

Eduardo Pereira Habkost

simEdu: Simulador de Circuitos de Caches

Curitiba

2006

Eduardo Pereira Habkost

simEdu: Simulador de Circuitos de Caches

Orientador: Prof. Dr. Roberto André Hexsel

Curitiba

2006

Sumário

1	Introdução e definições	p. 1
1.1	Introdução	p. 1
1.2	Caches e hierarquia de memória	p. 2
1.2.1	Hierarquia de memória	p. 2
1.2.2	Caches	p. 3
1.3	Simulação de circuitos	p. 6
1.3.1	TKGate	p. 6
1.3.1.1	Interface com o usuário do TKGate	p. 7
1.3.1.2	Arquitetura do TKGate	p. 8
1.4	Simulação de caches	p. 9
2	Arquitetura do projeto	p. 11
2.1	Implementação de caches no TKGate	p. 11
2.1.1	Caches implementadas	p. 12
2.1.2	Projeto do circuito	p. 12
2.1.3	Máquina de estados da cache	p. 12
2.1.4	Reutilização de módulos	p. 14
2.1.5	Caminho crítico	p. 14
2.2	Simulação dos circuitos de caches	p. 16
2.2.1	Alimentação das entradas do circuito simulado	p. 16
2.2.2	Obtenção de traçados de acessos à memória	p. 17
2.2.3	Execução de simulações longas no TKGate	p. 18

2.2.4	Validação dos resultados	p. 19
2.2.4.1	Simulação de escrita forçada no SimpleScalar	p. 19
2.2.4.2	Validação dos resultados da cache com pseudo-LRU	p. 20
3	Análise dos resultados	p. 21
3.1	Simulações executadas	p. 21
3.2	Resultados.....	p. 21
3.2.1	Taxas de acerto.....	p. 22
3.2.2	Tempo de simulação	p. 22
3.2.3	Comparação do tempo de simulação entre <i>SimpleScalar</i> e TKGate.....	p. 23
4	Conclusão	p. 25
	Referências	p. 28
	Apêndice A – Diagramas dos circuitos de cache	p. 30
	Apêndice B – Microcódigo dos circuitos de cache	p. 52
	Apêndice C – Código fonte da ferramenta trace-to-gsim.....	p. 54
	Apêndice D – Modificações realizadas em programas	p. 59
	Correção para simulações longas no TKGate	p. 59
	Suporte a escrita forçada no SimpleScalar	p. 61
	Suporte a Pseudo-LRU no SimpleScalar	p. 64

1 Introdução e definições

1.1 Introdução

A capacidade de processamento das CPUs cresce entre 35% e 55% ao ano, enquanto que o tempo de acesso à memória RAM decresce a uma taxa de 7% ao ano [4]. Além disso, tipos mais rápidos de memória têm um custo muito mais alto e capacidade de armazenamento menor. Um dos desafios no projeto de um sistema de memória é conseguir uma boa razão desempenho/custo, e uma maneira de melhorar esta razão é utilizar uma *hierarquia de memória*, composta de *memória cache* e *memória principal*.

Caches são memórias rápidas porém pequenas, que são utilizadas para armazenar os dados mais acessados da memória principal, permitindo um ganho de desempenho no acesso a memória.

Há vários parâmetros que podem variar entre os projetos de memória cache, tais como associatividade, tamanho de bloco e política de substituição de blocos. Ferramentas de simulação permitem analisar o comportamento da memória cache sendo projetada antes de sua fabricação.

Os simuladores de cache, em geral, calculam qual seria o comportamento da memória cache quando utilizando determinados parâmetros, mas nestes casos a definição da memória cache não inclui características de baixo nível do projeto, em nível de portas lógicas.

Este trabalho tem como objetivo desenvolver uma infraestrutura para simulação detalhada de caches em nível de portas lógicas, que permite a simulação baseada na execução de programas, ou baseada em *traçados*.

Para isso, foi utilizado o simulador de circuitos TKGate [3], e foi construído um sistema de alimentação do simulador, a partir de traçados de execução pré-armazenados ou durante da execução simulada de programas.

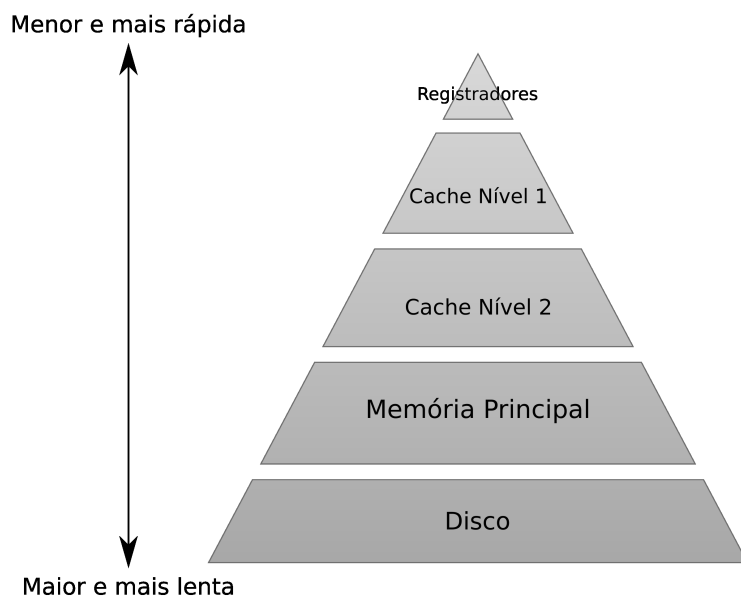


Figura 1: Exemplo de hierarquia de memória

Os resultados das simulações foram comparados com os resultados de simulações de projetos de caches simulares utilizando o simulador SimpleScalar [1].

Durante a execução do trabalho foram introduzidas melhorias no simulador no simulador TKGate – para correção de problemas – e no SimpleScalar – para implementar funcionalidades úteis para a validação dos resultados da simulação dos circuitos.

1.2 Caches e hierarquia de memória

Nesta seção o conceito de *hierarquia de memória* é apresentado, seguido de informações a respeito de memórias cache.

1.2.1 Hierarquia de memória

Uma *hierarquia de memória* permite que um sistema de memória ofereça uma velocidade de acesso média próxima à oferecida pelo tipo de memória mais rápido utilizado, porém com capacidade e custo por byte próximos do tipo de memória mais barato utilizado. Para isso, a memória é organizada em níveis, de modo que os níveis mais altos são compostos por memória mais rápida — porém menor e mais cara — e os níveis mais baixos são compostos por memória mais densa e mais barata — porém mais lenta. A Figura 1 mostra uma hierarquia de memória com todos os seus níveis.

Essa organização em níveis é eficiente e eficaz, por causa das propriedades de *localidade* temporal e espacial dos padrões de acesso à memória apresentados pelos programas. A *localidade espacial* é a tendência a acessar endereços vizinhos após o acesso a um endereço da memória. A *localidade temporal* é a tendência a acessar o mesmo endereço de memória novamente, após um breve intervalo.

Os dados mais recentemente acessados pelo processador são mantidos em um nível mais alto da hierarquia, e devido à *localidade*, a probabilidade de um próximo acesso à memória estar disponível no nível mais alto é grande. Com isso, o sistema de memória pode apresentar um tempo médio de acesso próximo ao tempo de acesso do nível mais alto da hierarquia, mas um custo por byte próximo ao custo do nível mais baixo.

Na computadores modernos, a hierarquia de memória comumente apresenta os seguintes níveis:

Registradores do processador Acesso mais rápido possível (geralmente apenas um ciclo do processador), geralmente com capacidade de algumas centenas de bytes.

Cache de nível 1 Tempo de acesso de uns poucos ciclos de processador. Geralmente possui algumas dezenas ou poucas centenas de kilobytes. As memórias cache são abordadas com mais detalhe na próxima seção.

Cache de nível 2 Maior tempo de acesso que a cache de nível 1, porém de maior capacidade. Geralmente possui algumas centenas de kilobytes, com tempo de acesso da ordem de uma dezena de ciclos.

Cache de nível 3 Maior tempo de acesso que a cache nível 2, porém de maior capacidade. Alguns sistemas possuem apenas dois níveis de cache, não apresentando cache de nível 3.

Memória principal Tempo de acesso de até centenas de ciclos, mas com capacidade para várias centenas de megabytes.

Disco Tempo de acesso de centenas de milhares de ciclos, porém com capacidade de dezenas ou centenas de gigabytes.

1.2.2 Caches

Os níveis da hierarquia de memória que estão entre o processador e a memória principal são denominados *memória cache*. A memória cache é um tipo de memória mais

rápido, que mantém um subconjunto da memória principal com os dados mais recentemente acessados, para acesso rápido caso sejam utilizados novamente [13].

Quando ocorre um acesso a memória a um endereço cujos dados estão armazenado na cache, o tempo de acesso aos dados é menor. Um acesso a dados que já estão disponíveis na cache é denominado *acerto* (*hit*). Quando ocorre um acesso a dados que não estão disponíveis na cache, os dados precisam ser trazidos do próximo nível da hierarquia de memória para a cache. Um acesso a dados que não estão disponíveis na cache é denominado *falta* (*miss*).

É comum haver vários níveis de cache, sendo que os níveis mais altos são compostos de memória mais rápida que os níveis mais baixos. Por exemplo: se houver 2 níveis de cache, uma falta na cache de nível 1 irá causar um acesso à cache de nível 2; e uma falta na cache de nível 2 irá causar um acesso à memória principal. Os computadores modernos comumente possuem 2 ou 3 níveis de cache.

Os dados são organizados na cache em *blocos*. Quando ocorre uma *falta*, todo o bloco é transferido para a cache. A organização em blocos aproveita a propriedade da *localidade espacial*: se um endereço dentro de determinado bloco foi acessado, é provável que endereços vizinhos, dentro do mesmo bloco, sejam acessados em breve.

Uma das decisões de projeto de uma cache diz respeito ao posicionamento dos blocos transferidos dentro da cache. Cada bloco da memória principal possui um conjunto de possíveis posições onde pode ser armazenado na cache. Esse conjunto geralmente é uma função do endereço do bloco. De acordo com a definição do conjunto, as caches podem ser classificadas como:

Mapeada diretamente (*direct mapped*) Cada bloco da memória principal possui apenas uma posição possível onde pode ser encontrado na memória cache.

Totalmente associativa (*fully-associative*) Cada bloco pode ser posicionado em qualquer posição na memória cache.

Associatividade N-ária (*N-way set associative*) Cada bloco pode ser posicionado em um conjunto de blocos de tamanho N.

A grande maioria das caches empregadas em processadores de uso geral são de mapeamento direto, ou de associatividade 2 ou 4 [4].

Quando há uma falta e um bloco precisa ser transferido do próximo nível da hierarquia para a cache, as posições da cache que podem armazenar os dados sendo

transferidos podem estar em uso, armazenando outros blocos. Nesse caso, um desses blocos deverá ser substituído pelo bloco que está sendo transferido. Para decidir qual bloco será substituído, é necessário uma *política de substituição de blocos*.

Em caches de mapeamento direto, só há uma posição possível para o bloco sendo transferido, logo basta substituir o bloco que esteja nessa posição, caso exista algum. Em caches N-associativas ou totalmente associativas, é necessário definir uma política para a substituição de blocos. Algumas políticas de substituição de blocos comuns são:

Aleatório (*Random*) O bloco a ser substituído é escolhido aleatoriamente.

LRU (*Least-recently used*) O bloco que foi acessado pela última vez há mais tempo é escolhido.

Pseudo-LRU Com associatividades mais altas, é muito caro manter um registro ordenado de todos os blocos acessados para utilizar o LRU. Existem algoritmos mais simples de implementar em *hardware* que se aproximam dos resultados do LRU [12].

FIFO (*First-In, First-Out*) O bloco que está há mais tempo na cache é substituído.

Outra variável a ser definida no projeto de uma cache é o como a cache trata escritas. São duas as opções para o tratamento de escrita [4]:

Escrita forçada (*write through*) Os dados são escritos ao mesmo tempo no bloco na cache e no próximo nível da hierarquia (no próximo nível de cache ou na memória principal).

Escrita preguiçosa (*write back*) Os dados são escritos apenas no bloco armazenado na cache, e são enviados para o próximo nível apenas quando este bloco na cache for substituído.

Há outras técnicas, não analisadas aqui, que podem ser utilizadas para o tratamento de escritas, cada uma com suas vantagens e desvantagens [6].

Durante o projeto de memórias cache, é necessário analisar qual será o comportamento e desempenho da cache. Há ferramentas que permitem que o comportamento da cache seja analisado antes de sua construção. Duas dessas ferramentas são os simuladores de circuitos e simuladores de cache, que são analisados nas próximas seções.

1.3 Simulação de circuitos

Entre as ferramentas que auxiliam no projeto de *hardware* estão os *simuladores de circuitos*, que permitem que o comportamento de circuitos seja simulado por software e possa ser analisado sem que o circuito seja construído fisicamente.

Há várias categorias de simuladores de circuitos. Os simuladores podem simular circuitos digitais ou analógicos. Os simuladores também podem se distinguir pela exatidão com que determinados componentes e características do circuitos são modelados.

Os simuladores comumente são acompanhados de bibliotecas de componentes reutilizáveis que facilitam a construção dos circuitos. Também pode ser fornecida uma interface gráfica para desenho dos circuitos e ferramentas que permitem acompanhar os estados dos sinais enquanto o circuito é simulado. Abaixo são listados alguns simuladores de circuitos existentes.

SPICE [9] [15] Simulador analógico de propósito geral. O modelo simulado é bastante detalhado, permitindo a simulação e análise de muitas características físicas do circuito que não são modeladas por simuladores de circuitos digitais.

DigLOG e AnaLOG [2] Simuladores de circuitos digitais e analógicos, respectivamente. Sua principal característica é a interface gráfica fornecida para desenho do diagrama esquemático dos circuitos.

TKGate [3] Simulador de circuitos digitais. Possui uma interface gráfica simples de usar, semelhante ao *DigLOG*, porém oferece algumas funcionalidades adicionais.

ArchC [11] ArchC é uma linguagem de descrição de arquitetura (*Architecture Description Language* — ADL). Oferece um nível mais alto de abstração que outras ferramentas. Permite que um simulador para o sistema modelado seja *interpretado* ou *compilado*.

1.3.1 TKGate

O simulador escolhido para a execução deste trabalho foi o TKGate. Os seguintes pontos foram analisados para a escolha do simulador:

Simulação de circuito digital Simulação de memória cache em nível lógico, não sendo necessária simulação analógica e de características físicas detalhadas do circuito.

Modelagem gráfica do circuito O resultado do trabalho poderá ser utilizado para fins didáticos por estudantes de circuitos digitais e arquitetura de computadores. Para esse fim, uma visualização gráfica do projeto do circuito é útil.

Extensibilidade Para a simulação da utilização da memória cache por um programa real, é necessário que o simulador possa ser utilizado de modo não interativo, que a entrada do circuito possa ser controlada por outro programa de acordo com as referências à memória do programa sendo simulado e que o resultado da simulação possa ser pós-processado.

Dentre os simuladores analisados, o TKGate mostrou-se o mais adequado aos propósitos deste trabalho.

1.3.1.1 Interface com o usuário do TKGate

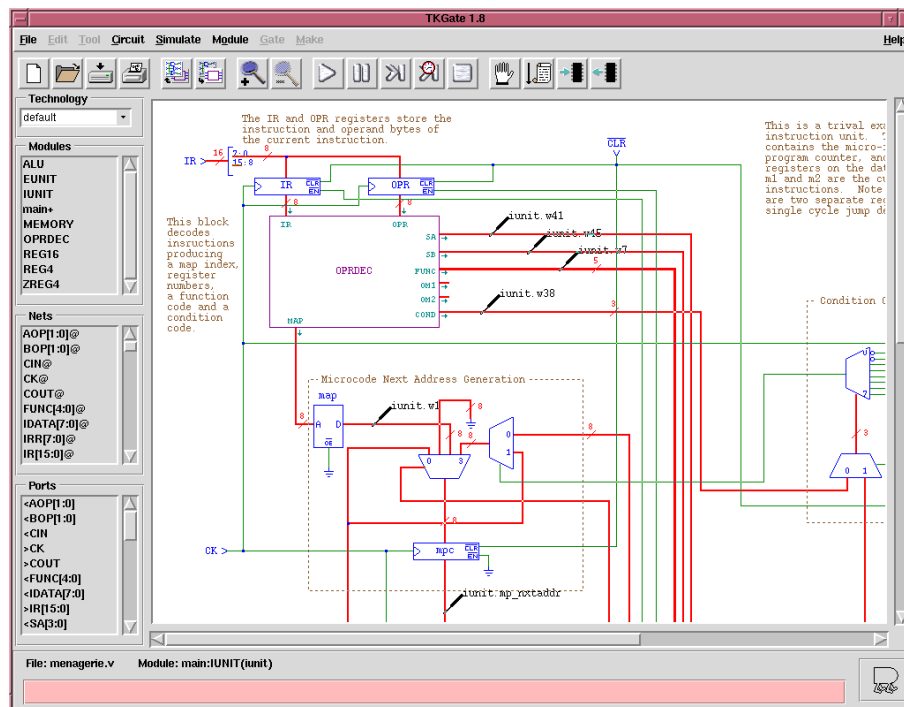


Figura 2: Interface com o usuário do TKGate

O TKGate possui uma interface gráfica para o desenho de circuitos, mostrada na Figura 2. Esta mesma interface pode ser utilizada para analisar o estado de sinais individuais do circuito durante a simulação.

Adicionalmente, o TKGate possui uma interface para análise do comportamento dos sinais ao longo do tempo, durante a execução da simulação, mostrada na Figura 3.

usuário troca mensagens com o simulador, indicando mudanças na entrada do circuito e solicitações de informações sobre sinais. O simulador envia mensagens indicando o estado dos sinais.

Essa arquitetura permite que o simulador seja utilizado independentemente da interface com o usuário. Deste modo, o simulador pode ser reutilizado para simulações não interativas de cache, como descrito na Seção 2.2.1.

A simulação de caches utilizando um simulador de circuitos como o TKGate possui diferenças em relação a métodos convencionais de simulação de caches. Essas diferenças são detalhadas na próxima seção.

1.4 Simulação de caches

Assim como no projeto de circuitos, simuladores ajudam a analisar o comportamento do circuito antes de sua construção, no projeto de memórias cache os simuladores podem ajudar a analisar o comportamento da memória cache.

Simuladores de memória cache permitem que o desempenho da memória cache sendo projetada possa ser medida e analisada, e projetos alternativos de caches possam ser testados e comparados.

Simuladores de cache podem executar a simulação a partir de *traçados* (*traces*) – registros armazenados previamente de todos os acessos à memória efetuados durante a execução de um programa – ou através da *execução de programas* ao mesmo tempo em que os acessos à memória são tratados pela simulação da cache.

Um simulador bastante utilizado é o *SimpleScalar* [1]. O SimpleScalar contém um simulador de uma arquitetura baseada em MIPS [8], e durante a simulação executa programas compilados para essa arquitetura. As informações sobre os acessos à memória para simulação da cache são geradas durante a simulação da execução do programa pelo processador. Ao final da simulação, o SimpleScalar emite estatísticas detalhadas sobre o comportamento da memória cache que foi simulada.

A ferramenta do pacote de simuladores do SimpleScalar que foi utilizada nesse trabalho é o *sim-cache*, que faz simulação simples de caches. Além da simulação de caches, o SimpleScalar contém outros simuladores e ferramentas, para análise de características como execução de instruções *fora de ordem* por processadores super-escalares [14], *profiling*, e tempo de execução.

Simulação de caches através de simulador de circuitos

Uma alternativa à simulação de caches através de programas como o SimpleScalar é a utilização de um *simulador de circuitos* para o projeto e simulação do comportamento da memória cache. Para isso, o circuito da memória cache é definido no nível de portas lógicas e células de memória, e esta definição é utilizada como entrada para o simulador de circuitos.

Uma diferença desse método com relação ao SimpleScalar é que o modelo simulado pode incorporar mais características da construção da memória cache, que não são modeladas pelo simulador do SimpleScalar. Isso ocorre porque a especificação e projeto da memória cache a ser simulada estão em um nível mais baixo de abstração.

Esse método possui vantagens e desvantagens. Com um modelo mais concreto, o projeto e definição da cache a ser simulada são mais complexos que a definição mais abstrata das caches utilizadas como entrada para simuladores como o SimpleScalar. Por exemplo, a alteração de certos parâmetros da cache — como tamanho dos blocos e associatividade — é mais complexa no projeto do circuito da cache que em um simulador de mais alto nível. Por outro lado, o projeto da cache em nível mais baixo permite que outras características possam ser analisadas, como por exemplo, o *caminho crítico*¹ do circuito da cache, e uma descrição mais detalhada da *máquina de estados* que define o comportamento da cache.

A simulação e análise do comportamento de caches utilizando simuladores de circuitos — permitindo que características de mais baixo nível do projeto das caches sejam definidas e analisadas — é o tema deste trabalho. Nos próximos capítulos, são descritos a implementação de circuitos de memória cache utilizando o TKGate como simulador de circuitos, soluções utilizadas e os resultados obtidos.

¹Mais detalhes sobre a análise do caminho crítico dos circuitos das caches podem ser encontrados na seção 2.1.5

2 Arquitetura do projeto

Neste capítulo são descritos os circuitos de cache projetados e implementados no TKGate, as características das caches simuladas, o sistema de simulação, e os resultados das simulações.

2.1 Implementação de caches no TKGate

As caches foram desenhadas como circuitos no TKGate, utilizando a interface de desenho de circuitos oferecida pelo próprio simulador. A Figura 5 mostra essa interface, com parte da lógica de acerto e associatividade do projeto de uma memória cache de associatividade 4.

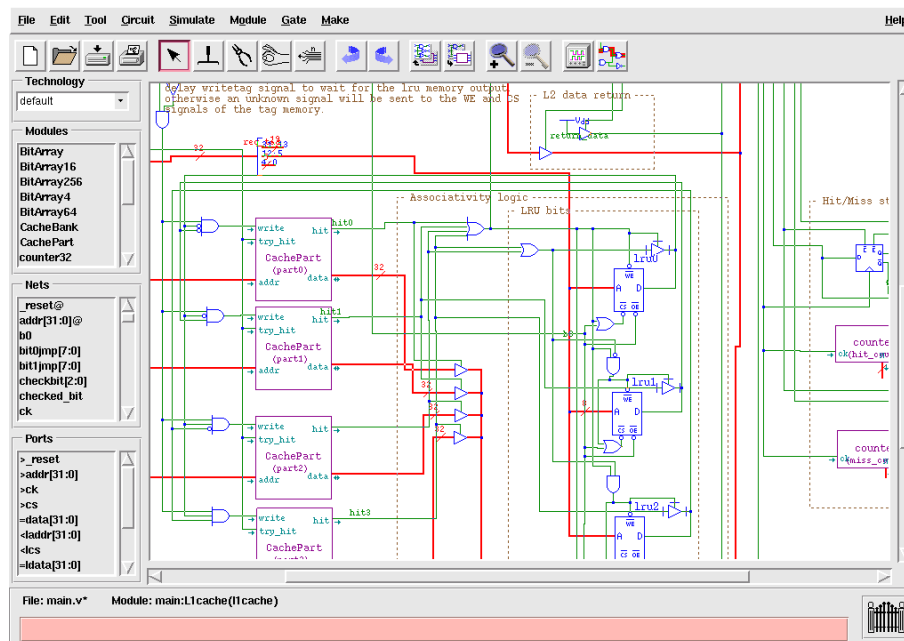


Figura 5: Interface de projeto de circuitos do TKGate com um projeto de memória cache

2.1.1 Caches implementadas

Foram implementados vários projetos de memória cache, com diferentes configurações. A Tabela 1 lista as caches implementadas, e os parâmetros utilizados no seus projetos. Cada projeto de cache possui um *nome* associado, para identificá-lo ao longo do texto. Todos os projetos de cache implementados utilizam *escrita forçada*.

Nome	Associatividade	Política de Substituição	Tamanho de bloco (bytes)	Tamanho da cache (bytes)
simples	1	-	32	8192
2-assoc-lru	2	LRU	32	16384
4-assoc-plru	4	Pseudo-LRU	32	32768

Tabela 1: Características dos projetos de cache implementados

2.1.2 Projeto do circuito

A Figura 6 ilustra o diagrama de blocos dos circuitos de cache que foram implementados, contendo apenas as principais linhas de comunicação no circuito. Diagramas com os circuitos detalhados das caches são encontrados no Apêndice A.

O componente de *Lógica de Acerto/Falta* é o responsável pela verificação de faltas e acertos na cache. Os *Blocos* são os blocos de memória da cache que armazenam os dados vindos dos níveis mais baixos da hierarquia de memória.

A máquina de estados controla os outros componentes do circuito e define o comportamento da memória cache. A máquina de estados ativa o circuito de verificação de acertos, define quando dados devem ser retornados dos blocos da cache, e quando solicitações de leitura ou escrita devem ser enviados para o próximo nível da hierarquia.

2.1.3 Máquina de estados da cache

O comportamento da máquina de estados é definido utilizando a ferramenta de geração de microcódigo do TKGate, *gmac*. O diagrama de estados é ilustrado na Figura 7.

No Apêndice B encontra-se uma listagem do código que define a máquina de estados, na linguagem da ferramenta *gmac*. A utilização da ferramenta de geração de microcódigo para definição do comportamento da cache torna o projeto do circuito mais flexível, permitindo que alguns pontos do comportamento da cache possam ser modificados, testados e corrigidos sem a modificação do circuito da cache.

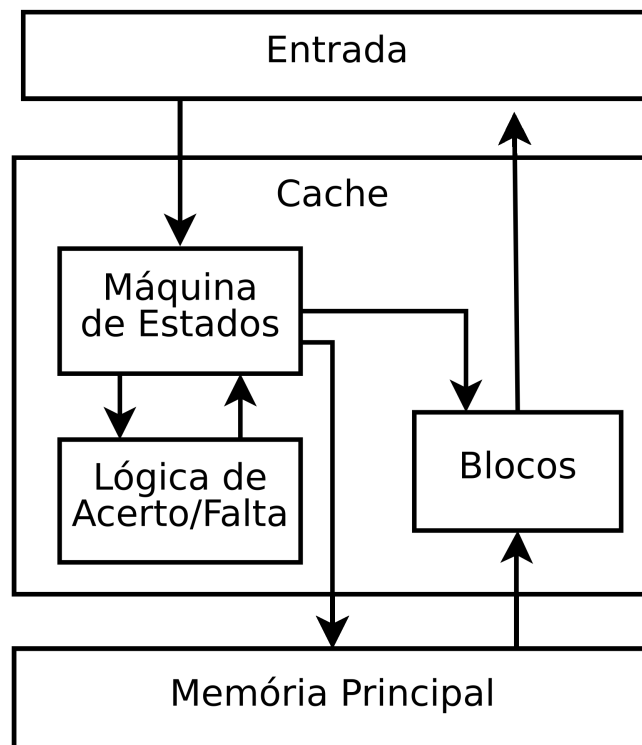


Figura 6: Visão geral do projeto do circuito de memória cache

Cada estado definido pela máquina de estado define as operações que serão feitas um ciclo do relógio. Os estados da cache são:

Aguardando / Verificando Acerto Principal e mais complexo estado da cache. A cache possui um único estado em que um acerto de leitura é tentado e os dados são retornados quando necessários. O cálculo de acerto e o retorno de dados ao processador em caso de acerto ocorrem em apenas um ciclo do relógio.

Aguardando leitura Em caso de uma falta de leitura da cache, é enviado um sinal de leitura para o próximo nível da hierarquia. A cache fica neste estado até que o próximo nível da hierarquia retorne os dados solicitados.

Retornando dados Os dados são retornados para o processador ou nível anterior da hierarquia. Os dados são retornados enquanto a linha *Chip Select* (CS) estiver ligada.

Escrevendo Os projetos de cache implementados utilizam *escrita forçada*, logo todas as escritas são enviadas para o próximo nível da hierarquia, havendo ou não um acerto. A cache fica nesse estado enquanto a escrita é enviada para o próximo nível da hierarquia.

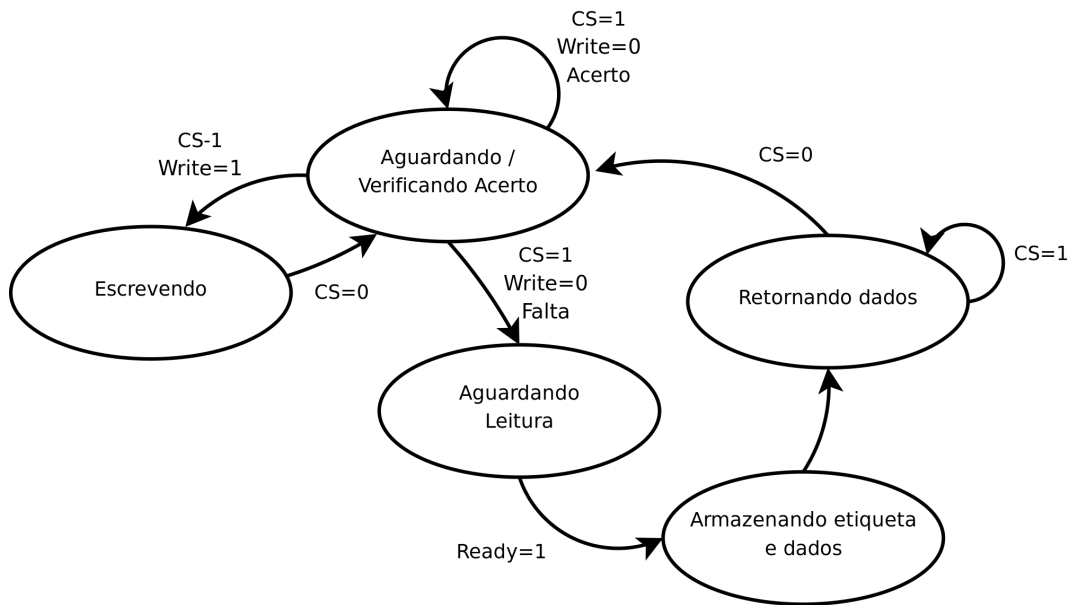


Figura 7: Máquina de estados das memórias cache implementadas

2.1.4 Reutilização de módulos

A ferramenta de projeto de circuitos do TKGate permite a construção de módulos reutilizáveis. Porém nem todos os parâmetros desses módulos podem ser redefinidos em apenas um ponto da definição do circuito. Parâmetros como o número de bits de uma entrada ou saída não podem ser redefinidos automaticamente. Por isso, há características que exigem mais esforço para modificação, como mencionado na Seção 1.4. Algumas dessas características são a técnica de escrita utilizada, tamanho dos blocos e a associatividade.

2.1.5 Caminho crítico

Um dos recursos oferecidos pelo TKGate é o cálculo do *caminho crítico* do circuito. O caminho crítico é a sequência de componentes do circuito em que os sinais levam mais tempo para se propagar de uma ponta a outra. Em geral, o caminho crítico do circuito impõe um limite inferior para o ciclo do relógio do circuito.

O cálculo de caminho crítico do TKGate mostra quais cadeias de componentes do circuito possuem o maior tempo de propagação. A Figura 8 mostra a interface de cálculo do caminho crítico do TKGate mostrando o caminho crítico da cache *simples*.

O cálculo do caminho crítico depende de valores realísticos de tempo de atraso para as portas lógicas e componentes do circuito. A biblioteca de componentes do TKGate

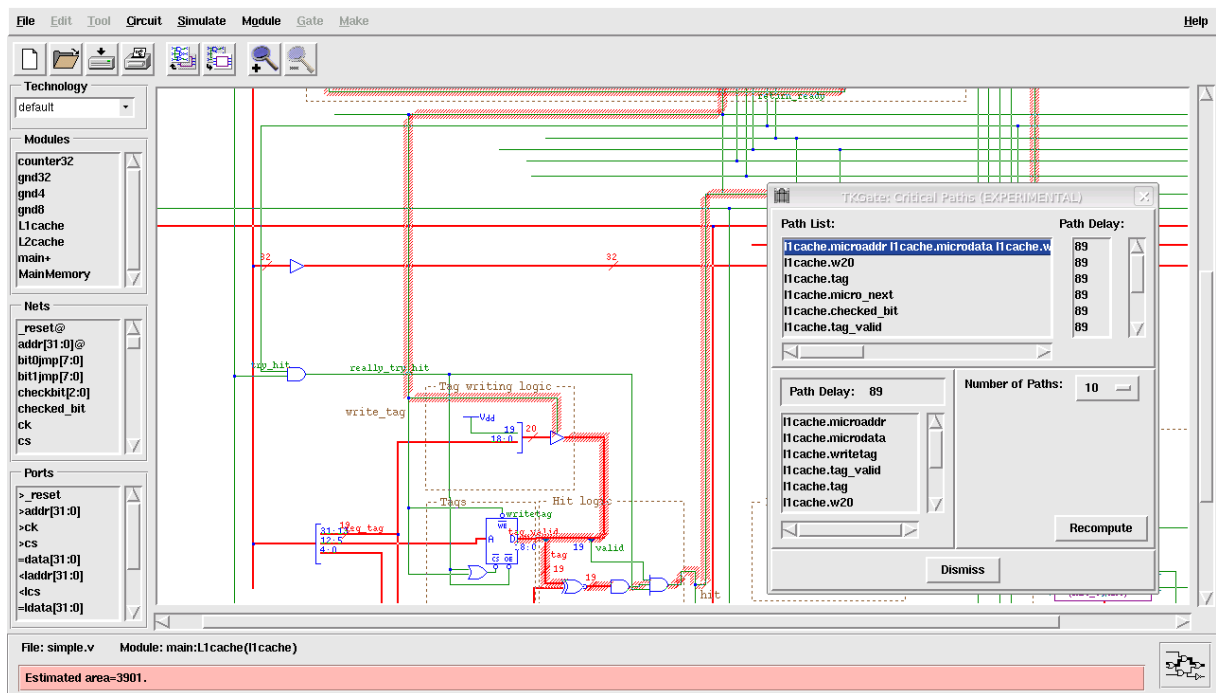


Figura 8: Caminho crítico do projeto de cache *simple*s

possui valores pré-configurados. Os valores de atraso de cada componente do circuito podem ser configurados para valores determinados da tecnologia de implementação caso seja necessário.

Em todos os projetos de cache implementados, o caminho crítico calculado pelo TKGate correspondeu ao cálculo mais custoso da cache, que é a verificação de acerto. A verificação de acerto da cache corresponde ao estado “*Aguardando / Verificando Acerto*” da máquina de estados.

Uma vez implementados os circuitos de memória cache, é necessário simulá-los utilizando dados de acesso a memória de um programa real. A próxima seção descreve as soluções utilizadas para realizar a simulação de operação das caches.

2.2 Simulação dos circuitos de caches

Para simular de forma realística a operação dos circuitos de cache é necessário utilizar padrões de acesso a memória de programas reais. Também é desejável validar a implementação do circuito, verificando se os resultados da simulação estão corretos.

São três os principais pontos e problemas a serem solucionados pela simulação:

1. Enviar dados sobre acessos a memória às entradas do circuito;
2. Obtenção de traçados de acesso à memória de programas reais; e
3. Validação dos resultados da simulação.

2.2.1 Alimentação das entradas do circuito simulado

Como descrito na Seção 1.3.1.2, a arquitetura do TKGate facilita a simulação de circuitos sem a utilização de sua interface de desenho de circuitos. O componente de simulação de circuitos em C — denominado *gsim* — pode ser utilizado diretamente para execução da simulação.

O simulador *gsim* espera comandos em sua entrada, para controle da simulação. Na simulação do circuito da cache, para cada acesso à cache é necessário modificar os sinais de entrada do circuito, avançar o relógio até o ponto em que os dados foram retornados, modificar os sinais de entrada novamente e reiniciar o processo. Para que seja seguido este protocolo para comunicação com a cache, foi desenvolvida uma ferramenta na linguagem Python que realiza estas tarefas, a partir de um traçado de execução de programa. Esta ferramenta foi denominada *trace-to-gsim*.

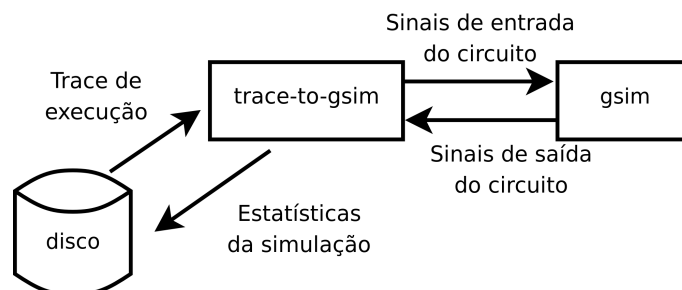


Figura 9: Uso da ferramenta *trace-to-gsim*

O *trace-to-gsim* pode ler os traçados do disco ou diretamente a partir da saída padrão de outro programa. A possibilidade de utilizar a saída de outro programa permite

que a execução do programa que gera os acessos à memória e a simulação da memória cache sejam executadas concomitantemente. Deste modo, tanto simulações a partir de *traçados* previamente armazenados quanto simulações através de *execução de programas* são possíveis.

Uma listagem do código fonte do *trace-to-gsim* pode ser encontrada no Apêndice C.

2.2.2 Obtenção de traçados de acessos à memória

Para uma simulação realista dos circuitos de memória cache é necessária a obtenção de traçados resultantes da execução de programas reais. Para isso, precisamos dispor previamente dos dados dos traçados, ou obter alguma ferramenta que gere as informações sobre acesso à memória enquanto simula a execução de programas reais.

A utilização de uma ferramenta que gere os traçados em tempo de execução apresenta a vantagem de que os traçados podem ser *consumidos* pelo simulador de caches à medida que são gerados, reduzindo drasticamente a capacidade de armazenamento necessária para a execução das simulações.

O *SimpleScalar* é uma ferramenta capaz simular a execução de programas e seus acessos a memória para simulação de caches. Porém, as informações sobre o acesso à memória são processadas internamente pelo simulador e não eram *exportadas* para o nosso simulador de circuitos.

Entretanto, como o *SimpleScalar* possui o código-fonte aberto, foi possível fazer modificações em seu código para que o simulador possa ser configurado para gerar um traçado da execução dos programas simulados.

Essa modificação adiciona um parâmetro adicional de configuração ao *sim-cache*: `-ctrace:d11`, que permite especificar o nome de um arquivo onde será escrito o traçado. O Apêndice D possui uma listagem com as modificações realizadas no código-fonte do *sim-cache* para adicionar essa funcionalidade.

Com essas modificações, a saída do *sim-cache* com o traçado dos acessos à memória pode ser utilizada como entrada para a ferramenta *trace-to-gsim*, através de um *pipe*. O sistema de simulação resultante é ilustrado na Figura 10.

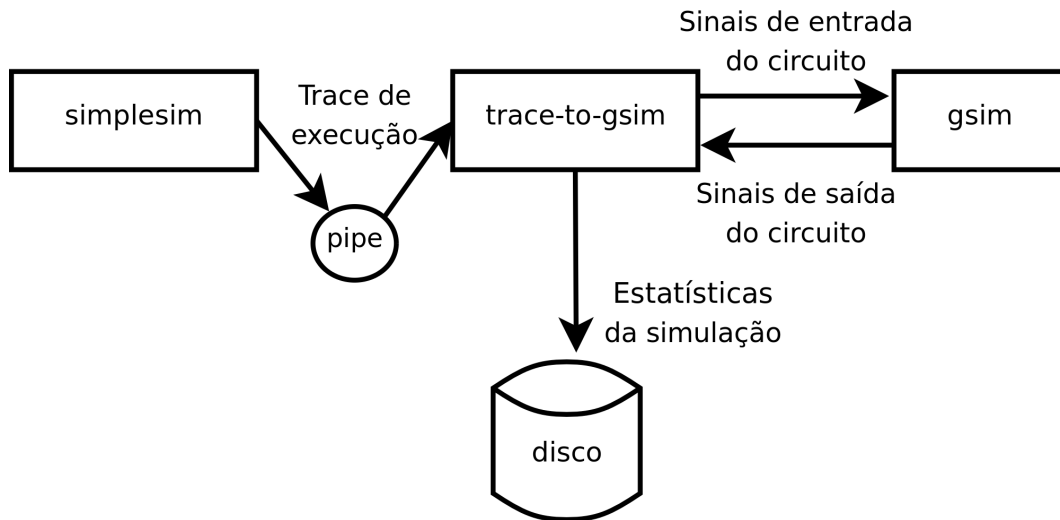


Figura 10: Sistema de simulação utilizando a saída de traçados do *sim-cache*

2.2.3 Execução de simulações longas no TKGate

Durante a execução de simulações dos circuitos no TKGate um defeito foi encontrado no *gsim*, o componente de simulação do TKGate. Após uma rodada de simulação longa, o simulador era abortado pelo sistema operacional devido a um acesso inválido à memória.

Após investigações a respeito do problema, a causa foi localizada. O simulador do TKGate utiliza números inteiros de 32 bits para representação de valores de medida de tempo da simulação¹. Porém em algumas das rodadas eram simulados mais de 200 milhões de acessos à memória. O caminho crítico de um dos circuitos de cache simulados possui um atraso de mais de 200 unidades de tempo. Nessas rodadas longas, as variáveis de 32 bits do simulador do TKGate não comportavam os valores de medidas de tempo resultantes, que passavam da ordem de dezenas de bilhões de unidades de tempo.

O código (em linguagem C) do TKGate possui um tipo de dados personalizado, denominado `simTime`, que era definido como equivalente ao tipo `int`. Isso deveria permitir que o tamanho das variáveis que armazenam medições de tempo seja escolhido em tempo de compilação modificando apenas um ponto do código, na declaração `typedef` correspondente ao tipo `simTime`. Havia pontos do código em que variáveis do tipo `int` eram utilizadas para representação de `simTime`. Por isso, mesmo após a modificação da declaração do tipo `simTime` para um tipo de dados maior, o problema persistia. Uma

¹O código utilizava o tipo `int` da linguagem C, que na plataforma para a qual o TKGate foi compilado possui 32 bits

análise de todo o código do TKGate foi realizada e todos os pontos do código em que uma medida de tempo era representada passaram a utilizar o tipo de dados `simTime`.

Após as modificações, o tipo de dados foi modificado para inteiros de 64 bits². Com as correções, todas as rodadas de simulação foram concluídas com sucesso. Os resultados das simulações são apresentados no Capítulo 3.

O Apêndice D possui uma listagem das modificações realizadas no código do simulador para a correção do problema.

2.2.4 Validação dos resultados

De posse de um sistema para simulação e obtenção de traçados, é possível rodar simulações completas de programas. Entretanto é desejável haver um modo de validar esses resultados, para verificar se a implementação dos circuitos e a alimentação de suas entradas está correta.

O sistema de simulação SimpleScalar já possui um componente que, além de gerar os traçados de acessos à memória do programa simulado, é capaz de simular memórias cache, o *sim-cache*. Com isso, nosso sistema de simulação existente já possui um mecanismo para comparação de duas implementações de caches com os mesmos parâmetros. É necessário apenas utilizar a capacidade de simulação de caches do *sim-cache*, os resultados da cache simulada no TKGate podem ser comparados com os resultados do simulador de caches do *sim-cache* a partir da mesma execução do programa. O sistema de simulação incluindo a validação dos resultados é ilustrado na Figura 11.

2.2.4.1 Simulação de escrita forçada no SimpleScalar

Os circuitos de cache implementados utilizam a técnica de *escrita forçada* para o tratamento de escritas. Porém o simulador *sim-cache* não é capaz de simular caches com essa técnica de escrita.

Para permitir que os resultados dos circuitos de cache simulados sejam comparados com os resultados do *sim-cache*, foi modificado o código-fonte do *sim-cache* e implementado o suporte a escrita forçada no simulador. O Apêndice D mostra as modificações realizadas para implementação dessa funcionalidade.

²Foi utilizado o tipo *long long*, que na plataforma para a qual o TKGate foi compilado, possui 64 bits

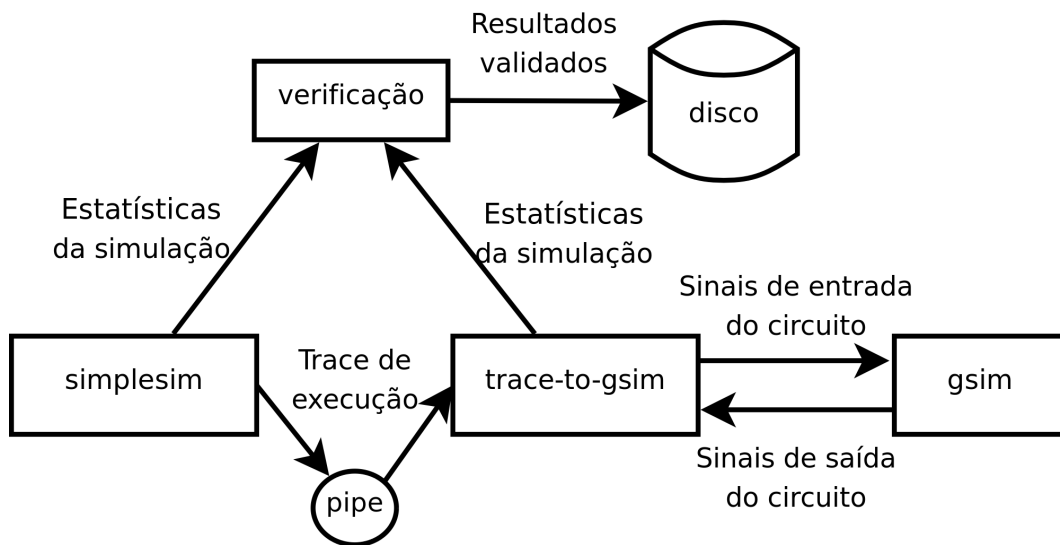


Figura 11: Sistema de simulação com validação dos resultados

2.2.4.2 Validação dos resultados da cache com pseudo-LRU

Uma das caches implementadas, identificada como *4-assoc-plru* na Tabela 1, utiliza a política de substituição *pseudo-LRU*, que não é originalmente suportada pelos simuladores do *SimpleScalar*. Sem um simulador capaz de simular a política de substituição utilizada pelo circuito, não seria possível validar os resultados da simulação do circuito.

Para resolver esse problema, foi implementado no simulador de caches do *SimpleScalar* o suporte ao mesmo algoritmo de pseudo-LRU utilizado pelo circuito *4-assoc-plru*. O Apêndice D contém uma listagem das modificações feitas no código do *SimpleScalar*.

Com o sistema de simulação montado e as modificações realizadas no código dos simuladores TKGate e sim-cache, foi possível realizar várias rodadas de simulações com os circuitos de cache, e os resultados puderam ser analisados e comparados com os resultados do simulador de caches sim-cache. No próximo capítulo são apresentados e analisados os resultados das simulações.

3 Análise dos resultados

Nos capítulos anteriores foi apresentado o sistema de simulação de cache através do simulador de circuitos TKGate e da validação dos resultados da simulação. Neste capítulo são apresentados e analisados os resultados das simulações executadas.

As seguintes características dos resultados são analisadas: (i) taxas de faltas e acertos; (ii) tempo de execução das simulações; e (iii) comparação dos resultados *sim-cache* e TKGate.

3.1 Simulações executadas

Para a simulação das caches utilizamos os binários pré-compilados do conjunto de *benchmarks* SPEC95 [10], que acompanham o *SimpleScalar*. Cada um dos circuitos de cache listados na Tabela 1 foi simulado com os traçados de acessos à memória dos seguintes programas do SPEC95 [10]:

129.compress Compressão de arquivos grandes de texto (em torno de 16MB).

130.li Interpretador Lisp.

134.perl Interpretador Perl.

102.swim Solução de *equações de água rasa*, aplicadas em previsão do tempo.

3.2 Resultados

Nas simulações foram medidos as taxas de acerto e tempo de simulação. Estes são detalhados a seguir.

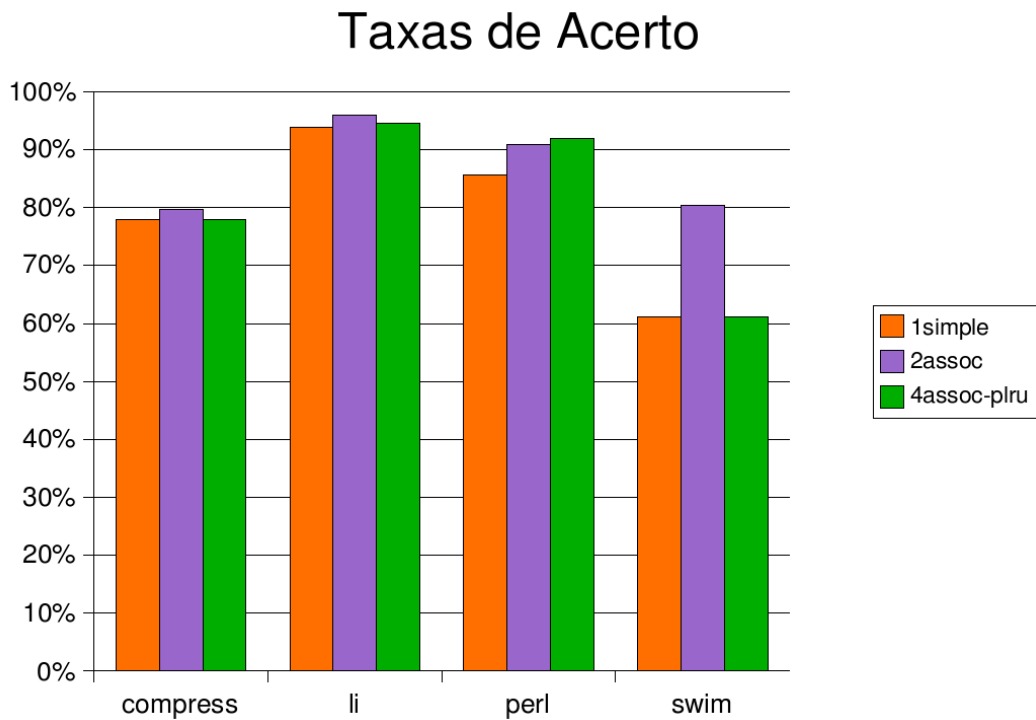


Figura 12: Taxas de acerto das caches para cada programa

3.2.1 Taxas de acerto

A Figura 12 ilustra as taxas de acerto de cada cache simulada através do simulador de circuitos TKGate. As taxas de acerto resultantes são as mesmas apresentadas pelo simulador *sim-cache* durante as simulações.

3.2.2 Tempo de simulação

As informações sobre o tempo de execução de cada simulação são apresentadas na Tabela 2, porém as simulações foram realizadas em períodos diferentes, em sistemas com configurações de *hardware* diferentes. Por isso, não é possível fazer uma comparação entre o tempo necessário para execução da simulação de diferentes circuitos de cache. O programa "perl" apresenta um tempo de simulação muito menor do que a média porque o código e os dados utilizados como entrada para o programa simulado eram relativamente pequenos.

A Tabela 3 apresenta a média de acessos a memória simulados por segundo de cada simulação. Algumas diferenças entre simulações podem ocorrer devido a diferenças na configuração de *hardware* utilizada em cada execução. A Figura 13 apresenta uma

	compress	li	perl	swim
simples	2688,3	17436,3	29,5	53518,3
2assoc	2626,8	16057,4	29,7	52166,5
4assoc-plru	2689,2	13317,2	26,1	43024,5

Tabela 2: Tempo total, em segundos, de execução de cada simulação

comparação dos resultados.

	compress	li	perl	swim
simples	5032,98	4465,08	4920,21	4931,28
2assoc	5150,77	4848,51	4886,89	5059,07
4assoc-plru	5031,35	5846,17	5558,44	6134,03

Tabela 3: Acessos a memória simulados por segundo, de cada simulação

3.2.3 Comparação do tempo de simulação entre *SimpleScalar* e TKGate

As medições do tempo de simulação apresentadas na seção anterior incluem o tempo total de simulação, incluindo o uso de CPU do simulador *sim-cache*, da ferramenta *trace-to-gsim* e do simulador de circuitos do TKGate. Algumas simulações foram executadas com medições do tempo de CPU utilizado pelo simulador *sim-cache* separadamente, para possibilitar uma comparação entre o desempenho do simulador *sim-cache* e da simulação baseada em circuitos. Os resultados são apresentados na Tabela 4.

programa	cache	Tempo de simulação sim-cache (segundos)	Tempo de simulação TKGate (segundos)
compress	simples	8,15	2407
li	simples	44,69	13804

Tabela 4: Comparação entre o tempo de simulação do *sim-cache* e do circuito de caches no TKGate

A partir da amostra apresentada Tabela 4, é possível estimar que a simulação das caches através de simulador de circuitos leva aproximadamente 300 vezes mais tempo para ser executada. Isso é esperado, já que o nível de detalhe é muito maior na simulação no nível de portas lógicas.

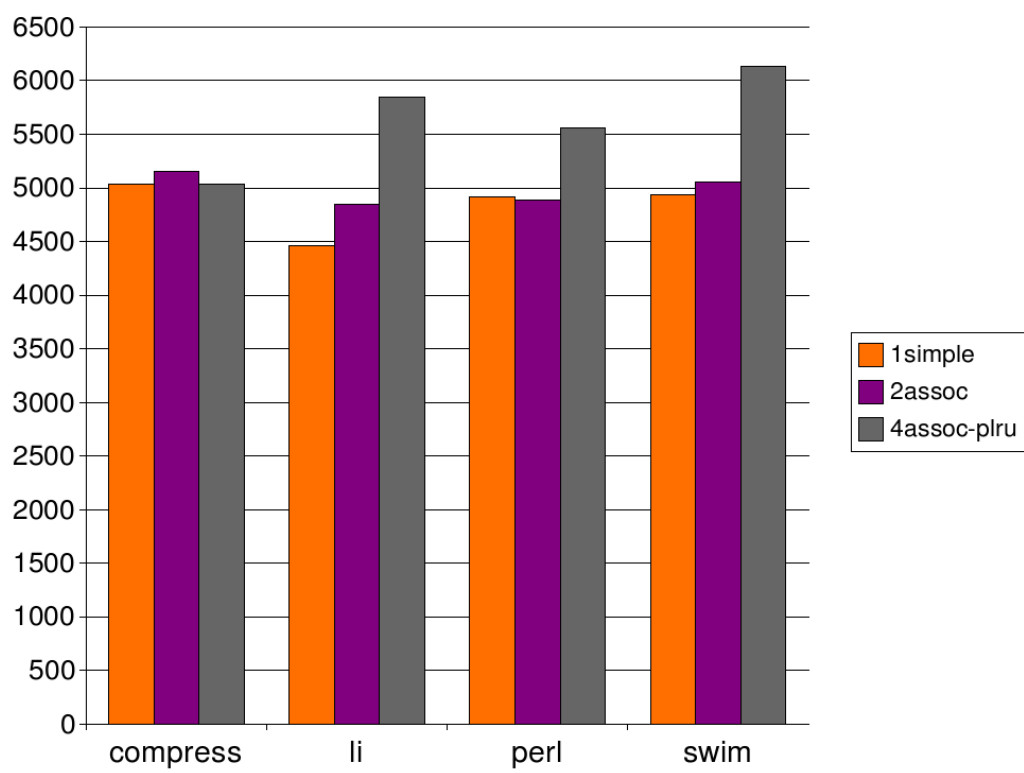


Figura 13: Acessos a memória simulados por segundo, de cada simulação

4 Conclusão

Resultados do trabalho

A simulação de caches utilizando um simulador de circuitos mostrou-se uma alternativa possível como ferramenta didática e de análise de projetos de cache. Uma simulação em nível baixo como o nível de portas lógicas exige mais recursos computacionais para a execução da simulação e mais esforço para a modificação dos projetos, porém o modelo simulado pode ser mais detalhado e apresentar características mais próximas de um projeto de cache real.

A ferramenta de projeto e simulação de circuitos TKGate também mostrou-se adequada para projetos de circuitos complexos. A possibilidade de utilização do componente de simulação separadamente do componente de projeto de circuitos também pode tornar o TKGate uma ferramenta flexível para utilização em projetos, simulação e análise de outros tipos de circuitos, além de memórias cache.

Os circuitos desenhados apresentaram resultados satisfatórios após as simulações, apresentando resultados que foram validados após comparados com os resultados de outro simulador de cache, o *sim-cache*.

Subprodutos do trabalho

Além do resultado principal, que foram os circuitos de cache e resultados obtidos de simulações destes circuitos, o trabalho realizado para permitir a simulação de caches trouxe subprodutos úteis não apenas para a simulação de memórias cache:

Sistema de alimentação de dados para o TKGate O sistema de alimentação de dados para o TKGate pode ser estendido para execução de simulações de outros circuitos desenhados com o TKGate.

Correções de problemas no código do TKGate Correções foram realizadas no có-

digo de simulação do TKGate, que são úteis para outros usuários do simulador.

Funcionalidades adicionais ao SimpleScalar O suporte a escrita forçada e a Pseudo-LRU foi adicionado ao código de simulação de caches do SimpleScalar.

Melhorias futuras

São três as melhorias futuras, ferramentas alternativas para simulação, sistema simulação mais eficiente, e extensões aos circuitos de cache implementados. Estas são descritas no que se segue.

Ferramentas alternativas para simulação

Uma ferramenta para projeto de circuitos que permita abstrações mais avançadas e que facilitem a personalização de parâmetros do circuito permitiria o uso mais simples dos projetos de cache para comparação de caches com diferentes parâmetros. Os circuitos implementados no TKGate apresentam a desvantagem de não permitir uma modificação simples e automática de alguns parâmetros do circuito, como tamanho de blocos ou associatividade.

É possível uma análise de outras ferramentas e linguagens de definição de arquitetura para verificar se outra ferramenta pode ser adequada para esse propósito. Algumas ferramentas podem permitir um nível de abstração mais alto e possivelmente permitem que o projeto implementado seja mais flexível que o implementado utilizando o TKGate. O *ArchC* [11] é um exemplo de ferramenta de simulação e especificação de arquiteturas com abstrações de nível mais alto, porém é necessária uma melhor análise para verificar se o *ArchC* também pode ser uma ferramenta adequada para especificação e simulação de circuitos de caches [16]. Mesmo que o resultado seja negativo, outras ferramentas podem ser analisadas ou até mesmo desenvolvidas para isso.

Outra possível alternativa é a implementação de melhorias no próprio TKGate para reduzir o tempo necessário para personalização de parâmetros do projeto de circuitos. A implementação de algumas abstrações de mais alto nível podem ajudar no projeto de circuitos que possam ser personalizados mais facilmente.

Sistema de simulação mais eficiente

Como a simulação de caches com o simulador de circuitos do TKGate mostrou-se muito mais lenta que a simulação realizada com o simulador *sim-cache*, há a possibilidade de otimizá-lo para que a simulação de circuitos possa apresentar melhor desempenho. Abstrações de mais alto nível para especificação dos circuitos podem facilitar a implementação dessas otimizações, trazendo vantagens tanto no esforço necessário para modificação dos projetos de circuitos quanto no desempenho da simulação.

Durante as simulações, o uso de CPU da ferramenta *trace-to-gsim* correspondeu a aproximadamente 32% do uso total de CPU do sistema de simulação. A utilização de uma linguagem compilada para o *trace-to-gsim* pode melhorar muito o desempenho de todo o sistema de simulação.

Extensões aos circuitos de cache implementados

Algumas características ainda podem ser adicionadas aos circuitos de cache implementados, para torná-los mais úteis como ferramenta didática, e para permitir análise de outras características de memória cache.

Outros tipos de caches Outros projetos de caches podem ser implementados, incluindo características como: *caches não bloqueantes* [7], *victim cache* [5], métodos alternativos para transferência dos blocos do próximo nível da hierarquia [4], *stream buffer* [5].

Modelagem mais fiel do tempo de resposta dos componentes A precisão do cálculo do caminho crítico do circuito, mencionado na Seção 2.1.5 depende de uma modelagem precisa dos tempos de atraso de todos os componentes do circuito. Uma modelagem precisa também permite uma melhor análise do desempenho do projeto de cache. O TKGate apresenta valores predefinidos para os tempos de atraso de cada componente de sua biblioteca de componentes e portas lógicas. Neste trabalho não foram analisados os valores predefinidos do simulador. Analisar esses valores e possivelmente modificá-los para serem mais fiéis à realidade é outro ponto onde pode haver trabalho posterior para melhorias no projeto.

Referências

- [1] D. Burger and T. Austin. The SimpleScalar tool set, version 2.0. *ACM SIGARCH Computer Architecture News*, 25(3):13–25, 1997.
- [2] D. Gillespie and J. Lazzaro. DigLOG and AnaLOG, Caltech VLSI CAD Tools. *California Institute*.
- [3] J. P. Hansen. TKGate Home Page. <<http://www.tkgate.org>>. Acessado em 29/06/2006.
- [4] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach (The Morgan Kaufmann Series in Computer Architecture and Design)*. Morgan Kaufmann, 2002.
- [5] N. P. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Proc 17th Intl Symp on Computer Arch*, pages 364–373. ACM Comp Arch News 18(2), ACM Press, 1990.
- [6] N. P. Jouppi. Cache write policies and performance. In *Proc 20th Intl Symp on Computer Arch*, pages 191–201. ACM Comp Arch News 21(2), May 1993.
- [7] D. Kroft. Lock-up free instruction fetch/prefetch cache organization. In *Proc 8th Intl Symp on Computer Arch*, pages 81–85, June 1981.
- [8] C. Price. MIPS IV Instruction Set, revision 3.1. MIPS Technologies. *Inc., Mountain View, California, January, 1995*.
- [9] J. M. Rabaey. The Spice Page. <<http://bwrc.eecs.berkeley.edu/classes/icbook/spice/>>. acessado em 29/06/2006.
- [10] J. Reilly. SPEC describes SPEC95 products and benchmarks. *SPEC Newsletter*, 7(3):4–6, 1995.
- [11] S. Rigo, G. Araujo, M. Bartholomeu, and R. Azevedo. ArchC: A SystemC-Based Architecture Description Language. *Computer Architecture and High Performance Computing, 2004. SBAC-PAD 2004. 16th Symposium on*, pages 66–73, 2004.
- [12] T. Shanley. *Pentium Pro Processor System Architecture*. Addison-Wesley Professional, 1996.
- [13] A. J. Smith. Cache memories. *ACM Computing Surveys*, 14(3):473–530, Sept. 1982.
- [14] G. Sohi. Instruction issue logic for high-performance, interruptible, multiple functional unit, pipelined computers. *Computers, IEEE Transactions on*, 39(3):349–359, 1990.

- [15] P. Tuinenga. *Spice: A Guide to Circuit Simulation and Analysis Using PSpice*. Prentice Hall PTR Upper Saddle River, NJ, USA, 1991.
- [16] P. Viana, E. Barros, S. Rigo, R. Azevedo, and G. Araujo. Modeling and simulating memory hierarchies in a platform-based design methodology. *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, 1:734–735, 2004.

APÊNDICE A – Diagramas dos circuitos de cache

As próximas páginas contêm os diagramas dos circuitos de cache desenhados. Os diagramas foram gerados pela função de impressão de diagramas do TKGate.

Cache *simples*

Cache *2assoc*

Cache *4assoc-plru*

APÊNDICE B – Microcódigo dos circuitos de cache

A seguinte listagem contém o microcódigo utilizado para a máquina de estados dos circuitos de cache, na linguagem da ferramenta *gmac*.

```

microcode bank[31:0] llcache.code;

// Caso waitbit != none,
// aguarda o bit antes de ir para o próximo estado
field waitbit[2:0] = {
    none = 0,
    cs = 1,
    nocs = 2,
    lready = 3,
    hit = 4,
    cs_and_miss = 5
};

// bit a ser testado para decidir
// entre estados diferentes
field checkbit[5:3] = {
    none = 0,
    we = 1,
    hit = 2
};

// Próximo estado
field next[15:8];

// Estados para onde pular ao verificar bit
field bit0jmp[15:8], bit1jmp[23:16];

field writetag[24], lwe[25], lcs[26]; // ready[27];

// Sinal com mais efeitos colaterais:
// indica que estamos no estado em que
// é tentado um hit. Se houver um hit,
// next, bit0jmp, bit1jmp e checkbit
// são ignorados
field try_hit[28];

// Passa para ready o valor de lready
field return_ready[27];

// Passa para data o valor de ldata
field return_data[29];

begin microcode @ 0

// Estado em que, em um ciclo do clock,
// verifica por hit, e retorna os dados
idle_try_hit:
    try_hit
    waitbit=cs_and_miss
    checkbit=we

```

```
    bit0jmp=read_miss
    bit1jmp=write_miss;

// Escrita forçada
write_miss:
    lwe
    lcs
    return_data
    return_ready
    waitbit=nocs
    next=idle_try_hit;

// Leitura
read_miss:
    lcs
    waitbit=lready
    next=writetag;

// Escreve tag e retorna
// palavra para processador
writetag:
    lcs
    writetag
    return_data
    return_ready
    next=keep_returning;

keep_returning:
    lcs
    return_data
    return_ready
    waitbit=nocs
    next=idle_try_hit;

end
```

APÊNDICE C – Código fonte da ferramenta trace-to-gsim

A seguinte listagem contém o código fonte da ferramenta *trace-to-gsim*, utilizada nas simulações, na linguagem Python.

```
#!/usr/bin/python

import sys, re, time, os, thread

term=open("/dev/tty", 'r')

WATCHES_DICT = {
    'simple':[
        'ready', 'llcache.microaddr', 'llcache.hit',
        'llcache.microdata', 'llcache.ck', 'llcache.waited_bit',
        'llcache.waitbit', 'llcache.miss_and_cs',
        'llcache.tag_valid',
        'llcache.valid',
        'llcache.tag_addr',
        'llcache.tag_equal',
        'llcache.valid',
        'llcache.really_try_hit',
        'llcache.tag_cs',
        'llcache.writetag',
    ],
    '2-assoc':[
        'ck',
        'ready', 'llcache.microaddr', 'llcache.hit',
        'llcache.cs', 'llcache.we',
        'llcache.hit0', 'llcache.hit1',
        'llcache.microdata', 'llcache.ck', 'llcache.waited_bit',
        'llcache.micro_next',
        'llcache.bitljmp', 'llcache.bit0jmp',
        'llcache.waitbit', 'llcache.miss_and_cs',
        'llcache.addr',
        'llcache.lru',
        'llcache.lru_cs',
        'llcache.lru_addr',
        'llcache.hit_id',
        'llcache.write0',
        'llcache.write1',
        'llcache.part0.tag_valid_towrite',
        'llcache.part1.tag_valid_towrite',
        'llcache.part0.try_hit',
        'llcache.part0.try_hit_delayed',
        'llcache.part0.try_hit_safe',
        'llcache.part1.try_hit',
        'llcache.part1.try_hit_delayed',
        'llcache.part1.try_hit_safe',
        'llcache.part0.write',
    ],
}
```

```

'llcache.part1.write',
'llcache.part0.tag-addr',
'llcache.part1.tag-addr',
'llcache.part0.tag-cs',
'llcache.part1.tag-cs',
'llcache.part0.write',
'llcache.part1.write',
'llcache.part0.tag-addr',
'llcache.part1.tag-addr',
'llcache.part0.tag-valid',
'llcache.part1.tag-valid',
'llcache.part0.valid',
'llcache.part1.valid',
'llcache.part0.tag-equal',
'llcache.part1.tag-equal',
'llcache.part0.valid',
'llcache.part1.valid',
'llcache.part0.hit',
'llcache.part1.hit',
'llcache.really_try_hit',
'llcache.part0.tag-cs',
'llcache.part1.tag-cs',
'llcache.writetag',
'llcache.writetag-delayed',
'llcache.writetag-safe',
],
'4-assoc':[
'ck',
'ready','llcache.microaddr','llcache.hit',
'llcache.cs', 'llcache.we',
'llcache.microdata','llcache.ck','llcache.waited_bit',
'llcache.micro_next',
'llcache.bitljmp', 'llcache.bit0jmp',
'llcache.waitbit','llcache.miss_and_cs',
'llcache.addr',
'llcache.lru0','llcache.lru1','llcache.lru2',
'llcache.lru_cs',
'llcache.lru_addr',
'llcache.really_try_hit',
'llcache.writetag',
'llcache.writetag-delayed',
'llcache.writetag-safe',
] + sum([
[s % (i) for i in [0,1,2,3] ] for s in
[
'llcache.hit%d',
'llcache.write%d',
'llcache.part%d.tag_valid_towrite',
'llcache.part%d.try_hit',
'llcache.part%d.try_hit_delayed',
'llcache.part%d.try_hit_safe',
'llcache.part%d.write',
'llcache.part%d.tag_addr',
'llcache.part%d.tag_cs',
'llcache.part%d.write',
'llcache.part%d.tag_addr',
'llcache.part%d.tag_valid',
'llcache.part%d.valid',
'llcache.part%d.tag_equal',
'llcache.part%d.valid',
'llcache.part%d.hit',
'llcache.part%d.tag_cs',
]
], [])
) # sum( [ [a,b], [c,d] ], []) -> [a,b,c,d]
}

WATCHES = []

ENABLE_WATCHES = 0
DEBUG_GSIM=0
DEBUG_CMDS=0

```

```

class SimError(Exception):
    def __init__(self, msg):
        Exception.__init__(self, msg)

class FatalError(Exception):
    def __init__(self, type, msg):
        self.type = type
        self.msg = msg

    def __str__(self):
        return 'Fatal_error.Type: %s Msg: %s' % (self.type, self.msg)

class SimStop(Exception):
    def __init__(self, s):
        Exception.__init__(self, s)

class NetValue:
    verilog_re = re.compile("([0-9]*)'(.){0,31}([0-9A-Za-z]*)")

    def __init__(self, width=0, value=0):
        self.width = width
        self.value = value

    def parseVerilog(s):
        m = NetValue.verilog_re.match(s)
        if not m:
            raise Exception("Invalid_verilog_constant: %s" % (s))
        width, radix, value = m.groups()

        if radix == 'h':
            base = 16
        elif radix == 'b':
            base = 2
        else:
            raise Exception("Invalid_radix_for_verilog_constant: %s" % (c))

        try:
            value = int(value, base)
        except:
            raise Exception("Invalid_value_in_verilog_constant: %s" % (s))

        return NetValue(width, value)

    parseVerilog = staticmethod(parseVerilog)

    def __str__(self):
        return "%d'h%x" % (self.width, self.value)

class GSimController:
    def __init__(self, gsim, gdf, file, basedir):
        i, o = os.popen2("%s %s -D %s -B %s" % (gsim, file, gdf, basedir))
        self.gsim_in, self.gsim_out = i, o
        #thread.start_new_thread(self.show_all_output, ())
        self.watches = []

    def setWatches(l):
        self.watches = l

    def show_all_output(self):
        while 1:
            l = self.gsim_out.readline()
            if l is None: break
            print l,

    def handleNet(self, parts):
        net, value, at, time = parts[1].split('_', 3)
        print "Value_of_%s_@_%s: %s" % (net, time, value)

    def handleValueof(self, parts):
        net, valustr = parts[1].split('_', 1)
        netvalue = NetValue.parseVerilog(valustr)

```

```

    print "Value_of_net %s: %d_(0x%x)" % (net, netvalue.value, netvalue.value)

def handleSimError(self, parts):
    raise SimError(parts[1])

def invalidCmd(self, parts):
    pass

def handleStop(self, parts):
    raise SimStop(parts[1])

def handleError(self, parts):
    type,msg = parts[1].split('_',1)
    raise FatalError(type, msg)

cmdHandlers = {
    'net':handleNet,
    'valueof':handleValueof,
    'simerror':handleSimError,
    'error':handleError,
    'stop':handleStop,
}

def handleLine(self, l):
    l = l.rstrip('\n')
    if DEBUG_GSIM:
        print "<" , l
    parts = l.split('_', 1)
    cmd = parts[0]

    handler = GSimController.cmdHandlers.get(cmd, GSimController.invalidCmd)
    return handler(self, parts)

def get_cmd_output(self):
    try:
        while 1:
            l = self.gsim_out.readline()
            if not l: break

            self.handleLine(l)
    except SimStop:
        # SimStop means that we should return
        pass

def genCmd(self, s):
    if DEBUG_CMDS:
        print ">" , s
        #l = term.readline()
        #if l.startswith('end'): sys.exit()
        #if l.startswith(':'): s=l[1:]
    self.gsim_in.write("%s\n" % (s))
    self.gsim_in.flush()
    self.get_cmd_output()

def initialize(self):
    # get initial messages
    self.get_cmd_output()
    self.genCmd("memload_llcode.mem")
    self.genCmd("memload_zerotags.mem")
    self.genCmd("set_cs_0")
    self.genCmd("set_we_0")
    self.genCmd("set_reset_0")
    self.genCmd("clock_+_2_10")
    self.genCmd("set_reset_1")
    self.genCmd("clock_+_2_10")
    if ENABLE_WATCHES:
        for w in self.watches:
            self.genCmd("watch_%s_1" % (w))

def show_counters(self):
    self.genCmd("show_llcache.hit_count")
    self.genCmd("show_llcache.miss_count")

```

```

def generate_access(self, cmd, addr, size):
    if cmd == 'r':
        we = 0
    elif cmd == 'w':
        we = 1
    else:
        raise "Invalid cmd: %s" % (cmd)
    self.gencmd("set_addr_%s" % (NetValue(32, addr)))
    self.gencmd("set_we_%d" % (we))
    self.gencmd("set_cs_1")
    #self.gencmd("break 10 ready==1'b1")
    #self.gencmd("go 1")
    #self.gencmd("go 0")
    #self.gencmd("delete_break 10")
    self.gencmd("clock_+_5_10")
    self.gencmd("set_cs_0")
    #self.gencmd("break 10 ready!=1'b1")
    #self.gencmd("go 1")
    #self.gencmd("go 0")
    #self.gencmd("delete_break 10")
    self.gencmd("clock_+_5_10")

def main(argv):
    trace_re = re.compile(".*^\[[^\]]*\]\s*(\w)\s*(\w+)\s*(\w+)\$")

    circuit_name = argv[1]
    cache_name = argv[2]

    ctl = GSimController(argv[3], argv[4], argv[5], argv[6])
    ctl.watches = WATCHES_DICT.get(circuit_name, [])

    ctl.initialize()

    while 1:
        l = sys.stdin.readline()
        if not l: break

        l = l.strip()

        if l.startswith('#'):
            continue

        m = trace_re.match(l)
        if not m:
            sys.stderr.write("Invalid trace line: %s\n" % (l))
            continue

        (cache, cmd, addr, size) = m.groups()
        if cache != cache_name:
            continue

        addr = int(addr, 0)
        size = int(size, 0)

        ctl.generate_access(cmd, addr, size)
        ctl.show_counters()

if __name__ == '__main__':
    main(sys.argv)

```

APÊNDICE D – Modificações realizadas em programas

As modificações realizadas nos programas utilizados no trabalho são apresentadas neste Apêndice. As modificações estão no formato utilizado pela ferramenta *patch* para descrição de modificações realizadas em código ou arquivo texto.

Correção para simulações longas no TKGate

Modificações foram realizadas no código fonte do TKGate para permitir simulações longas. Como 28 arquivos diferentes no código são modificados de modo semelhante, trocando a utilização do tipo `int` por `simTime`, apenas as principais modificações estão incluídas nas páginas seguintes.

```
From nobody Mon Sep 17 00:00:00 2001
From: Eduardo Pereira Habkost <ehabkost@raisama.net>
Date: Tue, 30 May 2006 08:40:56 -0300
Subject: [PATCH] simTime/simDelay implementation
```

```
This adds two types to be used when using time values: simTime
and simDelay. These are implemented as 'long_long' to fix the 'time
resetting' bug.
```

```
Signed-off-by: Eduardo Pereira Habkost <ehabkost@raisama.net>
```

```
---
```

```
src/gsim/add.c      | 2 +
src/gsim/arshift.c | 2 +
src/gsim/bufif.c   | 2 +
src/gsim/clock.c   | 42 ++++++-----
src/gsim/concat.c  | 26 ++++++-----
src/gsim/cpath.c   | 67 ++++++-----
src/gsim/demux.c   | 2 +
src/gsim/flipflop.c | 16 +++++-----
src/gsim/gateinfo.h | 12 +++++-----
src/gsim/generic.c | 48 ++++++-----
src/gsim/lshift.c  | 2 +
src/gsim/module.c  | 4 +-
src/gsim/module.h  | 14 +++++-----
src/gsim/mux.c     | 2 +
src/gsim/nmos.c    | 2 +
src/gsim/pmos.c    | 2 +
```

```

src/gsim/process.c | 2 +
src/gsim/ram.c | 34 ++++++-----
src/gsim/register.c | 26 ++++++-----
src/gsim/roll.c | 2 +
src/gsim/rom.c | 18 ++++++-----
src/gsim/rshift.c | 2 +
src/gsim/supply.c | 2 +
src/gsim/tap.c | 20 ++++++-----
src/gsim/thyme.c | 70 ++++++-----
src/gsim/thyme.h | 18 ++++++-----
src/gsim/tty.c | 6 ++-
src/gsim/verilog.c | 6 ++-
28 files changed, 231 insertions(+), 220 deletions(-)

07f9ccbda4e5046b75cbb174fad6a82b19617597
diff --git a/src/gsim/process.c b/src/gsim/process.c
index 4e9cd3b..61721ac 100644
--- a/src/gsim/process.c
+++ b/src/gsim/process.c
@@ -64,7 +64,7 @@ void SNet_process(SNet *N, EvQueue *Q, SEv
    p = buf;
    p += sprintf(p, "net_%s_", nname);
    p += SState_getstr(&N->n_state, p);
-   p += sprintf(p, "_@_%d", E->evnet.time);
+   p += sprintf(p, "_@_" SIMTIME_FMT, E->evnet.time.v);
    sendMsg("%s", buf);
}
}

diff --git a/src/gsim/thyme.c b/src/gsim/thyme.c
index 33c69fe..7c91695 100644
--- a/src/gsim/thyme.c
+++ b/src/gsim/thyme.c
@@ -682,7 +682,7 @@ static void EvQueue_execWatch(EvQueue *Q
    p = buf;
    p += sprintf(p, "net_%s_", net);
    p += SState_getstr(&N->n_state, p);
-   p += sprintf(p, "_@_%d", Q->curStep);
+   p += sprintf(p, "_@_" SIMTIME_FMT, Q->curStep.v);
    sendMsg("%s", buf);
}

@@ -795,7 +795,7 @@ void EvQueue_execute(EvQueue *Q, char *co
} else if (sscanf(command, "_wai%c", buf) == 1 && *buf == 't') {
    EvQueue_execWait(Q);
} else if (sscanf(command, "_tim%c", buf) == 1 && *buf == 'e') {
-   sendMsg("stop_@_%d_(time)", Q->curStep);
+   sendMsg("stop_@_" SIMTIME_FMT "__(time)", Q->curStep.v);
} else if (sscanf(command, "_clock_%c_%d_%d_%s",&c,&n,&d, buf2) == 4) {
    if ((Q->flags & EVF_HASCLOCK)) {
        Q->triggerClock = SModule_findGate(Q->mod, buf2);
@@ -948,7 +948,7 @@ void EvQueue_mainEventLoop(EvQueue *Q)
}

    if (!(Q->flags & EVF_RUN)) {
-   sendMsg("stop_@_%d_(empty)", Q->curStep);
+   sendMsg("stop_@_" SIMTIME_FMT "__(empty)", Q->curStep.v);
    input_ready(1);
    if (!get_line(buf, STRMAX)) return;
    EvQueue_execute(Q, buf);
diff --git a/src/gsim/thyme.h b/src/gsim/thyme.h
index e9a01bf..d88dc10 100644
--- a/src/gsim/thyme.h
+++ b/src/gsim/thyme.h
@@ -73,7 +73,14 @@ #define getEvNet(E) (((E)->evclass == E
    */
    #define IsChangeOn(E, g, n) (getEvNet(E) == (g)->g_ports.port[(n)]->p_net)

-typedef int simTime; /* Simulation time */
+
+typedef struct { long long v; } simTime; /* Simulation time */
+static inline simTime mkSimTime(long long x) { simTime r; r.v = x; return r; }
+#define SIMTIME_FMT "%lld"
+

```

```

+typedef struct { int v; } simDelay;          /* Gate delay */
+static inline simDelay mkSimDelay(int x) { simDelay r; r.v = x; return r; }
+#define SIMDELAY_FMT "%d"

/*
    Logic values
@@ -174,7 +181,8 @@ struct evqueue {
    int      clockCount;          /* Number of clock cycles to step */
    SGate    *triggerClock;       /* Clock to trigger on (null if any clock) */

-   int      curStep;             /* Current time step */
+   /*FIXME: check if it should be 'int' or simTime */
+   simTime  curStep;             /* Current time step */
    int      numPending;          /* Number of pending events */
    SEvent   *wheel_head[THYMEWHEEL_SIZE]; /* Event queues for each step (for dequeue) */
    SEvent   *wheel_tail[THYMEWHEEL_SIZE]; /* Event queues for each step (for enqueue) */
--
1.3.3

```

Suporte a escrita forçada no SimpleScalar

```

From e0753c4fa6a47e5d9079e401dcd26c6e42d689d3 Mon Sep 17 00:00:00 2001
From: Eduardo Habkost <ehabkost@raisama.net>
Date: Sat, 25 Feb 2006 01:09:20 -0300
Subject: [PATCH] Added trace and directwrite support

```

```

Signed-off-by: Eduardo Habkost <ehabkost@raisama.net>

```

```

cache.c      | 17 ++++++
cache.h      | 10 +++++
sim-cache.c  | 33 ++++++
textprof.pl |  2 +-
4 files changed, 60 insertions(+), 2 deletions(-)

```

```

diff --git a/cache.c b/cache.c
index f9036c6..a35149d 100644
--- a/cache.c
+++ b/cache.c

```

```

@@ -337,6 +337,7 @@ cache_create(char *name,          /* name of the
cp->misses = 0;
cp->replacements = 0;
cp->writebacks = 0;
+ cp->directwrites = 0;
cp->invalidations = 0;

/* blow away the last block accessed */
@@ -456,6 +457,9 @@ cache_reg_stats(struct cache_t *cp, /* c
printf(buf, "%s.writebacks", name);
stat_reg_counter(sdb, buf, "total number of writebacks",
                &cp->writebacks, 0, NULL);
+ printf(buf, "%s.directwrites", name);
+ stat_reg_counter(sdb, buf, "total number of direct writes",
+                &cp->directwrites, 0, NULL);
printf(buf, "%s.invalidations", name);
stat_reg_counter(sdb, buf, "total number of invalidations",
                &cp->invalidations, 0, NULL);
@@ -517,6 +521,10 @@ cache_access(struct cache_t *cp, /* cach
if (repl_addr)
    *repl_addr = 0;

+ if (cp->tracefile)
+   /*FIXME: print value of 'now' */
+   fprintf(cp->tracefile, "[%s] %c_0x%11x_%d\n", cp->name, cmd == Read?'r':cmd == Write?'w':'?', (
+       long long)addr, nbytes);
+
/* check alignments */

```


Suporte a Pseudo-LRU no SimpleScalar

As seguintes modificações foram realizadas no código do SimpleScalar para permitir a utilização de Pseudo-LRU.

```
From 1afbddd3f3fdbf822ff71a88506abf50e1c80ae07 Mon Sep 17 00:00:00 2001
From: Eduardo Pereira Habkost <ehabkost@raisama.net>
Date: Mon, 13 Mar 2006 22:05:46 -0300
Subject: [PATCH] Starting pseudo-lru implementation
```

```
- Adding PSEUDO_LRU to enum cache_policy
- Implement pseudo_lru_blk(): return block chosen for replacement by
  pseudo-lru algorithm
- Handle PSEUDO_LRU policy on cache block replacement: use
  pseudo_lru_blk() function
```

```
Signed-off-by: Eduardo Pereira Habkost <ehabkost@raisama.net>
```

```
---
cache.c | 61 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
cache.h | 4 +++
2 files changed, 64 insertions(+), 1 deletions(-)
```

```
diff --git a/cache.c b/cache.c
index a35149d..611b7f4 100644
--- a/cache.c
+++ b/cache.c
@@ -410,6 +410,7 @@ cache_char2policy(char c)          /* replacemen
     case 'l': return LRU;
     case 'r': return Random;
     case 'f': return FIFO;
+   case 'p': return PSEUDO_LRU;
     default: fatal("bogus replacement policy, '%c'", c);
   }
}
@@ -428,6 +429,7 @@ cache_config(struct cache_t *cp,      /* cach
    cp->policy == LRU ? "LRU"
    : cp->policy == Random ? "Random"
    : cp->policy == FIFO ? "FIFO"
+   : cp->policy == PSEUDO_LRU ? "Pseudo-LRU"
    : (abort(), "));
}

@@ -495,6 +497,62 @@ cache_stats(struct cache_t *cp,      /* cach
    (double)cp->invalidations/sum);
}

+
+/* Get block to be replaced using the Pseudo-LRU algorithm */
+struct cache_blk_t *
+pseudo_lru_blk(struct cache_t *cp, struct cache_set_t *set)
+{
+   int bindex, bit, i;
+
+   /*TODO: Implement pseudo-lru for other assoc. values */
+
+   /* number of bit comparisons made for pseudo-lru */
+   #define PSEUDO_LRU_CHECKS 2
+   /* first (MSB) bit whose value is found on pseudo-lru calculation */
+   #define PSEUDO_LRU_FIRST_BIT 2
+   assert(cp->assoc == 4);
+
+   /* first, search for an invalid entry */
+   for (bindex = 0; bindex < cp->assoc; bindex++) {
+       struct cache_blk_t *blk = CACHE_BINDEXT(cp, set->blks, bindex);
+       if ( !(blk->status & CACHE_BLK_VALID) )
+           return blk;
+   }
}
```

```

+
+ /* from <http://www.cs.clemson.edu/~mark/464/p-lru.txt>:
+     are all 4 lines valid?
+           /           \
+         yes         no, use an invalid line
+           |
+           |
+           |
+         bit_0 == 0?
+           /           \
+         y             n
+           /           \
+       bit_1 == 0?   bit_2 == 0?
+         / \         / \
+       y  n       y  n
+   line_0 line_1 line_2 line_3
+
+           state | replace           ref to | next state
+           -----+-----           -----+-----
+           /           \
+         00x | line_0           line_0 | 11_
+         01x | line_1           line_1 | 10_
+         1x0 | line_2           line_2 | 0_1
+         1x1 | line_3           line_3 | 0_0
+
+           ('x' means ('_' means unchanged)
+           don't care)
+
+ */
+
+ bindex = 0;
+ bit = 0;
+ for (i = 0; i < PSEUDO_LRU_CHECKS; i++) {
+     if (set->pseudo_lru_info & (1<<bit))
+         bindex |= (PSEUDO_LRU_FIRST_BIT >> i);
+ }
+
+ return CACHE_BINDEX(cp, set->blks, bindex);
+}
+
+void pseudo_lru_register_access(struct cache_set_t *set, int blk)
+{
+    /*TODO: implement me */
+    /*TODO: call this function on cache access */
+}
+
+/* access a cache, perform a CMD operation on cache CP at address ADDR,
+   places NBYTES of data at *P, returns latency of operation if initiated
+   at NOW, places pointer to block user data in *UDATA, *P is untouched if
+@@ -592,6 +650,9 @@ cache_access(struct cache_t *cp, /* cach
+     repl = cp->sets[set].way_tail;
+     update_way_list(&cp->sets[set], repl, Head);
+     break;
+
+ case PSEUDO_LRU:
+     repl = pseudo_lru_blk(cp, &cp->sets[set]);
+     break;
+
+ case Random:
+     {
+         int bindex = myrand() & (cp->assoc - 1);
+
+@@ -102,7 +102,8 @@ #define CACHE_HIGHLY_ASSOC(cp) ((cp)->as
+     enum cache_policy {
+         LRU, /* replace least recently used block (perfect LRU) */
+         Random, /* replace a random block */
+         FIFO, /* replace the oldest block in the set */
+         FIFO, /* replace the oldest block in the set */
+         PSEUDO_LRU, /* Use pseudo-LRU algorithm */
+     };
+
+ /* cache write handling */
+@@ -148,6 +149,7 @@ struct cache_set_t
+     struct cache_blk_t *blks; /* cache blocks, allocated sequentially, so
+                               this pointer can also be used for random
+                               access to cache blocks */
+
+     int pseudo_lru_info; /* bits for pseudo-lru algorithm */
+ };
+
+ /* cache definition */
+
+ 1.4.1.rc2.g9cc3d

```

From 60b39d0a314776f35b0985f0c487dff2549c81cd Mon Sep 17 00:00:00 2001
 From: Eduardo Pereira Habkost <ehabkost@raisama.net>
 Date: Wed, 15 Mar 2006 20:36:16 -0300
 Subject: [PATCH] Fix pseudo_lru_blk() implementation, implement pseudo_lru_register_access()

- pseudo_lru_blk() 'bit' variable was never being changed. Fix this.
 - Implement pseudo_lru_register_access(), that update the pseudo_lru_info field

Signed-off-by: Eduardo Pereira Habkost <ehabkost@raisama.net>

```
---
cache.c | 28 ++++++
1 files changed, 24 insertions(+), 4 deletions(-)
```

```
diff --git a/cache.c b/cache.c
index 611b7f4..2ae2f07 100644
--- a/cache.c
+++ b/cache.c
@@ -540,17 +540,37 @@ #define PSEUDO_LRU_FIRST_BIT 2
     bindex = 0;
     bit = 0;
     for (i = 0; i < PSEUDO_LRU_CHECKS; i++) {
-        if (set->pseudo_lru_info & (1<<bit))
+        int goright = 0;
+        if (set->pseudo_lru_info & (1<<bit)) {
             bindex |= (PSEUDO_LRU_FIRST_BIT >> i);
             goright = 1;
         }
+        /* next level: left: bit = bit*2 + 1; right: bit = bit*2 + 2; */
+        bit = bit*2 + 1 + goright;
     }

     return CACHE_BINDEXT(cp, set->blks, bindex);
 }

-void pseudo_lru_register_access(struct cache_set_t *set, int blk)
+void pseudo_lru_register_access(struct cache_t *cp, struct cache_set_t *set, int bindex)
 {
-    /*TODO: implement me */
-    /*TODO: call this function on cache access */
+    /*TODO: call this function on cache access */
     int i, bit, lru_info;
     assert(cp->assoc == 4);

     lru_info = set->pseudo_lru_info;

     bit = 0;
     for (i = 0; i < PSEUDO_LRU_CHECKS; i++) {
         int goright = 0;
         if (bindex & (PSEUDO_LRU_FIRST_BIT >> i)) {
             lru_info &= ~(1<<bit);
             goright = 1;
         } else {
             lru_info |= (1<<bit);
         }
         bit = bit*2 + 1 + goright;
     }
 }

/* access a cache, perform a CMD operation on cache CP at address ADDR,
```

1.4.1.rc2.g9cc3d

From 744c88fbf87efccd4971c1a0671032832c489395 Mon Sep 17 00:00:00 2001
 From: Eduardo Pereira Habkost <ehabkost@raisama.net>
 Date: Wed, 15 Mar 2006 20:52:24 -0300
 Subject: [PATCH] Call pseudo_lru_register_access() on cache access

Signed-off-by: Eduardo Pereira Habkost <ehabkost@raisama.net>

```
---
cache.c | 13 ++++++
1 files changed, 13 insertions(+), 0 deletions(-)
```

```

diff --git a/cache.c b/cache.c
index 2ae2f07..0421471 100644
--- a/cache.c
+++ b/cache.c
@@ -772,6 +772,19 @@ cache_access(struct cache_t *cp, /* cach
    /* move this block to head of the way (MRU) list */
    update_way_list(&cp->sets[set], blk, Head);
}
+ else if (cp->policy == PSEUDO_LRU) {
+ /* look for the bindex of the block */
+ /*FIXME: this is *ugly*, use a more efficient way to know the block index */
+ int bindex;
+ for (bindex = 0; bindex < cp->assoc; bindex++) {
+ struct cache_blk_t *b = CACHE_BINDEX(cp, cp->sets[set].blks, bindex);
+ if (b == blk)
+ break;
+ }
+ if (bindex >= cp->assoc) /*BUG!*/
+ abort();
+ pseudo_lru_register_access(cp, &cp->sets[set], bindex);
+ }

    /* tag is unchanged, so hash links (if they exist) are still valid */

--
1.4.1.rc2.g9cc3d

From 43f740e532d3da7479fffe1ef2f53e30e8b0ba0c Mon Sep 17 00:00:00 2001
From: Eduardo Pereira Habkost <ehabkost@raisama.net>
Date: Wed, 15 Mar 2006 20:55:19 -0300
Subject: [PATCH] Call pseudo_lru_register_access() on miss, also

- change pseudo_lru_blk() to return block index
- call pseudo_lru_register_access() after replacing a block

Signed-off-by: Eduardo Pereira Habkost <ehabkost@raisama.net>
---
cache.c | 12 ++++++-----
1 files changed, 8 insertions(+), 4 deletions(-)

diff --git a/cache.c b/cache.c
index 0421471..e549a4b 100644
--- a/cache.c
+++ b/cache.c
@@ -499,7 +499,7 @@ cache_stats(struct cache_t *cp, /* cach

    /* Get block to be replaced using the Pseudo-LRU algorithm */
-struct cache_blk_t *
+int
pseudo_lru_blk(struct cache_t *cp, struct cache_set_t *set)
{
    int bindex, bit, i;
@@ -516,7 +516,7 @@ #define PSEUDO_LRU_FIRST_BIT 2
    for (bindex = 0; bindex < cp->assoc; bindex++) {
        struct cache_blk_t *blk = CACHE_BINDEX(cp, set->blks, bindex);
        if ( !(blk->status & CACHE_BLK_VALID) )
-            return blk;
+            return bindex;
    }

    /* from <http://www.cs.clemson.edu/~mark/464/p-lru.txt>:
@@ -549,7 +549,7 @@ #define PSEUDO_LRU_FIRST_BIT 2
        bit = bit*2 + 1 + goright;
    }

-    return CACHE_BINDEX(cp, set->blks, bindex);
+    return bindex;
}

void pseudo_lru_register_access(struct cache_t *cp, struct cache_set_t *set, int bindex)
@@ -671,7 +671,11 @@ cache_access(struct cache_t *cp, /* cach
    update_way_list(&cp->sets[set], repl, Head);

```

```
        break;
    case PSEUDO_LRU:
-       repl = pseudo_lru_blk(cp, &cp->sets[set]);
+       {
+         int bindex = pseudo_lru_blk(cp, &cp->sets[set]);
+         repl = CACHE_BINDEXT(cp, cp->sets[set].blks, bindex);
+         pseudo_lru_register_access(cp, &cp->sets[set], bindex);
+       }
        break;
    case Random:
        {
-
1.4.1.rc2.g9cc3d
```