UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Rafael Westphal

COMPUTAÇÃO ENERGETICAMENTE EFICIENTE SOB RESTRIÇÕES DE TEMPO REAL EM DISPOSITIVOS MÓVEIS: O IMPACTO DE SE OTIMIZAR O USO DE MEMÓRIA

Florianópolis

2013

Rafael Westphal

COMPUTAÇÃO ENERGETICAMENTE EFICIENTE SOB RESTRIÇÕES DE TEMPO REAL EM DISPOSITIVOS MÓVEIS: O IMPACTO DE SE OTIMIZAR O USO DE MEMÓRIA

Dissertação submetida à Universidade Federal de Santa Catarina para a obtenção do Grau de Mestre em Ciência da Computação.
Orientador: Luiz Cláudio Villar dos Santos, Dr.
Coorientador: José Luís Almada Güntzel, Dr.

Florianópolis

2013

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Westphal, Rafael
Computação energeticamente eficiente sob restrições de tempo real em dispositivos móveis : o impacto de se otimizar o uso de memória / Rafael Westphal ; orientador, Luiz Cláudio Villar dos Santos ; co-orientador, José Luís Almada Güntzel. - Florianópolis, SC, 2013.
92 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Ciência da Computação.

Inclui referências

1. Ciência da Computação. 2. Sistemas embarcados. 3. Eficiência energética. 4. Tempo-real. 5. Hierarquia de memória. I. Santos, Luiz Cláudio Villar dos. II. Güntzel, José Luís Almada. III. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Ciência da Computação. IV. Título.

Rafael Westphal

COMPUTAÇÃO ENERGETICAMENTE EFICIENTE SOB RESTRIÇÕES DE TEMPO REAL EM DISPOSITIVOS MÓVEIS: O IMPACTO DE SE OTIMIZAR O USO DE MEMÓRIA

Esta Dissertação foi julgada aprovada para a obtenção do Título de "Mestre em Ciência da Computação", e aprovada em sua forma final pela Universidade Federal de Santa Catarina.

Florianópolis, 22 de fevereiro 2013.

Ronaldo dos Santos Mello, Prof. Dr.
Coordenador

Luiz Cláudio Villar dos Santos, Dr.
Orientador

José Luís Almada Güntzel, Dr.
Coorientador

Banca Examinadora:

Djones Vinicius Lettnin, Dr.

Rômulo Silva de Oliveira, Dr.

Rodolfo Jardim de Azevedo, Dr.

À minha família.

AGRADECIMENTOS

Aos meus pais, Sebastião e Lúcia, pela educação, formação e suporte incondicional.

Também ao meu irmão Diego, pelo apoio e compreensão.

A minha irmã Daniela que, mesmo estando longe, me apoiou durante este trajeto.

À Gabriela, minha namorada, pelo carinho, dedicação, paciência e compreensão nas diversas etapas deste mestrado, em especial na reta final.

Ao meu orientador, Professor Dr. Luiz Cláudio Villar dos Santos, pelas contribuições técnicas, pela orientação ao longo do mestrado e pela amizade.

Ao meu coorientador, Professor Dr. José Luís Almada Güntzel, pela coorientação, sugestões, críticas e importante contribuições para melhoria deste trabalho.

Aos membros da banca, Professor Dr. Djones Vinicius Lettnin, Professor Dr. Rômulo Silva de Oliveira e Professor Dr. Rodolfo Jardim de Azevedo, por aceitarem o convite para avaliar este trabalho e pelas contribuições.

Aos colegas de LAPS/ECL e NIME, pelas discussões, críticas, sugestões e momentos de descontração. Em particular ao meu amigo Daniel Pereira Volpato pela contribuição na infraestrutura experimental e colaboração direta.

Ao CNPq, pelo apoio financeiro concedido sob forma de bolsa de mestrado.

A mente que se abre a uma nova idéia jamais voltará ao seu tamanho original.

Albert Einstein

RESUMO

Um dispositivo móvel é composto essencialmente de dois subsistemas (um "PC" e um "rádio"), os quais são geralmente implementados como distintos sistemas-em-chip multiprocessados (MPSoC). O "PC" suporta processamento multimídia e implementa a interface com o usuário fazendo uso de computação *multi-thread* sob gerenciamento de um sistema operacional complexo; o "rádio" realiza processamento de banda-base por meio de computação multi-tarefa gerenciada por um sistema operacional de tempo real. O aumento das taxas de transferência e dos requisitos de segurança demandam um crescimento da vazão sob uma restrição de potência que vem se mantendo quase inalterável para dispositivos móveis. Para lidar com a crescente demanda por eficiência energética, arquiteturas *multicore* são utilizadas para processamento da pilha de protocolo e aplicações de segurança, o qual requer computação multi-tarefa sob restrições de tempo real. Esta dissertação aborda a eficiência energética de computação multi-tarefa em arquiteturas *multicore* homogêneas por meio do uso racional do subsistema de memória.

A dissertação caracteriza, sob restrições de tempo real, o consumo de energia, a vazão e a eficiência energética do subsistema de memória para uma importante classe de aplicações em dispositivos móveis: os algoritmos criptográficos. Os resultados mostram que os requisitos para escalonabilidade de tempo real limitam o crescimento da vazão com o aumento do tamanho da cache de tal forma que o consumo extra da cache não vale a pena. Também mostram que, no contexto da classe de aplicações-alvo, mais de 85% da energia gasta no subsistema de memória é devido ao consumo dinâmico. Tais resultados permitiram identificar otimizações de cache cruciais para lidar com o aumento das restrições de eficiência energética.

Posteriormente, para um dado conjunto de tarefas periódicas, esta dissertação propõe uma nova abordagem para otimizar a eficiência energética do subsistema de memória (um grande responsável pelo consumo de energia em dispositivos baseados em SoC), a qual leva em conta restrições de tempo real. O problema-alvo é decomposto em dois subproblemas fracamente acoplados: particionamento de tarefas entre *cores* e particionamento dos itens de memória entre espaços de endereçamento cacheáveis e não-cacheáveis. Tal abordagem reduz o consumo dinâmico sem causar um aumento significativo no tempo médio de execução ou no consumo estático. Experimentos realizados

com tarefas de um protocolo de segurança real mostraram que reduções entre 7,7% e 33% da energia consumida pelo subsistema de memória pode ser obtida com a decomposição proposta, em comparação com um sistema de referência que mantém todos os itens de programa em um espaço de endereçamento cacheável.

ABSTRACT

A mobile device is essentially a combination of two subsystems (a "PC" and a "radio"), which are often implemented as distinct multiprocessor systems-on-chip (MPSoC). The former supports multimedia processing and implements the end-user interface by relying on multi-thread computing under the management of a complex operating system; the latter implements baseband processing by performing multi-task computing under the management of a real-time operating system. Increasing data rates and security requirements ask for rising throughputs under the nearly unchanging power constraints imposed to mobile devices. To cope with the ever growing demand for energy efficiency, multicore architectures are used for protocol stack and security processing, which require multi-task computing under real-time constraints. This dissertation addresses the energy efficiency of multi-task computing on homogeneous multicore architectures through the rational use of the memory subsystem.

First, the dissertation reports, under real-time constraints, the energy consumption, the throughput, and the energy efficiency of the memory subsystem for an important class of applications within a mobile device: the cryptographic algorithms. The results show that real-time schedulability requirements limit the growth of throughput with increasing cache size in such a way that the extra cache consumption does not pay off. They also show that, in the context of multi-task computing, more than 85% of the energy spent in the memory subsystem is due to dynamic consumption. Such results allowed us to identify crucial cache optimizations to cope with increasing energy efficiency requirements.

Then, for a given set of periodic tasks, the dissertation proposes a new approach to optimize the energy efficiency of the memory subsystem (a major energy consumer in SoC-based devices) while taking into account real-time constraints. The target problem is decomposed into two loosely coupled subproblems: the partitioning of tasks among cores and the partitioning of memory items between cached and uncached address spaces. The approach reduces the dynamic energy consumption with no significant increase in average execution time and static consumption. Experiments performed with tasks from a real-life security protocol showed that memory energy reductions from 7.7% up to 33% can be obtained with the proposed decomposition, as compared to a baseline system that keeps all program items in cached address space.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

LISTA DE SÍMBOLOS

SUMÁRIO

1 INTRODUÇÃO

Computação Móvel compreende, essencialmente, interface com o usuário, processamento multimídia e processamento de banda-base. Uma vez que continua crescendo o uso de dispositivos pessoais móveis para acesso à Computação em Nuvem, a demanda para comunicação de dados em alta velocidade aumenta. Com isso, o processamento de banda-base (modulação/demodulação, processamento da pilha do protocolo de comunicação e de aplicações de segurança) passa a ter um papel-chave nos requisitos de eficiência energética. Enquanto as tecnologias dos dispositivos pessoais móveis (*personal mobile devices* - PMDs) para acesso a rádio evoluem de 3G para 4G, espera-se que a taxa de transferência de dados cresça de 14 Mbps para 100Mbps ou além, exigindo de 40 a 290 GOPS, respectivamente, ao passo que a dissipação máxima de potência precisa ficar restrita a cerca de 1 Watt (DALLY et al., 2008). Como resultado, a eficiência energética tem que crescer de 40 MOPS/mW para 290 MOPS/mW. Já que uma relaxação significativa da restrição de potência não é viável, a eficiência energética necessária pode vir ser uma ordem de magnitude superior para os futuros padrões com taxas da ordem de Gbps (*e.g. LTE-Advanced*). Entretando, o aumento dos requisitos de vazão não advém apenas do aumento das taxas de transferência, mas também da necessidade de proteger a comunicação sem fio. Quando a criptografia é utilizada, a transferência de dados é dividida em sessões (POTLAPALLY et al., 2006). Por exemplo, se o volume de dados a ser transferido em uma sessão é restrito a 1KB para um alto nível de segurança, 122K seções por segundo devem ser gerenciadas para uma taxa de transferência de 1Gbps. Em outra palavras, a cada período de 8,2 μs, cifradores simétricos e assimétricos precisam ser invocados, algumas tarefas não criptográficas tem que ser executadas e os dados criptografados devem ser transferidos.

Como mostra a Figura 1, um PMD é composto essencialmente por dois subsistemas (um "PC" e um "rádio"), os quais são geralmente implementados em distintos sistemas-em-*chip* multiprocessados (ARM HOLDINGS, 2011). O "PC" suporta processamento multimídia (GPU), implementa a interface com o usuário e executa suas aplicações (processador *multicore* homogêneo). Ele adota computação *multi-thread* gerenciada por um sistema operacional complexo que suporta memória virtual no topo de uma hierarquia com múltiplos níveis de cache. Já o "rádio" realiza processamento de modem (processador DSP/VLIW),

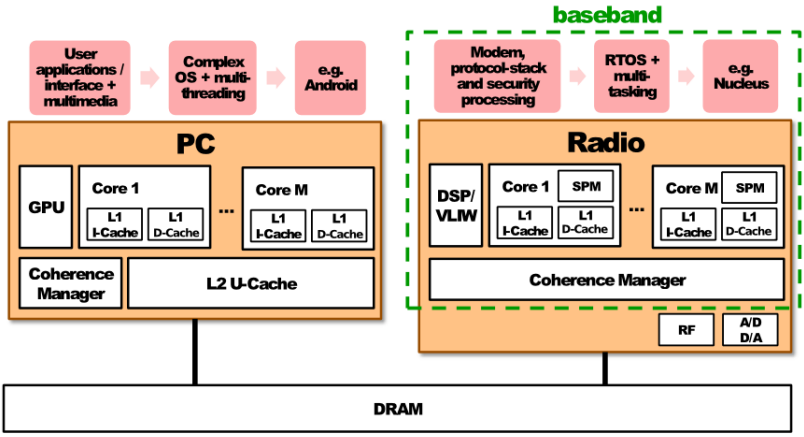


Figura 1: A anatomia de um dispositivo móvel (TURNER, 2011)

processamento da pilha do protocolo de comunicação e de aplicações de segurança (processador *multicore* homogêneo). Ele adota computação multi-tarefa gerenciada por um sistema operacional de tempo real (RTOS). Para lidar com os requisitos de tempo real, no "rádio", memórias de rascunho (SPM[1]) privativas geralmente coexistem com um único nível de memórias caches privativas.

O processamento da pilha do protocolo de comunicação e das aplicações de segurança consiste de um conjunto complexo de tarefas periódicas que correspondem a vários *megabytes* de código. Para manter uma comunicação sem perda de dados, tais tarefas precisam atender a restrições de tempo real que, por sua vez, ficam cada vez mais rígidas com o aumento das taxas de transferência e do nível de segurança. Enquanto a modulação/demodulação ocorre em um processador DSP/-VLIW de aplicação específica, o processamento da pilha do protocolo de comunicação e das aplicações de segurança é realizado em um processador *multicore* (e.g. Cortex-R da ARM e Snapdragon da Qualcomm). De acordo com estimativas dos fornecedores, transceptores operando a 1Gbps precisarão de pelo menos dois *cores* emitindo até duas instruções por ciclo à 1GHz (TURNER, 2011).

[1]Do inglês, *scratchpad memory*

1.1 A PLATAFORMA ALVO

A Figura 2 ilustra a classe de arquiteturas-alvo deste trabalho. Ela pode ser vista quer como o subsistema de banda-base de um único MPSoC (como no caso do Snapdragon) ou como um MPSoC independente realizando o papel de um processador de banda-base (como no caso do Cortex-R).

Ela consiste de ***M*** *cores* idênticos para processamento da pilha do protocolo de comunicação e das aplicações de segurança e de um processador DSP/VLIW para modulação/demodulação (ARM HOLDINGS, 2011). No primeiro nível da hierarquia, cada *core* possui três memória locais (privativas): uma cache de instruções (I-cache), uma cache de dados (D-cache) e uma memória de rascunho fortemente acoplada. No segundo nível, assumiu-se uma memória global compartilhada (SM[2]), a qual pode ser tanto uma cache nível dois unificada ou a memória principal.

Assume-se que os endereços de programa correspondem aos endereços físicos ou que a política de tradução é simples (como retirar alguns bits do endereço) e portanto, não requer uma unidade de gerenciamento de memória (MMU[3]). Assume-se que o espaço de endereçamento físico da memória está particionado em três subespaços. A partição de ***Boot*** é reservada para rotinas que devem exibir um comportamento previsível a partir do *reset* do sistema, como rotinas de inicialização e registradores de entrada/saída mapeados em memória. A partição **cacheável** contém os itens que são acessados via caches. Finalmente, a partição **não-cacheável** contém os itens que requerem acessos à memória com baixa latência e alta previsibilidade (tais como código de instruções para as rotinas de interrupção e dados que necessitam de um processamento intenso). Assume-se que as memórias de rascunho locais são mapeadas para a partição não-cacheável do espaço de endereçamento global.

Inspirados nos atuais MPSoCs para processamento de banda-base, os experimentos apresentados nesta dissertação assumem (sem perda de generalidade) um único nível de caches, i.e. somente a memória principal é compartilhada por todos os processadores. Como as aplicações-alvo (algoritmos criptográficos) não compartilham variáveis e podem ser modeladas como tarefas que executam independentemente (exceto talvez por restrições de precedência para o consumo do resultado final de uma tarefa por uma outra tarefa), por simplicidade, o protocolo de coerência não é modelado na nossa configuração experimental. Esta simplificação

[2]Do inglês, *shared memory*
[3]Do inglês, *memory management unit*

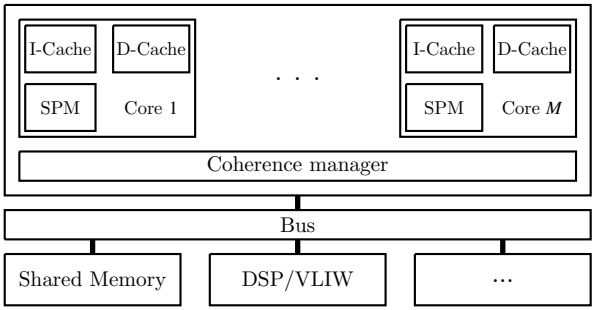


Figura 2: Um modelo de arquitetura para processamento da pilha de protocolo e aplicações de segurança (ARM HOLDINGS, 2011)

implicitamente assume que as tarefas de um protocolo de segurança raramente compartilham variáveis, embora possam obedecer a restrições de precedência.

1.2 A RELEVÂNCIA DAS CACHES PRIVATIVAS

O impacto das tecnologias de fabricação na distribuição de consumo de caches individuais já foi estudado por Rodriguez e Jacob (2006). Como aquele trabalho deliberadamente assume 100% de atividade de chaveamento, os valores reportados representam um limite superior para o efetivo consumo de potência. Embora útil para antever futuras escolhas para organização do *hardware*, aquele estudo não permite identificar otimizações de código promissoras para melhorar a eficiência energética, pois uma caracterização energética apropriada depende dos padrões de acesso à memória. Apesar da disponibilidade de vários modelos de energia baseados em *traces* (por exemplo, (HILL et al., 1993)), os resultados de composição de consumo de energia reportados na literatura limitam-se a caches individuais e não se conhecem relatórios sobre a composição de consumo de múltiplas caches privativas.

Uma caracterização da composição do consumo de energia em caches privativas seria provavelmente inconclusiva para aplicações baseadas em computação *multi-thread* (como no subsistema "PC" de um dispositivo móvel), devido à diversidade das cargas de trabalho dinâmicas. Entretanto, este não é o caso de aplicações baseadas em computação multi-tarefa (como no subsistema "rádio"), porque as cargas de trabalho

são definidas em tempo de projeto (de acordo com padrões de comunicação e de segurança) e muitas tarefas são periódicas (devido a taxas de dados pré-especificadas). Uma tal caracterização de caches privativas justifica-se pelo fato de que a contribuição das caches primárias para o consumo de energia é bastante significativa. Ela pode representar cerca de 50% da energia consumida por um processador embarcado (DALLY et al., 2008). Embora a energia consumida quando ocorre falta na cache possa ser uma ordem de magnitude superior à energia consumida quando ocorre um acerto, seu impacto para o consumo de energia total é relativamente pequeno para caches adequadamente projetadas, pois é atenuado pela taxa de faltas. Por estas razões, esta dissertação aborda a caracterização energética das caches primárias, pois elas provavelmente dominarão a otimização de software embarcado para eficiência energética.

1.3 ESCOPO

Esta dissertação aborda o problema de otimização da **eficiência energética** em sistemas multiprocessados sob **restrições de tempo real** para uma importante instância de **computação multi-tarefa**: o processamento associado a **aplicações de segurança**, em particular os **algoritmos criptográficos**.

Inicialmente, para permitir a identificação de otimizações de código promissoras, a dissertação reporta a **caracterização do consumo de energia de caches privativas sob restrições de tempo real**.

Posteriormente, a dissertação propõe uma **decomposição** do problema de maximizar a eficiência energética do subsistema de memória em dois subproblemas fracamente acoplados:

- **Particionamento das tarefas em *cores***: Dado um conjunto de tarefas periódicas e o número de *cores*, determinar um particionamento que satisfaça as restrições de tempo real.

- **Particionamento de elementos de programa em memória**: Dado um *core*, o conjunto de tarefas a ele assinaladas e a capacidade de sua memória de rascunho privativa, determinar o particionamento dos elementos de programa das tarefas em regiões cacheáveis e não cacheáveis, com o objetivo de minimizar o consumo de energia.

1.4 CONTRIBUIÇÕES

São as seguintes as principais contribuições desta dissertação:

- **Identificação de otimizações de código promissoras via caracterização energética:** Ao utilizar programas reais de criptografia, extraídos de um bem conhecido protocolo de segurança (OPRN, 2001), a dissertação reporta a composição do consumo de energia para caches privativas que são representativas dessa importante instância de computação multi-tarefa. Dados obtidos por meio de simulação com uma plataforma executável e fazendo uso de estimativas de energia por acesso e tempo de latência para diversas configurações de memória, mostraram que a cache de instruções domina o consumo de energia do subsistema de memória (em média 65% do consumo total). Esse resultado de caracterização permitiu identificar que otimizações que reduzem o consumo sem aumentar o tempo médio de execução são fundamentais para a classe de aplicações escolhida.

- **Melhoria da eficiência energética através de dois particionamentos:** Diferentemente das técnicas correlatas, esta dissertação resolve o mapeamento energeticamente eficiente de dados e instruções para distintas memórias privativas **levando em conta as restrições de tempo real**. Esse mapeamento é obtido através da decomposição daquele mapeamento em dois subproblemas de particionamento de forma a recair em soluções conhecidas. Embora, a solução proposta não ofereça garantias de tempo real (pois nada se provou sobre o impacto da otimização sobre o tempo de execução de pior caso (WCET[4])), sua consciência das restrições de tempo real aumenta as chances de se garantir escalonabilidade em uma análise de WCET posterior à otimização.

1.5 REPERCUSSÕES E EXPECTATIVAS

Os resultados obtidos com a caracterização apresentada no Capítulo 3 já tiveram seu mérito reconhecido pelo Comitê de Programa da *IEEE International Conference on Electronics, Circuits, and Systems* (ICECS 2012), resultando na publicação de trabalho intitulado *Energy-Efficient Multi-Task Computing on MPSoCs: a Case Study*

[4]Do inglês, *worst-case execution time*

from a Memory Perspective nos anais daquele evento (WESTPHAL; GÜNTZEL; SANTOS, 2012).

1.6 ORGANIZAÇÃO DESTA DISSERTAÇÃO

O Capítulo 2 introduz os modelos adotados nesta dissertação. O Capítulo 3 apresenta um estudo detalhado da eficiência energética em sistemas multiprocessados executando algoritmos criptográficos sob o modelo de computação multi-tarefa com restrições de tempo real. O Capítulo 4 descreve a técnica de otimização de código proposta para aumentar a eficiência energética do subsistema de memória, obedecendo às restrições de tempo real. O Capítulo 5 apresenta as conclusões e propõe trabalhos futuros para suplantar algumas das atuais limitações.

2 O MODELO DO SISTEMA

2.1 CARACTERIZAÇÃO DAS TAREFAS

Assumiu-se que a aplicação foi dividida em tarefas a serem gerenciadas por um RTOS. Entre as tarefas, algumas são periódicas (aquelas executadas a cada P_i unidades de tempo), enquanto outras são esporádicas (aquelas que requisitam o processador em momentos imprevisíveis). Assume-se também que as tarefas esporádicas são transformadas em tarefas periódicas por meio de um servidor de tarefas esporádicas i.e., uma tarefa periódica que verifica se existem tarefas esporádicas a serem disparadas[1].

Assume-se ainda que os programas de uma dada aplicação são representados como um conjunto de tarefas independentes $\tau = \{\tau_1, \ldots, \tau_n\}$. Uma tarefa τ_i é caracterizada pela tripla (C_i, P_i, ϕ_i), onde C_i é o tempo de computação (o qual é limitado pelo tempo de execução de pior caso - WCET), P_i é o período e ϕ_i é o *offset* (deslocamento em relação ao início do período). O período é implicitamente assumido como igual à *deadline* (tempo limite). Se $\phi_i = 0$ para $i = 1, \ldots, n$, o conjunto de tarefas é dito síncrono, caso contrário, é dito assíncrono. Note que a representação adotada permite a modelagem de restrições de precedência entre tarefas. Dada uma série de tarefas, atribui-se à primeira tarefa um *offset* zero e ao *offset* da segunda tarefa o valor do instante de tempo em que a tarefa anterior foi completada. Além disso, esta representação pode ser estendida para lidar com sincronização e compartilhamento de recursos (PELLIZZONI; LIPARI, 2005).

2.2 CARACTERIZAÇÃO DOS PROGRAMAS

Cada tarefa acessa itens de memória que correspondem a elementos de programa tais como procedimentos, dados estáticos e dados dinâmicos. Uma vez que se deseja estender o escopo da otimização para bibliotecas e código pré-compilado protegido por propriedade intelectual

[1] Poderia ser argumentado que algumas tarefas do processamento de banda-base são, na verdade, esporadicamente periódicas, já que quando não existe transmissão ou recepção o modem estaria ocioso. Uma vez que a análise de escalonabilidade de tarefas periódicas está mais madura do que a analise de escalonabilidade de tarefas periodicamente esporádicas, este trabalho assume que as tarefas periodicamente esporádicas são tratadas como tarefas periódicas regulares, de modo que o seu servidor consegue ativá-las ou desativá-las sob demanda.

(IP) sem a necessidade de utilizar uma unidade de gerenciamento de memória para tirar proveito das memórias de rascunho (EGGER et al., 2010), este trabalho concentra-se nos elementos de programa que podem ser extraídos de arquivos binários, i.e. procedimentos e dados estáticos[2].

Uma vez que a eficiência energética está associada ao consumo médio de energia, sua estimativa requer a caracterização do tempo de execução do caso médio (ACET). O padrão de acessos à memória correspondente ao cenário de ACET é capturado por um *trace*. Um *trace* é uma tupla $T = (\alpha_1, \cdots, \alpha_i, \cdots, \alpha_n)$ que representa uma sequência de sucessivos endereços de memória que caracterizam o comportamento no cenário de ACET.

Seja D_i a representação de um elemento de programa (variável, estrutura de dados, ou procedimento) e w_i seu tamanho (em bytes).O vetor $W = [w_1, \cdots, w_i, \cdots, w_n]$ é utilizado para representar os tamanhos de uma coleção de elementos de programa $D_1, \cdots, D_i, \cdots, D_n$, os quais são referenciados por um dado *core*.

2.3 MODELO DO CONSUMO DE ENERGIA

Seja a o tipo de acesso à memória, o qual pode ser uma escrita (W) ou uma leitura (R). Seja t o tempo médio execução de um programa. Seja ainda $M \in \{I-cache, D-cache, SPM, SM\}$ uma componente de memória do sistema de capacidade K_M. Denota-se a energia gasta pela memória M para realizar um acesso do tipo a a uma de suas posições por e^a_M, o número de acessos do tipo a à memória M por n^a_M e a potência estática da memória M por P_M. Inicialmente, os valores de e^a_M e P_M são obtidos através do modelo de potência/energia CACTI 6.5 (THOZIYOOR et al., 2008) ao passo que os valores n^a_M são obtidos através do monitoramento da execução do programa em um simulador.

Dado um programa p, seu tempo de execução t e a potência estática P_M de uma dada memória, calcula-se a energia total consumida pela memória M, discriminando-a em componentes estático (E^s_M) e dinâmico (E^d_M), conforme as equações 2.1,2.2 e 2.3.

$$E^d_M = n^R_M \times e^R_M + n^W_M \times e^W_M \tag{2.1}$$

[2]Como consequência, os salvamentos e a restaurações de contexto resultantes da preempção não se beneficiam das otimizações de memória porque eles envolvem itens de pilha. Entretanto, as garantias de tempo real podem ser preservadas, já que o *overhead* da preempção pode ser capturado como parte do tempo de computação de cada tarefa.

$$E_M^s = P_M \times t \tag{2.2}$$

$$E_M = E_M^d + E_M^s \tag{2.3}$$

Assim, a energia total gasta pelas memórias cache privativas é dada por:

$$E_{p-caches} = E_{I-cache} + E_{D-cache} \tag{2.4}$$

Finalmente, a energia total gasta no subsistema de memória é calculada por:

$$E = E_{p-caches} + E_{SPM} + E_{SM} \tag{2.5}$$

Ao definir o lucro decorrente de uma otimização, em certas situações é conveniente expressar a energia *média* por acesso, a fim de evitar a distinção entre a acessos de escrita e acessos leitura. Assim, e_M denota a energia média gasta pela memória M para ler ou escrever uma posição da memória. Poderia se argumentar que a decisão de utilizar a energia média não é adequada, porém, o impacto nos resultados obtidos não foi significativo visto que para as memórias que foram utilizadas tal valor entre escrita e leitura era muito semelhante.

2.4 PARTICIONAMENTO DE MEMÓRIA E CARACTERIZAÇÃO DO LUCRO

Para representar um particionamento dos elementos de programa entre os espaços cacheável e não-cacheável, adota-se um vetor binário $X = [x_1, \cdots, x_i, \cdots, x_n]^{-1}$, onde $x_i = 0$ e $x_i = 1$ denotam, respectivamente, o assinalamento de um elemento de programa D_i ao espaço de endereçamento cacheável ou ao espaço de endereçamento não-cacheável.

Seja a_i o número de acessos ao elemento de programa D_i em um *trace* T.

Seja m_i o número de vezes que D_i não foi encontrado na cache (i.e., o número de *misses* no acesso a D_i) para o trace T. Quando o elemento D_i é assinalado ao espaço de endereçamento cacheável ($x_i = 0$), cada acesso a ele consome uma quantidade de energia igual a:

$$e_i = e_{cache} + m_i \times e_{SM} \tag{2.6}$$

Por outro lado, quando o elemento D_i é assinalado ao espaço de endereçamento não-cacheável ($x_i = 1$), cada acesso a ele consome uma quantidade de energia referenciada por e_{SPM}.

Como resultado, o lucro resultante de se atribuir D_i para o espaço não-cacheável é dado por $p_i = a_i \times (e_i - e_{SPM})$. Seja o vetor $P = [p_1, \cdots, p_i, \cdots, p_n]$ a representação dos lucros individuais de todos os elementos de programa referenciados por um dado *core*. O lucro total de um dado particionamento de memória X é $p(X) = P \times X$.

2.5 A ESCOLHA DE ESCALONADORES-PARTICIONADORES

Caso se considere que as tarefas possam migrar de um processador para outro, será necessário considerar as penalidades decorrentes. Por exemplo, se uma tarefa é disparada em um processador e então preemptada, seu contexto precisa ser restaurado antes que sua execução seja retomada em outro processador. Isto não apenas resulta em carga de comunicação, como também pode induzir faltas adicionais na cache (DAVIS; BURNS, 2011), cujo impacto no consumo dinâmico não pode ser negligenciado[3]. Assim, a migração de tarefas tende a reduzir o desempenho e aumentar o consumo de energia e portanto, está em desacordo com os crescentes requisitos de eficiência energética das aplicações-alvo. Além disso, ela reduz a dependabilidade (se uma tarefa extrapola a sua *deadline*, então ela não afeta as tarefas executadas em outros processadores (DAVIS; BURNS, 2011)).

Considerando-se o exposto no parágrafo anterior, para contribuir com a robustez e a eficiência energética, este trabalho assume que as tarefas não migram entre processadores. Como consequência, o subconjunto de tarefas assinalado para um *core* é uma **partição** do conjunto de tarefas, o que permite o reuso de políticas de escalonamento, análise de escalonabilidade e técnicas de otimização de código que foram originalmente desenvolvidas para processadores com um único *core*. Por isso, adotou-se a classe de algoritmos denominada de escalonadores-particionadores (DAVIS; BURNS, 2011). Um algoritmo de tal classe deve primeiro realizar o particionamento das tarefas entre os processadores para depois escalonar de forma independente as tarefas assinaladas a cada processador. Assim sendo, um algoritmo desta classe é concebido pela composição de um algoritmo de particionamento e um algoritmo de escalonamento para sistema monoprocessado.

Algoritmos de escalonamento costumam empregar a noção de utilização de uma tarefa τ_i, definida como $u_i = C_i/P_i$, para testar se, ao

[3]Mesmo assumindo uma cache nível dois com associatividade alta para evitar demasiados acessos à memória principal, a potência dinâmica ainda representa entre 30% e 50% do consumo total da cache de nível dois, mesmo em tecnologia de 32nm (RODRIGUEZ; JACOB, 2006)

escalonar uma tarefa no sistema, nenhuma outra tarefa irá perder sua *deadline*. Desta forma, dado o subconjunto de tarefas assinaladas a um processador p, representado por τ^p, sua utilização total, usada para avaliar a qualidade do assinalamento das tarefas, é definida como:

$$U(\tau^p) = \sum_{\tau_i \in \tau^p} u_i. \tag{2.7}$$

2.5.1 Análise de Factibilidade e de Escalonabilidade

Os dois métodos mais empregados para realizar o escalonamento de um conjunto de tarefas independentes em um processador são: *Rate-monotonic* (RM) e *Earliest Deadline First* (EDF). Um estudo extensivo sobre as vantagens de cada um destes métodos é apresentado em (BUTTAZZO, 2005).

Um teste de factibilidade é um algoritmo que, dadas as especificações de um sistema de tarefas e restrições no ambiente de escalonamento [4], determina se existe um escalonamento para o sistema de tarefas que respeite todas as *deadlines* (BARUAH; GOOSSENS, 2004), ou em outras palavras, determina se ele é factível.

Um algoritmo de escalonamento é dito **ótimo**, com respeito a um sistema e a um modelo de tarefas, se ele consegue escalonar todos os conjuntos de tarefas que obedecem ao modelo de tarefas e que são factíveis no sistema (DAVIS; BURNS, 2011).

Sendo assim, o problema de determinar se um conjunto de tarefas é factível para um determinado modelo de tarefas pode ser reduzido ao problema de determinar a escalonabilidade de um conjunto de tarefas dado um algoritmo de escalonamento ótimo.

2.5.2 Critérios de escalonamento

2.5.2.1 O algoritmo *Rate-Monotonic* (RM)

O algoritmo RM atribui às tarefas prioridades que são inversamente proporcionais aos seus respectivos períodos ($1/P_i$). Em outras palavras, à tarefa com o menor período é atribuída a maior prioridade. Nesse sentido, RM utiliza assinalamento de prioridade estática.

[4] Por exemplo, se algoritmos de prioridade dinâmica são aceitáveis ou algoritmos de prioridade estática são requeridos; se preempção é permitida; etc.

Considerando um conjunto de tarefas independentes, síncronas e periódicas, o algoritmo RM é dito ótimo (LIU; LAYLAND, 1973). Ou seja, se existe um conjunto de tarefas que pode ser factivelmente escalonado em um único processador usando algum escalonamento com prioridade fixa, então esse conjunto é factível com o algoritmo RM (OH; SON, 1995). Uma condição de escalonabilidade necessária e suficiente foi encontrada para o algoritmo RM (LEHOCZKY; SHA; DING, 1989). Contudo, a complexidade de tempo ao utilizar tal condição para o algoritmo RM é dependente dos dados. No pior caso, a complexidade de tempo pode ser mais do que exponencial em relação ao número de tarefas (OH; SON, 1995). Oh e Son (1995) usa uma condição de escalonabilidade suficiente para o escalonamento das tarefas. Devido ao seu menor esforço computacional, esta dissertação adota essa condição suficiente. Entretanto, ao utilizar uma condição que é somente suficiente, todos os conjuntos de tarefas que são ditos escalonáveis, são realmente escalonáveis, mas (pela condição não ser necessária) nem todos os conjuntos de tarefas que são ditos não-escalonáveis são realmente não-escalonáveis (DAVIS; BURNS, 2011).

Enquanto o escalonamento com o algoritmo RM é ótimo para sistemas monoprocessados com tarefas síncronas, infelizmente, não se pode afirmar o mesmo em se tratando de tarefas assíncronas. Além disso, ao se considerar um sistema multiprocessado, o problema de escalonar otimamente um conjunto de tarefas periódicas utilizando prioridade fixa ou prioridade dinâmica é sabido ser NP-completo (OH; SON, 1995).

2.5.2.2 O algoritmo *Earliest Deadline First* (EDF)

No algoritmo EDF, as prioridades são assinaladas dinamicamente sendo que a tarefa cuja *deadline* está mais próxima do instante atual recebe maior prioridade. Considerando um conjunto de tarefas síncronas, independentes e periódicas, o algoritmo EDF é ótimo, no sentido de um algoritmo de escalonamento ótimo (STANKOVIC; RAMAMRITHAM; SPURI, 1998). Particularmente, qualquer conjunto de tarefas síncronas com *deadlines* iguais ao seus respectivos períodos, é factivelmente escalonado pelo EDF se e somente se a utilização do processador é tal que $U(\tau^p) = \sum_{i=1}^{n} u_i \leq 1$ (STANKOVIC; RAMAMRITHAM; SPURI, 1998). Se o conjunto de tarefas é assíncrono, o algoritmo EDF também é sabido ser ótimo, mas o teste de factibilidade, entretanto, não é (STANKOVIC; RAMAMRITHAM; SPURI, 1998).

2.5.3 Critérios de particionamento

2.5.3.1 Heurística *First-Fit*

Conforme já mencionado no início da presente seção, este trabalho adota algoritmos escalonadores-particionadores. Assim, uma primeira solução resulta da composição do algoritmo RM com a heurística de particionamento *First Fit Decreasing Utilization* (FFDU), que costuma ser denominada de RM-FFDU(OH; SON, 1995). A heurística FFDU é uma variante da já conhecida heurística para o Problema *bin-packing*, a qual resolve o problema de empacotar itens de tamanhos variados em caixas de tamanho fixo utilizando o menor número possível de caixas (Coffman, Jr.; GAREY; JOHNSON, 1984).

2.5.3.2 Particionamento ótimo

Como segunda opção para particionamento, escolheu-se a composição do algoritmo EDF com um particionamento baseado na abordagem Branch and Bound (B&B)(MARTELLO; TOTH, 1990), que será denominada de EDF-B&B. A escolha do B&B deveu-se ao fato de ser um método exato que permite encontrar a solução ótima quando combinada com o algoritmo EDF, devido ao baixo custo computacional da condição de escalonabilidade do EDF.

3 EFICIÊNCIA ENERGÉTICA DA CACHE EM COMPUTAÇÃO MULTI-TAREFA

Apesar de o impacto da evolução do processo de fabricação nos componentes da potência de caches individuais já ter sido abordado (e.g. (RODRIGUEZ; JACOB, 2006)) anteriormente, a caracterização dos componentes da energia de múltiplas caches privativas ainda não foi reportada na literatura, provavelmente porque ela depende dos padrões de acesso à memória, os quais, por sua vez, são influenciados pela divisão da carga de processamento entre os *cores*. Por um lado, tal caracterização não seria conclusiva para o subsistema de um PMD com computação *multi-thread* devido à diversidade das cargas de trabalho, as quais podem ser redefinidas pelo usuário a qualquer instante. Por outro lado, é relevante para o subsistema de computação multi-tarefa porque o conjunto de tarefas é definido em tempo de projeto (uma vez que as tarefas precisam estar em conformidade com os padrões de comunicação e segurança) e em sua maioria são periódicas (devido a taxas de dados pré-especificadas). Entretanto, vale ressaltar que tais cargas de trabalho poderem variar dinamicamente dependendo da escolha de um entre os vários padrões (e.g., cliente e servidor podem concordar quanto ao subconjunto de cifradores para transações distintas).

Tal caracterização das componentes da energia de múltiplas caches justifica-se, dado que mais de 50% da energia consumida por um *core* deve-se aos acessos às caches primárias de dados e de instruções (DALLY et al., 2008). Embora a energia consumida quando ocorre uma falta na cache possa ser uma ordem de magnitude maior do que aquela resultante de um acerto na cache, seu impacto para o consumo de energia total é relativamente pequeno para caches adequadamente projetadas, pois é atenuado pela taxa de faltas. Sendo assim, as avaliações foram realizadas sob a **perspectiva do subsistema de memória** para um *template* de MPSoC adequado ao processamento de banda-base.

Foram avaliadas diferentes configurações de memória para arquiteturas *dual-core*, *triple-core* e *quad-core* utilizando cargas de trabalho que simulam o lado do cliente em uma conexão SSL(OPRN, 2001). Para induzir padrões de acessos à memória típicos de computação multi-tarefa, foram empregados os algoritmos escalonadores-particionadores RM-FFDU e EDF-B&B para balancear as cargas entre os *cores* e escalonar as tarefas sob restrições de tempo real. Para avaliar o impacto em um escopo abrangente de tecnologias, a caracterização foi realizada para os nodos tecnológicos de 90nm, 65nm, 45nm e 32nm.

O restante deste capítulo esta organizado como segue. A Seção 3.1 revisa brevemente os trabalhos correlatos que estão mais relacionados com o presente trabalho. A configuração experimental está descrita na Seção 3.3 e a análise dos resultados na Seção 3.4.

3.1 TRABALHOS CORRELATOS

A eficiência energética de um MPSoC resulta da eficiência da potência dos seus componentes físicos (e.g., caches, memória principal), da otimização dos binários (e.g., particionamento de dados e código entre memórias cache e memórias de rascunho), do mecanismo de balanceamento de carga adotado pelo *middleware* ou pelo RTOS (e.g., algoritmo de escalonamento de tarefas), do modelo de programação adotado pela aplicação (e.g., memória compartilhada ou envio de mensagem) etc. Uma visão global dos desafios de projeto do *hardware* e do software pode ser encontrada em (WOLF; JERRAYA; MARTIN, 2008), onde aplicações que requerem o uso de MPSoCs estão categorizadas, as arquiteturas mais comuns são revisadas e os principais problemas são discutidos.

No nível do software de aplicação, o impacto na eficiência energética dos modelos de programação e das arquiteturas que suportam tais modelos foi abordado por Poletti et al. (2007).

No nível de RTOS e *middleware*, vários algoritmos foram propostos para tratar o problema de escalonar um conjunto de tarefas periódicas em sistemas multiprocessados, tal que as *deadlines* são garantidamente satisfeitas em cada processador (DAVIS; BURNS, 2011). Para tratar as prioridades das tarefas, o critério *Rate-Monotonic* (RM) é mais frequentemente utilizado devido à sua simplicidade. O algoritmo *Earliest Deadline First* (EDF) também é utilizado devido à alta taxa de utilização dos processadores por ele obtidas (BUTTAZZO, 2005). Para tratar do particionamento das tarefas, vários critérios foram propostos, tais como *First-Fit* (FF), *Best-Fit*, *Next-Fit*, *Worst-Fit*. Entretanto, elas são heurísticas e não atingem o resultado ótimo em todos os casos. Tais resultados ótimos podem ser encontrados utilizando uma abordagem *Branch & Bound*. A combinação RM-FFDU garante um dos melhores desempenhos teóricos (ZAPATA; ALVAREZ, 2005). Apesar de que o impacto da utilização de um algoritmo de escalonamento no consumo total de energia de um sistema multiprocessado de multi-tarefas aperiódicas já foi previamente abordado por (CONG; GURURAJ, 2009), tal trabalho não considera um sistema de tarefas periódicas, o qual é alvo

deste trabalho proposto.

No nível de hardware, o consumo de potência de memórias cache em tecnologias submicrônicas foi reportado por Rodriguez e Jacob (2006) detalhando as componentes dinâmicas e estáticas para nodos tecnológicos de 90nm, 65nm, 45nm e 32nm. Entretanto, os valores reportados por aquele trabalho representam um limite superior para o efetivo consumo de potência, visto que nele é assumido deliberadamente 100% de atividade de chaveamento. Tal caracterização é útil para antever futuras escolhas para organização do *hardware*, porém não permite identificar otimizações de código promissoras para melhorar a eficiência energética, pois uma caracterização energética apropriada depende de padrões de acesso à memória.

A falta da caracterização da **eficiência energética** para computação multi-tarefa em um MPSoC (tal como os que realizam processamento da pilha de protocolo e de aplicações de segurança em PMDs) (ARM HOLDINGS, 2011) motivou os experimentos realizados neste capítulo. A próxima seção resume a modelagem adotada nos experimentos.

3.2 FLUXO DE TRABALHO

Dada uma carga de trabalho composta por conjunto de tarefas criptográficas e um subsistema de memória pré-especificado, a Figura 3 descreve a instrumentação e as etapas necessárias para realizar a caracterização da energia de tal sistema.

Inicialmente, para cada uma das tarefas criptográficas τ_i, é estimado o WCET C_i, através de simulação, para cada tamanho de memória cache. Para realizar tal simulação são necessários os arquivos executáveis de cada tarefa.[1]

A partir do protocolo de comunicação utilizado (SSL) são obtidos o período P_i e offset ϕ_i de cada tarefa. Este processo será visto na Seção 3.3.1. Tais informações, junto com os valores de C_i previamente obtidos, são submetidas ao algoritmo escalonador-particionador (RM-FFDU ou EDF-B&B) que determina o particionamento e escalonamento das tarefas entre os processadores. Tal particionamento será visto na Seção 3.3.2.

A partir deste particionamento é realizada uma simulação, ob-

[1] Poderia ser argumentado que tal estimativa do WCET não é adequada. Entretanto, ela se deu devido às restrições impostas pela infra-estrutura experimental disponível.

tendo assim o trace T correspondente ao padrão de acessos induzidos pela execução das tarefas, a taxa de faltas m_{ij} induzida na memória cache para cada elemento de programa D_{ij} e o número de acessos a_{ij} de cada elemento de programa.

A partir da descrição do subsistema de memória, são estimadas a energia por acesso de escrita ou leitura e^a_M, bem como a potência estática P_M de cada memória. Para realizar tal estimativa é utilizada a ferramenta CACTI 6.5 (THOZIYOOR et al., 2008).

Finalmente, para possibilitar a caracterização da energia são necessários os arquivos objetos de cada uma das tarefas criptográficas, a fim de determinar quais elementos de programa (D_{ij}) contribuem no consumo de energia. Sendo assim, tais propriedades estimadas (e^a_M e P_M) junto com o resultado obtido na simulação (m_{ij}, a_{ij} e T) e os elementos de programa dos arquivos objetos (D_{ij}) permitem obter a caracterização da energia do conjunto de tarefas.

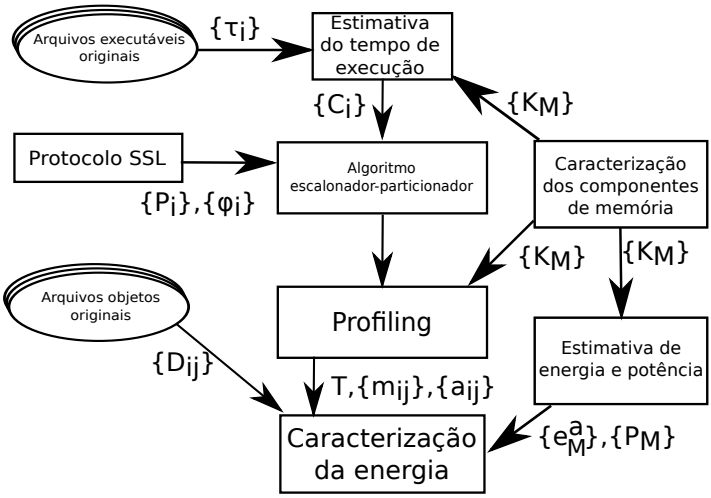


Figura 3: Fluxo proposto para caracterização da eficiência energética

3.3 CONFIGURAÇÃO EXPERIMENTAL

3.3.1 Caracterização das tarefas e escolha da carga de trabalho

Potlapally et al. (2006) reporta que entre 50–60% da energia gasta nas aplicações de segurança é devida aos algoritmos criptográficos. Sendo assim, foram selecionados programas do repositório OpenSSL (OPRN, 2001) para criar três cargas de trabalho realistas que simulam o lado do cliente em uma conexão SSL. Uma conexão SSL começa com um *handshake* entre o cliente e o servidor para estabelecer uma chave. Tal *handshake* requer a utilização de ambas as primitivas "Sign" e "Verify" de um cifrador assimétrico. O protocolo SSL define que cada pacote a ser transferido deve ser decriptado (*downstream*) ou encriptado (*upstream*) com um cifrador simétrico e então, submetido a um algoritmo de *hash*.

A Tabela 1 enumera os algoritmos de criptografia utilizados para criar as três cargas de trabalho adotadas como casos de uso. Utilizaram-se dois algoritmos assimétricos (RSA, DSA), dois algoritmos simétricos (AES, RC4) e dois algoritmos de *hash* (SHA-256, HMAC-SHA1). As primitivas "Sign" e "Verify" de cada cifrador assimétrico foram consideradas como uma única tarefa, ao passo que as primitivas "Encrypt" e "Decrypt" dos cifradores simétricos, assim como os algoritmos de *hash*, foram consideradas como tarefa individuais. A Tabela 1 também fornece o tempo de computação C_i para cada configuração de cache e o período P_i de cada tarefa. Para manter um determinada taxa de dados na transferência de pacotes, assumiu-se que todas as tarefas correspondentes aos cifradores simétricos e aos algoritmos de *hash* têm o mesmo período, o qual é calculado por $P_i = \frac{packet\_length}{data\_rate}$, onde $packet\_length$ = 4KBytes e $data\_rate$ = 28 Mbps (padrão HSPA+). Como consequência, o período das tarefas Sign+Verify foram definidos como $P_i = \frac{session\_length}{packet\_rate}$, onde $session\_length$ = 32KBytes. Para a instrumentação dos algoritmos criptográficos, foi utilizado um subconjunto do conjunto de dados especificados para validação de algoritmos criptográficos, propostos em (NIST, 2012). Para os algoritmos assimétricos, foram geradas chaves de 128bits. Já para os algoritmos simétricos, foram utilizados dados com tamanhos iguais ao *packet_length*. Por simplicidade, assumiu-se um conjunto de tarefas síncronas ($\phi_i = 0$) para os experimentos. As tarefas foram combinadas para criar cargas de trabalho que representam cenários distintos de conectividade: uma única conexão bidirecional (WL1), duas conexões unidirecionais (WL2), e uma conexão bidirecional simultânea com uma conexão unidirecional (WL3). As tarefas que compõem cada carga de

trabalho estão indicadas nas três últimas colunas da Tabela 1.

Tabela 1: Caracterização das cargas de trabalho

Tarefa		**Características[ms]**				**Cargas de trabalho**		
		C_i			P_i			
		8*kB*	16*kB*	32*kB*		**WL1**	**WL2**	**WL3**
τ_1	RSA Sign+Verify	5,19	4,67	3,66	9,3		✓	✓
τ_2	DSA Sign+Verify	7,07	6,65	6,13	9,3	✓	✓	✓
τ_3	AES Encrypt	0,55	0,55	0,55	1,1		✓	✓
τ_4	AES Decrypt	0,42	0,42	0,42	1,1			✓
τ_5	RC4 Encrypt	0,12	0,12	0,12	1,1	✓		
τ_6	RC4 Decrypt	0,12	0,12	0,12	1,1	✓	✓	✓
τ_7	SHA-256	0,40	0,40	0,40	1,1	✓✓	✓	✓✓
τ_8	HMAC-SHA1	0,17	0,17	0,17	1,1		✓	✓

3.3.2 Particionamento da carga e escalonamento

Em um primeiro passo, as oito tarefas utilizadas para construção das cargas de trabalho foram caracterizadas, obtendo-se seus respectivos tempos de computação (C_i) e períodos (P_i). Depois, para cada carga de trabalho as tarefas foram particionadas entre os processadores utilizando-se tanto o algoritmo RM-FFDU quanto o algoritmo EDF-B&B. As Tabelas 2 e 3 mostram, para os diferentes tamanhos de memória cache, o resultado do particionamento RM-FFDU e EDF-B&B das cargas de trabalho em estudo, respectivamente. É importante ressaltar que as cargas de trabalho WL1, WL2 e WL3 assumem implicitamente sistemas *dual-core* ($M = 2$), *triple-core* ($M = 3$) e *quad-core* ($M = 4$), respectivamente, exceto para o WL3 usando o algoritmo EDF-B&B.

Como visto no Capítulo 2, os algoritmos de particionamento tem como objetivo minimizar o número de *cores* utilizados. Observe que, por esta razão o algoritmo EDF-B&B precisou alocar somente três *cores* para WL3 quando caches de 32KB são usadas, como mostra a última linha da Tabela 3, ao contrário das demais configurações.

3.3.3 Configuração do *template* do MPSoC

Os experimentos fizeram uso de uma representação executável do *template* do MPSoC, o qual foi construído mediante a adaptação da plataforma de simulação GEM5 (BINKERT et al., 2006) para suportar multi-tarefas em um sistema *multicore* baseado no processador ARM. Adotou-se um memória DRAM de 8MB *off-chip*, a qual é compatível com os requisitos do cenário experimental em questão (BERKEL, 2009). Por simplicidade, os experimentos assumem que somente as tarefas pertencentes ao conjunto de tarefas assinaladas para um *core* tem acesso à cache local (privativas àquele *core*). Entretanto, em cenários reais, outros programas podem compartilhar a memória cache com as tarefas assinaladas, isto é, a capacidade efetiva dedicada às tarefas é menor do que a capacidade nominal da cache. Para modelar as capacidades efetivas distintas das caches, foram selecionados tamanhos de cache tais que a máxima taxa de faltas estivesse dentro do intervalo [5%,1%] para cada tarefa sendo executada individualmente em um processador. Isso corresponde a tamanhos no intervalo [8KB,32KB]. Sendo assim, ao selecionar tamanhos de cache de 8KB, 16KB e 32 KB para os experimentos, avaliou-se o impacto da técnica proposta para cenários com níveis alto, moderado ou baixo de competição. Adotou-se o mesmo tamanho tanto para a cache de instruções quanto para a cache de dados, a fim de seguir as tendências de um processador real, como por exemplo o Snapdragon da Qualcommn (QUALCOMM, 2011). Pela mesma razão, a associatividade da memória cache de instruções e da memória cache dados foram selecionadas como 1-*way* e 4-*way*, respectivamente. Foi selecionado o tamanho de bloco de 32 *bytes* para ambas as memórias e a política de escrita *write-back* para a cache de dados pois ambos são atributos padrão da plataforma GEM5.

O intervalo de simulação foi definido pelo hiper-período do conjunto de tarefas, ou seja, o mínimo múltiplo comum entre todos os períodos do conjunto de tarefas. Como resultado, as tarefas foram invocadas o mesmo número de vezes em cada simulação.

3.4 RESULTADOS EXPERIMENTAIS

Primeiramente, avaliou-se o resultado da simulação da carga de trabalho WL3 (executando em um sistema *quad-core*), para o nodo tecnológico de 32nm. Os resultados ao se usar RM-FFDU estão relatados nas Figuras 4, 5 e 6 para cada *core* individualmente e para cada

Tabela 2: Resultado do particionamento de tarefas para RM-FFDU

M	*Core*	**8kB**		**16kB**		**32kB**	
		τ^p	$U(\tau^p)$	τ^p	$U(\tau^p)$	τ^p	$U(\tau^p)$
2	1	τ_2, τ_6	0,863	τ_2, τ_6	0,819	τ_2, τ_6	0,819
	2	τ_5, τ_7, τ_7	0,794	τ_5, τ_7, τ_7	0,794	τ_5, τ_7, τ_7	0,762
3	1	τ_2, τ_6	0,863	τ_2, τ_8	0,859	τ_2, τ_8	0,802
	2	τ_1, τ_8	0,703	τ_1, τ_6	0,607	τ_3, τ_7	0,815
	3	τ_3, τ_7	0,815	τ_3, τ_7	0,815	τ_1, τ_6	0,499
4	1	τ_2, τ_6	0,863	τ_2, τ_8	0,859	τ_2, τ_8	0,802
	2	τ_1, τ_8	0,703	τ_1, τ_6	0,607	τ_3, τ_7	0,815
	3	τ_3, τ_7	0,815	τ_3, τ_7	0,815	τ_1, τ_4	0,753
	4	τ_4, τ_7	0,705	τ_4, τ_7	0,704	τ_6, τ_7	0,451

Tabela 3: Resultado do particionamento de tarefas para EDF-B&B

M	*Core*	**8kB**		**16kB**		**32kB**	
		τ^p	$U(\tau^p)$	τ^p	$U(\tau^p)$	τ^p	$U(\tau^p)$
2	1	τ_2, τ_5, τ_6	0,971	τ_2, τ_5, τ_6	0,927	τ_2, τ_7	0,998
	2	τ_7, τ_7	0,687	τ_7, τ_7	0,686	τ_5, τ_6, τ_7	0,559
3	1	τ_2, τ_8	0,903	τ_2, τ_6, τ_8	0,967	τ_2, τ_7	0,998
	2	τ_1, τ_7	0,898	τ_1, τ_3	0,971	τ_1, τ_3, τ_6	0,971
	3	τ_3, τ_6	0,580	τ_3	0,343	τ_8	0,147
4	1	τ_2, τ_8	0,903	τ_2, τ_6, τ_8	0,967	τ_2, τ_7	0,998
	2	τ_1, τ_4	0,917	τ_1, τ_3	0,971	τ_1, τ_3, τ_6	0,971
	3	τ_3, τ_6, τ_7	0,923	τ_4, τ_7	0,704	τ_4, τ_7, τ_8	0,852
	4	τ_7	0,343	τ_7	0,343	-	-

configuração de cache distinta.

3.4.1 Dominância da cache de instruções

A Figura 4 detalha a contribuição das componentes estática e dinâmica da energia das caches privativas para cada um dos quatro *cores* e para cada uma das três configurações de cache. A componente estática diz respeito ao consumo de todas as caches, ao passo que a componente dinâmica está subdividida considerando os acessos à cache de instruções (somente as leituras) e os acessos de leitura e escrita à cache de dados. Observa-se que, apesar de não se poder negligenciar

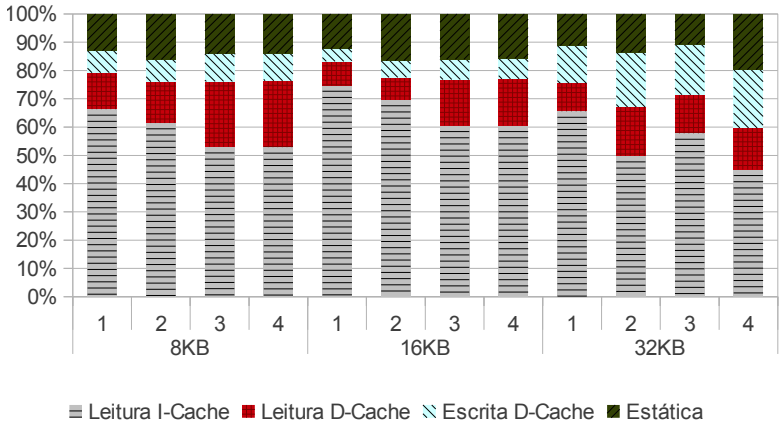


Figura 4: Composição do consumo de energia das caches privativas de cada *core* (32nm)

o impacto da corrente de *leakage*, o consumo das caches é dominado pela energia dinâmica (em média 85,6%). Apesar de o consumo estático aumentar com o tamanho das caches, o consumo dinâmico aumenta proporcionalmente e sendo assim, o impacto da corrente de *leakage* no consumo total permanece praticamente constante. Além disso, a cache de instruções predomina em relação ao consumo total (em média, 60% da energia total).

3.4.2 Limitação da vazão

A Figura 5 mostra a vazão de cada *core* expressa em milhões de operações de memória por segundo (MOPS). Intuitivamente, é de se esperar uma relação entre a vazão e a utilização de um *core*[2]. De fato, os *cores* com maior utilização tendem a apresentar um MOPS maior, porém a vazão também é afetada por características intrínsecas a cada tarefa. Por exemplo, as tarefas de criptografia assimétrica tendem a apresentar uma vazão menor do que as tarefas de criptografia simétrica devido às diferentes localidades de referência[3]. Por isso não existe uma

[2]A utilização representa um limite superior para a vazão (que é um valor médio), pois a primeira supõe que o tempo de computação seja o WCET

[3]Por exemplo, para o WL3 executado na Configuração A, a taxa de faltas na cache de instruções no *core* 2 (onde RSA executa) é 2% enquanto a taxa de faltas do *core* 4 (onde AES *Decrypt* executa) é menor que 0,1%.

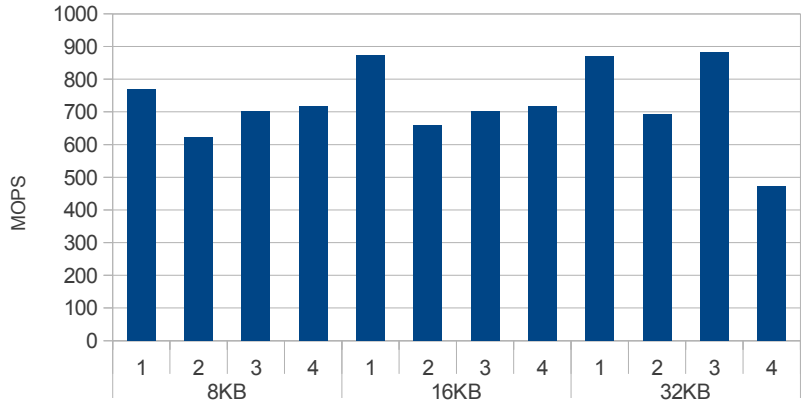


Figura 5: Vazão para cada *core* (32nm)

proporcionalidade entre a utilização e a vazão de cada *core*.

Além disso, intuitivamente, era esperado que o aumento do tamanho das caches levasse a uma vazão maior, devido a redução do número de faltas na cache. Todavia, apenas um pequeno aumento foi observado nos experimentos. Isso se deve ao fato de que a vazão é limitada pela periodicidade das tarefas, em outras palavras, pela taxa de transferência adotada no sistema.

Esse efeito leva a uma importante característica de sistemas com múltiplas tarefas periódicas, que pode ser melhor visualizado quando se divide a vazão pela potência, como reportado a seguir.

3.4.3 Escalonabilidade limita tamanho das caches

Como indica a Figura 6, esse efeito causado por adotar uma taxa de transferência pré-estabelecida leva a uma característica importante de computação multi-tarefa: para maximizar a eficiência energética, a capacidade das cache deve ser mantidas tão pequena quanto a escalonabilidade permite. Para uma dada carga de trabalho, isto pode ser obtido ao derivar uma instância do SoC a partir de uma plataforma de projeto.

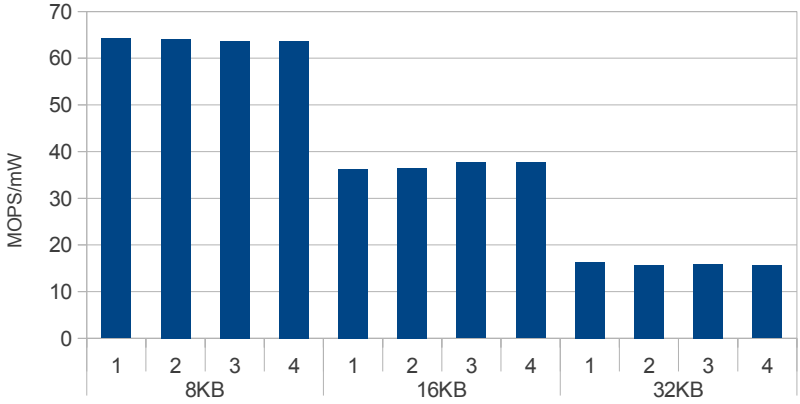


Figura 6: Eficiência energética das caches privativas de cada *core* (32nm)

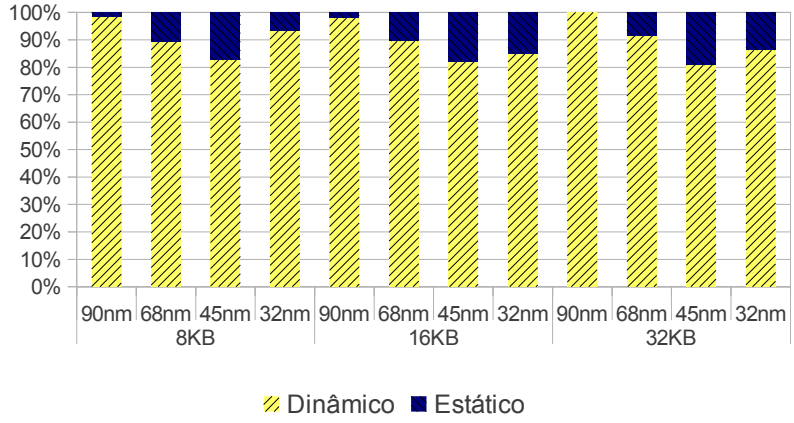


Figura 7: Caracterização da energia das caches privativas para cada nodo tecnológico

3.4.4 Consumo dinâmico domina de 90 a 32nm

O domínio do consumo dinâmico sobre o estático oferece uma oportunidade extra para melhoria: o software embarcado pode ser otimizado para eficiência energética ao racionalizar os acessos à memória. De fato, existe espaço para tais otimizações ao longo de uma ampla gama de processos tecnológicos, como mostra a Figura 7, que captura a contribuição das componentes dinâmica e estática de todos os *cores*

em relação ao total de energia ao utilizar-se o algoritmo RM-FFDU. Pode ser observado que ao transicionar do nodo tecnológico de 45nm para 32nm, existe uma inversão no comportamento da energia estática (estava crescendo e subitamente diminui). Tal efeito ocorre já que no nodo de 32nm, a ferramenta CACTI 6.5 (THOZIYOOR et al., 2008) utilizada para obter o consumo energético e de potência de uma memória, passa a utilizar um modelo que supõe SOI (*Silicon on insulator*) para a fabricação dos transistores. Tal recurso tecnológico diminui o consumo estático de um transistor, e consequentemente, da memória.

3.4.5 Independência da escolha do escalonador-particionador

Foram realizados experimentos similares para as cargas de trabalho 1 e 2 rodando em sistemas *dual-core* e *triple-core*, respectivamente. Como os resultados levaram essencialmente às mesmas conclusões tiradas para a carga de trabalho 3, eles são reportados no Apêndice A para não comprometer a clareza.

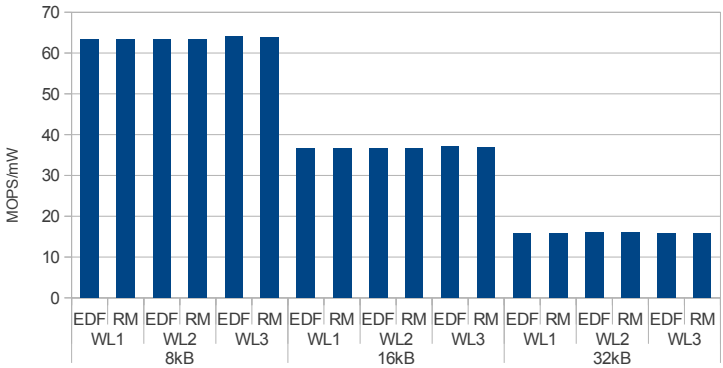


Figura 8: Eficiência energética das caches privativas para cada carga de trabalho (32nm)

Todos os experimentos foram repetidos utilizando EDF-B&B ao invés de RM-FFDU. Eles também são reportados no Apêndice A. Para efeito comparativo do impacto das cargas de trabalho e da escolha do escalonador-particionador, a Figura 8 resume os resultados de eficiência energética para o nodo tecnológico de 32nm.

Embora as cargas de trabalho sejam diferentes (tarefas e particionamentos distintos), observa-se que a eficiência energética é similar para

um dado tamanho de cache. Isso parece indicar que o escalonamento sob restrições de tempo real tem um papel marginal na otimização da eficiência energética, pois ele não afeta a taxa de utilização e, consequentemente, não afeta a vazão, embora ele acabe por definir um limite inferior para o tamanho da cache (apenas a capacidade suficiente para escalonabilidade).

Por outro lado, é de se esperar que o particionamento de tarefas entre *cores* tenha um impacto mais significativo na eficiência energética, pois determina a utilização do subconjunto de tarefas assinaladas para cada *core*, a qual correlata com a vazão. Ademais, o particionamento de instruções e dados entre memórias associativas (caches) e não associativas (memórias de rascunho) tende a afetar substancialmente o consumo de energia, pois parte do *working set* acaba sendo acomodado em uma memória não associativa, cujo consumo por acesso é inferior ao de uma cache de mesma capacidade. Isso sugere que o particionamento de tarefas (que afeta predominantemente a vazão) possa ser tratado como "ortogonal" ao particionamento de memória (que afeta predominantemente o consumo de energia). Assim, o problema de se maximizar a eficiência energética do subsistema de memória poderia ser decomposto em dois subproblemas cujo acoplamento é aparentemente fraco: o particionamento de tarefas e o particionamento de memória[4]. Essa decomposição é objeto do próximo capítulo.

[4]Também conhecido como alocação em SPM.

4 MELHORANDO A EFICIÊNCIA ENERGÉTICA ATRAVÉS DE OTIMIZAÇÃO DE CÓDIGO

O número de *cores* dedicados ao processamento de banda-base tende a crescer em face do aumento da vazão (induzida pela comunicação e segurança) e uma escalonabilidade mais restrita (induzida pelo crescente número de tarefas invocadas em períodos cada vez mais curtos). Para alguns poucos processadores, o particionamento de tarefas de grande granularidade pode ser feito manualmente pelos desenvolvedores do sistema (e.g. as camadas 2 e 3 da pilha assinaladas a *cores* distintos ou o *downlink* e *uplink* assinaladas para *cores* distintos). Entretanto, com o aumento do número de *cores* e requisitos cada vez mais apertados, um único processador pode não acomodar todas as tarefas de uma camada ou de um *link*. Ademais, é de se esperar que, para aumentar as chances de escalonabilidade, as tarefas de maior granularidade sejam subdivididas, a fim de gerar um conjunto de tarefas de menor granularidade. Apesar da existência de escalonadores-particionadores para tarefas periódicas em sistemas multiprocessados (DAVIS; BURNS, 2011), não é de conhecimento do autor que o impacto de um escalonador-particionador na eficiência energética tenha sido reportado.

Da mesma maneira, o particionamento de instruções e dados em memória pode ser feito manualmente pelos desenvolvedores do sistema para blocos de grande granularidade (e.g. partições de *boot*, não-cacheada e cacheada). Enquanto a atribuição de poucos itens de memória (código e dados) para o espaço não-cacheado é simples e direto para tarefas críticas (e.g. rotinas de tratamento de interrupção), a alocação de itens provenientes das tarefas de banda-base para o espaço não-cachável é complexa, já que afeta as garantias de tempo real, vazão e consumo de energia. Apesar de o particionamento automático de memória em espaço não-cacheável já ter sido extensivamente investigado para processadores com único *core* para aplicações de melhor esforço (VERMA; WEHMEYER; MARWEDEL, 2004) (UDAYAKUMARAN; DOMINGUEZ; BARUA, 2006) (CHO et al., 2007) (EGGER et al., 2010) e sistemas de tempo real (FALK; KLEINSORGE, 2009), o particionamento de memória para tarefas periódicas em processadores *multicore* ainda não foi reportado.

Este capítulo aborda o problema de otimizar a eficiência energética do subsistema de memória, dado um conjunto de tarefas periódicas. Propõe-se uma abordagem nova para enfrentar o problema por meio de uma decomposição em dois subproblemas: o particionamento de tarefas

entre *cores* e o particionamento de itens de memória (instruções e dados) entre os espaços de endereçamento cacheável e não-cacheável. A técnica melhora a eficiência energética levando em conta as restrições de tempo real.

Este capítulo está organizado como segue. A Seção 4.1 discute os trabalhos correlatos. A Seção 4.2 apresenta a decomposição proposta e a solução dos subproblemas resultantes. A Seção 4.4 reporta os resultados experimentais.

4.1 TRABALHOS CORRELATOS

4.1.1 Particionamento de tarefa

As técnicas com foco em processamento multimídia e DSP exploram o particionamento de tarefa ou para maximizar a vazão (e.g. (SUHENDRA; RAGHAVAN; MITRA, 2006)) ou para minimizar o consumo energético (e.g. (CHO et al., 2009)). Já as técnicas que tem como alvo aplicações de controle embarcadas particionam as tarefas para minimizar o pior tempo de resposta (e.g. (CHATTOPADHYAY; ROYCHOUDHURY, 2011)). Apesar de tais técnicas obterem uma melhoria significativa no desempenho, ou no consumo de energia ou na previsibilidade, por tratarem do particionamento de tarefas e de memória com um problema de otimização unificado, elas não provêm garantias de tempo real para tarefas periódicas.

4.1.2 Particionamento de memória

Um espaço de endereçamento unificado (**Un**) significa que ele não foi dividido em subespaços mutualmente exclusivos para cada tarefa. Já um espaço de endereçamento particionado (**Pt**) significa que para cada tarefa foi atribuída um intervalo de endereçamento fixo. Dizemos que um memória de rascunho é privativa (**Pv**) quando ela não pode ser acessada por algum outro *core*, caso contrário, ela é dita compartilhada (**Co**).

a) **Tarefa única em um único *core***

   Muitas das técnicas reportadas abordam o mapeamento de itens de memória em dois espaços de endereçamento disjuntos: um correspondente à memória principal (possivelmente cacheado), e

outro correspondente à memória de rascunho (AVISSAR; BARUA; STEWART, 2002) (STEINKE et al., 2002) (ANGIOLINI et al., 2004) (FALK; KLEINSORGE, 2009). Como resultado, nos espaços de endereçamento não sobrepostos (NOB[1]), o conteúdo da memória de rascunho é carregado antes do início da execução e, sendo assim, não impõe *overhead* ao tempo de execução, uma característica muito conveniente para sistemas de tempo real. Entretanto, algumas técnicas de alocação utilizam espaços de endereçamento sobrepostos (OVB[2]) (e.g. (VERMA; WEHMEYER; MARWEDEL, 2004), (UDAYAKUMARAN; DOMINGUEZ; BARUA, 2006) (CHO et al., 2007), (EGGER et al., 2010)). Nestas técnicas os itens frequentemente acessados são dinamicamente copiados da memória principal para o espaço da memória de rascunho, obtendo, em média, uma economia maior de energia do que a com endereços não sobrepostos. Infelizmente, a cópia desses itens impõe um *overhead* no tempo de execução e, dado que a análise estática do WCET pode ser prejudicada pelo código auto-modificável (inerente à essa abordagem), esta técnica é mais adequada para sistemas de melhor-esforço. É por isso que os trabalhos recentes prescrevem espaços particionados (não sobrepostos) para criar uma técnica de alocação em memória de rascunho consciente do WCET que preserva garantias de tempo real (FALK; KLEINSORGE, 2009).

b) **Múltiplas tarefas em um único *core***

As primeiras técnicas para compartilhamento do espaço de endereçamento da memória de rascunho sem sobreposição (VERMA et al., 2005) e com sobreposição de espaços de endereçamento (EGGER; LEE; SHIN, 2008) utilizaram processos com prioridades iguais e escalonados de acordo com a política de *round-robin*. Posteriormente, o particionamento simultâneo de tarefas e memória foi investigado para sistemas preemptivos e multi-tarefa baseados em prioridade (TAKASE; TOMIYAMA; TAKADA, 2010). Todas as abordagens oferecem variantes para espaços de endereçamento unificado, particionado e híbrido.

A Tabela 4 resume as técnicas mais recentes para particionamento de memória em um único *core*. Tais técnicas foram classificadas quanto ao seu modelo de execução (única tarefa ou multi-tarefa), quanto ao suporte (ou não) para restrições de tempo real, quanto

[1]Do inglês, *non-overlay based*
[2]Do inglês, *overlay based*

a arquitetura alvo e quanto aos itens de memórias manipulados (dados, códigos e/ou bibliotecas). A arquitetura alvo ainda foi classificada quanto ao suporte (ou não) de memórias cache, quanto a divisão do espaço de endereçamento da SPM (Un ou Pt) e quanto a técnica de mapeamento utilizada (OVB ou NOB).

c) **Múltiplas tarefas em um processador *multicore***

Os espaços de endereçamento das memórias de rascunho locais são organizados de acordo com o nível de privacidade/compartilhamento necessário. Em uma arquitetura de memória local limitada (LLM[3]) (e.g. Cell da IBM), não existe memória cache e o espaço de endereçamento de um *core* corresponde ao espaço de endereçamento da memória de rascunho. Sendo assim, acessos à memória principal ou à memória de rascunho de outro *core* requerem uma transferência de acesso direto à memória (DMA). Em uma arquitetura com espaço de endereçamento global particionado (PGAS[4]) (e.g. Cortex-R da ARM e InterAptive da MIPS), o espaço de endereçamento de um dado *core* consiste de dois espaços de endereçamento disjuntos: um correspondente à sua memória de rascunho local e outro correspondente às suas caches locais. Visto que os subespaços de endereçamento de todas as memórias de rascunho são independentes, um *core* não consegue acessar a memória de rascunho de um outro *core*. Já a memória principal é acessada quando ocorrem faltas na cache.

Algumas técnicas que utilizam arquiteturas LLM visam minimizar o tempo médio de execução (ACET) (SUHENDRA; RAGHAVAN; MITRA, 2006) (SALAMY; RAMANUJAM, 2012) para aplicações multimídia ou o pior tempo de resposta para aplicações de controle embarcadas (CHATTOPADHYAY; ROYCHOUDHURY, 2011). Tais técnicas não conseguem lidar com as *deadlines* impostas por tarefas periódicas. Por outro lado, arquiteturas de precisão de tempo (PRET[5]), que lidam corretamente com o WCET e requisitos temporais para as múltiplas *threads* que compartilham a mesma memória de rascunho, são mais indicadas para aplicações críticas de segurança e não substituem as arquiteturas PGAS empregadas no processamento de banda-base.

A Tabela 5 resume as técnicas mais recentes para particionamento de memória em *multicore*. Tais técnicas foram classificadas quanto

[3]Do inglês, *limited local memory*
[4]Do inglês, *partitioned global address space*
[5]Do inglês, *precision timed architecture*

ao seu modelo de execução (multi-thread, multi-tarefa aperiódica ou multi-tarefa periódica), quanto ao suporte (ou não) de restrições de tempo real, quanto a arquitetura alvo e quanto aos itens de memórias manipulados (dados, códigos e/ou bibliotecas).A arquitetura alvo ainda foi classificada quanto ao suporte (ou não) de memórias cache, quanto a divisão do espaço de endereçamento da SPM (Un ou Pt), quanto a acessibilidade do espaço de endereçamento (Co ou Pv) e quanto a técnica de mapeamento utilizada (OVB ou NOB).

4.1.3 Identificação de contribuição

Até agora, todos os trabalhos correlatos reportados para *multi-core* ou não levam em conta as restrições de tempo real para **tarefas periódicas** ou não exploram memórias cache. Tais limitações são particularmente inconvenientes para processamento de banda-base. Por isso foi desenvolvida uma abordagem pragmática que realiza uma otimização do código consciente de restrições de tempo real ao particionar itens em espaços de endereçamento cacheável ou não-cacheável.

4.2 DECOMPOSIÇÃO PROPOSTA

O problema original que se quer abordar é o de otimizar a eficiência energética de um subsistema de memória para um dado conjunto de tarefas periódicas. O problema-alvo foi decomposto em dois subproblemas. O primeiro é uma variação do já conhecido problema de otimização denominado *Bin-Packing* (*NP-hard*), o qual pode ser transformado em um problema de factibilidade ao se impor restrições de alocação de recursos e de tempo real. O segundo é uma instância do Problema Binário da Mochila (*Binary Knapsack*).

4.2.1 Escalonamento de tempo real sob restrição de recursos

Problema 1. Atribuição de tarefas em múltiplos *cores*: Dado um número fixo de *cores* M e um conjunto de tarefas $\tau = \{\tau_i = (C_i, P_i, \phi_i) | 1 < i \leq N\}$, encontrar uma partição $\{K_j | 1 < j \leq M\}$ com $K_i \cap K_j = \emptyset \mid$ para todo $i \neq j$, que satisfaça todas as restrições de tempo real.

Tabela 4: Características e limitações das técnicas de particionamento estático e gerenciamento dinâmico da memória de rascunho em um único *core*

Técnica	**Modelo de Execução**	**Restrições de tempo real**	**Arquitetura**			**Items de memória manipulados**		
			Cache	**SPM**		**Dados**	**Código**	**Bib.**
				Espaço	**Gerência**			
(AVISSAR; BARUA; STEWART, 2002)	Tarefa única	-	-	Un	NOB	✓	✓	-
(STEINKE et al., 2002)		-	-	Un	NOB	✓	-	-
(ANGIOLINI et al., 2004)		-	✓	Un	NOB	-	✓	✓
(VERMA; WEHMEYER; MARWEDEL, 2004)		-	-	Un	OVB	✓	✓	-
(UDAYAKUMARAN; DOMINGUEZ; BARUA, 2006)		-	-	Un	OVB	✓	✓	-
(CHO et al., 2007)		-	✓	Un	OVB	✓	-	✓
(EGGER et al., 2010)		-	✓	Pt	OVB	-	✓	✓
(FALK; KLEINSORGE, 2009)		✓	-	Un	NOB	-	✓	-
(VERMA et al., 2005)	*Multi-thread*	-	-	Un / Pt	Híbrido	✓	✓	-
(EGGER; LEE; SHIN, 2008)		-	✓	Un / Pt	OVB	-	✓	✓
(CHO et al., 2009)		-	-	Un	OVB	✓	-	-
(GAUTHIER et al., 2010)	Multi-tarefa, periódico	✓	-	Pt	OVB	✓	✓	-
(TAKASE; TOMIYAMA; TAKADA, 2010)		✓	✓	Un / Pt	Híbrido	-	✓	-
Este trabalho [a]	**Multi-tarefa, periódico**	✓	✓	**Pv e pt**	**NOB**	✓	✓	✓

[a]Este trabalho é direcionado para *multicore* e está nesta tabela a título de comparação

Tabela 5: Características e limitações das técnicas de particionamento estático e gerenciamento dinâmico da memória de rascunho em sistema *multicore*

Técnica	**Modelo de Execução**	**Restrições de tempo real**	**Arquitetura**			**Items de memória manipulados**		
			Cache	**SPM**		**Dados**	**Código**	**Bib.**
				Espaço	**Gerência**			
(KANDEMIR et al., 2004)	*Multi-thread*	-	-	Co e Pt	NOB	✓	-	-
(KOC et al., 2007)		-	-	Co e Pt	NOB	✓	-	-
(MARONGIU; BENINI, 2009)		-	-	Co e Pt	NOB	✓	-	-
(BAI; SHRIVASTAVA; KUDCHADKER, 2011)		-	-	Co e Un	OVB	✓	-	-
(BAI; SHRIVASTAVA, 2010)		-	-	Co e Un	OVB	✓	-	-
(JUNG; SHRIVASTAVA; BAI, 2010)		-	-	Co e Un	OVB	-	✓	✓
(PRAKASH; PATEL, 2012)		✓	-	Co e Pt	NOB	-	✓	✓
(CHE; PANDA; CHATHA, 2010)	Multi-tarefa, aperiódico	-	-	Co e Pt	NOB	-	✓	-
(SUHENDRA; RAGHAVAN; MITRA, 2006)		✓	-	Co e Pt	NOB	✓	-	-
(SALAMY; RAMANUJAM, 2012)		✓	-	Co e Pt	NOB	✓	-	-
(CHATTOPADHYAY; ROYCHOUDHURY, 2011)		✓	-	Co e Pt	OVB	✓	-	-
Este trabalho	**Multi-tarefa, periódico**	✓	✓	**Pv e Pt**	**NOB**	✓	✓	✓

4.2.2 Alocação em memória de rascunho

Seja B_j o conjunto de elementos de programa correspondentes ao subconjunto de tarefas K_j assinaladas ao *core* j.

Problema 2. Alocação de elementos de programa em múltiplas memórias de rascunho: Dado um conjunto de M memórias de rascunho de capacidades iguais, digamos C_{SPM}, e uma partição dos elementos de programa $\{B_j | 1 < j \leq M\}$ com $B_i \cap B_j = \emptyset \mid$ para $i \neq j$, onde cada conjunto B_j é caracterizado por um vetor W_j e pelo *trace* T_j, encontrar uma coleção de alocações $\{X_j\}$ que maximize o lucro $p(X_j)$ tal que $W_j \times X_j \leq C_{SPM}$ para cada $j = 1, 2, \ldots M$.

Já que B_j é uma partição de elementos de programa em processadores, o Problema 2 pode ser resolvido através da resolução de M instâncias independentes da alocação de elementos de programa em uma única memória de rascunho formulada a seguir:

Problema 3. Alocação de elementos de programa em uma única memória de rascunho: Dada uma memória de rascunho com capacidade C_{SPM} e um conjunto de elementos de programa B caracterizados pelo vetor W e pelo *trace* T, encontrar a alocação X que maximize uma dada função lucro $p(X)$ tal que $W \times X \leq C_{SPM}$.

4.2.3 Solução do problema e garantias teóricas

Para resolver o Problema 1 foram adotadas duas abordagens.

a) **Solução com RM-FFDU**
Pelo fato de o algoritmo RM-FFDU (OH; SON, 1995) utilizar uma heurística *First-Fit* para resolver um problema *NP-hard*, não é possível garantir um particionamento de tarefas ótimo. Ainda assim, para um dado conjunto de tarefas atribuídas a um dado *core*, o RM-FFDU produz um escalonamento ótimo (DAVIS; BURNS, 2011).

Embora exista uma condição necessária e suficiente para o teste de factibilidade do RM-FFDU (LEHOCZKY; SHA; DING, 1989), seu esforço computacional pode requerer tempo exponencial com o número de tarefas. Por isso, adotou-se o teste de factibilidade baseado em uma condição suficiente (mas não necessária) (OH;

SON, 1995) a seguir:

$$\frac{C_n}{T_n} \leq 2[\prod_{i=1}^{n-1}(1+u_i)]^{-1} - 1 \tag{4.1}$$

b) **Solução com EDF-B&B**
Ao adotar esse escalonador-particionador, obtém-se garantias de particionamento ótimo de tarefas, devido ao uso da abordagem *Branch&Bound*. Além disso, esta solução oferece plena garantia de factibilidade, já que a condição utilizada para escalonamento é necessária e suficiente (STANKOVIC; RAMAMRITHAM; SPURI, 1998) e requer baixo esforço computacional. O teste de factibilidade utilizado foi:

$$\sum_{i=1}^{n} u_i \leq 1 \tag{4.2}$$

Para tratar o Problema 2, são resolvidas M instâncias do Problema 3 usando o já conhecido algoritmo MinKnap (PISINGER, 1997), o qual foi selecionado por sua otimalidade para um dado *trace* e por sua eficiência no tempo de execução.

Sendo assim, pode-se afirmar que para um dado *trace* e um dado conjunto de tarefas assinaladas a um determinado *core*, a solução adotada para a decomposição proposta é ótima, ou seja, leva a um mínimo local para o consumo de energia.

4.3 FLUXO DE TRABALHO

Para realizar os experimentos do Capítulo 4 foi utilizada parte da infra-estrutura experimental e do fluxo de trabalho do Capítulo 3. Desta maneira, a caracterização das cargas de trabalho (C_i,P_i e ϕ_i), assim como o particionamento através dos algoritmos escalonador-particionadores (RM-FFDU e EDF-B&B), as estimativas de energia por acesso e_M^a e potência P_M de cada memória e a etapa de simulação (a qual obteve os valores das taxas de falta m_{ij} e número de acessos a_{ij} para cada elemento D_{ij}) foram realizados conforme vistos e devidamente explicados no Capítulo 3. Ademais, vale ressaltar a etapa que determina o particionamento e escalonamento das tarefas entre os processadores consiste na resolução do Problema 1.

A Figura 9 descreve a instrumentação e as etapas necessárias

para realizar a otimização do código visando a melhoria da eficiência energética, incluindo as etapas previamente mencionadas.

As propriedades estimadas (e_M^a e P_M) junto com o resultado obtido na simulação (m_{ij}, a_{ij} e T) e os elementos de programa dos arquivos objetos (D_{ij}) permitem a criação da matriz-linha P associada à função lucro. Tal matriz, junto com elementos de programa são então submetidos a etapa de mapeamento. Esta etapa consiste na resolução do Problema 2, o qual foi visto anteriormente.

Após determinado o vetor X, que representa o particionamento dos elementos de programa entre os espaços cacheável e não-cacheável, é realizada a linkedição para, por fim, obter os arquivos executáveis otimizados.

4.4 RESULTADOS EXPERIMENTAIS

4.4.1 Configuração

Nos experimentos, assume-se que cada *core* tem uma cache de instruções (I-cache) diretamente mapeada, uma cache de dados (D-cache) de quatro vias. Como justificado no Capítulo 3, foram adotados tamanhos de 8KB, 16KB e 32KB para as memórias cache. Pelo mesmo motivo, foi selecionado o tamanho de bloco de 32 *bytes* para ambas as memórias e a política de escrita *write-back* para a cache de dados. Cada *core* ainda possui uma memória de rascunho de 8KB. Tal tamanho de memória de rascunho se aproxima da diretriz sugerida por Volpato (2010), a qual sugere que C_{SPM} esteja no intervalo $[C_{CACHE}/2, C_{CACHE}]$. Para assegurar máxima previsibilidade, assume-se que o *dispatcher*, necessário para gerenciar as tarefas atribuídas para um dado *core*, é totalmente alocado na memória de rascunho local. Uma vez que os binários do *dispatcher* ocupam 2KB, a capacidade efetiva para alocar itens de programa advindo de tarefas periódicas é de 6KB. Para obter a energia por acesso e consumo de potência estática para cada memória, foi utilizado o modelo de energia/potência CACTI (MURALIMANOHAR; BALASUBRAMONIAN; JOUPPI, 2009), versão 6.5, tendo como alvo a tecnologia de fabricação de 32nm.

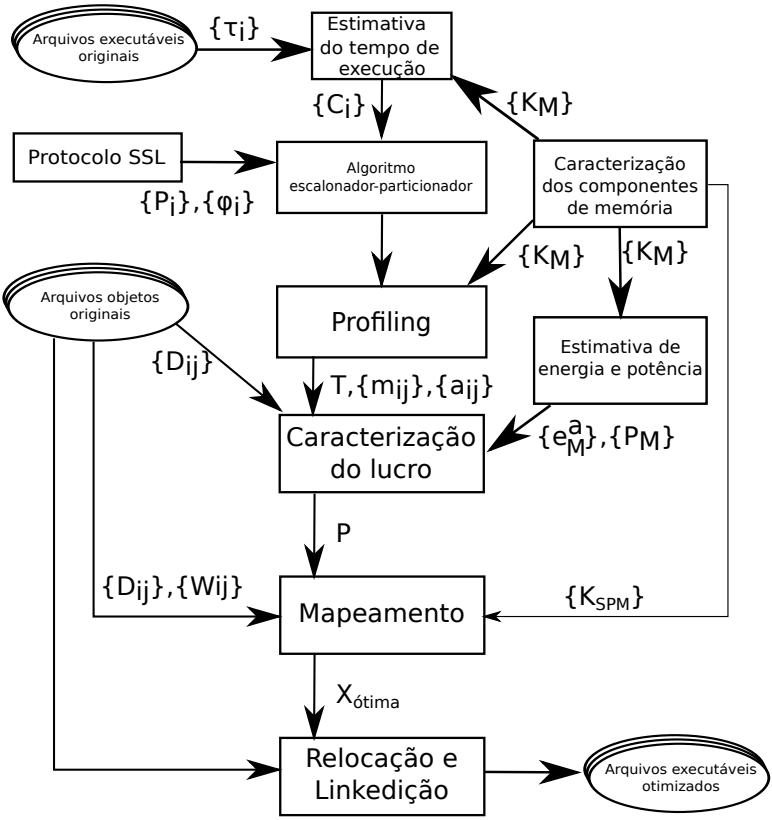


Figura 9: Fluxo proposto para otimização da eficiência energética

4.4.2 Avaliação do impacto

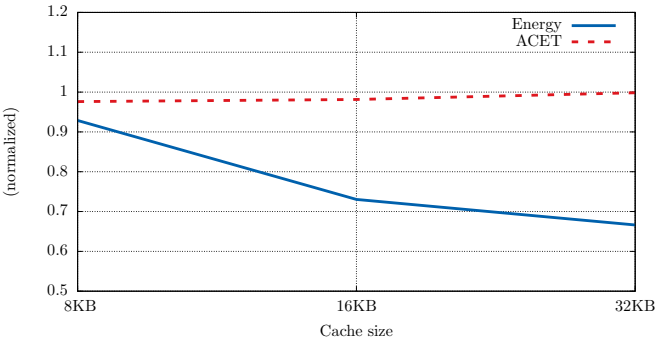
Inicialmente mediu-se a contribuição do subsistema de memória para o consumo de energia e para o ACET após o particionamento de tarefas, mas sem o particionamento de memória (ou seja, todos os elementos de memória residindo em espaço cacheável, exceto os binários do *dispatcher*, que residem em memória de rascunho). Em seguida, mediu-se as mesmas contribuições ao aplicar o particionamento de memória. As Figuras 10 (a), (b) e (c) plotam a razão entre os valores medidos nos cenários otimizado e não otimizado para energia e ACET, usando o algoritmo de particionamento RM-FFDU. Nas três diferentes configurações (*dual-core*, *triple-core* e *quad-core*) pode-se notar que a

técnica obtém uma redução de energia maior com o aumento do tamanho da cache. Isso se deve ao fato de que a diferença entre a energia por acesso da cache e da memória de rascunho aumenta. Assim, não só são alocados mais elementos na memória de rascunho (caso ela não ainda estivesse totalmente mapeada) como também obtém-se um lucro maior ao alocar cada elemento.

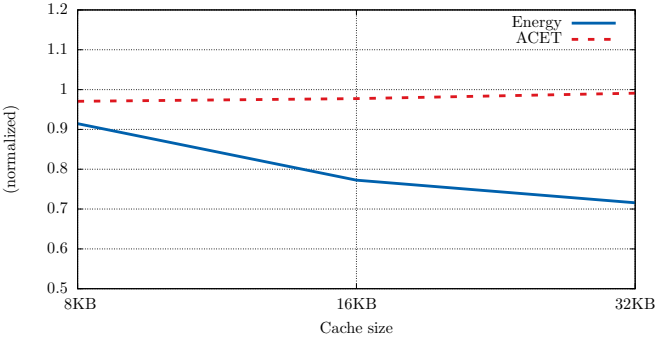
Da mesma maneira, as Figuras 11 (a), (b) e (c) plotam os mesmos dados usando o EDF-B&B. Um comportamento similar em relação aos experimentos com o algoritmo RM-FFDU foi observado, exceto na Figura 11(c). Ao aumentar a cache de 16KB pra 32KB, uma economia de energia ligeiramente menor foi observada. Tal fenômeno se deve ao fato de as memórias de rascunho estarem previamente cheias (não havia mais espaço para alocar itens de memória) e, ao mesmo tempo, devido à diminuição de um *core* nesta configuração (conforme foi explicado no Capítulo 3). Para a configuração de 32KB, o particionador conseguiu alocar no *core* 3 uma tarefa previamente assinalada ao *core* 4 (como se pode observar à direita da última coluna da Tabela 3). Entretanto, essa redução de um *core* também reduziu o espaço efetivo para a alocação dos elementos de programa (já que a memória de rascunho é privativa a cada *core*). Logo, a perda do espaço de alocação e a carga adicional imposta no processador três causaram este fenômeno.

Os resultados obtidos, em relação à redução do consumo de energia, utilizando o algoritmo RM-FFDU foram levemente melhores do que utilizando o algoritmo EDF-B&B (em média 5%). Isso decorre do fato de o algoritmo EDF-B&B possibilitar uma utilização teórica maior que a do RM (STANKOVIC; RAMAMRITHAM; SPURI, 1998). Sendo assim, por causa de um particionamento mais balanceado (em relação ao EDF-B&B) o algoritmo RM-FFDU conseguiu usufruir melhor dos recursos disponíveis (ocasionando uma redução maior no consumo de energia). Assim, conclui-se que, conforme visto no Capítulo 3, o escalonamento de tempo real propriamente dito tem um papel marginal em relação à otimização da eficiência energética, enquanto o critério de particionamento de tarefas tem mais impacto.

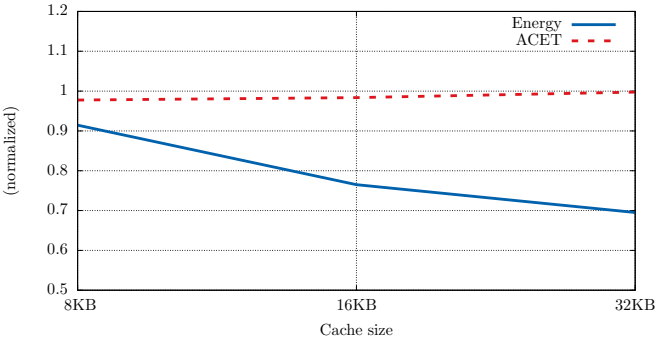
Os resultados mostram que a otimização proposta reduz o consumo da energia na memória para todas as configurações. Foram observadas reduções de até 33% (Figura 10(a) para cache de 32KB). A redução da energia média em todas as configurações e cargas de trabalho é de 20% (considerando o RM-FFDU e EDF-B&B). Esses valores de redução de consumo devem ser interpretados e relativizados à luz das conclusões do capítulo anterior. Como foi possível escalonar todas as cargas de trabalho com caches de 8KB, o uso de caches maiores



(a) Dual-core



(b) Triple-core



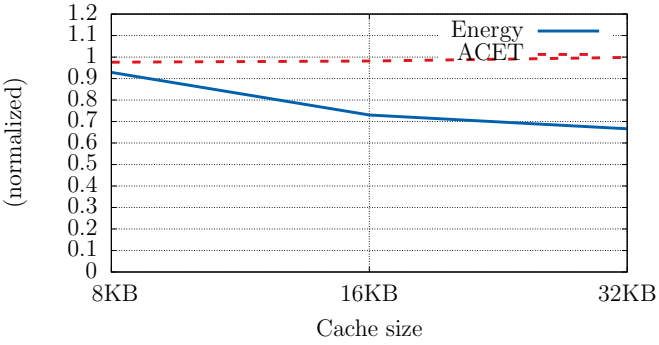
(c) Quad-core

Figura 10: O impacto na eficiência energética usando RM-FFDU

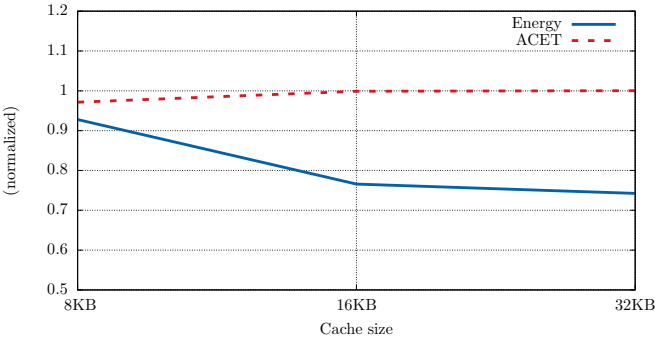
não aumentaria a eficiência energética. Assim, a redução energética obtida com a otimização apenas compensaria o impacto energético do

superdimensionamento das caches. Portanto, para caches bem dimensionadas, a redução energética obtida com a otimização proposta seria correspondente à configuração de 8KB, ou seja, 8% em média para RM-FFDU e 7,7% para EDF-B&B.

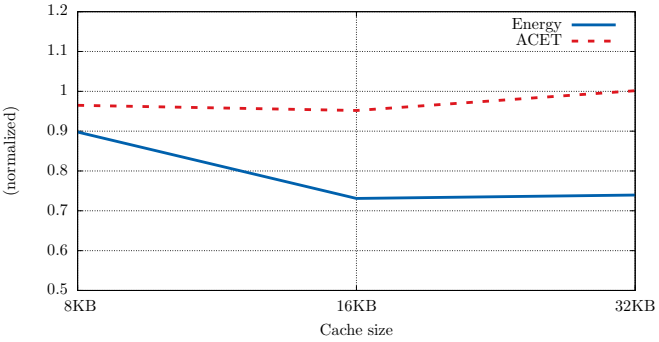
Observou-se um acréscimo máximo de menos de 1% e, em média, um decréscimo de 2% no ACET. Sendo assim, conclui-se que a decomposição proposta não somente reduz o consumo de energia do subsistema de memória, mas também leva a uma melhoria de até 1,33 vezes na eficiência energética do subsistema de memória, porque o ACET praticamente inalterado assegura que não há um aumento significativo no consumo estático do sistema.



(a) Dual-core



(b) Triple-core



(c) Quad-core

Figura 11: O impacto na eficiência energética usando EDF-B&B

5 CONCLUSÕES E PERSPECTIVAS

5.1 CONSEQUÊNCIAS DA ABORDAGEM PROPOSTA

Apesar do número limitado de casos de uso e do reduzido número de configurações de cache avaliadas, a caracterização apresentada no Capítulo 3 permitiu detectar fatores importantes para melhoria de eficiência energética em uma relevante área de aplicação de sistemas de computação com múltiplas tarefas periódicas: os algoritmos criptográficos usados em protocolos de segurança. Por um lado, os resultados obtidos no Capítulo 3 mostraram que, em MPSoCs sob restrições de tempo real (tais como os utilizados para processamento da pilha de protocolo e de aplicações de segurança em dispositivos móveis), o impacto na vazão ao se aumentar o tamanho das caches está limitado pelo caráter periódico das tarefas. Em outras palavras, a capacidade da cache deveria ser mantida tão pequena quanto possível, ou seja, apenas o suficiente para garantir escalonabilidade. Esta é a chave para evitar o superdimensionamento das caches privativas **ao derivar uma instância** de um MPSoC a partir de uma plataforma de projeto: a escalonabilidade acaba por definir **um limite inferior para o tamanho das caches privativas**.

Por outro lado, os resultados também mostraram que a energia dinâmica representa 85% do consumo das caches privativas. Em especial, mostrou-se que a cache de instruções é dominante (65% do consumo das caches privativas). Essa é a chave para otimizar a eficiência energética do *software* embarcado em um MPSoC **derivado** de uma plataforma (onde os tamanhos das cache estão pré-fixados): selecionar preferencialmente otimizações de código que possam reduzir o consumo da **cache de instruções**. Apesar de existirem otimizações que exploram a localidade de referência para diminuir a taxa de faltas e assim reduzir o consumo energético do acesso ao segundo nível de hierarquia (ou níveis inferiores), tais otimizações não conseguem reduzir o consumo de energia das caches primárias, que é o maior responsável pelo consumo em um *core* (DALLY et al., 2008). Afinal, o consumo das caches primárias é praticamente[1] independente da taxa de faltas, a qual afeta o consumo nos demais níveis de hierarquia de memória. Sendo assim, conclui-se que técnicas alternativas, como *software prefetching* e particionamento de memória

[1] No caso de uma falta na cache primária, o consumo por acesso nessa cache é muito inferior ao consumo do acesso aos demais níveis da hierarquia

(JACOB; NG; WANG, 2007) (as quais não se baseiam no princípio da localidade de referência) devam ser exploradas para reduzir o consumo da cache de instruções.

Dado que espaços de endereçamento sobrepostos[2] são mais adequados a sistemas de melhor-esforço (devido ao fato de que o código resultante é auto-modificável, o que compromete o cálculo de um limite superior preciso para o WCET), a decomposição foi deliberadamente projetada para **particionar** o conjunto de dados e instruções entre os espaços de endereçamento cacheável e não-cacheável. Os resultados experimentais do Capítulo 4 mostraram que aquela decomposição leva a melhorias de até 1,33 vezes na eficiência energética do subsistema de memória.

Também foi observado que a escolha do critério de escalonamento (por exemplo, RM ou EDF) tem um papel marginal em relação ao grau de otimização. Por outro lado, o particionamento de tarefas tem um impacto significativo na eficiência energética, pois define o escopo do Problema 3 (que procura minimizar o consumo de energia) e determina a utilização do subconjunto de tarefas assinaladas para cada *core* (a qual correlata com a vazão). Entretanto, observou-se que a melhoria de eficiência energética obtida com uma heurística simples (*First-Fit*) é muito similar à obtida com o particionamento exato.

5.2 LIMITAÇÕES ATUAIS E TRABALHOS FUTUROS

5.2.1 Representatividade do conjunto de dados

Para instrumentar os algoritmos de criptografia nos experimentos do Capítulo 3 e 4, foi utilizado um subconjunto do conjunto de dados especificados para validação de algoritmos criptográficos, propostos por (NIST, 2012). Especificamente, foram utilizados os arquivos de testes 186-3rsatestvectors, shabytetestvectors, 186-2dsatestvectors e aesmmt. Pretende-se variar o conjunto de dados de entradas a fim de melhor aferir a representatividade dos resultados obtidos.

[2] Que são a base de várias técnicas correlatas (VERMA; WEHMEYER; MARWEDEL, 2004), (UDAYAKUMARAN; DOMINGUEZ; BARUA, 2006) (CHO et al., 2007), (EGGER et al., 2010).

5.2.2 Uso do modelo de tarefas generalizado nos experimentos

Nos experimentos do Capítulo 3 e 4 adotou-se um *offset* nulo para todas as tarefas de forma a se poder adotar um critério de escalonabilidade mais simples. Entretanto, a utilização de um *offset* não nulo é importante para se modelar **restrições de precedência entre tarefas cuja execução é independente**. Isso pode ser feito atribuindo-se ao *offset* de uma tarefa o valor do instante de tempo em que a tarefa anterior foi completada. Pretende-se refinar os casos de uso, modelando as restrições de precedência e repetindo os experimentos para avaliar eventual variação nas propriedades energéticas. Para viabilizar os novos experimentos, serão usados testes de escalonamento mais gerais, como por exemplo os propostos em (PELLIZZONI; LIPARI, 2005). Note-se que, ao se considerar *offsets* não nulos, devem aumentar as chances de escalonabilidade (AUDSLEY; DD, 1991), viabilizando casos de uso mais complexos.

5.2.3 Ampliação da diversidade de casos de uso

Embora os algoritmos de criptografia adotados neste trabalho sejam bastante utilizados na prática em protocolos de comunicação reais, o número de combinações poderia ser aumentado. Pretende-se adicionar mais cifradores no conjunto de tarefas, a fim de obter uma maior variedade nas cargas de trabalho. Também pretende-se avaliar o impacto da eficiência energética no sistema, incluindo tarefas não criptográficas do protocolo SSL nas cargas de trabalho. Além disso, pretende-se particionar o código dos algoritmos criptográficos em segmentos menores, **diminuindo a granularidade das tarefas**, o que deve viabilizar o escalonamento de casos de uso mais complexos sem requerer processadores adicionais (para isso é necessário relaxar a limitação de *offset* nulo, como explicado acima). Finalmente, pretende-se ampliar o número de configurações de cache adotadas.

5.2.4 Comparação com técnicas da literatura

Nos experimentos do Capítulo 4 a abordagem proposta utiliza-se de um particionamento para decompor o Problema 2, e então, utilizar um já conhecido algoritmo MinKnap (PISINGER, 1997) para solucionar instâncias do Problema 3. Tal algoritmo já foi extensivamente estudado

pela comunidade acadêmica, logo, a comparação dos resultados obtidos por este trabalho com trabalhos de um único *core*, os quais não capturam as outras características específicas deste trabalho (como o escalonamento de tarefas), não seria apropriada.

5.2.5 Aferição da qualidade da solução obtida

Os resultados do Capítulo 3 indicaram que o particionamento de tarefas impacta sobretudo a vazão, enquanto que o particionamento de memória impacta sobretudo o consumo de energia. Essa evidência preliminar motivou a decomposição proposta no Capítulo 4, resultando em dois subproblemas que aparentam ser fracamente acoplados, o que leva a crer que a solução obtida com a decomposição possa estar próxima da solução ótima do problema original. Pretende-se aferir a hipótese de acoplamento aparentemente fraco, através da modelagem do problema original na forma de um Programa Linear Inteiro. Com isso, se poderá comparar a solução ótima do problema unificado com a solução obtida com a decomposição proposta (ou seja, a resolução do Problema 1 seguida da resolução do Problema 2). Diante da impossibilidade de comparação direta com trabalhos correlatos para essa classe de aplicação, a comparação da solução do problema unificado é a mais adequada para avaliar a qualidade do trabalho proposto.

5.2.6 Aferição do impacto nas garantias de tempo real

Ao contrário da maioria das técnicas correlatas para redução de consumo de energia ((KANDEMIR et al., 2004), (EGGER; LEE; SHIN, 2008), (CHO et al., 2009), (TAKASE; TOMIYAMA; TAKADA, 2010)), a técnica proposta no Capítulo 4 leva em conta as restrições de tempo real, o que deveria preservar a escalonabilidade. Entretanto, como nada se provou sobre o impacto da otimização proposta sobre o WCET, não se pode oferecer garantias de tempo real. Por isso, assim que se tenha acesso a uma infraestrutura apropriada, pretende-se realizar uma **análise de WCET posterior à otimização** de cada caso de uso para aferir em que medida a otimização proposta impacta tais garantias.

REFERÊNCIAS BIBLIOGRÁFICAS

ANGIOLINI, F. et al. A post-compiler approach to scratchpad mapping of code. *International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*, p. 259–267, 2004.

ARM HOLDINGS. *New Cortex™-R Processors for LTE and 4G Mobile Baseband*. February 2011. White Paper.

AUDSLEY, N. C.; DD, Y. *Optimal Priority Assignment And Feasibility Of Static Priority Tasks With Arbitrary Start Times*. dez. 1991.

AVISSAR, O.; BARUA, R.; STEWART, D. An optimal memory allocation scheme for scratch-pad-based embedded systems. *ACM Transactions on Embedded Computing Systems*, v. 1, n. 1, p. 6–26, 2002.

BAI, K.; SHRIVASTAVA, A. Heap data management for Limited Local Memory (LLM) multi-core processors. In: *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. [S.l.: s.n.], 2010. p. 317–326. ISBN 978-1-60558-905-3.

BAI, K.; SHRIVASTAVA, A.; KUDCHADKER, S. Stack data management for Limited Local Memory (LLM) multi-core processors. In: *International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*. [S.l.: s.n.], 2011. p. 231–234. ISSN 2160-0511.

BARUAH, S.; GOOSSENS, J. Scheduling real-time tasks: Algorithms and complexity. In: LEUNG, J. Y.-T. (Ed.). *Handbook of Scheduling — Algorithms, Models, and Performance Analysis*. Boca Raton-London-New York-Washington, D.C.: Chapman & Hall/CRC, 2004. (Computer and Information Science Series), p. 28–28.

BERKEL, C. H. K. van. Multi-core for mobile phones. In: *Design, Automation and Test in Europe Conference (DATE)*. [S.l.: s.n.], 2009. p. 1260–1265. ISBN 978-3-9810801-5-5.

BINKERT, N. et al. The M5 Simulator: Modeling networked systems. *IEEE Micro*, 2006.

BUTTAZZO, G. C. Rate monotonic vs. edf: judgment day. *Real-Time Systems*, Kluwer Academic Publishers, Norwell, MA, USA, v. 29, n. 1, p. 5–26, jan. 2005. ISSN 0922-6443. <http://dx.doi.org/10.1023/B:TIME.0000048932.30002.d9>.

CHATTOPADHYAY, S.; ROYCHOUDHURY, A. Static bus schedule aware scratchpad allocation in multiprocessors. In: *Conference on Languages, Compilers and Tools for Embedded Systems (LCTES)*. [S.l.: s.n.], 2011. p. 11–20. ISBN 978-1-4503-0555-6.

CHE, W.; PANDA, A.; CHATHA, K. Compilation of stream programs for multicore processors that incorporate scratchpad memories. In: *Design, Automation Test in Europe Conference (DATE)*. [S.l.: s.n.], 2010. p. 1118–1123.

CHO, D. et al. Adaptive scratch pad memory management for dynamic behavior of multimedia applications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 28, n. 4, p. 554 –567, 2009.

CHO, H. et al. Dynamic data scratchpad memory management for a memory subsystem with an MMU. *Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, p. 195–206, 2007.

Coffman, Jr., E. G.; GAREY, M. R.; JOHNSON, D. S. Approximation algorithms for bin-packing — an updated survey. In: AUSIELLO, G.; LUCERTINI, M.; SERAFINI, P. (Ed.). *Algorithm Design for Computer Systems Design*. [S.l.]: Springer-Verlag, 1984. p. 49–106.

CONG, J.; GURURAJ, K. Energy efficient multiprocessor task scheduling under input-dependent variation. In: *DATE*. [S.l.]: IEEE, 2009. p. 411–416.

DALLY, W. et al. Efficient embedded computing. *IEEE Computer*, v. 41, n. 7, p. 27–32, July 2008. ISSN 0018-9162.

DAVIS, R. I.; BURNS, A. A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Surveys*, ACM, New York, NY, USA, v. 43, n. 4, p. 35:1–35:44, october 2011. ISSN 0360-0300.

EGGER, B.; LEE, J.; SHIN, H. Scratchpad memory management in a multitasking environment. In: *International conference on Embedded Software (EMSOFT)*. [S.l.: s.n.], 2008. p. 265–274. ISBN 978-1-60558-468-3.

EGGER, B. et al. Scratchpad memory management techniques for code in embedded systems without an MMU. *IEEE Transactions on Computers*, v. 59, n. 8, p. 1047–1062, 2010.

FALK, H.; KLEINSORGE, J. C. Optimal static WCET-aware scratchpad allocation of program code. In: *Design Automation Conference (DAC)*. [S.l.: s.n.], 2009. p. 732–737.

GAUTHIER, L. et al. Minimizing inter-task interferences in scratch-pad memory usage for reducing the energy consumption of multi-task systems. In: *International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*. [S.l.: s.n.], 2010. p. 157–166. ISBN 978-1-60558-903-9.

HILL, M. D. et al. Wisconsin architectural research tool set. *SIGARCH Comput. Archit. News*, ACM, New York, NY, USA, v. 21, p. 8–10, September 1993.

JACOB, B.; NG, S.; WANG, D. Memory Systems: Cache, DRAM, Disk. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 2007.

JUNG, S. C.; SHRIVASTAVA, A.; BAI, K. Dynamic code mapping for limited local memory systems. In: *International Conference on Application-Specific Systems Architectures and Processors (ASAP)*. [S.l.: s.n.], 2010. p. 13–20. ISSN 1063-6268.

KANDEMIR, M. et al. Compiler-directed scratch pad memory optimization for embedded multiprocessors. *IEEE Transactions on VLSI Systems*, v. 12, n. 3, p. 281–287, 2004.

KOC, H. et al. Reducing off-chip memory access costs using data recomputation in embedded chip multi-processors. In: *Design Automation Conference (DAC)*. [S.l.: s.n.], 2007. p. 224–229. ISSN 0738-100X.

LEHOCZKY, J.; SHA, L.; DING, Y. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In: *Proceedings IEEE 10th Real-Time Systems Symposium*. [S.l.: s.n.], 1989. p. 166–171.

LIU, C. L.; LAYLAND, J. W. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, v. 20, n. 1, p. 46–61, jan. 1973. ISSN 0004-5411.

MARONGIU, A.; BENINI, L. Efficient OpenMP support and extensions for MPSoCs with explicitly managed memory hierarchy. In:

Design, Automation Test in Europe Conference (DATE). [S.l.: s.n.], 2009. p. 809–814. ISSN 1530-1591.

MARTELLO, S.; TOTH, P. *Knapsack problems: algorithms and computer implementations*. New York, NY, USA: John Wiley & Sons, Inc., 1990.

MURALIMANOHAR, N.; BALASUBRAMONIAN, R.; JOUPPI, N. *CACTI 6.0: A Tool to Model Large Caches*. [S.l.], 2009.

NIST. *Cryptographic Algorithm Validation Program*. 2012. <http://csrc.nist.gov/groups/STM/cavp/>.

OH, Y.; SON, S. H. *Fixed-Priority Scheduling of Periodic Tasks on Multiprocessor Systems*. [S.l.], 1995.

OPRN. *OpenSSL Project*. 2001. <http://www.openssl.org/>.

PELLIZZONI, R.; LIPARI, G. Feasibility analysis of real-time periodic tasks with offsets. *Real-Time Systems: The International Journal of Time-Critical Computing*, Kluwer Academic Publishers, Norwell, MA, USA, v. 30, n. 1-2, p. 105–128, May 2005. ISSN 0922-6443.

PISINGER, D. A Minimal Algorithm for the 0-1 Knapsack Problem. *Operations Research*, v. 45, p. 758–767, 1997.

POLETTI, F. et al. Energy-efficient multiprocessor systems-on-chip for embedded computing: Exploring programming models and their architectural support. *IEEE Transactions on Computers*, v. 56, n. 5, p. 606–621, may 2007.

POTLAPALLY achiketh R. et al. A study of the energy consumption characteristics of cryptographic algorithms and security protocols. *IEEE Transactions on Mobile Computing*, v. 5, n. 2, p. 128–143, February 2006.

PRAKASH, A.; PATEL, H. An instruction scratchpad memory allocation for the precision timed architecture. In: *Design, Automation Test in Europe Conference (DATE)*. [S.l.: s.n.], 2012. p. 659–664. ISSN 1530-1591.

QUALCOMM. *Snapdragon S4 Processors: System on Chip Solutions for a New Mobile Age*. October 2011. White Paper.

RODRIGUEZ, S.; JACOB, B. Energy/power breakdown of pipelined nanometer caches (90nm/65nm/45nm/32nm). In: *International Symposium on Low Power Electronics and Design (ISLPED)*. [S.l.: s.n.], 2006. p. 25–30.

SALAMY, H.; RAMANUJAM, J. An effective solution to task scheduling and memory partitioning for multiprocessor system-on-chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 31, n. 5, p. 717–725, 2012. ISSN 0278-0070.

STANKOVIC, J. A.; RAMAMRITHAM, K.; SPURI, M. *Deadline Scheduling for Real-Time Systems: Edf and Related Algorithms*. Norwell, MA, USA: Kluwer Academic Publishers, 1998. ISBN 0792382692.

STEINKE, S. et al. Assigning program and data objects to scratchpad for energy reduction. In: *Conference on Design, Automation and Test in Europe (DATE)*. [S.l.: s.n.], 2002. p. 409–415.

SUHENDRA, V.; RAGHAVAN, C.; MITRA, T. Integrated scratchpad memory optimization and task scheduling for MPSoC architectures. In: *International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*. [S.l.: s.n.], 2006. p. 401–410.

TAKASE, H.; TOMIYAMA, H.; TAKADA, H. Partitioning and allocation of scratch-pad memory for priority-based preemptive multi-task systems. In: *Design, Automation Test in Europe Conference (DATE)*. [S.l.: s.n.], 2010. p. 1124–1129. ISSN 1530-1591.

THOZIYOOR, S. et al. *CACTI 5.1*. [S.l.], abr. 2008.

TURNER, C. *New Cortex-R Processors for LTE and 4G Mobile Baseband*. February 2011. <www.arm.com>.

UDAYAKUMARAN, S.; DOMINGUEZ, A.; BARUA, R. Dynamic allocation for scratch-pad memory using compile-time decisions. *ACM Transactions on Embedded Computing Systems*, ACM, New York, NY, USA, v. 5, n. 2, p. 472–511, 2006.

VERMA, M. et al. Scratchpad sharing strategies for multiprocess embedded systems: a first approach. In: *Embedded Systems for Real-Time Multimedia (ESTMEDIA)*. [S.l.: s.n.], 2005. p. 115–120.

VERMA, M.; WEHMEYER, L.; MARWEDEL, P. Dynamic overlay of scratchpad memory for energy minimization. In: *International*

Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS). [S.l.: s.n.], 2004. p. 104–109.

VOLPATO, D. P. *Gerenciamento Explícito de Memória Auxiliar a partir de Arquivos-Objeto para Melhoria da Eficiência Energética de Sistemas Embarcados*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina, December 2010.

WESTPHAL, R.; GÜNTZEL, J. L.; SANTOS, L. C. V. Energy-efficient multi-task computing on mpsocs: a case study from a memory perspective. In: *The IEEE International Conference on Electronics, Circuits, and Systems (ICECS 2012)*. Sevilha, Espanha: [s.n.], 2012.

WOLF, W.; JERRAYA, A.; MARTIN, G. Multiprocessor system-on-chip (mpsoc) technology. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, v. 27, n. 10, p. 1701 –1713, oct. 2008. ISSN 0278-0070.

ZAPATA, O.; ALVAREZ, P. *EDF and RM Multiprocessor Scheduling Algorithms: Survey and Performance Evaluation*. Zacatenco, México, October 2005.

APÊNDICE A – Resultados Experimentais Complementares do Capítulo 3

As figuras a seguir são complementares ao Capítulo 3, no qual foram omitidas para clareza da dissertação. A seguir está a avaliação dos resultados experimentais para as cargas de trabalho 1 e 2 usando RM-FFDU e 1, 2 e 3 usando EDF-B&B, para o nodo tecnológico de 32nm.

A.1 DOMINÂNCIA DA CACHE DE INSTRUÇÕES

As Figuras 12, 13, 14, 15 e 16 detalham a contribuição das componentes estática e dinâmica da energia das caches privativas. Conforme visto no Capítulo 3, observou-se que o consumo da cache é dominado pela energia dinâmica. Apesar de o consumo estático aumentar com o tamanho da cache, o consumo dinâmico aumenta proporcionalmente e sendo assim, o impacto da corrente de *leakage* no consumo total permanece praticamente constante. Além disso, a cache de instruções predomina em relação ao consumo total.

Entretanto, pode ser observado que nas Figuras 15 e 16, nos *cores* 3 e 4, respectivamente, a contribuição da energia estática é maior em relação ao observado nas outras configurações. Isso se deve ao fato de que, conforme mostra a Tabela 3, a taxa de utilização desses processadores é inferior aos outros. Sendo assim, é evidente o acréscimo na energia estática, tendo em visto a energia dinâmica é oriunda dos acessos a memória das cargas de trabalho.

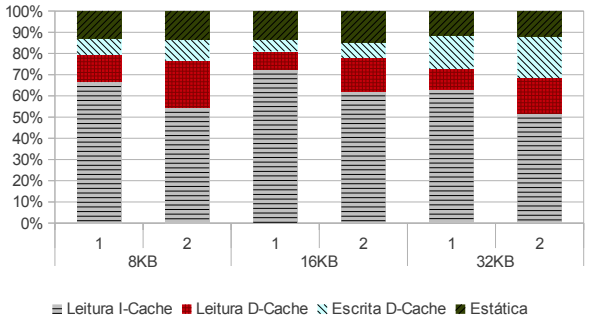


Figura 12: Composição do consumo de energia das caches privativas de cada *core* (RM-FFDU - WL1)

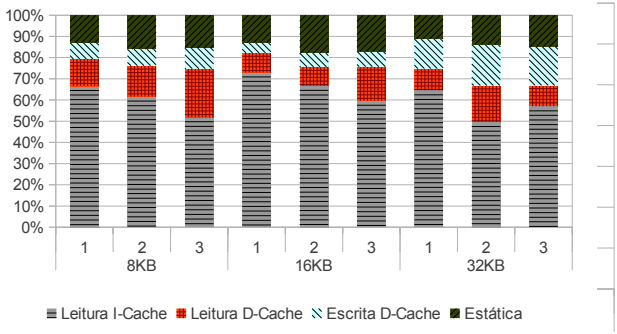


Figura 13: Composição do consumo de energia das caches privativas de cada *core* (RM-FFDU - WL2)

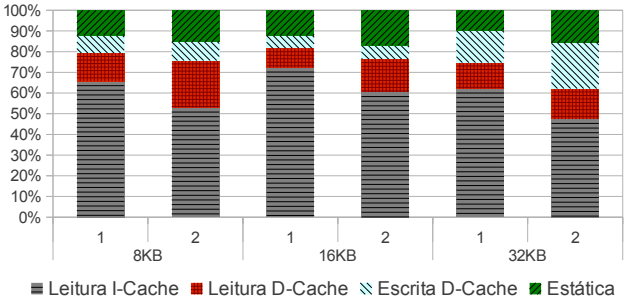


Figura 14: Composição do consumo de energia das caches privativas de cada *core* (EDF-B&B - WL1)

A.2 LIMITAÇÃO DA VAZÃO

As Figuras 17, 18, 19, 20 e 21 mostram a vazão de cada *core* expressa em milhões de operações de memória por segundo (MOPS). Conforme visto no Capítulo 3, apesar de os cores com maior utilização tenderem a apresentar um MOPS maior, pelos motivos previamente relatados, não existe uma proporcionalidade entre a utilização e a vazão de um *core*. Ademais, observou-se que aumento do tamanho das cache não levou a uma vazão maior. O acréscimo observado em alguns casos se deve à redistribuição das cargas dos *cores* (alterando assim a utilização dos mesmos).

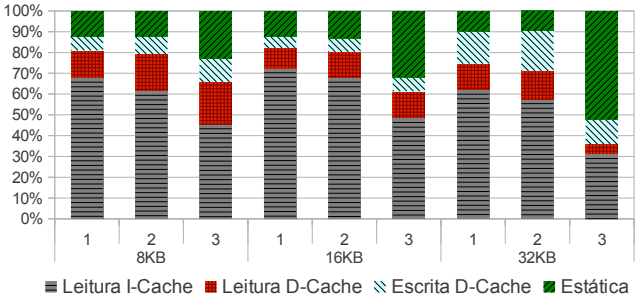


Figura 15: Composição do consumo de energia das caches privativas de cada *core* (EDF-B&B - WL2)

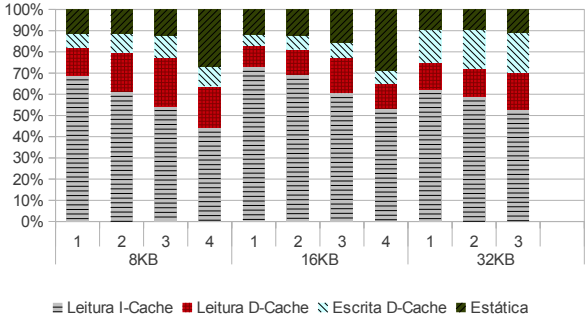


Figura 16: Composição do consumo de energia das caches privativas de cada *core* (EDF-B&B - WL3)

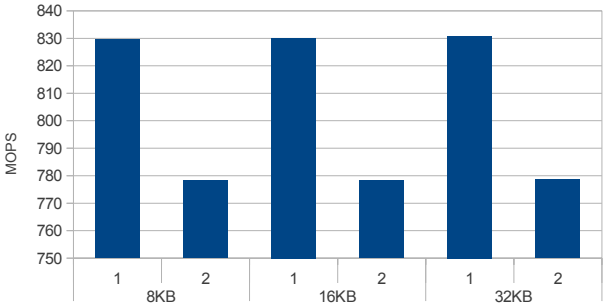


Figura 17: Vazão para cada *core* (RM-FFDU - WL1)

A.2.1 Escalonabilidade limita tamanho das caches

As Figuras 22, 23, 24, 25 e 26 mostram divisão da vazão pela potência para os diferentes casos. O efeito ao adotar-se uma taxa de

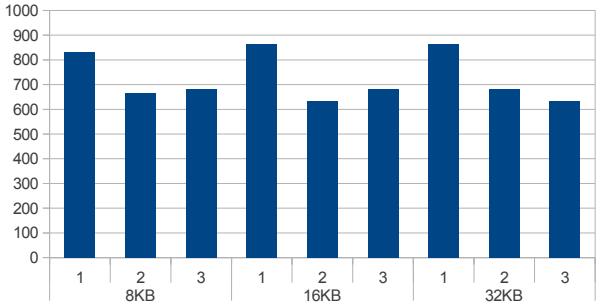


Figura 18: Vazão para cada *core* (RM-FFDU - WL2)

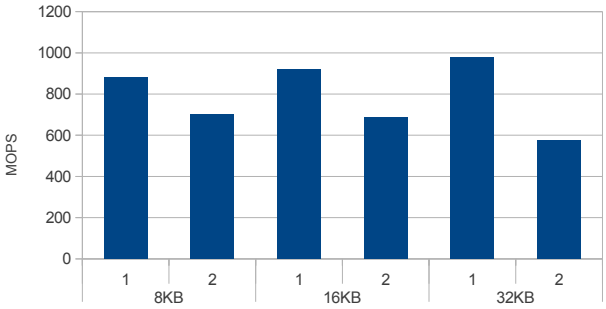


Figura 19: Vazão para cada *core* (EDF-B&B - WL1)

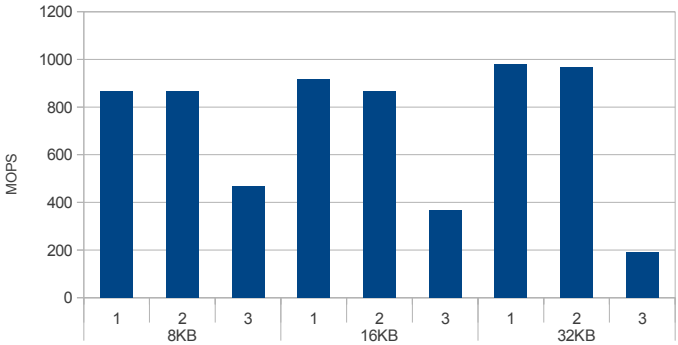


Figura 20: Vazão para cada *core* (EDF-B&B - WL2)

transferência pré-estabelecida leva a uma característica importante de computação multi-tarefa: para maximizar a eficiência energética, a

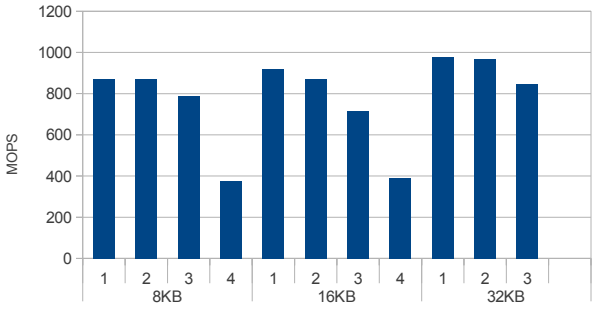


Figura 21: Vazão para cada *core* (EDF-B&B - WL3)

capacidade das cache deve ser mantidas tão pequena quanto a escalonabilidade permite.

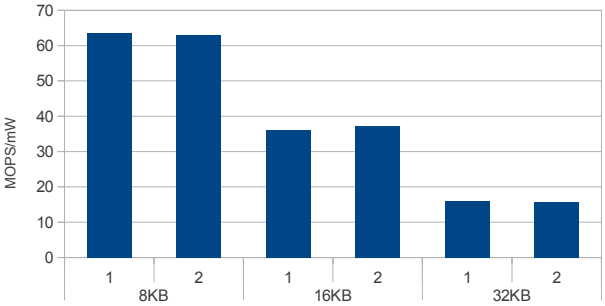


Figura 22: Eficiência energética das caches privativas de cada *core* (RM-FFDU - WL1)

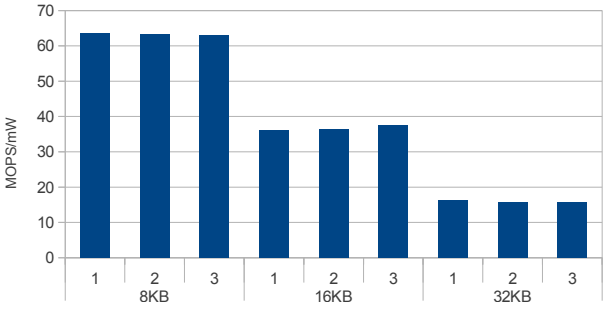


Figura 23: Eficiência energética das caches privativas de cada *core* (RM-FFDU - WL2)

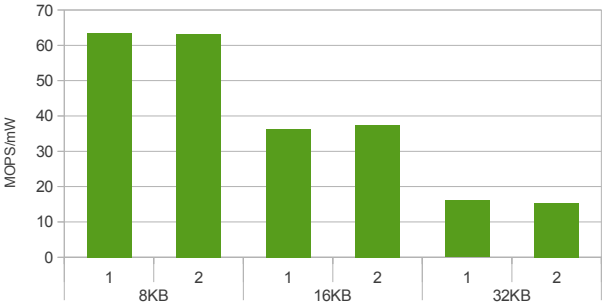


Figura 24: Eficiência energética das caches privativas de cada *core* (EDF-B&B - WL1)

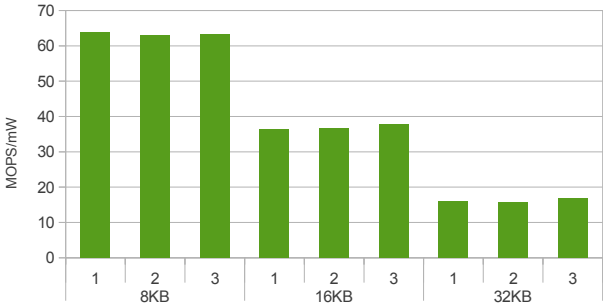


Figura 25: Eficiência energética das caches privativas de cada *core* (EDF-B&B - WL2)

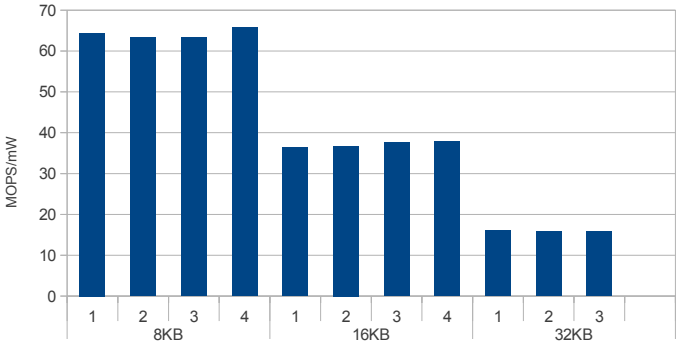


Figura 26: Eficiência energética das caches privativas de cada *core* (EDF-B&B - WL3)