UNIVERSIDADE FEDERAL DE PELOTAS
Centro de Desenvolvimento Tecnológico
Programa de Pós-Graduação em Computação

Dissertação

Estudo e avaliação de arquiteturas de processadores quanto ao uso de memória em sistemas embarcados

Lizandro de Souza Oliveira

Pelotas, 2015

Lizandro de Souza Oliveira

Estudo e avaliação de arquiteturas de processadores quanto ao uso de memória em sistemas embarcados

Dissertação apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Júlio Carlos Balzano de Mattos

Co-Orientadora: Profª Drª Lisane Brisolara de Brisolara

Pelotas, 2015

Universidade Federal de Pelotas / Sistema de Bibliotecas
Catalogação na Publicação

O48e Oliveira, Lizandro de Souza

Estudo e avaliação de arquiteturas de processadores quanto ao uso de memória em sistemas embarcados / Lizandro de Souza Oliveira ; Júlio Carlos Balzano de Mattos, orientador ; Lisane Brisolara de Brisolara, coorientadora. — Pelotas, 2015.

110 f. : il.

Dissertação (Mestrado) — Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, 2015.

1. Sistemas embarcados. 2. Plataformas virtuais. 3. Aplicações embarcadas. 4. Modelos de arquiteturas. 5. Acesso à memória. I. Mattos, Júlio Carlos Balzano de, orient. II. Brisolara, Lisane Brisolara de, coorient. III. Título.

CDD : 005

Elaborada por Aline Herbstrith Batista CRB: 10/1737

Lizandro de Souza Oliveira

Estudo e avaliação de arquiteturas de processadores quanto ao uso de memória em sistemas embarcados

Dissertação aprovada, como requisito parcial, para obtenção do grau de Mestre em Ciência da Computação, Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas.

Data da Defesa: 20/03/2015

Banca examinadora:

.....
Prof. Dr. Júlio Carlos Balzano de Mattos (Orientador)
Doutor em Computação pela Universidade Federal do Rio Grande do Sul

.....
Prof. Drª Lisane Brisolara de Brisolara (Co-Orientadora)
Doutora em Computação pela Universidade Federal do Rio Grande do Sul

.....
Prof. Dr. Mateus Beck Rutzig
Doutor em Computação pela Universidade Federal do Rio Grande do Sul

.....
Prof. Dr. Felipe de Souza Marques
Doutor em Computação pela Universidade Federal do Rio Grande do Sul

.....
Prof. Dr. Rafael Iankowski Soares
Doutor em Computação pela Pontifícia Universidade Católica do Rio Grande do Sul

Dedico este trabalho à Bárbara Britto, por todo amor, carinho e compreensão nos vários momentos difíceis, em especial nos diversos finais de semana envolvidos com trabalho e estudo. Sem teu apoio eu teria desistido. Muito obrigado! Te amo muito!

Dedico, ainda, à minha mãe Cleusa Maria Teixeira de Souza, que me ensinou, desde cedo, as primeiras palavras. Minha primeira e melhor professora! Te amo, mãe!

Por fim, dedico este trabalho aos meus avós Luís de Oliveira (*in memorian*), Iolanda Bernardes Oliveira (*in memorian*) e Marieta Teixeira de Souza (*in memorian*).

Agradecimentos

Ao meu orientador do Mestrado em Computação, professor Julio Carlos Balzano de Mattos, por toda confiança depositada durante o curso e pelas contribuições para a realização deste trabalho.

À minha co-orientadora, professora Lisane Brisolara, que contribuiu desde o início do trabalho, sanando dúvidas diversas, revisando artigos e propondo correções para o trabalho.

Aos professores do PPGC com os quais tive o privilégio de ter aulas.

Aos profissionais Rajesh Samat da empresa Wind River® e Duncan Graham da empresa Imperas Software, pela orientação na obtenção das licenças dos *softwares* e pelas valiosas contribuições técnicas que possibilitaram a correta instalação e utilização das ferramentas.

Às empresas Wind River® e Imperas Software pela concessão das licenças universitárias das ferramentas utilizadas no trabalho.

Aos colegas do GACI, Eduardo Nicola e Lisandro Silva, pelo trabalho incansável na utilização do Simics.

Aos professores Luís Cléber Carneiro Marques, Adão Antônio de Souza Júnior e Mauro André Barbosa Cunha pelas cartas de recomendação no Mestrado.

Aos colegas do Instituto Federal Sul-Rio-Grandense pelo apoio ao meu afastamento durante a realização do Mestrado.

À Martha Maia, secretária do PPGC, por todo o profissionalismo e também por sua palavra amiga, acolhedora e sempre positiva. Muito obrigado, minha amiga!

Aos colegas de curso Leonardo Fisher, Leonardo Freitas e Anderson Martins que muito me ajudaram a chegar até aqui, seja nos trabalhos de disciplinas ou ainda no incentivo em vencer os desafios que nos eram propostos.

Por fim, agradeço a todos os colegas e amigos pelo apoio e que contribuíram, ainda que indiretamente, para a realização deste trabalho.

Resumo

OLIVEIRA, Lizandro de Souza. **Estudo e avaliação de arquiteturas de processadores quanto ao uso de memória em sistemas embarcados.** 2015. 110 f. Dissertação (Mestrado em Ciência da Computação). Universidade Federal de Pelotas, Pelotas.

Os sistemas embarcados possuem diversos requisitos e restrições. Além do custo significativo, estes sistemas devem atender restrições rígidas de desempenho e limitações no consumo energético. O sistema de memória é um dos principais fatores que contribuem para o desempenho e consumo de energia em software embarcado. Plataformas virtuais oferecem uma alternativa ao protótipo de *hardware* e também apresentam muitas vantagens, tais como um início de desenvolvimento mais cedo, desempenho e acessibilidade. Existem diversas plataformas virtuais e simuladores que fornecem vários modelos de arquiteturas de processadores. Este trabalho apresenta um estudo e avaliação quanto ao uso de memória em arquiteturas de processadores embarcados. O trabalho detalha diferentes técnicas de otimização de memória para sistemas embarcados. Foram avaliadas as arquiteturas ARM e x86 quanto ao acesso à memória e consumo energético. Para estas avaliações, foram utilizados os *traces* de instruções e de dados gerados por diferentes aplicações embarcadas do pacote MiBench. As ferramentas de avaliação das arquiteturas de processadores embarcados utilizadas foram o Simics e o CACTI. Os resultados mostram que 34,42% do total de instruções para a arquitetura ARM são instruções do tipo *load* e *store*. Além disso, 64,41% dos acessos, na média, à memória de instrução são ocasionados por apenas 8KB do programa, já para 64KB o percentual é de 93,40%. Em relação a memória de dados, 8KB dos dados para esta arquitetura são responsáveis por 70,66% dos acessos à memória. Na arquitetura x86 predominam as instruções do tipo *move*, as quais representam, em média, 88,06% do total de instruções executadas. Na arquitetura x86, 95,94% dos acessos a memória de instrução são ocasionados por apenas 256 bytes, enquanto que uma memória de dados de 8 KB é responsável por 73,62% dos acessos à memória. Resultados experimentais apontaram grande potencial de otimização do acesso à memória e grande possibilidade de exploração do espaço de projeto para a arquitetura ARM com utilização de uma memória SPM.

Palavras-chave: Sistemas embarcados. Plataformas virtuais. Aplicações embarcadas. Modelos de arquiteturas. Acesso à memória.

Abstract

OLIVEIRA, Lizandro de Souza. **Estudo e avaliação de arquiteturas de processadores quanto ao uso de memória em sistemas embarcados.** 2015. 110 f. Dissertação (Mestrado em Ciência da Computação). Universidade Federal de Pelotas, Pelotas.

Embedded systems have many constraints and requirements. In addition to the significant cost, these systems must meet hard performance restrictions and limitations in the energy consumption. The memory system is one of the major contributing factors to the performance and power consumption in embedded software. Virtual platforms offer an alternative to hardware prototypes and they offer many advantages such as early development, performance and accessibility. There are many virtual platforms and simulators that provide several processor architecture models. This work presents a study and evaluation on the use of memory in embedded processor architectures. This work presents different memory optimization techniques for embedded systems. ARM and x86 architectures were evaluated to access to memory and energy consumption. For these evaluations, instructions and data trace were generated by different embedded applications from MiBench package. The evaluation tools of embedded processor architectures used were Simics and CACTI. The results show that 34,42% of all instructions to ARM architecture are *load* and *store* instructions. In addition, 64,41% of access, on average, to instruction memory are caused by only 8KB of the program, as 64KB for the percentage is 84,58%. In relation to data memory, 8KB of data for this architecture are responsible for 70,66% of the memory accesses. In the x86 architecture predominate move instructions, which represent, on average, 88,06% of executed instructions. In the x86 architecture, 95,94% of access to instruction memory are caused by only 256 bytes, while an 8 KB of data memory is responsible for 73,62% of the memory accesses. Experimental results showed great optimization potential of memory access and high possibility of the design space exploration for ARM architecture with the use of an SPM memory.

Keywords: Embedded systems. Virtual platforms. Embedded applications. Architecture models. Memory access.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

CISC	*Complex Instruction Set Computers*
CPI	Ciclos por instrução
CPU	*Central Processing Unit*
DRAM	*Dynamic RAM*
E/S	Entrada e saída
FFT	*Fast Fourier Transform*
FIFO	*First In First Out*
GPS	*Global Positioning System*
ISA	*Instruction Set Architecture*
MIPS	Milhões de instruções por segundo
MRAM	*Magnetic RAM*
NoC	*Network on Chip*
NVM	*Non-Volatile Memory*
PC	*Personal Computer*
PCM	*Phase Change Memory*
PDAs	*Personal Digital Assistants*
RAM	*Random-Access Memory*
RISC	*Reduced Instruction Set Computer*
RTOS	*Real-Time Operating Systems*
SO	Sistema Operacional
SoC	*System on Chip*
SPM	*Scratch Pad Memory*
TLB	*Translation Look-Aside Buffer*
ULA	Unidade Lógica Aritmética
UML	*Unified Modeling Language*
VLSI	*Very Large Scale Integration*

Lista de Símbolos

α	Atividade de chaveamento
C_L	Capacitância de carga
f	Frequência do sinal de relógio
V	Tensão de operação

Sumário

1 Introdução

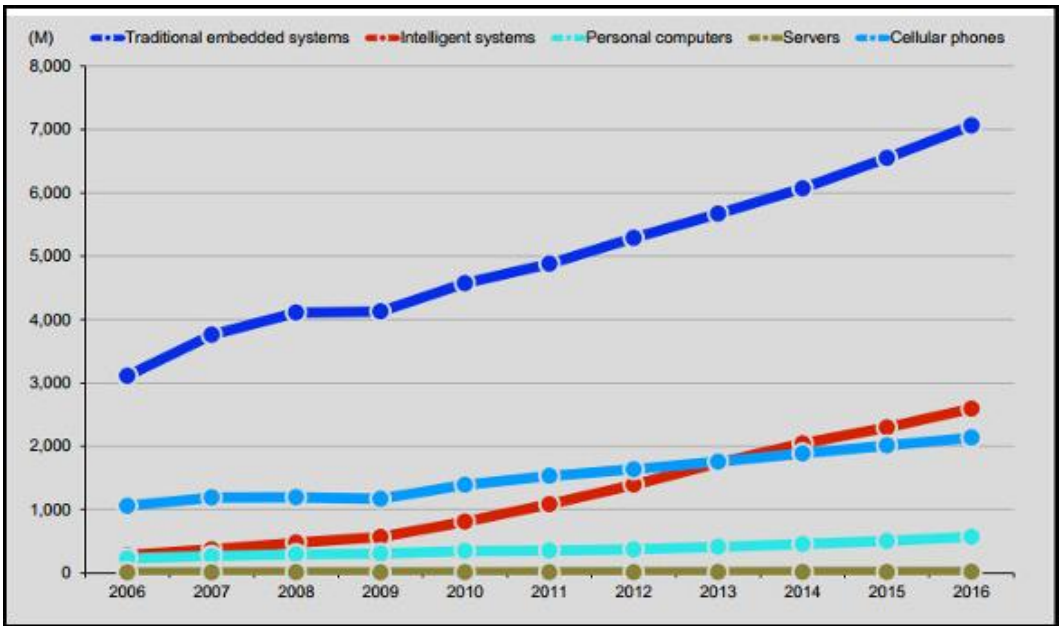
O desenvolvimento tecnológico das últimas décadas originou uma nova realidade: o uso intensivo de sistemas computacionais. Esses sistemas auxiliam os seres humanos em suas tarefas do cotidiano, seja através do uso de computadores ou de equipamentos que possuam sistemas computacionais para o seu controle. Quando utilizados em sistemas ainda mais complexos, são chamados de sistemas embarcados, pois constituem parte de um todo e desenvolvem tarefas específicas (MARWEDEL, 2006). Segundo (WOLF, 2005), sistemas embarcados têm caráter dedicado e possuem funcionalidade específica dentro de um sistema mais complexo.

Sistemas computacionais embarcados podem ser encontrados nas mais diversas aplicações, tais como em *smartphones*, *tablets*, relógios, fornos de micro-ondas, automóveis, equipamentos eletrodomésticos, dentre outros (CARRO e WAGNER, 2003). Nas últimas décadas o crescimento do mercado para estes sistemas tem sido muito grande. Fabricantes de sistemas embarcados preveem o crescimento para o mercado de sistemas embarcados, conforme apresentado na Figura 1.

Indicadores apresentaram crescimento de 497 milhões de novos dispositivos móveis em 2014 (CISCO, 2015). O número destes dispositivos cresceu para 7,4 bilhões em 2014, acima dos 6,9 bilhões em 2013. Os *smartphones* foram responsáveis por aproximadamente 88% deste crescimento com 439 milhões de novas unidades. Ainda conforme (CISCO, 2015), o tráfego global de dados móveis cresceu 69% em 2014. O tráfego de vídeo móvel, por sua vez, representou 55% do total do tráfego de dados móveis até o final de 2014. São observadas, ainda, projeções de crescimento de dispositivos móveis como *smartphones* e *tablets*.

Sistemas embarcados são heterogêneos por envolverem *software* e *hardware*. As aplicações embarcadas possuem características que os diferem dos demais sistemas. Características como: sistemas dedicados, sistemas reativos, confiabilidade, restrições de tempo real, tamanho do código, desempenho, baixo consumo de energia, físicas (tamanho e peso), devem ser consideradas no desenvolvimento de um sistema embarcado.

Figura 1. Previsão da AMD para o mercado de embarcados (em milhões de unidades).



Fonte: (CLARK, 2013).

O projeto deste tipo de sistema computacional é extremamente complexo, por envolver conceitos até agora pouco analisados pela computação de propósito geral. Por exemplo, as questões da portabilidade e do limite de consumo de potência sem perda de desempenho, a baixa disponibilidade de memória, a necessidade de segurança e confiabilidade, a possibilidade de funcionamento em uma rede maior, e o curto tempo de projeto tornam o desenvolvimento de sistemas computacionais embarcados uma área específica de pesquisa (WOLF, 2005).

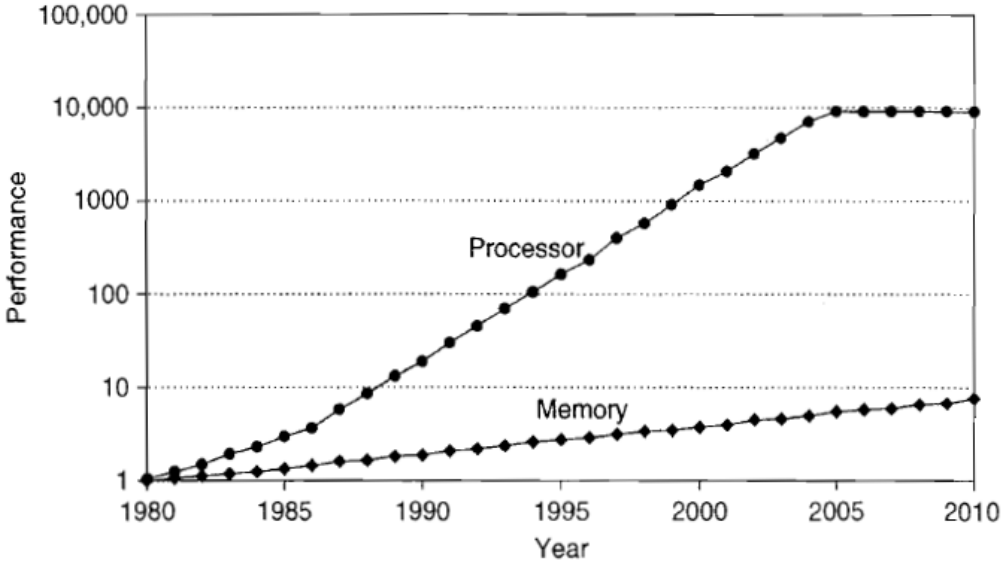
Diferentemente dos computadores de propósito geral, tais como os computadores pessoais, os sistemas embarcados são projetados para executar tarefas específicas. Estes sistemas, além do custo significativo, geralmente devem atender restrições rígidas de desempenho, bem como limitações no seu consumo energético. Considerando o exposto por (WOLF e KANDEMIR, 2003), o sistema de memória é um dos principais fatores que contribuem para o desempenho e consumo de energia em *software* embarcado. Dessa forma, são necessárias técnicas de otimização de memória a fim de atingir o desempenho desejado e também reduzir o consumo energético em sistemas embarcados. Esta redução no consumo energético assume especial importância em sistemas embarcados que utilizam bateria, como em dispositivos móveis e portáteis.

Diversas técnicas de otimização de memória tem sido propostas pela comunidade científica com a finalidade de atingir os desafios existentes, configurando-se como importante tema de pesquisa. Estas otimizações são classificadas em diferentes categorias, mas, em geral, referem-se a otimizações de *hardware* ou *software*, podendo ser combinadas (PANDA, CATHOOR, *et al.*, 2001).

Em relação ao *hardware*, são descritas técnicas arquiteturais. Diversos trabalhos concentram-se em melhorar o consumo de energia de memórias *cache*. Outros trabalhos, no entanto, propõem melhorar diferentes características, tecnologias ou arquitetura do sistema. Otimizações de *software* apresentam diversas vantagens para sistemas embarcados. Técnicas como permutação e fusão de *loop* podem melhorar dependência de dados. Projeto de *hardware* e *software* também estão descritos na literatura, assim como otimizações para aplicações específicas a fim de reduzir o consumo de energia.

Em sistemas de propósito geral, é conhecido o gargalo existente entre o desempenho do processador e da memória tradicional, como pode ser observado na Figura 2.

Figura 2. Diferença de desempenho entre memória e processador.



Fonte: (HENNESSY e PATTERSON, 2011).

Como resultado dessa diferença de desempenho, tem-se o aumento do tempo de acesso aos dados armazenados em memória, do ponto de vista do processador, que deve esperar um número maior de ciclos para seguir o processamento. É

necessário, portanto, o uso de uma hierarquia de memória a fim de explorar a localidade de processamento da informação.

Em sistemas embarcados o *gap* de desempenho existente entre processador e memória torna-se ainda mais significante. Projetistas destes sistemas devem considerar não somente desempenho, mas também potência e consumo de energia.

Os trabalhos de (CATTHOOR, RAGHAVAN, *et al.*, 2010), (VERMA e MARWEDEL, 2007) e (HENNESSY e PATTERSON, 2011) demonstram que a arquitetura de memória representa uma parcela significativa de área e também de consumo energético em circuitos integrados. Como descrito por CATTHOOR *et. al* (2010), estas arquiteturas representam de 40% a 60% do total de energia consumida por um processador embarcado de conjunto de instruções.

O custo de projeto de *hardware* dedicado tem aumentado de forma desproporcional quando comparado ao retorno de muitas aplicações. *Software* embarcado sendo executado sobre processadores eficientes e de baixo consumo não fornecem somente uma alternativa viável, mas também apresentam vantagens relevantes à equipe de projeto quanto às funcionalidades e desenvolvimento. O surgimento dos dispositivos de vários núcleos melhorou ainda mais o desempenho destas soluções. Esta situação criou uma tendência para a utilização de plataformas baseadas em processadores de propósito geral com funcionalidades personalizadas realizadas em *software* embarcado para muitos projetos de produtos eletrônicos. Desta forma, a quantidade e a complexidade do *software* embarcado têm aumentado significativamente.

Neste sentido, ferramentas para avaliação da eficiência de processadores embarcados são requeridas. Algumas destas ferramentas utilizam uma plataforma virtual que permite simular a execução de uma aplicação sobre um processador embarcado fornecendo dados sobre esta execução. A plataforma virtual representa um dos avanços mais importantes no mundo da computação embarcada. Empresas que projetam circuitos integrados (CI's), por exemplo, não consideram a criação de um CI sem, primeiramente, utilizar simulações (IMPERAS, 2014).

1.1 Objetivos e resultados esperados

Conforme apresentado na seção anterior, reduzir o acesso à memória em sistemas embarcados é necessário a fim de reduzir o consumo energético nestes dispositivos, além de minimizar o tempo de acesso a dados. O objetivo deste trabalho é

avaliar arquiteturas de processadores embarcados quanto ao acesso à memória. Para isso, é avaliado o comportamento em relação ao acesso a memória de instruções e de dados das arquiteturas ARM e x86.

Como metodologia é utilizada a ferramenta Simics para a geração dos traces das diferentes arquiteturas e como *benchmark* é utilizado o pacote MiBench. Também são desenvolvidos *scripts* para geração e análise dos *traces* a fim de classificar as instruções. Para obter os dados de latência da memória e consumo energético é utilizada a ferramenta CACTI.

A partir do trace de instruções e de dados dos *softwares benchmarks* foi possível analisar as arquiteturas com intuito de identificar possibilidades de otimizações de código e propor soluções para redução do acesso à memória nestas arquiteturas.

1.2 Organização do trabalho

Este trabalho está dividido em seis capítulos. No segundo capítulo é realizada uma revisão sobre os diferentes conceitos acerca de sistemas embarcados, suas principais características e restrições. No terceiro capítulo é apresentada revisão sobre hierarquia de memória e também apresenta um estudo sobre técnicas de otimização do sistema de memória para sistemas embarcados presentes na literatura.

O quinto capítulo apresenta a metodologia utilizada neste trabalho, sendo elencadas as ferramentas, arquiteturas e aplicações abordadas. O sexto capítulo apresenta os resultados obtidos. Por fim, o sétimo capítulo apresenta as conclusões deste trabalho e sugestões de pesquisas futuras.

Após a conclusão encontram-se, nos apêndices de A a L, as rotinas criadas para geração e análise dos traces de instruções e as tabelas com os resultados obtidos para os testes de cada arquitetura.

2 Sistemas embarcados

Conforme mencionado anteriormente, sistemas embarcados são sistemas de caráter dedicado e que possuem funcionalidade específica dentro de um sistema mais complexo (WOLF, 2005). Para (MARWEDEL, 2006) sistemas embarcados são sistemas de processamento de informação incorporados em um produto maior.

2.1 Características dos sistemas embarcados

Estes sistemas diferenciam-se dos demais porque eles devem abordar alguns requisitos e restrições. Alguns destes itens são muito importantes no projeto de sistemas embarcados, tais como (MATTOS, 2007):

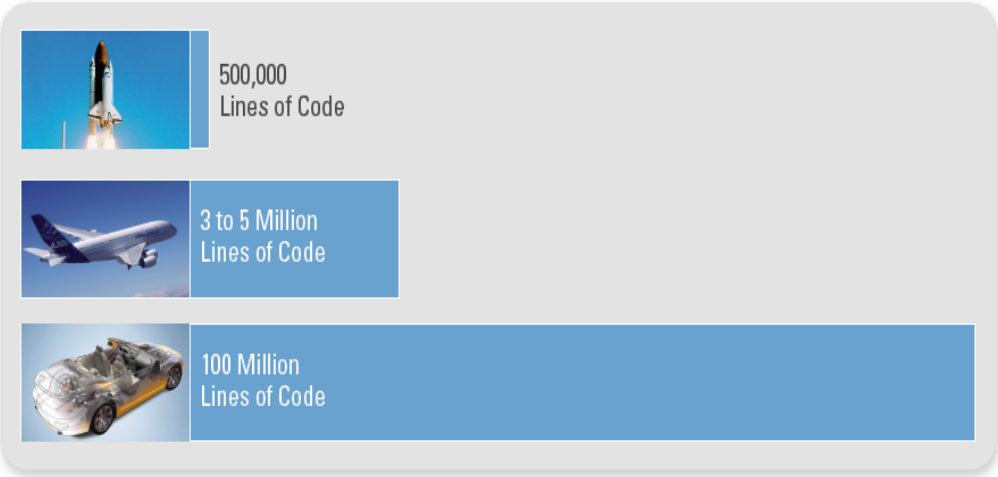
- desempenho: com a crescente complexidade dos sistemas embarcados, o desempenho do sistema geralmente é requisito importante a ser alcançado;
- tempo real: os sistemas embarcados geralmente estão envolvidos em tarefas onde os requisitos de tempo são rigorosos e devem responder a eventos externos, com uma avaliação precisa do tempo de execução;
- dimensão física e peso: estes sistemas estão inseridos em sistemas maiores onde o requisito de portabilidade é muito importante, assim processadores embarcados devem ser pequenos e leves;
- confiabilidade: alguns destes sistemas estão envolvidos em situações críticas onde os erros podem ter consequências dramáticas, envolvendo vidas humanas e enormes quantias de dinheiro;
- tempo e custo de projeto: atualmente, o tempo de colocação do produto no mercado é muito importante. Assim, o tempo e custo de projeto devem ser reduzidos tanto quanto possível;
- baixo consumo de energia: o consumo de energia apresenta uma questão crítica especialmente em dispositivos portáteis, pois estes utilizam bateria.

2.2 Áreas de aplicação dos sistemas embarcados

Sistemas embarcados são utilizados em áreas fundamentais, conforme citado por (MARWEDEL, 2006): eletrônica automotiva, eletrônica de aeronaves, trens, telecomunicações, sistemas médicos, aplicações militares, sistemas de autenticação, eletrônica de consumo, dentre outros. Algumas destas áreas são brevemente descritas a seguir.

Na área de eletrônica automotiva é apresentada grande quantidade de eletrônica embarcada, como o controle do sistema de *air bag*, o sistema de injeção eletrônica, o sistema ABS (*anti-lock braking system*), o GPS, dentre outros. Nos aviões atuais possuem uma enorme quantidade de sistemas que necessitam ser confiáveis. Em aeronaves os sistemas eletrônicos estão presentes desde o sistema de controle de voo até os sistemas de informações para os passageiros. Com a finalidade de observar a quantidade de código presente nas áreas de eletrônica automotiva e em aeronaves, é apresentada a Figura 3.

Figura 3. Quantidade de linhas de código em *software* embarcado diferentes projetos.



Fonte: (REDING, 2013).

Aparelhos celulares ou o controle de grandes centrais de telefonia são exemplos, na área das telecomunicações, de equipamentos baseados em sistemas computacionais embarcados. Outra área citada, a de sistemas médicos, apresenta diversos equipamentos que possuem eletrônica embarcada. Estes equipamentos contribuem para o avanço da medicina e apresentam desafios de projeto adicionais além daqueles já citados para os sistemas embarcados.

Dentre as aplicações militares podemos citar equipamentos de alta tecnologia, tais como o controle de radares ou até mesmo o controle de mísseis. Por fim, podemos mencionar a automação (industrial, residencial e predial) que fornece o controle de processos industriais, sistemas de controle de climatização, segurança e iluminação.

2.3 Metodologias para projeto de sistemas embarcados

Conforme (WOLF, 2005) e considerando os requisitos no projeto de sistemas embarcados, verifica-se que a metodologia de projeto é importante, essencialmente, por três razões. Primeiramente porque nos permite manter indicadores do projeto a fim de garantir que todas as ações estão sendo realizadas. Em segundo lugar porque permite desenvolver ferramentas de projeto auxiliado por computador e, com isso, automatizar etapas. Como terceira razão para a importância da metodologia de projeto pode-se citar a maior facilidade de comunicação para os membros de uma equipe de projeto.

Em geral, para as diferentes metodologias de projeto são utilizados vários níveis de abstração, iniciando de um nível mais alto até o desenvolvimento do produto. Ainda conforme (WOLF, 2005) o desenvolvimento de sistemas embarcados pode seguir duas abordagens: *top down* ou *bottom-up*. Na abordagem *top down* o projeto inicia com uma visão mais abstrata e, a partir de refinamentos sucessivos, o projeto prossegue para uma visão mais detalhada, obtendo-se o sistema propriamente dito. Na abordagem *bottom-up*, no entanto, o projeto inicia com pequenos componentes e a partir destes componentes obtém-se o sistema completo. Em projetos reais utiliza-se um misto destas duas abordagens, em especial devido ao reuso.

O projeto de um sistema embarcado, como exposto anteriormente, passa por diversos níveis de abstração, sendo orientado a um determinado domínio de aplicação como, por exemplo, aplicações orientadas a dados ou controle. Para a representação de cada domínio existe um modelo de computação mais adequado, conforme apresentado por (EDWARDS, LAVAGNO, *et al.*, 1997). Diferentes linguagens de especificação e de projeto têm sido adotadas para tratamento destes níveis de abstração e modelos de computação. Existem linguagens dedicadas ao domínio de *software*, linguagens dedicadas ao domínio de *hardware* e linguagens mistas.

Para a especificação funcional é importante que a linguagem seja executável, para fins de validação. As descrições de arquitetura, no entanto, devem ser feitas em

linguagens que possam ser utilizadas como entrada para processos de síntese automática de *hardware*, além de serem simuláveis.

Linguagens de programação como C e C++ têm sido bastante utilizadas. A linguagem C embora não seja ideal do ponto de vista do conjunto de requisitos para linguagens de especificação, em especial devido ao grau de abstração e à generalidade dos domínios de aplicação, ela tem a vantagem de permitir a geração de software para um grande número de processadores utilizados no contexto de sistemas embarcados. A adoção da orientação a objetos, como em C++, se por um lado é vantajosa do ponto de vista de especificações de alto nível onde reuso e especialização de componentes são características muito interessantes, por outro lado causa problemas para a síntese de *hardware* ou tamanho do *software* (CARRO e WAGNER, 2003).

Como citado, além das linguagens dedicadas ao domínio de software temos as linguagens dedicadas ao domínio de *hardware* e as linguagens mistas. Para a descrição de *hardware* as linguagens mais populares são VHDL e Verilog. Como linguagens mistas podemos citar, dentre outras, SystemC, SDL e Esterel. Estas linguagens tentam atender simultaneamente os requisitos de cobertura de múltiplos domínios e níveis de abstração em relação a implementações de *hardware* e *software*.

Ainda referente às linguagens, muitas vezes torna-se útil conceituar em diagramas as tarefas dos diferentes níveis de abstração do projeto de um sistema embarcado. Para tanto, tem-se a linguagem de modelagem unificada (*Unified Modeling Language* – UML). A UML é uma linguagem formal com alto nível de abstração que pode ser utilizada para modelar cada fase do projeto de sistemas embarcados (WOLF, 2005).

Ao trabalhar com sistemas embarcados, percebe-se, no entanto, que dados, programas e configurações FPGA devem ser armazenados em algum tipo de memória. Isto deve ser feito de forma eficiente, seja por meios eficientes em tempo de execução, tamanho de código ou energeticamente eficientes. A eficiência para o tamanho de código, por exemplo, requer um bom compilador e pode ser melhorada com compactação de código (MARWEDEL, 2006).

2.4 *Software* embarcado

O *software* é dito embarcado quando o mesmo é desenvolvido para rodar em um dispositivo ou plataforma específica e que será distribuído como parte de um produto (MATTOS e BRISOLARA, 2009).

Conforme já destacado neste trabalho, limitações e restrições presentes nos dispositivos e plataformas impõem exigências ao projeto de *software* embarcado. Estas restrições são mais rígidas do que aquelas consideradas para o *software* tradicional presente em computadores pessoais.

Em muitos casos o *software* embarcado é desenvolvido para um sistema específico. Assim, é possível realizar diversas otimizações de modo mais simples, quando comparadas a *softwares* multi-propósito ou multi-plataforma. Entretanto, com a crescente onda de dispositivos móveis, como celulares e *tablets*, este processo de otimização torna-se mais complexo (BANK, 2012).

Além do *software* de aplicação, alguns sistemas embarcados também operam com um sistema operacional (SO) (MATTOS e BRISOLARA, 2009). Em sistemas embarcados que possuam SO o desenvolvimento do *software* torna-se ainda mais simplificado, porém mais complexo. São exemplos de sistemas operacionais embarcados *Windows CE* (Microsoft Corporation, 2009), *Vx Works* (Wind River, 2009) e *Ecos* (Ecos, 2009), além de várias distribuições de Linux para sistemas embarcados, tais como uClinux e LynxOS, ou ainda do Android, um SO para dispositivos móveis.

Sistemas embarcados possuem diversos requisitos, conforme já citado neste trabalho. Dentre os diferentes requisitos encontram-se os de tempo real, os quais afetam principalmente o escalonamento de tarefas. Nestes casos os sistemas operacionais devem ser de tempo real, conhecidos como RTOS. Outra preocupação refere-se à qualidade de *software* embarcado, considerando que geralmente este *software* é vendido junto com um produto, o que pode resultar em alto custo no caso de algum tipo de *recall* de produtos. Além disso, o *software* embarcado pode fazer parte de um sistema de missão crítica envolvendo vidas humanas e/ou bens de alto valor, fatores que aumentam a exigência de alta qualidade (MATTOS e BRISOLARA, 2009).

O interesse por aplicações baseadas em software cresceu principalmente pelo aumento no poder computacional das plataformas de *hardware* que possibilitou mover mais funcionalidades para o *software*. Outro fator motivacional foi o aumento dos

custos de desenvolvimento de *hardware* que motivou o reuso de uma mesma plataforma em diferentes produtos.

A utilização desta abordagem de projeto baseado em *software* proporciona flexibilidade e portabilidade, enquanto diminui o tempo de projeto. O *software* de um sistemas embarcado é diferente de um *software* de aplicação (*software* executado em um microcomputador) devido, principalmente, à forte interconexão com os dispositivos de *hardware* periféricos, às restrições impostas pela aplicação na qual ele está inserido e ao modo como ele é desenvolvido (GOMES, 2010).

Ainda segundo (GOMES, 2010) o *hardware* de um sistema computacional de uso geral é controlado por um complexo sistema operacional, que gerencia diversos dispositivos, como, memória, vídeo, teclado e meios de comunicação enquanto que o *software* desenvolvido precisa apenas se deter a problemas referentes a aplicação. O *software* de um sistema embarcado, por outro lado,acessa diretamente o *hardware* e deverá ser escrito de forma a se adaptar aos recursos de *hardware* disponíveis, visando a execução de uma tarefa especifica.

O estudo e avaliação de sistemas embarcados torna-se ainda mais relevante ao considerar dados de mercado, os quais preveem que o mercado de sistemas embarcados deve atingir U$ 194,27 bilhões em 2018 (NEWSWIRE, PR, 2013).

2.5 *Hardware* embarcado

O *hardware* para sistemas embarcados é bem menos padronizado que o *hardware* dos computadores pessoais. Devido a natureza das aplicações dos sistemas embarcados, a implementação destes sistemas normalmente é realizada em uma grande variedade de *hardware* integrada em um único *chip* (SoC). Assim, esta seção apresenta apenas uma introdução dos principais componentes.

Os componentes de *hardware* frequentemente encontrados nos sistemas embarcados são os seguintes (MATTOS e BRISOLARA, 2009):

- sensores e atuadores: sensores podem ser projetados para uma variedade de medidas físicas. Em sistemas embarcados podem ser encontrados sensores de aceleração (utilizados em *videogames*), de chuva (para-brisa de automóveis), sensores de imagem (câmeras digitais), dentre outros;

- memórias: dados e programas devem ser armazenados em memória da maneira mais eficiente possível, visto que os sistemas embarcados possuem uma quantidade de memória disponível bem menor quando comparados aos computadores pessoais (PC);
- unidades de processamento: existe grande variedade de elementos de processamento, tais como ASICs, lógica reconfigurável, processadores de propósito geral e processadores de uso específico (DSPs e ASIPs);
- elementos de comunicação: utilizados para a interconexão das diversas unidades de processamento, memória e sensores/atuadores. Esta comunicação pode ser realizada dentro do *chip* (com utilização de barramentos ou de NoCs) ou ainda entre dispositivos (com fios, fibra óptica e *wireless*). Uma série de requisitos devem ser considerados nos meios de interconexão, tais como tempo real, largura de banda e tolerância a falhas.

Em projetos de *hardware* para sistemas embarcados, diversos fatores devem ser considerados, tais como desempenho, custo (incluindo o custo de projeto e fabricação), tamanho físico e peso, consumo de energia e potência. O consumo de energia e potência é um fator limitante na funcionalidade oferecida por alguns sistemas embarcados, principalmente os portáteis, que operam por bateria (SHEARER, 2008). Este problema de consumo é causado por diversos fatores: demanda de equipamentos com mais funcionalidades, maior necessidade de processamento e tempo de vida das baterias, dentre outros.

Diante do exposto, prolongar a vida útil da bateria em dispositivos móveis como *smartphones* é um grande desafio, pois estes dispositivos reúnem todas as características citadas anteriormente. Com isso muitos pesquisadores têm como objeto de estudo metodologias de otimização e eficiência em energia (THOMPSON, WHITE, *et al.*, 2009).

Logo, o baixo consumo de potência deve ser uma característica de projeto em todos os componentes de sistemas embarcados tanto em *hardware* como *software*, assim a redução de consumo dos diversos componentes levará a uma alta redução do consumo total. O consumo de potência é a taxa na qual a energia é consumida (SHEARER, 2008). Com uma capacidade fixa de armazenamento de energia de uma bateria, o consumo de potência determina diretamente o tempo de vida da bateria do dispositivo embarcado. O grande desafio é realizar o maior uso da unidade central de

processamento (CPU) quanto possível, consumindo o mínimo de energia necessário. O consumo de potência afeta diretamente o custo do hardware, já o consumo de energia afeta diretamente o tempo de vida da bateria. Costumeiramente, os conceitos de potência e energia são confundidos. Em (1) e (2) são definidas potência e energia, respectivamente.

$$Potência = \frac{Trabalho}{Tempo}\ (Watts) \quad (1)$$

$$Energia = Potência\ .Tempo\ (Joules) \quad (2)$$

A potência utilizada por um dispositivo é a energia consumida por unidade de tempo. Já a energia é a integral da potência. Desta maneira, as baterias armazenam uma dada quantidade de energia e o objetivo é minimizar a quantidade de energia necessária para realizar cada tarefa. A energia pode ser definida como indicado em (3).

$$EnergiaTotal = \int PotênciaTotal\ dt \quad (3)$$

Desta forma, reduzir o consumo de potência também reduz o consumo de energia (considerando o mesmo intervalo de tempo). No entanto, em alguns casos, um pequeno aumento na potência pode resultar em uma grande redução no tempo de execução e reduzindo, assim, o consumo de energia. Em alguns casos, minimizar a potência significa reduzir o consumo de energia, porém esta afirmação não é sempre verdadeira (MATTOS e BRISOLARA, 2009).

A potência pode ser dividida em dois tipos: potência estática e dinâmica. A primeira é proveniente de características de circuitos CMOS e independe da atividade do circuito. A segunda, no entanto, é resultado da atividade de chaveamento do circuito, sendo definida de acordo com (4):

$$P = \alpha . C_L . V^2 . f \quad (4)$$

A variável α, como indicada em (4), é a atividade de chaveamento. A variável C_L é a capacitância de carga, V é a tensão de operação e f a frequência de relógio. Nota-se, a partir de (4), que a potência dinâmica está diretamente ligada à tensão e frequência de operação do circuito. Assim, geralmente os processadores embarcados não operam com frequências elevadas, o que normalmente ocorre nos computadores

pessoais. Manter frequências baixas significa manter o consumo de potência baixo (MATTOS e BRISOLARA, 2009).

As unidades de processamento são responsáveis pelo processamento de informações. Em geral, sistemas embarcados são compostos de diversos elementos de processamento que podem ser ASICs, lógica reconfigurável ou processadores. Estas tecnologias possuem diferentes características em termos de desempenho, eficiência energética e tempo de projeto.

Os processadores, ou *cores*, são elementos de processamento onde sua funcionalidade é definida pelo *software*. Estes dispositivos variam desde processadores simples até processadores altamente sofisticados, podendo ser classificados como processadores para computadores (processadores de propósito geral), processadores embarcados, processadores para DSP e microcontroladores.

2.5.1 Processadores embarcados

Os processadores embarcados, que possuem maior interesse na investigação deste trabalho, são componentes projetados para uma grande variedade de aplicações, especialmente eletrônica de consumo e comunicação, tipicamente com 32 bits. Estes processadores devem ser eficientes e não necessitam possuir o conjunto de instruções compatível com os computadores pessoais. Algumas das arquiteturas utilizadas em embarcados são ARM, PowerPC, MIPS, 68K e x86.

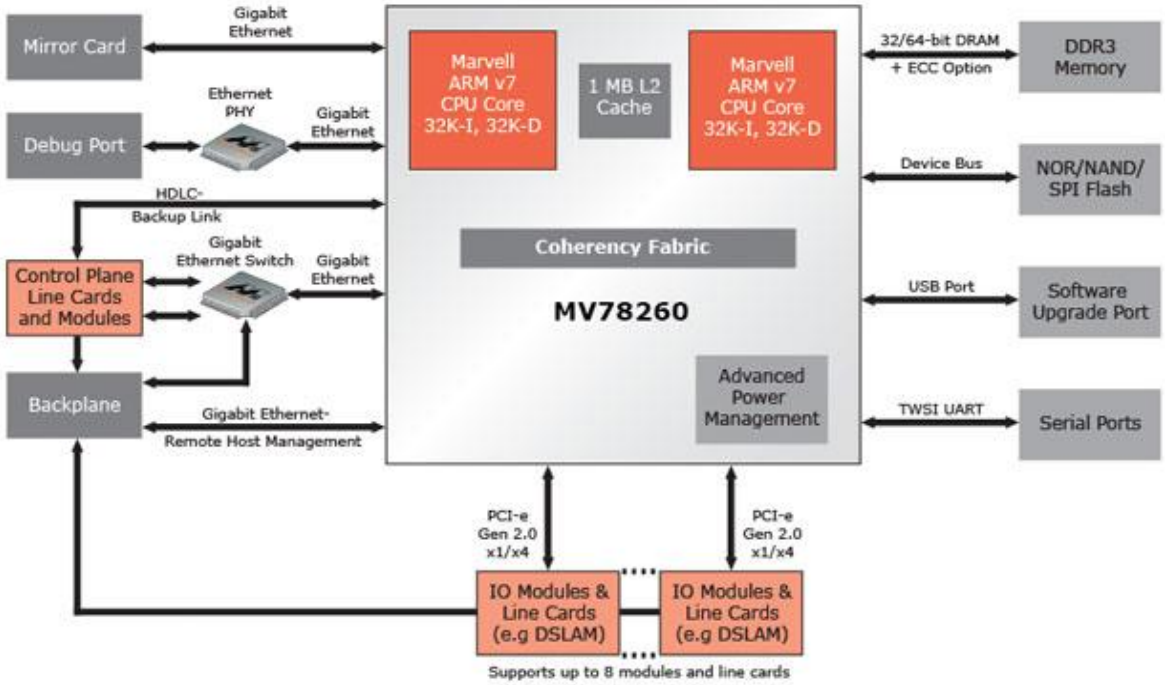
Diversas características impulsionaram o projeto de processadores ARM. Em primeiro lugar, sistemas embarcados portáteis exigem alguma forma de energia fornecida por baterias. O processador ARM foi especialmente projetado para ser pequeno a fim de reduzir o consumo de energia e prolongar o funcionamento com bateria, essencial em aplicações tais como telefones celulares e PDAs (SLOSS, SYMES e WRIGHT, 2004). Além disso, sistemas embarcados utilizam dispositivos de memória lentos e de baixo custo, o que produz economias substanciais ao projeto.

Outro requisito importante é a redução da área ocupada pela matriz do processador embarcado. Para um único *chip*, quanto menor a área usada pelo processador embarcado, maior será o espaço disponível para os periféricos especializados. Isto reduz o custo de projeto e fabricação uma vez que é necessário um menor número de *chips* discretos para o produto final.

Neste contexto, são observados no mercado processadores embarcados projetados para obter melhor desempenho, baixo consumo de energia e altos níveis de

integração. Estes processadores embarcados utilizam projetos avançados da arquitetura ARM. A Figura 4 apresenta o diagrama em blocos de um processador multicore projetado para aplicações de computação em nuvem de classe empresarial.

Figura 4. Diagrama em blocos - ARMADA XP.



Fonte:(MARVELL, 2015).

Atualmente energia e potência são restrições primárias de projeto e o cenário da computação é significantemente diferente: o crescimento de *tablets* e *smartphones* executando ARM está superando o crescimento de *desktops* e *laptops* que utilizam x86. Além disso, uma ISA ARM tradicionalmente de baixo consumo de energia está entrando no mercado de servidores de alto desempenho, enquanto a ISA x86, tradicionalmente de alto desempenho, está entrando no mercado de dispositivos móveis de baixo consumo de energia. Assim, a investigação a fim de descobrir se a ISA tem um papel intrínseco no desempenho ou na eficiência energética é cada vez mais importante (BLEM, MENON e SANKARALINGAM, 2013).

3 Sistemas e Técnicas de Otimização de Memória

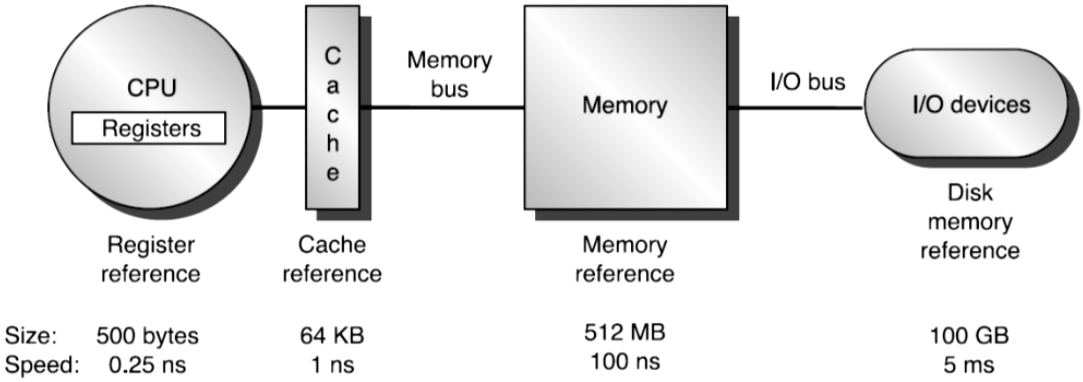
Conforme mencionado por (WOLF e KANDEMIR, 2003) e já citado anteriormente neste trabalho, o sistema de memória é um dos principais fatores que contribuem para o desempenho e consumo de energia em *software* embarcado. Neste contexto surge a investigação de hierarquias de memória.

O principal motivo para explorar hierarquias de memória deve-se ao fato de que grandes memórias necessitam de mais energia por acesso e também são mais lentas do que memórias pequenas. Dessa forma, hierarquias de memória podem ser exploradas a fim de obter bom tempo de execução e boa eficiência energética. A razão fundamental para isso é que grandes memórias necessitam mais energia por acesso e também são mais lentas do que memórias menores (MARWEDEL, 2006). O uso de memória *cache*, no entanto, provoca alguns problemas para aplicações embarcadas, principalmente nas questões de previsibilidade (MATTOS e BRISOLARA, 2009).

O conceito de hierarquia de memória explora o princípio da localidade (HENNESSY e PATTERSON, 2011). Este princípio ocorre no tempo (localidade temporal) e no espaço (localidade espacial). O princípio da localidade se baseia no fato de que, num intervalo virtual de tempo, os endereços virtuais gerados por um programa tendem a ficar restritos a pequenos conjuntos do seu espaço. Isto se deve a iterações, sequenciamento das instruções e estruturas em bloco. Assim sendo, é desejável manter os itens mais recentemente utilizados na memória mais rápida e o mais próximo possível do processador.

Na localidade temporal há uma tendência a que um processo faça referências futuras a posições feitas recentemente. Já na localidade espacial existe uma tendência a que um processo faça referências a posições na vizinhança da última referência. O princípio da localidade, aliado ao fato de que quanto menor um *hardware* mais rápido ele é, conduziu a hierarquias baseadas em memórias de diferentes desempenhos e tamanhos. A Figura 5 mostra uma hierarquia de memória multinível, incluindo tamanhos típicos e tempos de acesso.

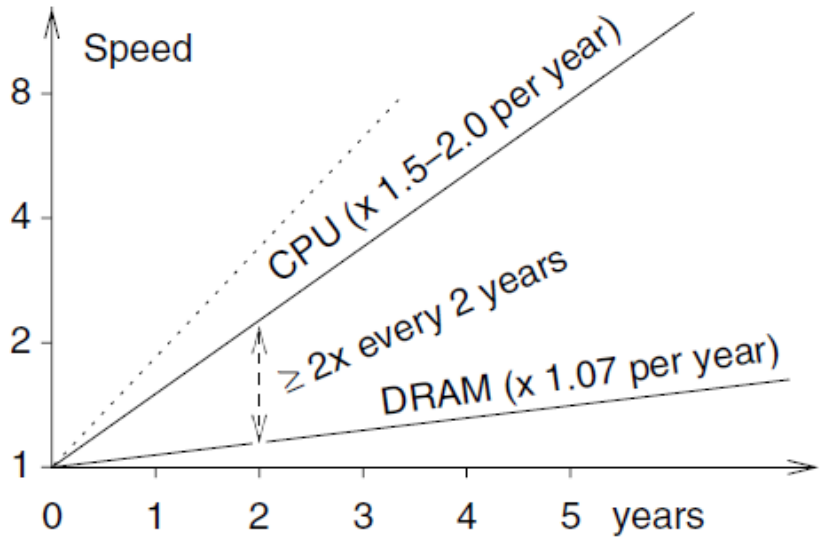
Figura 5. Níveis em uma hierarquia de memória típica em computadores embarcados, *desktop* e servidores.



Fonte: (HENNESSY e PATTERSON, 2011).

A importância da hierarquia de memória aumentou com o avanço no desempenho dos processadores. Enquanto o desempenho das memórias está aumentando em um fator de 1,07 por ano, o desempenho dos processadores possuem um fator de aumento de 1,5 a 2 por ano (MARWEDEL, 2006). Isto significa que a diferença de desempenho entre o processador e a memória é cada vez maior, conforme ilustrado na Figura 6.

Figura 6. Distância crescente entre o desempenho do processador e da memória.

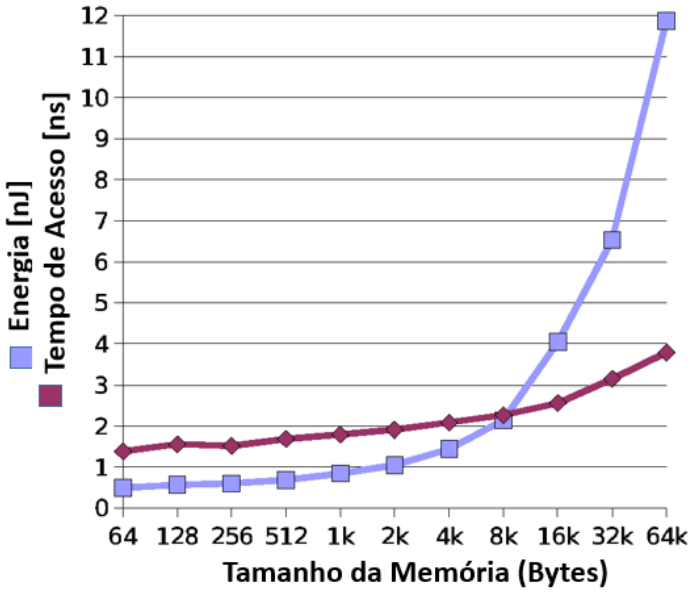


Fonte: (MARWEDEL, 2006).

Dessa forma, é importante a utilização de memórias menores e mais rápidas que atuam como *buffer* entre a memória principal e o processador. Uma combinação de pequenas memórias, que contenham dados e instruções usados com frequência e uma maior quantidade de memória que contém os dados e instruções restantes, geralmente é mais eficiente energeticamente do que uma única grande memória. Estas são memórias *cache*, as quais visam resolver o problema de desempenho de memória e processador. Contudo, o uso de memórias *cache* em algumas classes de sistemas embarcados apresentam problemas devido a previsibilidade e também ao consumo energético.

Um outro problema associado as memórias, principalmente em sistemas embarcados, é o tempo de acesso e consumo energético considerando o tamanho da memória. A Figura 7 apresenta o gráfico que demonstra um aumento significativo no consumo energético em função do tamanho da memória na tecnologia desenvolvida para as memórias.

Figura 7. Comportamento energético e tempo gasto em acesso em relação ao tamanho da memória.



Fonte: Adaptado de (MARWEDEL, 2013).

As tecnologias de memória emergentes, devido à seu maior desempenho, maior densidade e não volatilidade estão tendo significativo impacto sobre o projeto tradicional de hierarquia de memória (XIE, 2011). Observada a importância do sistema de memória para sistemas embarcados, uma grande atenção tem sido dada para a

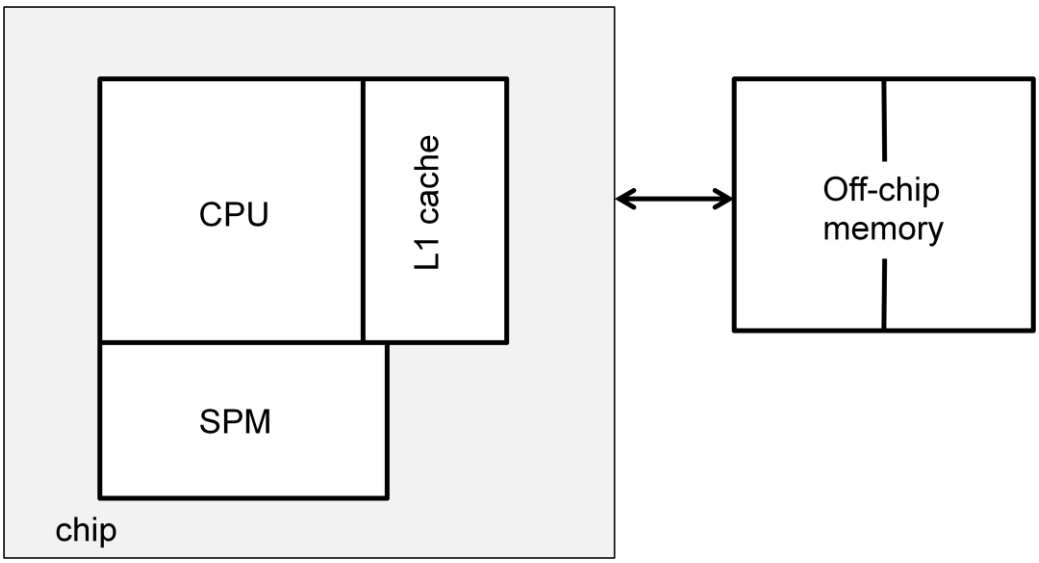
otimização das características de memória de *software* embarcado. Neste sentido, técnicas têm sido propostas, tanto em projetos de *hardware* quanto em projetos de *software*, além de metodologias propostas para projetos que consideram tanto *hardware* quanto *software*.

3.1 Técnicas de Otimização de Memória

Nesta seção são apresentadas as técnicas de otimização do sistema de memória para sistemas embarcados presentes na literatura. Como citado, os sistemas de memória estão entre os principais fatores que contribuem para o desempenho e consumo de energia em *software* embarcado. O consumo de energia do sistema de memória assume especial importância em sistemas embarcados que utilizam bateria.

A
Figura 8 apresenta os principais componentes em relação a memória atualmente encontrados em sistemas embarcados. Estes sistemas possuem uma memória principal *off-chip* e podem incluir uma memória *cache* e ainda uma *scratch pad memory* (SPM). A maioria dos sistemas embarcados utiliza somente um único nível de *cache*, porém com a necessidade de aumento de desempenho, podem existir vários níveis de *cache*. *Scratch pad memories* (SPMs) tem sido propostas como um tipo de memória *on-chip* para aumento de desempenho.

Figura 8. Memórias em um sistema computacional embarcado.



Fonte: (WOLF e KANDEMIR, 2003)

Neste contexto, técnicas têm sido propostas, tanto em projetos de *hardware* quanto em projetos de *software*, além de metodologias propostas para projetos que consideram tanto *hardware* quanto *software*.

A seguir serão apresentadas pesquisas referentes às técnicas de otimização de memória existentes. Estas técnicas foram divididas em uma categoria de *hardware*, onde serão abordados trabalhos em nível arquitetural, e outra categoria de *software*, onde serão relacionadas técnicas para otimizar projetos de *software* embarcado.

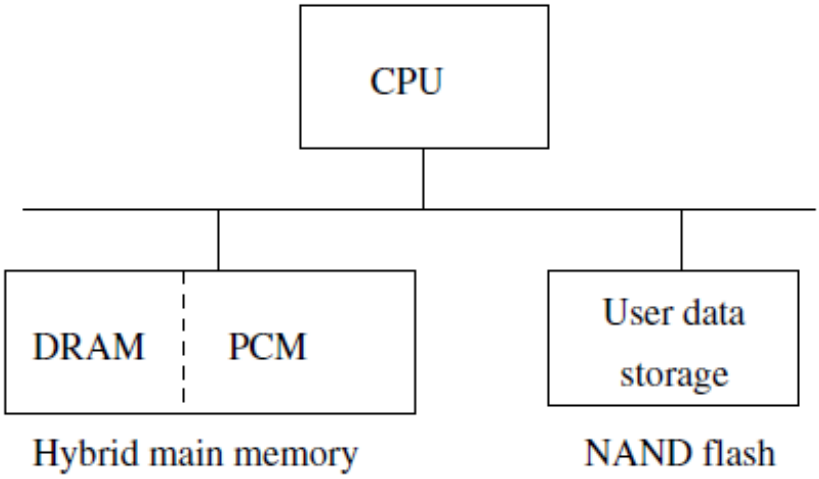
3.1.1 Otimização de *hardware*

As técnicas existentes para otimização de memória em nível arquitetural estão presentes na literatura, porém em menor volume quando comparadas às técnicas de otimização de *software*.

Memórias NVM, tais como memória flash, PCM e MRAM, são empregadas em diversos dispositivos móveis e sistemas embarcados (HUANG, LIU e XUE, 2011). Estas memórias são boas candidatas para serem utilizadas no lugar de memórias DRAM como memória principal em sistemas embarcados e possuem diversas características desejáveis. No entanto, além das vantagens é preciso considerar suas desvantagens, tais como tempo de vida limitado pelo número de vezes que ela é apagada e também o desempenho assimétrico entre as operações de leitura e escrita neste tipo de memória.

Em sistemas embarcados, especialmente em dispositivos móveis que utilizam bateria, energia é uma das métricas de desempenho mais importantes. Dessa forma, tornasse interessante a utilização de memórias PCM para otimização energética em dispositivos móveis. No trabalho de (SHAO, LIU, *et al.*, 2012) é proposta uma arquitetura híbrida para o sistema de memória que utiliza memória PCM para substituir a memória DRAM tanto quanto possível. O objetivo desta proposta é reduzir a energia do sistema com a utilização da baixa potência das memórias PCM quando estas encontram-se no estado de espera (*standby*). A Figura 9 mostra esta arquitetura do sistema utilizando uma memória híbrida PCM/DRAM como memória principal em sistemas embarcados. Diferente da memória principal baseada em DRAM, esta memória híbrida pode armazenar código e dados do mesmo usuário utilizando a não volatilidade da PCM.

Figura 9 Arquitetura do sistema com memória híbrida PCM/DRAM.



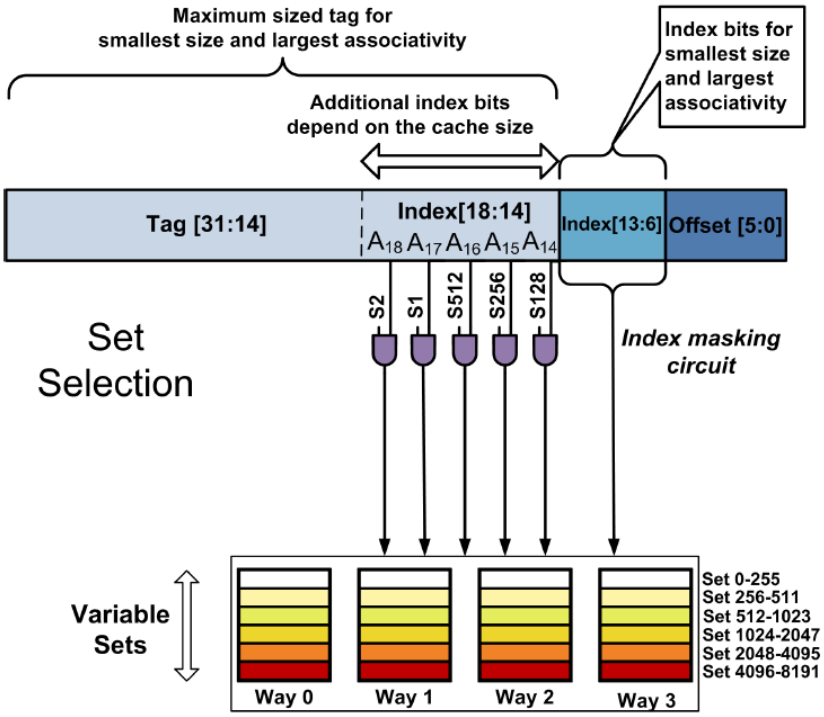
Fonte: (SHAO, LIU, *et al.*, 2012).

É sabido, conforme já citado neste trabalho, que sistemas embarcados de baixo consumo energético exigem que os projetistas ajustem os parâmetros do processador a fim de evitar o desperdício de energia sem significativa redução de desempenho.

O trabalho de (SUNDARARAJAN, JONES e TOPHAM, 2011) é proposta uma arquitetura de gerenciamento de *cache* por conjunto (*set*) e por via (*way*) para reconfiguração em tempo de execução chamada de *cache* inteligente. Esta memória *cache* permite a reconfiguração de tamanho e de associatividade. A variação do tamanho da *cache* é feita através dos circuitos de seleção de conjunto, conforme Figura 10. Paralelamente a isto, a alteração da associatividade é realizada através da lógica de seleção de via, como mostrado na Figura 11.

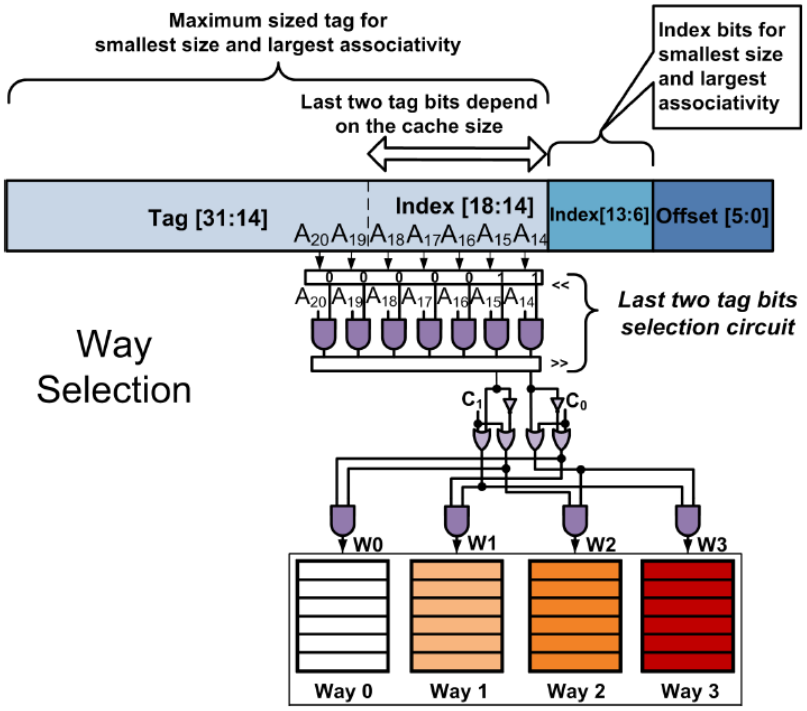
As Figuras 10 e 11 mostram como cada endereço é mapeado na memória *cache* proposta por (SUNDARARAJAN, JONES e TOPHAM, 2011) para uma memória *cache* nível 2 de 2MB. São estes dois circuitos complementares utilizados em paralelo que executam o mapeamento, permitindo que o endereço seja encaminhado tanto para o conjunto quanto para a via correta.

Figura 10 Organização arquitetural da *cache* inteligente (seleção de conjunto).



Fonte: (SUNDARARAJAN, JONES e TOPHAM, 2011).

Figura 11 Organização arquitetural da *cache* inteligente (seleção de via).



Fonte:(SUNDARARAJAN, JONES e TOPHAM, 2011).

Como tanto o tamanho como a associatividade da memória *cache* variam, o mesmo ocorre com o número de bits necessários para as *tags*. Nesta arquitetura são armazenadas as *tags* com tamanho máximo para cada linha, ou seja, para a *cache* com menor tamanho e maior associatividade.

Com a abordagem proposta no trabalho de (SUNDARARAJAN, JONES e TOPHAM, 2011) estes demonstram uma redução de energia de 17% na *cache* de dados e uma redução de 34% na *cache* nível 2 com redução de desempenho inferior a 2% quando comparada com a *cache* inicial.

Uma otimização muito utilizada em sistemas embarcados é o uso de memórias *scratchpad* (SPM). Uma SPM é um *array* de memória com a parte de decodificação e lógica de decodificação (BANAKAR, STEINKE, *et al.*, 2002). A ideia por trás de uma *scratchpad* é manter objetos de memória mapeados pelo compilador no seu último estágio. A SPM ocupa uma parte do espaço de endereçamento e o restante é ocupado pela memória principal. A grande vantagem das SPMs é que não existe necessidade de checagem da disponibilidade do dado ou instrução como as memórias *cache*, eliminando a questão dos rótulos (*tags*) e comparadores existentes nas *caches*. Isso contribui significativamente na redução da área e consumo de energia.

No trabalho de (BANAKAR, STEINKE, *et al.*, 2002) esta memória foi utilizada a fim de reduzir o consumo de energia e de área apresentando redução de 40% no consumo energético e uma redução média de 46% para a área. Quando comparada à memória *cache*, a memória *scratch pad* é mais eficiente em desempenho, potência e área, além de possuir a vantagem adicional de melhor tempo de previsibilidade (LI, GAO e XUE, 2005).

3.1.2 Otimização de *software*

O sistema de memória em geral determina o comportamento de um sistema embarcado no que se refere ao desempenho, à potência e ao custo de fabricação. Uma grande variedade de técnicas foram desenvolvidas nas últimas décadas a fim de otimizar projetos de *software* embarcados visando a melhoria das características citadas.

No trabalho de (PANDA, CATHOOR, *et al.*, 2001) são apresentadas técnicas de otimização de dados e de memória para sistemas embarcados. Este trabalho aborda otimizações de memória independentes de plataforma e técnicas aplicáveis às

estruturas de memória. Estas otimizações independentes de plataforma são feitas pelo compilador, por exemplo, com transformações de loop. Por outro lado, o trabalho de (WOLF e KANDEMIR, 2003) aborda apenas em técnicas de *software* para melhorar o sistema de memória. É preciso observar, no entanto, que as técnicas de *software* apresentadas nestes dois trabalhos são baseadas em técnicas de compilação.

O projeto de *software* e compilação de um sistema embarcado pode tirar vantagem de dois fatos importantes: o *hardware* de destino da implementação é conhecido; e podemos gastar mais tempo e esforço computacional para otimizar o software (WOLF e KANDEMIR, 2003).

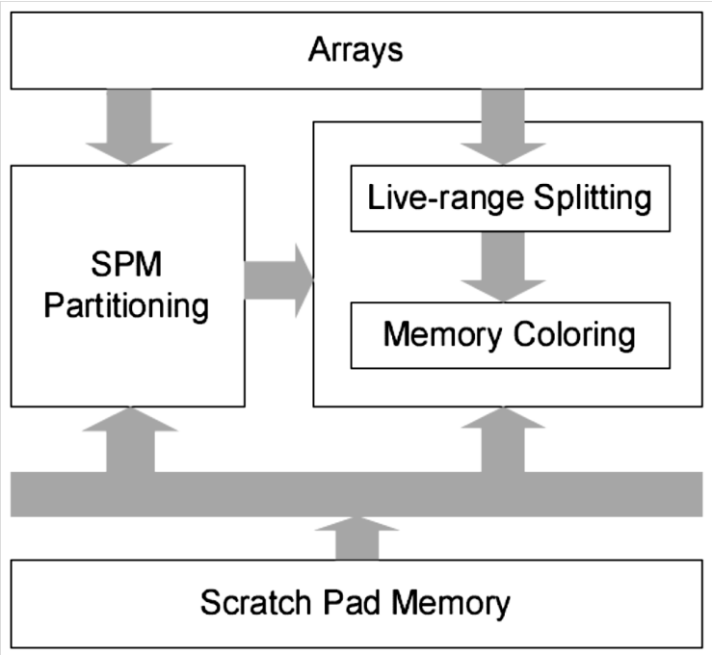
Existem várias categorias que podem ser utilizadas para classificação dos diversos tipos de otimização de memória propostos. Algumas otimizações, como técnicas orientadas a *loop*, são principalmente voltadas para o comportamento de dados de programas. Outras como o procedimento de *inlining*, são voltadas para o comportamento do código. Deve ser lembrado, porém, que um tipo de otimização pode ter implicação em outro domínio. Como exemplo, podemos citar o desenrolamento de laços (*loop unrolling*) que pode ser utilizado para melhorar o desempenho de dados, porém pode prejudicar o desempenho do código do programa, pois reduz a reutilização de instruções. A técnica de *loop unrolling*, conforme descrita por (WOLF e KANDEMIR, 2003), consiste em replicar o corpo do loop várias vezes, ajustando o *loop* ao código. No trabalho de (WHITE, 2011) é exemplificado o uso desta técnica.

Conforme citado na seção anterior, quando comparada à memória *cache*, a memória *scratch pad* é mais eficiente em desempenho, potência e área, além de possuir a vantagem adicional de melhor tempo de previsibilidade (LI, GAO e XUE, 2005). Existem, porém, duas tarefas interligadas que devem ser resolvidas: mapeamento da matriz de endereços para o espaço SPM e geração da declaração de transferência de dados.

Apesar de ser uma técnica de otimização de memória por *hardware*, existem abordagens de *software* para memórias SPM. O trabalho de (LI, GAO e XUE, 2005) propõe uma metodologia para o problema de gerenciamento de memória *scratch pad*, o qual pode ser resolvido por um algoritmo de coloração de grafos para a alocação de registradores. A metodologia proposta neste trabalho está mostrada na Figura 12. Esta metodologia é formada por três componentes principais: *SPM Partitioning, Live-Range Splitting* e *Memory Coloring*. Os componentes *Live-Range Splitting* e*Memory Coloring* visam resolver, respectivamente, a geração da declaração de transferência de dados e

o mapeamento da matriz de endereços para o espaço SPM. Já a componente *SPM Partitioning* é utilizada para o particionamento da memória SPM.

Figura 12. Metodologia de gerenciamento SPM.



Fonte:(LI, GAO e XUE, 2005).

Algumas otimizações, no entanto, destinam-se em reduzir o tamanho de código. Estas otimizações tentam identificar a ocorrência de comum de segmentos de código e automaticamente construir sub-rotinas transparentes ao usuário. Outras técnicas, com foco nos tamanhos dos dados tentam reutilizar uma dada posição de memória para tantas variáveis quanto possível.

A maior parte da literatura sobre compiladores concentra-se na otimização de uma única aplicação. No entanto, muitos sistemas embarcados executam vários programas simultaneamente. Estes processos são executados periodicamente e a taxas diversas. A interação entre processos pode ser prejudicial para o desempenho e para potência. Uma vez que a interação entre processos ocorre durante um longo período de tempo e durante um grande número de estados, técnicas especiais são necessárias para reconhecer efeitos importantes (WOLF e KANDEMIR, 2003).

Diversas linguagens têm sido propostas para lidar com o fluxo de dados, descrever paralelismo, reduzir a quantidade de controle em um programa, dentre outros aspectos de programação que podem causar dificuldades para a compilação. Algumas destas linguagens foram desenvolvidas para processamento de sinal. Outras

poderiam ser aplicadas a *software* embarcado, mesmo não sendo desenvolvidas para este fim.

Em relação aos compiladores, este tipo de otimização tem por objetivo melhorar a localidade de dados e pode ser dividida em dois grandes grupos, conforme (WOLF e KANDEMIR, 2003): 1) otimizações que visam reduzir a latência da memória; e 2) otimizações que visam esconder a latência de memória.

No primeiro grupo encontram-se técnicas que modificam o padrão de acesso ao programa, o leiaute de variáveis de memória, ou ambos. Neste sentido, a maior parte destas técnicas tenta melhorar a localidade de dados reduzindo, assim, o número de acessos a níveis mais baixos na hierarquia de memória. Esta melhoria na localidade de dados é feita por transformações de *loop* e/ou por transformações no leiaute de dados. Um exemplo deste tipo de otimização é mostrado para o código da Figura 13, o qual é gerado a partir de uma aplicação de processamento de imagem:

Figura 13. Código de iteração de Jacobi.

```
for (J=1;J < N-1;J++)
      for (I=1;I < N-1;I++)
            B[I][J] = (A[I-1][J]+A[I+1][J]+A[I][J-1]+A[I][J+1]) * (1/k);
```

Percebe-se que, para valores fixos do loop *J*, as sucessivas iterações do *loop* interno *I* neste código acessam elementos de diferentes linhas e isto ocorre para cada referência no corpo do *loop*. Um compilador de otimização pode transformar este código para o código apresentado na Figura 14:

Figura 14. Código de iteração de Jacobi otimizado.

```
for (I=1;I < N-1;I++)
      for (J=1;J < N-1;J++)
            B[I][J] =(A[I-1][J]+A[I+1][J]+A[I][J-1]+A[I][J+1]) *(1/k);
```

Agora, para cada referência, as sucessivas iterações do *loop* interno acessam elementos da mesma linha, embora diferentes referências possam acessar linhas diferentes. Portanto, pode-se esperar um comportamento muito melhor no que se refere à localidade de dados a partir deste código transformado.

A permutação de *loop* pertence a um conjunto de classes de transformações unificado chamado transformações unimodulares. Estas transformações incluem

permutação de *loop*, inversão de *loop* e inclinação de *loop*. No trecho de código da Figura 15 é mostrado um exemplo para a fusão de *loop*.

Figura 15. Fragmento de código antes da fusão de *loop*.

```
for (I=0;I < N;I++)
      k = k + A[I] * B[I];
for (I=0;I < N;I++)
      c = c * (A[I]+B[I]);
```

Dada uma memória *cache* de pequena capacidade, este fragmento pode ler cada elemento das matrizes A e B na *cache* duas vezes, sendo uma para cada loop. Com isso, o número de cargas da *cache* é reduzido pela metade a partir da fusão destes dois *loops*, como apresenta a Figura 16.

Figura 16. Fragmento de código após fusão de *loop*.

```
for (I=0;I < N;I++)
{
k = k + A[I] * B[I];
c = c * (A[I]+B[I]);
}
```

As transformações de *loop* possuem como vantagem a forte teoria acerca do assunto e uma série de implementações sólidas tanto do meio acadêmico quanto da indústria.

O segundo grupo de técnicas, no entanto, procura colocar os dados na memória rápida antes que estes dados sejam realmente necessários. Caso isto seja conseguido, o dado pode ser acessado a partir da memória *cache* quando for necessário. Neste sentido são propostos na literatura algoritmos baseados nestas técnicas, conforme descrito no trabalho de (WOLF e KANDEMIR, 2003).

No trabalho de (WOLF e KANDEMIR, 2003) são abordadas, ainda, otimizações para códigos baseados em ponteiros. Segundo os autores, estes códigos apresentam especial dificuldade para um compilador otimizado, uma vez que não é possível, em geral, determinar o tempo de compilação das posições de memória que serão apontadas por uma dada variável de ponteiro. Isto impede a análise de dependência de dados e, com isso, impede o compilador de modificar o código. Dessa forma, são relatados trabalhos que visam otimizar códigos baseados em ponteiros.

Conforme já abordado neste trabalho, sistemas embarcados possuem rigorosas restrições referentes à energia e área, além de suas exigências de desempenho. Por este motivo, a otimização de compiladores para sistemas embarcados devem levar em consideração estas restrições.

Devem ser estudadas, portanto, técnicas clássicas de otimização de compilador voltadas ao desempenho. Com base neste estudo, pode-se decidir se estas otimizações são adequadas do ponto de vista energético. Transformações lineares de loop tais como troca e reversão de *loop*, podem melhorar o desempenho da memória *cache*. A partir do ponto de vista energético, através da aplicação destas transformações, pode-se esperar uma redução na energia de memória devido melhor utilização da *cache*. Para decidir se uma dada transformação deve ou não ser aplicada, no entanto, é preciso considerar seu impacto sobre outros componentes do sistema.

Percebe-se grande quantidade de técnicas referentes às memórias *cache* pois, como já mencionado, estas memórias visam resolver o problema de desempenho de memória e do processador. Além disso, as memórias *cache* apresentam benefícios de desempenho, além de reduzir significativamente o consumo de energia. No entanto, alguns trabalhos estão assumindo diferentes abordagens em relação às memórias.

O trabalho de Huang (HUANG, LIU e XUE, 2011) foca em sistemas embarcados utilizando memória não volátil (NVM) como memória principal, sendo propostas técnicas de alocação de registradores com recomputação a fim de reduzir o número de instruções armazenadas. Neste trabalho são propostos novos métodos, baseados em coloração de grafos, para reduzir atividades de escrita em NVM. Resultados experimentais mostram que o algoritmo proposto (PRS) tem a maior eficiência em reduzir atividades de escrita e o custo total quando comparado a métodos anteriores. Conforme mostrado, o algoritmo proposto no trabalho de (HUANG, LIU e XUE, 2011) obtém uma redução média de 25% quando comparado à técnica de coloração de grafos tradicional.

No trabalho de Catthoor (CATTHOOR, DE GREEF, *et al.*, 1998) foi desenvolvida uma metodologia para o projeto de sistemas de memória customizados, especialmente para aplicações multimídia que exigem *streaming*. A metodologia otimiza tanto o projeto de *hardware* quanto de *software*. Esta metodologia passa por altos níveis de abstração da aplicação até otimizações de baixo nível.

4 Metodologia e Ferramentas do Trabalho

O sistema de memória é um dos principais fatores que contribuem para o desempenho e consumo de energia em *software* embarcado. Neste contexto, este trabalho tem por objetivo analisar arquiteturas de processadores embarcados em relação ao comportamento do acesso a memória de instruções e de dados a partir do *trace* gerados para determinados *softwares benchmarks* e, assim, identificar possibilidades de otimizações de desempenho e redução do consumo energético.

Neste capítulo está descrita a metodologia empregada e também as ferramentas utilizadas para atingir os objetivos propostos. Este desenvolvimento pode ser dividido em duas grandes etapas. A primeira etapa consistiu na geração e análise do *trace* de instruções das diferentes aplicações selecionadas para uma arquitetura ARM e X86 e a segunda etapa na geração e análise do *trace* de dados das diferentes aplicações/arquiteturas selecionadas. Foram desenvolvidos *scripts* para a geração e análise dos *traces* de cada uma das aplicações.

Após a geração dos *traces* de instruções das diferentes aplicações, foi criada rotina para análise destes *traces*. Estas rotinas foram necessárias devido ao grande número de instruções para análise e classificação. Por fim, as instruções foram classificadas em grupos (*load*, *store*, aritméticas, lógicas, etc.). Esta classificação foi feita a partir da análise do conjunto de instruções das arquiteturas.

Após concluir as etapas descritas anteriormente, foi possível a análise e discussão dos resultados obtidos para as diferentes aplicações. Estes resultados são apresentados e discutidos no capítulo 5.

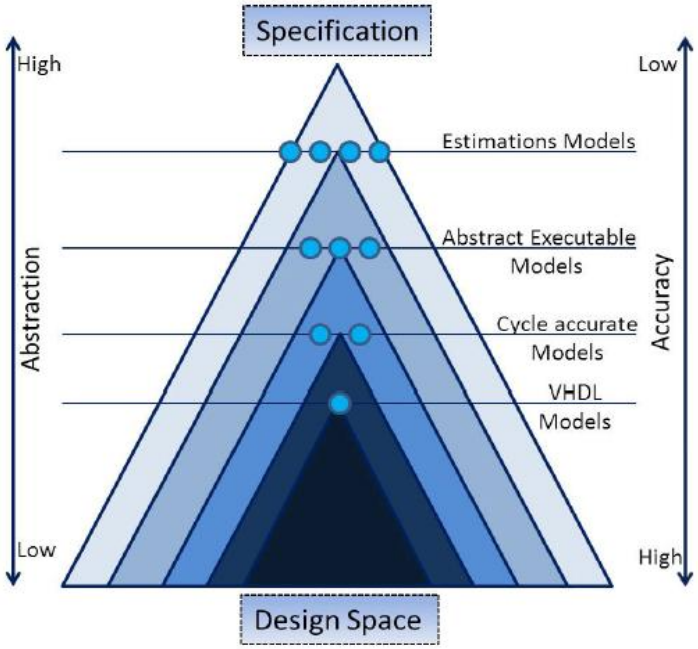
Por fim, foi realizada a análise do potencial de otimização no acesso à memória utilizando uma memória *scratchpad*. Para os cálculos do desempenho e energia foi utilizada a ferramenta CACTI (MURALIMANOHAR, BALASUBRAMONIAN e JOUPPI, 2009) na versão 6.0.

4.1 Ferramentas utilizadas

Existem diferentes abordagens para a exploração do espaço de projeto. A Figura 17 apresenta uma pirâmide dos níveis de abstração que compõem um projeto de sistema a partir da especificação de uma possível solução ótima. O ponto ótimo de

projeto é obtido a partir de uma abordagem que especifica arquiteturas em um nível muito detalhado, utilizando linguagens de descrição de *hardware* tais como VHDL ou Verilog. Esta abstração limitada oferece alta precisão, porém impõe severas limitações sobre a exploração do espaço de projeto, além de consumir muito tempo (BUTKO, GARIBOTTI, *et al.*, 2012).

Figura 17. Pirâmide de níveis de abstração.



Fonte: (BUTKO, GARIBOTTI, *et al.*, 2012).

4.1.1 Simics

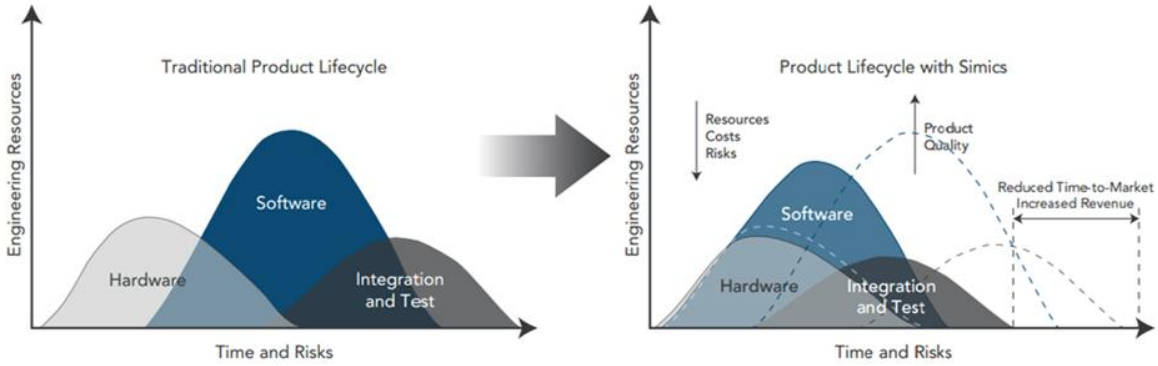
O ambiente de simulação Simics é uma plataforma para simulação de um sistema computacional completo que busca encontrar um equilíbrio entre precisão e desempenho (MAGNUSSON, CHRISTENSSON, *et al.*, 2002). Simics é um simulador no qual toda a arquitetura modelada é programável como, por exemplo, processadores, memórias, conjunto de instruções, entre outras, conforme (ENGBLOM e EKBLOM, 2006).

Este simulador permite às empresas a adoção de novas abordagens para o ciclo de desenvolvimento de produto, ocasionando redução de riscos do projeto, *time-to-market* e custos de desenvolvimento (WIND RIVER, 2013). A

Figura 18 compara o desenvolvimento tradicional de produtos com o ciclo de desenvolvimento utilizando o Simics.

Com o Simics é possível a modelagem e parametrização de componentes em nível de arquitetura, podendo ser utilizado os módulos disponíveis no simulador ou então pode ser feita a modelagem de novos módulos. Além disso, o simulador oferece total suporte ao sistema operacional, sendo possível a execução de programas e sistemas operacionais completos sem necessidade de modificações (FREITAS, ALVES, *et al.*, 2008).

Figura 18. Comparação do desenvolvimento tradicional e com uso do Simics.



Fonte: (WIND RIVER, 2014).

Uma vez que o simulador Simics não modela todos os componentes de um *hardware* real, mas sim o nível de instruções, a execução no simulador é de uma instrução por ciclo (parâmetro definido pelo IPC padrão da ferramenta), enquanto uma máquina real consegue gerenciar a execução de mais instruções utilizando superescalaridade e outros componentes de desempenho.

Os diversos componentes, tais como barramentos, pré-busca e comportamentos do sistema, não são totalmente simulados. Assim, a simulação não é adequada para a comparação direta entre a máquina real e a máquina simulada, considerando as diferenças temporais entre o sistema real e o simulado. Apesar disso, o simulador é bastante útil para a comparação de cargas de trabalho em diferentes configurações modeladas dentro do próprio simulador, uma vez que as tendências ocorridas nas simulações devem se repetir em sistemas reais.

Diante do exposto, o Simulador Simics fornece um ambiente com determinismo controlado, propício para avaliação de sistemas computacionais futuros. Para simulações mais detalhadas, alguns modos de operação estão disponíveis: *fast* (rápido); *stall* (lento) e *mai* (microarquitetura). Sendo assim é possível observar

informações a respeito de tempo, passos de execução, ciclos de execução, registradores em diferentes graus de complexidade, detalhamento e velocidades de simulação (VIRTUTECH, 2007).

A Figura 19 apresenta a janela de comando do Simics. Nela estão contidos todos os ícones e menus que controlam a ferramenta e também um resumo da sessão de simulação atual. A partir dos menus é possível abrir janelas adicionais como, por exemplo, a janela de linha de comando do Simics, mostrada na Figura 20. A partir desta janela o usuário pode interagir, por linhas de comando, com a ferramenta. Nesta janela são mostrados avisos e mensagens de erro e o usuário pode inspecionar e modificar a simulação. Qualquer procedimento realizado a partir dos menus da janela de controle do Simics poderá ser realizado na interface por linha de comando.

Figura 19. Janela de controle do Simics.

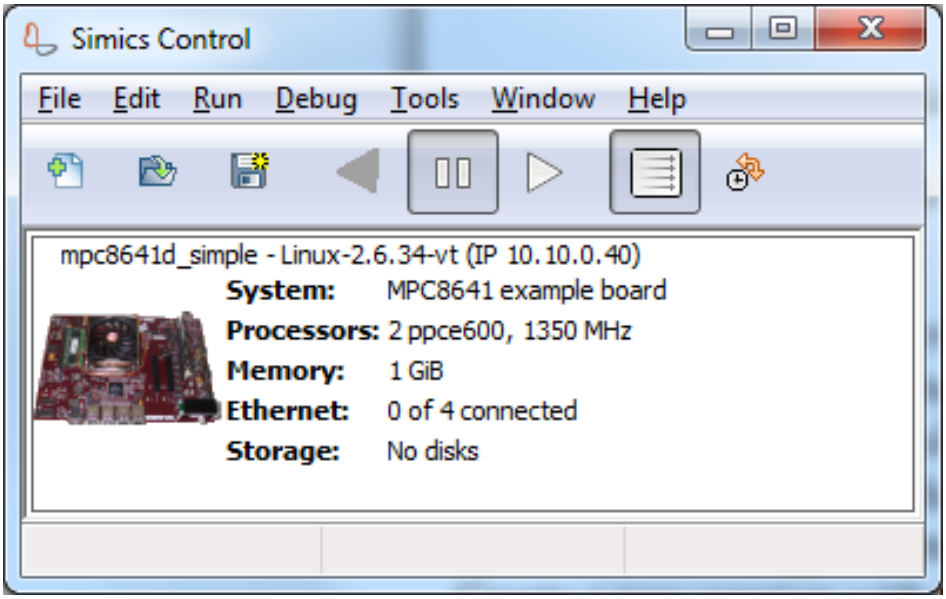
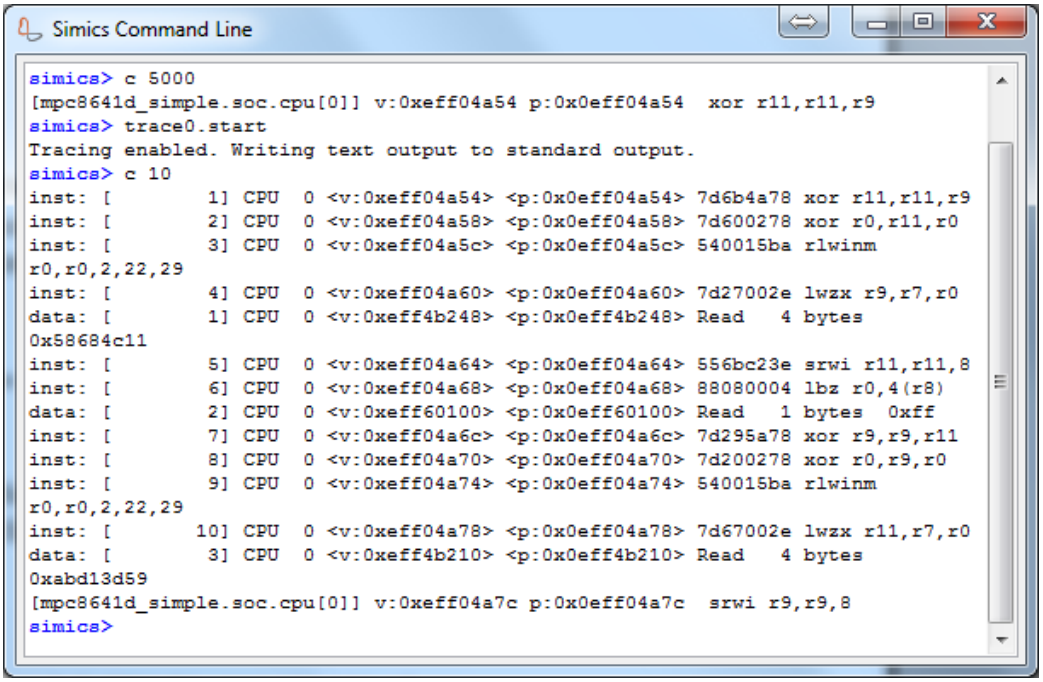


Figura 20. Janela de linha de comandos do Simics.



Após carregar o sistema a ser simulado, é mostrada a janela *serial console*. Esta janela é parte da simulação e está conectada à arquitetura simulada. Nesta janela serão exibidas todas as saídas da máquina simulada. Também é possível interagir o *software* simulado, ao contrário da interface por linha de comando que é utilizada para controlar a própria simulação.

4.1.2 CACTI

O *software* CACTI (THOZIYOOR, AHN, *et al.*, 2008) foi utilizado para estimar parâmetros de memória *SPM* e memória principal. Esta ferramenta está disponível em duas formas: uma versão *web* e uma versão do código-fonte em C++.

Esta ferramenta está sendo continuamente atualizada e, com isso, os resultados das simulações para uma determinada configuração de memória ou tecnologias de *caches* específicas podem sofrer alterações à medida que as novas versões são lançadas e seus *bugs* corrigidos.

Com a utilização dos resultados obtidos na ferramenta CACTI foi possível obter consumo energético e desempenho dos *benchmarks*. Neste trabalho comparamos dois tipos de memória: uma formada por uma memória principal e outra formada por uma memória principal em conjunto com uma memória *scratchpad*.

4.2 Arquiteturas analisadas

Dentre as ferramentas elencadas, diversas arquiteturas podem ser utilizadas. No entanto, várias arquiteturas não são disponibilizadas nas licenças universitárias concedidas pelos desenvolvedores. Assim, foram selecionadas as arquiteturas de acordo com suas características e considerando as limitações expostas.

Para as simulações com a utilização da ferramenta Simics, foram selecionadas as arquiteturas ARM (QSP-ARM) e x-86 (x86-440bx). A arquitetura ARM foi desenvolvida para possibilitar implementações enxutas e, sem deixar de lado o alto desempenho. Isso só é possível pela simplicidade dos processadores ARM. Dentre as principais características da arquitetura, destacam-se as seguintes: processador de 32 bits; 16 registradores de propósito geral; conjunto de instruções extensível com o uso de coprocessadores; baixo consumo de energia e tamanho do núcleo reduzido.

Considerando a aplicação em sistemas embarcados, a característica de baixo consumo de energia de uma arquitetura ARM é desejável para estas aplicações. O ARM é tipicamente um RISC (*Reduced Instruction Set Computer*). Algumas das características interessantes das instruções ARM são as seguintes: conjunto grande e uniforme de registradores; arquiteturas de *load*/*store*; modos de endereçamento simples; uniformidade e tamanho fixo dos campos das instruções; controle da ULA e sobre o *shifter*; instruções de múltiplos *loads*/*stores* para maximizar o desempenho; e execução condicional da maioria das instruções (ARM LIMITED, 2010).

Após definição das ferramentas e escolha das arquiteturas de processadores, foi feita a seleção das aplicações a serem utilizadas.

4.3 Aplicações selecionadas

Após definição das ferramentas e arquiteturas de processadores de interesse, foram selecionadas as aplicações a serem utilizadas nos experimentos. Para as simulações na plataforma Simics, foi utilizado o pacote de aplicações embarcadas *MiBench* (GUTHAUS, RINGENBERG, *et al.*, 2001) que, ao contrário de outros pacotes de *benchmarks*, não é focado em uma área específica e enfatiza a diversidade das aplicações.

O *MiBench* é dividido em seis categorias diferentes: *automotive and industrial control*, *consumer devices*, *office automation*, *network*, *security*, e *telecommunications*. Os programas são escritos em código fonte C, o que garante portabilidade para

qualquer plataforma que tenha suporte a compiladores. A Figura 21 apresenta todos os *benchmarks* presentes nas categorias descritas.

A categoria *automotive and industrial control* destina-se a demonstrar o uso de processadores embarcados em sistemas de controle. Exige desempenho em operações matemáticas básicas, manipulação de bits, dados de entrada/saída e simples organização de dados. Controles de *airbags* e sistemas de sensores são aplicações típicas desta categoria.

Os *benchmarks* presentes na categoria *consumer services*, por sua vez, são destinados a representar os diversos dispositivos de consumo como *scanners*, máquinas fotográficas digitais e *Personal Digital Assistants* (PDAs). A categoria aborda principalmente aplicações multimídia, com algoritmos de codificação/decodificação de JPEGS/MP3's, conversão de formato de imagens, redução de cores de imagem, dentre outros.

As aplicações presentes na categoria *office automation* são essencialmente algoritmos de manipulação de texto utilizados para representar máquinas de escritório como impressoras, aparelhos de *fax* e processadores de texto. Já a categoria *network* representa processadores embarcados em dispositivos de rede, tais como *switches* e roteadores.

Dada a importância crescente na segurança de dados, em especial em atividades de comércio eletrônico, a segurança constitui uma categoria própria no MiBench. A categoria *security* inclui algoritmos comuns para criptografia de dados, descriptografia e *hashing*.

Por fim, a categoria *telecommunications* apresenta *benchmarks* compostos por algoritmos de codificação e decodificação de voz, análise de frequência e um algoritmo *checksum*.

Juntamente com os *benchmarks* é disponibilizado, quando necessário, um conjunto de testes (pequeno e grande). O primeiro serve para simulações leves, enquanto o segundo é utilizado para situações similares a um caso real em termos de tempo e processamento.

Os *benchmarks* do MiBench foram compilados através de *cross-compilers*, ou seja, compiladores capazes de criar código executável para uma plataforma diferente da qual o compilador está sendo executado.

Figura 21. *Benchmarks* do pacote MiBench.

Auto./Industrial	Consumer	Office	Network	Security	Telecomm.
basicmath	jpeg	ghostscript	dijkstra	blowfish enc.	CRC32
bitcount	lame	ispell	patricia	blowfish dec.	FFT
qsort	mad	rsynth	(CRC32)	pgp sign	IFFT
susan (edges)	tiff2bw	sphinx	(sha)	pgp verify	ADPCM enc.
susan (corners)	tiff2rgba	stringsearch	(blowfish)	rijndael enc.	ADPCM dec.
susan (smoothing)	tiffdither			rijndael dec.	GSM enc.
	tiffmedian			sha	GSM dec.
	typeset				

Fonte: (GUTHAUS, RINGENBERG, *et al.*, 2001).

Foram escolhidos alguns *benchmarks*, dentre os presentes no pacote MiBench, adotando como critério ao menos um *benchmark* por categoria. Na Tabela 1 é apresentada uma lista e breve descrição das aplicações selecionadas para as arquiteturas ARM e x86. Para a arquitetura x86 foi selecionado um conjunto menor de aplicações devido a problemas de compilação.

Tabela 1 Lista de aplicações selecionadas.

Benchmark	**Descrição**	**ARM**	**X86**
basicmath	Executa cálculos matemáticos simples	✓	✓
bitcount	Contagem do número de bits de uma matriz	✓	✓
qsort	Algoritmo *quick sort*	✓	✓
susan	Pacote de reconhecimento de imagem	✓	✓
jpeg	Algoritmo JPEG	✓	
mad	Decodificador de áudio MPEG	✓	
tiff2bw	Converte imagem TIFF colorida para preto e branco	✓	
tiff2rgba	Converte imagem TIFF colorida em RGBA TIFF	✓	
typeset	Ferramenta de diagramação geral	✓	
ispell	Verificador ortográfico	✓	
stringsearch	Algoritmo String search	✓	
djikstra	Algoritmo Djikstra	✓	✓
patricia	Algoritmo Patricia	✓	✓

CRC32	Algoritmo CRC32	✓	✓
rijndael	Algoritmo Rijndael	✓	
sha	Algoritmo SHA	✓	✓
fft	Algoritmo FFT	✓	✓

Fonte: Elaborada pelo autor.

4.4 Fluxo da Metodologia

Nesta seção é detalhado o fluxo da metodologia de desenvolvimento do trabalho. A Figura 23 apresenta, como exemplo, o fluxo de desenvolvimento resumido apenas para arquitetura ARM.

A partir da compilação dos *benchmarks* das aplicações foi utilizado a ferramenta Simics para a geração dos traces. Assim, foi realizada a geração dos *trace* de instruções a partir dos *scripts* desenvolvidos. A Figura 22 apresenta um pequeno fragmento do *trace* de instruções gerado a partir da execução da aplicação *fft* sobre a arquitetura ARM.

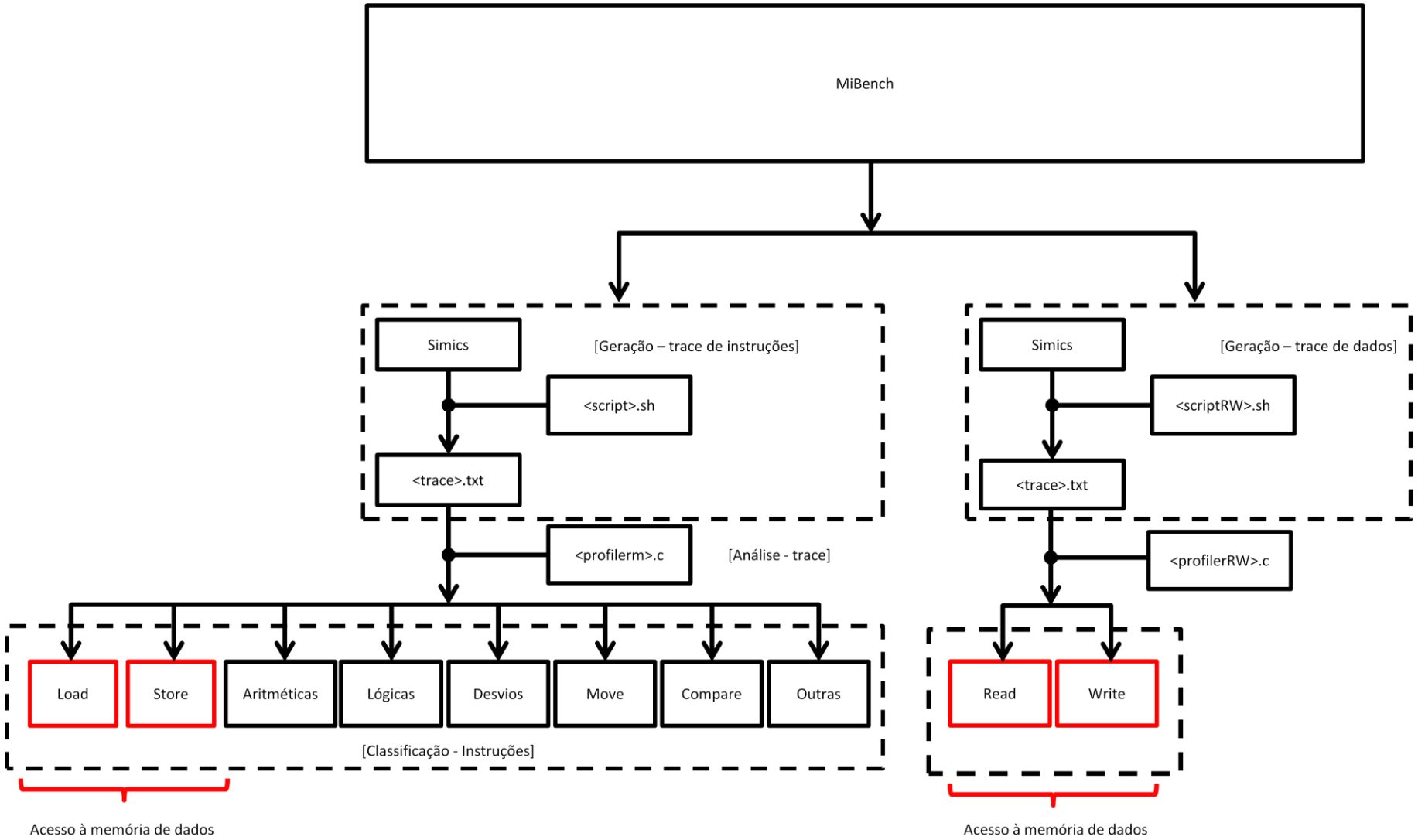
Figura 22. Trace de instruções da aplicação *fft* em arquitetura ARM (fragmento).

```
inst: [         1] CPU  0 <v:0xc0167808> <p:0x010167808> 883095e5 ldr r3, [r5, #136]
inst: [         2] CPU  0 <v:0xc016780c> <p:0x01016780c> 013083e2 add r3, r3, #0x1
inst: [         3] CPU  0 <v:0xc0167810> <p:0x010167810> 033aa0e1 mov r3, r3, LSL #20
inst: [         4] CPU  0 <v:0xc0167814> <p:0x010167814> 233aa0e1 mov r3, r3, LSR #20
inst: [         5] CPU  0 <v:0xc0167818> <p:0x010167818> 883085e5 str r3, [r5, #136]
inst: [         6] CPU  0 <v:0xc016781c> <p:0x01016781c> 3130d4e5 ldrb r3, [r4, #49]
inst: [         7] CPU  0 <v:0xc0167820> <p:0x010167820> 082094e5 ldr r2, [r4, #8]
inst: [         8] CPU  0 <v:0xc0167824> <p:0x010167824> 541094e5 ldr r1, [r4, #84]
inst: [         9] CPU  0 <v:0xc0167828> <p:0x010167828> 1c33a0e1 mov r3, r12, LSL r3
inst: [        10] CPU  0 <v:0xc016782c> <p:0x01016782c> 011081e2 add r1, r1, #0x1
inst: [        11] CPU  0 <v:0xc0167830> <p:0x010167830> 541084e5 str r1, [r4, #84]
inst: [        12] CPU  0 <v:0xc0167834> <p:0x010167834> 033092e7 ldr r3, [r2, r3]
inst: [        13] CPU  0 <v:0xc0167838> <p:0x010167838> 4ff07ff5 dsb #15
inst: [        14] CPU  0 <v:0xc016783c> <p:0x01016783c> 010013e3 tst r3, #0x1
inst: [        15] CPU  0 <v:0xc0167840> <p:0x010167840> e5ffff1a bne 0xc01677dc
```

Fonte: Elaborada pelo autor.

Na Figura 22 os endereços virtuais e físicos são identificados por *v* e *p*, respectivamente. Para a arquitetura em questão estes endereços são iguais. Além disso, são observadas as instruções sendo executadas.

Figura 23. Fluxo de desenvolvimento do trabalho para a arquitetura ARM.



Após a geração dos *traces* de instruções das diferentes aplicações, foram feitas suas análises. Estas análises utilizaram rotina criada em linguagem C, sendo apresentada no Apêndice B deste trabalho.

Após análise dos *traces* de instruções gerados para as diferentes aplicações selecionadas, foi feita a classificação das instruções geradas na saída da rotina de análise dos *traces*. Nesta etapa as instruções foram classificadas em oito grupos: *load*, *store*, aritméticas, lógicas, desvios, *move*, *compare* e outras. Esta classificação foi feita a partir da análise do conjunto de instruções da arquitetura e teve como objetivo primeiramente separar as instruções que acessam a memória, tais como instruções do tipo *load* e *store*.

Além do *trace* de instruções, foram obtidos os *traces* de dados referentes às aplicações selecionadas. Para esta situação, os dados podem ser classificados como sendo de leitura (*read*) ou escrita (*write*). A Figura 24 apresenta um pequeno fragmento do *trace* de dados gerado a partir da execução da aplicação *fft* sobre a arquitetura ARM.

Figura 24 Trace de dados da aplicação *fft* em arquitetura ARM (fragmento).

```
data: [         1] CPU  0 <v:0xdf8c0088> <p:0x02f8c0088> Read   4 bytes  0x5b
data: [         2] CPU  0 <v:0xdf8c0088> <p:0x02f8c0088> Write  4 bytes  0x5c
data: [         3] CPU  0 <v:0xc036c35d> <p:0x01036c35d> Read   1 bytes  0x0
data: [         4] CPU  0 <v:0xc036c334> <p:0x01036c334> Read   4 bytes  0xf8010000
data: [         5] CPU  0 <v:0xc036c380> <p:0x01036c380> Read   4 bytes  0xb5
data: [         6] CPU  0 <v:0xc036c380> <p:0x01036c380> Write  4 bytes  0xb6
data: [         7] CPU  0 <v:0xf8010004> <p:0x0e0010004> Read   4 bytes  0x3
data: [         8] CPU  0 <v:0xdf8c0088> <p:0x02f8c0088> Read   4 bytes  0x5c
data: [         9] CPU  0 <v:0xdf8c0084> <p:0x02f8c0084> Read   4 bytes  0x5c
data: [        10] CPU  0 <v:0xc036c368> <p:0x01036c368> Read   4 bytes  0xdf8c0000
data: [        11] CPU  0 <v:0xdf8c0000> <p:0x02f8c0000> Read   4 bytes  0xdf8cb000
data: [        12] CPU  0 <v:0xc0311ec8> <p:0x010311ec8> Write  4 bytes  0xc036c32c
data: [        13] CPU  0 <v:0xc0311ecc> <p:0x010311ecc> Write  4 bytes  0xdf8c0000
data: [        14] CPU  0 <v:0xc0311ed0> <p:0x010311ed0> Write  4 bytes  0x0
data: [        15] CPU  0 <v:0xc0311ed4> <p:0x010311ed4> Write  4 bytes  0xc01678f4
```

Fonte: Elaborada pelo autor.

Após a geração dos *traces* de dados das diferentes aplicações, foi realizada a classificação dos dados em leitura (*read*) e escrita (*write*). A Figura 25 apresenta um fragmento da saída do *script* de verificação dos endereços acessados juntamente com o tipo de acesso para o *trace* de dados do *benchmark fft*. Este *script* é apresentado no Apêndice F deste trabalho.

Figura 25 Fragmento da saída do *script* de verificação dos endereços para o *trace* de dados do *benchmark fft*.

```
df8c0088 1308 Read
df8c0088 295 Write
c036c35d 1705 Read
c036c334 1705 Read
c036c380 295 Read
c036c380 295 Write
f8010004 714 Read
df8c0084 1013 Read
c036c368 285 Read
df8c0000 188 Read
c0311ec8 121 Write
c0311ecc 126 Write
c0311ed0 134 Write
c0311ed4 134 Write
df8cb09c 877 Read
```

Fonte: Elaborada pelo autor.

Além do exposto, foi realizada a separação dos endereços mais acessados para os *traces* de dados e de instruções de todos os *benchmarks* analisados. Ainda para o *trace* de dados, foram separados os endereços de leitura e os de escrita.

O mesmo processo, de geração e análise do traces de instruções e dados, realizado para a arquitetura ARM foi realizado para a arquitetura x86.

5 Resultados e discussão

Neste capítulo são apresentados os resultados obtidos a partir da execução das aplicações selecionadas em ambas as arquiteturas selecionadas (ARM e x86). O capítulo está entre três grandes seções: análise da arquitetura ARM, análise da arquitetura x86 e análise do potencial de otimização utilizado SPM.

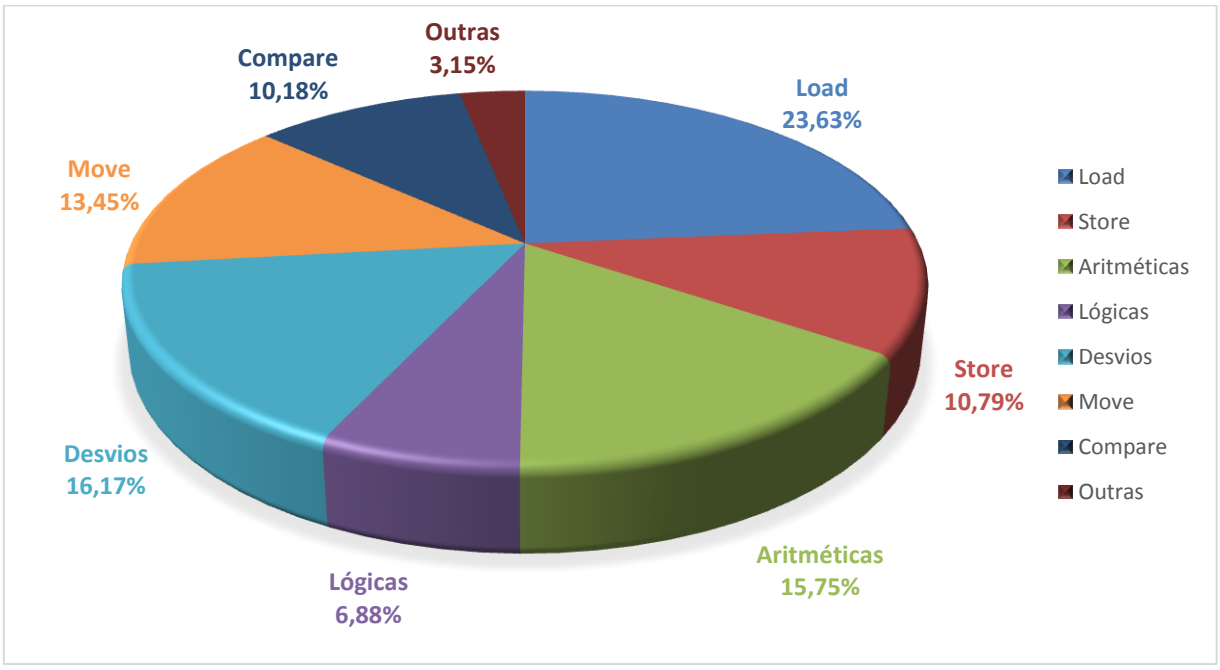
5.1 Análise da arquitetura ARM

Nesta seção são apresentados os resultados referentes ao acesso à memória de instruções e dados da arquitetura ARM.

5.1.1 Análise do *trace* de instruções

Após análise dos *traces* de instruções gerados para as diferentes aplicações selecionadas, foi feita a classificação das instruções. Nesta etapa as instruções foram classificadas em oito grupos: *load*, *store*, aritméticas, lógicas, desvios, *move*, *compare* e outras. A Figura 26 apresenta os percentuais dos grupos de instruções para a arquitetura ARM utilizada.

Figura 26. Percentual de grupos de instruções para ARM (MiBench).



Fonte: Elaborada pelo autor.

Percebe-se, a partir da Figura 26, que aproximadamente 35% das instruções (34,42%), ou seja, mais que 1/3 do total de instruções acessam a memória a partir das instruções *load* e *store* (valor médio para todas as aplicações selecionadas.

No Apêndice C deste trabalho são apresentados os resultados obtidos para cada aplicação selecionada. Destas aplicações, a aplicação *dijkstra* é a que apresenta o maior percentual de instruções do tipo *load* (37,16%) dentre as aplicações selecionadas. Por outro lado, a aplicação *basicmath* é a que apresenta o menor percentual de instruções do tipo *load* (15,07%).

Para as instruções do tipo *store*, verifica-se que a aplicação *mad* apresenta o maior percentual de ocorrências para este tipo de instrução (12,40%). Enquanto isso, a aplicação *basicmath* é a que apresenta o menor percentual de instruções do tipo *store* (7,89%).

Dentre as instruções do tipo *aritméticas*, a aplicação *sha* é a que apresenta o maior percentual de instruções executadas (23,65%). A aplicação *dijkstra*, no entanto, apresenta o menor percentual de instruções executadas para este tipo de instrução (10,98%). Um maior percentual de instruções lógicas foi obtido para a aplicação *basicmath* (13,85%). Para esta aplicação também foi obtido um maior percentual de instruções do tipo *move* (18,96%). A aplicação *dijkstra* foi a que apresentou menor percentual de instruções lógicas executadas (3,19%).

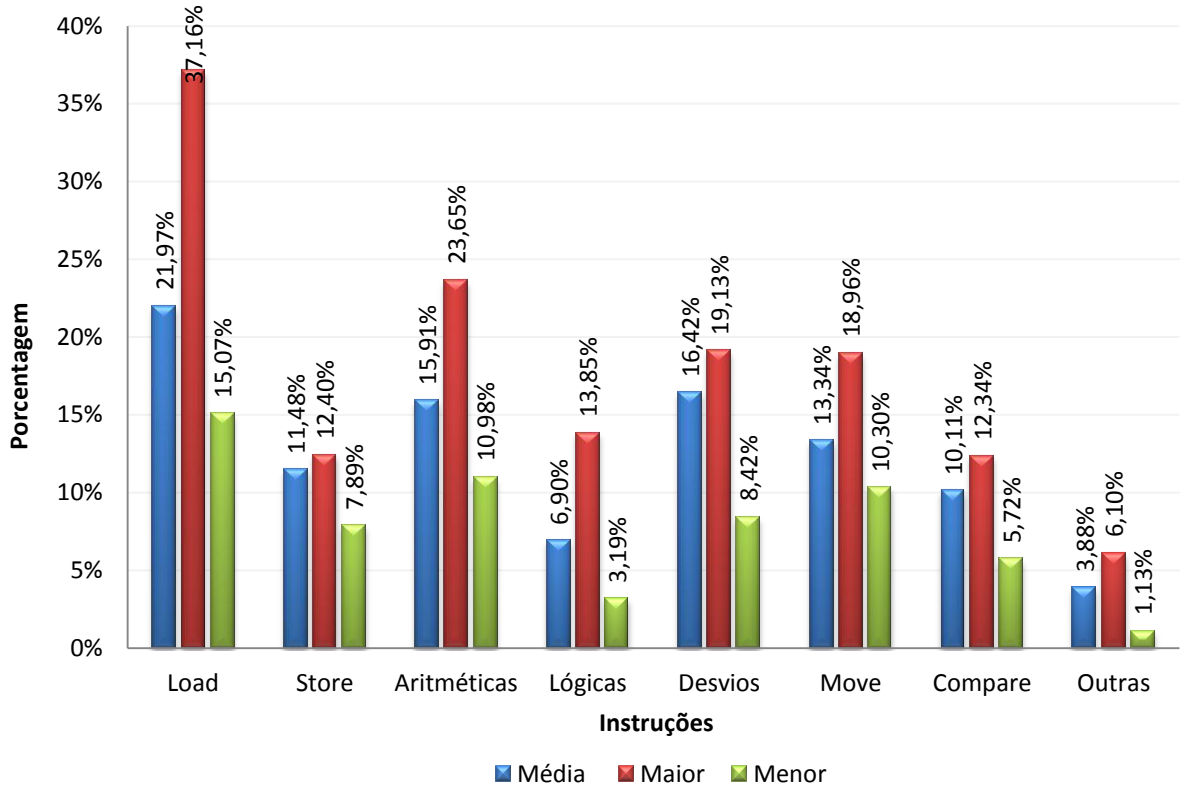
Para as instruções classificadas como desvios, a aplicação *qsort* foi a que apresentou maior percentual de instruções em seu *trace* de instruções (19,13%). O menor percentual de instruções de desvio foi apresentado pela aplicação *sha* (8,42%).

Dentre as instruções classificadas como sendo do tipo *move*, a aplicação *basicmath* foi a que apresentou o maior percentual (18,96%), enquanto a aplicação *sha* foi a que apresentou o menor percentual (10,30%).

Para as aplicações do tipo *compare*, a aplicação *qsort* foi a que apresentou o maior percentual (12,34%), enquanto a aplicação *sha* foi a que apresentou o menor percentual (5,72%). Por fim, a aplicação *mad* é a que apresenta um maior percentual de instruções classificadas como sendo do tipo *outras* (6,1%). Para este tipo de instruções a aplicação *dijkstra* é a que apresenta o menor percentual (1,13%).

A Figura 27 apresenta a porcentagem dos valores médio, maior e menor para cada grupo de instruções de todos os *benchmarks*, considerando a arquitetura ARM.

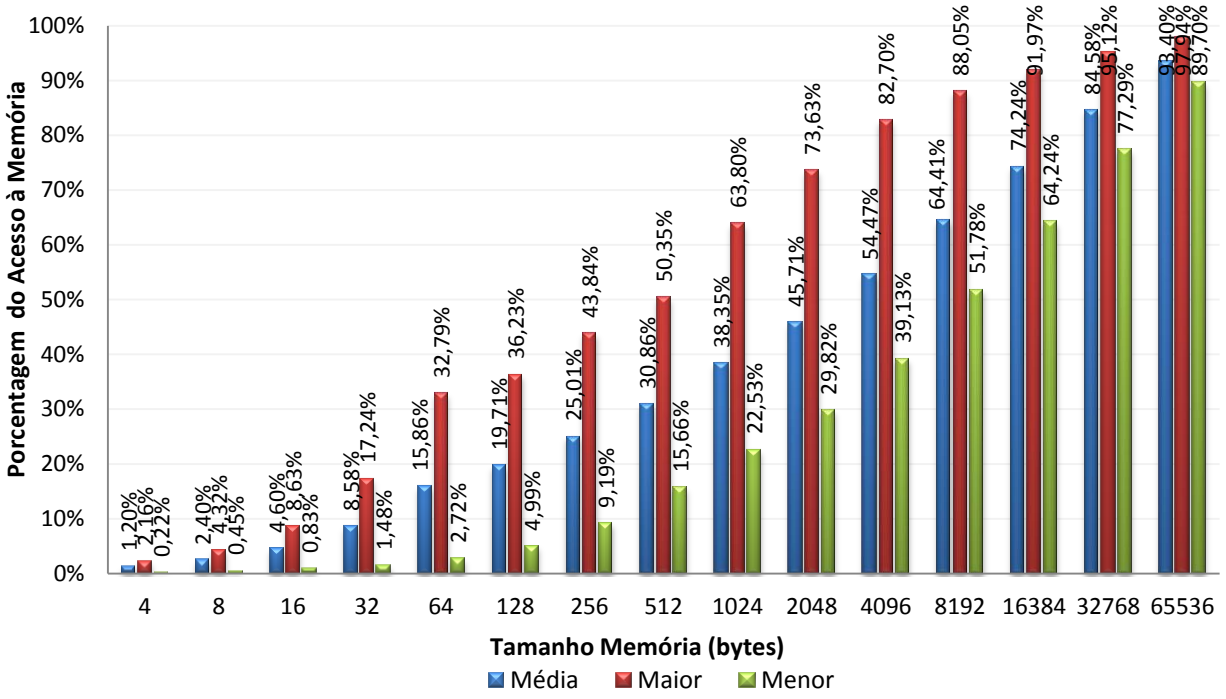
Figura 27 Porcentagem (valor médio, maior valor, menor valor) para cada grupo de instruções de todos os *benchmarks*.



Considerando o tamanho de cada endereço, foram realizados experimentos comparando memórias de 4 *bytes* a 64 KB, as quais possuem o maior número de acessos a endereços para o trace de instruções. Neste sentido, o objetivo foi analisar o tamanho dos trechos de código e o seu impacto no número de instruções executadas.

A Figura 28 apresenta a porcentagem média dos endereços mais acessados dos *traces* de instruções, juntamente com a maior e menor porcentagem de cada tamanho de memória. Todos os resultados para os *benchmarks* são mostrados na Tabela 4 apresentada no Apêndice deste trabalho. Este resultado foi obtido analisando os endereços mais acessados, ou seja, considerando o tamanho de memória de 4 *bytes* (o que significa uma instrução de 32 bits). Observa-se que, em média, de todos os *benchmarks* 1,20% do total de acesso à memória de instruções é realizado por um único endereço. Também é possível observar que, em média, um trecho de código de 64KB é responsável por 93,40% dos acessos à memória e, desta forma, possibilitando alternativas para exploração do espaço de projeto.

Figura 28. Porcentagem (valor médio, maior valor e menor valor) para diferentes tamanhos de memória para todos os *benchmarks*, para o *trace* de instruções.



As Figuras 29 e 30 demonstram como o tamanho de memória influencia na porcentagem total de endereços acessados (instruções buscadas). A Figura 30 é uma continuação da Figura 29. Para fins de ilustração, as figuras apresentam apenas a comparação entre memórias de 512 bytes e 8KB para cada um dos *benchmarks* escolhidos. Pode ser observado que, para 512 bytes, o percentual de acessos está entre 15,66% (*susan*) até 50,35% (*dijkstra*) e para 8 KB o percentual de acessos está entre 51,78% (*FFT*) até 88,05% (*dijkstra*).

Figura 29. Trace de instruções - comparação do tamanho de memória (512 bytes e 8192 bytes) com a porcentagem de endereços totais de cada *benchmark*.

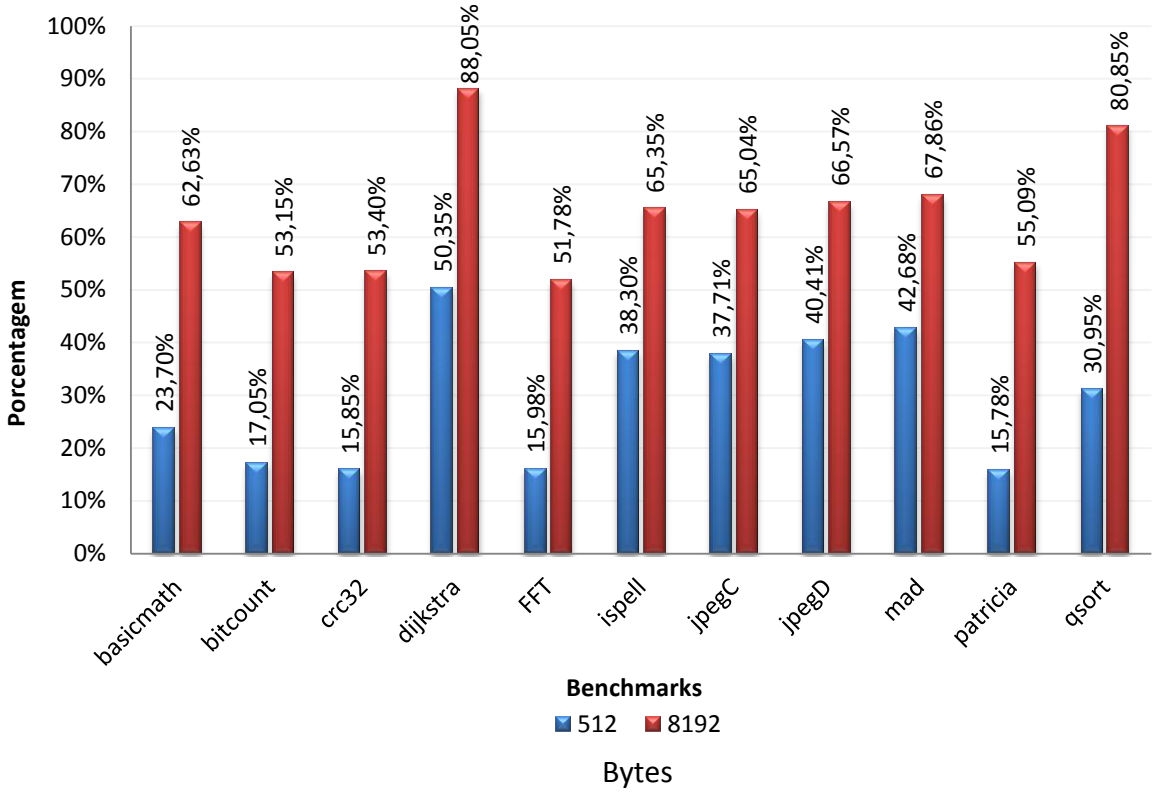
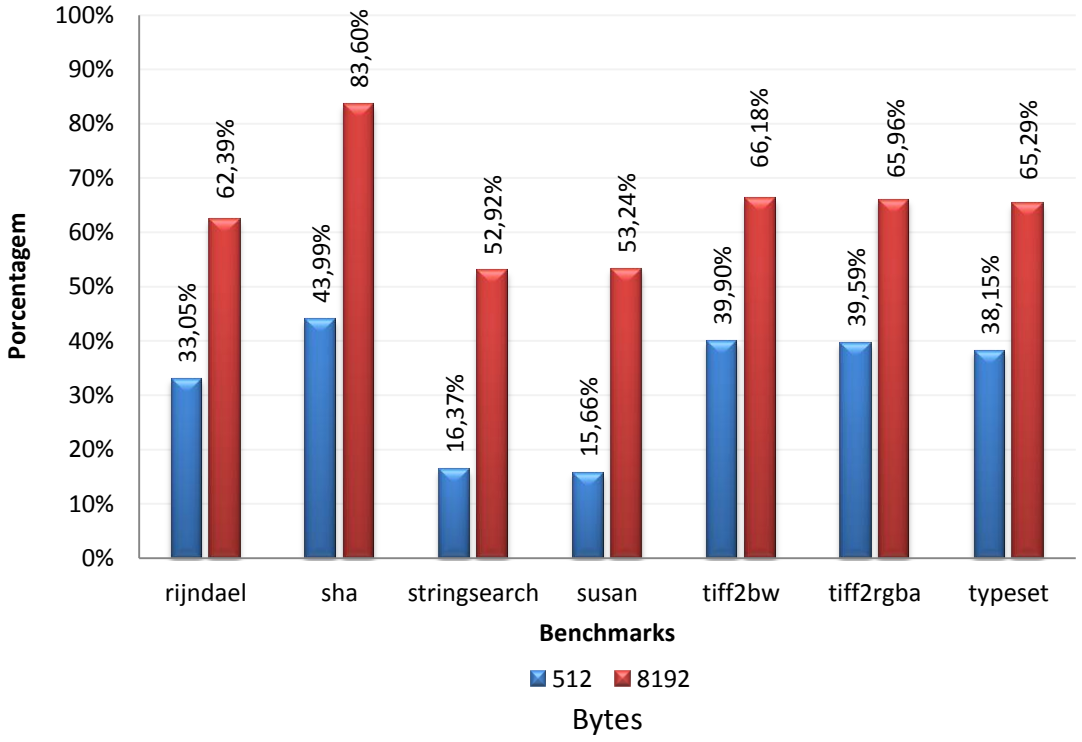


Figura 30. Trace de instruções - comparação do tamanho de memória (512 bytes e 8192 bytes) com a porcentagem de endereços totais de cada *benchmark* (cont.).



Os resultados destes experimentos tanto é apresentado por completo nas tabelas do apêndice C, D e E deste trabalho. Os resultados apontam um potencial de exploração do espaço de projeto tendo em vista que para um conjunto significativo de *benchmarks* um grande número de acessos ocorre devido ao pequeno percentual de instruções (ocupando um pequeno espaço em memória).

5.1.2 Análise do *trace* de dados

Para acesso a memória de dados, a mesma análise para a memória de instruções foi realizada. Considerando o tamanho de cada endereço, foram realizados experimentos comparando memórias de 4 bytes a 8 KB, as quais possuem o maior número de acessos a endereços para o trace de dados Também foram comparadas memórias de 8 bytes a 8 KB dos endereços mais acessados de leitura e escrita para o trace de dados.

As Figuras 31, 32 e 33 demonstram os resultados encontrados. A Figura 31 representa a porcentagem média dos endereços mais acessados dos *traces* de dados (tanto leituras e escritas), juntamente com a maior e menor porcentagem de cada tamanho de memória. Todos os resultados dos *benchmarks* são demonstrados nos apêndices D e E.

Pode-se observar que na média existe uma grande localidade temporal dos dados, pois 8KB dos dados são responsáveis por 70,66% dos acessos à memória. Da mesma forma que com os acessos à memória de instruções, os acessos à memória de dados também permitem uma grande exploração do espaço de projeto.

As Figuras 32 e 33 apresentam os resultados para os *benchmarks* apenas considerando 512 bytes e 8Kbytes. A Figura 33 é uma continuação da Figura 32. Pode ser observado que para 512 bytes o percentual de acessos está entre 17,82% (*susan*) até 73,98% (*sha*) e para 8KB o percentual de acessos está entre 58,34% (*crc32*) até 91,88% (*dijkstra*).

Figura 31. Porcentagem (valor médio, maior valor, menor valor) para tamanhos de memórias diferentes para todos os *benchmarks*, para o *trace* de dados.

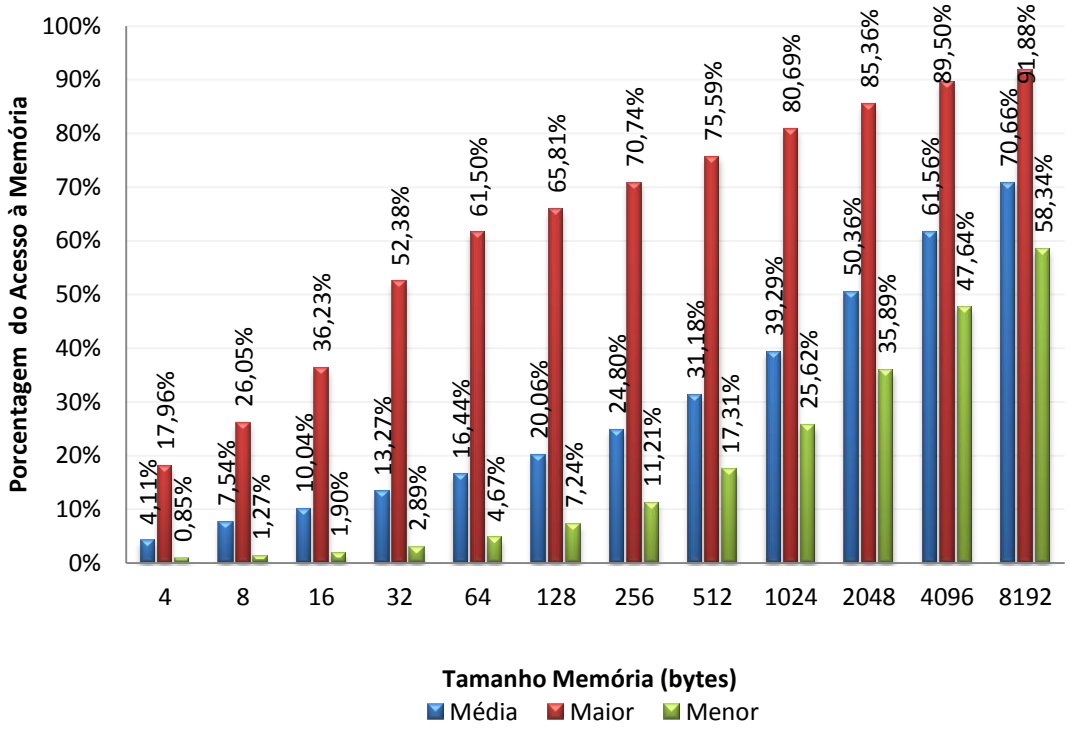


Figura 32. *Trace* de dados - comparação do tamanho de memória (512 bytes – 8192 bytes) com a porcentagem de endereços totais de cada *benchmark*.

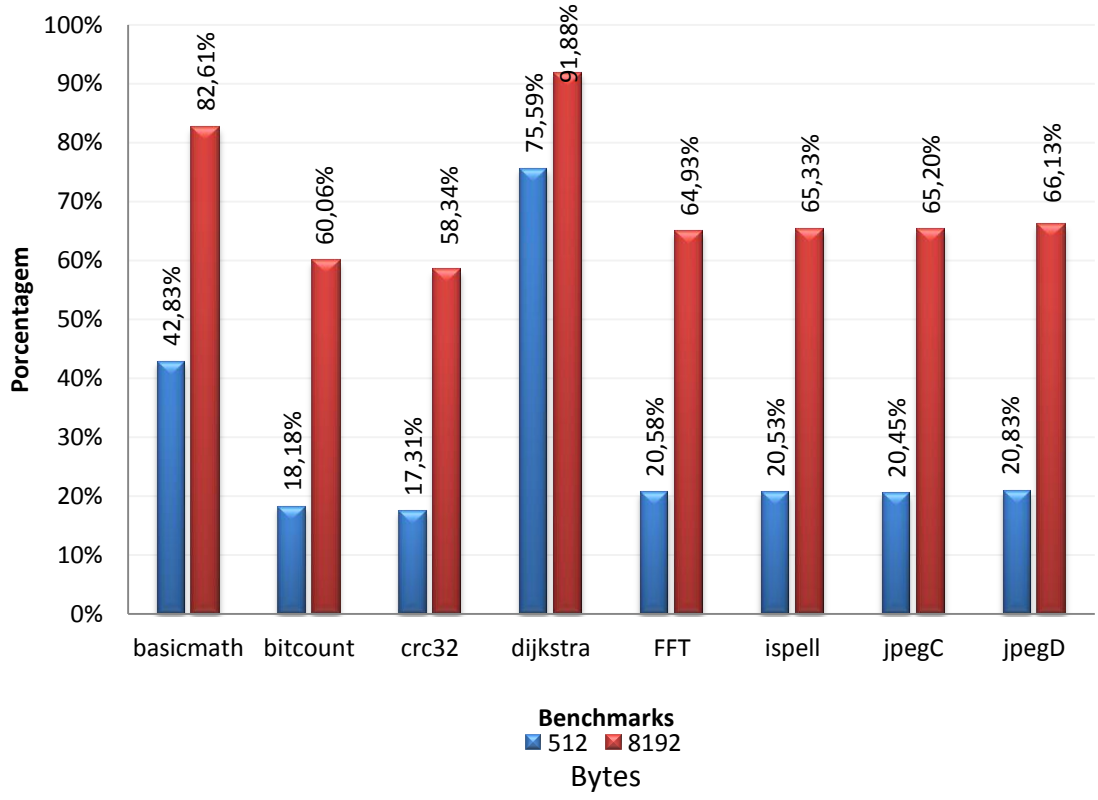
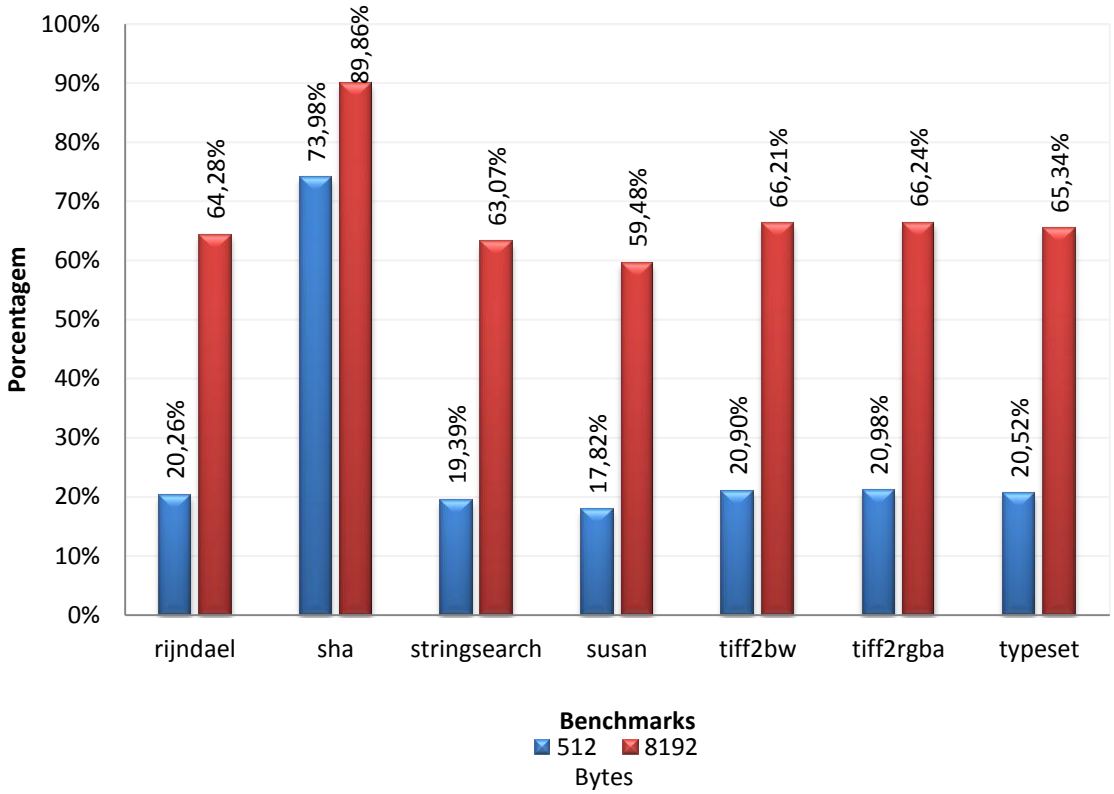
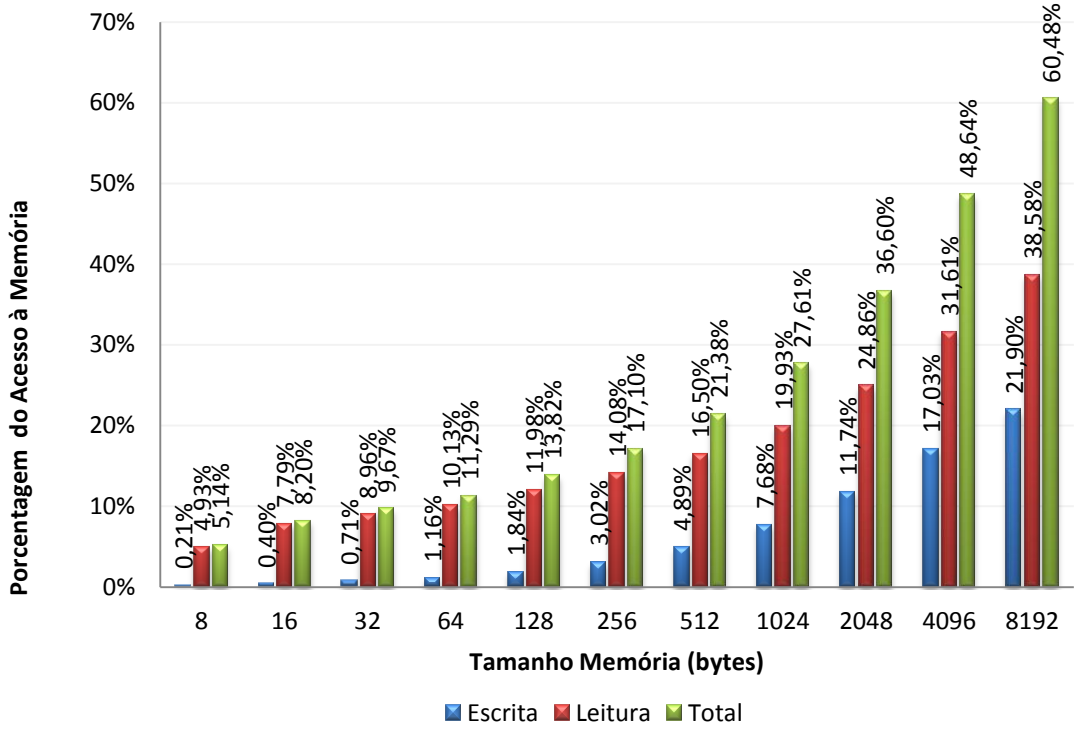


Figura 33. *Trace* de dados - comparação do tamanho de memória (512 bytes – 8192 bytes) com a porcentagem de endereços totais de cada *benchmark*.



A Figura 34 apresenta os acessos à memória de dados classificando em leitura e escrita em relação ao total de acessos e quantidade de dados. Os resultados complementares são encontrados na Tabela 10 localizada no Apêndice G. Observa-se que a leitura, como já esperado, é responsável pela maior parte dos acessos à memória de dados.

Figura 34. Porcentagem média dos endereços (leitura, escrita) mais acessados para o *trace* de dados dos *benchmarks*, dado o aumento do tamanho da memória.



5.2 Análise da arquitetura x86

Nesta seção são apresentados os resultados referentes ao acesso à memória de instruções e dados da arquitetura x86.

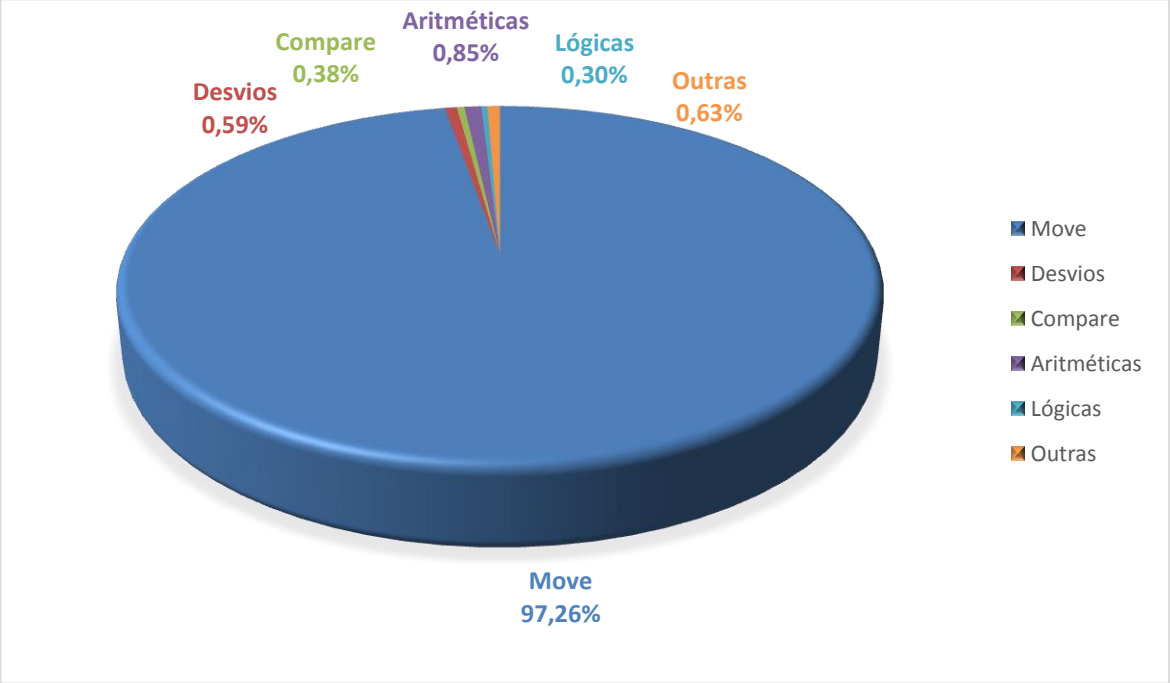
5.2.1 Análise do *trace* de instruções

Assim como para a arquitetura ARM, foi realizada a geração do *trace* de instruções para a arquitetura x86. Diferentemente da arquitetura ARM, para a arquitetura x86 foram selecionados somente nove *benchmarks* para análise: *basicmath*, *bitcount*, *crc32*, *dijkstra*, *fft*, *patricia*, *qsort*, *sha* e *susan*.

A Figura 35 apresenta os percentuais dos grupos de instruções para a arquitetura x86 utilizada. No Apêndice I deste trabalho são apresentados os resultados obtidos para cada aplicação selecionada. Destas aplicações, a aplicação basicmath é a que apresenta o maior percentual de instruções do tipo *move* (99,21%) dentre as aplicações selecionadas. A aplicação *bitcount*, no entanto, é a que apresenta o menor percentual deste tipo de instruções (32,33%). Com exceção desta aplicação, as demais

possuem seus traces de instruções concentrados em grande parte na execução de instruções do tipo *move*.

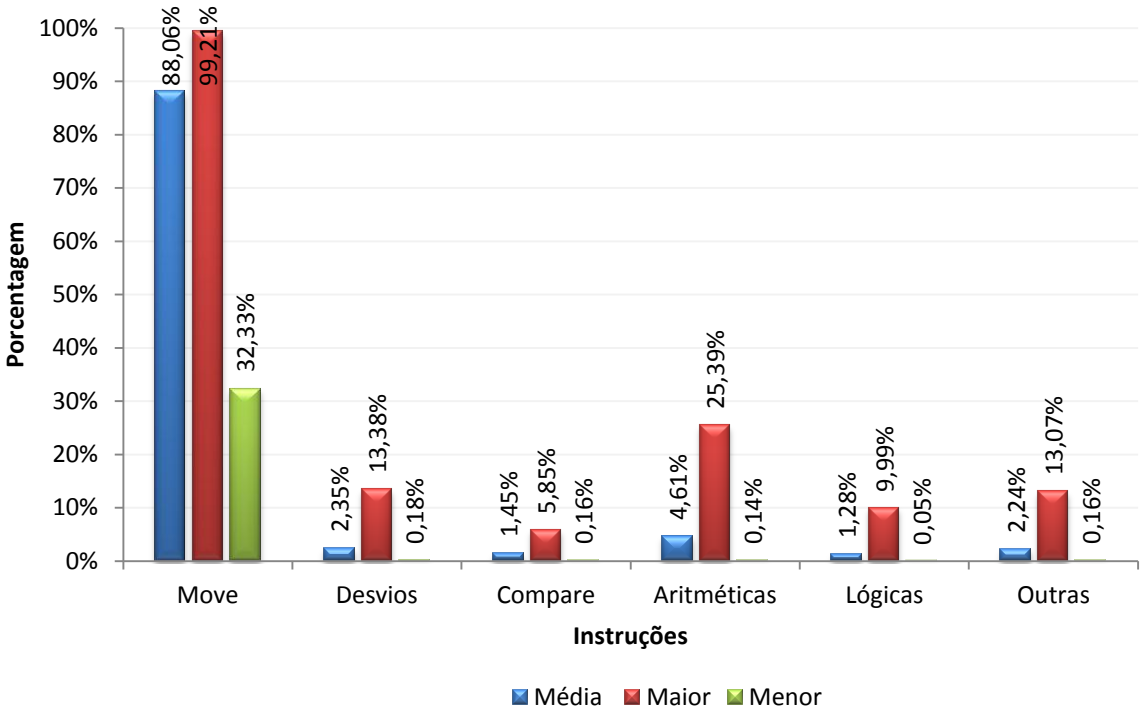
Figura 35 Percentual de grupos de instruções para x86 (MiBench).



Fonte: Elaborada pelo autor.

A Figura 36 apresenta os resultados para o trace de instruções referente à arquitetura x86. Ao contrário da arquitetura ARM, para a arquitetura x86 as instruções predominantes no *trace* são as instruções do tipo *move*. Estas representam, em média, 88,06% do total de instruções executadas para os *benchmarks* selecionados. Isto é uma característica relacionada com natureza CISC da arquitetura x86.

Figura 36. Porcentagem (valor médio, maior valor, menor valor) para cada grupo de instruções – x86.



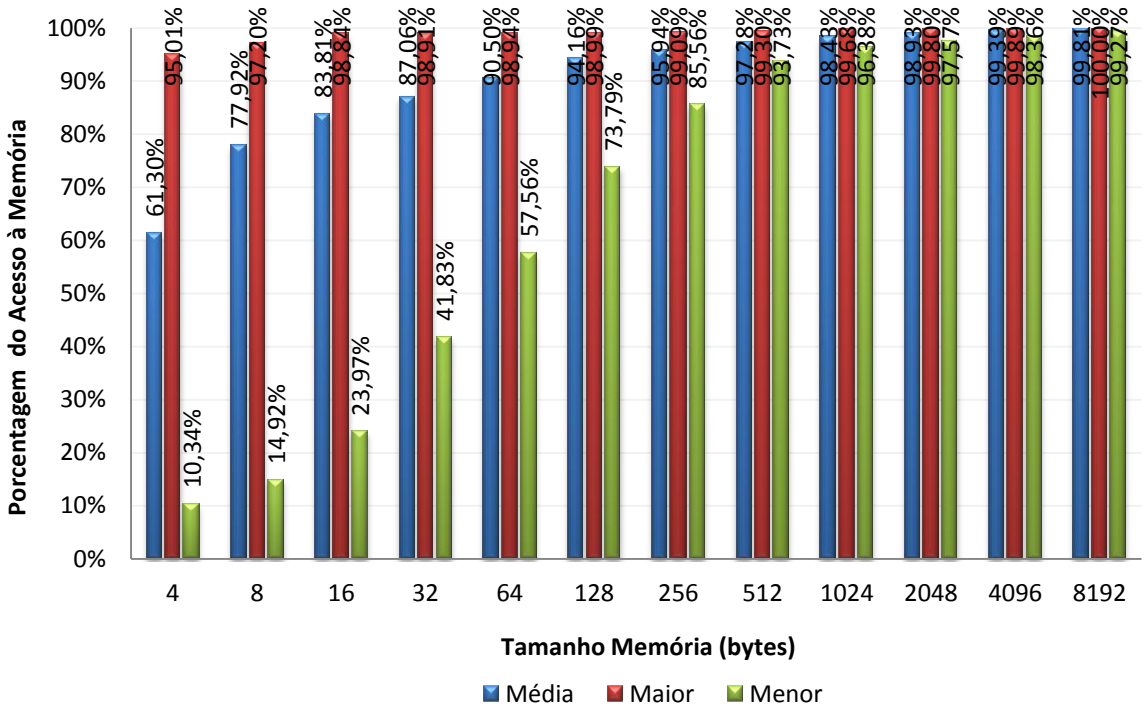
Como realizado na arquitetura ARM, foram realizados experimentos comparando memórias de 4 bytes a 8 KB as quais possuem o maior número de acessos a endereços para o trace de instruções. A Figura 37 apresenta a porcentagem média dos endereços mais acessados dos traces de instruções, juntamente com a maior e menor porcentagem de cada tamanho de memória.

Observa-se que, em média, um programa de 8 KB é responsável por 99,81% dos acessos à memória e, desta forma, melhorando as possibilidades para exploração do espaço de projeto quando comparado com este mesmo tamanho de memória para a arquitetura ARM. Isto deve-se ao fato da arquitetura x86 possuir um código mais compacto do que a arquitetura ARM.

A Figura 38 demonstra como o tamanho de memória influencia na porcentagem total de endereços acessados para a arquitetura x86. Assim como adotado para a arquitetura ARM, para fins de ilustração a figura apresenta apenas a comparação entre memórias de 512 bytes e 8KB para cada um dos *benchmarks* escolhidos.

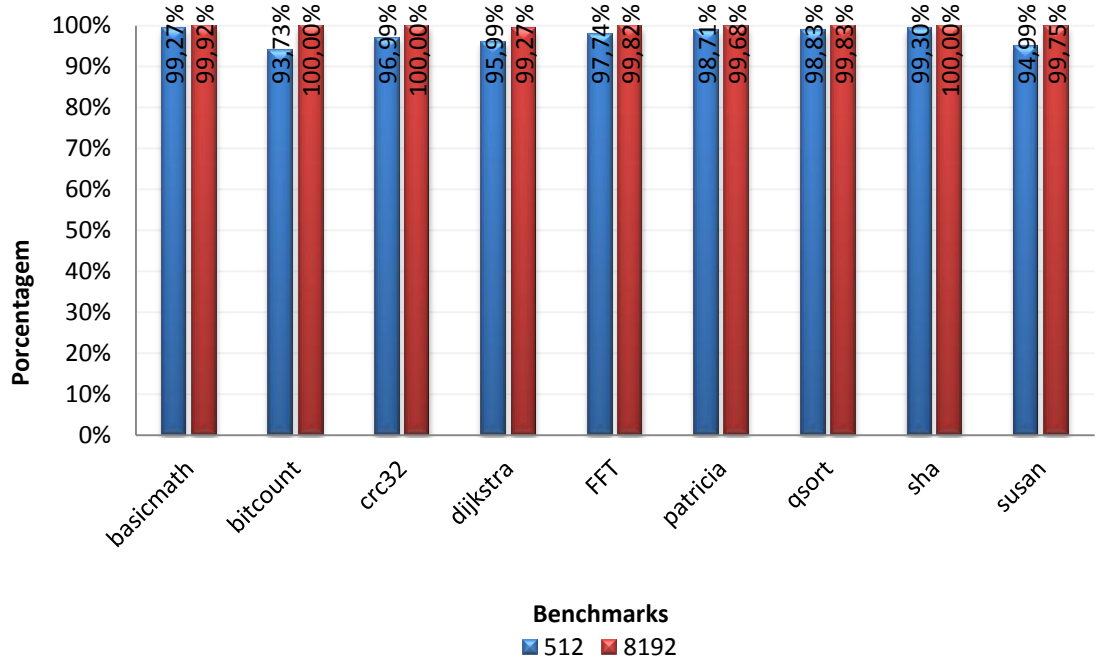
Pode ser observado que, para 512 bytes, o percentual de acessos está entre 93,73% (*bitcount*) até 99,30% (*sha*) e para 8 KB o percentual de acessos está entre 99,27% (*dijkstra*) até 100% (*bitcount, crc32* e *sha*).

Figura 37. Porcentagem (valor médio, maior valor e menor valor) para diferentes tamanhos de memória para todos os *benchmarks*, para o *trace* de instruções.



Fonte: Elaborada pelo autor.

Figura 38. Trace de instruções - comparação do tamanho de memória (512 bytes e 8192 bytes) com a porcentagem de endereços totais de cada *benchmark*.

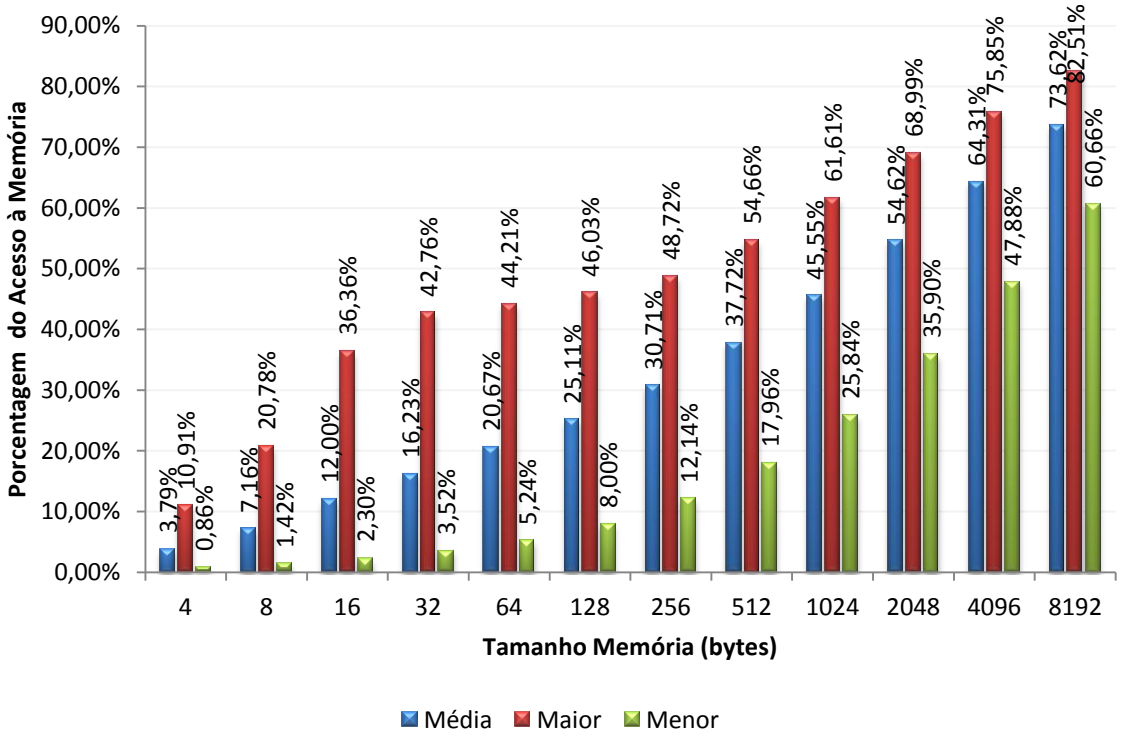


5.2.2 Análise do trace de dados

Como realizado para a arquitetura ARM foi realizado para a arquitetura x86 análise do comportamento referente a memória de dados. Foram realizados experimentos comparando memórias de 4 bytes a 8 KB, as quais possuem o maior número de acessos a endereços para o *trace* de dados. Também foram comparadas memórias de 8 bytes a 8 KB dos endereços mais acessados de leitura e escrita para o trace de dados.

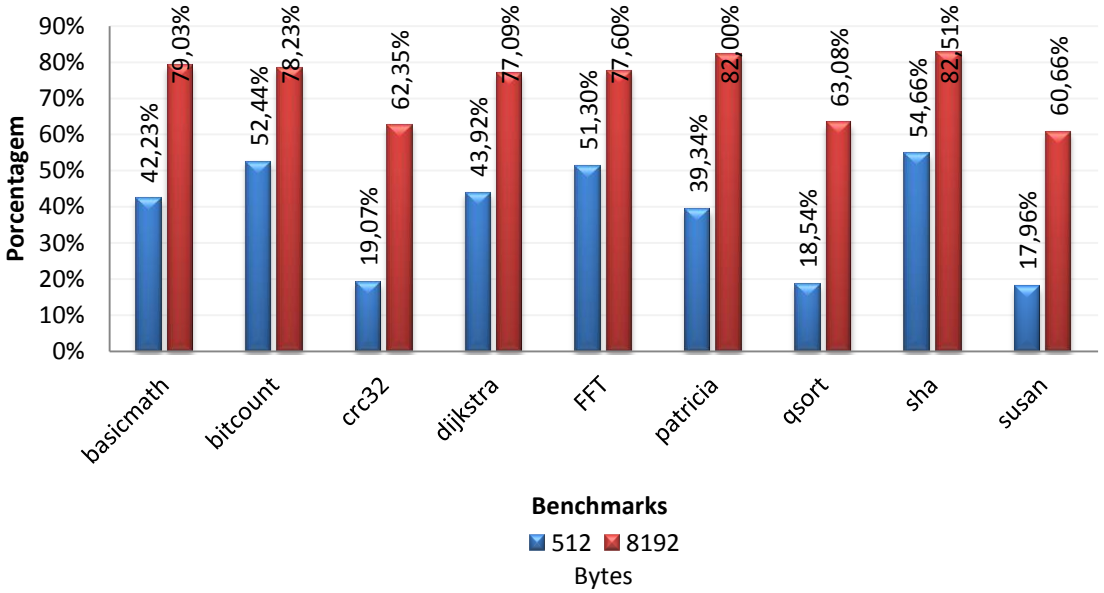
A Figura 39 representa a porcentagem média dos endereços mais acessados dos traces de dados, juntamente com a maior e menor porcentagem de cada tamanho de memória. Todos os resultados para os *benchmarks* selecionados são apresentados nos Apêndices J e K. Comparativamente a arquitetura ARM não é observada uma grande diferença quando é comparado o percentual médio.

Figura 39. Porcentagem (valor médio, maior valor, menor valor) para tamanhos de memórias diferentes para o trace de dados.



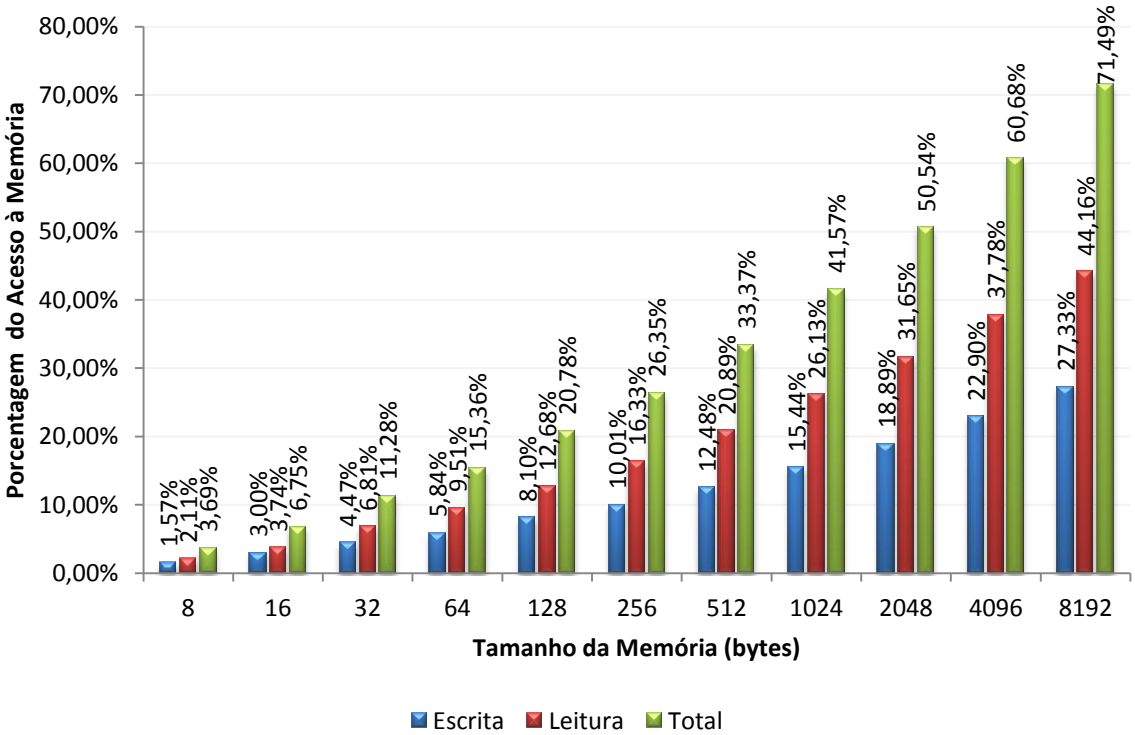
A Figura 40 apresenta os resultados para os *benchmarks* apenas considerando 512 bytes e 8Kbytes. Pode ser observado que para 512 bytes o percentual de acessos está entre 17,96% (*susan*) até 54,66% (*sha*) e para 8KB o percentual de acessos está entre 60,66% (*susan*) até 82,51% (*sha*).

Figura 40. *Trace* de dados - comparação do tamanho de memória (512 bytes – 8192 bytes) com a porcentagem de endereços totais de cada *benchmark*.



A Figura 41 apresenta os acessos à memória de dados classificando em leitura e escrita em relação ao total de acessos e quantidade de dados. Os resultados complementares são encontrados na Tabela 20 localizada no Apêndice J.

Figura 41. Porcentagem média dos endereços (leitura, escrita) mais acessados para o *trace* de dados, considerando o aumento do tamanho da memória–x86.



5.3 Análise do potencial de otimização utilizando memória *scratchpad*

Nesta seção é apresentada uma avaliação do potencial de otimização do acesso à memória utilizando uma memória *scratchpad* (SPM). Como citado anteriormente, uma SPM mantém os objetos de memória mapeados pelo compilador em uma parte do espaço de endereçamento, sendo o restante ocupado pela memória principal. As SPMs possuem um maior desempenho quando comparadas às memórias principais e um grande ganho energético comparado às memórias principais e *caches*. Esta análise se configura em uma análise de potencial visto que a técnica de memória *scratchpad,* como apresentado no capítulo 3, deve apresentar modificações arquiteturais e também de *software* (pelo compilador).

A análise deste potencial de otimização utiliza dados de desempenho e energia através da ferramenta CACTI. Para isto, foi realizada a comparação utilizando apenas uma memória principal e utilizando uma memória principal e uma memória *scratchpad.* Para obter os dados de latência da memória, e também consumo energético, foram modeladas no CACTI duas memórias: uma memória principal de 2MB e *scratchpads* de 128 bytes a 8KB.

A partir destes dados obtidos do CACTI e com base na quantidade de acessos a memória de dados para as diferentes aplicações foram geradas as comparações de dois tipos de memórias para a arquitetura ARM: uma formada somente pela memória principal e outra formada pela memória principal juntamente com uma memória *scratchpad*. Como resultado foram obtidas comparações energéticas e de desempenho entre os dois tipos de memórias analisados.

As Figuras 42 e 43 apresentam os resultados para consumo energético e tempo de acesso à memória respectivamente. São apresentados para cada tamanho de SPM o valor médio das aplicações e as aplicações com o maior e menor consumo ou tempo de acesso total. A aplicação com o menor consumo energético o *benchmark stringsearch* e com o maior foi o *patricia*. A redução no consumo energético, mesmo na aplicação de menor percentual (*stringsearch*) de acesso a memória de dados chega a 2x e na aplicação com o maior percentual chega a 3,85x. Em relação ao tempo de acesso total a memória, os resultados também são do menor tempo para o *benchmark stringsearch* e maior tempo para o *benchmark patrícia.*

A partir destes dados é observado o potencial de otimização e a grande possibilidade de exploração do espaço de projeto.

Figura 42. Consumo energético para o MiBench em diferentes modelos de memória SPM (arquitetura ARM).

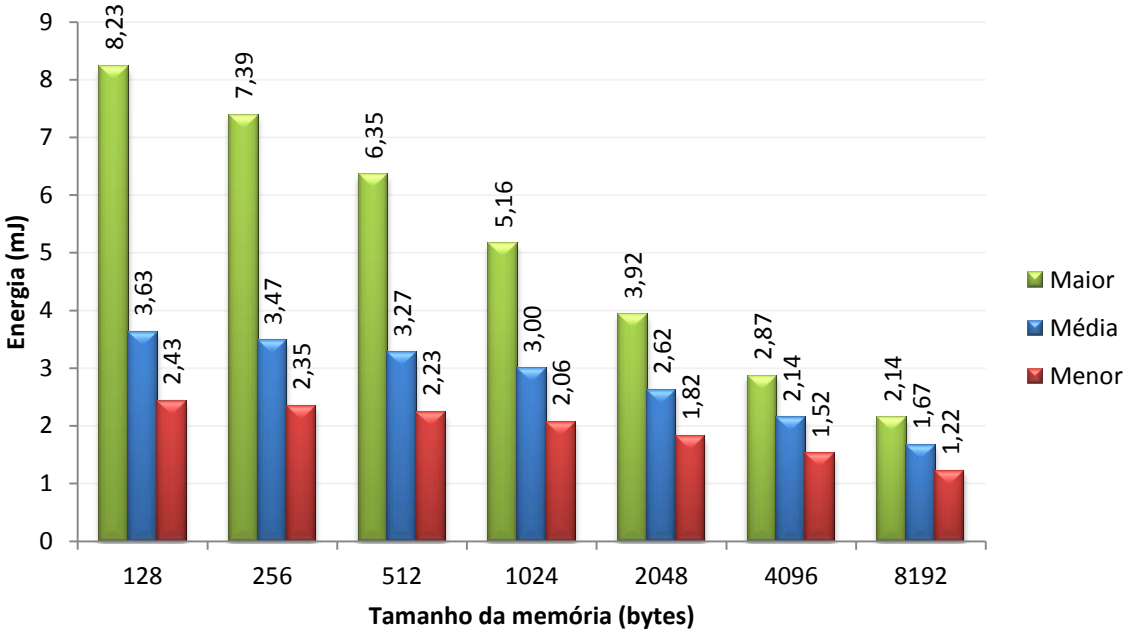
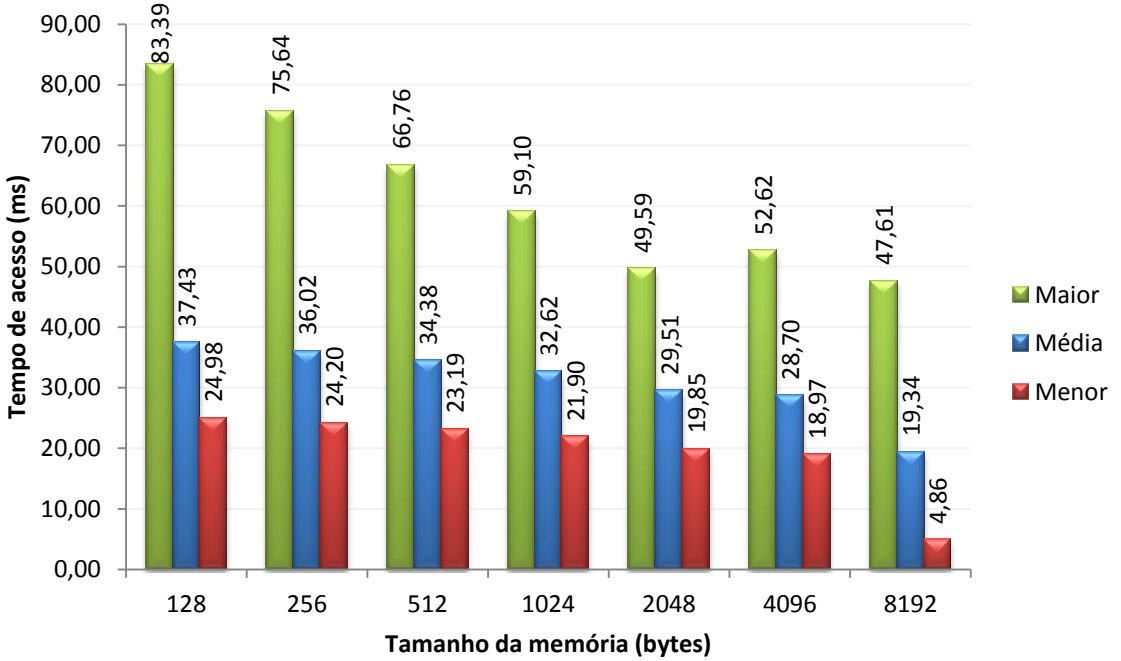


Figura 43. Tempo de acesso para o MiBench em diferentes modelos de memória SPM (arquitetura ARM).



6 Conclusões e trabalhos futuros

Este trabalho apresentou um estudo sobre o acesso a memória em arquiteturas de processadores embarcados. A dissertação apresentou uma revisão sobre sistemas embarcados e suas principais características dos sistemas embarcados. Também o trabalho revisa técnicas de otimização do sistema de memória para sistemas embarcados presentes na literatura.

O objetivo deste trabalho foi avaliar arquiteturas de processadores embarcados quanto ao acesso à memória para fins de identificar possibilidades de otimizações de código e propor soluções para redução do acesso à memória nestas arquiteturas. Para isso, foi avaliado o comportamento em relação ao acesso a memória de instruções e de dados das arquiteturas ARM e x86. Como metodologia foi utilizada a ferramenta Simics para a geração dos traces das diferentes arquiteturas e como *benchmark* é utilizado o pacote MiBench.

Relatou-se, então, os resultados obtidos a partir do trabalho proposto. Foram avaliados os *traces* de instruções e o *trace* de dados para cada uma das arquiteturas selecionadas. Além disso, foram analisados o acesso à memória para cada arquitetura. Ainda em relação aos resultados, foi apresentada uma avaliação do potencial de otimização do acesso à memória utilizando uma memória *scratchpad* (SPM). A análise deste potencial de otimização utilizou dados de desempenho e energia através da ferramenta CACTI.

Os resultados dos experimentos evidenciaram que aproximadamente 35% das instruções (34,42%), ou seja, mais que 1/3 do total de instruções acessam a memória a partir das instruções *load* e *store* para a arquitetura ARM. Destas aplicações, a aplicação *dijkstra* é a que apresenta o maior percentual de instruções do tipo *load* (37,16%) dentre as aplicações selecionadas.

Para as instruções do tipo *store*, verifica-se que a aplicação *mad* apresenta o maior percentual de ocorrências para este tipo de instrução (12,40%). Ainda para a arquitetura ARM, é possível observar que, em média, um programa de 8KB é responsável por 64,41% dos acessos à memória e, desta forma, possibilitando alternativas para exploração do espaço de projeto.

Da mesma forma que com os acessos à memória de instruções, os acessos à memória de dados também permitem uma grande exploração do espaço de projeto para a arquitetura ARM. Pode-se observar que na média existe uma grande localidade temporal dos dados, pois 8KB dos dados são responsáveis por 70,66% dos acessos à memória.

Ao contrário da arquitetura ARM, para a arquitetura x86 as instruções do tipo *move* são predominantes no *trace* de instruções. Estas representam, em média, 88,06% do total de instruções executadas para os *benchmarks* selecionados.

Os experimentos realizados para o *trace* de dados da arquitetura x86 apontam que, em média, um programa de 8 KB é responsável por 99,81% dos acessos à memória. Assim, são melhoradas as possibilidades para exploração do espaço de projeto para a arquitetura x86 quando comparada com este mesmo tamanho de memória para a arquitetura ARM.

Resultados experimentais apontaram grande potencial de otimização do acesso à memória e grande possibilidade de exploração do espaço de projeto para as arquiteturas com utilização de uma memória SPM.

Como sugestão de pesquisas futuras, poderão ser empregadas outras técnicas de otimização a fim de reduzir o acesso à memória e, assim, reduzir o consumo energético gerado pela busca de dados e instruções na memória. Sugere-se, ainda, a avaliação de outras arquiteturas de processadores embarcados e possíveis comparações com as arquiteturas abordadas neste trabalho.

Referências Bibliográficas

ARM LIMITED. **ARMv7-M Architecture Reference Manual**. ARM Limited. Cambridge. 2010.

BANAKAR, R. et al. Scratchpad Memory : A Design Alternative for Cache On-chip memory in Embedded Systems. **Proceedings of the Tenth International Symposium on Hardware/Software Codesign, 2002. CODES 2002**, Estes Park, Agosto 2002. 73-78.

BANK, W. Information and Communications for Development. **Maximizing Mobile**, Washington, 2012. ISSN 978-0-8213-8991-1/978-0-8213-9587-5.

BLEM, E.; MENON, J.; SANKARALINGAM, K. Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures. **IEEE International Symposium on High Performance Computer Architecture (HPCA2013)**, Shenzhen, 23-27 Fevereiro 2013. 1-12.

BUTKO, A. et al. Accuracy Evaluation of GEM5 Simulator System. **International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)**, York, July 2012. 1-7.

CARRO, L.; WAGNER, F. Capítulo 2 - Sistemas Computacionais Embarcados. **Jornadas de Atualização em Informática - XXII JAI**, Campinas, 2003. 45-94.

CATTHOOR, F. et al. **Custom Memory Management Methodology:** Exploration of Memory Organization for Embedded Multimedia System Design. Boston: Kluwer, 1998.

CATTHOOR, F. et al. **Ultra-low energy domain-specif instruction-set processors**. Berlim: Springer, 2010.

CISCO. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update 2014–2019 White Paper. **CISCO**, 03 Fevereiro 2015. Disponivel em: <http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html>. Acesso em: 08 Março 2015.

CLARK, J. The Register. **The Register**, 2013. Disponivel em: <http://www.theregister.co.uk/2013/04/23/amd_gseries_embedded_chips/>. Acesso em: 8 Janeiro 2014.

ECOS. **Ecos home page**, 2009. Disponivel em: <http://ecos.sourceware.org/>. Acesso em: Setembro 2009.

EDWARDS, S. et al. Design of Embedded Systems: Formal Models, Validation, and Synthesis. **Proceedings of the IEEE**, Março 1997. 366-390.

ENGBLOM, J.; EKBLOM, D. Simics: a commercially proven full-system simulation framework. **Workshop on Simulation in European Space Programmes (SESP 2006)**, Noordwijk, 6-8 Novembro 2006.

FREITAS, H. C. et al. **Ensino de Arquiteturas de Processadores Multi-Core Através de um Sistema de Simulação Completo e da Experiência de um Projeto de Pesquisa**. Workshop sobre Educação em Arquitetura de Computadores - WEAC. Campo Grande: [s.n.]. 2008.

GOMES, H. V. **Metodologia de Projeto de Software Embarcado Voltada ao Teste**. Porto Alegre: UFRGS, 2010.

GUTHAUS, M. R. et al. MiBench: A free, commercially representative embedded benchmark suite. **Proceedings of the IEEE International lWorkshop on Workload Characterization**, 2001. 3-14.

HENNESSY, J. L.; PATTERSON, D. A. **Computer Architecture – A quantative approach**. 5ª. ed. San Francisco: Morgan Kaufmann, 2011.

HUANG, Y.; LIU, T.; XUE, C. J. Register Allocation for Write Activity Minimization. **16th Asia and South Pacific Design Automation Conference (ASP-DAC)**, Yokohama, 2011. 129-134.

HUANG, Y.; LIU, T.; XUE, C. J. Register Allocation for Write Activity Minimization on Non-volatile Main Memory. **16th Asia and South Pacific Design Automation Conference (ASP-DAC)**, Yokohama, Março 2011. 129-134.

IMPERAS. Imperas Software Ltda. **Utilizing Virtual Platforms for Embedded Software Development**, 2014. Disponivel em: <http://www.imperas.com/utilizing-virtual-platforms-for-embedded-software-development>. Acesso em: 20 Outubro 2014.

IMPERAS SOFTWARE. Technology OVPsim. **Open Virtual Platforms - the source of Fast Processor Models & Platforms**, 2013. Disponivel em: <http://www.ovpworld.org/technology_ovpsim>. Acesso em: Abril 2014.

LI, L.; GAO, L.; XUE, J. Memory Coloring: A Compiler Approach for Scratchpad Memory Management. **Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques (PACT'05)**, Setembro 2005. 329-338.

MAGNUSSON, P. S. et al. Simics: a full system simulation platform. **IEEE Computer Society**, 35, n. 2, Feb 2002. 50 - 58.

MARVELL. Embedded Processors. **Site da Marvell Technology Group Ltd.**, Janeiro 2015. Disponivel em: <http://www.marvell.com/embedded-processors/>. Acesso em: 30 Janeiro 2015.

MARWEDEL, P. **Embedded Systems Design**. Dordrecht: Springer, 2006.

MARWEDEL, P. Optimizations - Compilation for Embedded Processors, Dortmund, 2013. Disponivel em: <http://cse.iitkgp.ac.in/~soumya/embcs/Marwedel-Compilation-for-Embedded-Processors-2.pdf>. Acesso em: Julho 2014.

MATTOS, J. C. B. **Design Space Exploration of SW and HW IP based on Object Oriented Methodology for Embedded System Applications**. Porto Alegre: UFRGS, 2007. (Tese de Doutorado).

MATTOS, J.; BRISOLARA, L. Desafios no Projeto de Sistemas Embarcados. In: MATTOS, J.; ROSA, L.; PILLA, M. **Desafios e Avanços em Computação - o estado da arte**. Pelotas: Editora e Gráfica Universitária, UFPel, 2009. Cap. 7, p. 153-175.

MICROSOFT Corporation. **Windows CE**, 2009. Disponivel em: <http://msdn.microsoft.com/pt-br/windowsembedded/ce/>. Acesso em: Setembro 2009.

MURALIMANOHAR, N.; BALASUBRAMONIAN, R.; JOUPPI, N. P. CACTI 6.0: A Tool to Model Large Caches. **HP Laboratories**, 2009. Disponivel em: <http://www.hpl.hp.com/techreports/2009/HPL-2009-85.pdf>. Acesso em: Julho 2014.

NEWSWIRE, PR. Global Embedded System Market is Expected to Reach USD 194.27 Billion in 2018: Transparency Market Research, 2013. Disponivel em: <http://www.prnewswire.com/news-releases/global-embedded-system-market-is-expected-to-reach-usd-19427-billion-in-2018-transparency-market-research-217360981.html>. Acesso em: Setembro 2014.

PANDA, P. R. et al. Data and memory optimization techniques for embedded systems, New York, Abril 2001. 149-206.

REDING, N. How Engineers Are Reinventing the Automobile. **National Instruments**, 2013. Disponivel em: <http://www.ni.com/newsletter/51684/en/>. Acesso em: 19 Fevereiro 2013.

SHAO, Z. et al. **Utilizing PCM for Energy Optimization in Embedded Systems**. IEEE Computer Society Annual Symposium on VLSI (ISVLSI). Amherst, MA: [s.n.]. 2012. p. 398-403.

SHEARER, F. **Power Management in Mobile Devices**. Massachusetts: Elsevier, 2008.

SLOSS, A. N.; SYMES, D.; WRIGHT, C. **ARM system developer's guide:** designing and optimizing system software. San Francisco: Elsevier, 2004.

SUNDARARAJAN, K. T.; JONES, T. M.; TOPHAM, N. **Smart Cache:** A Self Adaptive Cache Architecture. International Conference on Embedded Computer Systems (SAMOS). Samos: IEEE. 2011. p. 41-50.

THOMPSON, C. et al. **Optimizing Mobile Application Performance with Model-Driven Engineering**. SEUS '09 Proceedings of the 7th IFIP WG 10.2 International Workshop on Software Technologies for Embedded and Ubiquitous Systems. Heidelberg: Springer. 2009. p. 36-46.

THOZIYOOR, S. et al. **A Comprehensive Memory Modeling Tool and its Application to the Design and Analysis of Future Memory Hierarchies**. Beijing: [s.n.]. 2008. p. 51-62.

VERMA, M.; MARWEDEL, P. **Advanced memory optimization techniques for low-power embedded processors**. Berlim: Springer, 2007.

VIRTUTECH. Simics User Guide for Unix, 2007. Disponivel em: <http://www.cs.utah.edu/~manua/sim_doc/simics-user-guide-unix.pdf>. Acesso em: Fevereiro 2013.

WHITE, E. **Making Embedded Systems:** Design Patterns for Great Software. Sebastopol: O'Reilly Media, 2011.

WIND River. **Vxworks**, 2009. Disponivel em: <http://www.windriver.com/products/vxworks/>. Acesso em: Setembro 2009.

WIND RIVER. **Simics - Getting Started**. Wind River. Alameda. 2013.

WIND RIVER. Wind River Simics - Full System Simulation, 2014. Disponivel em: <http://www.windriver.com/products/simics/>. Acesso em: Setembro 2014.

WOLF, W. **Computers As Components:** Principles of Embedded Computing System Design. San Francisco: Morgan Kaufmann, 2005.

WOLF, W.; KANDEMIR, M. Memory System Optimization of Embedded Software. **Proceedings of the IEEE**, Los Alamitos, Janeiro 2003. 165-182.

XIE, Y. Modeling Architecture, and Applications for Emerging Memory Technologies. **IEEE Design & Test of Computers**, Fevereiro 2011. 44-51.

APÊNDICES

Apêndice A – Exemplo de *script* para geração de trace

```
new-tracer
#trace0.start
trace0->print_physical_address=1
trace0->trace_data=0
trace0->trace_exceptions=0

script-branch {
        board.console.con.wait-for-string "10.10.0.2"
        board.console.con.input "mount /host/ \n"
        board.console.con.input "cd /host/apps_binario_ARM/ \n"
        board.console.con.input "cd basicmath_arm \n"
        board.console.con.input "./exeBasicmath_small_ARM \n"
        board.console.con.input "\n exit"
        trace0.start
}

script-branch {
        board.console.con.wait-for-string "exit"
        trace0.stop
        stop
}

c
```

Apêndice B – Rotina de análise dos traces de instruções

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define mnemonicos 100000
#define registradores 100000

int main (int argc,char *argv[])
{
	FILE *in = fopen(argv[1],"r");
	FILE *outmne,*outreg;
	long long contador_mne[mnemonicos],contador_reg[registradores],i;
	char line[256],*token;
	char matriz_mne[mnemonicos][10],matriz_reg[registradores][10];
	for (i=0;i<mnemonicos;i++)
	{
		strcpy(matriz_mne[i],"");
		contador_mne[i]=0;
	}
	for (i=0;i<registradores;i++)
	{
		strcpy(matriz_reg[i],"");
		contador_reg[i]=0;
	}
	if (in)
	{
		while (fgets(line, sizeof(line), in))
		{
			token=strtok(line+30,">");
			if (token[0]<72)
			{
				for (i=0;i<registradores;i++)
				{
					if (!strcmp(matriz_reg[i],""))
					{
						strcpy(matriz_reg[i],token);
						contador_reg[i]++;
						break;
					}else
					{
						if (!strcmp(token,matriz_reg[i]))
						{
							contador_reg[i]++;
							break;
						}
					}
```

```
                        }
                }
                token=strtok(line+65," \n");
                for (i=0;i<mnemonicos;i++)
                {
                        if (!strcmp(matriz_mne[i],""))
                        {
                                strcpy(matriz_mne[i],token);
                                contador_mne[i]++;
                                break;
                        }else
                        {
                                if (!strcmp(token,matriz_mne[i]))
                                {
                                        contador_mne[i]++;
                                        break;
                                }
                        }
                }
        }
        outmne = fopen("saidamne","w");
        for (i=0;i<mnemonicos;i++)
        {
                if (!strcmp(matriz_mne[i],""))
                        break;
                fprintf(outmne,"%s %lld\n",matriz_mne[i],contador_mne[i]);
        }
        fclose(outmne);
        outreg = fopen("saidareg","w");
        for (i=0;i<registradores;i++)
        {
                if (!strcmp(matriz_reg[i],""))
                        break;
                fprintf(outreg,"%s %lld\n",matriz_reg[i],contador_reg[i]);
        }
        fclose(outreg);
    }
    return 0;
}
```

Apêndice C – Tabela referente ao *trace* de instruções (ARM).

Tabela 2 Classificação das instruções dos *benchmarks*.

Benchmark	Tipo de instrução								
	Load	%	Store	%	Aritméticas	%	Lógicas	%	
basicmath	1525933	15,07%	798646	7,89%	1859963	18,36%	1402604	13,85%	...
bitcount	874922	21,11%	498177	12,02%	610954	14,74%	319403	7,71%	
crc32	793619	21,10%	446761	11,88%	552020	14,68%	301326	8,01%	
dijkstra	8103103	37,16%	1998203	9,16%	2395071	10,98%	695853	3,19%	
fft	1043235	20,97%	613665	12,34%	789051	15,86%	305237	6,14%	
ispell	1073045	20,99%	628919	12,30%	823674	16,11%	297036	5,81%	
jpegC	1050441	20,99%	614851	12,29%	804334	16,08%	294486	5,89%	...
jpegD	1105861	20,99%	650296	12,34%	855747	16,24%	296005	5,62%	
mad	1167586	21,00%	689591	12,40%	910804	16,38%	299239	5,38%	
patricia	2720183	21,38%	1335513	10,50%	1820475	14,31%	910305	7,16%	
qsort	2788259	16,41%	1462188	8,61%	2644022	15,56%	1708855	10,06%	
rijndael	983621	21,03%	571507	12,22%	738131	15,78%	294758	6,30%	...
sha	3949401	31,80%	1527386	12,30%	2936790	23,65%	729618	5,87%	
stringsearch	641281	21,16%	353043	11,65%	444159	14,66%	242526	8,00%	
susan	779246	21,21%	434172	11,82%	529957	14,43%	293653	7,99%	
tiff2bw	1109161	21,01%	651118	12,33%	855502	16,20%	298096	5,65%	
tiff2rgba	1114706	21,02%	654150	12,34%	857858	16,18%	300053	5,66%	...
typeset	1068322	21,00%	626283	12,31%	819267	16,10%	296441	5,83%	
	Load	%	Store	%	Aritméticas	%	Lógicas	%	
Soma	31891925	23,63%	14554469	10,79%	21247779	15,75%	9285494	6,88%	
Média	1771773,611	21,97%	808581,6111	11,48%	1180432,17	15,91%	515860,7778	6,90%	...
Maior	8103103	37,16%	1998203	12,40%	2936790	23,65%	1708855	13,85%	
Menor	641281	15,07%	353043	7,89%	444159	10,98%	242526	3,19%	
IPC (*default*)	1								

Fonte: Autor

Tabela 3 Continuação da Tabela 2.

Benchmark	Tipo de instrução								
	Desvios	%	Move	%	Compare	%	Outras	%	Total
basicmath	1641649	16,21%	1919928	18,96%	805917	7,96%	173469	1,71%	10128109
bitcount	725401	17,50%	585197	14,12%	408431	9,86%	121840	2,94%	4144325
crc32	656069	17,44%	535001	14,22%	370557	9,85%	105773	2,81%	3761126
dijkstra	3486800	15,99%	2411687	11,06%	2471909	11,33%	246193	1,13%	21808819
fft	830393	16,69%	629461	12,66%	513450	10,32%	249492	5,02%	4973984
ispell	839105	16,41%	628088	12,29%	537455	10,51%	284605	5,57%	5111927
jpegC	822215	16,43%	617761	12,35%	524896	10,49%	274642	5,49%	5003626
jpegD	858675	16,30%	637513	12,10%	557501	10,58%	306523	5,82%	5268121
mad	899545	16,18%	661177	11,89%	592178	10,65%	339303	6,10%	5559423
patricia	2238838	17,60%	2049170	16,11%	1349086	10,60%	297937	2,34%	12721507
qsort	3249300	19,13%	2722586	16,03%	2096608	12,34%	317078	1,87%	16988896
rijndael	780891	16,69%	596528	12,75%	483022	10,32%	229833	4,91%	4678291
sha	1046006	8,42%	1279199	10,30%	710495	5,72%	240697	1,94%	12419592
stringsearch	538432	17,77%	431164	14,23%	303463	10,01%	76219	2,52%	3030287
susan	647609	17,63%	532937	14,51%	357944	9,74%	97962	2,67%	3673480
tiff2bw	862731	16,34%	641334	12,15%	558002	10,57%	303779	5,75%	5279723
tiff2rgba	867776	16,37%	645094	12,17%	559685	10,56%	302883	5,71%	5302205
typeset	835237	16,42%	625537	12,29%	534656	10,51%	282057	5,54%	5087800
	Desvios	%	Move	%	Compare	%	Outras	%	Total
Soma	21826672	16,17%	18149362	13,45%	13735255	10,18%	4250285	3,15%	134941241
Média	1212592,889	16,42%	1008297,889	13,34%	763069,7222	10,11%	236126,94	3,88%	7496735,611
Maior	3486800	19,13%	2722586	18,96%	2471909	12,34%	339303	6,10%	
Menor	538432	8,42%	431164	10,30%	303463	5,72%	76219	1,13%	

Fonte: Autor

Apêndice D – Tabela dos endereços mais acessados do *trace* de instruções (ARM).

Tabela 4 Endereços mais acessados do *trace* de instruções.

Benchmark	Tamanho (bytes)								
	4		8		16		32		
	N. Acessos	% Total	N. Acessos	% Total	N. Acessos	% Total	N. Acessos	% Total	
basicmath	24494	0,24%	48988	0,48%	95448	0,94%	183312	1,81%	
bitcount	36653	0,88%	73306	1,77%	118931	2,87%	154819	3,74%	
crc32	32634	0,87%	65268	1,74%	103757	2,76%	127177	3,38%	
dijkstra	199661	0,92%	399322	1,83%	798644	3,66%	1590828	7,29%	
FFT	36668	0,85%	73336	1,71%	118460	2,75%	152284	3,54%	...
ispell	93111	1,82%	186222	3,64%	372444	7,29%	743937	14,55%	
jpegC	88979	1,78%	177958	3,56%	355916	7,11%	710917	14,21%	
jpegD	104474	1,98%	208948	3,97%	417896	7,93%	834742	15,85%	
mad	119969	2,16%	239938	4,32%	479876	8,63%	958567	17,24%	...
patricia	28555	0,22%	57110	0,45%	106484	0,83%	189760	1,48%	
qsort	71651	0,42%	143302	0,84%	286604	1,69%	570165	3,36%	
rijndael	66253	1,42%	132506	2,83%	265012	5,66%	529307	11,31%	
sha	69676	0,56%	139352	1,12%	278704	2,24%	555828	4,48%	
stringsearch	28580	0,94%	57160	1,89%	90349	2,98%	108777	3,59%	...
susan	32634	0,89%	65268	1,78%	103591	2,82%	126335	3,44%	
tiff2bw	102408	1,94%	204816	3,88%	409632	7,76%	818232	15,50%	
tiff2rgba	101375	1,91%	202750	3,82%	405500	7,65%	809977	15,28%	
typeset	92078	1,81%	184156	3,62%	368312	7,24%	735682	14,46%	...
Média		1,20%		2,40%		4,60%		8,58%	
Maior		2,16%		4,32%		8,63%		17,24%	
Menor		0,22%		0,45%		0,83%		1,48%	
Desvio		0,65%		1,30%		2,73%		5,94%	...

Fonte: Autor

Tabela 5 Continuação da Tabela 4.

Benchmark	Tamanho (bytes)								
	64		128		256		512		
	N. Acessos	% Total	N. Acessos	% Total	N. Acessos	% Total	N. Acessos	% Total	
basicmath	358902	3,54%	701438	6,93%	1327104	13,10%	2400570	23,70%	
bitcount	221915	5,35%	323820	7,81%	475536	11,47%	706747	17,05%	
crc32	170527	4,53%	252083	6,70%	391261	10,40%	596060	15,85%	
dijkstra	2884380	13,23%	5458620	25,03%	9560350	43,84%	10979972	50,35%	
FFT	218348	5,08%	314949	7,32%	462045	10,74%	687446	15,98%	...
ispell	1421688	27,81%	1597183	31,24%	1743057	34,10%	1957795	38,30%	
jpegC	1360002	27,18%	1532428	30,63%	1676897	33,51%	1887094	37,71%	
jpegD	1591347	30,21%	1773193	33,66%	1917662	36,40%	2128709	40,41%	
mad	1822695	32,79%	2014180	36,23%	2159090	38,84%	2372747	42,68%	...
patricia	348134	2,72%	639018	4,99%	1177167	9,19%	2020221	15,78%	
qsort	1104410	6,50%	1932191	11,37%	3494987	20,57%	5258615	30,95%	
rijndael	1020684	21,82%	1180554	25,23%	1328374	28,39%	1546351	33,05%	
sha	1097955	8,84%	2131137	17,16%	4153529	33,44%	5463292	43,99%	
stringsearch	145633	4,81%	211549	6,98%	325392	10,74%	496030	16,37%	...
susan	171823	4,68%	248997	6,78%	378611	10,31%	575192	15,66%	
tiff2bw	1560492	29,56%	1742012	32,99%	1888586	35,77%	2106352	39,90%	
tiff2rgba	1545063	29,14%	1727019	32,57%	1876173	35,38%	2099098	39,59%	
typeset	1406262	27,64%	1580911	31,07%	1726294	33,93%	1940849	38,15%	...
Média		15,86%		19,71%		25,01%		30,86%	
Maior		32,79%		36,23%		43,84%		50,35%	
Menor		2,72%		4,99%		9,19%		15,66%	
Desvio		11,81%		12,12%		12,48%		12,02%	...

Fonte: Autor

Tabela 6 Continuação da Tabela 5.

Benchmark	Tamanho (bytes)									
	1024		2048		4096		8192			
	N. Acessos	% Total	N. Acessos	% Total	N. Acessos	% Total	N. Acessos	% Total	T.Endereços	T.Acessos
basicmath	3425259	33,82%	4540660	44,83%	5473185	54,04%	6343356	62,63%	97029	10128109
bitcount	989325	23,87%	1292419	31,19%	1684826	40,65%	2202567	53,15%	89584	4144325
crc32	865563	23,01%	1153725	30,68%	1528229	40,63%	2008431	53,40%	89392	3761124
dijkstra	12541068	57,50%	15101068	69,24%	18034948	82,70%	19203267	88,05%	92150	21808817
FFT	973634	22,64%	1282511	29,82%	1683126	39,13%	2227257	51,78%	92684	4301139
ispell	2232334	43,67%	2514524	49,19%	2872343	56,19%	3340572	65,35%	84417	5111925
jpegC	2157951	43,13%	2437393	48,71%	2792596	55,81%	3254440	65,04%	84449	5003624
jpegD	2400676	45,57%	2681059	50,89%	3039080	57,69%	3507195	66,57%	84508	5268119
mad	2649801	47,66%	2933019	52,76%	3296573	59,30%	3772679	67,86%	84505	5559421
patricia	2922216	22,83%	4099728	32,02%	5449614	42,57%	7052883	55,09%	94622	12802658
qsort	7537080	44,36%	9178365	54,03%	11812223	69,53%	13735523	80,85%	94632	16988894
rijndael	1820150	38,91%	2103261	44,96%	2459246	52,57%	2918674	62,39%	84299	4678289
sha	7923290	63,80%	9144629	73,63%	9732473	78,36%	10382276	83,60%	90616	12419592
stringsearch	711716	23,49%	943244	31,13%	1232655	40,68%	1603676	52,92%	76426	3030285
susan	827465	22,53%	1105435	30,09%	1481961	40,34%	1955673	53,24%	83174	3673478
tiff2bw	2380393	45,09%	2662175	50,42%	3021854	57,24%	3494073	66,18%	84294	5279721
tiff2rgba	2376115	44,81%	2659779	50,16%	3022528	57,01%	3497398	65,96%	84296	5302203
typeset	2215942	43,55%	2497823	49,09%	2854991	56,11%	3321929	65,29%	84465	5087798
Média		38,35%		45,71%		54,47%		64,41%		
Maior		63,80%		73,63%		82,70%		88,05%		
Menor		22,53%		29,82%		39,13%		51,78%		
Desvio		12,79%		13,03%		12,84%		10,79%		

Fonte: Autor

Apêndice E –Tabela dos endereços mais acessados do *trace* de dados (ARM).

Tabela 7 Endereços mais acessados do *trace* de dados.

Benchmark	Tamanho (bytes)								
	4		8		16		32		
	N. Acessos	% Total	N. Acessos	% Total	N. Acessos	% Total	N. Acessos	% Total	
basicmath	108830	1,82%	215128	3,60%	412797	6,91%	711615	11,90%	
bitcount	18974	0,91%	26460	1,27%	39930	1,91%	60411	2,89%	
crc32	18807	0,97%	26322	1,36%	39406	2,03%	58855	3,03%	
dijkstra	2324744	11,55%	4649487	23,09%	7254351	36,03%	10357087	51,44%	...
FFT	80917	3,22%	161834	6,44%	187060	7,44%	207334	8,25%	
ispell	91921	3,68%	183842	7,35%	208165	8,32%	227780	9,11%	
jpegC	91921	3,70%	183842	7,40%	208121	8,37%	227464	9,15%	
jpegD	107656	4,13%	215312	8,26%	239802	9,21%	259205	9,95%	...
mad	123391	4,50%	246782	8,99%	271752	9,90%	291431	10,62%	
patricia	151998	1,21%	277233	2,21%	499251	3,98%	840834	6,70%	
qsort	498329	3,67%	944087	6,95%	1642450	12,09%	2703162	19,89%	
rijndael	73039	3,08%	146078	6,16%	170727	7,19%	190324	8,02%	
sha	1939664	17,96%	2812924	26,05%	3912813	36,23%	5657292	52,38%	...
stringsearch	13416	0,88%	19716	1,29%	29002	1,90%	45104	2,96%	
susan	19351	0,85%	37617	1,65%	64112	2,82%	87257	3,84%	
tiff2bw	105558	4,05%	211116	8,09%	235568	9,03%	255481	9,79%	
tiff2rgba	104509	3,99%	209018	7,97%	233971	8,92%	254212	9,69%	...
typeset	95068	3,76%	190136	7,52%	214520	8,49%	234079	9,26%	
Média		4,11%		7,54%		10,04%		13,27%	
Maior		17,96%		26,05%		36,23%		52,38%	
Menor		0,85%		1,27%		1,90%		2,89%	...
Desvio		4,23%		6,80%		9,96%		14,61%	

Fonte: Autor

Tabela 8 Continuação da Tabela 7.

Benchmark	Tamanho (bytes)								
	64		128		256		512		
	N. Acessos	% Total	N. Acessos	% Total	N. Acessos	% Total	N. Acessos	% Total	
basicmath	1047826	17,53%	1516706	25,37%	2011241	33,64%	2560230	42,83%	
bitcount	97584	4,67%	158969	7,61%	244588	11,71%	379608	18,18%	
crc32	90989	4,69%	140399	7,24%	217548	11,21%	335761	17,31%	
dijkstra	12384681	61,50%	13250848	65,81%	14244871	70,74%	15221607	75,59%	...
FFT	242755	9,66%	301617	12,00%	385468	15,34%	517261	20,58%	
inspell	260965	10,43%	312867	12,51%	391424	15,65%	513378	20,53%	
jpegC	260099	10,47%	310524	12,49%	388296	15,62%	508272	20,45%	
jpegD	291840	11,20%	342265	13,14%	420175	16,13%	542642	20,83%	...
mad	324311	11,82%	375425	13,68%	455569	16,60%	583028	21,24%	
patricia	1431019	11,40%	2264092	18,03%	3568105	28,42%	5300253	42,21%	
qsort	3946555	29,04%	5654159	41,61%	7500613	55,20%	9198039	67,69%	
rijndael	224094	9,44%	278113	11,72%	357806	15,08%	480864	20,26%	
sha	6592091	61,04%	7008588	64,89%	7411757	68,63%	7990127	73,98%	...
stringsearch	74479	4,88%	120870	7,93%	190408	12,49%	295743	19,39%	
susan	125172	5,51%	186196	8,19%	271188	11,93%	405103	17,82%	
tiff2bw	288984	11,08%	341946	13,11%	421073	16,14%	545454	20,90%	
tiff2rgba	288414	11,00%	343602	13,10%	423847	16,16%	550134	20,98%	...
typeset	267091	10,57%	318530	12,61%	396861	15,71%	518381	20,52%	
Média		16,44%		20,06%		24,80%		31,18%	
Maior		61,50%		65,81%		70,74%		75,59%	
Menor		4,67%		7,24%		11,21%		17,31%	...
Desvio		17,22%		18,31%		19,47%		20,38%	

Fonte: Autor

Tabela 9 Continuação da Tabela 8.

Benchmark	Tamanho (bytes)									
	1024		2048		4096		8192			
	N. Acessos	% Total	N. Acessos	% Total	N. Acessos	% Total	N. Acessos	% Total	T.Endereços	T.Acessos
basicmath	3207620	53,66%	3795463	63,49%	4420176	73,94%	4938061	82,61%	236335	5977887
bitcount	563785	27,00%	779690	37,34%	1026179	49,15%	1254004	60,06%	220699	2087969
crc32	497126	25,62%	696372	35,89%	924245	47,64%	1131918	58,34%	222951	1940157
dijkstra	16247397	80,69%	17189051	85,36%	18022734	89,50%	18500145	91,88%	255295	20136170
FFT	695911	27,69%	1001627	39,85%	1351383	53,76%	1631996	64,93%	278470	2513600
inspell	696670	27,85%	1032192	41,27%	1368587	54,72%	1633946	65,33%	278661	2501060
jpegC	690350	27,78%	1024462	41,22%	1356815	54,59%	1620531	65,20%	278103	2485381
jpegD	750052	28,79%	1122009	43,07%	1457166	55,93%	1722880	66,13%	278312	2605110
mad	820949	29,91%	1223481	44,58%	1563271	56,96%	1833142	66,80%	280169	2744397
patricia	7228319	57,57%	8898497	70,87%	10231981	81,49%	11133439	88,67%	293103	12555632
qsort	10180206	74,92%	10684542	78,63%	11354684	83,56%	11892655	87,52%	392181	13588194
rijndael	646395	27,24%	924792	38,97%	1260193	53,11%	1525298	64,28%	278893	2372935
sha	8428094	78,04%	8836927	81,82%	9357602	86,64%	9704985	89,86%	299864	10799954
stringsearch	442033	28,99%	624111	40,93%	805915	52,85%	961731	63,07%	177184	1524858
susan	591817	26,03%	817256	35,95%	1083969	47,68%	1352099	59,48%	240150	2273194
tiff2bw	749759	28,74%	1120024	42,93%	1458888	55,91%	1727431	66,21%	278293	2609215
tiff2rgba	753287	28,73%	1123829	42,86%	1466010	55,91%	1736764	66,24%	278719	2622118
typeset	705473	27,92%	1048810	41,51%	1384485	54,79%	1650878	65,34%	278796	2526777
Média		39,29%		50,36%		61,56%		70,66%		
Maior		80,69%		85,36%		89,50%		91,88%		
Menor		25,62%		35,89%		47,64%		58,34%		
Desvio		19,91%		17,09%		14,25%		11,51%		

Fonte: Autor

Apêndice F – Rotina de análise dos traces de dados

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define aux 250000
int main ( int argc, char *argv[])
{
	FILE *in = fopen(argv[1],"r");
	  FILE *outmne,*outreg;
	  long int contador_reg[aux],i;
	  char line[256],*token,*token2;
	  char matriz_reg[aux][10],matriz_reg_data[aux][10];



for (i=0;i<aux;i++)
		{
			strcpy(matriz_reg[i],"");
			strcpy(matriz_reg_data[i],"");
			contador_reg[i]=0;
		}
		if (in)
		{
			while (fgets(line, sizeof(line), in))
			{

				token=strtok(line+30,">");
			token2=strtok(line+56," \n");
				for (i=0;i<aux;i++)
				{


if (!strcmp(matriz_reg[i],""))
					{
						strcpy(matriz_reg[i],token);
						contador_reg[i]++;
goto salva;
						break;
					}
else
					{
						if
((!strcmp(token,matriz_reg[i]))&&(!strcmp(token2,matriz_reg_data[i])))
						{
							contador_reg[i]++;
							break;
						}
else
```

```
                                {
if (!strcmp(matriz_reg[i],""))
{
strcpy(matriz_reg[i],token);
contador_reg[i]++;
goto salva;
break;
}
}

}

salva:
if (!strcmp(matriz_reg_data[i],""))
{
strcpy(matriz_reg_data[i],token2);
break;
                                }
                                }
                        }
                        outreg = fopen("saidareg_data","w");
                        for (i=0;i<aux;i++)
                        {
                                if (!strcmp(matriz_reg[i],""))
                                        break;
                                fprintf(outreg,"%s %ld %s
\n",matriz_reg[i],contador_reg[i],matriz_reg_data[i]);
                        }
                        fclose(outreg);
                }
                return 0;
}
```

Apêndice G –Tabela dos endereços mais acessados (leitura e escrita) do *trace* de dados (ARM).

Tabela 10 Endereços mais acessados do *trace* de dados (separação em leitura e escrita).

Benchmark		Tamanho (bytes)						
		8		16		32		
	Tipo	N.Acessos	% Total	N.Acessos	% Total	N.Acessos	% Total	
basicmath	Escrita	17705	0,57%	35391	1,15%	70369	2,28%	...
	Leitura	42046	1,36%	84092	2,72%	147960	4,79%	
	Total	59751	1,93%	119483	3,87%	218329	7,07%	
bitcount	Escrita	3789	0,19%	6087	0,31%	10444	0,52%	
	Leitura	18974	0,95%	25709	1,29%	36027	1,81%	
	Total	22763	1,14%	31796	1,59%	46471	2,33%	
crc32	Escrita	2824	0,15%	4800	0,26%	8610	0,47%	...
	Leitura	18807	1,02%	26322	1,43%	37656	2,05%	
	Total	21631	1,17%	31122	1,69%	46266	2,51%	
dijkstra	Escrita	4829	0,14%	7762	0,23%	12351	0,37%	
	Leitura	1160229	34,57%	1187139	35,38%	1222171	36,42%	
	Total	1165058	34,72%	1194901	35,61%	1234522	36,79%	...
FFT	Escrita	3471	0,16%	6753	0,30%	11082	0,50%	
	Leitura	80917	3,63%	158552	7,11%	182845	8,20%	
	Total	84388	3,78%	165305	7,41%	193927	8,70%	
ispell	Escrita	3057	0,14%	5999	0,27%	9968	0,45%	
	Leitura	91921	4,19%	180900	8,25%	205223	9,36%	...
	Total	94978	4,33%	186899	8,53%	215191	9,82%	
jpegC	Escrita	2942	0,14%	5864	0,27%	9800	0,45%	
	Leitura	91921	4,22%	180900	8,31%	205179	9,42%	
	Total	94863	4,36%	186764	8,58%	214979	9,87%	
jpegD	Escrita	3182	0,14%	6104	0,27%	10040	0,44%	...
	Leitura	107656	4,75%	212130	9,37%	236620	10,45%	
	Total	110838	4,90%	218234	9,64%	246660	10,89%	
mad	Escrita	3422	0,14%	6414	0,27%	10363	0,44%	
	Leitura	123391	5,20%	243360	10,25%	268330	11,30%	
	Total	126813	5,34%	249774	10,52%	278693	11,73%	...
patricia	Escrita	44595	0,74%	89171	1,48%	176751	2,93%	
	Leitura	111009	1,84%	222018	3,68%	392286	6,51%	
	Total	155604	2,58%	311189	5,16%	569037	9,44%	
qsort	Escrita	5183	0,18%	10025	0,35%	14702	0,52%	
	Leitura	109611	3,84%	214380	7,52%	255160	8,95%	
	Total	114794	4,03%	224405	7,87%	269862	9,46%	
rijndael	Escrita	3265	0,16%	5919	0,28%	9936	0,47%	
	Leitura	73039	3,47%	143424	6,81%	167440	7,95%	...
	Total	76304	3,62%	149343	7,09%	177376	8,43%	
		⋮	⋮	⋮	⋮	⋮	⋮	

Tabela 11 Continuação da Tabela 10.

Benchmark		Tamanho (bytes)						
		8		16		32		
	Tipo	N.Acessos	% Total	N.Acessos	% Total	N.Acessos	% Total	
sha	Escrita	4140	0,18%	7827	0,34%	11854	0,51%	...
	Leitura	103091	4,42%	202495	8,68%	226427	9,70%	
	Total	107231	4,59%	210322	9,01%	238281	10,21%	
stringsearch	Escrita	3183	0,21%	5109	0,34%	8661	0,57%	
	Leitura	13416	0,88%	18021	1,18%	26213	1,72%	
	Total	16599	1,09%	23130	1,52%	34874	2,29%	
susan	Escrita	4154	0,20%	6360	0,31%	10436	0,51%	...
	Leitura	19351	0,94%	37617	1,83%	61008	2,97%	
	Total	23505	1,14%	43977	2,14%	71444	3,48%	
tiff2bw	Escrita	3158	0,14%	6308	0,28%	10301	0,45%	
	Leitura	105558	4,64%	207966	9,15%	232418	10,22%	...
	Total	108716	4,78%	214274	9,42%	242719	10,67%	
tiff2rgba	Escrita	3378	0,15%	6512	0,28%	10563	0,46%	
	Leitura	104509	4,52%	205884	8,90%	230390	9,96%	
	Total	107887	4,66%	212396	9,18%	240953	10,42%	...
typeset	Escrita	3002	0,14%	5992	0,27%	9949	0,45%	
	Leitura	95068	4,29%	187146	8,44%	211530	9,54%	
	Total	98070	4,42%	193138	8,71%	221479	9,99%	
Média Escrita			0,21%		0,40%		0,71%	
Média Leitura			4,93%		7,79%		8,96%	...
Média Total			5,14%		8,20%		9,67%	

Fonte: Autor

Tabela 12 Continuação da Tabela 11.

Benchmark		Tamanho (bytes)						
		64		128		256		
	Tipo	N.Acessos	% Total	N.Acessos	% Total	N.Acessos	% Total	
basicmath	Escrita	98585	3,19%	136192	4,41%	197056	6,38%	...
	Leitura	225772	7,31%	335096	10,84%	422240	13,66%	
	Total	324357	10,50%	471288	15,25%	619296	20,04%	
bitcount	Escrita	18677	0,94%	32388	1,62%	55549	2,79%	
	Leitura	54773	2,75%	91081	4,57%	132835	6,66%	
	Total	73450	3,68%	123469	6,19%	188384	9,45%	
crc32	Escrita	15529	0,84%	27020	1,47%	47749	2,59%	...
	Leitura	55736	3,03%	85758	4,66%	121357	6,59%	
	Total	71265	3,87%	112778	6,13%	169106	9,19%	
dijkstra	Escrita	20639	0,62%	34711	1,03%	58030	1,73%	
	Leitura	1260377	37,56%	1303560	38,85%	1361554	40,57%	
	Total	1281016	38,17%	1338271	39,88%	1419584	42,30%	...
FFT	Escrita	19106	0,86%	32378	1,45%	54676	2,45%	
	Leitura	201400	9,03%	235389	10,56%	275904	12,37%	
	Total	220506	9,89%	267767	12,01%	330580	14,83%	
ispell	Escrita	17341	0,79%	28949	1,32%	50023	2,28%	
	Leitura	223040	10,17%	254410	11,60%	291829	13,31%	...
	Total	240381	10,96%	283359	12,92%	341852	15,59%	
jpegC	Escrita	17062	0,78%	28424	1,31%	49340	2,27%	
	Leitura	222934	10,24%	253503	11,64%	290166	13,33%	
	Total	239996	11,02%	281927	12,95%	339506	15,59%	
jpegD	Escrita	17302	0,76%	28664	1,27%	49580	2,19%	...
	Leitura	254435	11,24%	285004	12,59%	321952	14,22%	
	Total	271737	12,00%	313668	13,85%	371532	16,41%	
mad	Escrita	17670	0,74%	29189	1,23%	50488	2,13%	
	Leitura	286312	12,05%	317198	13,36%	354865	14,94%	
	Total	303982	12,80%	346387	14,58%	405353	17,07%	...
patricia	Escrita	277071	4,60%	412274	6,84%	624810	10,37%	
	Leitura	592892	9,84%	912481	15,14%	1201871	19,94%	
	Total	869963	14,43%	1324755	21,98%	1826681	30,30%	
qsort	Escrita	23307	0,82%	38718	1,36%	65910	2,31%	
	Leitura	315468	11,06%	361889	12,69%	446353	15,65%	
	Total	338775	11,88%	400607	14,05%	512263	17,96%	
rijndael	Escrita	17479	0,83%	29429	1,40%	50719	2,41%	
	Leitura	185249	8,80%	217789	10,35%	256059	12,16%	...
	Total	202728	9,63%	247218	11,74%	306778	14,57%	
		⋮	⋮	⋮	⋮	⋮	⋮	

Tabela 13 Continuação da Tabela 12.

Benchmark		Tamanho (bytes)						
		64		128		256		
	Tipo	N.Acessos	% Total	N.Acessos	% Total	N.Acessos	% Total	
sha	Escrita	19322	0,83%	31766	1,36%	52699	2,26%	...
	Leitura	245497	10,52%	281707	12,07%	322800	13,83%	
	Total	264819	11,35%	313473	13,43%	375499	16,09%	
stringsearch	Escrita	15000	0,98%	25743	1,69%	44567	2,92%	
	Leitura	41818	2,74%	70339	4,61%	101823	6,68%	
	Total	56818	3,73%	96082	6,30%	146390	9,60%	
susan	Escrita	18165	0,88%	31131	1,51%	53576	2,61%	...
	Leitura	80876	3,93%	118139	5,75%	165543	8,05%	
	Total	99041	4,82%	149270	7,26%	219119	10,66%	
tiff2bw	Escrita	17760	0,78%	29546	1,30%	50716	2,23%	
	Leitura	250301	11,01%	282237	12,41%	320455	14,09%	...
	Total	268061	11,79%	311783	13,71%	371171	16,32%	
tiff2rgba	Escrita	18184	0,79%	30333	1,31%	51776	2,24%	
	Leitura	248390	10,74%	281551	12,17%	321006	13,88%	
	Total	266574	11,52%	311884	13,48%	372782	16,12%	...
typeset	Escrita	17279	0,78%	28802	1,30%	49827	2,25%	
	Leitura	229392	10,35%	260526	11,75%	297815	13,43%	
	Total	246671	11,13%	289328	13,05%	347642	15,68%	
Média Escrita			1,16%		1,84%		3,02%	
Média Leitura			10,13%		11,98%		14,08%	...
Média Total			11,29%		13,82%		17,10%	

Fonte: Autor

Tabela 14 Continuação da Tabela 13.

Benchmark		Tamanho (bytes)						
		512		1024		2048		
	Tipo	N.Acessos	% Total	N.Acessos	% Total	N.Acessos	% Total	
basicmath	Escrita	276192	8,94%	378863	12,26%	504860	16,34%	...
	Leitura	555397	17,97%	703709	22,77%	890854	28,83%	
	Total	831589	26,91%	1082572	35,03%	1395714	45,17%	
bitcount	Escrita	95447	4,79%	157881	7,92%	239483	12,01%	
	Leitura	180841	9,07%	254046	12,74%	360696	18,08%	
	Total	276288	13,85%	411927	20,65%	600179	30,09%	
crc32	Escrita	81897	4,45%	135675	7,37%	207751	11,28%	...
	Leitura	163584	8,89%	228647	12,42%	320960	17,43%	
	Total	245481	13,33%	364322	19,79%	528711	28,72%	
dijkstra	Escrita	98139	2,92%	160674	4,79%	246824	7,36%	
	Leitura	1415768	42,19%	1493164	44,50%	1606773	47,88%	
	Total	1513907	45,11%	1653838	49,28%	1853597	55,24%	...
FFT	Escrita	93574	4,20%	154260	6,92%	237186	10,64%	
	Leitura	323298	14,50%	394225	17,68%	496211	22,26%	
	Total	416872	18,70%	548485	24,60%	733397	32,89%	
ispell	Escrita	85406	3,90%	141227	6,44%	230219	10,50%	
	Leitura	335484	15,30%	401655	18,32%	498499	22,74%	...
	Total	420890	19,20%	542882	24,76%	728718	33,24%	
jpegC	Escrita	84127	3,86%	138816	6,38%	227561	10,45%	
	Leitura	333281	15,31%	398596	18,31%	494270	22,70%	
	Total	417408	19,17%	537412	24,68%	721831	33,15%	
jpegD	Escrita	84740	3,74%	140814	6,22%	244491	10,80%	...
	Leitura	365118	16,13%	431154	19,04%	536343	23,69%	
	Total	449858	19,87%	571968	25,26%	780834	34,49%	
mad	Escrita	86879	3,66%	147487	6,21%	264799	11,15%	
	Leitura	398897	16,80%	466845	19,66%	585757	24,66%	
	Total	485776	20,45%	614332	25,87%	850556	35,81%	...
patricia	Escrita	889578	14,76%	1180712	19,59%	1414201	23,46%	
	Leitura	1565877	25,98%	1993298	33,07%	2473523	41,03%	
	Total	2455455	40,73%	3174010	52,66%	3887724	64,50%	
qsort	Escrita	111779	3,92%	188882	6,62%	304483	10,68%	
	Leitura	511490	17,94%	601126	21,08%	735517	25,79%	
	Total	623269	21,86%	790008	27,70%	1040000	36,47%	
rijndael	Escrita	86915	4,13%	143020	6,79%	219205	10,41%	
	Leitura	300436	14,27%	367577	17,46%	462714	21,98%	...
	Total	387351	18,40%	510597	24,25%	681919	32,39%	
		⋮	⋮	⋮	⋮	⋮	⋮	

Tabela 15 Continuação da Tabela 14.

Benchmark		Tamanho (bytes)						
		512		1024		2048		
	Tipo	N.Acessos	% Total	N.Acessos	% Total	N.Acessos	% Total	
sha	Escrita	88947	3,81%	144765	6,20%	242771	10,40%	...
	Leitura	366781	15,71%	435105	18,64%	537926	23,05%	
	Total	455728	19,53%	579870	24,84%	780697	33,45%	
stringsearch	Escrita	75182	4,93%	123342	8,09%	189540	12,43%	
	Leitura	140577	9,22%	197944	12,98%	280483	18,39%	
	Total	215759	14,15%	321286	21,07%	470023	30,82%	...
susan	Escrita	91931	4,47%	152713	7,43%	233850	11,37%	
	Leitura	214754	10,45%	288504	14,03%	396184	19,27%	
	Total	306685	14,92%	441217	21,46%	630034	30,65%	
tiff2bw	Escrita	86639	3,81%	143754	6,32%	245386	10,79%	...
	Leitura	364481	16,03%	431667	18,98%	536448	23,59%	
	Total	451120	19,84%	575421	25,31%	781834	34,38%	
tiff2rgba	Escrita	88485	3,83%	146647	6,34%	247268	10,69%	
	Leitura	365814	15,81%	434131	18,77%	539255	23,31%	
	Total	454299	19,64%	580778	25,11%	786523	34,00%	...
typeset	Escrita	85058	3,84%	140566	6,34%	232184	10,47%	
	Leitura	341302	15,39%	407366	18,37%	505667	22,81%	
	Total	426360	19,23%	547932	24,71%	737851	33,28%	
Média Escrita			4,89%		7,68%		11,74%	
Média Leitura			16,50%		19,93%		24,86%	...
Média Total			21,38%		27,61%		36,60%	

Fonte: Autor

Tabela 16 Continuação da Tabela 15.

Benchmark		Tamanho (bytes)			
		4096		8192	
	Tipo	N.Acessos	% Total	N.Acessos	% Total
basicmath	Escrita	650640	21,06%	777822	25,17%
	Leitura	1124614	36,40%	1359888	44,01%
	Total	1775254	57,45%	2137710	69,18%
bitcount	Escrita	333495	16,72%	434478	21,78%
	Leitura	486422	24,39%	630333	31,60%
	Total	819917	41,11%	1064811	53,39%
crc32	Escrita	296321	16,10%	389604	21,16%
	Leitura	435864	23,67%	569123	30,91%
	Total	732185	39,77%	958727	52,07%
dijkstra	Escrita	353335	10,53%	463225	13,80%
	Leitura	1747319	52,07%	1906718	56,82%
	Total	2100654	62,60%	2369943	70,62%
FFT	Escrita	366406	16,43%	486770	21,83%
	Leitura	643665	28,87%	808764	36,27%
	Total	1010071	45,30%	1295534	58,11%
ispell	Escrita	360115	16,43%	475500	21,69%
	Leitura	652429	29,76%	811099	37,00%
	Total	1012544	46,19%	1286599	58,69%
jpegC	Escrita	356261	16,36%	470334	21,60%
	Leitura	646781	29,70%	803790	36,92%
	Total	1003042	46,07%	1274124	58,52%
jpegD	Escrita	376611	16,63%	491220	21,69%
	Leitura	697565	30,81%	855190	37,77%
	Total	1074176	47,44%	1346410	59,46%
mad	Escrita	398483	16,78%	515061	21,69%
	Leitura	753321	31,72%	913294	38,45%
	Total	1151804	48,50%	1428355	60,14%
patricia	Escrita	1599149	26,53%	1758015	29,16%
	Leitura	2913698	48,34%	3188032	52,89%
	Total	4512847	74,87%	4946047	82,05%
qsort	Escrita	484450	16,99%	647372	22,70%
	Leitura	936630	32,85%	1164693	40,84%
	Total	1421080	49,83%	1812065	63,55%
rijndael	Escrita	339987	16,15%	454831	21,61%
	Leitura	598872	28,45%	757274	35,97%
	Total	938859	44,60%	1212105	57,58%
		⋮	⋮	⋮	⋮

Tabela 17 Continuação da Tabela 16.

Benchmark		Tamanho (bytes)			
		4096		8192	
	Tipo	N.Acessos	% Total	N.Acessos	% Total
sha	Escrita	383721	16,44%	511091	21,90%
	Leitura	705917	30,24%	878489	37,64%
	Total	1089638	46,69%	1389580	59,54%
stringsearch	Escrita	269937	17,70%	340422	22,32%
	Leitura	383398	25,14%	495759	32,51%
	Total	653335	42,85%	836181	54,84%
susan	Escrita	330951	16,10%	432891	21,06%
	Leitura	523920	25,48%	671380	32,66%
	Total	854871	41,58%	1104271	53,71%
tiff2bw	Escrita	379156	16,68%	494750	21,76%
	Leitura	699256	30,75%	858474	37,76%
	Total	1078412	47,43%	1353224	59,51%
tiff2rgba	Escrita	383429	16,58%	501912	21,70%
	Leitura	703843	30,43%	866358	37,45%
	Total	1087272	47,00%	1368270	59,15%
typeset	Escrita	362566	16,35%	478188	21,57%
	Leitura	661765	29,85%	820897	37,02%
	Total	1024331	46,20%	1299085	58,59%
Média Escrita			17,03%		21,90%
Média Leitura			31,61%		38,58%
Média Total			48,64%		60,48%

Fonte: Autor

Apêndice I – Tabela referente ao *trace* de instruções (x86).

Tabela 18 Classificação das instruções dos *benchmarks*.

Benchmark	Tipo de instrução								
	Move	%	Desvios	%	Compare	%	Aritméticas	%	
basicmath	15348249	99,21%	30793	0,20%	25764	0,17%	21549	0,14%	
bitcount	259082	32,33%	107208	13,38%	46864	5,85%	203506	25,39%	...
crc32	74976	97,13%	511	0,66%	411	0,53%	375	0,49%	
dijkstra	129669	88,95%	4658	3,20%	4472	3,07%	2634	1,81%	
fft	4292433	98,01%	19409	0,44%	16365	0,37%	16125	0,37%	...
patricia	7299983	98,85%	21990	0,30%	18372	0,25%	15016	0,20%	
qsort	6198197	98,88%	18481	0,29%	16100	0,26%	12085	0,19%	
sha	851592	98,58%	1532	0,18%	1417	0,16%	4907	0,57%	...
susan	165604	80,60%	5234	2,55%	4980	2,42%	25250	12,29%	
	Move	%	Desvios	%	Compare	%	Aritméticas	%	
Soma	34619785	97,26%	209816	0,59%	134745	0,38%	301447	0,85%	...
Média	3846643	88,06%	23312,89	2,35%	14971,67	1,45%	33494,11	4,61%	
Maior	15348249	99,21%	107208	13,38%	46864	5,85%	203506	25,39%	
Menor	74976	32,33%	511	0,18%	411	0,16%	375	0,14%	...

Fonte: Autor

Tabela 19 Continuação da Tabela 18.

Benchmark	Tipo de instrução				
	Lógicas	%	Outras	%	Total
basicmath	8151	0,05%	35209	0,23%	15469715
bitcount	80023	9,99%	104734	13,07%	801417
crc32	205	0,27%	716	0,93%	77194
dijkstra	315	0,22%	4036	2,77%	145784
fft	5293	0,12%	30178	0,69%	4379803
patricia	4972	0,07%	24704	0,33%	7385037
qsort	4455	0,07%	18860	0,30%	6268178
sha	3017	0,35%	1379	0,16%	863844
susan	867	0,42%	3531	1,72%	205466
	Lógicas	%	Outras	%	Total
Soma	107298	0,30%	223347	0,63%	35596438
Média	11922	1,28%	24816,33	2,24%	3955160
Maior	80023	9,99%	104734	13,07%	
Menor	205	0,05%	716	0,16%	

Fonte: Autor

Apêndice J –Tabela dos endereços mais acessados do *trace* de dados (x86).

Tabela 20 Endereços mais acessados do *trace* de dados.

Benchmark	Tamanho (bytes)										
	4		8		16		32		64		
	N. Acessos	% Total	N. Acessos	% Total	N. Acessos	% Total	N. Acessos	% Total	N. Acessos	% Total	
basicmath	342046	2,13%	616429	3,83%	1078626	6,70%	1635303	10,16%	2444439	15,19%	
bitcount	1580138	10,39%	3160218	20,78%	5530682	36,36%	6503934	42,76%	6723399	44,21%	
crc32	80548	0,91%	134159	1,52%	218133	2,47%	331677	3,76%	491624	5,57%	
dijkstra	406104	2,71%	812129	5,43%	1334962	8,92%	1999843	13,36%	2792728	18,66%	...
FFT	518937	3,44%	1007816	6,67%	1919703	12,71%	3144544	20,83%	5591441	37,03%	
patricia	353272	1,86%	636982	3,35%	1120420	5,89%	1724951	9,07%	2536445	13,34%	
qsort	82078	0,90%	135689	1,48%	219663	2,40%	336459	3,68%	500108	5,47%	
sha	2230907	10,91%	4082928	19,98%	6181224	30,24%	7948753	38,89%	8445708	41,32%	...
susan	81820	0,86%	135431	1,42%	219405	2,30%	335749	3,52%	499097	5,24%	
Média		3,79%		7,16%		12,00%		16,23%		20,67%	
Maior		10,91%		20,78%		36,36%		42,76%		44,21%	
Menor		0,86%		1,42%		2,30%		3,52%		5,24%	...
Desvio		3,99%		7,71%		12,64%		15,04%		15,94%	

Fonte: Autor

Tabela 21 Continuação da Tabela 20.

Benchmark	Tamanho (bytes)										
	128		256		512		1024		2048		
	N. Acessos	% Total	N. Acessos	% Total	N. Acessos	% Total	N. Acessos	% Total	N. Acessos	% Total	
basicmath	3654943	22,71%	5180562	32,19%	6797343	42,23%	8296358	51,55%	9942463	61,77%	
bitcount	7001185	46,03%	7410042	48,72%	7975479	52,44%	8726801	57,38%	9668599	63,57%	
crc32	751505	8,52%	1136127	12,88%	1681307	19,07%	2421279	27,46%	3349352	37,98%	
dijkstra	3853273	25,75%	5164520	34,51%	6573555	43,92%	7801577	52,13%	9008005	60,19%	...
FFT	6590303	43,65%	7109540	47,09%	7745615	51,30%	8543643	56,58%	9520603	63,05%	
patricia	3685547	19,38%	5368298	28,23%	7481366	39,34%	9652755	50,76%	11989295	63,04%	
qsort	760266	8,32%	1145821	12,54%	1694833	18,54%	2439090	26,69%	3387660	37,06%	
sha	8917501	43,63%	9838171	48,13%	11172724	54,66%	12592167	61,61%	14100464	68,99%	...
susan	762068	8,00%	1156670	12,14%	1711939	17,96%	2462450	25,84%	3421155	35,90%	
Média		25,11%		30,71%		37,72%		45,55%		54,62%	
Maior		46,03%		48,72%		54,66%		61,61%		68,99%	
Menor		8,00%		12,14%		17,96%		25,84%		35,90%	...
Desvio		15,86%		15,46%		15,23%		14,57%		13,45%	

Fonte: Autor

Tabela 22 Continuação da Tabela 21.

Benchmark	Tamanho (bytes)					
	4096		8192			
	N. Acessos	% Total	N. Acessos	% Total	T.Endereços	T.Acessos
basicmath	11470532	71,27%	12719887	79,03%	450000	16094776
bitcount	10760827	70,75%	11898284	78,23%	450000	15209126
crc32	4407507	49,98%	5498923	62,35%	450000	8818774
dijkstra	10279734	68,68%	11537053	77,09%	450000	14966595
FFT	10610601	70,27%	11716765	77,60%	450000	15099271
patricia	14110733	74,20%	15594468	82,00%	450000	19018236
qsort	4565472	49,95%	5765706	63,08%	450000	9139997
sha	15502272	75,85%	16863734	82,51%	450000	20439333
susan	4563315	47,88%	5781235	60,66%	450000	9530919
Média		64,31%		73,62%		
Maior		75,85%		82,51%		
Menor		47,88%		60,66%		
Desvio		11,49%		8,90%		

Fonte: Autor

Apêndice K –Tabela dos endereços mais acessados (leitura e escrita) do *trace* de dados (x86)

Tabela 23 Endereços mais acessados do trace de dados (separação em leitura e escrita).

Benchmark		Tamanho (bytes)						
		8		16		32		
	Tipo	N.Acessos	% Total	N.Acessos	% Total	N.Acessos	% Total	
basicmath	Leitura	258053	1,84%	460796	3,28%	831142	5,92%	...
	Escrita	171837	1,22%	246377	1,76%	392557	2,80%	
	Total	429890	3,06%	707173	5,04%	1223699	8,72%	
bitcount	Leitura	790123	5,98%	1580244	11,97%	3160414	23,94%	
	Escrita	790013	5,98%	1579987	11,97%	2385401	18,07%	
	Total	1580136	11,97%	3160231	23,94%	5545815	42,01%	
crc32	Leitura	80356	1,18%	118975	1,74%	189985	2,78%	...
	Escrita	15448	0,23%	30440	0,45%	57951	0,85%	
	Total	95804	1,40%	149415	2,19%	247936	3,63%	
dijkstra	Leitura	266256	2,08%	422643	3,29%	675320	5,26%	
	Escrita	391718	3,05%	783407	6,11%	947958	7,39%	
	Total	657974	5,13%	1206050	9,40%	1623278	12,65%	...
FFT	Leitura	335747	2,57%	641243	4,91%	1247378	9,56%	
	Escrita	183377	1,41%	366555	2,81%	672832	5,16%	
	Total	519124	3,98%	1007798	7,72%	1920210	14,72%	
patricia	Leitura	273163	1,62%	481391	2,85%	861103	5,09%	
	Escrita	177882	1,05%	257364	1,52%	412328	2,44%	...
	Total	451045	2,67%	738755	4,37%	1273431	7,53%	
qsort	Leitura	81846	1,15%	120465	1,70%	191475	2,70%	
	Escrita	15448	0,22%	30440	0,43%	57951	0,82%	
	Total	97294	1,37%	150905	2,13%	249426	3,51%	
sha	Leitura	151648	1,49%	233488	2,29%	348171	3,42%	...
	Escrita	80658	0,79%	161273	1,58%	197849	1,94%	
	Total	232306	2,28%	394761	3,88%	546020	5,36%	
susan	Leitura	81594	1,11%	120213	1,64%	191223	2,61%	
	Escrita	15448	0,21%	30440	0,42%	57951	0,79%	
	Total	97042	1,32%	150653	2,06%	249174	3,40%	...
Média Escrita			1,57%		3,00%		4,47%	
Maior Leitura			2,11%		3,74%		6,81%	
Média Total			3,69%		6,75%		11,28%	

Tabela 24 Continuação da Tabela 23.

Benchmark		Tamanho (bytes)						
		64		128		256		
	Tipo	N.Acessos	% Total	N.Acessos	% Total	N.Acessos	% Total	
basicmath	Leitura	1168329	8,33%	1688808	12,03%	2386277	17,00%	...
	Escrita	611947	4,36%	959552	6,84%	1467014	10,45%	
	Total	1780276	12,69%	2648360	18,87%	3853291	27,46%	
bitcount	Leitura	4106019	31,10%	4272721	32,36%	4470054	33,86%	
	Escrita	2440574	18,49%	2522877	19,11%	2613731	19,80%	
	Total	6546593	49,59%	6795598	51,47%	7083785	53,65%	
crc32	Leitura	268464	3,94%	390093	5,72%	566793	8,31%	...
	Escrita	106870	1,57%	169935	2,49%	257025	3,77%	
	Total	375334	5,50%	560028	8,21%	823818	12,08%	
dijkstra	Leitura	1044927	8,14%	1497576	11,67%	2056983	16,03%	
	Escrita	1184196	9,23%	1514716	11,80%	1908639	14,87%	
	Total	2229123	17,37%	3012292	23,48%	3965622	30,91%	...
FFT	Leitura	1890551	14,49%	3114859	23,87%	3969339	30,42%	
	Escrita	1284527	9,84%	2507242	19,22%	2725915	20,89%	
	Total	3175078	24,33%	5622101	43,09%	6695254	51,31%	
patricia	Leitura	1243882	7,35%	1822028	10,77%	2525886	14,93%	
	Escrita	615718	3,64%	911528	5,39%	1409908	8,34%	...
	Total	1859600	11,00%	2733556	16,16%	3935794	23,27%	
qsort	Leitura	273659	3,85%	395527	5,57%	574426	8,09%	
	Escrita	106870	1,51%	172047	2,42%	258986	3,65%	
	Total	380529	5,36%	567574	7,99%	833412	11,74%	
sha	Leitura	478952	4,70%	671096	6,59%	1054178	10,35%	...
	Escrita	251931	2,47%	337953	3,32%	489586	4,81%	
	Total	730883	7,18%	1009049	9,91%	1543764	15,16%	
susan	Leitura	273115	3,73%	401727	5,48%	587130	8,01%	
	Escrita	106870	1,46%	171741	2,34%	258672	3,53%	
	Total	379985	5,19%	573468	7,83%	845802	11,54%	...
Média Escrita			5,84%		8,10%		10,01%	
Maior Leitura			9,51%		12,68%		16,33%	
Média Total			15,36%		20,78%		26,35%	

Tabela 25 Continuação da Tabela 24.

Benchmark		Tamanho (bytes)						
		512		1024		2048		
	Tipo	N.Acessos	% Total	N.Acessos	% Total	N.Acessos	% Total	
basicmath	Leitura	3494918	24,91%	4546926	32,40%	5389063	38,40%	...
	Escrita	2008273	14,31%	2492089	17,76%	3048429	21,72%	
	Total	5503191	39,22%	7039015	50,16%	8437492	60,13%	
bitcount	Leitura	4715558	35,72%	5034193	38,13%	5455710	41,32%	
	Escrita	2750990	20,84%	2934135	22,22%	3177338	24,07%	
	Total	7466548	56,55%	7968328	60,35%	8633048	65,39%	
crc32	Leitura	789877	11,58%	1100442	16,13%	1513772	22,19%	...
	Escrita	389715	5,71%	570066	8,36%	809553	11,87%	
	Total	1179592	17,29%	1670508	24,49%	2323325	34,05%	
dijkstra	Leitura	2732865	21,30%	3521247	27,44%	4158990	32,41%	
	Escrita	2547507	19,85%	3204960	24,98%	3621203	28,22%	
	Total	5280372	41,15%	6726207	52,42%	7780193	60,63%	...
FFT	Leitura	4312169	33,05%	4668332	35,78%	5119649	39,24%	
	Escrita	2889457	22,15%	3094450	23,72%	3350702	25,68%	
	Total	7201626	55,19%	7762782	59,50%	8470351	64,92%	
patricia	Leitura	3660464	21,64%	5053552	29,88%	6336457	37,47%	
	Escrita	2020797	11,95%	2758791	16,31%	3577221	21,15%	...
	Total	5681261	33,59%	7812343	46,19%	9913678	58,62%	
qsort	Leitura	798943	11,25%	1109327	15,62%	1525734	21,49%	
	Escrita	391836	5,52%	573005	8,07%	818827	11,53%	
	Total	1190779	16,77%	1682332	23,69%	2344561	33,02%	
sha	Leitura	1773825	17,42%	2472101	24,27%	3146723	30,90%	...
	Escrita	672849	6,61%	986668	9,69%	1492234	14,65%	
	Total	2446674	24,02%	3458769	33,96%	4638957	45,55%	
susan	Leitura	819572	11,19%	1137880	15,53%	1570741	21,44%	
	Escrita	391496	5,34%	572766	7,82%	816354	11,14%	
	Total	1211068	16,53%	1710646	23,35%	2387095	32,58%	...
Média Escrita			12,48%		15,44%		18,89%	
Maior Leitura			20,89%		26,13%		31,65%	
Média Total			33,37%		41,57%		50,54%	

Tabela 26 Continuação da Tabela 25.

Benchmark		Tamanho (bytes)			
		4096		8192	
	Tipo	N.Acessos	% Total	N.Acessos	% Total
basicmath	Leitura	6283302	44,78%	7076821	50,43%
	Escrita	3615615	25,77%	4087549	29,13%
	Total	9898917	70,54%	11164370	79,56%
bitcount	Leitura	5945210	45,03%	6506924	49,28%
	Escrita	3506707	26,56%	3912651	29,64%
	Total	9451917	71,59%	10419575	78,92%
crc32	Leitura	1994227	29,23%	2538528	37,21%
	Escrita	1135636	16,65%	1513571	22,19%
	Total	3129863	45,88%	4052099	59,39%
dijkstra	Leitura	4792607	37,35%	5449990	42,47%
	Escrita	4025615	31,37%	4492703	35,01%
	Total	8818222	68,72%	9942693	77,49%
FFT	Leitura	5628975	43,14%	6189163	47,43%
	Escrita	3685586	28,25%	4073848	31,22%
	Total	9314561	71,39%	10263011	78,66%
patricia	Leitura	7640679	45,18%	8757716	51,78%
	Escrita	4449099	26,31%	5132477	30,35%
	Total	12089778	71,48%	13890193	82,13%
qsort	Leitura	2019713	28,45%	2621396	36,92%
	Escrita	1163737	16,39%	1624186	22,88%
	Total	3183450	44,84%	4245582	59,80%
sha	Leitura	3906982	38,36%	4594721	45,11%
	Escrita	1941879	19,07%	2445686	24,01%
	Total	5848861	57,43%	7040407	69,13%
susan	Leitura	2089791	28,52%	2696722	36,81%
	Escrita	1150325	15,70%	1576731	21,52%
	Total	3240116	44,22%	4273453	58,33%
Média Escrita			22,90%		27,33%
Maior Leitura			37,78%		44,16%
Média Total			60,68%		71,49%

Apêndice L – Publicações geradas durante o curso

Resultados parciais do trabalho apresentado nesta dissertação foram publicados nos anais dos seguintes eventos:

- OLIVEIRA, LIZANDRO ; MATTOS, JULIO C. B. ; BRISOLARA, Lisane . Survey of Memory Optimization Techniques for Embedded Systems. In: 2013 III Brazilian Symposium on Computing Systems Engineering (SBESC), 2013, Niteroi. 2013 III Brazilian Symposium on Computing Systems Engineering. p. 65-6. (Qualis B4)
- OLIVEIRA, L. S. ; MATTOS, J. C. B. ; BRISOLARA, Lisane . Estudo e Avaliação de Arquiteturas de Processadores Embarcados quanto ao uso Eficiente de Memória. In: ENPOS - Encontro de Pós-Graduação UFPel, 2013, Pelotas. XV ENPOS - Encontro de Pós-Graduação UFPel, 2013. p. 1-4.
- OLIVEIRA, L. S. ; MATTOS, J. C. B. ; BRISOLARA, Lisane . Utilização de plataforma virtual para análise do desempenho de arquiteturas de processadores embarcados. In: Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul, 2014, Alegrete. ERAD/RS 2014, 2014. p. 129-130. (Qualis B5)
- SILVA, Lisandro Luiz ; OLIVEIRA, L. S. ; BRISOLARA, Lisane ; MATTOS, J. C. B. . Estudo do acesso à memória em processadores embarcados. In: XXIII Congresso de Iniciação Científica da Universidade Federal de Pelotas, 2014, Pelotas. Anais do XXIII Congresso de Iniciação Científica da Universidade Federal de Pelotas, 2014.

Outros trabalhos publicados durante o curso:

- OLIVEIRA, L. S. . Método das Diferenças no Domínio do Tempo e algoritmo de Yee: uma proposta para simulação do funcionamento de antenas *microstrip*. 2011. I Workshop-Escola de Informática Teórica (WEIT 2011).
- OLIVEIRA, L. S. ; ROSA JUNIOR, L. S. ; MARQUES, F. S. ; MATTOS, J. C. B. . Descrição do fluxo de projeto e do processo de otimização de um circuito digital na tecnologia CMOS. In: XIV Encontro de Pós-Graduação - PRPPG/UFPel, 2012, Pelotas. Anais do XIV Encontro de Pós-Graduação - PRPPG/UFPel, 2012.