UNIVERSIDADE FEDERAL DE PERNAMBUCO

GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO

CENTRO DE INFORMÁTICA

**Framework para Análise e Exploração de Arquitetura visando Aspectos de Consumo de Energia e Desempenho**

TRABALHO DE GRADUAÇÃO

ALUNO: Gabriel Rattacaso Carvalho {grc@cin.ufpe.br}

ORIENTADOR: Abel Guilhermino da Silva Filho {agsf@cin.ufpe.br}

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

**Framework para Análise e Exploração de Arquitetura visando Aspectos de Consumo de Energia e Desempenho**

Gabriel Rattacaso Carvalho

*Monografia apresentada ao Centro de*

*Informática da Universidade Federal de*

*Pernambuco como requisito parcial*

*para a obtenção do título de Engenheiro*

*da Computação.*

*Orientador: Abel Guilhermino da Silva Filho*

**Recife, dezembro de 2010**

*“Não importa o quanto você bate, mas sim o quanto aguenta apanhar e continuar. O quanto pode suportar e seguir em frente. É assim que se ganha.”*

**Rocky Balboa**

À minha família.

Agradecimentos

Esta parte do trabalho é uma das mais difíceis de serem escritas, pois, é de suma importância que sejam lembradas todas as pessoas que me fizeram chegar onde estou no curso de graduação e, esta tarefa, não é a das mais fáceis de serem feitas.

Primeiramente, gostaria de agradecer a toda a minha família que sempre me ajudou nas dificuldades passadas pelo curso, me incentivando a cumprir com todas as obrigações. Essas pessoas são muito importantes na minha vida. Agradeço ao meu pai, Anselmo de Oliveira Carvalho; à minha mãe, Maria Cristina Antonino Rattacaso Carvalho; ao meu irmão, Anselmo de Oliveira Carvalho Filho e, a todos os outros membros da família.

Agradeço, também, a minha namorada, Mariana de Almeida Dornelas Camara, pelo mesmo motivo de incentivo incondicional e pelo entendimento de não estar presente com ela em muitos finais de semana, devido à realização dos projetos e estudos para as provas.

Agradeço aos meus amigos que me acompanharam ao longo do curso, pelas várias noites em claro para concluir as obrigações; e aos amigos que encontrava no dia a dia da faculdade, por me fazerem ter um pouco de distração e brincadeiras.

Agradeço ao meu orientador, Abel Guilhermino, pela paciência e orientação que me proporcionaram fazer um trabalho de graduação com esforço, dedicação e qualidade.

Por fim, agradeço a Deus pela saúde, oportunidade e responsabilidade que me foram concedidas para no decorrer do curso.

Obrigado!!!

Resumo

Estudos recentes, no que diz respeito ao consumo de energia exigido para uma arquitetura, demonstram que estruturas de memória cache podem consumir uma parcela muito grande da energia de um microprocessador. Além disso, o ajuste dos parâmetros de uma memória cache para uma aplicação específica pode economizar em média 60% do consumo de energia do sistema. Portanto, estratégias para explorar o espaço de projeto de configurações devem ser adotadas para reduzir o consumo de energia sem degradação do desempenho e com custo computacional reduzido. Porém, para simular tais estratégias demandam um longo tempo para a realização de *scripts* e para manipulação dos resultados para análise.

É neste contexto que este trabalho propõe um ambiente capaz de prover um ambiente de simulação e exploração adequado a análise de arquiteturas visando aspectos de consumo de energia e desempenho da aplicação. Consiste na implementação de um mecanismo automático de exploração de arquitetura de memória cache, onde o projetista escolhe diversos parâmetros de configuração, bem como a hierarquia desejada. Com isso, a ferramenta realiza as simulações obtendo-se os resultados de energia e o número total de ciclos. Através dos resultados, ainda é possível gerar diversos tipos de gráficos para a visualização e confrontá-los, podendo fazer diversas comparações entre as diversas opções de configurações possíveis, dando um poder de análise ao projetista de escolher a configuração que lhe for mais vantajosa.

Abstract

Recent studies, with regard to energy consumption required for an architecture show that cache memory structures can consume a very large portion of the energy of a microprocessor. Furthermore, adjusting the parameters of a cache for a specific application can save on average 60% of energy consumption of the system. Therefore, strategies to explore the design space of configurations should be adopted to reduce energy consumption without degradation of performance with reduced computational cost. However, to simulate these strategies require a long time to perform scripts and to manipulate results for analysis.

In this context, this work proposes an environment capable of providing a simulation environment suitable for analysis and exploration of architectures aiming aspects of energy consumption and application performance. Is to implement an automatic mechanism to exploit cache memory architecture, where the designer chooses various configuration parameters and the desired hierarchy. Thus, the tool performs the simulations obtaining the results of energy and the total number of cycles. Through the results, it is still possible to generate different types of graphics for viewing and confront them and can make several comparisons between the various possible configuration options, giving a power of analysis to the designer to choose the configuration that it is advantageous.

Sumário

[Lista de Figuras 10](#_Toc279254211)

[Lista de Tabelas 12](#_Toc279254212)

[1. Introdução 13](#_Toc279254213)

[1.1 Motivação 13](#_Toc279254214)

[1.2 Objetivos Gerais 15](#_Toc279254215)

[1.3 Trabalhos Relacionados 15](#_Toc279254216)

[1.4 Estrutura do Trabalho 24](#_Toc279254217)

[2. Ambiente de Exploração 25](#_Toc279254218)

[2.1 Módulos da ferramenta 25](#_Toc279254219)

[2.1.1 Interface Gráfica 26](#_Toc279254220)

[2.1.2 Simulação 26](#_Toc279254221)

[2.1.3 Projeto 27](#_Toc279254222)

[2.1.4 Aplicação 27](#_Toc279254223)

[2.1.5 Util 27](#_Toc279254224)

[2.2 Hierarquias 27](#_Toc279254225)

[2.2.1 Hierarquia de dois níveis separados 28](#_Toc279254226)

[2.2.2 Hierarquia de dois níveis, com o segundo nível unificado 29](#_Toc279254227)

[2.2.3 Hierarquia com um nível separado 29](#_Toc279254228)

[2.3 Ferramentas 30](#_Toc279254229)

[2.3.1 SimpleScalar 31](#_Toc279254230)

[2.3.2 eCACTI 33](#_Toc279254231)

[2.4 Funcionamento 35](#_Toc279254232)

[2.4.1 Tipos de Simulação 38](#_Toc279254233)

[2.4.2 Arquivos de Registro 46](#_Toc279254234)

[3. Resultados 48](#_Toc279254235)

[4. Conclusões e Trabalhos Futuros 55](#_Toc279254236)

[5. Referências 56](#_Toc279254237)

[ANEXOS 59](#_Toc279254238)

[Anexo 1: Arquivo do SimpleScalar. 59](#_Toc279254239)

[Anexo 2: Arquivo do eCACTI. 63](#_Toc279254240)

[Anexo 3: Exemplo de arquivo XML gerado de um projeto. 66](#_Toc279254241)

[Anexo 4: Exemplo de arquivo XML gerado para resultado da simulação de uma aplicação. 68](#_Toc279254242)

# Lista de Figuras

**Figura 1.** Arquitetura do ARM920T (Fonte: Segars ISSCC’01).

**Figura 2.** View do Pdesigner.

**Figura 3.** Exploração de Arquitetura de Memória na decodificação de voz.

**Figura 4.** Resultados para a aplicação de compressão de JPEG.

**Figura 5.** Localidade de superfície, localidade temporal e localidade espacial para um aaaaaaaaalgoritmo do filtro da mediana.

**Figura 6.** Comparação entre TEMGA, TECH-CYCLES e NSGAII para o Susan\_Small.

**Figura 7.** Consumo de energia normalizado com uma configuração base(linha em aaaaaaaaanegrito) para a heurística de exploração de cache e a configuração ótima de aaaaaaaaacache.

**Figura 8.** Arquitetura dos módulos de implementação do sistema.

**Figura 9.** Hierarquia de dois níveis separados.

**Figura 10.** Hierarquia com dois níveis, com o segundo nível unificado.

**Figura 11.** Hierarquia com um nível separado.

**Figura 12.** Fórmula para determinar o Tamanho do Conjunto de uma memória cache.

**Figura 13.** Linha de comando para uso do simulador sim-cache do SimpleScalar.

**Figura 14.** Linha de comando do eCACTI.

**Figura 15.** Parâmetros para a linha de comando do eCACTI.

**Figura 16:** Visão Geral da interface do ambiente.

**Figura 17:** Configurando a simulação do ambiente.

**Figura 18:** Configurando hierarquia de memória e tecnologia de transistor.

**Figura 19.** Adicionando uma simulação do tipo MOEA.

**Figura 20.** Método manual para adicionar novos tipos de simulações.

**Figura 21.** Arquivo das simulações criadas na interface.

**Figura 22.** Arquivo com caminho da pasta de cada projeto.

**Figura 23.** Telas para configuração de parâmetros para gerar gráfico personalizado do aaaaaaaaaresultado. Escolha da cache para se ver o resultado. Em (a), caso seja uma aaaaaaaaahierarquia com caches separadas e em (b), caso seja uma hierarquia com aaaaaaaaacaches unificadas. Em (c), temos a definição dos eixos do gráfico.

**Figura 24.** Gráfico gerado a partir do ambiente.

**Figura 25.** Sobreposição de dois gráficos diferentes.

**Figura 25.** Efeito de aproximação nos gráficos.

**Figura 26:** Variação do tipo de gráfico: taxa de *misses* X tamanho da cache.

# Lista de Tabelas

**Tabela 1.** Confronto das ferramentas apresentadas.

**Tabela 2.** Algoritmo do NSGA-II.

**Tabela 3.** Algoritmo do SPEA2.

**Tabela 4.** Algoritmo do PAES.

**Tabela 5.** Algoritmo do PESA-II.

**Tabela 6.** Algoritmo do NPGA.

**Tabela 7.** Comparação do estado da arte com o ambiente proposto.

# Introdução

O objetivo deste capítulo visa, primeiramente, em abordar as motivações que levaram ao desenvolvimento deste trabalho, identificando os motivos relativos à necessidade do desenvolvimento de um ambiente de análise e exploração de arquiteturas visando o consumo de energia e desempenho. Além disso, será mostrado os objetivos gerais do trabalho, os trabalhos relacionados no estado da arte, e por fim, a estrutura a ser seguida ao longo do documento.

## Motivação

O grau de complexidade das aplicações atuais está extremamente alto, o que tem incentivado o aumento na geração de novas ferramentas de CAD e novas metodologias de projeto. Uma dessas metodologias consiste no desenvolvimento de ferramentas de simulação de arquiteturas, onde se pode extrair o comportamento da mesma observando diversos objetivos, tais como energia, área, desempenho, entre outros.

Um dos grandes problemas no desenvolvimento dos atuais sistemas embarcados é com relação ao consumo de energia [1]. Estes sistemas precisam de uma estrutura e configuração de memória adequada para cada aplicação específica. Para auxiliar nesta escolha de projeto, uma estratégia de exploração de memória é necessária [5].

Dados bibliográficos demonstram que estruturas de memória cache podem consumir acima de 50% da energia de um microprocessador [2], e o processador, outra parcela considerável como na Figura 1, que ilustra, a contribuição de cada componente do microprocessador ARM9TDMI da ARM.

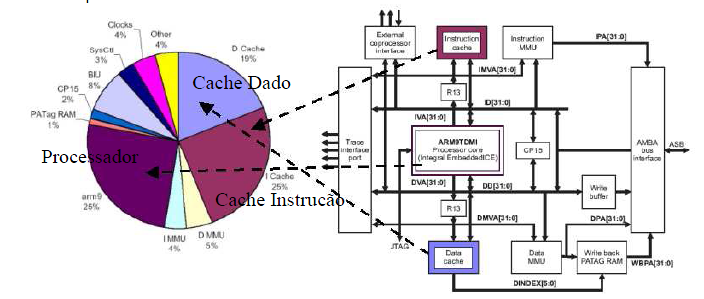
Isto tem guiado muitos pesquisadores e projetistas de processadores a analisar e entender a relação entre as várias configurações, estruturas de memória e o consumo de energia envolvido. A composição da memória pode conter memória principal e memória auxiliar tal como memória cache. Trabalhos relatam que o ajuste dos parâmetros de uma memória cache para uma aplicação específica pode economizar em média 60% do consumo de energia do sistema [5].

Encontrar uma configuração adequada (tamanho total da cache, tamanho da linha e associatividade) para uma aplicação específica não é uma tarefa fácil e pode levar um tempo computacional muito elevado para análise e simulação. Por esta razão, a exploração exaustiva de todas as possíveis configurações de cache no espaço de projeto não é uma solução adequada. Com isso, estratégias para explorar o espaço de projeto de configurações devem ser adotadas para reduzir o consumo de energia sem degradação do desempenho e com custo computacional reduzido.

Daí que surge a abordagem do desenvolvimento de heurísticas a fim de se obter uma redução da quantidade de simulações necessárias para que o projetista obtenha resultados ótimos da aplicação desejada, simulando apenas uma pequena parcela do espaço amostral.

De uma maneira geral, tem sido o objetivo de projetistas e engenheiros de sistemas, a exploração do espaço de projeto, de arquiteturas e comparação com soluções alternativas que possam ser as mais adequadas para um determinado conjunto de aplicações, mantendo um desempenho aceitável e um tempo de projeto competitivo.

Portanto, boas heurísticas, associadas com um ambiente capaz de extrair comportamento da arquitetura, pode ser uma boa solução para encontrar resultados ótimos em uma fração reduzida do espaço de exploração. Neste sentido, um ambiente que integre ferramentas de simulação de arquitetura específica com estratégias para extração de aspectos tais como desempenho e consumo de energia se faz necessário.



**Figura 1:** Arquitetura do ARM920T (Fonte: Segars ISSCC’01)

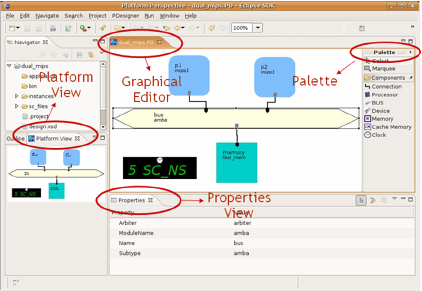
## Objetivos Gerais

O objetivo deste trabalho está focado no desenvolvimento de um framework capaz de prover um ambiente de simulação e exploração a adequado a análise de arquiteturas visando aspectos de consumo de energia e desempenho da aplicação.

Além disso, o framework será capaz de realizar exploração de diferentes arquiteturas bem como prover um ambiente propício para agregar as mais diversas pesquisas voltadas ao problema de exploração de arquitetura desenvolvida pelo grupo de pesquisa.

## Trabalhos Relacionados

Na maioria dos casos, torna-se inviável analisar todas as configurações possíveis de um determinado problema, surgindo a necessidade de criação de ambientes de desenvolvimento que automatizem ou encontrem as melhores configurações para o problema desejado. Pensando nisso, vários trabalhos têm sido desenvolvidos no estado da arte, como em [6], onde foi desenvolvido o Pdesigner, que consiste de um conjunto de ferramentas que possibilita a modelagem, simulação e análise de plataformas multiprocessadas SOC(System on Chip) em diferentes níveis de abstração. Ela foi desenvolvida sobre a IDE do Eclipse, com a criação de plugins, que se comunicam através de arquivos XML.



**Figura 2:** View do Pdesigner.

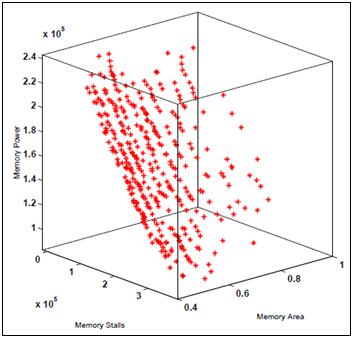
Os três principais plugins,o PBuilder, baseado em SystemC, é utilizado para modelagem e simulação de plataformas, onde o projetista pode modelá-las num fluxo bem definido que vai desde a descrição dos componentes até a geração dos simuladores. Outro plugin é o PDLibrary, que armazena todos os componentes utilizados na modelação, facilitando a inserção e remoção de IP-Cores feitas através do SPIRIT. E por último, o PArchC que consiste de um plugin de modelagem de processadores descritos na linguagem ArchC 2.0. Ainda existe dois outros plugins, para análise de memórias cache, o PCacheAnalyzer, tendo informações de acertos e faltas das configurações de cache, e o PCacheEnergyAnalyzer, que provê estimativas de consumo de energia de memórias caches. A arquitetura descrita pode ser vista na figura 2. O Pdesigner ainda oferece uma interface gráfica que ajuda bastante na construção de seus sistemas, como pode ser visto na figura 2. Nela, temos o *Graphical Editor*, onde o projetista pode editar sua arquitetura, temos a p*alette*, que consiste dos componentes que podem ser adicionados; a *properties view*, com as configurações relativas aos componentes e a *platform view*, onde o projetista pode acompanhar a forma da sua arquitetura.

Portanto, pode-se notar que o Pdesigner, dado uma aplicação de interesse, permite ao projetista a definir os componentes, a arquitetura da plataforma com base nos resultados obtidos. A ferramenta não faz uso de heurísticas, apesar de ter o suporte para obter essa característica.

Em [8], *T.S. Rajesh* diz que um dos principais fatores que impulsiona o custo e a dissipação de energia em um SoC é a arquitetura de memória. Esse fato pode ser tratado como um problema de otimização de dois níveis, sendo um no nível externo, explorando diversas composições de arquiteturas; e no nível interno, explorando as configurações para cada tipo de arquitetura.

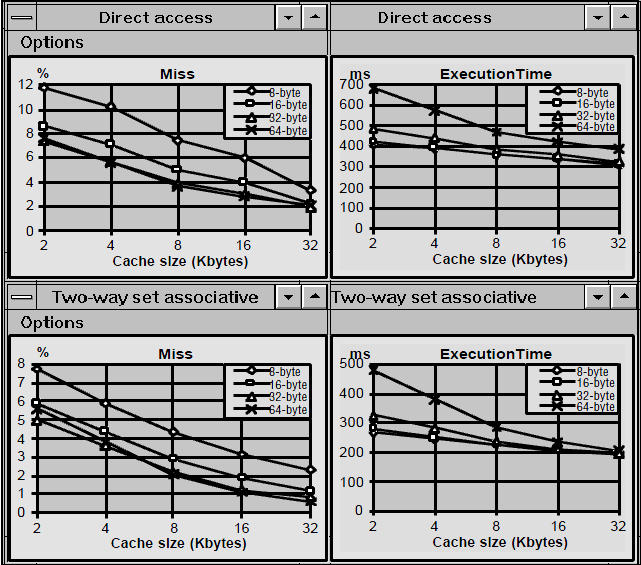
No trabalho desenvolvido por T.S. Rajesh, foi abordado o problema de exploração de arquitetura de memória através da combinação de um Algoritmo Genético multi-objetivo para modelar a exploração de arquitetura de memória tendo como objetivos: desempenho, potência e área, e um Algoritmo Heurístico eficiente, que aborda o problema da distribuição de dados para  
arquitetura de memória com vários bancos de memória, portas dual de memória, latências diferentes e particionamento on-chip/off-chip.

O objetivo do trabalho é de explorar o espaço de memória e obter todas as soluções (arquiteturas de memória) não dominadas que estão no Pareto ótimo com respeito à área, potência e desempenho. Na figura 3, pode ser visto o resultado desse processo para a aplicação de decodificação de voz, onde são mostradas no gráfico todas as soluções não dominadas resultantes para essa aplicação, onde o projetista pode analisar a melhor solução, levando em consideração se ele quer priorizar em um dos eixos do gráfico: potência, *Stalls* ou área ocupada da memória. Com isso, o projetista pode encontrar o ponto ideal maximizando ou minimizando os eixos no gráfico.



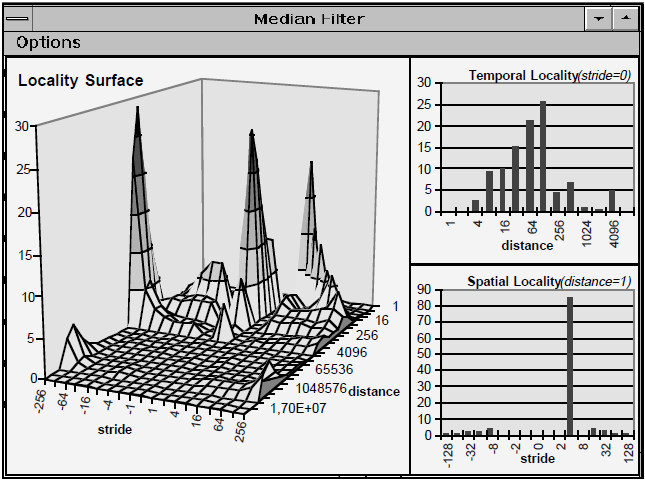
**Figura 3:** Exploração de Arquitetura de Memória na decodificação de voz.

A ferramenta ChARM[7] consiste de uma ferramenta de simulação para configurar sistemas embarcados baseados num ARM que contém memórias cache. A ferramenta tem por objetivo ajudar ao projetista em várias questões como: determinar se uma aplicação pode ser beneficiada com o uso de memória cache; selecionar a configuração ideal do sistema (velocidade da CPU, estrutura de memória cache, velocidade do barramento do sistema e a velocidade da memória principal) para balancear desempenho e custos; investigar como melhorar o desempenho com pequenas mudanças no código fonte ou mapeando a memória do programa; reduzinho o *time-to-market*. O processo do ChARM é composto por 5 fases: fase de desenvolvimento do programa, fase de configuração do sistema, fase de simulação do sistema, fase de análise de resultado e fase de análise de *trace*.

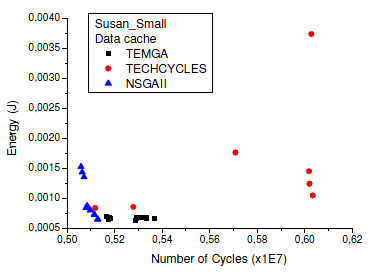
A fase de desenvolvimento do programa consiste da construção da aplicação, debugá-la e gerar um *trace*. Na fase de configuração do sistema, é definida a arquitetura do sistema e as características dos componentes. A fase de simulação do sistema permite ao projetista executar uma simulação ou um experimento de avaliação de desempenho. Na fase de análise de resultados, são produzidos gráficos e diagramas do desempenho do sistema. E por último, a fase de análise de *trace* ajuda ao projetista analisar o comportamento do sistema.

**Figura 4:** Resultados para a aplicação de compressão de JPEG

Nas figuras 4 e 5, pode-se ver alguns exemplos de resultados que a ferramenta dá ao projetista dando um maior poder para analisar as melhores soluções. Na primeira figura, temos a quantidade de misses e o tempo de execução correspondente ao mesmo *trace* de duas configurações de cache diferentes, enquanto que na segunda figura, temos um exemplo sobre uma aplicação de um algoritmo do filtro da mediana, obtendo como resultado a localidade de superfície, localidade temporal e localidade espacial do programa. Com base nesses resultados, vemos que as variações de parâmetros apresentadas influenciam no desempenho da cache.

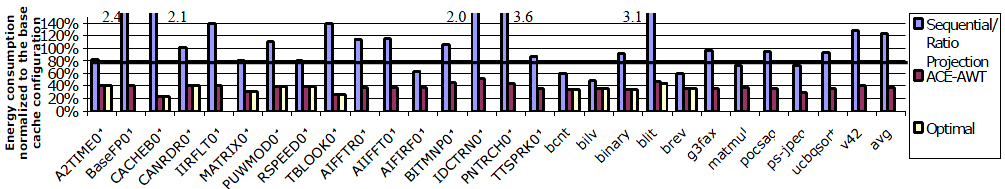
**Figura 5:** Localidade de superfície, localidade temporal e localidade espacial para um algoritmo do filtro da mediana.

Além de criar ferramentas de modelagem dos mais variados problemas, os pesquisadores têm feito esforços para dar contribuições no desenvolvimento de heurísticas para obter os melhores pontos das aplicações sem ter que simular todo o espaço de exploração. Silva-Filho [9] realiza uma aplicação de otimização de memória cache utilizando o algoritmo NSGAII para encontrar soluções de configuração de arquitetura de memória que atinham os objetivos de redução de consumo de energia e aumento do desempenho. No mesmo trabalho é mostrado como uma aplicação real pode se beneficiar através da utilização de algoritmos de otimização multi-objetiva. O experimento foi realizado para 12 diferentes aplicações do *suite* Mibench. A heurística foi validada em 67% das aplicações analisadas em comparação com outras heurísticas analisadas no trabalho. A figura 6 consiste de um dos resultados obtidos, onde é mostrado um gráfico comparativo entre 3 heurísticas, TEMGA, TECHCYCLES e NSGA II, confrontando a energia, em Joules, com o número de ciclos para uma cache de dados com a aplicação Susan\_Small. Nesse tipo de resultado, o projetista poderá analisar qual a heurística se comporta melhor para a aplicação desejada visando um dos objetivos apresentados.



**Figura 6:** Comparação entre TEMGA, TECH-CYCLES e NSGAII para o Susan\_Small.

Outro exemplo pode ser visto no trabalho de Vahid[10], onde é apresentado a ACE-AWT, que consiste de uma heurística para ajustar uma hierarquia de memória cache de 2 níveis com o segundo nível unificado.

**Figura 7:** Consumo de energia normalizado com uma configuração base(linha em negrito) para a heurística de exploração de cache e a configuração ótima de cache.

A heurística foi aplicada a 27 benchmarks rendendo uma média de 62% de economia de energia no acesso a memória quando comparado com uma configuração base de memória cache. A heurística foi eficiente para todos os benchmarks testados, utilizando apenas 0,2% de todo o espaço de exploração. Podemos ver esses resultados na figura 7, onde é mostrado um gráfico comparativo entre o resultado ótimo e a aplicação das heurísticas *Sequential Rate* e *ACE-AWT* para cada um dos benchmarks, onde é mostrado o nível de energia consumido por cada um deles, normalizado para uma configuração base de cache.

Então, frameworks que automatizam todo um processo de geração de resultados para uma determinada aplicação são bastante úteis, pois torna um longo processo de configurações, cálculos e bastante tempo demandado para uma ação automatizada, economizando tempo e esforço, além de prover um ambiente para analisar diferentes soluções para o mesmo problema. Além disso, a inclusão de heurísticas nesses processos pode contribuir na obtenção dos melhores resultados de uma forma mais rápida.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Ferramenta | Diferentes Arquiteturas para Simulação | Análise Simultânea | Método Heurístico | Consumo de Energia | Ambiente Gráfico | Suporte a Multiprocessamento |
| Pdesigner | SIM | - | - | SIM | SIM | SIM |
| MAX | SIM | - | SIM | SIM | - | - |
| ChARM | SIM | - | - | - | SIM | - |
| Silva-Filho | - | - | SIM | SIM | - | - |
| F.Vahid | - | - | SIM | SIM | - | - |

**Tabela 1:** Confronto das ferramentas apresentadas.

Na tabela 1, temos um comparativo dos trabalhos que foram mostrados anteriormente, com a avaliação de 6 características importantes, sendo elas: possibilidade de simulação com diferentes arquiteturas(na tabela: Diferentes Arquiteturas para Simulação), possibilidade de se confrontar resultados de duas simulações diferentes(na tabela: Análise Simultânea), possibilidade de uso de algum método heurístico(na tabela: Método Heurístico), fazer estimativas do consumo de energia de uma dada configuração de simulação(na tabela: Consumo de Energia), se possui uma interface com o usuário gráfica(na tabela: Ambiente Gráfico) e a possibilidade de instanciar mais de um processador na arquitetura(na tabela: Suporte a Multiprocessamento).

Podemos observar que nenhuma ferramenta é capaz de analisar simultaneamente diferentes simulações, o que é uma característica muito importante, já que facilita o projetista na análise da melhor solução de acordo com os seus objetivos. Outro ponto que aparece somente em duas das ferramentas, consiste de um ambiente gráfico, pois essa característica dá uma maior percepção ao projetista de ver exatamente a estrutura que está sendo utilizada em suas simulações. O consumo de energia consiste de algo essencial, já que é um dos maiores problemas encontrados no desenvolvimento de sistemas embarcados [1]. Além disso, as outras características também são muito importantes, pois para ser possível realizar uma boa análise, é preciso, pelo menos, ter mais de um tipo de arquitetura de simulação para comparações. E o uso de método heurístico, pode ser muito útil para diminuir bastante o espaço de exploração de projeto, economizando tempo e esforço, além de dar um maior poder de análise. E temos um diferencial para a ferramenta Pdesigner, que é a única que dá suporte a possibilidade de multiprocessamento, aumentando bastante as possíveis variações do sistema.

## Estrutura do Trabalho

O presente trabalho foi estruturado em capítulos, onde cada um deles será descrito a seguir. O capítulo 2 apresentará a forma em que o ambiente foi desenvolvido, mostrando as ferramentas que foram utilizadas, e a metodologia de como a implementação está feita. O capítulo 3 apresentará o que o ambiente provê para o usuário, mostrando suas potencialidades de análise. E por fim, o capítulo 4 apresentará as considerações finais a serem feitas, bem como propor alguns aspectos que podem ser melhorados na ferramenta.

# Ambiente de Exploração

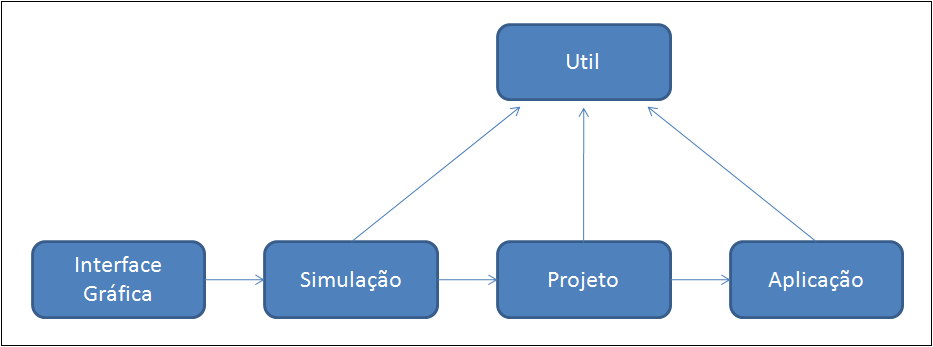
A ferramenta foi desenvolvida na linguagem de programação Java [11], com auxílio do ambiente de programação Eclipse [12]. Foram utilizadas algumas bibliotecas como a Swing Application Framework [13] [14], para gerenciar as threads e a interface gráfica, a JDOM [15] [16], para manipular arquivos XML e a JFreeChart[17] para geração dos gráficos. Além disso, foram utilizadas ferramentas, que serão descritas mais adiante, como a SimpleScalar[18] e a eCACTI[19], para prover informações relativas aos cálculos de energia e ciclos.

Para o desenvolvimento da ferramenta, foi necessário levar algumas características importantes, como modularidade, tendo uma boa organização do código para fácil entendimento, dando a possibilidade de futuras extensões do código desenvolvido, além de possuir uma boa interface, para uma melhor praticidade do uso de suas funcionalidades.

A seguir, serão descritos os módulos da ferramenta, assim como as dependências que venham a aparecer.

## Módulos da ferramenta

A arquitetura do sistema consiste de diversos pacotes que podem ser dispostos em módulos, onde interagem entre si, a fim de prover o funcionamento do sistema. Cada módulo possui sua funcionalidade específica, que agrupa um conjunto de classes relativas às atividades daquele módulo. Na figura 8, vemos como se encontra essa arquitetura dos módulos da implementação do sistema.

**Figura 8:** Arquitetura dos módulos de implementação do sistema.

### Interface Gráfica

É responsável pela interface do sistema com o usuário. Ela possui todas as classes que realizam a construção de janelas gráficas, menus de configuração e exibição de resultados. Também contém classes que utilizam a biblioteca JFreeChart[17], que são responsáveis pela apresentação dos resultados em um gráfico.

### Simulação

É responsável por tudo que envolve simulações no sistema. Tais simulações podem ser a simulação principal do sistema que controla o fluxo do mesmo, e as simulações do propósito do sistema em si, onde é feito todos os passos da simulação de um projeto, contendo classes com cálculos dos objetivos, os tipos de heurísticas que cada simulação pode seguir, uso das ferramentas que o sistema utiliza, organizador para os resultados obtidos com cada simulação, bem como, classes básicas para auxílio do funcionamento desse módulo. Algumas pendências, como as ferramentas e heurísticas utilizadas, iniciadas nesse tópico, serão explicadas mais adiante.

### Projeto

É responsável pelo controle dos projetos, com classes que tem o acesso e que fazem as modificações no repositório de projetos do sistema.

### Aplicação

É responsável pelo controle das aplicações de cada projeto, com classes que tem o acesso e que fazem as modificações no repositório de aplicações do sistema. Também possui algumas classes básicas para funcionamento como a das hierarquias que o sistema suporta e as unidades de cache que compõem as hierarquias. As unidades de cache e as hierarquias serão explicadas na outra seção.

### Util

É responsável por auxiliar o fluxo principal do sistema. Possui todas as classes relativas às operações que contribuem de alguma forma para que o sistema funcione como criação e leitura de arquivos XML que são utilizados para armazenar informações importantes, conversões XML para XLS, assim como diversas outras operações necessárias.

## Hierarquias

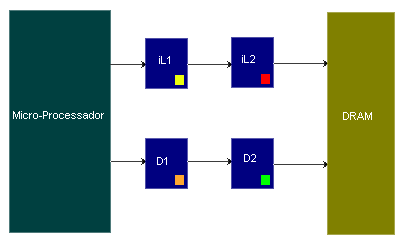
Num ambiente de análise de exploração, é muito importante que se tenha o máximo de variações para que o projetista possa analisar qual das configurações se comporta melhor com uma determinada aplicação. Entre essas variações está o tipo de hierarquia em que se aplica.

Nesse tópico será descrito as hierarquias presentes no ambiente de desenvolvimento.

Vale salientar que a escolha das hierarquias que foram introduzidas no ambiente não foi feita de qualquer jeito. Foi realizado um estudo das hierarquias que estão sendo mais utilizadas no estado da arte, juntamente do que se tem produzido a nível comercial. Além disso, só foi possível utilizar as hierarquias que fossem compatíveis com a ferramenta de interesse do projeto, onde foi preciso eliminar algumas delas, como a hierarquia com três níveis, que apesar de existir trabalhos que adaptaram a ferramenta, não consiste de uma versão livre da ferramenta.

### Hierarquia de dois níveis separados

Essa hierarquia é composta por dois níveis de memória cache, sendo ambos níveis separados. Portanto, cada nível possui uma cache de dados e uma cache de instruções. A figura 9 pode ilustrar como é essa hierarquia.

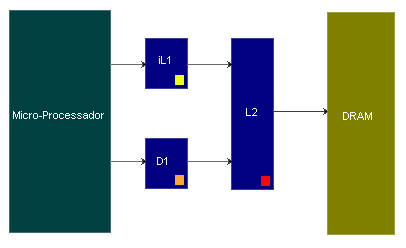


**Figura 9:** Hierarquia de dois níveis separados.

Na figura 9, temos iL1 como cache de instruções do primeiro nível; iL2 como cache de instruções do segundo nível. O mesmo acontece para D1 e D2, sendo caches de dados. No esquema da figura, uma instrução vai ser procurada em iL1, se ocorrer um *miss*, irá ser procurada em iL2. O mesmo acontece quando vai ser procurado um dado. Em ambos os casos, se ocorrer um *miss* na cache do segundo nível, o dado ou instrução, passa a ser procurado na memória principal.

### Hierarquia de dois níveis, com o segundo nível unificado

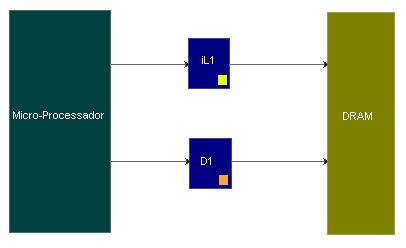
Essa hierarquia é composta por dois níveis, sendo o primeiro nível separado e o segundo nível unificado. Como pode se ver na figura 10, ela é composta por uma cache de instruções (iL1) e uma de dados(D1) no primeiro nível, e uma cache unificada no segundo nível(L2). A diferença para a primeira hierarquia apresentada, é que quando ocorre um *miss* no primeiro nível de ambas as caches, o dado ou instrução passa a ser procurada na cache unificada do segundo nível. Porém, se ocorrer um *miss* no segundo nível, a instrução ou dado será procurada na memória principal do mesmo jeito.



**Figura 10:** Hierarquia com dois níveis, com o segundo nível unificado.

### Hierarquia com um nível separado

Essa hierarquia é composta por um único nível separado. Como mostrado na figura 11, temos apenas uma cache de instruções[iL1] e uma cache de dados[D1]. Nela, quando ocorre um *miss* em uma das caches, a procura passa a ser direto na memória principal.



**Figura 11:** Hierarquia com um nível separado.

## Ferramentas

No cenário de medições na exploração de memória cache, pode-se realizar a medição em baixo nível, diretamente no *hardware*, obtendo resultados bem precisos, porém para se ter uma boa variação de arquiteturas, é necessário um alto tempo para reconfiguração do *hardware* a fim de se obter uma nova variação, demandando um elevado gasto em custo e tempo. Em contrapartida, o uso de ferramentas em alto nível vem se tornando cada vez mais atrativas, já que elas simulam diversas variações de forma simples e rápida com apenas algumas pequenas alterações de parâmetros. De fato, a sua precisão não é tão boa quanto a medição em baixo nível, mas chegam a valores muito próximos do real.

Nessa seção, serão mostradas as duas ferramentas que foram utilizadas no sistema, o SimpleScalar e o eCACTI. A primeira resultará em informações relativas ao funcionamento da arquitetura como *hits*, *misses*; enquanto, a segunda, dará resultados inerentes a cada memória cache da arquitetura como potências: dinâmica e estática.

### SimpleScalar

O SimpleScalar consiste no conjunto de ferramentas públicas, que suportam simulações detalhadas para algumas características encontradas nos processadores. Possui oito simuladores, são eles: sim-fast, sim-safe, sim-profile, sim-cache, sim-cheetah, sim-eio, sim-bpred e o sim-outorder. Dentre esses simuladores, apenas dois deles, o sim-cheetah e o sim-cache, são específicos para análise de memórias cache, gerando parâmetros de erros e acertos. As vantagens das ferramentas do SimpleScalar são sua alta flexibilidade, portabilidade, extensibilidade e desempenho[20].

O sim-cheetah gera resultados de simulação para múltiplas configurações de cache em uma única execução da simulação, abordagem esta chamada de *Single-Pass* *Simulation*[21], ao contrário de outros simuladores, sendo necessário uma grande quantidade de memória para a simulação de aplicações mais complexas. Isto se torna eficaz para um fluxo de instrução somente de leitura, porém não sendo indicado para caches *write-back*, por se tornar mais custosa a substituição de um bloco ao ser escrito na volta para a memória RAM.

O sim-cache recebe parâmetros de entrada das configurações da memória cache, indicando quantidade de níveis, e, se esses níveis são unificados ou separados. Ainda recebe a aplicação que será usada na execução da simulação para a medição das informações relacionadas aos aspectos diversos dos acertos e erros ocorridos no decorrer da simulação.

No presente trabalho, só será utilizado o sim-cache. Este recebe o seguinte formato de configuração <config> de memórias:

<name>:<nsets>:<bsize>:<assoc>:<repl>

Onde:

* <name>: nome da cache. Deve ser único.
* <nsets>: número de conjuntos na cache.\*
* <bsize>: tamanho da linha.
* <assoc>: associatividade da cache (potência de dois).
* <repl>: política de substituição ( l,f,r ), sendo: l=LRU, f=FIFO,

r= randômico/aleatório.

\*O número de conjuntos da cache é obtido através do tamanho da cache dividido pela multiplicação do tamanho de linha com a associatividade da cache, como mostrado na figura 12 a seguir.



**Figura 12:** Fórmula para determinar o Tamanho do Conjunto de uma memória cache.

Para as hierarquias apresentadas no tópico anterior, temos os seus formatos estabelecidos em para configuração das caches.

Para a hierarquia com dois níveis, ambos separados:

-cache:il1 il1: <config> => configura a cache de instrução de nível um.

-cache:dl1 dl1: <config> => configura a cache de dados de nível um.

-cache:il2 il2 <config> => configura a cache de instrução de nível dois.

-cache:dl2 dl2: <config> => configura a cache de dados de nível dois.

Para a hierarquia com dois níveis, com o primeiro nível separado e o segundo nível unificado:

-cache:il1 il1: <config> => configura a cache de instrução de nível um.

-cache:dl1 dl1: <config> => configura a cache de dados de nível um.

-cache:il2 dl2 -cache:dl2 ul2 <config> => configura a cache unificada de nível dois.

Para a hierarquia com um nível separado:

-cache:il1 il1: <config> => configura a cache de instrução de nível um.

-cache:dl1 dl1: <config> => configura a cache de dados de nível um.

Além das configurações da hierarquia, é necessário que outros campos sejam preenchidos na linha de comando. Para isso, será utilizado um exemplo como mostrado na figura 13. Vemos então que além do que foi detalhado, ainda é necessário especificar alguns caminhos de arquivos como do simulador Sim-Cache, do resultado da simulação, da aplicação a ser simulada e das possíveis entradas e saidas da aplicação.

**Figura 13:** Linha de comando para uso do simulador sim-cache do SimpleScalar.

Essa simulação gera um arquivo de saída que é utilizado para obtenção de informações. Estas, retiradas do arquivo, se referem a cada cache, onde é extraído o número de acessos, *misses*, *hits* e taxa de *miss*. No arquivo, está representado respectivamente, como: <nome da cache>.*accesses*, <nome da cache>.*misses*, <nome da cache>.*hits*, <nome da cache>.*miss\_rate*. O termo <nome da cache> depende da cache. No exemplo do arquivo gerado que se encontra presente no anexo 1, temos <nome da cache> = il1, il2, dl1, dl2; representando uma hierarquia com dois níveis separados, significando, respectivamente, a cache de instrução do primeiro nível, instrução do segundo nível, dados do primeiro nível e dados do segundo nível.

### eCACTI

O eCACTI é uma melhoria do modelo CACTI, corrigindo algumas limitações observadas que levavam a um cálculo que desprezava a componente estática e fixava o valor das cargas capacitivas, gerando uma imprecisão no modelo de potência, principalmente quando se trata de nanotecnologia(DSM). Consiste de um modelo analítico de memória cache que gera resultados da energia consumida por acesso, tempo de acesso e área de um componente de memória cache, O eCACTI é utilizado para o cálculo, através de parâmetros de entrada, da potência consumida pela memória, tanto dinâmica quando estática, de cada módulo de memória cache, e da área ocupada, assim como informações de consumo de tempo no decorrer das operações realizadas na cache[20]. Para executar o eCACTI, são necessários os seguintes parâmetros:

<C><B><A><TECH><NSubbanks>

Onde:

• <C>: Tamanho da cache em bytes

• <B>: Tamanho da linha da cache em bytes

• <A>: Associatividade da Cache

• <TECH>: Tecnologia de transistor dada em micrometros

• <NSubbanks>: Número de Subbancos (matriz de armazenamento)

Pode-se ainda acrescentar ao comando do eCACTI uma configuração de organização da cache, além da configuração de parâmetros de entrada descrita acima, a partir de um arquivo que conterá estas especificações, ou executar o comando sem acrescentá-la. Nesta segunda opção, o próprio eCACTI avaliará a melhor configuração de organização para uma otimização levando-se em conta os parâmetros de avaliação, que são a área, o tempo e a potência consumidas. Gerando, com isso, os demais resultados da simulação baseados nesta configuração otimizada. Parâmetros estes que se encontram no formato:

(Ndwl, Nspd, Ndbl, Ntwl, Ntspd, Ntbl)

Onde:

* Ndwl : tamanho do segmento de *wordline*(array de dados);
* Nspd : tamanho do segmento de *bitline*(array de dados);
* Ndbl : número de conjuntos mapeados para casa *wordline*(array de dados);
* Ntwl : tamanho do segmento de *wordline*(array de instruções);
* Ntspd : tamanho do segmento de *bitline*(array de instruções);
* Ntbl : número de conjuntos mapeados para casa *wordline*(array de instruções).

Além disso, tem-se a possibilidade de inserir argumentos específicos, para o tipo de avaliação e resultados que se deseja obter do eCACTI, além do especificado acima, que seria a opção “-verbose”, onde através deste argumento, o eCACTI mostrará os resultados completos de sua simulação, com detalhes de dissipação de potência estática e dinâmica, consumo de tempos de acesso e áreas ocupadas pelo banco de memória, além de gerar configurações otimizadas para cada um de seus componentes de análise: área, potencia e tempo. Este comando pode ser visto na figura 14.



**Figura 14:** Linha de comando do eCACTI.

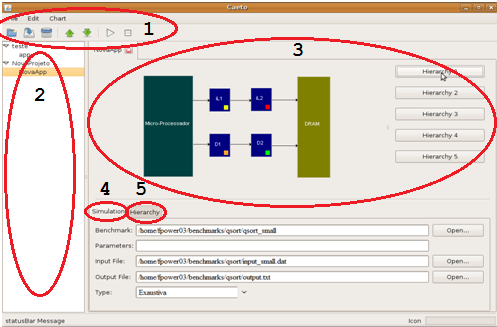
A figura 15 mostra um exemplo de uma linha de comando utilizando uma configuração de uma cache com tamanho de 2048 kbytes, tamanho de linha de 16 bytes, associatividade igual a 1, tecnologia de transistor igual a 0.18 micrômetros, número de subbancos igual a 1. Além desses parâmetros, temos a passagem de parâmetros de um arquivo de texto de organização de cache, já mencionado anteriormente, contendo a opção verbose, como mostrado na figura 14. No arquivo gerado pelo eCACTI, são sempre extraídos os campos relativos ao tempo de acesso, representado por *Acess Time* no arquivo, e as potências dinâmica e estática, que consistem nos dois números, logo após *Total Power all Banks* no arquivo. No anexo 2, temos um exemplo desse resultado de saída do eCACTI.



**Figura 15:** Parâmetros para a linha de comando do eCACTI.

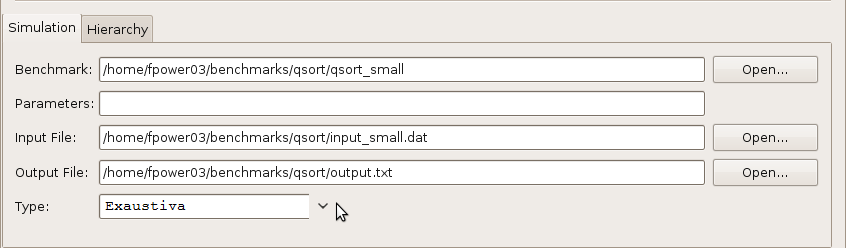
## Ferramenta de Exploração

O ambiente proposto desenvolvido se encontra de forma simples e intuitiva para a realização das suas funcionalidades. Antes de explicar com mais detalhes, uma visão geral da interface do mesmo é mostrado na figura 16. Essa visão é dividida por áreas onde o círculo 1, consiste das opções de criar projetos, aplicações, e as funcionalidades presentes; o círculo 2, consiste nos projetos e suas aplicações que foram criadas; o círculo 3, consiste no lugar onde será selecionado a hierarquia da aplicação escolhida; e o círculo 4 e 5, são as configurações de cada aplicação, sendo o primeiro, parâmetros para a simulação; e o segundo, parâmetros para a hierarquia.

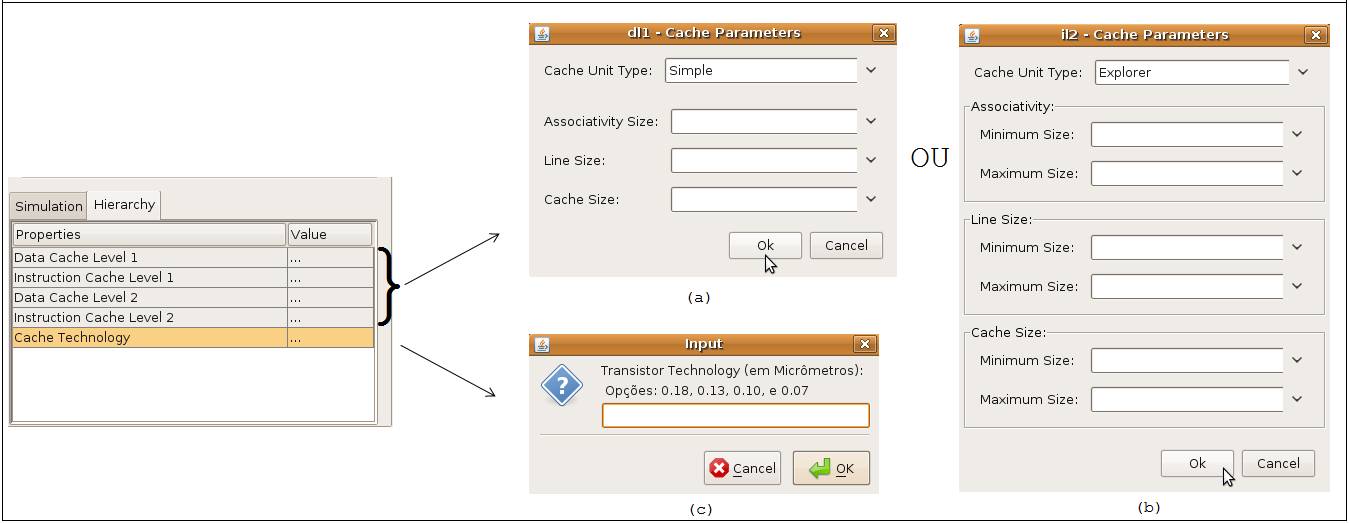


**Figura 16:** Visão Geral da interface do ambiente.

Inicialmente, o usuário criará um projeto na interface de forma simples, apenas tendo que fornecer um nome e o local de uma pasta no sistema. Em cada projeto, podem-se criar várias aplicações. Na criação das aplicações, existem dois conjuntos de configurações a se fazer. Primeiramente, configurações da simulação (figura 17) como a escolha da hierarquia do sistema, as definições da aplicação que irá simular e o tipo de simulação a ser feita. E depois, configurações relativas à hierarquia escolhida (figura 18), com definições dos aspectos de cada cache.

**Figura 17:** Configurando a simulação do ambiente.

Na configuração, escolhemos a hierarquia, que como foi dito anteriormente, pode ser de três tipos. Vale salientar que as outras duas hierarquias mostradas na interface estão incompletas e serão mencionadas nos trabalhos futuros. Após a escolha da hierarquia, temos a escolha do benchmark com seus parâmetros de entrada e saída, como exemplo da figura, escolhe-se o *quick sort small*, e por fim, escolhe-se o tipo de simulação a ser feita. Os tipos de simulação serão explicados mais adiante ainda nesse capítulo. Na figura, escolhe-se a simulação exaustiva.



**Figura 18:** Configurando hierarquia de memória (a) e (b); e a tecnologia de transistor (c).

Após o término dessa parte da configuração, inicia-se a segunda parte, que consiste na configuração da hierarquia escolhida. No caso, iremos configurar a hierarquia 1, com dois níveis totalmente separados. Nessa parte da configuração, serão definidos os parâmetros das memórias caches e a tecnologia de transistor.

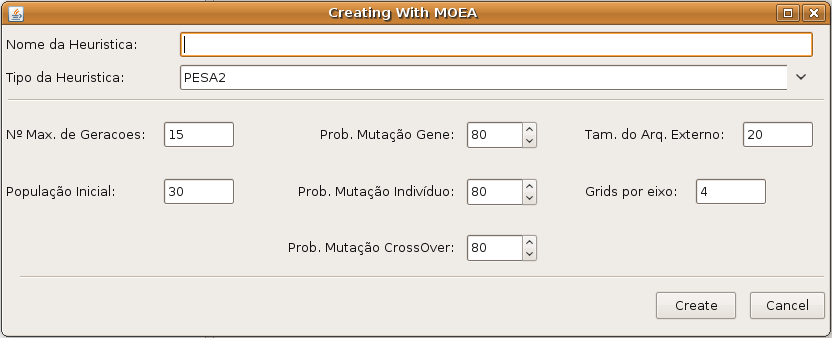
As memórias caches podem ser configuradas de dois tipos: *Simple* (Figura 18.a) e *Explorer* (Figura 18.b). Em cada cache, é configurado o tamanho da cache, o tamanho de linha e a associatividade. A diferença entre os tipos *Simple* e *Explorer* é que o primeiro define uma configuração exata da cache, por exemplo, define-se que a cache de dados do primeiro nível terá o tamanho de 2048 Kbytes com tamanho de linha de 32 bytes e com associatividade igual a 1, enquanto que no segundo tipo, é definido um *range* para os mesmos parâmetros, como por exemplo, a cache de dados do primeiro nível vai ter valores de 2048 até 8192 Kbytes com um tamanho de linha de 16 até 64 bytes e uma associatividade de 1 até 4. Vale salientar que todos os parâmetros variam multiplicando-se por dois. E por fim, define-se a tecnologia de transistor (Figura 18.c), que pode ter somente alguns valores específicos como: 0.18, 0.13, 0.10 e 0.07**.** Todos estão na unidade de micrômetros. Após isso, a aplicação já está devidamente configurada.

O próximo passo é realizar a simulação. Para simular, temos a opção de simular apenas uma aplicação ou todas as aplicações do mesmo projeto. Essa é uma característica importante do ambiente, pois existem algumas configurações em que se deseja simular todo o espaço de uma determinada hierarquia, o que pode demandar muito tempo de simulação, e com essa característica permite ao projetista que ele crie todas as suas aplicações e deixe elas simulando até seu término. Após a completa simulação, o projetista poderá ver seus resultados em gráficos, além de poder fazer uso de algumas funcionalidades que serão descritas no próximo capítulo.

### Tipos de Simulação

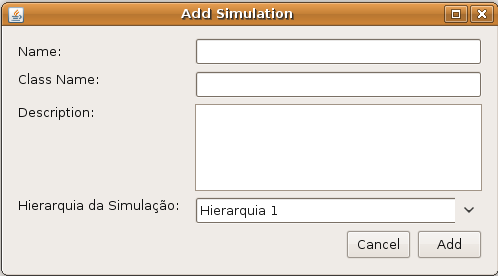
Para cada aplicação criada, existe um tipo de simulação associada, em que a partir dela serão definidos os rumos de como as configurações são escolhidas, sendo posteriormente simuladas.

No ambiente proposto, é possível selecionar dois tipos de simulação. Primeiramente, temos a exaustiva, que consiste em fazer a simulação de todo o espaço de exploração definido pelo projetista no ambiente gráfico. Na verdade, alguns pontos não são simulados por não satisfazer critérios de configuração individuais de uma memória cache ou critérios de configuração da hierarquia.

Outro tipo de simulação possível consiste do uso de heurísticas, onde trata-se de um tipo de simulação que segue alguns critérios para determinar a próxima configuração a ser simulada. Mais precisamente, fez-se uso de MOEAs, que são Algoritmos Evolucionários Multi-Objetivos. Foram selecionados os algoritmos evolucionários NSGA-II, SPEA2, PAES, PESA-II e o NPGA. Para cada algoritmo evolucionário, é possível criar diferentes configurações de simuladores de forma automática mudando alguns critérios na interface.

**Figura 18:** Adicionando uma simulação do tipo MOEA.

Na figura 18, temos o exemplo da tela do pesa-II, com seus campos para preenchimento específicos, apenas para ilustrar como se encontra essa tela para adicionar esse tipo de simulação. Os algoritmos serão descritos no tópico a seguir.

Ainda existe outra forma de adicionar heurísticas ao ambiente, que consiste de dizer o caminho do arquivo onde está o código da heurística. Esse método foi utilizado para adicionar o código referente à simulação exaustiva. Porém, esse método está totalmente manual, onde o projetista teria que saber onde encaixar a estrutura do seu algoritmo, juntamente com os trechos necessários para comunicação com o ambiente. Além de mostrar o caminho do algoritmo, é necessário preencher informações relativas ao algoritmo como para que tipo de hierarquia o algoritmo seja aplicável, como pode ser visto na figura 19.

**Figura 19:** Método manual para adicionar novos tipos de simulações.

O método manual apresentado não consiste de uma solução útil para um leigo no ambiente, pois teria que ter conhecimento prévio do código em questão, porém facilita bastante aos dominadores do ambiente na fácil integração para testes de novas heurísticas. Enquanto que a forma em que se pode adicionar uma nova simulação de um algoritmo evolucionário multi-objetivo se encontra de forma satisfatória para qualquer projetista fazer uso de qualquer configuração desejada para os algoritmos evolucionários até então suportados.

#### NSGA-II

Um dos primeiros algoritmos evolucionários desenvolvido foi o NSGA [22]. Porém, com o passar do tempo, foi alvo de muitas críticas devido ao grau de complexidade de regiões do algoritmo, pela ausência de elitismo, entre outros. Devido a isso, foi proposta uma nova versão para o mesmo, o NSGA-II [23], que apresentava melhorias, resolvendo as dificuldades encontradas na versão anterior.

A tabela 2 mostra o NSGA-II, de como ocorre todas as fases do algoritmo.

|  |
| --- |
| 1. Gera população aleatória inicial *P*0 de tamanho *N*  2. Ordena *P*0 com base em não-dominância  3. População de tamanho *N* é obtida a partir de *P0* através da aplicação dos operadores de seleção, cruzamento e mutação.  4. **Repetir G - 1** vezes, onde *G* é o número de gerações  5. , onde t é a iteração atual  6. F = ordenar por não-dominância  7. F = (), todos os fronts de  8. **enquanto**  9, atribuir *crowding distance* para cada solução em  10.  11. ordenar com base em não dominância e *crowding distance* (*crowded sdsdsdoperator)*  12.  13. População de tamanho *N* é obtida a partir da aplicação dos operadores de dsdsdsseleção (baseado no *crowded operator*), cruzamento e mutação sobre a sdsdsdpopulação  14. t = t+1 |

**Tabela 2:** Algoritmo do NSGA-II.

O algoritmo do *crowding distance* que aparece na descrição do algoritmo do NSGA-II consiste de uma métrica que seleciona os melhores indivíduos a partir de um conjunto de soluções não-dominadas.

Na interface para a criação de um simulador NSGA-II, é necessário configurar alguns valores para criar um simulador com a configuração personalizada para o projetista. Os campos que precisam ser preenchidos para ser possível simular o NSGA-II são: número máximo de gerações, população inicial, probabilidade de mutação do gene, probabilidade de mutação do indivíduo e probabilidade de mutação de Crossover.

#### SPEA2

O SPEA2 [24] é baseado no conceito de dominância e densidade dos vizinhos para a avaliação do *fitness* de cada ponto. O *fitness* consiste de uma medida de qualidade da solução encontrada. Através da densidade dos vizinhos de uma solução, pode-se observar quais os indivíduos são os mais representativos para o conjunto da solução final. Desta forma, a densidade é um critério de decisão entre os indivíduos com mesmo grau de dominância. O algoritmo faz uso de um arquivo externo para armazenar as melhores soluções encontradas entre as gerações. O algoritmo do SPEA2 está descrito na tabela 3.

|  |
| --- |
| 1: **Entrada :** N(Tamanho da população)  2: (Tamanho do arquivo)  3: *T* (Número máximo de gerações)  4**: Saída :** *A* (Soluções não dominadas)  5 : **Inicialização** :Gere população inicial e cria um arquivo externo vazio .  6 : Seta *t* = 0.  7 : **Atribuição de *fitness*** *:*Calcule o *fitness* de e .  8: **Seleção** : Copie os indivíduos não dominados de e para .  9: Se = então  10: Teste condição de parada.  11: Senão se > então  12: Aplique o truncamento a  13: Senão  14: Complete o arquivo com elementos dominados de e  15: **Condição de Parada**: se *t* >= *T* ou outro critério for satisfeito, atribua *A* = e sdsdsdsdsdsdsdssdsdsdsdspare.  16: **Cruzamento**: Seleção por torneio direto (binário) em . Aplique *crossover* em asdasdssasdasdasd até completar o  *mating pool.*  17: **Diversidade**: Aplique o operador de mutação ao *mating pool* e sete a população sdsdsdsdsdsdsdsdsresultante em . Incremente o contador de geração e volte sdsdsdsdsdsdsdsdspara a linha 7. |

**Tabela 3:** Algoritmo do SPEA2.

Na interface para a criação de um simulador SPEA2, é necessário configurar alguns valores para criar um simulador com a configuração personalizada para o projetista. Os campos que precisam ser preenchidos para ser possível simular o SPEA2 são: número máximo de gerações, população inicial, probabilidade de mutação do gene, probabilidade de mutação do indivíduo, probabilidade de mutação de Crossover e tamanho do arquivo externo.

#### PAES

O PAES foi inicialmente proposto na tentativa de resolver um problema de roteamento de redes. Foi quando Knowles e Corne[25] observaram que uma busca local, em problemas mono-objetivos, apresentava um melhor desempenho do que as abordagens baseadas em população. A partir disso, um algoritmo evolucionário de otimização multi-objetiva com utilização de busca local foi desenvolvido. O algoritmo utilizado pode ser visto na tabela 4.

|  |
| --- |
| 1. Gera solução aleatória *c* e adiciona ao arquivo 2. Gera mutação *m* a partir de *c* e avalia *m* 3. **Se** *c* domina *m* **então** 4. Descarta *m* 5. **Senão se** *m* domina *c* **então** 6. Substitui *c* por *m* e adiciona *m* ao arquivo 7. **Senão se** *m* é dominado por algum membro do arquivo **então** 8. Descarta *m* 9. **Senão** aplique *testeDensidade(c, m,* arquivo) para determinar qual será a nova aasolução e se *m* deverá ser adicionado ao arquivo. |

**Tabela 4:** Algoritmo do PAES.

A função de testeDensidade consiste em encontrar a solução baseando-se na densidade da região para determinar se ela fará parte da próxima iteração.

Na interface para a criação de um simulador PAES, é necessário configurar alguns valores para criar um simulador com a configuração personalizada para o projetista. Os campos que precisam ser preenchidos para ser possível simular o PAES são: número máximo de gerações, probabilidade de mutação do gene, tamanho do arquivo externo e quantidade de grids por eixo.

#### PESA-II

O PESA-II foi um algoritmo evolucionário criado por Knowles[26], sendo baseado no algoritmo PESA[27]. A principal diferença entre os dois algoritmos consiste numa nova estratégia de seleção apresentada no PESA-II, tendo a abordagem do uso de hiper grades. Nesta abordagem, o espaço de busca é dividido em células com intervalos proporcionais a cada dimensão, formando hiper cubos. A análise dessas regiões permite a verificação da densidade das soluções, auxiliando no processo de seleção e atualização do arquivo externo utilizado. O algoritmo do PESA-II é mostrado na tabela 5.

|  |
| --- |
| 1. Inicializa a População Inicial *PI* aleatoriamente com *N* indivíduos. 2. Move os indivíduos não-dominados para o arquivo externo *PE*. 3. **Se** Critério de Parada foi atingido 4. Pára e retorna *PE* como o Pareto. 5. **Senão** 6. Apaga a PI. 7. **Repetir** até serem criados novos *N* indivíduos em *PI*. 8. Selecionar 2 indivíduos de *PE*. 9. Usando-se hiper cubos, cruzar e produzir um único filho. 10. Muta-se o filho criado. 11. Selecionar 1 indivíduo e produzir um novo filho.   **Retornar** para o passo 2. |

**Tabela 5:** Algoritmo do PESA-II.

Na interface para a criação de um simulador PESA-II, é necessário configurar alguns valores para criar um simulador com a configuração personalizada para o projetista. Os campos que precisam ser preenchidos para ser possível simular o PESA-II são: número máximo de gerações, população inicial, probabilidade de mutação do gene, probabilidade de mutação do indivíduo, probabilidade de mutação de Crossover, tamanho do arquivo externo e quantidade de grids por eixo.

#### NPGA

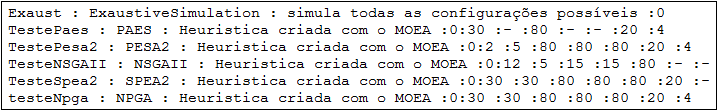
Consiste de um algoritmo proposto por Horn[28] baseado no conceito de não dominância. É um algoritmo caracterizado por não precisar calcular um valor de aptidão para as soluções não dominadas, utilizando um método de seleção por torneio, denominado de Torneio de Pareto. O algoritmo do NPGA pode ser visto na tabela 6.

|  |
| --- |
| **Entrada:** P : População pai  N : Tamanho da população P.  Q : População filha  Qshare : Parâmetro de compartilhamento  tdom : Tamanho do torneio  n : Número de geração atual  nMax : Número máximo de gerações  **Inicialização:**  1: Criar a população inicial P0 e a população filha Q0 vazia.  2: Fazer t = 0, i = 0.  3: Realizar a seleção por torneio com os indivíduos i e i + 1, o ganhador  será o pai p1.  4: i = i + 2.  5: Aplicar a seleção por torneio com os indivíduos i e i + 1, o ganhador  será o pai p2.  6: Fazer o cruzamento de p1 e p2 para gerar os filhos c1 e c2.  7: Aplicar o operador de mutação.  8: Atualizar a população filha Qn = Qn [ {c1, c2}.  9: i = i + 1.  10: **Se** i < N ir para o passo 3.  11: **Se** |Q| = N/2  12: misturar a população P  13: i = 1  14: Voltar para o passo 3.  15: **Se** n < nMax **terminar**.  16: **Senão** fazer Pn+1 = Qn, Qn+1 = vazio, e voltar para o passo 3. |

**Tabela 6:** Algoritmo do NPGA.

Na interface para a criação de um simulador NPGA, é necessário configurar alguns valores para criar um simulador com a configuração personalizada para o projetista. Os campos que precisam ser preenchidos para ser possível simular o NPGA são: número máximo de gerações, população inicial, probabilidade de mutação do gene, probabilidade de mutação do indivíduo, probabilidade de mutação de Crossover, tdom e sigma share.

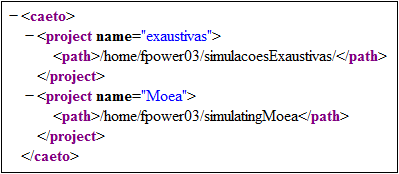
### Arquivos de Registro

O ambiente faz uso de arquivos de texto e XML para controle de seus registros de operação. O primeiro arquivo a ser apresentado consiste do tipo texto para registro de todas as simulações criadas na interface, como pode ser visto na figura 20.

**Figura 20:** Arquivo das simulações criadas na interface.

Nesse arquivo, temos todas as informações pertinentes aos simuladores. Na ordem, temos o nome de exibição, o tipo da simulação, uma descrição, o tipo da hierarquia que se aplica, e as demais são parâmetros personalizados pelo projetista passados através da interface.

Um arquivo bastante simples é gerado para registro do caminho da pasta de cada projeto. Na figura 21, temos o registro de dois projetos para ilustrar como é esse arquivo.



**Figura 21:** Arquivo com caminho da pasta de cada projeto.

O ambiente também gera um arquivo XML para cada projeto criado na interface gráfica, contendo informações relativas a configurações de simulação e de hierarquia de cada aplicação do projeto. A estrutura desse arquivo pode ser vista no anexo 3 (três).

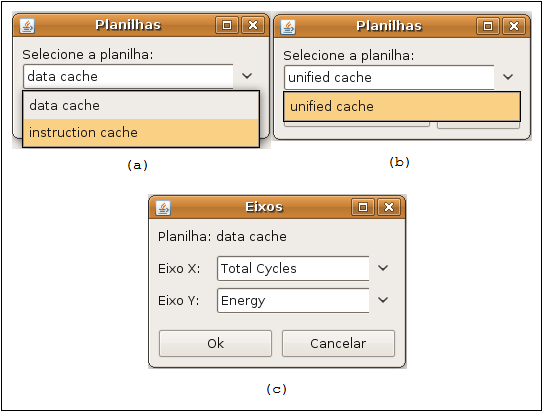
Além disso, quando se simula alguma aplicação, é gerado um arquivo XML que contém os resultados de todos os pontos que foram simulados para a aplicação, onde cada resultado de um ponto de simulação possui informações da configuração de cada cache que foi simulada, assim como os resultados propriamente ditos. O anexo 4 (quatro) mostra como é esse arquivo. Foi necessário citar a existência desse último arquivo, porém os resultados serão discutidos no próximo capítulo.

# Resultados

Neste capítulo será descrito tudo que a ferramenta tem para prover um melhor ambiente de simulação e análise para o projetista.

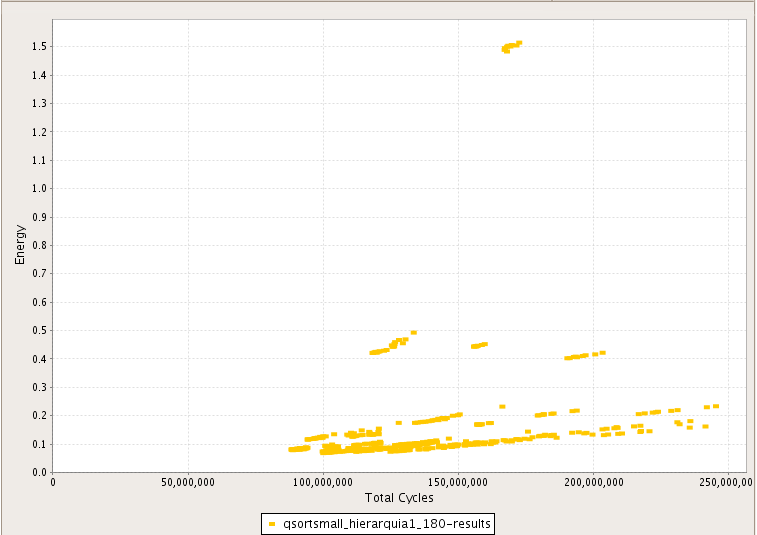
Primeira característica importante observada consiste na questão de possuir diversas variáveis para simulação de configurações diferentes para uma aplicação escolhida. Essas variáveis são a hierarquia, tipo da simulação, configuração de cada cache (Tamanho de cache, Tamanho de linha, associatividade), tecnologia de transistor. Isso dá ao projetista o poder de mexer em vários parâmetros para analisar a melhor configuração, visando uma função de custo de sua aplicação. Isso pode ser visto na descrição da ferramenta no capítulo anterior.

Um ponto importante consiste em um valor específico das variáveis de simulação, que é o uso de heurísticas. Sem elas, o projetista tem de simular todos os possíveis pontos até encontrar o seu resultado ótimo de configuração, mas com o uso delas, pode-se reduzir bastante o espaço amostral. Na ferramenta, temos a possibilidade de usar heurísticas MOEA. Foram incorporadas cinco diferentes técnicas, dando uma maior diversidade de escolha ao projetista. Como por exemplo, utilizando as mesmas configurações, uma simulação exaustiva com o segundo nível separado simula 458 configurações, enquanto que no uso de um MOEA, mais especificamente o PESA2, foi alcançado bons resultados próximos do pareto ótimo com 50 simulações, representando o uso de aproximadamente 11% do espaço amostral da exaustiva.

Porém, não adianta poder simular todas essas diferentes configurações sem ter uma forma de analisá-los, pois o projetista só teria valores de diversas configurações. Para isso, o ambiente permite a visualização de gráficos, permitindo mais uma forma de análise. Para realizar essa ação, o projetista terá que abrir um resultado de simulação através da interface, onde o projetista escolhe o resultado que ele gostaria de ver. Com o arquivo escolhido, é necessário selecionar que porção da cache ele deseja ver os resultados. Caso seja uma hierarquia com caches separadas, tem-se que escolher entre dados ou instruções (Figura 22(a)); caso seja uma hierarquia com caches unificadas, só tem uma opção, que é o resultado dos dados unificados (Figura 22(b)). Definido isso, é preciso escolher quais serão os eixos presentes no seu gráfico como mostrado na figura 22 (c). A ferramenta dá opção de escolha de confrontação de vários parâmetros, desde parâmetros que foram passados como entrada como tamanho da cache; resultados que foram obtidos através das ferramentas como energia dinâmica e estática, número de *misses* e *hits,* ou então, dados que foram calculados como energia e ciclos consumidos. Finalmente, depois dessas passagens de parâmetros, o gráfico aparece para o usuário como mostrado na figura 23.

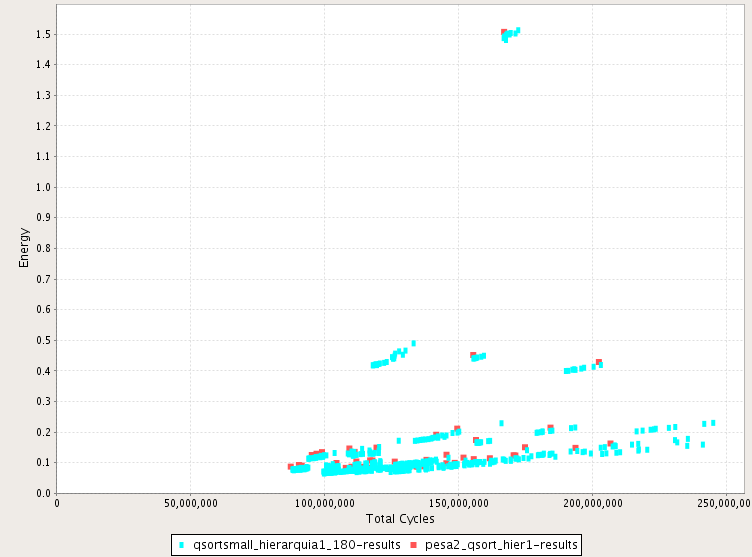
**Figura 22:** Telas para configuração de parâmetros para gerar gráfico personalizado do resultado. Escolha da cache para se ver o resultado. Em **(a)**, caso seja uma hierarquia com caches separadas e em **(b)**, caso seja uma hierarquia com caches unificadas. Em **(c)**, temos a definição dos eixos do gráfico.

Na figura 23, temos o resultado exaustivo do quick sort small para a hierarquia com dois níveis separados. Para as caches do primeiro nível variando o tamanho da cache entre 2048 bytes até 8192 byes, associatividade entre 1 e 4, tamanho de linha entre 16 bytes e 64 bytes; as caches do segundo nível variando o tamanho da cache entre 16384 bytes até 65536 bytes, associatividade entre 1 e 4, tamanho de linha entre 16 bytes e 64 bytes. Todas as ordens aumentam multiplicando-se por 2 (dois). Além disso, com uma tecnologia de transistor de 0,18 micrômetros.



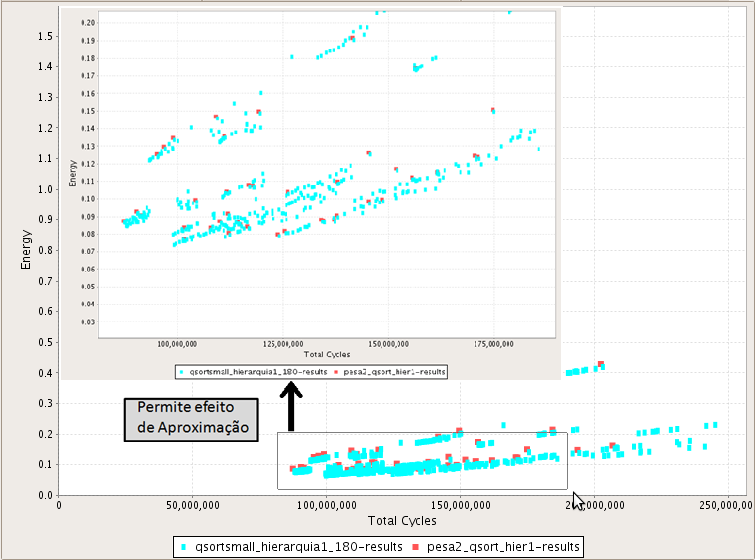
**Figura 23:** Gráfico gerado a partir do ambiente.

A partir da geração dos gráficos, ainda pode-se mesclar os gráficos obtidos, ou seja, juntar dois ou mais gráficos em um único gráfico. Isso dá um grande poder de análise, pois os resultados ficam sobrepostos, sendo mais fácil a visualização. Isso pode ser visto na figura 24.



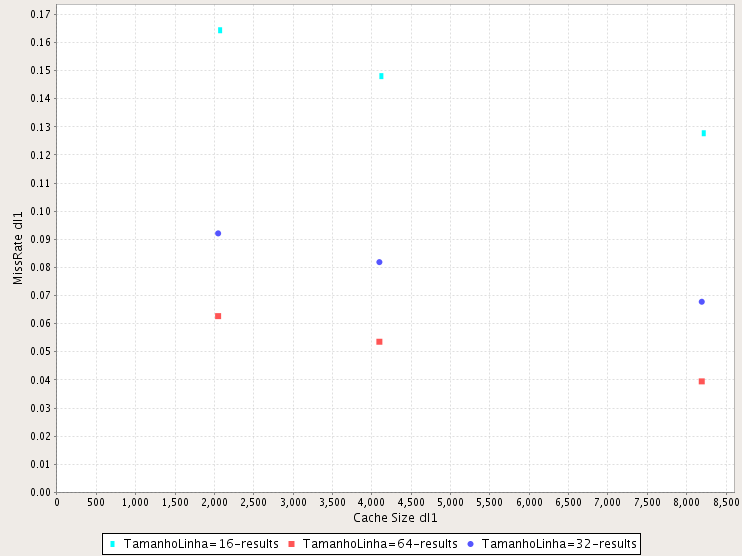
**Figura 24:** Sobreposição de dois gráficos diferentes.

Na figura 24, temos a junção de duas simulações distintas, sendo uma delas a mesma exaustiva definida no exemplo da figura 23 e a outra uma simulação com as mesmas configurações de hierarquia e a mesma aplicação que a exaustiva, porém com o uso da heurística pesa2. Como parâmetros da heurística, foi utilizado um número máximo de gerações de 50, uma população inicial de 10, probabilidade de mutação do gene de 15%, probabilidade de mutação do indivíduo de 15%, probabilidade de mutação de Crossover de 90%, tamanho do arquivo externo de 10 e 5 grids por eixo. A partir da visualização dessa figura, podemos citar outra funcionalidade, pois podemos ver que existem alguns pontos não tão visíveis no gráfico, e a partir disso, temos a possibilidade de aproximação em qualquer parte do gráfico, como mostrado na figura 25, detalhando mais a região escolhida. Esse efeito pode ser realizado quantas vezes forem necessárias.



**Figura 25:** Efeito de aproximação nos gráficos.

O principal objetivo dos gráficos consiste na análise entre energia e total de ciclos consumidos, porém para cada eixo, podem-se selecionar outros tipos de análise nos eixos do gráfico como de medidas da configuração da cache, assim como medidas resultantes das ferramentas como potência dinâmica e estática, tempo de acesso, quantidade de hits, misses. Para exemplificar, utilizaremos o mesmo tipo de gráfico mostrado anteriormente na figura 4 da ferramenta ChARM. É mostrado na figura 26, consistindo num gráfico do tamanho da cache de dados pela taxa de misses do mesmo em diferentes configurações do tamanho de linha da cache.

**Figura 26:** Variação do tipo de gráfico: taxa de *misses* X tamanho da cache.

Outra característica importante consiste no uso de simulações com threads, onde cada aplicação é tratada como uma thread. No caso do ambiente estar sendo simulado numa arquitetura que possui uma unidade de processamento, essa característica não terá muito valor, pois surtirá o efeito como uma simulação feita em sequência, porém o mercado atual possui cada vez mais unidades de processamento nas suas arquiteturas e em processadores como esse, as aplicações de um projeto são simuladas simultaneamente, já que cada aplicação consiste de um processo diferente. Isso permite uma maior dedicação do processamento para a realização das simulações do ambiente. Além disso, o fato de se poder designar diversas aplicações para simular, automatiza o processo de simulação como em um *script*, quando normalmente é necessário dizer para a ferramenta o que simular, logo depois que uma simulação finaliza, o que dependendo do número de aplicações do projeto que se deseja simular, pode se perder muito tempo dando essas diretrizes para o ambiente.

Para simular todas as aplicações de um projeto, no momento em que se seleciona o botão para simular, basta escolher o projeto desejado e na escolha da aplicação, escolher a opção *All*. A partir disso, a ferramenta simula todas as aplicações do projeto.

Além das características apresentadas, o ambiente pode exportar os resultados, no formato XML, encontrados para cada simulação para o formato XLS. Isso é bastante útil, pois nesse formato, é mais fácil de manipular os dados, para gerar novos resultados que ainda não estão presentes na ferramenta.

Com o que foi mostrado nesse capítulo, podemos fazer uma breve comparação do ambiente proposto com o estado da arte apresentado no inicio do trabalho. Para isso, veremos através da tabela 7. Todas as características apresentadas são as mesmas descritas nesse mesmo capítulo, exceto o suporte a multiprocessamento, que consiste de uma característica apresentada pelo Pdesigner.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Ferramenta | Diferentes Arquiteturas para Simulação | Análise Simultânea | Método Heurístico | Consumo de Energia | Ambiente Gráfico | Simulação em *Threads* | Suporte a Multiprocessamento |
| Pdesigner | SIM | - | - | SIM | SIM | - | SIM |
| MAX | SIM | - | SIM | SIM | - | - | - |
| ChARM | SIM | - | - | - | SIM | - | - |
| S. Filho | - | - | SIM | SIM | - | - | - |
| F.Vahid | - | - | SIM | SIM | - | - | - |
| Ambiente Proposto | SIM | SIM | SIM | SIM | SIM | SIM | - |

**Tabela 7:** Comparação do estado da arte com o ambiente proposto.

# Conclusões e Trabalhos Futuros

Foi apresentado nesse trabalho, um ambiente de simulação e análise visando aspectos de consumo de energia e desempenho. O desenvolvimento do ambiente foi realizado de forma modular, garantindo que futuras funcionalidades sejam adicionadas de uma maneira mais simplificada.

Um estudo do estado da arte foi feito para que fosse feito uma validação das características essências e dos diferenciais obtidos com o ambiente proposto. Vimos que o ambiente conseguiu reunir diversas características presentes nos outros estudos, apresentando com uma visão *high-level* na exploração das hierarquias de memória cache.

Portanto, foi desenvolvido um ambiente prático e eficiente que vai facilitar e acelerar a pesquisa do projetista, simulando e analisando várias configurações com diferentes tipos de simulações. Além disso, podendo-se adicionar novas variações para explorá-las no ambiente.

Como trabalhos futuros, pretende-se adicionar novas variantes para o ambiente, como novas hierarquias, podendo-se estendê-las ao terceiro nível de cache,assim como novos tipos de heurística, utilizando técnicas bastante utilizadas no estado da arte como o uso de PSO, *Particle Swarm Optimization* ou o uso do *Ant Colony*. Deseja-se que essas adições ao sistema sejam cada vez mais automáticas e simples para o projetista. Além disso, melhorar a forma com que os resultados são mostrados, adicionando informações a cada ponto do gráfico, dando novas formas de análise como novos tipos de gráficos e novos critérios para serem analisados.

# Referências

[1] S. SEGARS, **Low power design techniques for microprocessors**, Int. Solid-State Circuits Conf. Tutorial, 2001.

[2] A. MALIK, B. MOYER and D. CERMAK, **A Low Power Unified Cache Architecture Providing Power and Performance Flexibility**, Int Symp. On Low Power Electronics and Design, June 2000, pp.241-243.

[3] **Phd Thesis**: Henric Mussoul.

[4] E. MACII, L. BENINI, M. PONCINO, **Energy-Aware Design of Embedded Memories: A Survey of Technologies, Architectures and Optimization Techniques**, ACM Transactions on EmbeddedComputing Systems; Vol. 2, No. 1, pp. 5-32, February 2003.

[5] P. Viana, E. Barros, S. Rigo, R. Azevedo, G. Araújo, **Exploring Memory Hierarchy with ArchC**, SBAC-PAD 2003, pp.2-10.

[6] A. SOUZA, M. ALMEIDA, W. AZEVEDO, C. ARAÚJO, E. BARROS, F. ROLIM, A. SILVA, **Pdesigner – Conjunto de ferramentas para modelagem de MPSoC’s**, In: WSCAD-CTIC 2007, 2007, Gramado - RS. SBAC-PAD 2007, 2007.

[7] [C. A. PRETE , M. GRAZIANO , F. LAZZARINI, **The ChARM Tool for Tuning Embedded Systems**, IEEE Micro, v.17 n.4, p.67-76, July 1997](http://portal.acm.org/citation.cfm?id=624117&CFID=198100&CFTOKEN=22510665)

[8] T.S. Rajesh Kumar, C.P. Ravikumar, R. Govindarajan, **MAX: A Multi Objective Memory Architecture eXploration Framework for Embedded Systems-on-Chip,** vlsid, pp.527-533, 20th International Conference on VLSI Design held jointly with 6th International Conference on Embedded Systems (VLSID'07), 2007.

[9] A. G. SILVA-FILHO, **An Optimization Mechanism Intended for Two-Level Cache Hierarchy to Improve Energy and Performance using the NSGAII Algorithm,** In: SBAC-PAD, 2008, Campo Grande - MS. 20th International Symposium on Computer Architecture and High Performance Computing. Danvers, MA : IEEE Computer Society - Conference Publishing Services, 2008. v. v1. p. 19-2.

[10] A. Gordon-Ross, F. Vahid, and N. Dutt, **Fast configurable-cache tuning with a unified second-level cache,** In ISLPED '05: Proceedings of the 2005 international symposium on Low power electronics and design, pages 323-326, New York, NY,USA, 2005. ACM Press.

[11] Site da Sun. Disponível em <http://java.sun.com/> Último acesso em 02/12/2010.

[12] Site do Eclipse. Disponível em <www.**eclipse**.org/> Último acesso em 02/12/2010.

[13] Swing Application Framework. Em <https://appframework.dev.java.net/> Último acesso em 02/12/2010.

[14] Suporte ao Swing Application Framework. Disponível em <http://java.sun.com/developer/technicalArticles/javase/swingappfr/> Último acesso em 02/12/2010.

[15] JDOM. Disponível em <http://www.jdom.org/> Último acesso em 02/12/2010.

[16] Suporte ao JDOM. Disponível em <http://www.javaworld.com/javaworld/jw-05-2000/jw-0518-jdom.html> Último acesso em 02/12/2010.

[17] JFreeChart. Disponível em <http://www.jfree.org/jfreechart/> Último acesso em 02/12/2010.

[18] Ferramenta de Simulação SimpleScalar. Disponível em <http://www.simplescalar.com/> Último acesso em 02/12/2010.

[19] Ferramenta de Simulação eCACTI. Disponível em <http://www.ics.uci.edu/~maheshmn/eCACTI/main.htm> Último acesso em 02/12/2010.

[20] A. Guilhermino, **Uma Metodologia Para Exploração De Configurações De**

**Memória Cache Visando Redução Do Consumo De Energia,** 2006. 157 f. Tese

de Doutorado em Ciências da Computação. Universidade Federal de

Pernambuco. Recife.

[21] R.A. Sugumar, S.G. Abraham, **Efficient simulation of multiple cache**

**configurations using binomial trees**, Technical Report CSE-TR-111-91, CSE

Division,Univ. of Michigan, 1991.

[22] N. SRINIVAS, K. DEB, **Multiobjective Function Otimization Using Nondominated Sorting Genetic Algorithms**, Evol. Comput., vol 2, no 3, p. 221-248, 1995.

[23] K. DEB, S. AGRAWAL, T. MEYARIVAN, **A Fast Elitist Nondominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II, Parallel Problem Solving from Nature**, 2000, p. 849-858.

[24] E. ZITZLER, M. LAUMMANS, L. THIELE, **SPEA2: Improving the Strength of Pareto Evolutionary Algorithm for Multiobjective Optimization**, Technical Report 103, Computer Engineering and Networks Laboratory(TIK), Swiss Federal Institute of Technology (ETH), Switzerland, 2001.

[25] J. KNOWLES, D. CORNE, **Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy**, Evolutionary Computation 8(2): 149-172.

[26] D.W. CORNE, N.R. JERRAM, J.D. KNOWLES, M.J. OATES, **PESA-II: Region-based Selection in Evolutionary Multiobjective Optimization**. Proceedings of the Genetic and Evolutionary Computation Conference, São Francisco, 2001.

[27] D.W. CORNE, D.J. KNOWLES, M.J. OATES, **The Pareto Envelope-based selection algorithm for multiobjective optimization**. Proceedings of sixth international conference on parallel problem solving from Nature, p. 18-20, 2000.

[28] J. Horn, N. Nafpliotis, D.E. Goldberg, **A Niched Pareto Genetic Algorithm for Multiobjective Optimization**, in ‘Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence’, Vol. 1, IEEE Service Center, Piscataway, New Jersey, pp. 82–87.

# ANEXOS

## Anexo 1: Arquivo de Resultado do SimpleScalar.

sim-cache: SimpleScalar/PISA Tool Set version 3.0 of August, 2003.

Copyright (c) 1994-2003 by Todd M. Austin, Ph.D. and SimpleScalar, LLC.

All Rights Reserved. This version of SimpleScalar is licensed for academic

non-commercial use. No portion of this work may be used by any commercial

entity, or for any commercial purpose, without the prior written permission

of SimpleScalar, LLC (info@simplescalar.com).

sim: command line: /home/fpower03/Simplescalar/simplesim-3.0/sim-cache -redir:sim /home/fpower03/benchmarks/qsort/output.txt -cache:il1 il1:32:64:4:r -cache:dl1 dl1:32:64:4:r -cache:il2 il2:256:64:4:r -cache:dl2 dl2:256:64:4:r /home/fpower03/benchmarks/qsort/qsort\_small /home/fpower03/benchmarks/qsort/input\_small.dat > caeto\_output.txt

sim: simulation started @ Mon Dec 6 16:14:28 2010, options follow:

sim-cache: This simulator implements a functional cache simulator. Cache

statistics are generated for a user-selected cache and TLB configuration,

which may include up to two levels of instruction and data cache (with any

levels unified), and one level of instruction and data TLBs. No timing

information is generated.

# -config # load configuration from a file

# -dumpconfig # dump configuration to a file

# -h false # print help message

# -v false # verbose operation

# -d false # enable debug message

# -i false # start in Dlite debugger

-seed 1 # random number generator seed (0 for timer seed)

# -q false # initialize and terminate immediately

# -chkpt <null> # restore EIO trace execution from <fname>

# -redir:sim /home/fpower03/benchmarks/qsort/output.txt # redirect simulator output to file (non-interactive only)

# -redir:prog <null> # redirect simulated program output to file

-nice 0 # simulator scheduling priority

-max:inst 0 # maximum number of inst's to execute

-cache:dl1 dl1:32:64:4:r # l1 data cache config, i.e., {<config>|none}

-cache:dl2 dl2:256:64:4:r # l2 data cache config, i.e., {<config>|none}

-cache:il1 il1:32:64:4:r # l1 inst cache config, i.e., {<config>|dl1|dl2|none}

-cache:il2 il2:256:64:4:r # l2 instruction cache config, i.e., {<config>|dl2|none}

-tlb:itlb itlb:16:4096:4:l # instruction TLB config, i.e., {<config>|none}

-tlb:dtlb dtlb:32:4096:4:l # data TLB config, i.e., {<config>|none}

-flush false # flush caches on system calls

-cache:icompress false # convert 64-bit inst addresses to 32-bit inst equivalents

# -pcstat <null> # profile stat(s) against text addr's (mult uses ok)

The cache config parameter <config> has the following format:

<name>:<nsets>:<bsize>:<assoc>:<repl>

<name> - name of the cache being defined

<nsets> - number of sets in the cache

<bsize> - block size of the cache

<assoc> - associativity of the cache

<repl> - block replacement strategy, 'l'-LRU, 'f'-FIFO, 'r'-random

Examples: -cache:dl1 dl1:4096:32:1:l

-dtlb dtlb:128:4096:32:r

Cache levels can be unified by pointing a level of the instruction cache

hierarchy at the data cache hiearchy using the "dl1" and "dl2" cache

configuration arguments. Most sensible combinations are supported, e.g.,

A unified l2 cache (il2 is pointed at dl2):

-cache:il1 il1:128:64:1:l -cache:il2 dl2

-cache:dl1 dl1:256:32:1:l -cache:dl2 ul2:1024:64:2:l

Or, a fully unified cache hierarchy (il1 pointed at dl1):

-cache:il1 dl1

-cache:dl1 ul1:256:32:1:l -cache:dl2 ul2:1024:64:2:l

sim: \*\* starting functional simulation w/ caches \*\*

sim: \*\* simulation statistics \*\*

sim\_num\_insn 44390974 # total number of instructions executed

sim\_num\_refs 23042271 # total number of loads and stores executed

sim\_elapsed\_time 10 # total simulation time in seconds

sim\_inst\_rate 4439097.4000 # simulation speed (in insts/sec)

il1.accesses 44390974 # total number of accesses

il1.hits 44150405 # total number of hits

il1.misses 240569 # total number of misses

il1.replacements 240441 # total number of replacements

il1.writebacks 0 # total number of writebacks

il1.invalidations 0 # total number of invalidations

il1.miss\_rate 0.0054 # miss rate (i.e., misses/ref)

il1.repl\_rate 0.0054 # replacement rate (i.e., repls/ref)

il1.wb\_rate 0.0000 # writeback rate (i.e., wrbks/ref)

il1.inv\_rate 0.0000 # invalidation rate (i.e., invs/ref)

il2.accesses 240569 # total number of accesses

il2.hits 240081 # total number of hits

il2.misses 488 # total number of misses

il2.replacements 74 # total number of replacements

il2.writebacks 0 # total number of writebacks

il2.invalidations 0 # total number of invalidations

il2.miss\_rate 0.0020 # miss rate (i.e., misses/ref)

il2.repl\_rate 0.0003 # replacement rate (i.e., repls/ref)

il2.wb\_rate 0.0000 # writeback rate (i.e., wrbks/ref)

il2.inv\_rate 0.0000 # invalidation rate (i.e., invs/ref)

dl1.accesses 23149357 # total number of accesses

dl1.hits 22406353 # total number of hits

dl1.misses 743004 # total number of misses

dl1.replacements 742876 # total number of replacements

dl1.writebacks 402149 # total number of writebacks

dl1.invalidations 0 # total number of invalidations

dl1.miss\_rate 0.0321 # miss rate (i.e., misses/ref)

dl1.repl\_rate 0.0321 # replacement rate (i.e., repls/ref)

dl1.wb\_rate 0.0174 # writeback rate (i.e., wrbks/ref)

dl1.inv\_rate 0.0000 # invalidation rate (i.e., invs/ref)

dl2.accesses 1145153 # total number of accesses

dl2.hits 650253 # total number of hits

dl2.misses 494900 # total number of misses

dl2.replacements 493876 # total number of replacements

dl2.writebacks 271059 # total number of writebacks

dl2.invalidations 0 # total number of invalidations

dl2.miss\_rate 0.4322 # miss rate (i.e., misses/ref)

dl2.repl\_rate 0.4313 # replacement rate (i.e., repls/ref)

dl2.wb\_rate 0.2367 # writeback rate (i.e., wrbks/ref)

dl2.inv\_rate 0.0000 # invalidation rate (i.e., invs/ref)

itlb.accesses 44390974 # total number of accesses

itlb.hits 44390957 # total number of hits

itlb.misses 17 # total number of misses

itlb.replacements 0 # total number of replacements

itlb.writebacks 0 # total number of writebacks

itlb.invalidations 0 # total number of invalidations

itlb.miss\_rate 0.0000 # miss rate (i.e., misses/ref)

itlb.repl\_rate 0.0000 # replacement rate (i.e., repls/ref)

itlb.wb\_rate 0.0000 # writeback rate (i.e., wrbks/ref)

itlb.inv\_rate 0.0000 # invalidation rate (i.e., invs/ref)

dtlb.accesses 23149357 # total number of accesses

dtlb.hits 23145144 # total number of hits

dtlb.misses 4213 # total number of misses

dtlb.replacements 4085 # total number of replacements

dtlb.writebacks 2380 # total number of writebacks

dtlb.invalidations 0 # total number of invalidations

dtlb.miss\_rate 0.0002 # miss rate (i.e., misses/ref)

dtlb.repl\_rate 0.0002 # replacement rate (i.e., repls/ref)

dtlb.wb\_rate 0.0001 # writeback rate (i.e., wrbks/ref)

dtlb.inv\_rate 0.0000 # invalidation rate (i.e., invs/ref)

ld\_text\_base 0x00400000 # program text (code) segment base

ld\_text\_size 101504 # program text (code) size in bytes

ld\_data\_base 0x10000000 # program initialized data segment base

ld\_data\_size 12288 # program init'ed `.data' and uninit'ed `.bss' size in bytes

ld\_stack\_base 0x7fffc000 # program stack segment base (highest address in stack)

ld\_stack\_size 16384 # program initial stack size

ld\_prog\_entry 0x00400140 # program entry point (initial PC)

ld\_environ\_base 0x7fff8000 # program environment base address address

ld\_target\_big\_endian 0 # target executable endian-ness, non-zero if big endian

mem.page\_count 662 # total number of pages allocated

mem.page\_mem 2648k # total size of memory pages allocated

mem.ptab\_misses 662 # total first level page table misses

mem.ptab\_accesses 224493254 # total page table accesses

mem.ptab\_miss\_rate 0.0000 # first level page table miss rate

## Anexo 2: Arquivo de Resultado do eCACTI.

Command line: /home/fpower03/Desktop/eCACTI/Ecacti 65536 64 4 0.07 1 /tmp/ecacti\_parametroshierarchy14.txt

--------------------------------

---------- eCACTI 1.0 ----------

--------------------------------

Cache Subarray Parameters (C, B, A): (65536 B, 64 B, 4)

Number of Subarrays: 1

Ports (RW, R, W): (1, 0, 0)

Technology: 0.07 um, Vdd: 1.0 V

#Cache configurations expressed in (Ndwl, Nspd, Ndbl, Ntwl, Ntspd, Ntbl) format

Configuration: (4, 1, 1, 2, 1, 1)

Power Stats:

------------

Read hit power: 393.2234 mw

Write hit power: 312.3276 mw

Read miss power: 699.9266 mw

Write miss power: 324.4357 mw

Timing Stats:

-------------

Access time: 0.82914 ns

Cycle time (wave pipelined): 0.319536 ns

Area Stats:

-----------

Aspect ratio (height/width): 1.286588

Total area one subarray : 0.010387 cm^2

Power Components (dyn, lkg mW):

-------------------------------

#NOTE: The power values are for a cache read hit

Total Power all Banks : 117.9, 267

Data Array Split:

-----------------

decode : 1.789, 7.843

wordline : 0.6516, 0.008201

bitline : 95.77, 238.9

sense\_amp : 2.62, 5.571

read\_control : 3.263, 0.1894

write\_control : 0, 0.4167

Tag Array Split:

----------------

decode : 1.591, 0.3581

wordline : 0.02911, 0.002275

bitline : 2.811, 6.999

sense\_amp : 0.5919, 2.317

read\_control : 0.1538, 0.008931

write\_control : 0, 0.01194

tag comparison : 0.3233, 0.07462

mux driver : 2.309, 0.145

sel inverter : 0.006427, 0

data output driver : 6.015, 4.156

Time Components:

----------------

data side (with Output driver) (ns): 0.617519

tag side (with Output driver) (ns): 0.82914

subbank address routing delay (ns): 0

decode\_data (ns): 0.221416

wordline and bitline data (ns) : 0.253958

sense\_amp\_data (ns): 0.072625

decode\_tag (ns): 0.319536

wordline and bitline tag (ns): 0.113704

sense\_amp\_tag (ns): 0.044625

compare time (ns): 0.0862182

mux driver (ns): 0.171884

sel inverter (ns): 0.023652

data output driver (ns): 0.0695202

total\_out\_driver (ns): 0

total data path (without output driver) (ns): 0.547999

total tag path is set assoc (ns): 0.759619

Area Components:

----------------

Aspect Ratio Total height/width: 1.286588

Data array (cm^2): 0.009779

Data predecode (cm^2): 0.000013

Data colmux predecode (cm^2): 0.000003

Data colmux post decode (cm^2): 0.000000

Data write signal (cm^2): 0.000093

Tag array (cm^2): 0.000385

Tag predecode (cm^2): 0.000013

Tag colmux predecode (cm^2): 0.000003

Tag colmux post decode (cm^2): 0.000000

Tag output driver decode (cm^2): 0.000005

Tag output driver enable signals (cm^2): 0.000093

Percentage of data ramcells alone of total area: 79.980276 %

Percentage of tag ramcells alone of total area: 2.499384 %

Percentage of total control/routing alone of total area: 17.520341 %

Subbank Efficiency : 82.479659

Total Efficiency : 84.066579

Total area One Subbank (cm^2): 0.010387

Total area subbanked (cm^2): 0.010191

## Anexo 3: Exemplo de arquivo XML gerado de um projeto.

O anexo foi simplificado, minimizando o conteúdo da segunda aplicação em diante, deixando apenas o conteúdo referente à primeira aplicação. O conteúdo das demais consiste apenas de configurações específicas seguindo a mesma estrutura da aplicação mostrada.

<?xml version="1.0" encoding="UTF-8"?>

<project name="Moea" path="/home/fpower03/simulatingMoea">

- <application name="nsgaII\_qsort\_hier1">

<benchmark>/home/fpower03/benchmarks/qsort/qsort\_small</benchmark>

<inputFile>/home/fpower03/benchmarks/qsort/input\_small.dat</inputFile>

<outputFile>/home/fpower03/benchmarks/qsort/output.txt</outputFile>

<simulationType>nsgaii\_demo</simulationType>

<hierarchy name="hierarchy1">

<tensaoAlimentacao />

<transistorTechnology>0.18</transistorTechnology>

<cacheUnit type="CacheUnitExplorer" name="dl1">

<cacheSizeMinimum>2048</cacheSizeMinimum>

<cacheSizeMaximum>8192</cacheSizeMaximum>

<lineSizeMinimum>16</lineSizeMinimum>

<lineSizeMaximum>64</lineSizeMaximum>

<associativityMinimum>1</associativityMinimum>

<associativityMaximum>4</associativityMaximum>

</cacheUnit>

<cacheUnit type="CacheUnitExplorer" name="dl2">

<cacheSizeMinimum>16384</cacheSizeMinimum>

<cacheSizeMaximum>65536</cacheSizeMaximum>

<lineSizeMinimum>16</lineSizeMinimum>

<lineSizeMaximum>64</lineSizeMaximum>

<associativityMinimum>1</associativityMinimum>

<associativityMaximum>4</associativityMaximum>

</cacheUnit>

<cacheUnit type="CacheUnitExplorer" name="il1">

<cacheSizeMinimum>2048</cacheSizeMinimum>

<cacheSizeMaximum>8192</cacheSizeMaximum>

<lineSizeMinimum>16</lineSizeMinimum>

<lineSizeMaximum>64</lineSizeMaximum>

<associativityMinimum>1</associativityMinimum>

<associativityMaximum>4</associativityMaximum>

</cacheUnit>

<cacheUnit type="CacheUnitExplorer" name="il2">

<cacheSizeMinimum>16384</cacheSizeMinimum>

<cacheSizeMaximum>65536</cacheSizeMaximum>

<lineSizeMinimum>16</lineSizeMinimum>

<lineSizeMaximum>64</lineSizeMaximum>

<associativityMinimum>1</associativityMinimum>

<associativityMaximum>4</associativityMaximum>

</cacheUnit>

</hierarchy>

</application>

+ <application name="paes\_qsort\_hier1"> </application>

+ <application name="npga\_qsort\_hier1"> </application>

+ <application name="spea2\_qsort\_hier1"> </application>

+ <application name="pesa2\_qsort\_hier1"> </application>

</project>

## Anexo 4: Exemplo de arquivo XML gerado para resultado da simulação de uma aplicação.

O anexo foi simplificado, retirando todos os outros resultados obtidos para a simulação da aplicação em questão. Os resultados removidos consistem de resultados específicos a cada configuração de cache com a mesma estrutura do resultado mostrado.

<?xml version="1.0" encoding="UTF-8"?>

<simulationResults project="Moea" application="npga\_qsort\_hier1">

- <results>

<hierarchy name="hierarchy1">

<tensaoAlimentacao />

<transistorTechnology>0.18</transistorTechnology>

<cacheUnit type="CacheUnitSimple" name="dl1">

<cacheSize>8192</cacheSize>

<lineSize>32</lineSize>

<associativity>1</associativity>

</cacheUnit>

<cacheUnit type="CacheUnitSimple" name="dl2">

<cacheSize>32768</cacheSize>

<lineSize>64</lineSize>

<associativity>1</associativity>

</cacheUnit>

<cacheUnit type="CacheUnitSimple" name="il1">

<cacheSize>4096</cacheSize>

<lineSize>16</lineSize>

<associativity>1</associativity>

</cacheUnit>

<cacheUnit type="CacheUnitSimple" name="il2">

<cacheSize>65536</cacheSize>

<lineSize>64</lineSize>

<associativity>4</associativity>

</cacheUnit>

</hierarchy>

<result cacheType="data">

<totalEnergy>0.10449875258554454</totalEnergy>

<totalCycles>1.28807952E8</totalCycles>

</result>

<result cacheType="instruction">

<totalEnergy>0.026280595513240818</totalEnergy>

<totalCycles>8.0747112E7</totalCycles>

</result>

<cache name="dl1">

<hits>2.1580816E7</hits>

<accessNumber>2.3149356E7</accessNumber>

<misses>1568542.0</misses>

<missRate>0.06780000030994415</missRate>

<acessTime>1.8211899995803833</acessTime>

<pdyn>47.38999938964844</pdyn>

<pleak>0.014340000227093697</pleak>

<edyn>0.0863061929685482</edyn>

<eleak>2.6115865007563465E-5</eleak>

</cache>

<cache name="dl2">

<hits>1619843.0</hits>

<accessNumber>2400357.0</accessNumber>

<misses>780514.0</misses>

<missRate>0.32519999146461487</missRate>

<acessTime>2.096479892730713</acessTime>

<pdyn>93.25</pdyn>

<pleak>0.053619999438524246</pleak>

<edyn>0.19549674999713898</edyn>

<eleak>1.124132506710982E-4</eleak>

</cache>

<cache name="il1">

<hits>3.8338868E7</hits>

<accessNumber>4.4390976E7</accessNumber>

<misses>6052105.0</misses>

<missRate>0.1362999975681305</missRate>

<acessTime>1.833340048789978</acessTime>

<pdyn>31.84000015258789</pdyn>

<pleak>0.008627999573946</pleak>

<edyn>0.05837354743321839</edyn>

<eleak>1.5818057159858068E-5</eleak>

</cache>

<cache name="il2">

<hits>6051605.0</hits>

<accessNumber>6052105.0</accessNumber>

<misses>500.0</misses>

<missRate>9.999999747378752E-5</missRate>

<acessTime>2.1338300704956055</acessTime>

<pdyn>200.6999969482422</pdyn>

<pleak>0.17720000445842743</pleak>

<edyn>0.42825968863653546</edyn>

<eleak>3.7811469800534784E-4</eleak>

</cache>

</results>

+ <results> </results>

+ <results> </results>

.

.

.

+ <results> </results>

+ <results> </results>

</simulationResults>