ANÁLISE DO COMPORTAMENTO DA TÉCNICA DE REUSO DE TRAÇOS EM ARQUITETURAS ARM

GIOVANE DE OLVEIRA TORRES[1]; MAURÍCIO LIMA PILLA[2]

[1]*Universidade Federal de Pelotas – gdotorres@inf.ufpel.edu.br*
[2]*Universidade Federal de Pelotas – pilla@inf.ufpel.edu.br*

1. INTRODUÇÃO

Existe hoje uma grande difusão de dispositivos portáteis os quais possuem no consumo de energia uma questão crítica. Neste contexto inserem-se as arquiteturas ARM, já que elas necessitam possuir um consumo de energia reduzido sem abdicar completamente do desempenho. Outras características desta categoria de arquiteturas incluem a simplicidade e a sua modularidade (RYZHYK, 2006), além da presença de instruções condicionais, as quais podem ou não executar dada uma determinada condição (KNAGGS, 2004).

O presente trabalho procura efetuar uma melhoria no desempenho nas arquiteturas ARM, através de uma técnica em *hardware* chamada de reuso de traços (PILLA, 2007). Esta técnica tem como objetivo principal explorar redundância encontrada em sequência de instruções executadas dentro de uma aplicação – isto é feito evitando a execução de instruções redundantes, e funciona da seguinte maneira: conforme uma aplicação é executada, as entradas e saídas de um traço são armazenadas em uma tabela no *hardware*, permitindo serem analisados no decorrer da aplicação. Quando esse traço for computado novamente, suas entradas são comparadas com as que estão guardadas na tabela. Se as entradas forem iguais, isso significa que o traço pode ser reusado, sendo necessário somente capturar os valores de saída que estão armazenados na tabela para colocá-los nos devidos registradores de saída.

Logo, o objetivo deste trabalho é verificar como a reutilização de traços pode influenciar em uma arquitetura ARM. Para isso, será avaliado a quantidade de instruções redundantes nas aplicações simuladas, bem como o tamanho ideal do traço redundante.

2. METODOLOGIA

Para que o trabalho pudesse ser realizado, escolheu-se um *software* capaz de simular uma arquitetura ARM, Sim-Panalyzer (MUDGE, 2001. O simulador permite a extração de arquivos de traço que contêm todas as instruções executadas por um programa. Além disso, o Sim-Panalyzer faz análises envolvendo ganhos e perdas na relação entre consumo de energia e desempenho.

Também foi escolhido um conjunto de aplicações para executar sobre o Sim-Panalyzer a fim de gerar resultados. Escolheu-se o conjunto de *benchmarks* MiBench (GUTHAUS, 2001a), o qual é composto de 27 aplicações de diversas áreas computacionais, tais como redes, segurança e telecomunicações (GUTHAUS, 2001b). Além disso, o MiBench disponibiliza também todos os

códigos dos *benchmarks* gratuitamente na linguagem C, bem como possui arquivos binários pré-compilados para ARM.

O simulador então foi modificado com a finalidade de avaliar o comportamento da técnica de reuso de traços. Este trabalho visa então fazer uma abordagem realista do mecanismo de reuso, impondo limites no tamanho do *buffer* de reuso, o qual suportou 1024 traços. Além disso, os traços foram construídos em tempo de execução, i.e. a cada instrução executada, vai-se montando um traço para ser comparado com os que estão armazenados em tabela. Como política de substituição de traços, foi escolhida a LRU (*Least Recently Used*), visando explorar a proximidade de traços reusáveis.

Com o simulador alterado, *benchmarks* do MiBench foram simuladas na ferramenta, buscando analisar o percentual de instruções reusadas, além de categorizar os traços mais redundantes pela sua quantidade de instruções. O domínio de reuso considerado por este trabalho não inclui instruções de acesso a memória, por envolver maior complexidade e maior quantidade de valores para serem armazenados (endereços e valores de memória), além de instruções de ponto flutuante, já que nem todos *benchmarks* incluem esta categoria de instruções.

3. RESULTADOS E DISCUSSÃO

A primeira análise do trabalho preocupou-se em verificar o percentual de instruções que podem ser reaproveitadas com o reuso de traços. O gráfico da Figura 1 exibe duas porcentagens, onde a barra em preto é o percentual de redundância considerando todas as instruções executadas. Já a barra em cinza é o percentual de redundância considerando somente as instruções do domínio de reuso.

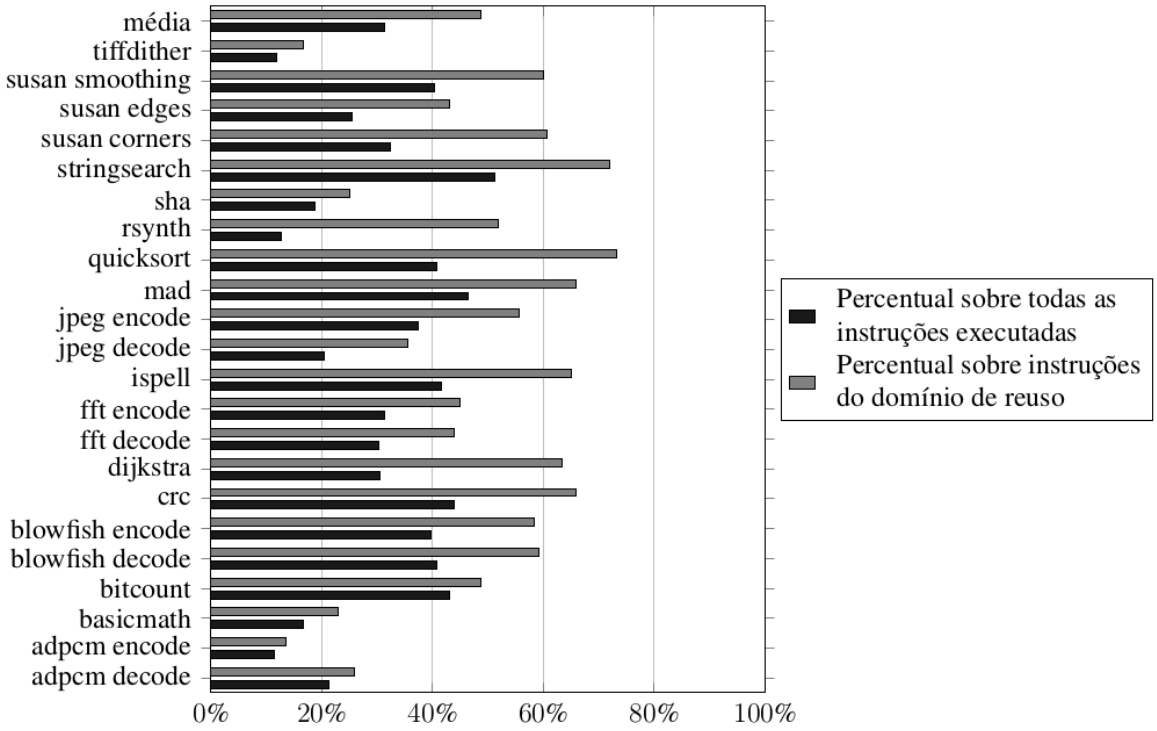

Figura 1: Percentual de instruções reusadas com os traços construídos em tempo de execução com tamanho da tabela de reuso com 1024 entradas

Os *benchmarks* simulados mostram que, mesmo limitando o tamanho do *buffer* para 1024 entradas, existe bons percentuais de redundância aproveitados pelo mecanismo de reuso. Os percentuais considerando todas as instruções e somente o domínio de reuso em média ficaram em torno dos 31% e 48%, respectivamente. A aplicação `stringsearch` obteve o melhor percentual de instruções redundantes aproveitáveis, 51% considerando todas instruções. Isto se deu porque este é um *benchmark* de rápida execução, então há bastante localidade entre os traços reusados - isto também explica os bons percentuais vistos na aplicação `mad`. Por outro lado, as aplicações `basicmath`, `tiff2dither` e as duas etapas do `adpcm`, onde as aplicações obtiveram percentuais abaixo do esperado, mostrando baixa localidade entre os traços redundantes.

A outra análise do trabalho visou estudar o tamanho médio do traço redundante nas aplicações simuladas. São mostrados os percentuais categorizados por tamanho na Figura 2.

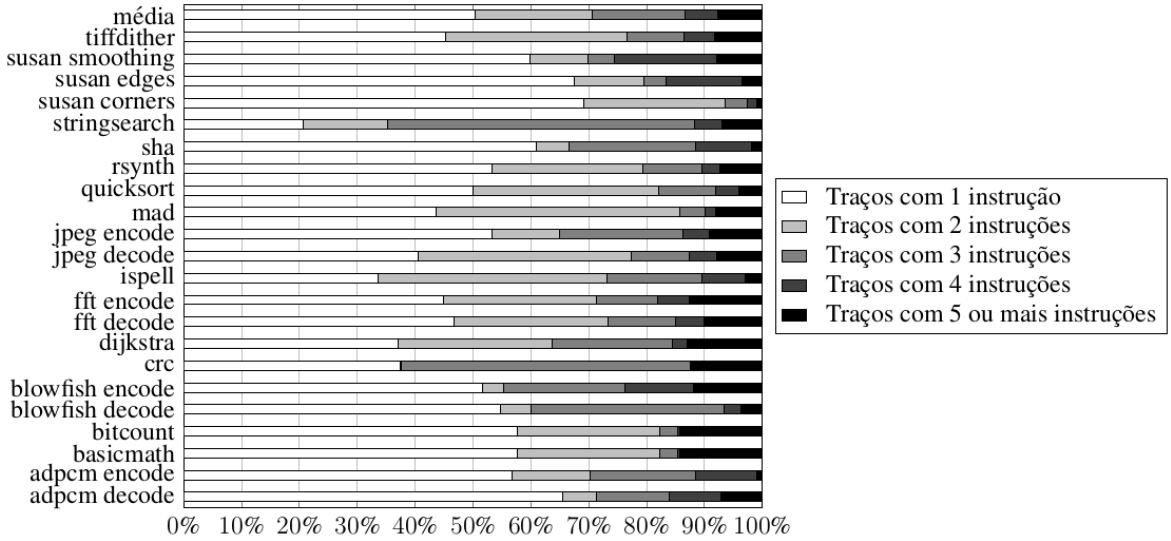

Figura 2: Percentual de traços reusados construídos em tempo de execução com tamanho da tabela de reuso com 1024 entradas

Observa-se que na maioria das aplicações, os traços com 1 instrução foram os mais reaproveitados pelo mecanismo de reuso. Isto se deve muito em função que os traços são constantemente encerrados quando há uma presença de uma instrução fora do domínio de reuso, além da variação de estados das instruções condicionais durante a execução de uma aplicação. Porém, existem boa quantidade de traços redundantes com mais de uma instrução, existindo alguns casos onde estes são maioria (*Benchmarks* `crc`, `ispell` e `stringsearch`).

Uma análise que pode ser feita é a questão que os traços redundantes dependem bastante da aplicação que é simulada, não sendo observados padrões. Isso pode ser observado nos percentuais de traços de 3 instruções, onde os mesmos são a maioria na aplicação `stringsearch`, com percentual por volta dos 53%, enquanto que no *benchmark* `basicmath` os mesmos traços correspondem a somente 3% dos traços redundantes.

4. CONCLUSÕES

O trabalho desenvolvido fez uma análise da técnica de reuso de traços em arquiteturas ARM, utilizando-se de uma abordagem realista implementando-a num simulador para verificar o seu comportamento quanto à quantidade de instruções reusadas e o tamanho médio do traço reusado.

Foi observado que os *benchmarks* possuem bons percentuais de redundância em média, mesmo limitando o tamanho do *buffer* de reuso para 1024 entradas. Algumas aplicações tiveram os percentuais abaixo do esperado, sendo estas `basicmath`, `tiffdither` e as duas etapas do `apdcm`.

O trabalho também viu que a maioria das aplicações apresentam traços com uma instrução como sendo os mais reusados, já que nem todas as instruções fazerem parte do domínio de reuso, além da presença de instruções condicionais. Mas existem também bons percentuais de traços redundantes com mais de uma instrução, o que ficou bem evidenciado nas aplicações `crc`, `ispell` e `stringsearch`.

Como trabalhos futuros, pode-se realizar mais simulações no Sim-Panalyzer modificando o tamanho do *buffer* de reuso para avaliar diferentes comportamentos nos percentuais de instruções reusadas. Além disso, é relevante calcular o *speedup* das aplicações com a técnica de reuso, a fim de avaliar se há ganhos em desempenho.

5. REFERÊNCIAS BIBLIOGRÁFICAS

GUTHAUS, M.; RINGENBERG, J.; AUSTIN, T.; MUDGE, T.; BROWN, R. **MiBench Version 1.0**. Disponível em <http://www.eecs.umich.edu/mibench/>. Acesso em: julho de 2015.

GUTHAUS, M.; RINGENBERG, J.; ERNST, D.; AUSTIN, T.; MUDGE, T.; BROWN, R. Mibench: A free, commercially representative embedded benchmark suite. In Workload Characterization. In: WWC-4. 2001 IEEE INTERNATIONAL WORKSHOP ON, 2001., 2001. **Anais. . .** Austin: Texas: IEEE, 2001. p.3–14.

KNAGGS, P.; WELSH, S. **ARM: Assembly Language Programming**. Bournemouth, England: School of Design, Engineering & Computing, Bournemouth University, 2004.

MUDGE, T.; AUSTIN, T.; GRUNWALD, D. **Sim-Panalyzer 2.0 Reference Manual**. Ann Arbor, MI, USA: Electrical Engineering and Computer Science – University of Michigan, 2001.

PILLA, M. L.; CHILDERS, B. R.; FRANCA, F. M. G.; DA COSTA, A. T. Limits for a feasible speculative trace reuse implementation. **International Journal of High Performance Systems Architecture**, Genève, Switzerland, v.1, n.1, p.69–76, 2007.

RYZHYK, L. **The ARM Architecture**. University of New South Wales, Sydney, Australia, 2006.