Rafael Westphal

Modelos Eficientes de Hierarquia de Memória para Sistemas Embarcados

Florianópolis - SC, Brasil

Dezembro de 2009

Rafael Westphal

Modelos Eficientes de Hierarquia de Memória para Sistemas Embarcados

Monografia apresentada para obtenção do Grau de Bacharel em Ciência da Computação pela Universidade Federal de Santa Catarina.

Orientador:
Luiz Cláudio Villar dos Santos

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CENTRO TECNOLÓGICO
UNIVERSIDADE FEDERAL DE SANTA CATARINA

Florianópolis - SC, Brasil

Dezembro de 2009

Sumário

Monografia de Projeto Final de Graduação sob o título *"Modelos Eficientes de Hierarquia de Memória para Sistemas Embarcados"*, defendida por Rafael Westphal e aprovada em Dezembro de 2009, em Florianópolis, Estado de Santa Catarina, pela banca examinadora constituída pelos professores:

Prof. Dr. Luiz Cláudio V. Santos
Orientador

Prof. Dr. José Luís A. Güntzel
Universidade Federal de Santa Catarina

Prof. Dr. Olinto José Varela Furtado
Universidade Federal de Santa Catarina

Resumo

A hierarquia de memória é o gargalo tanto em energia quanto em performance em um sistema. Grande parte dos softwares não considera a diversidade de opções de hieraquia de memória em que poderia estar rodando, consumindo menos energia e até levando menos tempo. Poder simular o comportamento dessas hierarquias e estimar seu consumo de potência, pode contribuir para o desenvolvimento de um hardware mais eficiente. Este trabalho propõe um simulador de hierarquia de memória que disponibiliza mais opções, podendo refletir melhor o caso real. Uma precisão de até 98% foi encontrado nos experimentos realizando, conseguindo manter um tempo de execução aceitável, comparado com a carga computacional realizada.

Abstract

Memory hierarchy is the bottle-neck of energy and performance efficience in Embedded Systems. Most softwares doesnt take in account the diverse possibilities of memory hierarchy they could run, saving energy and even running faster. Being able to simulate those hierarchy behavior and estimate the power efficience can contribute with the develpment of the hardware. This work proposes a memory hierarchy simulator that brings more options closer to the developer, that way being reflect the real case. An accuracy of 98% was found on the executed experiments, being able to mainten an acceptable run-time, compared to the computacional load executed.

Agradecimentos

Agradeço a todos aqueles que ajudaram na realização deste trabalho.

Ao CNPq pelo suporte através de bolsa de iniciação científica.

Ao professores Dr. Luiz Cláudio V. Santos e Dr. José Luís A. Güntzel pelo suporte durante todo desenvolvimento do trabalho.

Aos Laborátorio de Automação de Projetos de Sistemas e o Núcleo Interdepartamental de Microeletrônica que me forneceram a infraestrutura necessária para realizar este trabalho.

A todos meu companheiros do laboratório que me ajudaram de alguma forma, especialmente para Alexandre Mendonca e Daniel Volpato.

Lista de Figuras

Lista de Tabelas

1 *O Papel da hierarquia de memória nos sistemas embarcados*

1.1 Os elementos de um subsistema de memória embarcada

Um subsistema de memória embarcada pode ser composto pelos seguintes componentes:

- Memória principal.
- Memória auxiliar.
- Memória de rascunho.

A seguir cada um dos componentes é discutido.

1.1.1 Memória principal

Muitas aplicações envolvem um grande volume de dados, sendo necessária uma forma de armazená-los, além das memórias internas ao sistema. Uma memória dinâmica de acesso aleatório (DRAM), geralmente externa ao sistema (*off-chip*), é usada para suprir essa necessidade [Panda, Dutt e Nicolau 1998].

O tempo de acesso à memória pode ser divido em três componentes. O tempo de decodificação da linha T_l, tempo de decodificação da coluna T_c e o tempo de pré-carga T_p, conforme mostra a Figura 1.1 [Panda, Dutt e Nicolau 1998]. O tempo de decodificação da linha é o tempo gasto para que uma linha da memória seja selecionada. O tempo de decodificação da coluna é o tempo gasto para selecionar uma palavra da linha. Já o tempo de pré-carga é o tempo gasto para deixar as linhas de bit prontas para a próxima operação da memória.

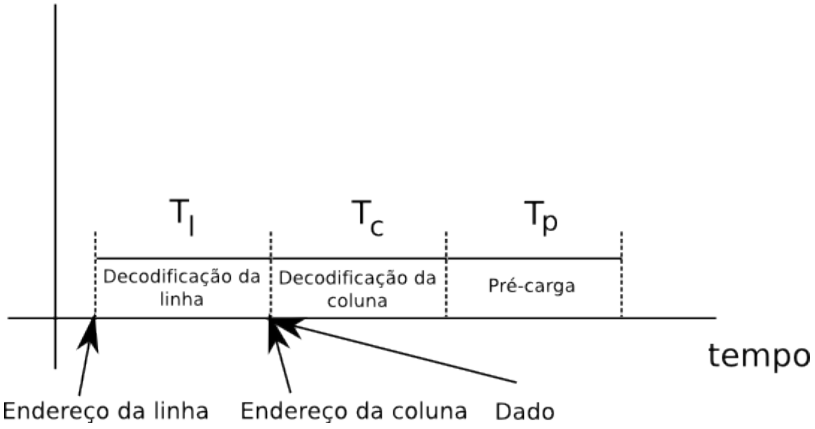

Figura 1.1: Modelo para uma operação de memória principal.

Na tecnologia DRAM o valor de tensão que representa um bit é armazenado na capacitância do gate de um transistor. Porém, por causa da corrente de fuga esse valor se deteriora. Para prevenir a degradação do valor é necessário periodicamente recarregar o conteúdo do bit.

A memória DRAM precisa de decodificadores de linha e coluna, buffer para armazenamento de dados e um arranjo de células da tecnologia DRAM. Existem células DRAM de três, dois e um transistores, sendo que a de um transistor é a mais utilizada no projeto comercial de memórias[Rabaey 2006]. A Figura 1.2 mostra como é uma organização de uma memória DRAM típica e seus componentes [Panda, Dutt e Nicolau 1998].

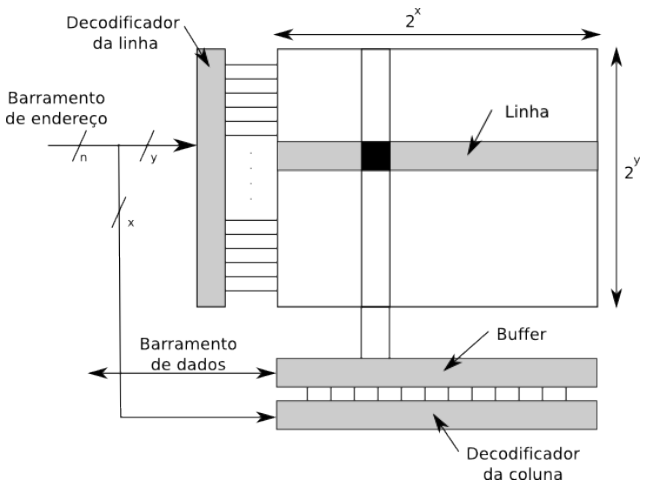

Figura 1.2: Estrutura típica de uma memória DRAM.

Embora o acesso à memória geralmente requer as etapas de decodificação e a de pré-carga, existem otimizações em um nível mais alto que podem ser feitas para reduzir o tempo de acesso

à memória. O modo de paginação pode ser eficientemente explorado com o uso de escalonadores que acessam dados em uma mesma página sucessivamente. Assim ao carregar uma página da memória ela é colocada em um "buffer" e futuros acessos a mesma página não necessitam da etapa de decodificação de linha [Panda, Dutt e Nicolau 1998].

1.1.2 Memória auxiliar

Também conhecida como memória cache, ela é utilizada para compensar a latência causada pela diferença de velocidades entre o processador e a memória principal. É uma memória interna ao sistema (*on-chip*) que armazena cópia de um subconjunto dos dados e das instruções presentes na memória principal. A tecnologia usada nessas memórias é SRAM.

As memórias cache são caracterizadas por três características básicas. Seu tamanho total, sua associatividade e seu tamanho de bloco.

Ao ser diretamente mapeada (quando sua associatividade é um) cada endereço na memória principal tem um único mapeamento na memória cache. Se tiver associatividade dois, cada endereço na memória principal tem dois possíveis mapeamentos na memória cache, assim sucessivamente. Quando a associatividade atinge o tamanho total da cache, ela é chamada de totalmente associativa, onde cada endereço na memória principal pode ser mapeado para qualquer endereço na memória cache [Patterson e Hennessy 2008].

Diz-se que ocorre uma falta ao se referenciar um endereço que não está localizado na cache. A unidade de controle da memória cache detecta essa falta e requisita ao nível inferior de memória o bloco que falta. O nível inferior então retorna o dado que então é escrito no nível atual e transmitido para o processador.

No caso de caches diretamente mapeadas, o bloco pode ser escrito em somente uma única posição. Já para as memórias cache com associatividade dois ou maior, existe mais de um lugar onde o mesmo bloco pode ser mapeado. Por isso, caso ao buscar um dado do nível inferior a cache tenha que trocar um bloco que está armazenado atualmente por outro, ela segue um política de troca ("replacement policy"). A política mais usada atualmente é a LRU ("least recently used"), que retira da memória o bloco que não é refenciado há mais tempo. Existem outras políticas, como o FIFO ("first-in first-out") que retira o bloco que está há mais tempo armazenado [Patterson e Hennessy 2008].

Ao escrever um dado na memória cache, pode-se adotar diversas políticas de escrita. Caso um dado seja escrito somente em um nível de cache, não repassando para o nível inferior de memória, poderá haver inconsistência nos dados. Uma solução é toda vez que um dado seja

modificado na cache, essa modificação é propagada para o nível inferior. Essa política de escrita é chamada de "write-through". Uma outra solução é o "write-back". Nesta política de escrita o dado é escrito somente no nível atual de cache, mantido assim até antes desse mesmo dado ser trocado por algum outro. Antes da troca acontecer, a escrita é propagada ao nível inferior de memória [Patterson e Hennessy 2008].

1.1.3 Memória de rascunho

A memória de rascunho (SPM) é uma memória SRAM interna ao sistema (*on-chip*), portanto próxima ao processador, assim garantindo tempo de acesso de um único ciclo [Panda, Dutt e Nicolau 2000]. Diferente das memórias cache, o espaço de endereçamento dessa memória é disjunto do espaço da memória principal como ilustrado na Figura 1.3. Sendo assim, cabe ao desenvolvedor do sistema escolher o conteúdo da SPM [Banakar et al. 2002].

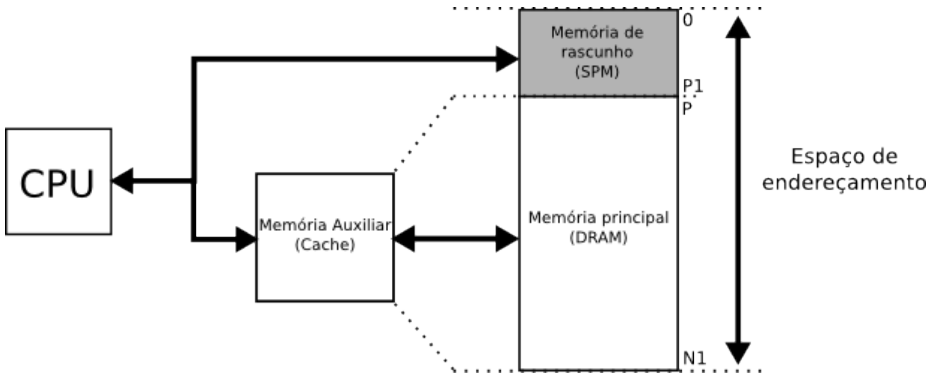

Figura 1.3: Divisão do espaço de endereçamento entre SRAM e DRAM [Banakar et al. 2002]

A SPM pode ser facilmente implementada com um arranjo de célula de memória de SRAM com 6 transistores. São necessários também decodificadores de linha e coluna, "sense amplifiers" e "drivers", confome mostra esquematicamente a Figura 1.4 [Banakar et al. 2002].

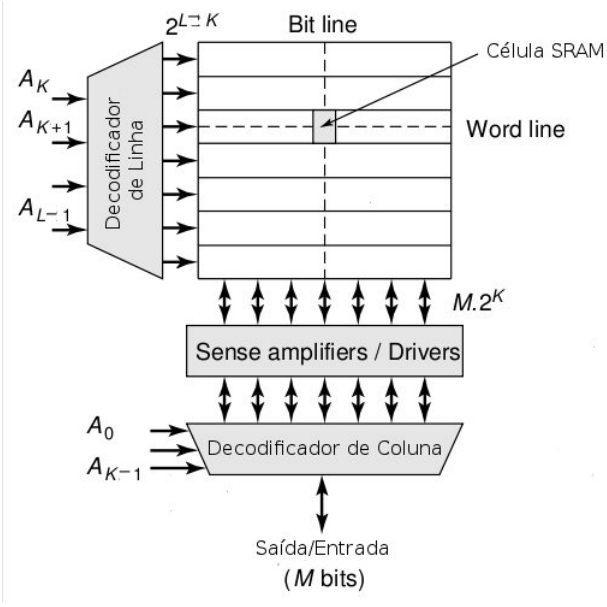

Figura 1.4: Estrutura de uma memória de rascunho [Rabaey 2006].

1.1.4 Aplicação

Um circuito integrado que implementa grande parte ou até mesmo todas as funcionalidades de um sistema eletrônico completo é chamado de *System-on-chip* (SoC). Exatamente quais componentes são agregados ao circuito varia de acordo com sua aplicação. Os *Systems-on-chip* podem ser encontrados em diversas categorias de produtos como telefones, televisões, produtos de telecomunicação, etc [Jerraya e Wolf 2004].

A Figura 1.5 mostra o diagrama de blocos de um SoC para multimídia, em destaque os subsistemas de hierarquia de memória. Trata-se de um sistema para aplicações de vídeo digital contendo dois processadores, controladores de memória, memórias internas (cache e SPM), interfaces de áudio e vídeo, etc [Talla e Golston 2007].

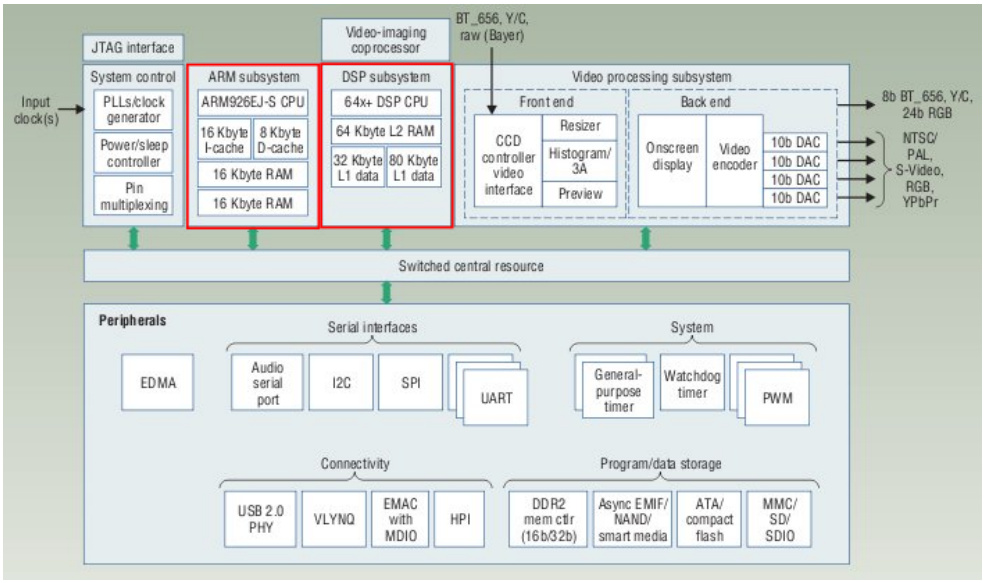

Figura 1.5: Diagrama de blocos de um SoC.

As memóras são essenciais para o funcionamento de qualquer sistema moderno. Nos projetos digitais contemporâneos, uma grande parte da área do chip é dedicada para armazenar valores e instruções do programa [Rabaey 2006]. No desenvolvimento dos *SoCs* deve-se levar em consideração características como performance, consumo de energia, predictibilidade, custo de desenvolvimento, custo unitário e *time-to-market*. Sendo que os três objetivos considerados mais importantes são: performance, eficiência energética e predictibilidade. O subsistema de memória é um dos responsáveis pelo gargalo do sistema em relação à performance e consumo de energia do sistema, oferecendo assim o maior potencial de otimização [Verma e Marwedel 2005].

1.2 Impacto da hierarquia de memória

As memórias diferem em várias características, não só sua estrutura e aplicações, como também funcionalidades e custo.

O objetivo de uma hierarquia de memória é fazer com que o usuário disponha da maior quantidade de memória, sem ser penalizado pelo tempo de acesso alto. Para isso são utilizados estruturas como memórias cache e SPM, que conseguem um tempo de acesso reduzido em relação às outras estruturas, porém com um custo mais elevado, conforme ilustra a Figura 1.6.

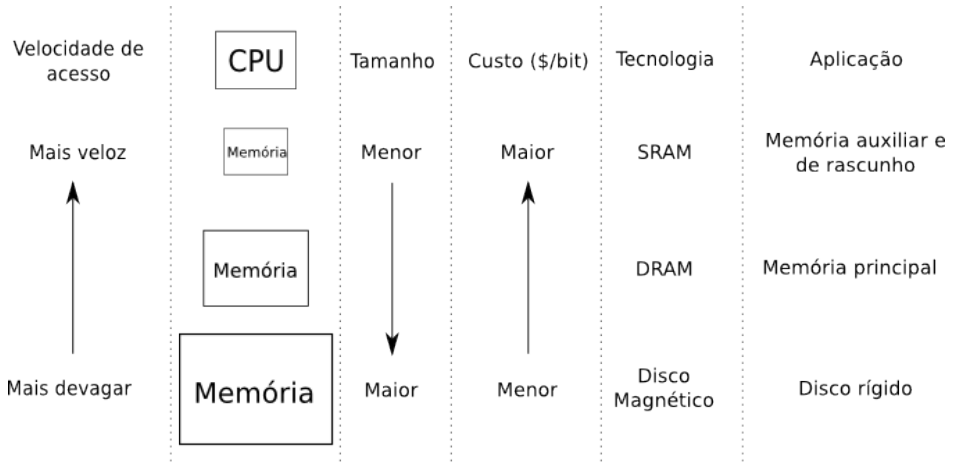

Figura 1.6: A base de uma hierarquia de memória. [Patterson e Hennessy 2008]

Entretanto, desde 1980 a velocidade dos processadores tem apresentado um crescimento de 50-100% por ano, enquanto a velocidade das memórias DRAM tem crescido aproximadamente 7% por ano [Henessy, J. L. e Patterson, D. A. 2002]. Assim, os processadores passam maior parte do seu tempo suspensos, esperando a resposta do sistema de memória. Esse problema é conhecido como "Memory Wall Problem", onde a velocidade do sistema é governada pela velocidade das memórias, e não do processador [Wulf e McKee 1994], conforme ilustra a Figura 1.7, retirada de [Henessy, J. L. e Patterson, D. A. 2002].

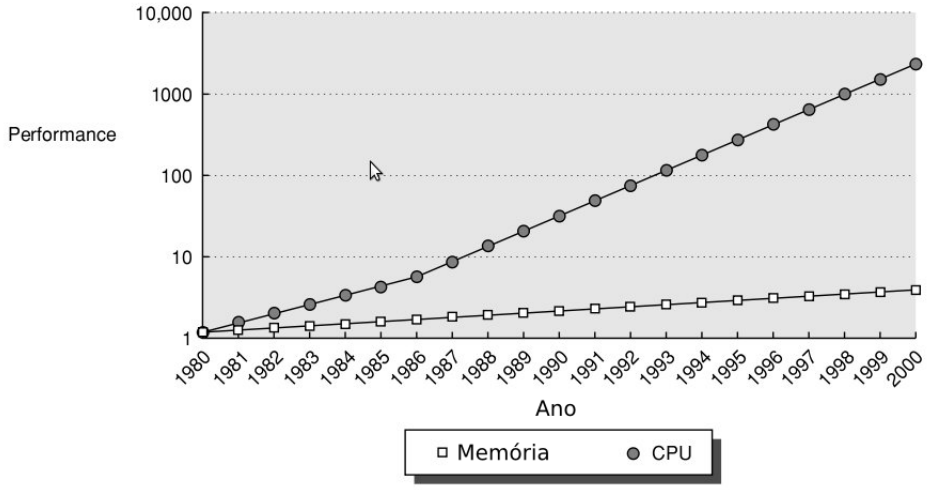

Figura 1.7: O crescimento da diferença de velocidades entre o processador e a memória.

1.2.1 Impacto no desempenho

O desempenho de uma hierarquia de memória pode ser medido pelo tempo médio de acesso da hierarquia [Henessy, J. L. e Patterson, D. A. 2002]. Tal medida está fortemente ligada à sua taxa de acerto (miss rate) e o seu tempo de acesso. Nas memórias de rascunho a taxa de acerto é 100%, isso porque o dado sempre estará presenta nela. Sendo assim, a taxa de acerto é um conceito fortemente ligado às memórias cache, sendo que suas configurações (tamanho, associatividade, etc...) influenciam nela.

$$\text{Tempo médio de acesso} = \text{Tempo de acesso} + \text{Taxa de acerto} \times \text{Penalidade}$$

A Figura 1.8 ilustra como pode variar a performance de uma hierarquia de memória, ao escolher diferentes configurações da memória cache [Henessy, J. L. e Patterson, D. A. 2002].

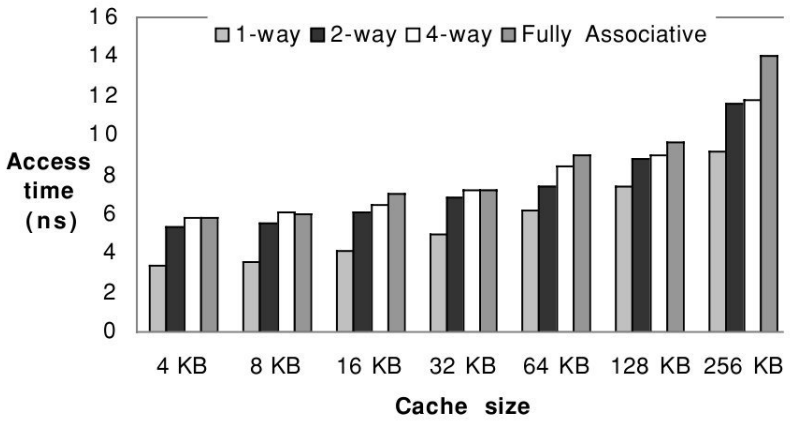

Figura 1.8: Tempo de acesso ao variar a associatividade e tamanho em Caches CMOS.

Contudo deve-se levar em conta não somente o tempo de acesso da memória, mas também sua taxa de acerto. A figura 1.9, retirada de [Patterson e Hennessy 2008], demonstra como varia a taxa de falta de uma memória conforme variam algumas de suas características. Apesar do caso ideal ser uma memória grande com um tamanho de bloco também grande, essas características podem aumentar o tempo de acesso da memória. Logo, deve-se encontrar um equilíbrio na configuração da memória afim de minimizar o tempo médio de acesso.

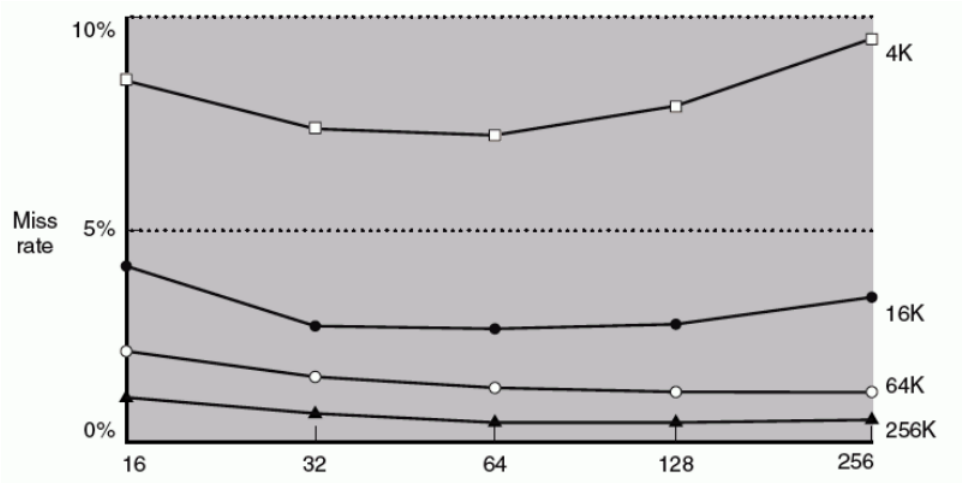

Figura 1.9: Taxa de falta variando parâmetros em uma cache [Patterson e Hennessy 2008].

O tempo médio de acesso de uma hierarquia de memória não depende exclusivamente da configuração de uma memória cache, mas também como os componentes da hierarquia estão dispostos. Em uma hierarquia sem memória cache, o tempo de acesso da memória principal pode ser de 10-20 vezes mais lento que uma operação no "data-path" [Panda, Dutt e Nicolau 1998]. Logo, é importante explorar o comportamento da memória principal no sistema de memória. O uso de SPM pode apresentar melhora no desempenho de 22.9% [Egger et al. 2006] e 29-33% [Panda, Dutt e Nicolau 2000], dependendo das técnicas utilizadas. Cada mudança nas características em um componente da hierarquia afeta o padrão de acessos em outro.

Para usar efetivamente a memória *on-chip* é preciso avaliar as vantagens de ambas memória cache e memória de rascunho, incluindo os dois tipos de módulos em um mesmo chip. Assim é importante avaliar a hierarquia de memória como um todo e não cada componente individualmente.

1.2.2 Impacto na energia consumida

A hierarquia de memória, além de ser o gargalo quanto ao desempenho dos sistemas, também tem se mostrado o gargalo quanto à energia consumida [Verma e Marwedel 2005]. Conforme ilustra a Figura 1.10, retirada de [Dally et al. 2008], um processador gasta a maior parte de sua energia suprindo dados e instruções e uma pequena parte executando operações aritméticas, operações de lógica de controle e com o clock.

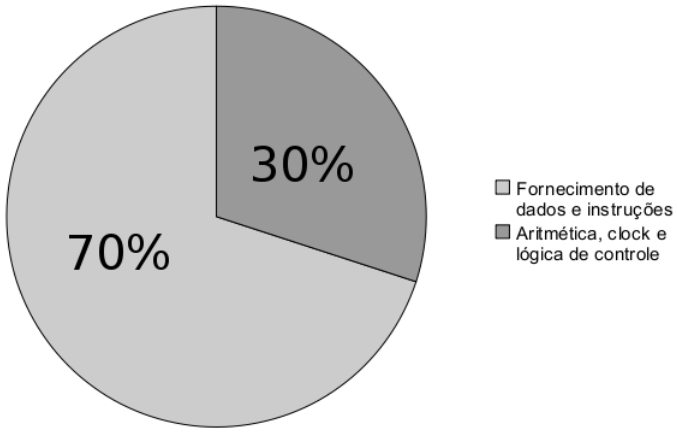

Figura 1.10: Distribuição da energia gasta em um processador [Dally et al. 2008].

Dentre os componentes da hierarquia de memória, um grande responsável pelo consumo de energia é a memória principal. Por ter uma capacidade de armazenamente bem maior que as memórias auxiliares e SPM, a memória consome mais energia para manter os valores de seus bits estáveis.

As memórias cache consomem cerca de 25% e 24%, para caches de intruções e dados (tamanho de 8k bytes e associatividade dois) respectivamente [Dally et al. 2008], do total de sua energia consumida com o lógica de controle e tags. Por isso as memórias de rascunho, por não necessitarem de um controle feito por hardware, são ótimas candidatas, considerado o consumo energético. Porém sua desvantagem é que exigem um suporte por software.

Torna-se claro que para melhorar a eficiência dos processadores, deve-se focar no suprimento de dados e instruções [Dally et al. 2008]. Para isso é preciso avaliar o consumo de energia de uma hierarquia de memória como um todo.

1.3 Contribuições técnico-científicas

Este trabalho propõe geração de modelos de hierarquia de memória para sistemas embarcados, capazes de fornecer informações sobre o consumo energético, área e desempenho da hierarquia de memória. Através da modelagem é possível construir diversas hierarquias contendo como componentes: memória principal, memória de rascunho e memória auxiliar.

1.4 Organização da monografia

Esta monografia está organizada da seguinte forma:

O Capítulo 2 revisa os modelos de hierarquias de memória voltados a SoCs com um único core e com múltiplos cores já existente, e compara com o modelo prosposto.

O Capítulo 3 descreve o modelo proposto para SoCs com único core. Aborda aspectos de implementação, validação do modelo e otimizações.

O Capítulo 4 visa introduzir alguns conceitos sobre hierarquia de memórias para MPSoCs.

No Capítulo 5 são mostradas as conclusões e trabalhos futuros. Também é abordada a adequação dos modelos à automação de projetos.

2 *Trabalhos correlatos em modelagem de memória*

Este capítulo tem como objetivo relatar os trabalhos correlatos realizados no meio científico e acadêmico. Inicialmente se relatará os trabalhos referentes a simuladores de hierarquias de memória com um único core, posteriormente serão abordados modelos com multi cores.

2.1 Modelos voltados a SoCs com um único core

Dinero IV [Edler 2004] é um simulador de hierarquias de memórias cache com um único processador. Sua simulação é dirigida por traces de memória. Em sua simulação, o Dinero IV fornece informações muito detalhadas sobre o desempenho das hierarquias de memória cache, suportando diversos níveis de memória e diversas configurações. Porém, ele não considera aspectos energéticos da hierarquia, não fornecendo nenhum suporte ou informação sobre o consumo de energia ou eficiência energética. O Dinero IV também não possibilita a modelagem de outros componentes da hierarquia de memória, como por exemplo a memória de rascunho ou memória principal, restringindo-se à memórias cache.

Em [Benini et al. 2000] é proposto que, ao gerar uma hierarquia de memória para um SoC, crie-se uma lógica de decodificação que mapeie endereços não consecutivos de memória, em uma pequena memória local. Com sua estrutura similar à memória de rascunho, esta pequena memória local permite que, ao contrário do que é geralmente feito nas memórias de rascunho, sejam mapeados endereços não-contíguos do programa. Com sua simulação também dirigida por traces, é realizado um *profiling* para definir quais endereços de memória são os melhores candidatos para serem mapeados para a pequena memória local, conseguindo-se assim uma melhora significativa na eficiência energética. Porém, o método proposto não leva em consideração a memória principal da hierarquia de memória. Consequentemente, a energia que é gasta pela mesma. O método também não possibilita o uso de memórias caches concorrentemente ao mapeamento dos endereços para a memória local e não provê dados quanto ao desempenho da

hierarquia.

A abordagem proposta em [Panda, Dutt e Nicolau 1997] propõe um método para geração de uma hierarquia de memória interna ao SoC (*on-chip*). Essa hierarquia pode ser composta com uma memória de rascunho controlada por software (SPM) e uma memória cache. Tendo como restrição um tamanho máximo para a memória interna ao SoC e baseando-se na estimativa da performance da hierarquia de memória, é feita uma busca no espaço de soluções, alterando-se alguns parâmetros da memória de rascunho e da cache. Apesar de permitir o uso simultâneo de memória caches e de rascunho na mesma hierarquia, o método não modela memórias externas ao SoC (*off-chip*). Observou-se também que o método apresenta restrições quanto à configurabilidade das memórias cache, o que restringe o espaço de busca do método. Ele também se restringe quanto a modelagem dos aspectos energéticos da hierarquia, não fornecendo nenhum suporte ou informação sobre o consumo de energia ou eficiência energética.

O CACTI [Thoziyoor Shyamkumar; Muralimanohar 2008] é um modelo analítico que, a partir das características de uma configuração de memória (tamanho, associatividade, tamanho do bloco, etc), realiza diversos cálculos e fornece características físicas da memória. Entra essas características fornecidas está o tempo médio de acesso à uma memória, o consumo médio de energia por acesso e a área da memória. O CACTI consegue calcular essas tanto para memórias cache como de rascunho ou uma memória principal.

2.2 Modelos voltados a SoCs com múltiplos cores

O CMP$im é um simulador de múltiplos cores. Seu fluxo de dados é muito parecido com o modelo proposto, no qual a os endereços de um trace são capturados e executados na hierarquia de memória. Apesar de ser um simulador de múltiplos cores, o modelo não suporta a execução de mais de uma aplicação por vez, porém essa aplicação pode ser multi-threading. O CMP$im permite a total customização de uma hierarquia de memórias cache, podendo até mesmo conter memória compartilhada entre processadores. Entretanto o modelo não fornece informações quanto ao consumo de energia da hierarquia de memória ou sobre o tempo de acesso. O simulador também não apresentou suporte a simulação de memórias externas ao SoC (*off-chip*) e memória de rascunho [Jaleel R. S. Cohn e Jacob 2008].

[Girao e Wagner 2009] propõe uma abordagem diferenciada nas hierarquias de memória avaliadas. Neste modelo é proposto simular inteiramente o SoC, incluindo processadores, rede de interconexão e as memórias. Nele são utilizadas *networks-on-chip* (NoCs) para realizar a comunicação entre *cores*. São propostas quatro hierarquias diferentes de memória: memória

distribuída (*distributed memory*), memória compartilhada (*shared memory*), memória compartilhada distribuída (*distributed shared memory*) e memória nDMA. Também são suportadas memórias cache, sendo que suas configurações podem variar quanto tamanho da memória, política de troca, associatividade e tamanho do bloco. O modelo fornece informações quanto ao consumo de energia da hierarquia de memória, assim também como o número de ciclos gasto por uma determinada aplicação. Porém não são fornecido dados quanto ao tempo absoluto de execução de cada simulação. Tal informação é relevante pois caso apresentado um tempo de simulação muito elevado, tal modelo se torna proibitivo para a busca no espaço de soluções. O modelo também não oferece suporte à memória de rascunho.

Em [Tao, Schulz e Karl 2003] é proposto o SIMT, uma ferramenta para simular e avaliar hierarquias de memória compartilhada. Diferentemente da maioria das abordagens propostas até agora, o SIMT utiliza um simulador de multiprocessadores Intel x86 para gerar os acessos a hierarquia de memória. Internamente o SIMT é composto por três módulo básicos. Um simulador de memória cache customizável em política de escrita, tamanho da memória, associatividade e tamanho do bloco. O simulador de memória cache também é responsável pelas políticas de coerência de cache. Os outros dois componentes restantes são um simulador de memória compartilhada distribuída e um modelo de rede de interconexão. Entretanto o SIMT não é direcionado para a simulação de sistemas embarcados. Sendo assim não suporta memórias de rascunho e não tem suporte quanto ao consumo de energia da hierarquia de memória.

3 *Modelagem de hierarquia de memória para core único*

Este capítulo inicialmente formaliza alguns conceitos necessários para a formulação da implementação do modelo proposto. Logo após é descrito o fluxo de projeto do modelo, descrevendo as ferramentas utilizadas e os componentes que foram implementados. Em seguida são relatados aspectos de implementação do código do modelo; linguagem, compilador, decisões de implementação visando eficiência. Depois são relatados os resultados experimentais obtidos.

3.1 Descrição do modelo proposto

Quando uma aplicação embarcada é compilada, seus os dados podem ser armazenados tanto nas memórias de rascunho quando na memória principal.

Como acessos aos elementos de memória são dependentes da computação efetuada pelo software embarcado para cada conjunto de estímulos, a otimização deve se basear em um padrão de acessos típicos, o qual é capturado na forma de trace [Mendonca et al. 2009].

Um trace de memória T é uma tupla $(\alpha_1, \alpha_2, ..., \alpha_i, ..., \alpha_n)$ que representa uma sequência de sucessivos endereços a serem acessados no subsistema de memória, onde a_i representa o i-ésimo endereço [Uhlig e Mudge 1997].

O tempo de acesso de uma memória M, denotada por λ_M, é o tempo gasto em ciclos do processador para acessar uma posição na memória M.

A capacidade de armazenamento de uma memória M, denotado por C_M, é o seu tamanho em bytes.

A energia consumida para acessar a memória M, denotado por E_M, é a energia gasta para efetuar uma leitura ou escrita na memória.

O número de acessos a uma memória M, denotado por N_M, é quantos acessos foram realizados à memória M em uma simulação.

Uma memória M pode ser uma memória principal (MP), memória de rascunho (SPM) ou memória cache de intruções (I-Cache) , dados (D-Cache) ou unificada (U-Cache) [Mendonca et al. 2009].

Caso seja uma memória principal, ela é caracterizada também pelo número de bits para páginação P_M [Panda, Dutt e Nicolau 1998].

Caso seja uma memória cache, ela é caracterizada também pela associatividade A_M, tamanho do bloco B_M em bytes, política de escrita WP_M, e política de troca de blocos RP_M. A política de escrita pode ser *write-through* ou *write-back.* A memória cache também pode ser classificada como cache de instruções (I-cache), cache de dados (D-cache) ou cache unificada (U-cache).

Caso seja uma memória de rascunho, ela é caracterizada também pelo endereço inicial, denotado por I_M, que é o ínicio do endereço que será disjunto ao da memória principal, tendo como limite $I_M + C_M$.

A computação realizada na simulação de um determinado programa pode ser vista na figura 3.1. A configuração de hierarquia refere-se a configuração de memória que irá ser testada.

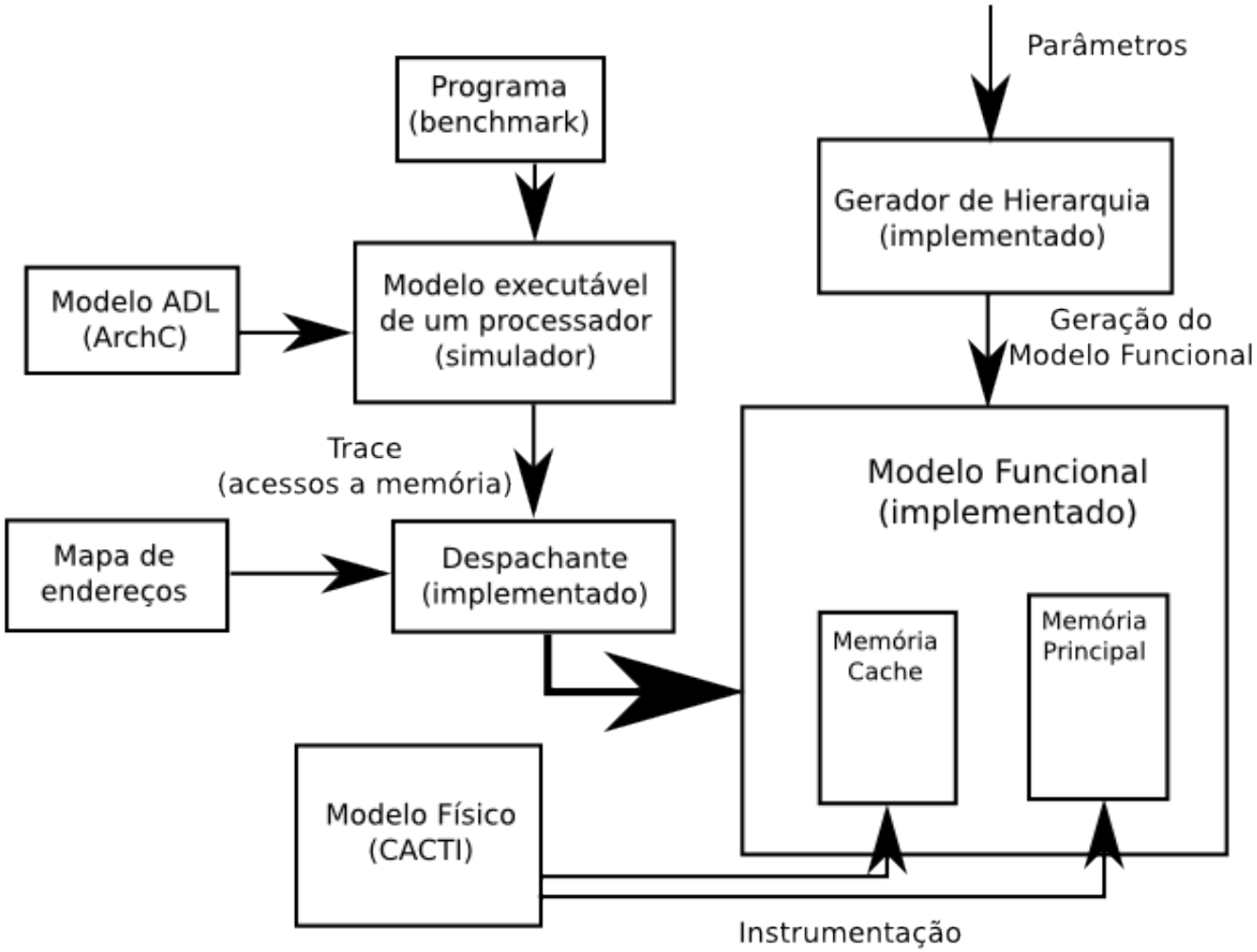

Figura 3.1: Fluxo esquemático de execução de uma simulação.

3.2 Implementação do modelo

O modelo de hierarquia de memória para core único foi escrito na linguagem C++, compilado com g++ 4.3.3. A seguir é descrito passo-a-passo o fluxo de execução (Figura 3.1) de uma simulação.

3.2.1 Gerador de Hierarquia de Memória

Para possibilar a geração automática de diferentes hierarquias de memória, foi proposto este Gerador de Hierarquia de Memória (Figura3.2) como parte do fluxo de execução.

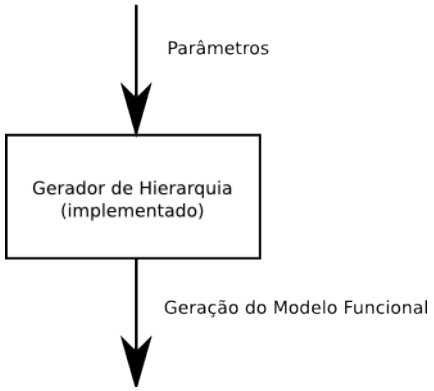

Figura 3.2: Gerador de Hierarquia de Memória.

Este módulo tem como entrada determinados parâmetros (passados pelo usuário) que irão definir a hierarquia a ser criada. Caso nenhum parâmentro seja passado ao gerador, ele irá assumir uma hierarquia padrão, sendo esta:

- Memória Principal.
- Nível 1 de memória cache de instrução.
- Nível 1 de memória cache de dados.

Caso se deseja uma arquitetura diferente, is parâmetros que são reconhecidos pelo gerador, afim de modelar a arquitetura, são:

- SPM : Adiciona SPM à hierarquia.
- U1CACHE : Faz com que o nível 1 de cache seja unificado (dados e instrução).
- U2CACHE : Faz com que o nível 2 de cache seja unificado (dados e instrução).
- DI2CACHE : Faz com que o nível 2 de cache seja disjunto (1 memória de dados e uma memória de instrução).
- NO_CACHE : Desabilita qualquer nível de cache na hierarquia de memória.

Dados os parâmetros fornecido pelo usuário, o gerador de hierarquia irá produzir em sua saída um Modelo Funcional da hierarquia em questão. Tal modelo será utilizado para a simulação da hierarquia.

3.2.2 Modelo Funcional

O modelo funcional (também implementado) é a representação da hierarquia de memória em si. É no modelo funcional (Figura 3.3) que está descrito o comportamento de cada módulo da hierarquia de memória (memória cache, memória de rascunho e memória principal). Também é nesse modelo que está descrita a interconexão entre os módulos.

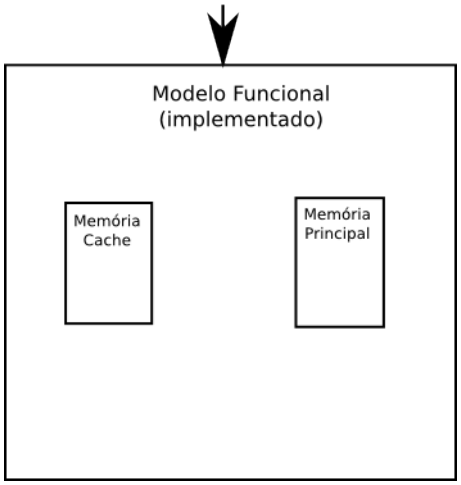

Figura 3.3: Modelo Funcional.

Neste passo do fluxo de execução, cada memória da hierarquia que foi gerada no passo anterior, é responsável por armazenar sua própria configuração. Suas características estão definidas em seu módulo (3.1). Os aspectos de configuração (tamanho, associatividade, etc) são definidos pelo usuário. Já os parâmetros dependentes de tecnologia (energia gasta por acesso, tempo gasto por acesso) serão fornecidas em um próximo passo pelo modelo físico (3.2.3).

Após estar totalmente instrumentado, o modelo funcional irá contabilizar o total de acessos a cada módulo, simulando os acessos à memória de um programa (determinado pelo usuário).

3.2.3 Modelo Físico

Neste passo do fluxo de execução da simulação, o modelo funcional é instrumentado pelo modelo físico. O modelo físico (no caso deste trabalho foi utilizado o CACTI [Thoziyoor Shyamkumar; Mu
obtém do modelo funcional as características estruturais de cada módulo individualmente. Com
essas características em mãos, ele fornece informações dependentes de tecnologia como o consumo de médio energia por acesso e tempo médio por acesso de cada módulo (Figura 3.4).

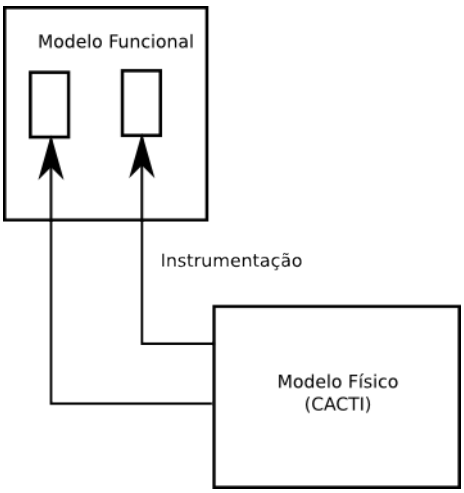

Figura 3.4: Modelo Físico.

Após a intrumentação do modelo funcional pelo modelo físico, a hierarquia está pronta para ser simulada. O fluxo de execução tem que decidir agora qual programa e qual processor será utilizado.

3.2.4 Modelo Executável de um processador

Neste passo, é definido qual processador será utilizado na simulação da hierarquia, assim também como o programa a ser executado. Através de uma linguagem de descrição de arquitetura (no caso deste trabalho foi utilizado o ArchC [Rigo et al. 2004]) é gerado um modelo executável de um processador (no caso deste trabalho foi adotado o MIPS).

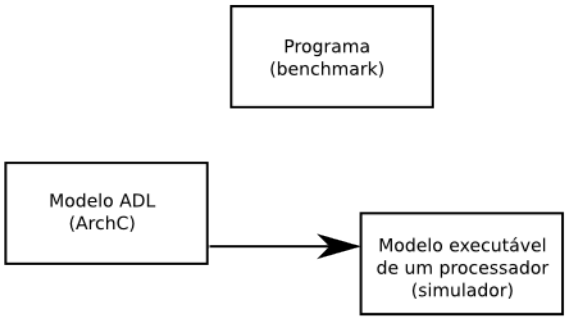

Figura 3.5: Modelo Executável de um processador.

O programa a ser utilizado no simulador pode ser qualquer um. Para fins de comparação, neste trabalho foram utilizados programas do pacote MiBench [Guthaus et al. 2001].

O modelo executável do processador, ao executar um programa, irá gerar acessos à memória na forma de um Trace. Esses acessos serão retransmitidos, em tempo de execução, ao modelo funcional da hierarquia, que irá se comportar como se fosse um hierarquia de memória real.

3.2.5 Despachante

Tendo em mãos o modelo funcional da hierarquia totalmente caracterizado e o Trace de acessos à memória (gerado pela representação executável de um processador), observou-se a necessidade de criar-se um intermediário na comunicação entre os dois módulos, denomidade de despachante. Conforme o nome diz, este módulo é responsável por receber os acessos à memória e repassar-los ao módulo correspondente, de acordo com o mapa de endereços fornecido pelo usuário. Este mapa de endereço indica quando que o acesso deve ser direcionado para memória de rascunho e quando ele deve ser direcionado para a memória cache (tal fato se deve ao espaço de endereçamento dos mesmos ser disjunto).

Apesar de inicialmente o despachante ser exclusivamente feito para rotear os acessos à memória para os respectivos módulos, posteriormente foram delegadas outras funções. Essas funções prevêem detectar algumas anomalias no Trace e corrigir-las.

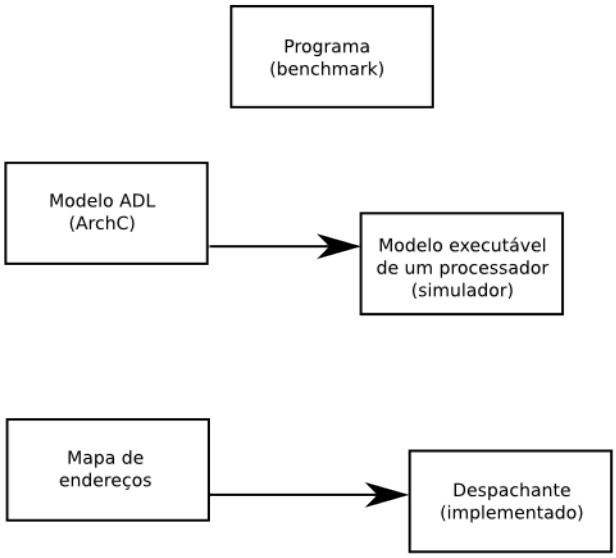

Figura 3.6: Despachante.

Com este último componente, conclui-se o fluxo de execução do simulador, sendo somente necessário executar o programa e aguardar a resposta do modelo funcional. O modelo irá fornecer várias características sobre a hierarquia de memória, conforme mencionado anteriormente.

3.3 Resultados experimentais

Para a realização dos experimentos foram utilizados programas extraídos do benchmark MiBench [Guthaus et al. 2001]. Os estímulos utilizados foram os que acompanham o pacote, chamados de *large inputs*.

Os experimentos foram executados em um processador Intel Xeon E5430 (*quad-core)* rodando na frequência de 2.66GHz com 4Gb de memória principal rodando no sistema operacional Debian GNU Linux (kernel 2.6.28).

3.3.1 Validação funcional

A validação do funcional modelo teve que ser restrita a validação da hierarquia de memória cache. Não foi possível extender o processo para validar memória principal e memória de rascunho por não se ter encontrado uma ferramenta que possibilitasse a comparação dos resultados, e consequentemente a validação. Para validar a hierarquia de memória cache, os resultados obtidos nos ensaios do modelo proposto foram comparados com os resultados forne-

cidos pela ferramenta DINERO IV [Edler 2004]. O DINERO fornece para cada memória cache da hierarquia: quantidade de acessos (leitura, escrita), quantidade de acerto (leitura, escrita), quantidade de falta (leitura, escrita) e entre outros, sendo que os três primeiros foram utilizados para comparação na validação.

Infraestrutura de experimentação

Conforme pode ser observado na Figura 3.6, o Trace utilizado pelo modelo proposto é obtido através da execução em um modelo executável de processador. Esse Trace é retransmitido em tempo de execução para a hierarquia de memória (Figura3.7 a)). Já na abordagem proposta pelo DINERO IV, este mesmo Trace é armazenado em disco e sua leitura é efetuada ao simular a hierarquia de memória cache (Figura3.7 b)).

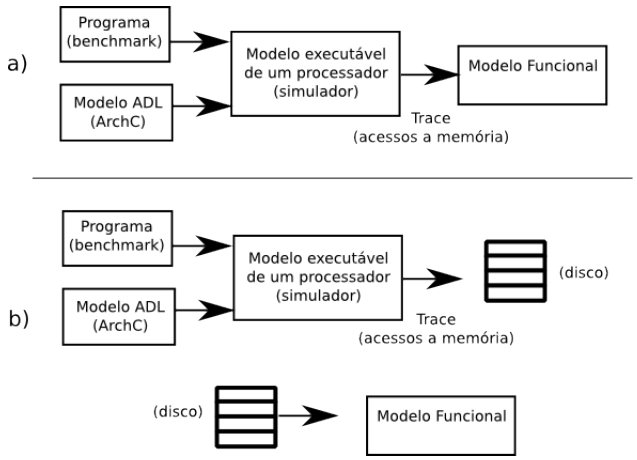

Figura 3.7: Infraestrutura de experimentação.

Arquiteturas para validação

Para validar o modelo foram propostas duas arquiteruras de hierarquia de memória, conforme ilustra a Figura 3.8. Apesar de no modelo proposto ser utilizado uma memória principal, ela não é factível de validação, estando presente somente para completude da hierarquia. Vale ressaltar que ela não influie no resultado final das memórias cache.

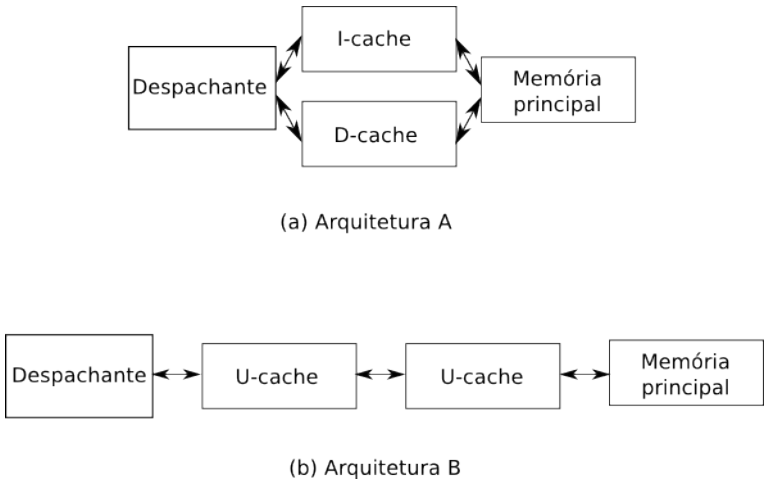

Figura 3.8: Arquiteturas de hierarquias de memória utilizadas na validação.

Ensaios

Para ambas as arquiteturas propostas foram realizados diversos ensaios, variando os diversos atributos possíveis. As memórias cache tiveram variações quanto ao seu tamanho, associatividade, tamanho do bloco e política de escrita, sendo que:

$C_{cache}\ \varepsilon\ \{\ 1024,\ 2048,\ 4096\}$ em bytes,

$A_{cache}\ \varepsilon\ \{\ 1,\ 4,\ 8,\ 16\}$,

$B_{cache}\ \varepsilon\ \{\ 4,\ 8,\ 16,\ 32\}$ em bytes e

$WP_{cache}\ \varepsilon\ \{\ wt,\ wb\}$.

Já a memória principal teve sua configuração sendo que:

$C_M\ =\ 4Mb$

$P_M\ =\ 16\,bits$.

Foram simuladas todas as possíveis combinações dos atributos das memórias, tanto para a Configuração A quanto para a Configuração B. Deve-se ressaltar que para a Configuração B os tamanhos dos blocos são iguais para ambas as memórias cache. Sendo assim foram totalizadas 4608 simulações para a Arquitetura A e 2304 simulações para a Arquitetura B.

Resultados

Dado o elevado volume de simulações necessárias para realizar a validação, foi utilizado somente o programa *Qsort* do pacote de benchmarks. Foi escolhido o Qsort pois o tempo de

execução de cada diferente configuração era aceitável e possibilitava a realização em tempo hábil.

Para a Configuração A, foram validados 100% das simulações ocorridas. Todos os resultados foram consistentes com os encontrados pelo ferramente usada na comparação. Nessa configuração pode-se observar o funcionamento correto do particionamento das cache de instrução e dados. Contudo, por existir apenas memória cache nível um na Configuração A, não ficou evidente a validação quanto as políticas de escritas, sendo necessário a Configuração B.

Para a Configuração B, não foram validados 100% das simulações. Houveram pequenas diferenças entre os resultados obtidos e os resultados esperados. Em 52% das simulações, foi encontrado um erro de aproximadamente 1% resultados obtidos, não representando grande siginificado no resultado final.

3.3.2 Avaliação da eficiência

Comunicação

Conforme pode ser visto na Figura 3.9, a comunicação entre componentes da hierarquia de memória (i.e. memória cache com a memória principal) é feita por chamada de métodos. Já a comunicação entre o representação executável (que gera o Trace) e o Despachante (que irá receber o Trace) é realizada através de socket [Group 1997]. Tal abordagem apresentava uma degradação no desempenho por apresentar um gargalo na comunicação utilizando o socket.

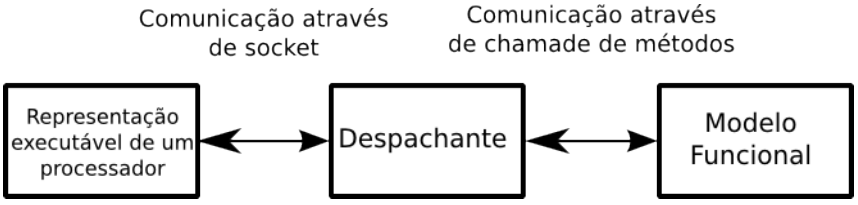

Figura 3.9: Esquema de comunicação entre os módulos

Para contornar esse problema foi proposto substituir o mecanismo de socket por uma biblioteca de troca de mensagem (MPI), sendo que nesse trabalho foi utilizada a implementação MPICH [MPICH-A 2005].

Para a avaliação do mecanismo de comunicação foram utilizadas oito configurações de hierarquia de memória, conforme mostra a tabela 3.1.

Configurações	U-cache (bytes)	I-cache (bytes)	D-cache (bytes)	SPM (bytes)
A	-	1 k	1 k	-
B	-	4 k	4 k	-
C	-	1 k	1 k	1 k
D	1 k	-	-	1 k
E	-	1 k	1 k	4 k
F	-	4 k	1 k	4 k
G	1 k	-	-	4 k
H	4 k	-	-	4 k

Tabela 3.1: Tamanho das memórias adotadas nas configurações para a avaliação do mecanismo de comunicação.

Para cada uma das configurações de hierarquia de memória, foram executados seis programas do pacote de benchmarks MiBench [Guthaus et al. 2001]. Sendo eles: Dijkstra, Jpeg-6a, Susan, Qsort, CRC32, Bitcount (todos utilizando os estímulos de entrada *large*).

Os resultados obtidos foram muito satisfatórios. Com o uso de MPI ao invés do socket, o sistema teve uma melhora média de 55% nos programas executados, conforme mostra a figura 3.10.

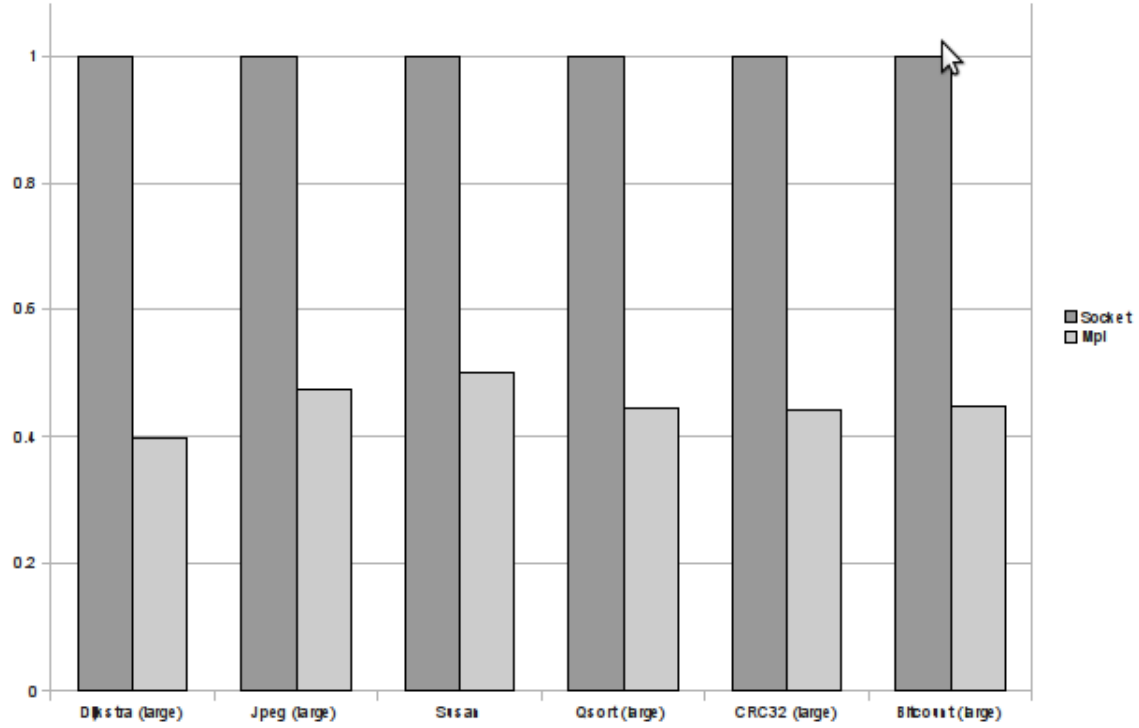

Figura 3.10: Tempo médio de execução dos programas nas hierarquias de memória, normalizado em relação ao tempo de execução com socket.

4 *Hierarquia de memória para múltiplos cores*

Um *system-on-chip* com mais de uma unidade de processamento é denomidado *multiprocessor system-on-chip* (MPSoC). Tipicamente tal sistema tem processadores heterogeneos: diferente tipos de unidade processamento, sistema de memória heterogeneamente distribuído e uma rede de interconexão também heterogenea. MPSoCs geralemente precisam de uma grande quantidade de memória, podendo ser composta tanto de memórias on-chip quanto off-chip [Jerraya e Wolf 2004]. Os multiprocessadores existentes podem ser categorizados em duas classes dependendo do número de unidades de processamento, que consequentemente irá ser mais adequado para uma determinada hierarquia de memória e rede de interconexão. Os multiprocessadores são classificados pela sua organização de memória, já que uma quantidade grande ou pequena de unidades de processamento pode variar muito conforme o decorrer dos anos[Henessy, J. L. e Patterson, D. A. 2002].

4.1 Memória centralizada compartilhada

Essas arquiteturas suportam algumas dúzias de processadores. Com uma memória única e compartilhada é possível interligar as unidades de processamento com um barramento simples, conforme ilustra a figura 4.1. Com caches grandes e múltiplos bancos de memória, essa hierarquia pode satisfazer a demanda da aplicação. Conforme aumenta o número de processadores, essa abordagem se torna menos atrativa, já que se torna menos eficiente. Por ter uma memória única que tem relação simétrica pra todos os processadores, esses multiprocessadores são chamados de multiprocessadores simétricos (SMPs) e por apresentar um tempo de acesso igual a todos os processadores, essa arquitetura é categorizada como acesso uniforme à memória (UMA) [Henessy, J. L. e Patterson, D. A. 2002].

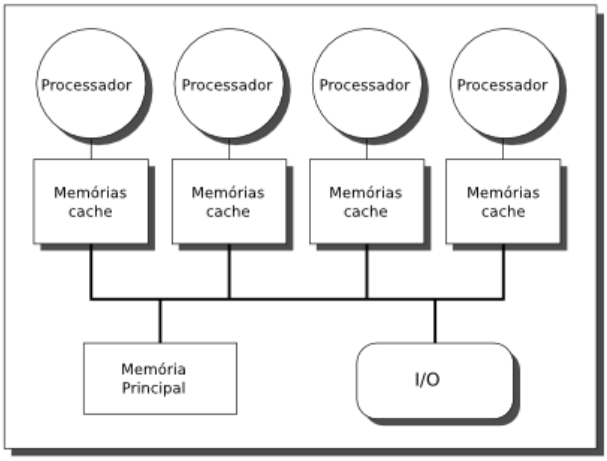

Figura 4.1: Arquitetura típica de um SMP [Henessy, J. L. e Patterson, D. A. 2002].

4.2 Memória distribuída

Para suportar uma quantidade mais elevada de processadores, ao invés de uma memória única e centralizada, a memória precisa ser distribuída entre os processadores conforme ilustra a figura 4.2. Assim, é possível suportar o tráfego de dados e ter um tempo de resposta mais rápido. Isso porque uma memória menor é mais rapidamente acessada, e a memória pode estar localizada perto do processador. Entretanto quando as memórias estão fisicamente espalhadas pelo sistema a comunicação de dados entre os processadores se torna mais complexa, podendo assim se diferir entre dois tipos: memória compartilhada distribuída o e multicomputador [Henessy, J. L. e Patterson, D. A. 2002].

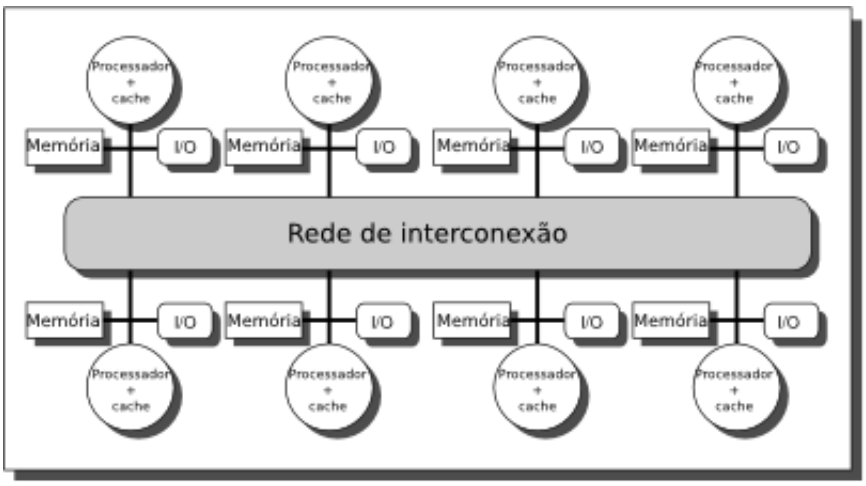

Figura 4.2: Organização típica de uma de memória distribuída.

4.2.1 Memória compartilhada distribuída

Nesse método a memória física pode ser acessada como um espaço de endereçamento lógico único. Esses multiprocessadores são chamados de arquiteturas de memória compartilhada distribuída (DSM). Ao contrário da memória centralizada compartilhada conhecidos como UMAs, os DSM tem acesso não uniforme a memória, conhecidos assim por nUMA [Henessy, J. L. e Patterson, D. A. 2002].

4.2.2 Multicomputadores

Uma outra alternativa quanto a comunicação entre processadores é ter os espaços de enderaçamentos disjuntos para cada um. Nessa arquitetura um processador não enxerga os dados do outro, sendo assim para se comunicarem os processadores usam troca de mensagens [Henessy, J. L. e Patterson, D. A. 2002].

4.3 Problema de coerência de cache

Um problema encontrado em arquiteturas com multiprocessadores é o uso de memórias cache. Para garantir a integridade dos dados e previnir que um processador acesse algum dado desatualizado, é necessário que a arquitetura tenha algum protocolo de coerência de cache, garantindo assim o funcionamento correto do software[Henessy, J. L. e Patterson, D. A. 2002].

5 *Conclusões e trabalhos futuros*

5.1 A adequação dos modelos à automação de projeto

A utilização da biblioteca de troca de mensagem MPICH, no contexto do modelo hierarquia de memória apresentou uma melhora na eficiência de 50% em relação ao uso de Socket.

O modelo de hierarquia de memória proposto se demonstrou adequado ao desevolvimento de pesquisa e automação no âmbito de sistemas embarcados. Ele foi encorporado com sucesso no otimizador de SPM para uma hierarquia fixa [**?**].

Com um tempo de execução aceitável para a simulação e avaliação de hierarquias de memórias, porém a validação do modelo ainda pode ser melhorada.

5.2 Extensões e generalizações sugeridas

As hierarquias de memória são cada vez mais importantes no desenvolvimento de um *system-on-chip*. Elas são responsáveis por grande parte da energia consumida e também podem limitar o desempenho do sistema caso não projetados de forma consciente. Cada dia surgem novas técnicas de otimização, sendo necessário formas de avaliar o impacto delas em uma hieraquia de memória.

Hierarquias de memória ainda é um tema aberto para muitas pesquisas e muito promissor no desenvolvimento de sistemas embarcados. Principalmente com a evolução de sistemas tipicamente de core único para a área de multiprocessadores em um único chip, a hierarquia de memória e a energia consumida pode apresentar um impacto ainda maior no sistema.

Com o propósito de avaliar hierarquias de memória com multiprocessadores, sugere-se a generalização do modelo de hierarquia de memória com um único core, para hierarquias de memória com múltiplos cores. Para tal realização o modelo deve levar em conta diversos aspectos. Primeiramente deve-se priorizar a modelagem para processadores ou homogêneos ou

heterogêneos. Também deve se levar em consideração protocolos de coerência de cache e redes de interconexão no chip (NoCs).

Referências Bibliográficas

[Banakar et al. 2002]BANAKAR, R. et al. Scratchpad memory : A design alternative for cache on-chip memory in embedded systems. In: *Proceedings of the Tenth International Symposium on Hardware/Software Codesign (CODES-02)*. New York: ACM Press, 2002. p. 73–78.

[Benini et al. 2000]BENINI, L. et al. Increasing energy efficiency of embedded systems by application-specific memory hierarchy generation. *IEEE Design & Test of Computers*, v. 17, n. 2, p. 74–85, 2000.

[Dally et al. 2008]DALLY, W. J. et al. Efficient embedded computing. *IEEE Computer*, v. 41, n. 7, p. 27–32, 2008.

[Edler 2004]EDLER, J. *Dinero IV Trace-Driven Uniprocessor Cache Simulator*. Jan 2004. `http://pages.cs.wisc.edu/ markhill/DineroIV/`.

[Egger et al. 2006]EGGER, B. et al. A dynamic code placement technique for scratchpad memory using postpass optimization. In: HONG, S. et al. (Ed.). *CASES*. [S.l.]: ACM, 2006. p. 223–233. ISBN 1-59593-543-6.

[Girao e Wagner 2009]GIRAO, D. B. G.; WAGNER, F. R. Performance and energy evaluation of memory hierarchies in noc-based mpsocs under latency. In: . [S.l.: s.n.], 2009.

[Group 1997]GROUP, T. O. *sys/socket.h - Internet Protocol family*. Jan 1997. `http://opengroup.org/onlinepubs/7990989775/xns/syssocket.h.html`.

[Guthaus et al. 2001]GUTHAUS, M. R. et al. MiBench: A free, commercially representative embedded benchmark suite. In: *Proceedings of the IEEE 4th Annual Workshop on Workload Characterization*. [S.l.: s.n.], 2001.

[Henessy, J. L. e Patterson, D. A. 2002]Henessy, J. L.; Patterson, D. A. *Computer Architecture a Quantitative Approach*. [S.l.]: Morgan Kaufmann Publishers, 2002.

[Jaleel R. S. Cohn e Jacob 2008]JALEEL R. S. COHN, C. K. L. A.; JACOB, B. Cmpsim: A pin-based on-the-fly multi-core cache simulator. *Workshop on Modeling, Benchmarking and Simulation*, 2008.

[Jerraya e Wolf 2004]JERRAYA, A.; WOLF, W. *Multiprocessor Systems-on-Chips (The Morgan Kaufmann Series in Systems on Silicon)*. [S.l.]: Morgan Kaufmann, 2004. Hardcover. ISBN 012385251X.

[Mendonca et al. 2009]MENDONCA, A. K. et al. Mapping data and code into scratchpads from relocatable binaries. *VLSI, IEEE Computer Society Annual Symposium on*, IEEE Computer Society, Los Alamitos, CA, USA, v. 0, p. 157–162, 2009.

[MPICH-A 2005]MPICH-A. *MPICH-A Portable Implementation of MPI*. Jan 2005. `http://www.mcs.anl.gov/research/projects/mpi/mpich1/`.

[Panda, Dutt e Nicolau 1997]PANDA, P. R.; DUTT, N. D.; NICOLAU, A. Architectural exploration and optimization of local memory in embedded systems. In: *ISSS*. [S.l.: s.n.], 1997. p. 90.

[Panda, Dutt e Nicolau 1998]PANDA, P. R.; DUTT, N. D.; NICOLAU, A. Incorporating DRAM access modes into high-level synthesis. *IEEE Trans. on CAD of Integrated Circuits and Systems*, v. 17, n. 2, p. 96–109, 1998.

[Panda, Dutt e Nicolau 2000]PANDA, P. R.; DUTT, N. D.; NICOLAU, A. On-chip vs. off-chip memory: the data partitioning problem in embedded processor-based systems. *ACM Trans. Design Autom. Electr. Syst*, v. 5, n. 3, p. 682–704, 2000.

[Patterson e Hennessy 2008]PATTERSON, D. A.; HENNESSY, J. L. *Computer organization and design: the hardware/software interface*. Fourth. Boston, MA, USA: Elsevier Morgan Kaufmann, 2008. ISBN 0-12-374493-8.

[Rabaey 2006]RABAEY, J. M. *Digital Integrated Circuits: A Design Perspective*. [S.l.]: Prentice Hall, 2006.

[Rigo et al. 2004]RIGO, S. et al. ArchC: A systemC-based architecture description language. 2004.

[Talla e Golston 2007]TALLA, D.; GOLSTON, J. Using davinci technology for digital video devices. *IEEE Computer*, v. 153, n. 10, p. 53–61, out. 2007.

[Tao, Schulz e Karl 2003]TAO, J.; SCHULZ, M.; KARL, W. A simulation tool for evaluating shared memory systems. In: *ANSS '03: Proceedings of the 36th annual symposium on Simulation*. Washington, DC, USA: IEEE Computer Society, 2003. p. 335. ISBN 0-7695-1911-3.

[Thoziyoor Shyamkumar; Muralimanohar 2008]THOZIYOOR SHYAMKUMAR; MURALIMANOHAR, N. A. J. H. J. N. P. *An integrated cache and memory access time, cycle time, area, leakage, and dynamic power model*. Jan 2008. `http://www.hpl.hp.com/techreports/2008/HPL-2008-20.html`.

[Uhlig e Mudge 1997]UHLIG; MUDGE. Trace-driven memory simulation: A survey (errata). *CSURV: Computing Surveys*, v. 29, 1997.

[Verma e Marwedel 2005]VERMA, M.; MARWEDEL, P. Memory optimization techniques for low-power embedded processors. In: CREMERS, A. B. et al. (Ed.). *GI Jahrestagung (1)*. [S.l.]: GI, 2005. (LNI, v. 67), p. 445. ISBN 3-88579-396-2.

[Wulf e McKee 1994]WULF, W. A.; MCKEE, S. A. *Hitting the Memory Wall: Implications of the Obvious*. [S.l.], nov. 1 1994. Wed, 13 Dec 1995 19:29:36 GMT.