Análise de código ARM e Thumb em Raspberry Pi

Aulos Plautius Martines Marino[1], **Sarita Mazzini Bruschi**[1]

[1]Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo (USP)
Avenida Trabalhador São-carlense, 400, 13566-590 - São Carlos - SP

aulos@usp.br, sarita@icmc.usp.br

Abstract. *The usage of ARM processors has become widespread in the lives of most people in the 21st century, whether directly in the hands of people as consumer products such as smartphones and tablets or in infrastructure and embedded computing. This study analyses the differences in performance between the ARM and Thumb instruction sets on the popular Raspberry Pi single board computer as to better understand code optimization strategies for the ARM architecture.*

Resumo. *O uso de processadores ARM se tornou muito difundido na vida da maioria das pessoas durante o século 21, seja diretamente nas mãos das pessoas como produtos de consumo como smartphones e tablets ou em infraestrutura e computação embarcada. Este estudo analisa as diferenças de performance entre os conjuntos de instrução ARM e Thumb no Raspberry Pi, um popular computador de placa única, para entender melhor estratégias de otimização de código para a arquitetura ARM.*

1. Introdução

A arquitetura ARM é uma das mais prevalentes na indústria. Em 2018, 33% de todos os chips com processadores do mundo possuiam arquitetura ARM. Em 2019 foram vendidos cerca de 22,8 bilhões de dispositivos ARM, totalizando mais de 140 bilhões de processadores feitos em toda a história da arquitetura. A arquitetura ARM domina o mercado mobile, tendo mais de 90% de todos os processadores móveis para smartphones e tablets como chips ARM. Assim como no mercado mobile, processadores ARM dominam o mercado de sistemas embarcados e IoT (*Internet of Things* – Internet das Coisas), estando em mais de 90% de todos os dispositivos desta área e 75% de todos os chips automotivos [Arm 2109].

A arquitetura ARM possui várias características interessantes e que podem contribuir para que esta tenha um desempenho melhor na execução das aplicações. Uma delas é a definição de um conjunto de instruções, denominado Thumb, que é codificado em apenas 16 bits, e não em 32 bits, como o conjunto de instruções padrão da arquitetura. Outra característica é a execução condicionada de instruções, onde a própria instrução define a condição de execução, sem precisar passar por uma instrução anterior de comparação. Essas duas características podem levar a um código mais compacto (ocupa menos espaço na memória) e que consuma menos energia (menos instruções para executar).

Este projeto foi concebido com a intenção de comparar várias métricas de performance entre o conjunto de instruções ARM e Thumb no processador BCM2836, de arquitetura Cortex-A7, presente no SBC Raspberry Pi 2 Model B, em particular a diferença

de tamanho do código gerado, tempo de execução, predição de desvio correta e taxa de acerto na cache entre os conjuntos de instrução. Houve um objetivo inicial de análise de consumo de energia porém, devido à pandemia de COVID-19 de 2020, a medição de consumo de energia foi impossibilitada, pois ela necessita de equipamentos e auxilio de expertise eletrônica que ficaram de díficil acesso devido ao isolamento social e lockdown praticados durante a crise.

O artigo está estruturado da seguinte forma: a Seção 2 apresenta um referencial teórico necessário para o entedimento do problema. A Seção 3 apresenta os passos seguidos para que a pesquisa fosse desenvolvida. A Seção 4 ilustra os resultados obtidos com a experimentação e a Seção 5 as conclusões do projeto.

2. Referencial Teórico

ARM é uma família de arquiteturas de tipo RISC frequentemente utilizada em dispositivos móveis e embarcados devido a seu baixo consumo de energia em relação à arquiteturas CISC, como a x86-64. Muitos processadores ARM são flexíveis dado que possuem vários modos de excução e suportes para mais de um conjunto de instrução. Os modos relevantes a este trabalho são o modo ARM e o modo Thumb.

O conjunto de instruções ARM é o padrão utilizado pelo processador utilizado neste projeto opera em 32-bits, enquanto o conjunto de instruções Thumb-2 e opera em 16-bits. Para o processador executar instruções Thumb-2 é necessário mudar seu modo de operação para *thumb state*. O registrador **CPSR** (*Current Processor Status Register*) [Arm 2020] contêm o bit **T** que determina o modo de operação do processador, quando este bit é definido como **1** o processador entra em *thumb state*. A Figura 1 ilustra a associação entre instruções ARM e Thumb [Shrivastava 2006] e na Figura 2 é possível observar o registrador CPSR

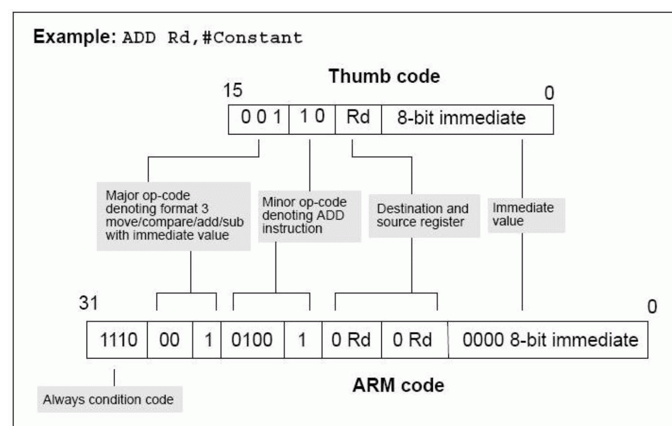

Figura 1. Associação entre a codificação de instrução ARM e Thumb [Shrivastava 2006]

Uma vez que esteja operando neste estado, a instrução thumb carregada na memória é primeiro enviada a um circuito de decodificação e neste circuito a instrução é convertida para ARM 32-bits para ser executada. O processo de decodificação não acata em ciclos de clock adicionais.

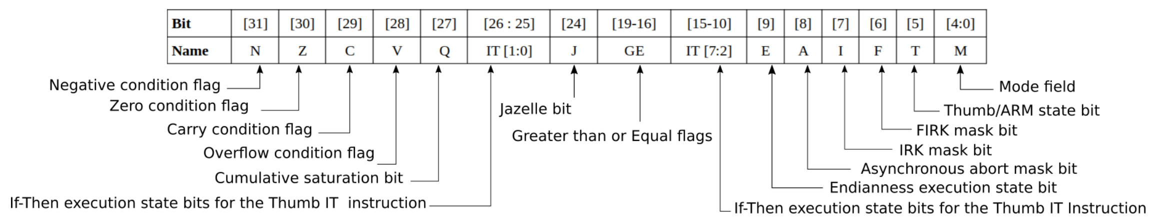

Figura 2. Conteúdo do registrador CPSR

Otimização	Conjunto de instrução
O0	Thumb
O0	ARM
O1	Thumb
O1	ARM
O2	Thumb
O2	ARM
O3	Thumb
O3	ARM

Tabela 1. Experimentos

Devido as instruções thumb serem mais simples do que as ARM dado o seu tamanho reduzido, um programa escrito em thumb necessita de mais instruções do que a versão ARM. Porém, mesmo tendo um maior número de instruções, o executável final terá uma redução de tamanho em 35% em comparação com a versão compilada em ARM [Arm].

3. Metodologia

Para obter os dados de desempenho inicialmente foi escolhido um algoritmo de multiplicação de matrizes por ser um programa usual para medição de desempenho e também devido ao modelo de acesso à memória.

3.1. Implementação

Para este projeto foi utilizada a linguagem de programação C e o compilador GCC 6.3.0 em ambiente ARM nativo. O código instancia três matrizes 1000x1000 (A, B, C), sendo que as matrizes A e B são parâmetros e são populadas com valores aleatórios a partir de uma semente pré-definida e a matriz C (do resultado) é inicializada com zeros.

O programa foi compilado em duas versões: uma com compilação padrão, gerando um código com instruções ARM, e outro com a flag de compilação `-mthumb`, gerando código com instruções Thumb.

Cada versão foi compilada utilizando quatro níveis de otimização do GCC; **O0**, **O1**, **O2**, **O3**. Em **O0** o compilador não utiliza nenhuma estratégia de otimização e em **O3** o compilador utiliza estratégias agressivas de otimização.

Na Tabela 1 pode ser observado um resumo de todos os experimentos realizados.

3.2. Medição de performance

As métricas foram obtidas utilizando o software **perf** de medição de performance. As métricas escolhidas foram: Tempo de execução, L1-dcache-loads, L1-dcache-misses, pagefaults, branches, branch-misses, LCC-loads, LCC-misses e número de instruções executadas.

Para atenuar diferenças nos resultados que podem ser causadas pelo sistema operacional, cada programa foi executado cinquenta vezes e foram calculadas a média de cada parâmetro e o intervalo de confiança com nível de confiança de 95%.

Além da compilação do código em C gerando instruções ARM e Thumb, foi feito também uma codificação manual da multiplicação de matrizes em assembly ARM e Thumb. Porém foi descoberto que as implementações manuais não eram afetadas pelos níveis de otimização do compilador, então essa linha de questionamento foi abandonada.

4. Resultados

O tamanho dos executáveis gerados não diferem entre a versão ARM e a versão Thumb, com as versões escritas em assembly tendo 8.8 KB e as versões escritas puramente em C tendo 8.5 KB. A redução do tamanho do executável causada pelo Thumb é bem documentada na literatura, de fato, é a razão pela qual este conjunto de instruções foi implementado na arquitetura. Logo é levantada a hipótese que a redução de tamanho apenas ocorre a partir de um certo tamanho de código.

Os gráficos ilustrados na próximas figura demonstram os resultados de cada nivel de otimização e implementação para cada parâmetro inspecionado. Os gráficos nas Figuras 3, 4 e 5 estão em sua escala original, porém os demais gráficos tiveram a escala vertical multiplicada por 1.3 para facilitar a visualização das diferenças entre os resultados.

Como esperado, os experimentos mostram uma clara melhoria em utilizar as opções de otimização do compilador, porém algumas métricas não mostram melhoria significativa entre **O2** e **O3**.

É visível que não houve diferença entre os conjuntos de instruções ARM e Thumb em relação ao tempo de execução (Figura 3), ao número de instruções (Figura 4) e ao número de loads efetuados em L1 (Figura 5).

A diferença entre os conjuntos de instruções fica notável quando são estudados o número de L1 misses (Figura 6) onde ocorrem um maior número de misses em implementações Thumb.

A implementação Thumb possui uma leve vantagem em relação ao número de branches (Figura 7) ao código ARM, porém apresenta mais branch misses (Figura 8) quando compilado com os níveis de otimização **O0** e **O3**.

A quantidade de pagefault (Figura 9) não varia consideravelmente entre entre o código ARM e Thumb, assim como entre os níveis de otimização.

Também foi observado que o código Thumb executa mais LCC loads (Figura 10) e obtém menos LCC misses (Figura 11) do que o código ARM.

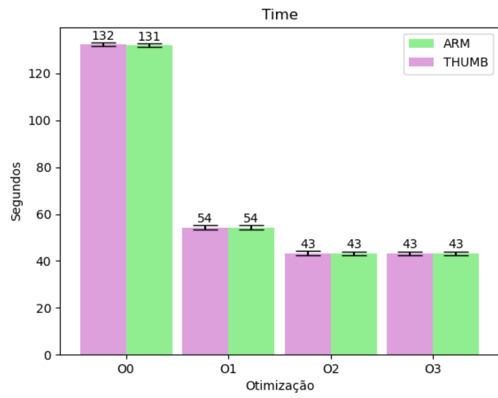

Figura 3. Tempo de execução (sem escala)

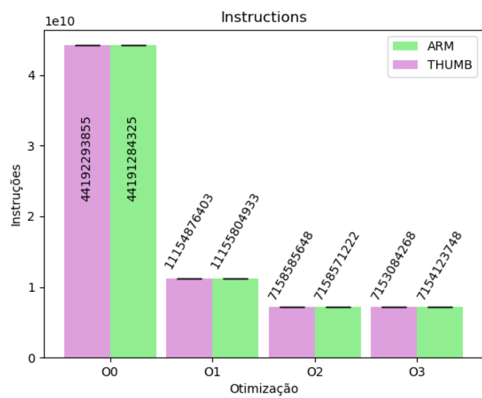

Figura 4. Número de instruções (sem escala)

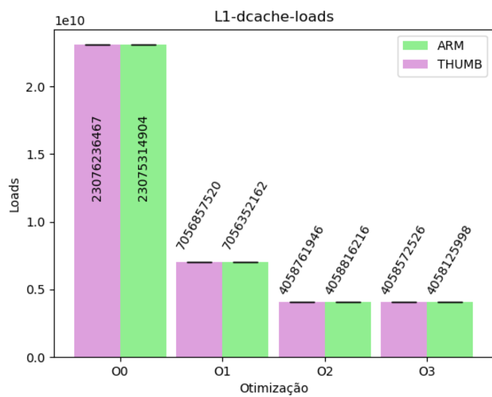

Figura 5. L1-dcache-load (sem escala)

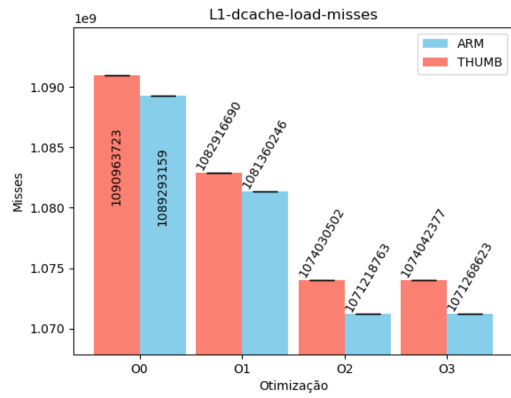

Figura 6. L1-dcache-load-misses

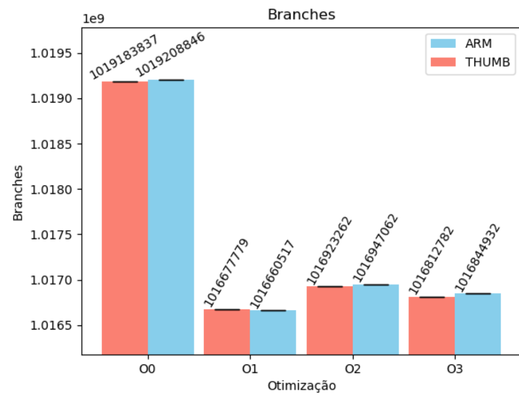

Figura 7. Branches

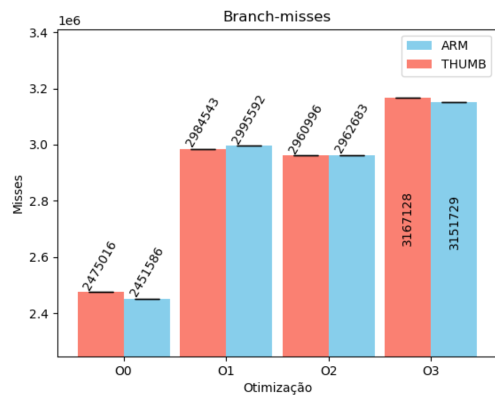

Figura 8. Branch misses

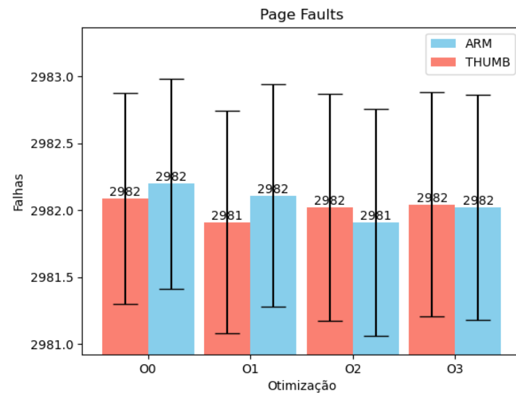

Figura 9. Page faults

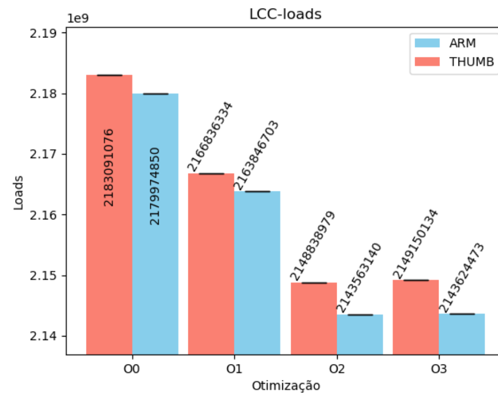

Figura 10. LCC loads

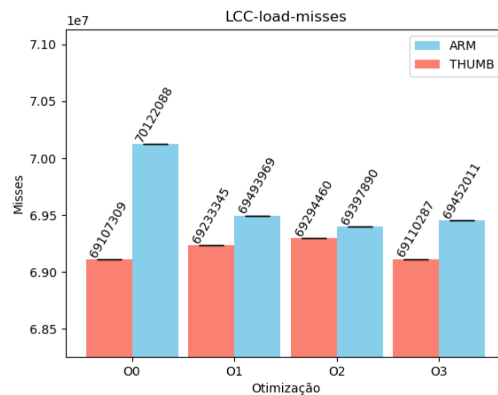

Figura 11. LCC misses

5. Conclusão

O objetivo deste trabalho foi fazer uma análise de desempenho entre códigos gerados com o conjunto de instruções ARM e Thumb, atráves da comparação de certas métricas

obtidas pelos programas teste compilados em quatro níveis de otimização presentes no compilador GCC.

Para o algoritmo escolhido, não houve diferença notável entre os tempos de execução e page faults entre as implementações ARM e Thumb. A implementação Thumb obteve um maior número de misses ao tentar carregar dados da cache L1 o que leva a um número elevado de carregamentos na cache LCC, o fato disso não ter afetado de forma significante o tempo de execução do código Thumb pode ser atribuído ao pequeno tamanho da função testada o que também pode ser indicativo da pequena diferença de número de instruções observado.

Com isso concluímos que o conjunto de instruções Thumb possuí um desempenho inferior ao ARM em relação ao uso de cache.

Esta linha de pesquisa pode ser explorada no futuro utilizando algoritmos de benchmarking mais complexos, avaliando tamanho dos executáveis e mais interessantemente, o consumo de energia, cujos resultados podem levar a uma nova pesquisa em técnicas de otimização para redução do consumo em dispósitivos móveis e embarcados.

Através deste projeto o aluno pode se familializar com conceitos de otimização de código e aspectos de programação baixo-nível em processadores ARM.

Referências

Arm. The thumb instruction set. https://developer.arm.com/documentation/ddi0210/c/Introduction/Architecture/The-Thumb-instruction-set.

Arm (2020). Cmis-core (cortex-a). https://arm-software.github.io/CMSIS_5/Core_A/html/group__CMSIS__CPSR.html.

Arm (2109). Q1 2019 roadshow slides. https://group.softbank/system/files/pdf/ir/presentations/2019/arm-roadshow-slides_q4fy2019_01_en.pdf.

Shrivastava, A. (2006). *Compiler-in-the-Loop Exploration of Programmable Embedded Systems*. PhD thesis, USA.