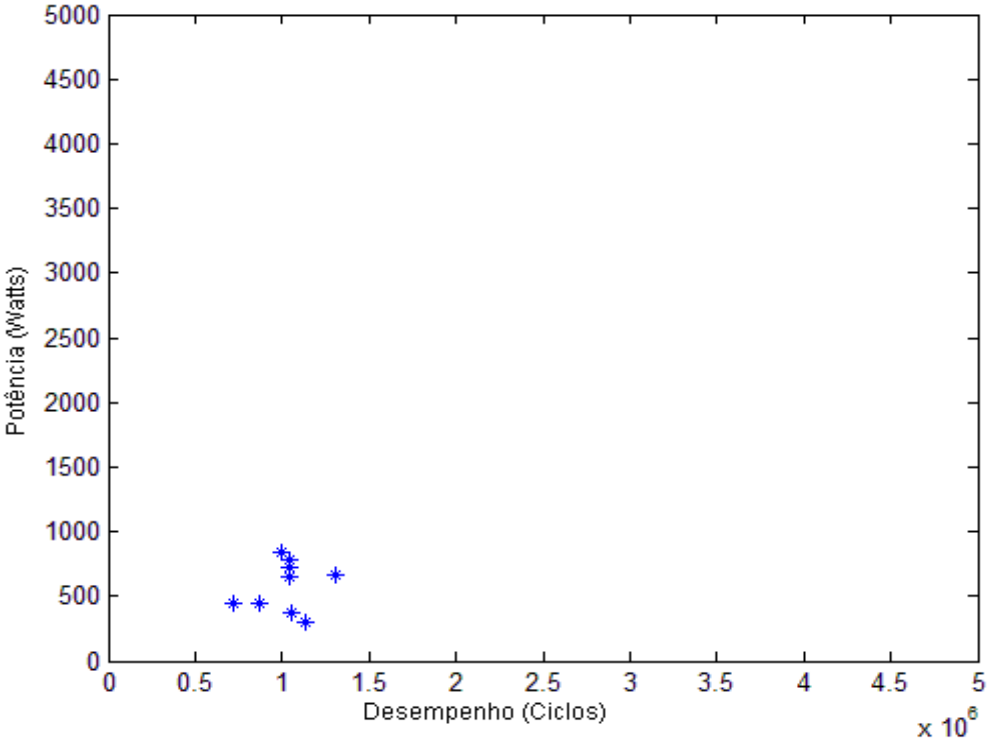

Figura 3.14: Geração 300, relação entre desempenho e potência

Analisando os resultados obtidos pode-se observar que, em cada geração o AG busca tornar os indivíduos o mais balanceados possível, tentando reduzir tanto a potência quanto o desempenho. A melhor arquitetura obtida pelos testes exaustivos teve um desempenho de 461046 ciclos e um consumo de 201,741 watts, comprovando assim a eficiência do AG que obteve uma arquitetura com um desempenho de 48558 ciclos porém, com um consumo de apena 154,262 watts. Este resultado foi obtido na geração 202 e foi encontrada no teste sem o critério de convergência.

3.3 Exploração do espaço de projeto - Redes Neurais Artificiais

Visando eliminar o tempo gasto pelo simulador, optou-se pela utilização de Redes Neurais Artificiais (RNA) para estimar a potência e o desempenho das arquiteturas. Foi decidido pelo modelo MLP (*Multi-Layer Perceptron*), *feedforward* utilizando o algoritmo *backpropagation* para treinamento. Para garantir maior precisão nos resultados foram criadas duas redes, uma para estimar a potência consumida pela arquitetura e outra para estimar o desempenho.

Em ambas as redes foram passados como entrada os parâmetros de configuração de cada processador. Inicialmente, ao serem simuladas 30 arquiteturas, 60% das mesmas são utilizadas para treinamento das redes e 40% são utilizadas para teste (HAYKIN, 2001). Caso o erro gerado pela rede seja maior que o estipulado pelo usuário, o VIPEX-VMP simula 10 novas arquiteturas, das quais 60% são utilizadas para treinamento e 40% para teste. Este processo se repete até que a rede neural artificial esteja totalmente treinada. Para as RNA estarem treinadas, como citado no início do capítulo, o erro entre os resultados estimados pelas RNA e o resultado real, obtido pelo VIPRO-MP, não pode ser maior que o erro estipulado e o número de gerações atual dos AG deve ser maior que o definido pelo usuário. O ambiente utilizado foi o simulador desenvolvido no trabalho realizado por Bonifácio (2010).

Um fator que influencia tanto nos erros apresentados quanto nas estimativas da RNA, é a configuração da mesma. Para a estimativa de desempenho de software, Oyamada (2007) concluiu que uma RNA com maior número de camadas e maior número de neurônios em cada

camada tende a convergir mais rápidamente, entretanto, fica mais específica (*overfit*), ao passo que uma RNA com menos camadas e menos neurônios em cada camada se torna mais abrangente, porém, necessita de mais treinamento.

Para realização dos testes foi utilizada como função de ativação a função logística, definida pela Equação (3).

$$1 / (1 + e^{-x}) \qquad (3)$$

Foram realizados vários testes variando o número de camadas de 1 a 3 e o número de neurônios em cada camada entre 46 e 500. E foi possível observar que os melhores comportamentos foram obtidos mantendo 2 camadas com 46 neurônios cada ou 100 neurônios cada. As Figuras 3.15 a 3.18 representam os gráficos gerados a partir do número de treinamentos em relação à média de erros.

Os grupos de treinamento são compostos pelos indivíduos simulados que estão presentes no banco de dados. O erro médio é calculado sobre a diferença entre as estimativas feitas pela rede para os 40% dos indivíduos, após ter sido treinada, e seus respectivos resultados reais obtidos através do VIPRO-MP.

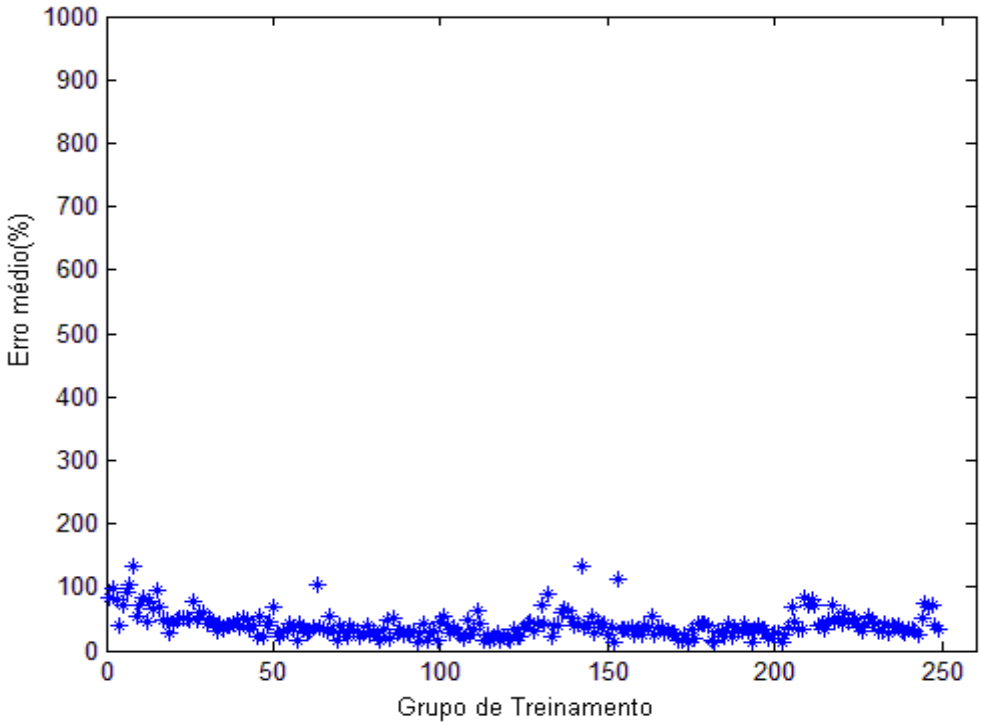

Figura 3.15: Erro médio utilizando o RNA com 2 camadas ocultas de 46 neurônios para estimativa do número de ciclos

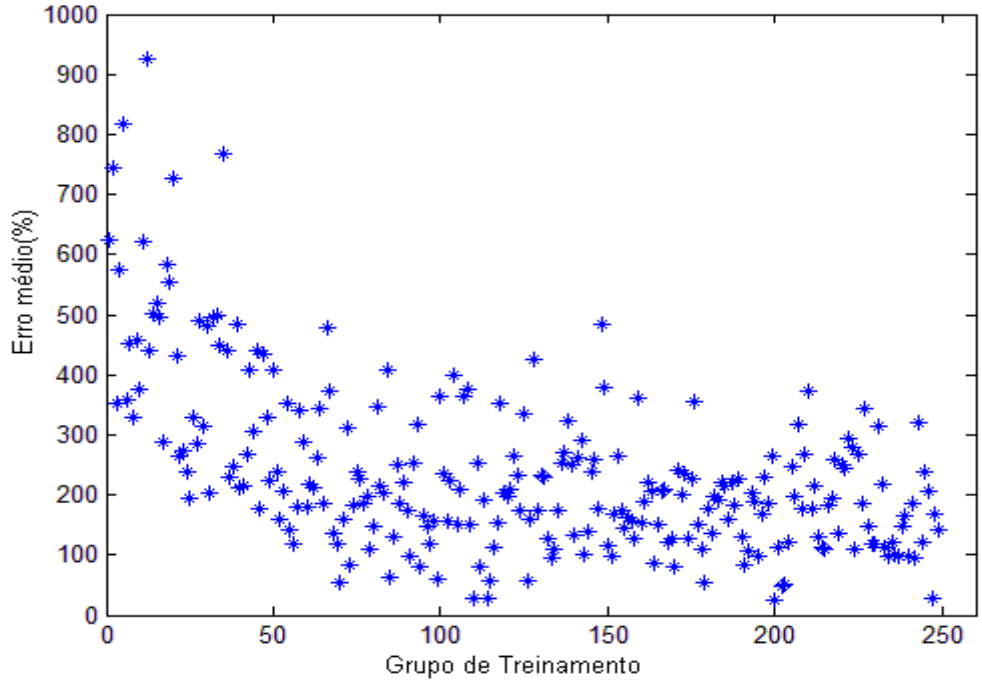

Figura 3.16: Erro médio utilizando o RNA com 2 camadas ocultas de 46 neurônios para estimativa da potência

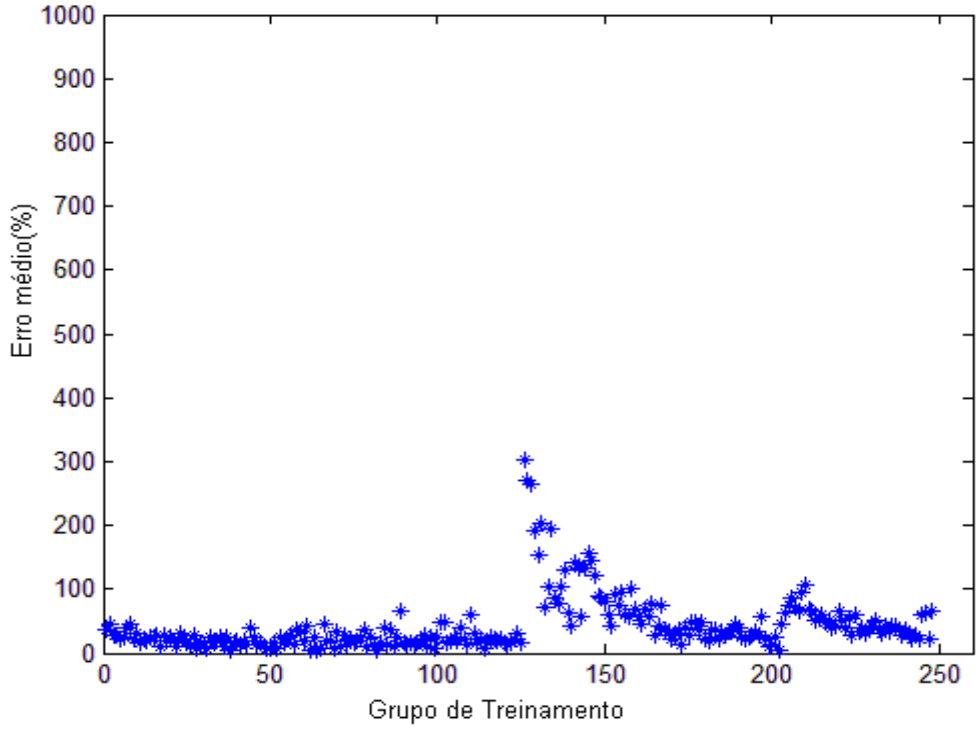

Figura 3.17:Erro médio utilizando o RNA com 2 camadas ocultas de 100 neurônios para estimativa do número de ciclos

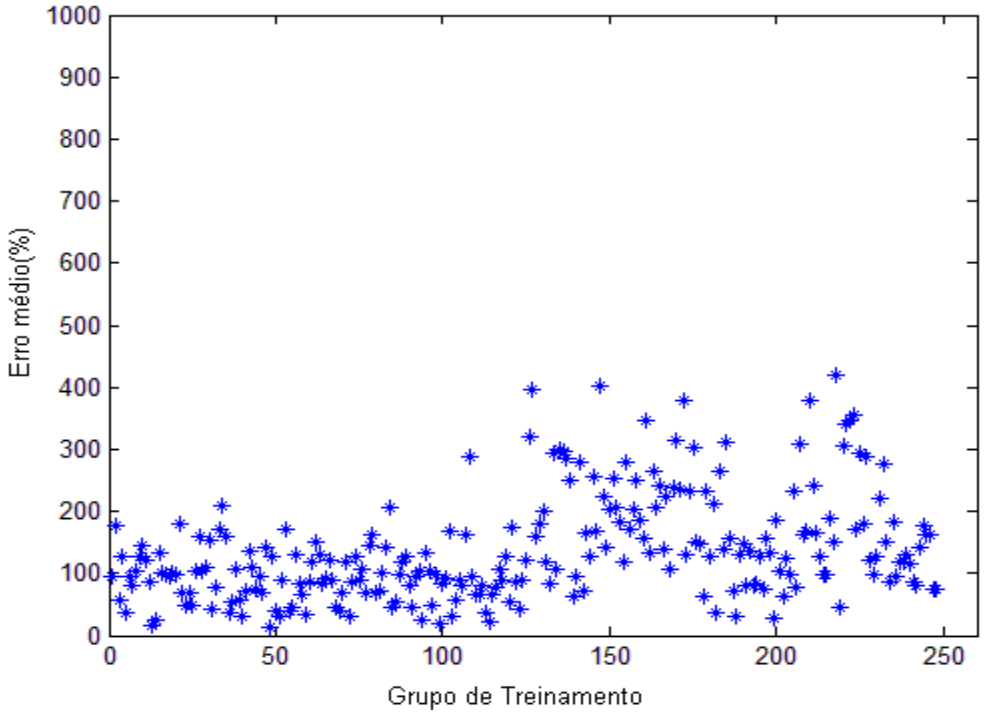

Figura 3.18: Erro médio utilizando o RNA com 2 camadas ocultas de 100 neurônios para estimativa da potência

Analisando os gráficos, foi possível observar que, mesmo obtendo o melhor comportamento entre as configurações utilizadas nos testes, a função logística não se adaptou bem ao problema. Os erros máximos e mínimos no melhor caso foram de 934,7% e 0,045%. Com base nesses resultados é necessário realizar mais testes utilizando outras funções de ativação, outras configurações e até outros modelos de RNA , sendo tal exploração colocada como trabalho futuro.

Capítulo 4

Conclusões e Trabalhos Futuros

Cada vez mais os projetistas se preocupam em desenvolver um SE que possua o máximo desempenho juntamente com o mínimo de consumo de potência. Para isso são amplamente utilizadas plataformas virtuais, que permitem uma rápida configuração e provêem resultados confiáveis. Entretanto, mesmo facilitando a simulação de uma arquitetura, faz-se necessário adicionar a essas plataformas métodos que possam otimizar a arquitetura base de forma eficiente.

A plataforma VIPRO-MP permite a avaliação de vários modelos de arquiteturas, oferecendo uma simulação eficiente, provendo a melhor relação custo/benefícioentre os requisitos para a aplicação. Entretanto, a configuração base de entrada no VIPRO-MP deve ser toda descrita em linguagem C++, tornando muito mais custosa a simulação de muitas arquiteturas.

Visando facilitar a simulação pelo VIPRO-MP, o VIPEX-VMP foi desenvolvido. Este sistema possui um ambiente que permite ao projetista criar e configurar arquiteturas com rapidez e facilidade, a partir da qual são geradas as entradas configuradas para o VIPRO-MP de forma automática. Porém, o tempo gasto para realizar a otimização de um arquitetura tende a ser grande, por ser necessária a simulação de todas as possíveis arquiteturas, exaustivamente.

Os algoritmos genéticos (AG) foram utilizados para gerar arquiteturas e diminuir o tempo de exploração do espaço de projeto, apresentando melhores resultados quando comparados aos testes exaustivos, comprovando sua eficiência. Entretanto, como em alguns casos, existe a possibilidade da arquitetura encontrada pelo AG não ser melhor do que a encontrada pelos testes exaustivos.

Este trabalho também realizou um estudo da aplicação da RNA MLP para estimativa de desempenho e potência. As RNA foram implementadas para substituir o simulador após estarem treinadas. Nos testes realizados, não foi possível obter uma RNA com precisão

suficiente para substituir o simulador, tornando necessário realizar testes com configurações e modelos diferentes.

A utilização do sistema, após concluído, tornou possível a otimização da arquitetura de entrada, composta por dois processadores e uma memória, interligados por um barramento, em menos de 2000 simulações atingindo resultados satisfatórios, se comparados com a simulação exaustiva.

Como trabalhos futuros podem ser realizadas melhorias, principalmente com relação à interface gráfica, que, para se tornar uma ferramenta de uso comercial deve prover maior usabilidade ao projetista. Também deve ser modelada de modo que tanto os parâmetros a serem variados na simulação como seus respectivos intervalos sejam definidos pelo projetista. Na parte do AG, podem ser testados outros métodos de seleção, cruzamento e mutação, assim como outras configurações.

Com relação à RNA MLP, devem ser testadas outras funções de ativação, com outras configurações e até mesmo outro modelo de RNA, pois o modelo utilizado nos testes não se adaptou ao problema. Pode ser analisada também a forma de implementação da RNA, visando obter uma RNA que seja capaz de substituir o simulador.

No sistema como um todo, também deve ser ampliada a capacidade de avaliação VIPEX-VMP para prover resultados mais confiáveis, analisando não somente a potência consumida e o desempenho, como também a área ocupada e os custos.

Referências Bibliograficas

ARM, Disponível em <www.arm.com>, acessado em: 7 de junho de 2010.

BENINI, L.; BOGLIOLO, A; DE MICHELI, G. A ***Survey of Design Techniques for System-Level Dynamic Power Management. IEEE Transactions on Very Large Scale Integration(VLSI) Systems***, Boston, v.8, n. 3, p.299-316, June 2000.

BONIFACIO, F. N. **Comparação entre as Redes Neurais Artificiais MLP, RBF e LVQ na Classificação de Dados.** Universidade Estadual do Oeste do Paraná – Centro de Ciências Exatas e Tecnológicas - CCET, Curso de Ciência da Computação, Novembro, 2010. Monografia de Graduação.

BUENO, F. **Métodos Heurísticos, teoria e implementação**, 2009.

CARRO, L ; WAGNER, F, R. . Capitulo 2 das **Jornadas de Atualização em Informática**, In: JAI 2003. Sistemas Computacionais Embarcados. Campinas: Sociedade Brasileira de Computação, 2003, v .1.

CONVERGENSC, Disponível em <http://www.soccentral.com/ results.asp?EntryID=5229>, acessado em: 10 de julho de 2010

CORRE, E. **Redes-em-chip para sistemas embarcados visando a otimização de medidas de qualidade de serviço para aplicações de tempo real**, 2007. Disponível em <http://www.lume.ufrgs.br/handle/10183/13659>, acessado em: 10 de julho de 2010.

COWARE, Disponível em <www.coware.com>, acessado em: 10 de julho de 2010

DARWIN, C. (2004), "**A origem das espécies**". Rio de Janeiro: Ediouro.

GARCIA, M. S. **VIPRO-MP: uma plataforma virtual multiprocessada baseada na arquitetura SimpleScalar**. Universidade Estadual do Oeste do Paraná – Centro de Ciências Exatas e Tecnológicas - CCET, Curso de Informática, Novembro, 2008. Monografia de Graduação.

GARCIA, M. S.; SCHUCK, M ; OYAMADA, M. S. **VIPRO-MP: a virtual prototype for multiprocessor architectures based on the SimpleScalar**. In: 17th Annual IFIP International Conference on Very Large Scale Integration VLSI-SoC, 2009, Florianopilis. Proceedings of VLSI-SoC 2009, 2009. v. 1.

GELSINGER, P.P.; GARGINI, P.A.; PARKER, G.H.; YU, A.Y.C.; , ***"Microprocessors circa 2000" Spectrum****, IEEE* , vol.26, no.10, pp.43-47, Oct 1989

GIRVARGIS, T ; VARID, F. PLATUNE: ***A Tuning Framework for Systems-on-a-Chip Platforms. In: IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems***, VOL. 21, NO. 11, 2002. Disponivel em http://www.cs.ucr.edu/~vahid/pubs/tcad02_platune.pdf. Acessado em Março de 2010.

HAREL, D. (1987). ***StateCharts: A visual formalism for complex systems.*** *Science of Computer Programming*, pages 1140-1151.

HAYKIN, S. **Redes neurais: princípios e prática**. 2.ed. Porto Alegre, Bookman, 2001

HOFFMAN, A. ***A Novel Methodology for Design of Aplication Specific Instruction Set Processor (ASIP) Using a Machine Description Language***. In IEEE Transactions on Computer-Aided-Design, p. 1338-1354, novembro, 2001.

HUSMANN, R. Apostila de VHDL, 2001.

Intel PXA250, Disponível em <http://www.alldatasheet.com/datashe et-pdf/pdf/144909/INTEL/PXA250.html>, acessado em: 12 de junho de 2010

Intel, Disponível em <http://techresearch.intel.com/articles/Tera-Scale/1449.h tm>, acessado em: 10 de junho de 2010

JUNIOR, A, A, A ; GARIBOTTI, R, F. .**Memorias cache em uma plataforma multiprocessada (MPSoC)**. Universidade Católica do Rio Grande do Sul, Porto Alegre, março de 2007, Monografia de Graduação.

LEI, T., YANHUI, Y., SHAOJUN, W. ***Optimizing SoC Platform Architecture for Multimedia Applications***. Shangai, 2005, v.1.

LINDEN, R. **Algoritmos genéticos: uma importante ferramenta da inteligência computacional**. 2 ed. Rio de Janeiro: Brasport, 2008. xxii, 400 p. edição 2, 2008.

MARRAGHELLO, Norian. Redes de Petri: Conceitos e Aplicações, 2005

MARWEDEL, P. ***Embedded System Design***, Editora Springer 2006.

MICHALEWICZ, Z. (1996), "***Genetic algorithm + data structures = evolution programs***", 3º ed. Springer-Verlag.

MOORE, G, E. ***Cramming more components onto integrated circuits***, Eletronics, Volume 38, Number 8, April 19, 1965.

NETBEANS, Disponível em <http://netbeans.org/>, acessado em: 28 de outubro de 2010

NORMADIK, Disponível em <http://imec.be/ScientificReports/SR2 007/html/pdf/Brochures/Nomadic%20Embedded%20System.pdf>, acessado em: 5 de julho de 2010

OMAP, Disponível em <http://www.eetimes.com/design/signal-processing-dsp/4017760/OMAP-4-ups-performace-with-dual-Cortex-A9-cores>, acessado em: 15 de julho de 2010

OYAMADA, M, S. ***Software Performace Estimation in MPSoC Design***. Tese de Doutorado – Universidade Federal do Rio Grande do Sul, Porto Alegre, Dezembro, 2007.

PALESI, M., GIRVARGIS, T. ***Multi-Objective Design Space Exploration Using Genetics Algoritms***, 2003.

REGO, R. S. L. S. **Projeto e Implementação de uma Plataforma MP-SoC usando SystemC**. Natal: Universidade Federal do Rio Grande do Norte, Janeiro, 2006. Dissertação.

Register Transfer Level. Disponivel em <http://www.cs.ucla.edu/Logic_Desig n/SLPDF/ch13.pdf, acessado em julho/2010>, acessado em: 15 d julho de 2010

SILVA-FILHO, A. G.; BASTOS-FILHO, C. J. A; FALCÃO D. M. A; CORDEIRO F. R.; CASTRO M. S. C. ***An Optimization Mechanism Intended for Two-Level Cache Hierarchy to Improve Energy and Performance using the NSGAII Algorithm,*** *University of Pernambuco – Departament of Compuing and Systems.*

SYSTEMC. Disponivel em <http://www.systemc.org/home>, acessado em: 17 de julho de 2010

UML, Disponível em <www-01.ibm.com/software/rational/uml/>, acessado em: 28 de julho de 2010.

WHITLEY, D. (1994), "***A genetic algorithm tutorial***", Springer Science + Business Media B.V., Formerly Kluwer Academic. p. 65-85.

WOLF, W. ***Computers as Componentes Principles of Embedded Computing System Design***, 2001.

ZHANG, C., VAHID, F., LYSECKY, R.L. ***A Self-Tuning Cache Architecture for Embedded Systems***. In: design automation and test in europe, 2004, Paris, France. Proceedings... IEEE Computer Society Press, 2004. p. 142-147.