UNIVERSIDADE FEDERAL DO PIAUÍ
CAMPUS: SENADOR HELVÍDIO NUNES DE BARROS
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

FRANCISCO ARMANDO AVELINO DE OLIVEIRA

Estudo e Implementação de uma interface gráfica para o simulador sim-cache da ferramenta SimpleScalar

Picos
2013

ANCISCO ARMANDO AVELINO DE OLIVEIRA

Estudo e Implementação de uma interface gráfica para o simulador sim-sache da ferramenta SimpleScalar

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Sistemas de Informação Campus Senador Helvídio Nunes de Barros da Universidade Federal do Piauí como parte dos requisitos para obtenção do Grau de Bacharelado em Sistemas de Informação, sob a orientação da Msc. Juliana Oliveira de Carvalho.

Picos
2013

Eu, **Francisco Armando Avelino de Oliveira** , abaixo identificado(a) como autor(a), autorizo a biblioteca da Universidade Federal do Piauí a divulgar, gratuitamente, sem ressarcimento de direitos autorais, o texto integral da publicação abaixo discriminada, de minha autoria, em seu site, em formato PDF, para fins de leitura e/ou impressão, a partir da data de hoje.

Picos-PI 23 de setembro de 2013.

Francisco Armando Avelino de Oliveira
Assinatura

FICHA CATALOGRÁFICA
Serviço de Processamento Técnico da Universidade Federal do Piauí
Biblioteca José Albano de Macêdo

O48e Oliveira, Francisco Armando Avelino de.
Estudo e implementação de uma interface gráfica para o simulador sim-cache da ferramenta simpleScalar / Francisco Armando Avelino de Oliveira. – 2013.
CD-ROM : il. ; 4 ¾ pol. (50 p.)

Monografia(Bacharelado em Sistemas de Informação) – Universidade Federal do Piauí. Picos-PI, 2013.
Orientador(A): Profa. Msc. Juliana Oliveira de Carvalho

1. SompleScalar. 2. Simulador. 3. Memória Cache. I. Título.

CDD 004.6

FRANCISCO ARMANDO AVELINO DE OLIVEIRA

Estudo e Implementação de uma interface gráfica para o simulador sim-sache da ferramenta SimpleScalar

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Sistemas de Informação Campus Senador Helvídio Nunes de Barros da Universidade Federal do Piauí como parte dos requisitos para obtenção do Grau de Bacharelado em Sistemas de Informação, sob orientação da MSc. Juliana Oliveira de Carvalho.

Data de Aprovação:
16/09/2013

MSc. Juliana Oliveira de Carvalho _____ UFPI/SHNB
Esp. Dennis Sávio Martins da Silva _____ UFPI/SHNB
MSc. Frank César Lopes Véras _____ UFPI/SHNB

Picos
2013

À minha família.

Primeiramente agradeço a Deus pela minha vida e por tudo. Aos meus pais, pelos seus ensinamentos, aos meus avós, enfim e de um modo geral, a toda minha família pelo apoio ao longo dessa jornada. Aos meus amigos que me ajudaram e apoiaram nas horas mais difíceis enfrentadas na universidade. Aos professores, que proporcionaram momentos de felicidades, superação, conquistas e de bastante conhecimento ao longo desse curso. A Universidade Federal do Piauí que me concedeu essa grande conquista e agradecer a minha orientadora, Juliana Oliveira de Carvalho, pela ajuda, disponibilidade, paciência e dedicação.

Obrigado!

“[...] E o conhecimento se multiplicará”.
(Daniel 12.4)

Resumo

A simulação é uma técnica bastante utilizada na área de arquitetura de computadores para reproduzir o comportamento de um sistema real. Este trabalho foca no estudo do simulador sim-cache pertencente ao conjunto de ferramentas SimpleScalar. O simulador se baseia em um processador superescalar para simular *caches*, e avaliar o desempenho desta arquitetura. Este trabalho apresenta uma interface em português baseada em níveis de conhecimento para essa ferramenta. A interface possibilita ao projetista configurar parâmetros arquiteturais, e facilita a análise e a interpretação dos resultados obtidos do simulador sim-cache.

Palavras-chave: SimpleScalar, Simulador, Memória *Cache*.

Abstract

Simulation is a widely used technique in the field of computer architecture to reproduce the behavior of a real system. This work focuses on the study of the simulator sim-cache belonging to the SimpleScalar toolset. The simulator is based on a superscalar processor caches to simulate and evaluate the performance of this architecture. This paper presents a Portuguese interface based on knowledge levels for this tool. The interface allows the designer to configure architectural parameters, and facilitates the analysis and interpretation of the results of the simulator sim-cache

Keywords: SimpleScalar, Simulator, Memory Cache.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CPU *Central Processing Unit*

E/S Entrada/Saída

FIFO *First In First Out*

ISA *Instruction Set Architecture*

LRU *Least Recently Used*

MIPS Milhões de Instruções por segundo

PISA *Portable* ISA

RISC *Reduced Instruction Set Computers*

TLB *Table Lookaside Buffer*

SUMÁRIO

1. Introdução

Com o avanço da tecnologia, percebe-se um aumento considerável da complexidade dos sistemas computacionais, principalmente no que diz respeito ao processamento e armazenamento de dados. Com isso, a memória de um computador se tornou um componente rápido e complexo. Por esse motivo, o ensino de algumas disciplinas, como as de Arquitetura e Organização de computadores, torna-se uma tarefa um pouco difícil, já que não é algo palpável. Para facilitar o entendimento do funcionamento das memórias, simuladores são utilizados para representar o funcionamento das mesmas tornando-as mais visíveis.

A memória de um computador apresenta grande diversidade em relação ao tipo, à tecnologia, à organização, ao desempenho e ao custo. Um sistema de computação típico é equipado com uma hierarquia de subsistemas de memória, sendo algumas delas internas e outras externas (STALLINGS, 2010).

Para acompanhar o avanço rapidamente alcançado pelo processador em termos de velocidade, o acesso do mesmo à memória passou a utilizar técnicas, afim de compensar esse desequilíbrio e, entre essas técnicas, incluem as memórias *cache* (STALLINGS, 2010).

Para a modelagem em *software* de processadores superescalares, um dos simuladores mais conhecidos é o SimpleScalar, o qual é um conjunto de ferramentas e simuladores que permitem desde uma simples simulação funcional a uma complexa simulação e que simulam o processamento em arquiteturas superescalares, para análise de desempenho de processadores, desenvolvido e gratuitamente disponibilizado pelo departamento de computação da universidade de *Wisconsin-Madison(EUA)* (AUSTIN, 1997).

Este trabalho descreve um estudo sobre o simulador sim-cache que pertence ao conjunto de ferramentas SimpleScalar, e apresenta uma interface gráfica em português para esse simulador, que é um simulador funcional de memória *cache* que possibilita simulações em *cache* de instrução, *cache* de dados e *cache* unificada, e ainda avalia *caches* até dois níveis.

A interface gráfica Sim-Cache permite a análise e comparação dos vários dados da memória *cache* através dos resultados gerados nas

simulações. Procurando apresentar visualmente, de forma simples e interativa, o que acontece nos acessos aos sistemas de memória *cache* reais. Estes apresentam-se de forma a facilitar o aprendizado sobre funcionamento da *cache*, com a possibilidade de ter mais facilidade para compreender o funcionamento e o desempenho de um sistema de memória *cache* do que usando somente os exercícios convencionais.

2. Memória Cache

Nos sistemas informatizados houve grandes evoluções tecnológicas em relação a desempenho e custo, exemplo disso é que entre 50 anos de evolução, uma máquina que custava 10 milhões de dólares executava uma instrução por segundo e, cinquenta anos depois, havia máquinas que custavam mil dólares e podiam executar um bilhão de instruções por segundo (TANENBAUM, 2000). A verdade é que com toda a evolução e desenvolvimento, a memória principal sempre foi mais lenta que o processador. Apesar de técnicas desenvolvidas para amenizar essa diferença, a tendência é que ela aumente ainda mais (MORIMOTO, 2009). Uma solução para esse problema está no uso da memória *cache*, que é utilizado como uma memória temporária que fica entre o processador e a memória principal, fazendo que o processador busque os dados mais utilizados na memória *cache*, em vez de buscar na memória principal, ou seja, essa busca realizada na *cache* reduz significantemente o tempo de execução de cada instrução (SMITH, 1982).

A *cache* é constituída por uma memória pequena e veloz que auxilia a execução da CPU (*Central Processing Unit*). A memória *cache* é uma palavra de origem francesa que significa “um lugar seguro para esconder ou armazenar coisas” (PATTERSON, 2005, p. 358).

2.1 Fundamentos da Cache

Segundo (STALLINGS, 2010), “o uso de memória *cache* visa obter uma velocidade de acesso à memória próxima da velocidade das memórias mais rápidas e, ao mesmo tempo, disponibilizar no sistema uma memória de grande capacidade, a um custo equivalente ao das memórias de semicondutor mais baratas”.

O objetivo do uso da memória *cache* está relacionado ao fato de ter uma memória rápida próxima aos registradores e com um preço relativamente baixo em relação aos demais tipos de memórias. É mostrada na Figura 1 a relação entre a CPU, memória *cache* e a memória principal.

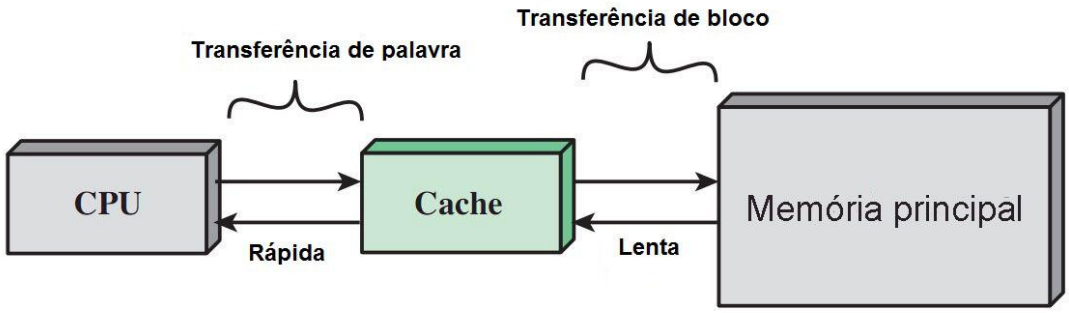


Figura 1 - Relação entre CPU, Cache e Memória principal.

Fonte: (STALLINGS, 2010)

Sabe-se que a memória *cache* está localizada entre a CPU e a memória principal, sendo assim quando a CPU tentar fazer um processo de leitura da memória principal ele primeiramente irá fazer uma verificação na memória *cache* que contém várias partes da memória principal, estas partes são chamadas de blocos (MORIMOTO, 2005). Todo esse processo vai ser mais rápido se for feito na memória *cache* já que a mesma é mais rápida que a memória principal, mas poderão ocorrer situações em que CPU ao tentar ler alguma palavra da *cache* não a encontre ocorrendo assim uma *cache miss,* ou seja, isto quer dizer que ocorreu uma falha no acesso a *cache*. Quando há uma situação desse tipo a CPU recorre à memória principal e traz uma cópia desse bloco para a *cache*.

A memória *cache* atualmente é organizada em níveis como é mostrada na Figura 2.

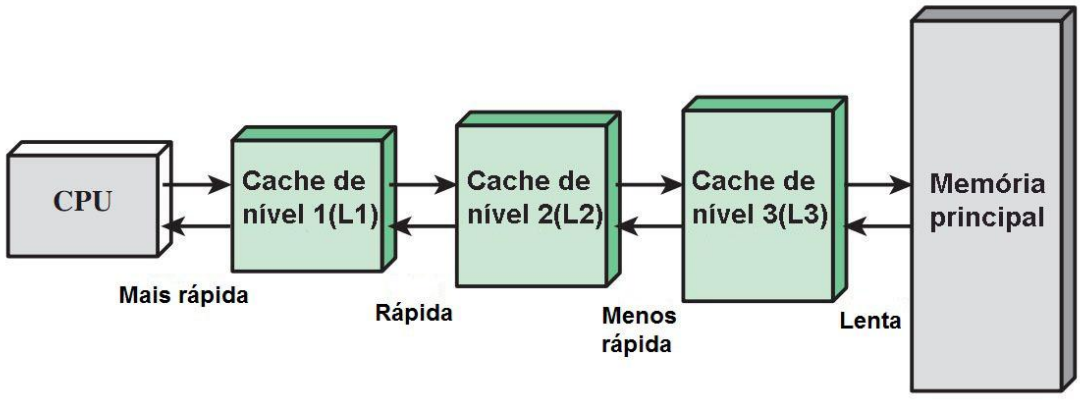


Figura 2 - Organização da cache em três níveis

Fonte: (STALLINGS, 2010)

A *cache* de nível 3 (L3) é mais lenta e geralmente maior que a *cache* de nível 2 (L2), que por sua vez é mais lento e consequente maior que a *cache* de nível 1 chamada de L1.

A Figura 3 apresenta a estrutura da memória *cache* e da memória principal.

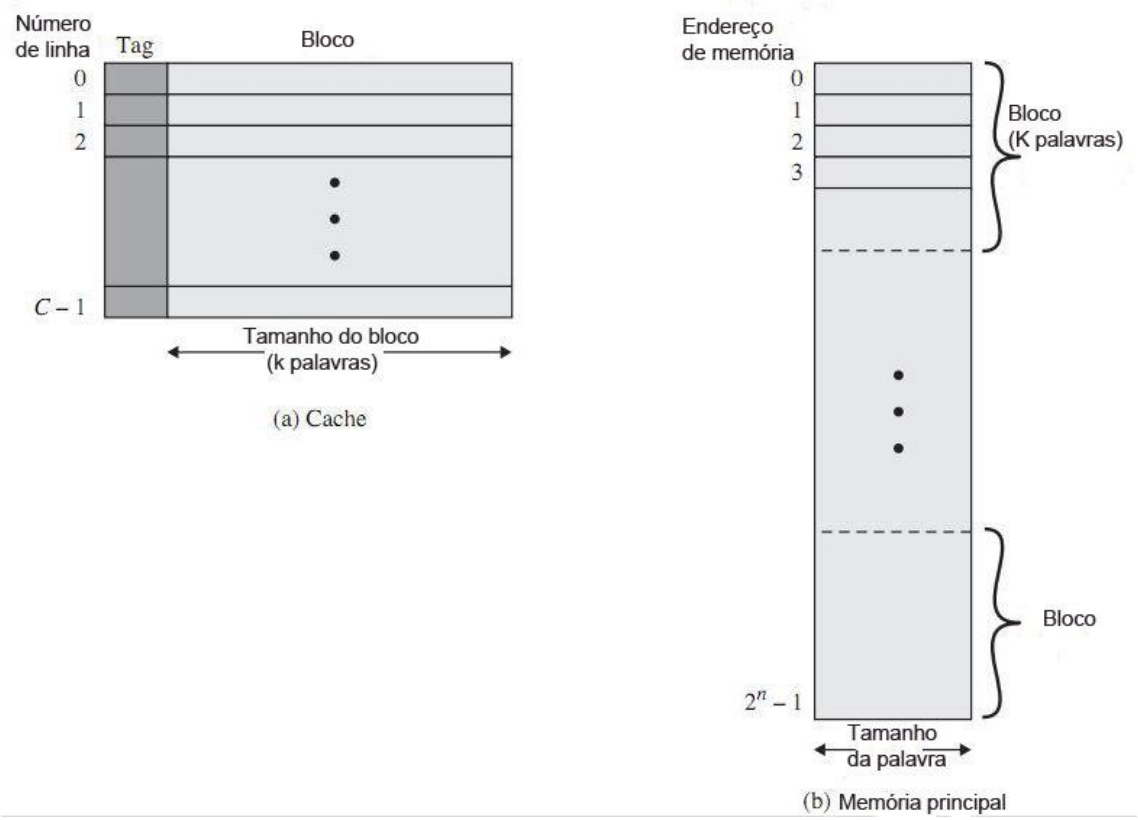


Figura 3 - Estrutura da cache e da memória principal

Fonte: (STALLINGS, 2010)

2.2 Hierarquias de Memória

A memória é dividida em hierarquias, onde uma hierarquia pode ser utilizada para a execução de uma aplicação em específico, então para o desenvolvimento de um projeto é preciso definir e optar pelas várias formas de hierarquias que pode apresentar uma memória (TANENBAUM, 2000).

2.2.1 Hierarquia de dois níveis separados

Na Figura 4, são apresentados dois níveis de memória *cache*, sendo dois para instruções e dados.

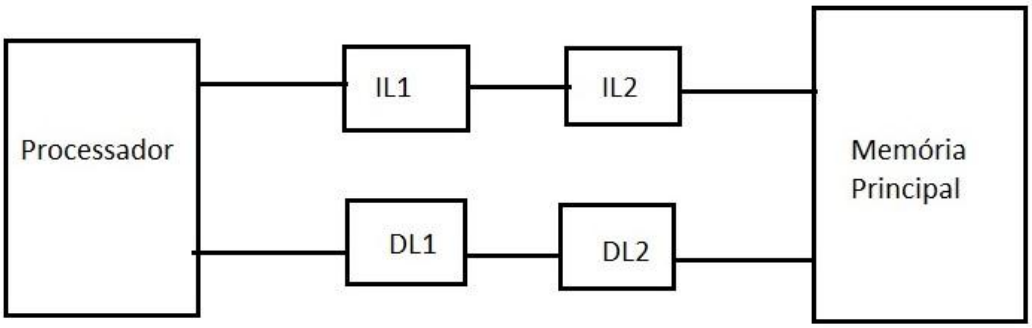


Figura 4 - Hierarquia de dois níveis separados

Fonte: (CARVALHO, 2010)

O IL1 apresenta o primeiro nível da *cache* de instruções: quando uma instrução for procurada na *cache* ela passará primeiramente ao IL1, caso ocorra uma falha (*miss)* a instrução será procurada no IL2, caso não ocorra uma *miss* novamente no IL2, a instrução será procurada na memória principal. O mesmo ocorre para os níveis de dados DL1 e DL2.

2.2.2 Hierarquia de dois níveis, com o segundo nível unificado

Na Figura 5, são apresentados dois níveis de memória *cache*, sendo o primeiro composto por instruções e dados com o segundo sendo unificado.

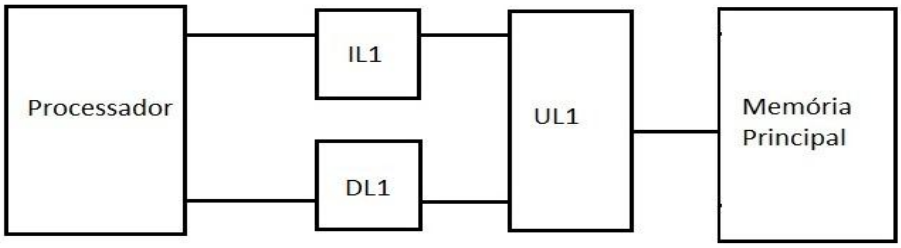


Figura 5 - Hierarquia de dois níveis, com o segundo nível unificado.

Fonte: (CARVALHO, 2010)

O IL1 apresenta o primeiro nível da *cache* de instruções, quando uma instrução for procurada na *cache* ela passará primeiramente ao IL1, caso ocorra uma falha (*miss)* a instrução será procurada no UL1, a instrução será procurada na memória principal. O mesmo ocorre para o nível de dados DL1.

2.2.3 Hierarquia com um nível separado

Na figura 6, é apresentado um nível de memória *cache*, sendo esse nível composto por instruções (IL1) e dados (DL1).

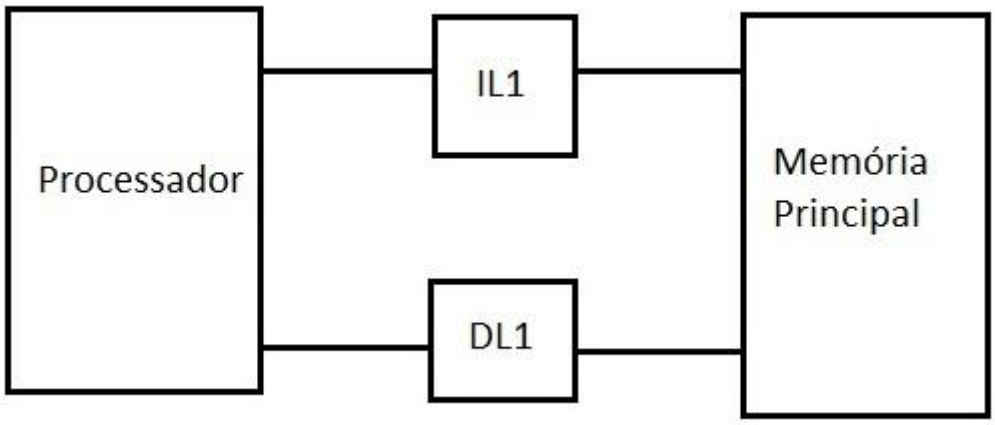


Figura 6 - Hierarquia com um nível separado

Fonte: (CARVALHO, 2010)

O IL1 apresenta o primeiro nível da *cache* de instruções: quando uma instrução for procurada na *cache* ela passará primeiramente ao IL1, caso ocorra uma falha (*miss)* a instrução será procurada na memória principal, o mesmo ocorre para o DL1.

2.3 Políticas de substituição da cache

Quando um novo bloco precisa ser colocado em uma *cache* mapeada de forma associativa, uma fenda disponível deve ser identificada. Se existem fendas livres, como por exemplo, existe quando o programa inicia sua execução, a primeira fenda com um *bit* válido zero pode ser usada. Quando todos os bits válidos de todas as fendas das *caches* forem 1, contudo, uma das fendas ativas deve ser liberada para o novo bloco (STALLINGS, 2002).

As políticas de reposição são comumente usadas pelo processador fazendo com que as informações contidas na *cache* sejam substituídas por outra mais eficiente naquele determinado momento. Comumente são usadas quatro políticas de substituição. São elas: menos recentemente usada (*Least Recently Used* – LRU), primeira a entrar primeira a sair (*First in First Out* – FIFO), menos frequentemente usada (*Least Frequently Used* – LFU) e a aleatória.

A menos recentemente usada (*Least Recently Used* – LRU) é a política de substituição mais popular, por ser a mais eficaz. Este método utiliza uma pilha que agrupa as linhas da *cache* recentemente menos usada. Quando uma linha da *cache* precisar ser substituída por uma informação vinda da memória principal, ela será substituída pela menos recentemente usada que é descartada e o novo bloco é escrito na mesma.

A menos frequentemente usada (Least Frequently Used – LFU) é uma política que funciona de forma semelhante à LRU, ou seja, quando uma informação é necessária para um novo bloco, a menos frequentemente usada é liberada. Essa política possui uma contador que acompanha cada frequência de uso da linha da memória. Quando for escolhida a linha para a substituição, a linha com a contagem recentemente menos usada é descartada, ou seja, substitui o bloco menos recentemente utilizado.

A primeira a entrar primeira a sair (*First in First Out* – FIFO) é uma política que faz referência a uma estrutura de dados em fila, onde o primeiro a entrar, é o primeiro a sair. A ideia fundamental dessa política é de que um novo bloco na *cache* é inserido no final da fila e só se pode retirar o bloco do início da fila. A escolha desse bloco não leva em consideração a quantidade de vezes que o bloco está sendo utilizado, por isso não é muito utilizado, pois pode comprometer o sistema, apesar de ser de fácil implementação.

A política aleatória simplesmente escolhe aleatoriamente um bloco da *cache*, sem precisar da implementação de pilha ou fila (MURDOCA, 2000).

2.4 Mapeamento da cache

O mapeamento da *cache* dá-se entre as linhas da *cache* e os blocos da memória principal, sabendo que há mais blocos da *cache* do que linhas da *cache*. A função de mapeamento vai atuar para verificar qual o bloco da memória principal está ocupando a linha da *cache*. Nas seções seguintes serão apresentadas três funções de mapeamento, são elas: mapeamento direto, mapeamento associativo e associativo em conjunto (*set associative*).

2.4.1 Mapeamento direto

A Figura 7 mostra uma técnica de mapeamento direto para uma memória *cache* L2 de 512KB e memória principal de 1GB. Cada linha da *cache* corresponde a um conjunto de blocos da memória principal, no mapeamento direto cada bloco da memória principal pode ser mapeado exclusivamente para uma linha. A técnica de mapeamento direto se mostra simples e de baixo custo para se implementar, sendo que sua principal desvantagem é a existência de um local fixo para cada bloco.

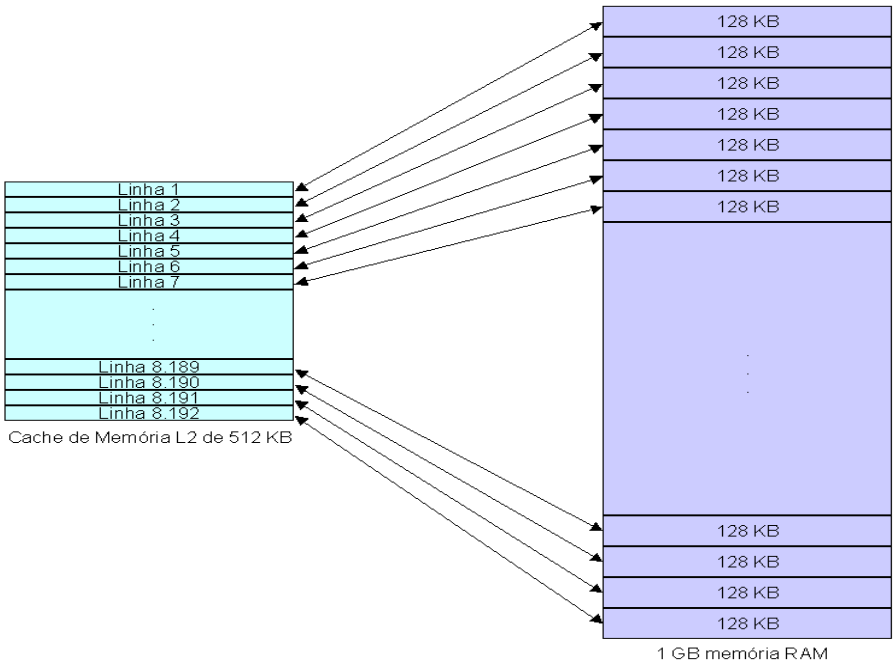


Figura 7 - mapeamento direto

Fonte: (STALLINGS, 2010)

2.4.2 Mapeamento associativo

A Figura 8 mostra uma técnica de mapeamento associativo para uma memória *cache* L2 de 512KB e memória principal de 1GB. Esta técnica de mapeamento sobrepõe às desvantagens do mapeamento direto, pois permite que os blocos da memória principal sejam utilizados nas diversas linhas da *cache* de forma não fixa, oferecendo uma enorme flexibilidade para os blocos que estão sendo substituídos da memória principal para a *cache*, dessa a *cache* passará a ter uma boa taxa de acerto.

A principal desvantagem do mapeamento associativo é a complexidade de circuito para comparar as *tags* de todas as linhas da *cache* em paralelo (STALLINGS, 2010).

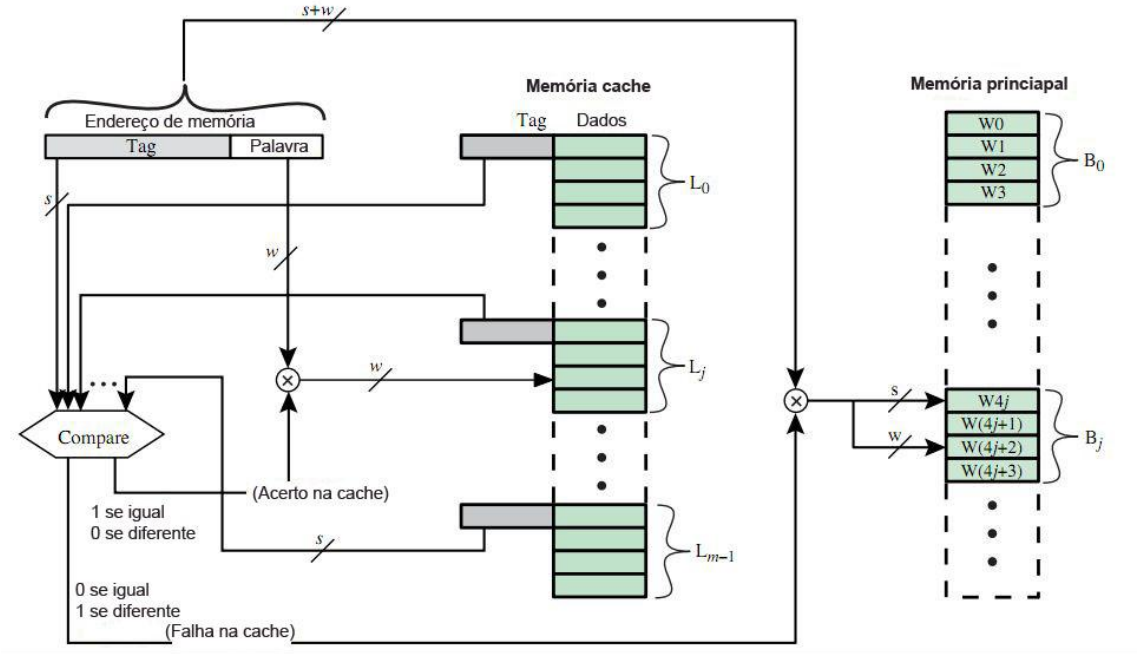


Figura 8 - Mapeamento associativo

Fonte: (STALLINGS, 2010)

2.4.3 Mapeamento associativo em conjunto

O mecanismo de mapeamento associativo por conjunto combina a simplicidade do mapeamento direto com a flexibilidade do mapeamento associativo. Mapeamento associativo por conjunto é mais prático do que mapeamento completamente associativo porque a porção associativa é limitada apenas a algumas fendas que constituem um conjunto (NETO, 2004).

O mapeamento associativo por conjunto visa unir as vantagens do mapeamento direto e associativo reduzindo suas vantagens. Nesta técnica a *cache* representa vários conjuntos possuindo várias linhas como é mostrada na Figura 9.

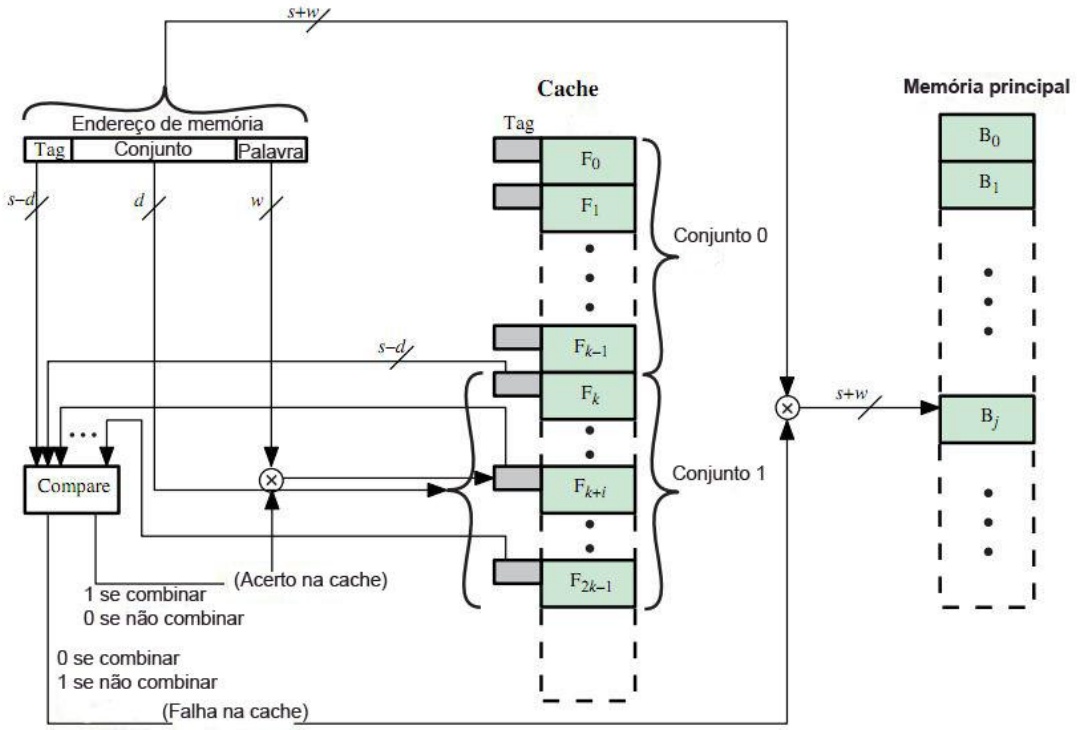


Figura 9 - Mapeamento associativo por conjunto
Fonte: (STALLINGS, 2010)

3. Sim-Cache

O sim-cache é um simulador funcional de memória *cache* que possibilita o teste em *cache* de instrução, *cache* de dados e *cache* unificada, e ainda avalia *caches* até dois níveis. No relatório de saída podem ser obtidos parâmetros como: valores absolutos de acertos e erros na busca de dados e instruções na *cache*, taxas de erros e acertos, número de instruções testadas, tempo de simulação, acessos em cada nível de *cache* e a velocidade de simulação (TODD AUSTIN, 1997; DOUG BURGER, 1997).

Neste capítulo será abordada a ferramenta SimpleScalar, já que o simulador sim-cache faz parte da mesma, suas vantagens, descrição dos simuladores e o seu simulador de memória *cache* sim-cache com suas principais características.

3.1 Ferramenta SimpleScalar

O conjunto de ferramentas SimpleScalar é um sistema de infra-estrutura de *software* utilizado para construir aplicações de modelagem para a análise de desempenho de um programa, a modelagem da micro arquitetura detalhada de *hardware* e *software* (AUSTIN, 1997; BURGER, 1997).

Utilizando as ferramentas SimpleScalar, os usuários podem criar aplicações de modelagem que simulam programas que podem ser executados em uma gama de sistemas e processadores modernos. O conjunto de ferramentas inclui simuladores de vários tipos variando de um simulador rápido funcional a um detalhado modelo do processador, de forma dinâmica agendada que suporta *caches* não-bloqueantes, execução especulativa, preditores de desvios estáticos e dinâmicos, simuladores com estatísticas funcionais e programas de avaliação de desempenho. As ferramentas SimpleScalar são amplamente utilizados para pesquisa, exemplo disso é que, em 2000 mais de um terço de todos os trabalhos publicados em conferências de arquitetura da computação utilizaram as ferramentas SimpleScalar para avaliar seus projetos. Além de simuladores, o conjunto de ferramentas SimpleScalar inclui ferramentas de visualização de desempenho, recursos de análise estatística e de depuração e de infraestrutura de verificação(AUSTIN, 1997).

O conjunto de ferramentas SimpleScalar é distribuído com todo o código fonte para uso acadêmico não comercial, os usuários comerciais podem adquirir uma licença de uso comercial a partir do SimpleScalar LLC, onde os usuários poderão encontrar informações detalhadas sobre o licenciamento

comercial, permitindo que os usuários possam expandir o código, e adaptar os modelos existentes para suas próprias ideias.

As ferramentas de simulação do SimpleScalar foram escritos por Todd Austin, quando ele realizava seu Ph.D. na Universidade de Wisconsin-Madison nos Estados Unidos. O conjunto de ferramentas SimpleScalar atualmente são mantidos e distribuídos gratuitamente para uso acadêmico. As ferramentas de simulação são uma das ferramentas mais utilizadas por pesquisadores acadêmicos na área de microarquitetura.

Ele não é apenas um simulador de processadores, mas um conjunto de simuladores, compiladores e ferramentas que permitem desde uma simples simulação funcional a uma complexa simulação de um processador do estado-da-arte. As primeiras versões do conjunto de ferramentas incluem contribuições de Doug Burger e Sohi Guri.

3.1.1 Vantagens do SimpleScalar

O simulador SimpleScalar apresenta algumas características que o diferem de outros simuladores por ser bastante complexo e de apoio a vários tipos de simulações devido a seus simuladores e ferramentas de apoio. Suas principais vantagens são:

Extensibilidade: Ao adquirir o simulador o usuário tem ao seu alcance todo código fonte e documentação das ferramentas, o que possibilita o usuário expandir a ferramenta.

Portabilidade: O SimpleScalar possui uma boa portabilidade isso permite sua utilização em várias arquiteturas como mostra a tabela 1.

Tabela 1 – Portabilidade do SimpleScalar

Arquitetura	Sistema Operacional	Compilador
x86	Free BSD 2,2	Gcc
x86	CygWin32/Windows NT	Gcc
x86	Linux1.3	Gcc
x86	Solaris2	Gcc
SPARC	SunOS 4.1.3	Gcc
SPARC	Solaris 2	Gcc
RS6000	AIX 4.1.3	Gcc
RS6000	AIX 4.1.3	Xlc

PA-RISC	HPUX	Gcc
Alfa	DEC Unix 3,2	Gcc
Alfa	DEC Unix 3,2	c89

Fonte: www.simplescalar.com

Detalhado: Vários simuladores são incluídos na distribuição possibilitando uma gama de testes e simulações detalhada dos aspectos de *hardware* e *software*.
Desempenho: Seus simuladores possuem um bom desempenho podendo executar milhões de instruções por segundo.

A ferramenta possui algumas outras vantagens que são: Facilitar o processo de validação de uma arquitetura, além de permite avaliar o desempenho do sistema através do modelo computacional, e atualmente, por ser *open source*, possui suporte para arquiteturas como ARM, Intel, Alpha, entre outras e permite flexibilidade na exploração de diferentes projetos.

3.1.2 Instalação

Nesta seção são descritos alguns passos para a instalação do *SimpleScalar* em um sistema operacional Linux, no *Ubuntu* 11.10.

A instalação do *SimpleScalar* é um processo detalhado em que, primeiramente é preciso instalar o compilador GCC (*GNU Compiler Collection*) para realizar a execução dos arquivos em C.

Observando que para realizar a instalação é preciso fazer o *download* dos arquivos disponíveis no *site* SimpleScalar como é mostrado na figura 10.

Figura 10 - Arquivos de instalação do SimpleScalar

Através do *site* de manutenção da ferramenta SimpleScalar é possível adquirir um guia que apresenta ao usuário a maneira como realizar a instalação completa da ferramenta seguindo alguns passos.

A partir dos *downloads* os arquivos deveram ser copiados em um diretório, descompactados e fixar as variáveis de ambiente para facilitar a instalação como mostra a figura 11.

```
$ tar xzvf simpletools-2v0.tgz
$ tar xzvf simplesim-3v0d.tgz
$ tar xzvf simpleutils-990811.tar.gz
$ rm-rf gcc-2.6.3
```

Figura 11 – Descompactação dos arquivos SimpleScalar

Durante a instalação do simplesim-3.0 o usuário deverá configurar o simulador para as opções PISA ou ISA de acordo com a figura 12.

```
$ cd $IDIR/simplesim-3.0
$ make config-pisa
$ make CC="gcc32"
```

Figura 12 – Configuração do Simulador

No final da instalação da ferramenta o usuário pode realizar testes para identificar se a instalação foi bem-sucedida de acordo com o comando apresentado na Figura 13. A opção do seguinte comando permite ao usuário escolher qualquer um dos oitos simuladores disponíveis, além de poder escolher o programa a ser testado.

```
$ cd $IDIR/simplesim-3.0
$ ./<nome_simulador> tests/bin.little/<nome_do_arquivo_teste>
```

Figura 13 – Teste SimpleScalar

3.1.3 Conjunto de simuladores

O SimpleScalar é composto por oito simuladores e algumas ferramentas de auxilio a estes simuladores. A seguir é feita uma descrição dos seus simuladores com exceção do sim-cache que é reservado uma parte do trabalho para falar especificadamente sobre ele.

- **Sim-Fast:** É o simulador mais simples, rápido e menos detalhado (simulação funcional). Este simulador retorna estatística como: taxa de instruções executadas por segundo e número total de instruções.

- **Sim-Safe:** Versão melhorada do *Sim-Fast* permite alta velocidade de simulação e retorna os mesmos resultados do *Sim-Fast*. Acrescenta resultados como o total de acesso a dados (*Loads* / *Stores*) executados, verifica se há erros nas instruções e permite a limitação no total de instruções executadas.

- **Sim-Profile:** Tem como função principal a determinação do perfil dos programas de avaliação de desempenho *(benchmarks)*. Este simulador fornece estatísticas como o número de instruções executadas, tipos de instruções, número de vezes que foram executadas e distribuição de classes de instruções.

- **Sim-Cheetah:** Permite a simulação de várias configurações de *cache* em apenas uma execução do simulador. Isto simplifica a simulação de um *benchmark*. O desempenho da *cache*, nas simulações, pode ser avaliado em grande faixa de associatividade e número de conjuntos.

  O objetivo principal desta versão de simulador de *cache* é agilizar os resultados de simulação para *cache* de tal forma que possamos contar com resultados sem ter que parametrizar item por item (SUGUMAR, 1993).

- **Sim-Eio:** É usado para gerar *external event traces* (EIO *traces*) e *checkpoint* dos executáveis. Os *external event traces* capturam a execução de um programa e armazenam os dados em um simples arquivo para uma re-execução mais tarde.

- **Sim-Bpred:** É um simulador de preditores de desvio e tem como função principal o teste de desempenho de preditores estatísticos e dinâmicos como, por exemplo, preditor bimodal, dois níveis, combinado e etc.

- **Sim-Outorder:** Este simulador introduz a possibilidade de análise de tempo/ritmo das simulações, permitindo o teste das características funcionais e desempenho das configurações.

3.1.4 Estruturas do SimpleScalar

O SimpleScalar foi desenvolvido na linguagem C e em código aberto permitindo a alteração do seu código. Ele é utilizado para realizar simulações precisas, flexíveis e rápidas de arquiteturas superescalares. Essa ferramenta para manter sua portabilidade utiliza duas versões a *big-endian* e a *little-endian.* Ela pode ser utilizada em arquiteturas Alfa e PISA (*Portable* ISA), mas outras estão sendo adicionadas ao SimpleScalar. O conjunto de instruções PISA é um conjunto de instruções simples MIPS (*Millions of Instructions Per Second*) que é usado principalmente para fins didáticos e baseados no uso de registradores (PRICE, 1995). As suas instruções tem à disposição um conjunto de 32 registradores para realizar as operações.

Os processadores MIPS são do tipo RISC (*Reduced Instruction Set Computer*), ou seja, possui um conjunto de instruções reduzido e não há micro programação ou microcódigo, suas instruções são executadas diretamente pelo *hardware* com baixo nível de complexidade (TANENBAUM, 2007).

O conjunto de ferramentas utiliza arquivos binários compilados para a arquitetura SimpleScalar e a partir dai simula a sua execução em um dos seus simuladores. O *cross*-compilador cria um tipo de ilusão em que a máquina subjacente em que as instruções são executadas, passam a executar instruções do PISA. A Figura 14 mostra essa estrutura de simulação.

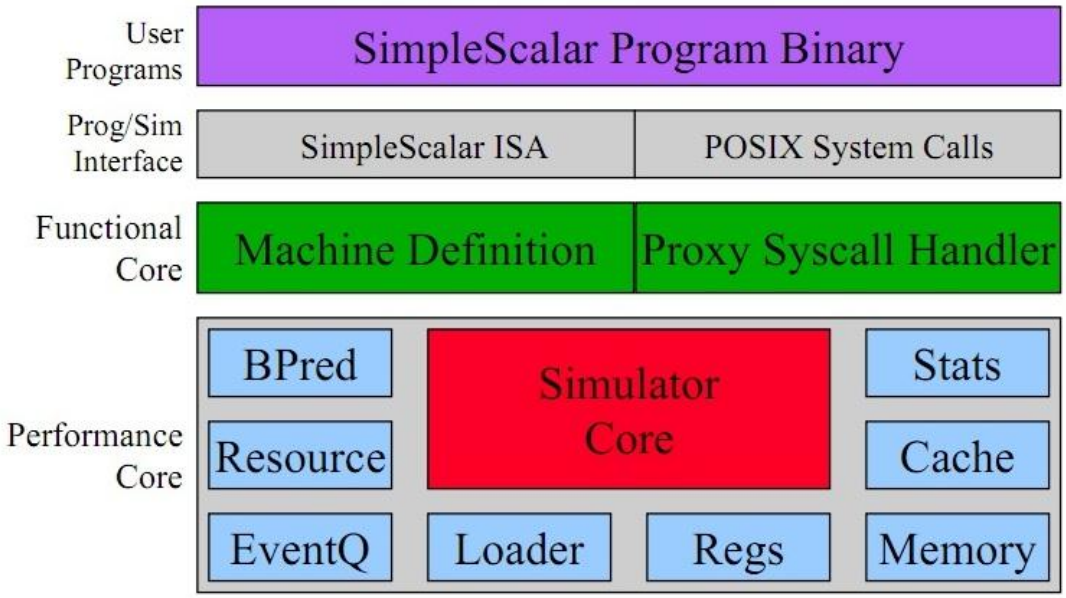


Figura 14- Diagrama de blocos da arquitetura SimpleScalar

Fonte: www.simplescalar.com

O SimpleScalar apresenta algumas ferramentas de desenvolvimento que incluem: compilador C, *Assembler*, *Linker*/*Loader*, simuladores, bibliotecas e utilitários que é apresentado na Figura 15.

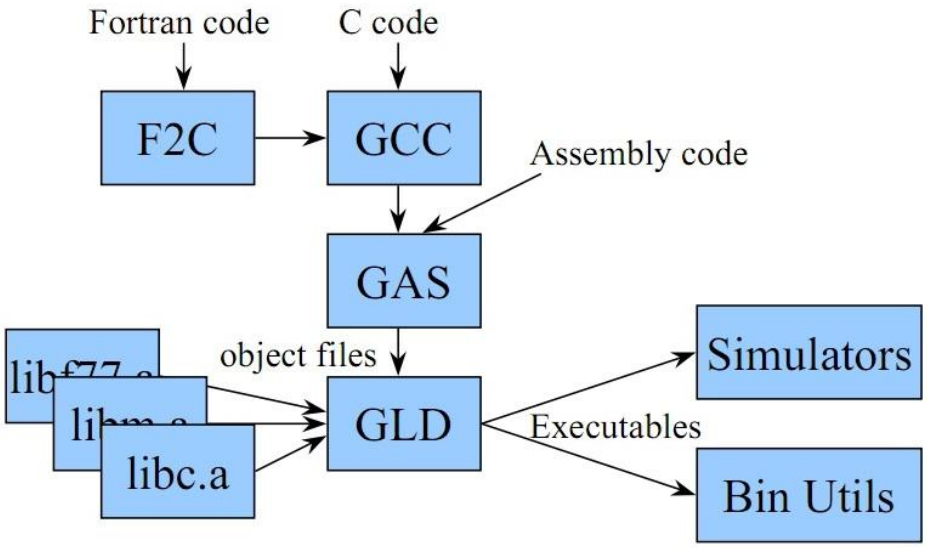


Figura 15 - Visão geral do conjunto de ferramentas SimpleScalar

Fonte: www.simplescalar.com

A Figura 16 mostra a arquitetura de *software* de um modelo de *hardware* no SimpleScalar. Os programas simulados rodam nos simuladores usando uma técnica chamada *execution-driven simulation* (LARSON; et al, 2002), a qual requer a inclusão de um emulador do conjunto de instruções da arquitetura e de um módulo de emulação de entrada e saída. O emulador do conjunto de instruções interpreta cada instrução direcionando as atividades do modelo de *hardware* através de *interfaces* de *callback* que o interpretador fornece. Os interpretadores são escritos usando uma linguagem de definição que provê um mecanismo para a descrição de como instruções alteram o estado de registradores e da memória (SOARES, 2005). Sendo que a ferramenta também possui uma arquitetura de *software* compostas pelo modelo de *hardware* e simuladores, núcleo de desempenho funcional.

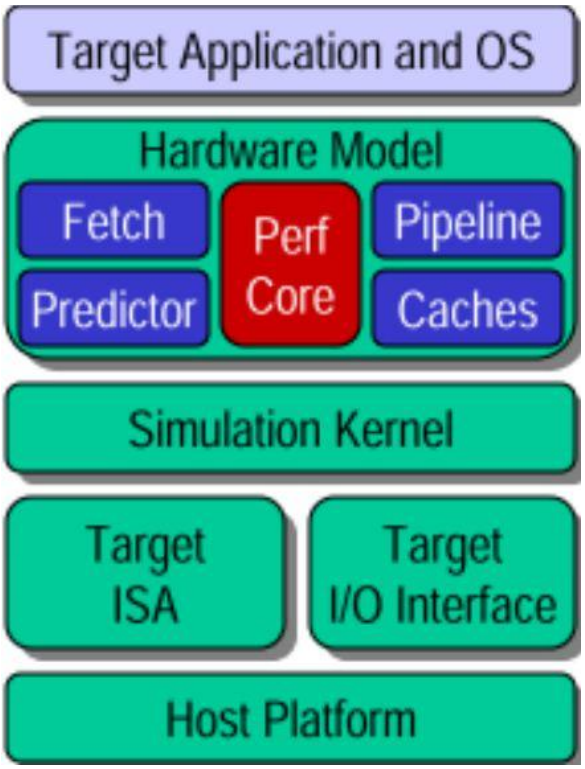


Figura 16 - Arquitetura de software SimpleScalar

Fonte: www.simplescalar.com

3.2 simulador sim-cache

O sim-cache é um simulador funcional de memória *cache* que pertence a um conjunto chamado SimpleScalar, essa ferramenta apresenta simuladores específicos sendo que para a simulação de memória *cache* há o *sim-cheetah* e o sim-cache. Sendo que o presente trabalho consiste em estudar e analisar apenas o sim-cache por suas características consideradas essenciais para o desenvolvimento do trabalho.

O sim-cache gera e possibilita analisar várias estatísticas de simulação entre elas destacam-se: o número total de instrução que foram executadas, tempo que durou a simulação em segundos, velocidade da simulação (em instruções por segundo), número de acertos e falhas, total de substituições, taxas de substituição de falhas, total de acessos. Estes são apenas alguns resultados gerados por esse simulador, além disso, cada resultado desses é gerado para os níveis l1, l2 da *cache* e para *cache* unificada.

O sim-cache possui como um de seus parâmetros o limite de instrução, o qual é apenas uma amostragem, como é feito em outros simuladores. Além deste parâmetro o sim-cache possui: **-cache:dl1, -cache:il1, cache:l2, -cache:dtlb e -cache:itlb**, onde cada um deste possui seus respectivos parâmetros sendo eles mostrados na Figura 17.

```
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
```

Figura 17 - Parâmetros de configuração do sim-cache

A Figura 18 apresenta um exemplo que é usado para configurar o parâmetro da *cache* dl1 com um tamanho de 4096 KB, tamanho do bloco da *cache* igual a 32 KB e conjunto de 1 caminho de associatividade, com política de substituição aleatória, é usado o seguinte comando:

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

Figura 18 - configuração do parâmetro da cache DL1

Na tabela 2, são mostrados alguns parâmetros de configuração que é possível estabelecer nos dados de entrada da simulação do sim-cache:

Tabela 2 - Parâmetros de configuração do sim-cache

–config	Configurar o simulador via arquivo externo
-h	Imprimir auxílio na tela
-d	Habilitar mensagem de debug
-redir:sim	Redirecionar saída da simulação
-redir:prog	Redirecionar saída do programa
-max:inst	Limitar o número de instruções testadas
-cache:il1 ou -cache:il2	Configurar parâmetros com: número de conjuntos, tamanho de bloco, associatividade e política de reposição da *cache* de instruções.
-cache:dl1 ou -cache:dl2	Configurar parâmetros como: número de conjuntos, tamanho de bloco, associatividade e política de reposição da *cache* de dados.
-cache:ul1 ou -cache:ul2	Configurar parâmetros como: número de conjuntos, tamanho de bloco, associatividade e política de reposição da *cache* unificada.

Fonte: www.simplescalar.com

4. Interface Sim-Cache

Neste capítulo será abordada a interface desenvolvida para o simulador sim-cache, chamada de interface Sim-Cache, apresenta a interface desenvolvida e as simulações utilizadas com a interface Sim-Cache.

A interface sim-cache foi desenvolvida utilizando a linguagem de programação *Java*® (HORSTMANN, 1999), o ambiente de desenvolvimento foi o *Netbeans* IDE 6.9, por ser um ambiente gratuito e oferecer vários recursos para a implementação. O sistema operacional no qual a interface sim-cache foi desenvolvida e implementada é o *Linux*, distribuição *Ubuntu* 11.10. Esta foi escolhida por ser uma linguagem orientada a objetos, que facilita a modularização e a especificação das classes do projeto e, principalmente, por ser uma linguagem multiplataforma, permitindo que o *software* seja utilizado em qualquer sistema e por ser uma linguagem simples, distribuída, interpretada, robusta, segura, portátil, multifunções, dinâmica e apresentar uma grande facilidade de uso. Com isso, o usuário precisa apenas ter uma máquina virtual *Java* (JVM – *Java Virtual Machine*) instalada no computador para que seja possível a execução do programa. Sendo assim, a interface sim-cache é um *software* independente de plataforma, além de ser gratuito e de código aberto.

4.1. Aplicação

A interface gráfica sim-cache foi desenvolvida com o intuito de facilitar o uso do simulador sim-cache. Através da interface gráfica com o usuário, o simulador permite executar vários tipos de simulações, que são técnicas de avaliação que representa o comportamento, de um sistema no domínio do tempo por meio de um modelo (FERRARI, 1988).

A interface sim-cache inicialmente apresenta uma tela principal, nessa tela são exibidas as opções de simulações baseadas em níveis de conhecimento, apresentado na figura 19.

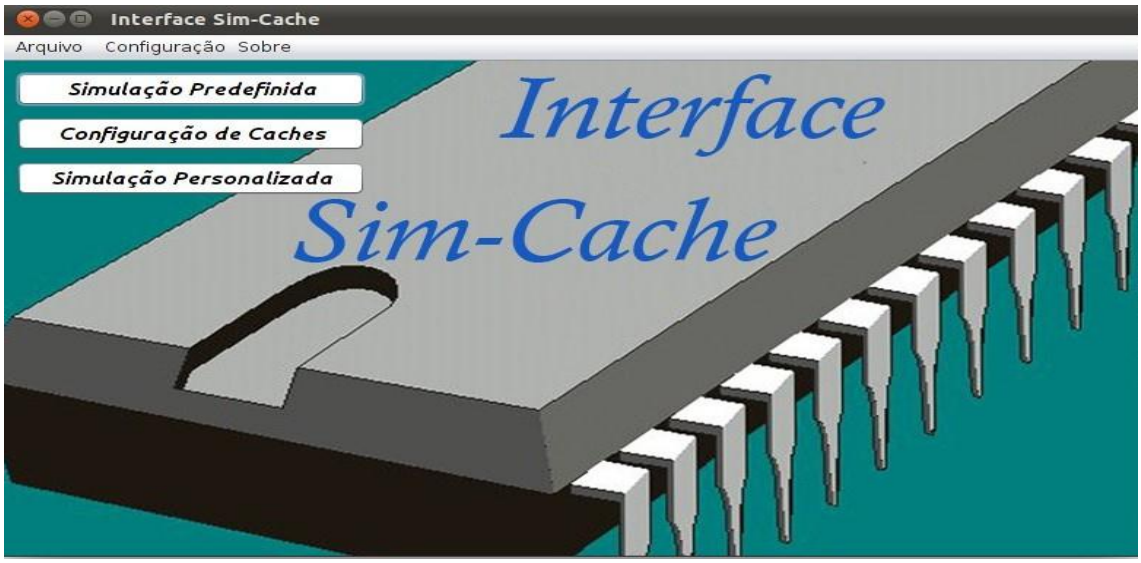


Figura 19 – Tela principal da interface gráfica sim-cache

A partir da tela principal o usuário tem acesso a três botões que dão acesso a três tipos de simulações, são elas: simulação predefinida, configuração de *caches* e simulação personalizada, além do item de menu *sobre* que permite abrir uma janela contendo informações a respeito do simulador.

Na simulação predefinida é apresentada uma janela contendo informações de configurações usadas pelo sim-cache, a janela permite ao usuário escolher configurações que deseja usar para fazer a simulação. Essas configurações são: configurações de *cache* de dados L1 e L2, configuração de *cache* de instruções L1 e configurações de instruções e de dados TLB. Sendo que a primeira opção de escolha é o teste que vai ser utilizado na simulação, como é mostrado na Figura 20.

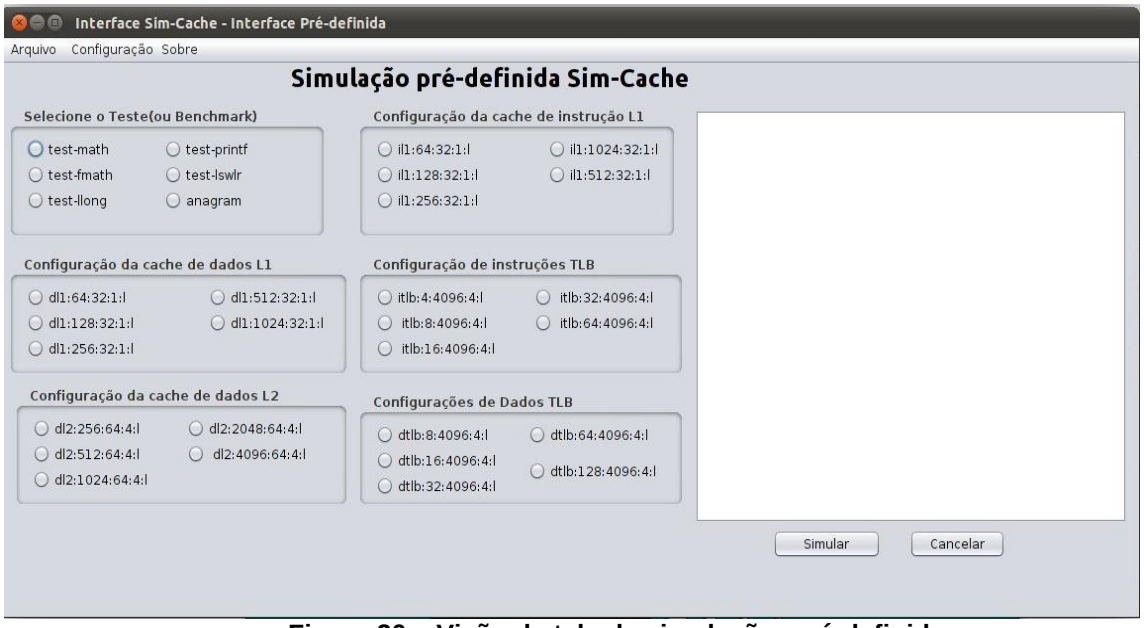


Figura 20 – Visão da tela de simulação: pré-definida

Para realizar uma simulação na tela exibida acima é necessário que todas as opções sejam preenchidas, não sendo necessário que isto seja feito em ordem. Na opção de configuração de *cache* de dados L1, temos no primeiro item as letras dl1 que é um parâmetro usado no simulador sim-cache que significa que a *cache* de dados L1 vai ser configurada. Ainda no primeiro item o número 64 significa que o tamanho da *cache* dados dl1 é de 64KB, o número 32 apresenta o número de conjuntos da *cache*, o número 1 a presenta a associatividade da *cache* e a letra l significa que política de substituição usada na simulação é a menos recentemente usada (LRU), essa política é adotada nas demais opções por ela ser a política usada de forma padrão no simulador sim-cache.

A interface configuração de *cache* permite ao usuário definir a configuração dos níveis da memória *cache* que deseja simular. É possível definir manualmente os parâmetros de cada nível, assim como o programa teste, tamanho, conjunto e associatividade e política de substituição da *cache*, como é mostrada na Figura 21.

A opção de selecionar o programa para simulação possibilita ao projetista da simulação escolher o programa a ser utilizado na simulação, esses programas são utilizados da pasta *tests* do pacote da ferramenta SimpleScalar, sendo que para um outro programa ser utilizado na simulação ele precisa primeiramente ser convertido para ser reconhecido e,

posteriormente, usado nas simulações. A tela ainda apresenta uma legenda na parte esquerda da interface, mostrando as políticas de substituição. A interface compõe três botões, o primeiro é o botão executar que realiza os eventos executando a simulação, o segundo nova simulação limpa as informações da tela que apresenta o resultado da simulação, o botão cancelar limpa todos os dados da interface, permitindo uma nova simulação.

Os parâmetros da *cache* possibilitam a simulação e configuração de acordo com a necessidade do projetista. A *cache* de instruções chamada de IL1, através de um campo permite que o projetista digite o tamanho da mesma em KB, o número de conjuntos, a associatividade da *cache* e definir uma das três políticas que podem ser utilizadas na simulação. O projetista pode utilizar das mesmas configurações ou do mesmo esquema para a *cache* de dados DL1 e unificada UL2.

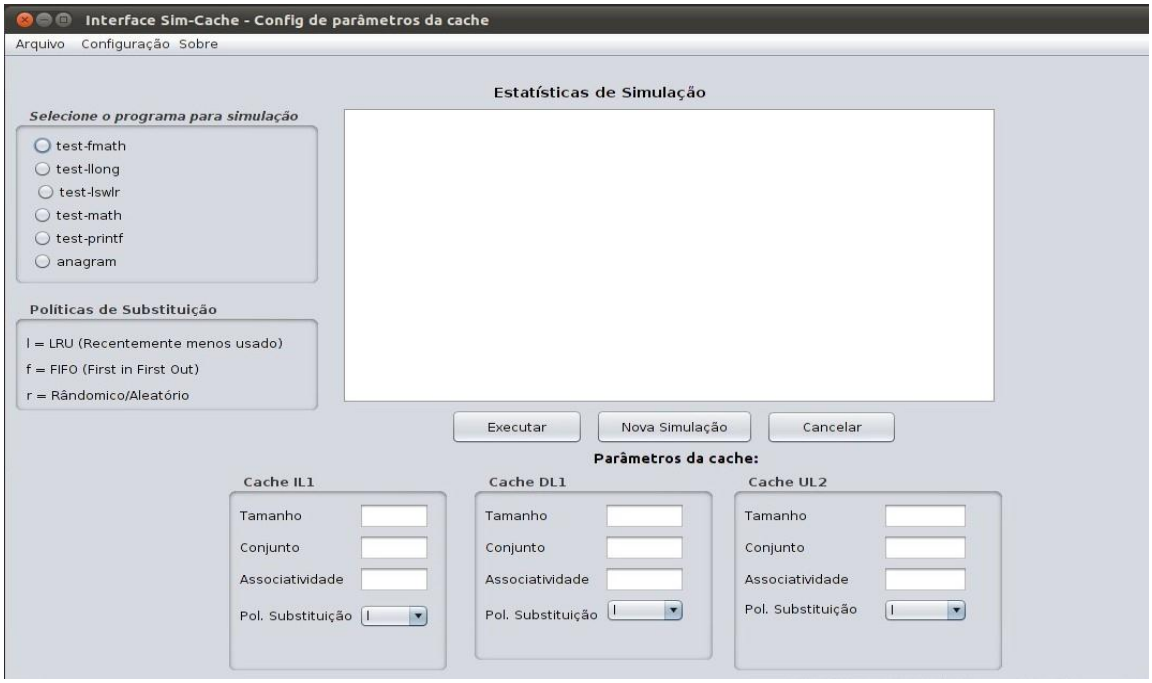


Figura 21 - Visão da tela de simulação: configuração de cache

A simulação personalizada apresenta uma interface simples que procura simular o uso do simulador através da linha de comando usada em terminal Linux. Como é mostrada na Figura 22.

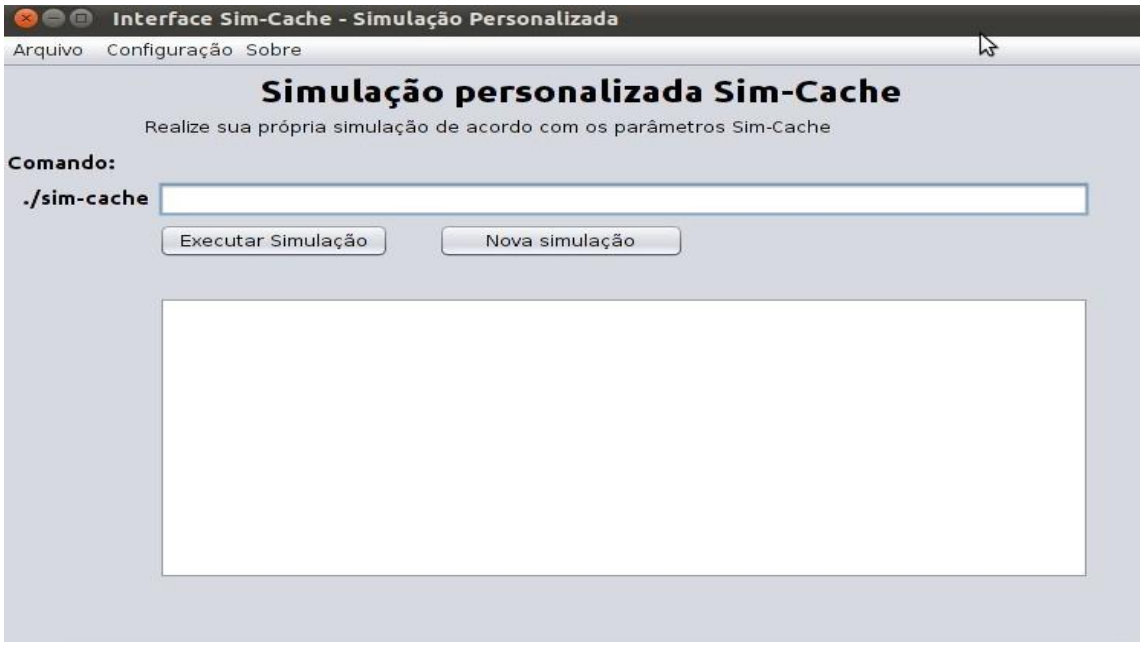


Figura 22 - Visão da tela de simulação: simulação personalizada

Por fim, a Figura 23 mostra a tela de menu Sobre. A interface gráfica sim-cache apresenta essa tela que contém informações sobre o uso do simulador sim-cache. Essas informações ajudarão aos usuários, caso tenham alguma dúvida ou busquem informações adicionais sobre o simulador.

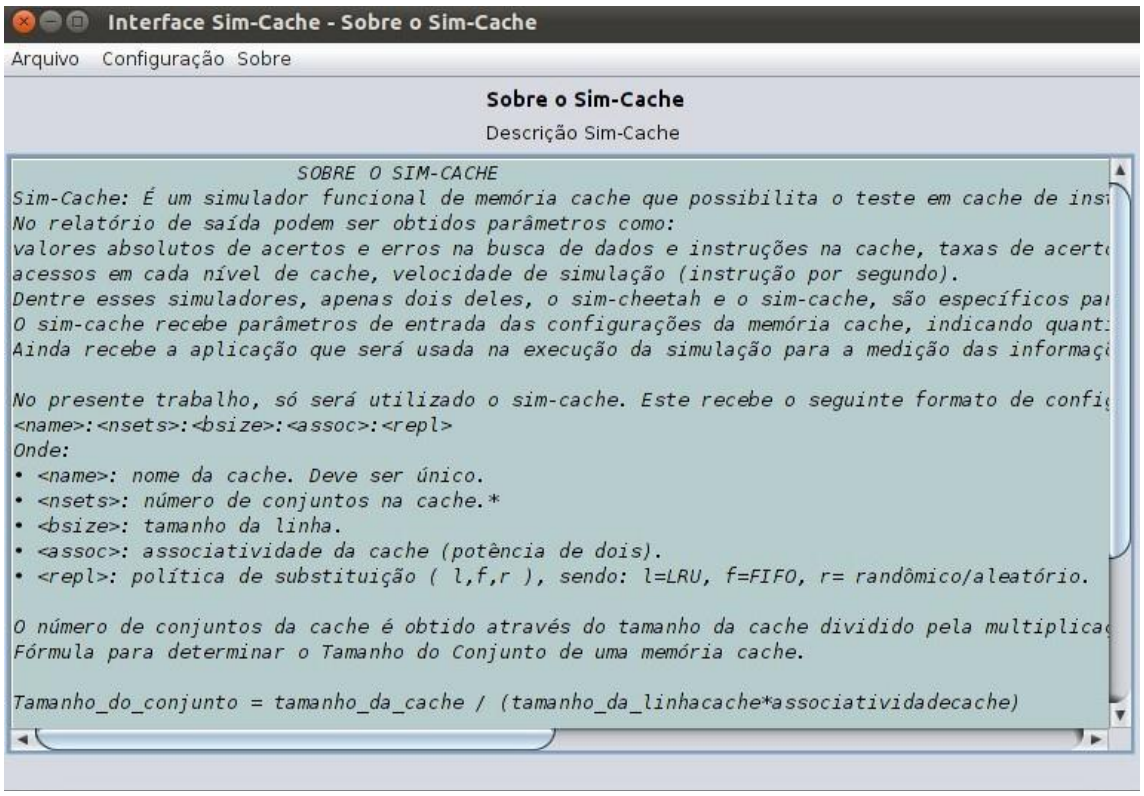


Figura 23 - Visão da tela de informação sobre a interface sim-cache

4.2 Simulação

O ambiente de simulação é basicamente composto de duas partes: a primeira é o simulador sim-cache do conjunto de ferramentas SimpleScalar, que realiza as simulações e gera os resultados. A segunda parte é a interface gráfica sim-cache, a qual realiza a configuração do sim-cache, carrega os dados a serem executados pelo simulador e seus respectivos argumentos de execução como também realizam a leitura e processamento dos resultados obtidos pelo simulador, apresentando-os na interface.

A comunicação entre a ferramenta e o simulador, é realizada pelas chamadas do sistema operacional, ou seja, a interface gráfica sim-cache realiza a montagem da linha de comando, e a realiza no simulador sim-cache. Este último gera o resultado da simulação em um arquivo de texto e, em seguida, captura os valores da simulação e envia-os à interface gráfica, como está esquematizado na Figura 24.

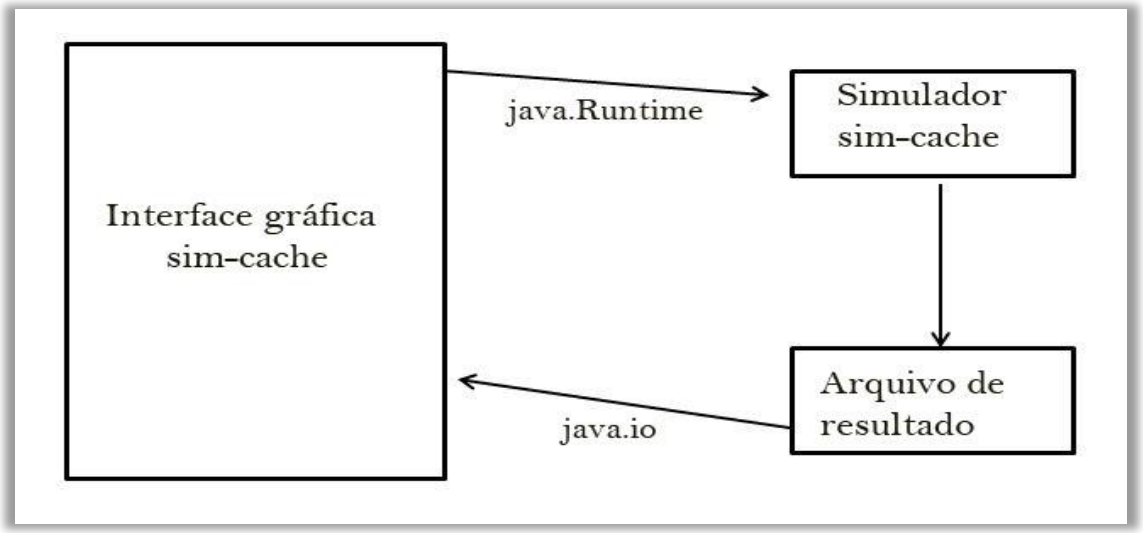


Figura 24 – Comunicação entre as ferramentas que compõe a interface gráfica sim-cache

Uma das principais dificuldades foi encontrar um meio que fizesse a ligação entre a interface e o simulador sim-cache que funciona por meio de terminal no *Linux* para realizar as simulações e obter os resultados estatísticos. A solução para esses problemas foi encontrada em algumas classes do Java. As principais são a classe *Runtime* que executa uma aplicação externa ou

comandos através de um de seus métodos, o *exec(),* que combinado com os parâmetros do simulador sim-cache é possível salvar as estatísticas de simulação em um arquivo de texto. A partir dai, outro ponto crucial do projeto é a análise desse arquivo de texto gerado com os resultados da simulação, a classe *BufferedReader* e, em alguns de seus métodos, é possível ler o arquivo de texto e, posteriormente, capturar apenas os dados que representam os dados mais relevantes para o projeto proposto.

Para testar as funcionalidades da interface gráfica sim-cache, foram feitas diversas simulações com programas teste, já que os mesmos vem com a própria ferramenta SimpleScalar. A interpretação de resultados obtidos por simulações via terminal é uma atividade complexa, como ocorre com a ferramenta SimpleScalar. Isto torna difícil a análise e o aprendizado sobre esses dados. Portanto, uma das mais importantes funcionalidades da interface gráfica sim-cache é permitir que os resultados obtidos através das simulações sejam dispostos de forma clara e objetiva por meio de interpretações sobre cada linha que é apresentada na simulação.

A Figura 25 apresenta o exemplo de uma simulação realizada com a interface de configuração de *cache*, onde o projetista tem opção de definir a configuração desejada da *cache* de instrução IL1, da *cache* de dados IL2 e da *cache* unificada UL2, configurando os parâmetros que são: Tamanho da *cache*, Número de conjunto da *cache*, associatividade da *cache* e a política de substituição da *cache*, que é disponibilizada uma legenda na interface apresentando as políticas de substituição da *cache*. Pelo resultado da simulação, podem-se analisar as estatísticas da simulação que são: total de acesso a *cache*, total de acertos *cache*, total de falhas, total de substituições e taxa de falha a *cache*.

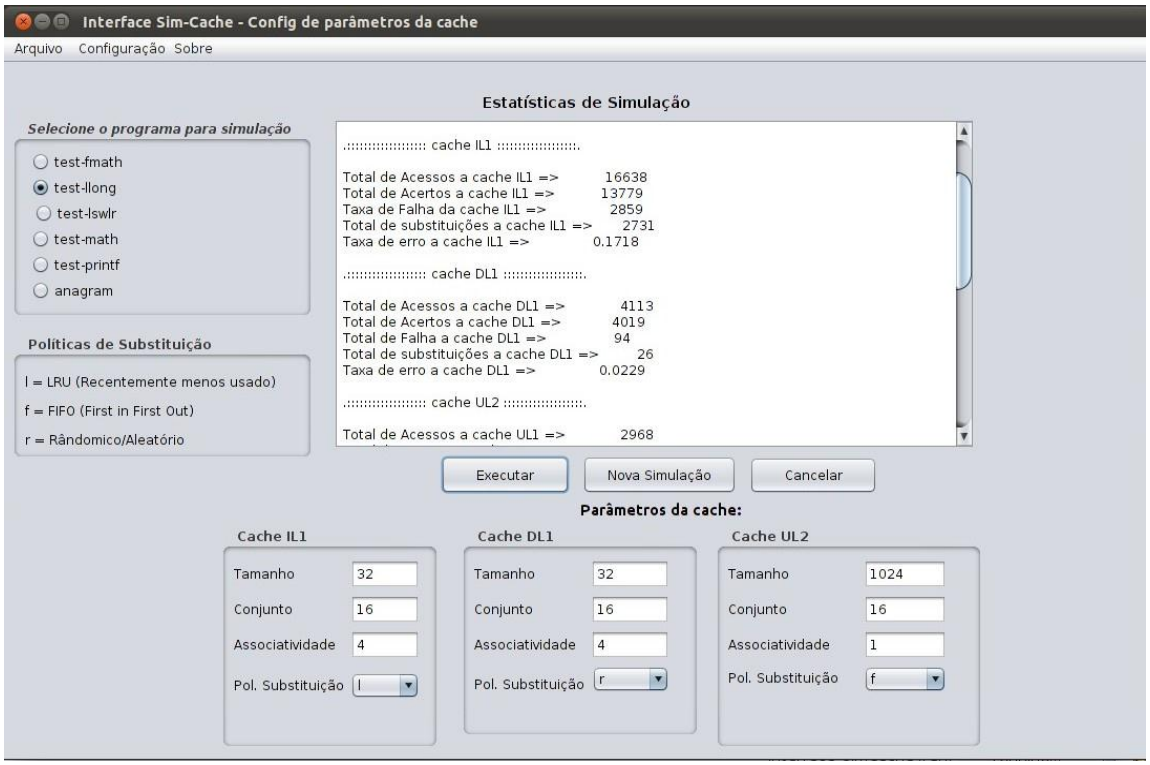


Figura 25 – Simulação realizada através da interface: Configuração de cache

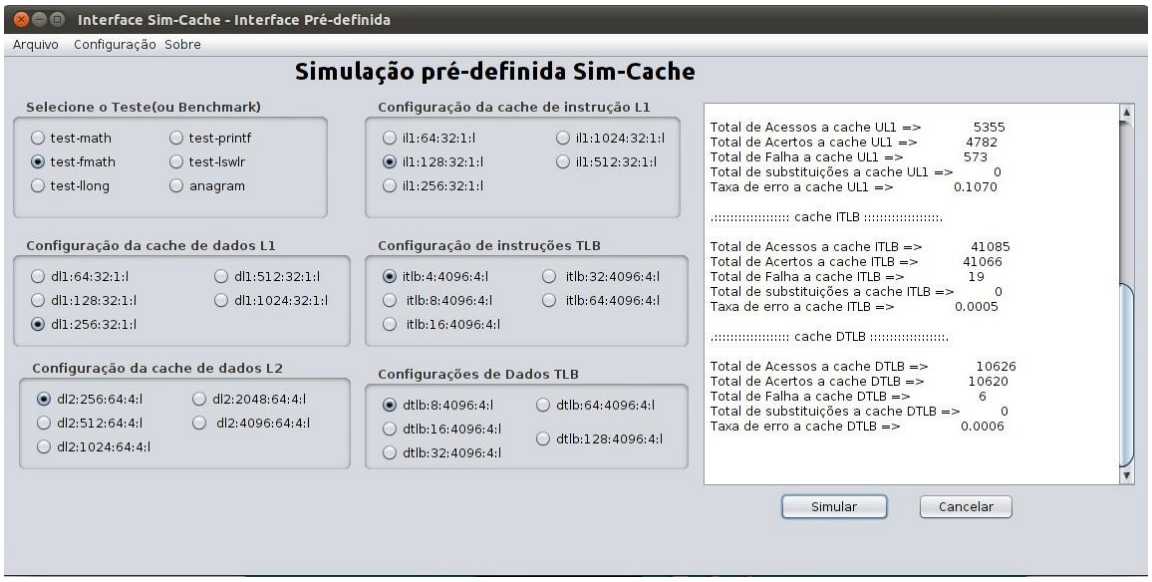
A Figura 25 demostra uma simulação com as *caches* IL1 e IL2 de tamanhos de 32 KB, de 16 conjuntos e contendo um total de 4 associatividade respectivamente, porém há uma diferença entre elas em relação a política de substituição, o que torna seus dados estatísticos bastante diversos. Sendo assim, o projetista poderá analisar o comportamento da *cache* UL2 e das demais que desejar.

Além dos resultados acima, a simulação apresenta um cabeçalho com informações como mostra a Figura 26. São obtidas informações como: data em que ocorreu a simulação, número total de simulações executadas e velocidade em que ocorreu a simulação. Tais informações podem ser obtidas em as interfaces de simulação.

Figura 26 - Simulação realizada através da interface: Simulação personalizada

A imagem da interface, apresentada na Figura 26, apresenta o comando com a execução de um programa chamado *anagram* que se encontra no caminho percorrido acima. Os dados obtidos nesta simulação são referentes ao programa acima. Quando uma simulação é feita dessa maneira, sem o uso de parâmetros das *caches*, ele utiliza valores padrões nos parâmetros para a realização da simulação.

Por fim, na Figura 27, a simulação apresenta os resultados relacionados ao TLB que funciona como um *buffer,* que armazena endereços em uso da *cache* (MORIMOTO, 2011).



27 - Simulação realizada através da interface: Simulação pré-definida

A interface pré-definida sim-cache apresenta ao projetista apenas opções de marcar os itens requeridos para realizar as simulações, geralmente os resultados estatísticos variam muito de acordo com o programa teste utilizado, já que cada um possui um código personalizado. Dos parâmetros de configuração das *caches,* apenas o tamanho de cada item varia, facilitando a análise do funcionamento que uma simulação ocorre.

Essas interfaces facilitam as análises dos resultados no processo de simulação que é possível verificar o resultado de comportamento de qualquer um dos parâmetros, podendo os resultados ser visualizados de diversas formas.

5. Conclusão

Neste trabalho foi apresentado a ferramenta SimpleScalar, analisando um de seus simuladores de memória *cache* e desenvolvido uma interface gráfica para o mesmo utilizando a linguagem *Java*. A interface foi desenvolvida adicionando as principais funcionalidades do sim-cache, podendo ser adicionados futuros dados para a simulação.

A interface Sim-Cache apresenta um ambiente simples, prático e eficiente que vai facilitar a simulação e a análise dessas simulações, utilizando vários tipos de simulações podendo o usuário configurá-las. Isto pode tornar muito mais interessante do ponto de vista dos usuários do sim-cache que poderão ser motivados e demostrar interesses em conceitos ligados à simulações e a memória *cache*.

Para a interface Sim-Cache chegar ao ponto de realizar as mesmas simulações usadas no simulador sim-cache foi necessário utilizar de várias classes e métodos da linguagem Java a fim de prover uma ligação com o simulador sim-cache. Posteriormente fazer com que os comandos contendo os parâmetros sejam reconhecidos pelo sim-cache, a partir dai capturar os resultados da simulação em um arquivo de texto que é carregado pelo Java e analisado em linhas e colunas desse arquivo afim de capturar apenas a parte principal e demonstrar os valores através de um simples *click* deixando todo esse processo de forma transparente para facilitar o entendimento do usuário da interface gráfica Sim-Cache.

Como trabalhos futuros, pretende-se inserir à interface Sim-Cache gráficos que possibilitem uma melhor visualização dos dados estatísticos, como a porcentagem dos níveis das *caches* mais acessadas, permitir à interface adicionar outros programas criados, podendo até serem criados pelo próprio usuário para ser usado na simulação, e poder estender todas essas funcionalidades aos demais simuladores da ferramenta SimpleScalar, sabendo que cada simulador possui parâmetros próprios e possui funcionalidades diferentes.

REFERÊNCIAS

AUSTIN, T. M. *A User and Hacker's Guide to the SimpleScalar Architectural Research Tool Set (for tool set release 2.0). Slides presented at 30th Annual International Symposium on Microarchitecture*, Jan. 1997. Disponível em <http://www.simplescalar.com > Acesso em 03 de Novembro de 2011.

AUSTIN, TODD. M. Um Guia do usuário e *Hacker* para Pesquisa Arquitetural SimpleScalar. Conjunto de Ferramentas, 1997. Disponível em: <http://www.cs.virginia.edu/~skadron/cs654/slides/hack_guide.pdf>, Acesso em 25 de Novembro de 2011.

BURGER, D. C.; AUSTIN, T. M. *The SimpleScalar Tool Set: version 2.0. ACM SIGARCH Computer Architecture News*, v. 25, n. 3, p. 13-25, 1997.

CARVALHO, G.R. *Framework* para Análise e Exploração de Arquitetura visando Aspectos de Consumo de Energia e Desempenho. 2010. 69 f.. Trabalho de graduação – Centro de informática, UFPE, Recife, 2010.

LARSON, E. et al. SimpleScalar: *An Infrastructure for Computer System Modeling. Computer, New York*, v.35, n.2, p. 59-67, Feb. 2002

FERRARI, D. *Computer systems performance evaluation. Englewood Cliffs: Prentice Hall*, 1988.

HORSTMANN, Cay S.; Cornell, Gary. "*Core Java 2", Vol. 1, Prentice Hall*, 1999.

MORIMOTO, CARLOS. E. Memória Cache - Definição de Memória Cache. 2005. Artigo em *Hypertexto*. Disponível em: <http://www.aa.htm>. Acesso em 10 de janeiro de 2012.

MURDOCA, M.J.; HEURING, V. P. Introdução à Arquitetura de Computadores. Rio de Janeiro: Campus, 2000.

NETO, G.F.O. Proposta de Conjunto de Simulações para Análise de Desempenho de Processadores Superescalares e Ensino de Arquitetura de Computadores. 2004. 171 f.. Dissertação (Mestrado) – Instituto de informática, UFRGS, Porto Alegre, 2004.

PATTERSON, David A., Hennessy, John L. Organização e Projeto de Computadores - A Interface *Hardware/Software*.

PRICE, C. MIPS IV *Instruction Set, revision 3.1. Mountain View*, CA, USA: MIPS *Technologies*, 1995.

SOARES, N.S. T&D-Bench - Explorando o Espaço de Projeto de Processadores em Ensino e em Pesquisa. 2005. 168 f.. Tese (Doutorado em Ciência da Computação) – Instituto de informática, UFRGS, Porto Alegre, 2005.

STALLINGS, William. Arquitetura e organização de computadores, oitava edição, Stallings*, Prentice Hall*, São Paulo - 2010.

SUGUMAR, R. A.; ABRAHAM, S. G. *Efficient Simulation of Caches under Optimal Replacement with Applications to Miss Characterization. IN: ACM SIGMETRICS CONFERENCE ON MEASUREMENT AND MODELING OF COMPUTER SYSTEMS. Proceedings... pp. 24-35, May* 1993.

TANENBAUM, Andrew S; Woodhull, A.S. Sistemas Operacionais: Projetos e Implementação. 2.ed. Porto Alegre: *Bookman*, 2000, 341p.

TANENBAUM, Andrew S. Organização Estruturada de Computadores. 5ª ed. Prentice Hall, São Paulo, SP p. 33, 2007.

ANEXO A: ARQUIVO DE RESULTADO DO SIM-CACHE

```
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: ./sim-cache tests-pisa/bin.little/test-math
sim: simulation started @ Thu Apr  5 12:08:32 2012, 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            <null> # 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       dl1:256:32:1:l # l1 data cache config, i.e.,
{<config>|none}
-cache:dl2       ul2:1024:64:4:l # l2 data cache config, i.e.,
{<config>|none}
-cache:il1       il1:256:32:1:l # l1 inst cache config, i.e.,
{<config>|dl1|dl2|none}
-cache:il2                 dl2 # 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 **
pow(12.0, 2.0) == 144.000000
pow(10.0, 3.0) == 1000.000000
pow(10.0, -3.0) == 0.001000
str: 123.456
x: 123.000000
str: 123.456
x: 123.456000
str: 123.456
x: 123.456000
123.456 123.456000 123 1000
sinh(2.0) = 3.62686
sinh(3.0) = 10.01787
h=3.60555
atan2(3,2) = 0.98279
pow(3.60555,4.0) = 169
169 / exp(0.98279 * 5) = 1.24102
3.93117 + 5*log(3.60555) = 10.34355
cos(10.34355) = -0.6068,  sin(10.34355) = -0.79486
x     0.5x
x0.5     x
x   0.5x
-1e-17 != -1e-17 Worked!

sim: ** simulation statistics **
sim_num_insn                213703 # total number of instructions
executed
sim_num_refs                 56899 # total number of loads and stores
executed
sim_elapsed_time                 1 # total simulation time in seconds
sim_inst_rate           213703.0000 # simulation speed (in insts/sec)
il1.accesses                213703 # total number of accesses
il1.hits                    189940 # total number of hits
il1.misses                   23763 # total number of misses
il1.replacements             23507 # total number of replacements
il1.writebacks                   0 # total number of writebacks
il1.invalidations                0 # total number of invalidations
```

```
il1.miss_rate                0.1112 # miss rate (i.e., misses/ref)
il1.repl_rate                0.1100 # 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)
dl1.accesses                  57480 # total number of accesses
dl1.hits                      56676 # total number of hits
dl1.misses                      804 # total number of misses
dl1.replacements                548 # total number of replacements
dl1.writebacks                  416 # total number of writebacks
dl1.invalidations                 0 # total number of invalidations
dl1.miss_rate                0.0140 # miss rate (i.e., misses/ref)
dl1.repl_rate                0.0095 # replacement rate (i.e.,
repls/ref)
dl1.wb_rate                  0.0072 # writeback rate (i.e., wrbks/ref)
dl1.inv_rate                 0.0000 # invalidation rate (i.e.,
invs/ref)
ul2.accesses                  24983 # total number of accesses
ul2.hits                      23780 # total number of hits
ul2.misses                     1203 # total number of misses
ul2.replacements                  0 # total number of replacements
ul2.writebacks                    0 # total number of writebacks
ul2.invalidations                 0 # total number of invalidations
ul2.miss_rate                0.0482 # miss rate (i.e., misses/ref)
ul2.repl_rate                0.0000 # replacement rate (i.e.,
repls/ref)
ul2.wb_rate                  0.0000 # writeback rate (i.e., wrbks/ref)
ul2.inv_rate                 0.0000 # invalidation rate (i.e.,
invs/ref)
itlb.accesses                213703 # total number of accesses
itlb.hits                    213680 # total number of hits
itlb.misses                      23 # 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               0.0001 # 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                 57480 # total number of accesses
dtlb.hits                     57470 # total number of hits
dtlb.misses                      10 # total number of misses
dtlb.replacements                 0 # total number of replacements
dtlb.writebacks                   0 # 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               0.0000 # replacement rate (i.e.,
repls/ref)
dtlb.wb_rate                 0.0000 # 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                  91744 # program text (code) size in
bytes
ld_data_base             0x10000000 # program initialized data segment
base
ld_data_size                  13028 # 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                   33 # total number of pages allocated
mem.page_mem                   132k # total size of memory pages
allocated
mem.ptab_misses                  34 # total first level page table
misses
mem.ptab_accesses           1546911 # total page table accesses
mem.ptab_miss_rate           0.0000 # first level page table miss rate
```