ANÁLISE PRELIMINAR DO REUSO DE TRAÇOS EM ARQUITETURAS ARM

GIOVANE DE O. TORRES[1*]; RODRIGO C. DE MOURA[1]; MAURÍCIO L. PILLA[1]

[1]UFPEL – Universidade Federal de Pelotas
CDTEC – Centro de Desenvolvimento Tecnológico
LUPS – Laboratory of Ubiquitous and Parallel System
{gdotorres, rcmoura, pilla}@inf.ufpel.edu.br

1. INTRODUÇÃO

Hoje há 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 objetivam possuir consumo energético reduzido sem abdicar completamente do desempenho (RYZHYK, 2006).

Esta categoria de arquiteturas ainda possui como característica a presença de instruções condicionais. Estas instruções podem ser executadas ou não dependendo de uma determinada condição, a qual é indicada no *assembly* de ARM como um sufixo da instrução. (KNAGGS; WELSH, 2004) É importante notar que, quando não há a presença de sufixo na instrução, isto indica que a mesma sempre será executada.

Este resumo visa estudar um meio para reduzir o consumo de energia nesta categoria de arquiteturas, utilizando uma técnica de *hardware* chamada de reuso de traços. Essa técnica procura explorar redundância encontrada em sequências de instruções (traços) executadas dentro de uma aplicação. O reuso de traços baseia-se na premissa de que um conjunto de instruções executado com uma determinada entrada sempre irá gerar a mesma saída. Logo, um traço pode ser executado somente uma vez, visto que depois de processado o mesmo é armazenado em um *buffer*. Quando o mesmo traço for processado, basta aproveitar os valores de saída, evitando a sua reexecução. Reduzindo a quantidade de instruções que uma aplicação executa, pode ser possível obter redução no consumo energético, além de afetar positivamente o desempenho. Uma aplicação desta técnica pode ser visto em (PILLA, 2004), onde é proposta uma arquitetura que envolve reuso de traços, além de incluir técnicas de especulação de resultados.

O presente trabalho é uma sequência de um trabalho anterior o qual teve como objetivo principal fazer uma análise preliminar da técnica de reuso de instruções na arquiteura ARM (TORRES et al., 2013). Neste caso, o artigo fará uma análise preliminar da técnica de reuso de traços na arquitetura ARM, verificando a viabilidade desse método nessa categoria de arquiteturas.

2. METODOLOGIA

Para que o trabalho pudesse ser realizado, foi necessário efetuar a escolha de um simulador de arquiteturas ARM e um conjunto de *benchmarks* para executar sobre o simulador. A ferramenta escolhida para simular arquitetura ARM foi o Sim-Panalyzer, o qual tem o objetivo de permitir análises entre ganhos e perdas na relação entre consumo de energia e desempenho (MUDGE et al., 2001). O *software* escolhido ainda permite a extração de *tracefiles* das aplicações executadas, o que é essencial para a realização deste trabalho. *Tracefiles* são arquivos gerados pelo Sim-Panalyzer que exibem todas as instruções executadas em *assembly* de ARM.

*Bolsista PBIP UFPel

Já o conjunto de *benchmarks* escolhido foi o MiBench (GUTHAUS et al., 2001a), o qual engloba 27 aplicações de diversas áreas computacionais, as quais são: Controle industrial e automotivo, dispositivos de consumo, automação de serviços, redes, segurança e telecomunicações (GUTHAUS et al., 2001b). Todos os *benchmarks* possuem código-fonte disponível livremente em C, além de arquivos binários pré-compilados para arquiteturas ARM.

Escolhidas as ferramentas, foram simuladas 4 aplicações sobre o Sim-Panalyzer até o presente momento, gerando os *tracefiles* para posterior análise. Somente estes *benchmarks* foram executados devido ao volume de dados gerado por eles que foi menor comparado às demais aplicações. Segue abaixo as aplicações que foram simuladas:

- *jpeg*: Algoritmo para compressão de imagens com perdas na qualidade;
- *mad*: Decodificador de alta qualidade de áudio em formato MPEG;
- *pgp*: Algoritmo de criptografia de chave pública;
- *stringsearch*: Aplicação que busca palavras utilizando um algoritmo de comparação *case insensitive*;

Com os *tracefiles* gerados, foi desenvolvido um *script* para buscar traços redundantes. A cada traço processado, o mesmo é inserido em uma tabela o qual permanece até o final da execução do *script* para comparação com outros traços gerados. Todas as instruções do conjunto do *assembly* de ARM foram consideradas na formação de um traço, exceto as instruções condicionais (discutidas na Seção 1) que não foram executadas, i.e. não geraram nenhum valor de saída. Para que fosse possível efetuar as comparações entre traços, é necessário guardar uma série de informações, que são:

- Mnemônico das instruções;
- Identificadores e conteúdos dos registradores usados como entrada;
- Identificadores dos registradores utilizados como saída;
- Endereços de memória e os valores armazenados nos mesmos usados como entrada (Se o traço possui instruções de acesso à memória);
- Endereços de memória de saída (No caso de o traço ter instruções de armazenamento em memória).

3. RESULTADOS E DISCUSSÃO

Os primeiros resultados mostram a quantidade de instruções que são redundantes com a verificação de traços redundantes. O gráfico da Figura 1 exibe que há redundância nos *benchmarks* simulados. O eixo vertical mostra o percentual de instruções que seriam reusadas, o qual foi calculado a partir da divisão entre o número de instruções redundantes com o total de instruções executadas. A quantidade de instruções redundantes foi contada após a verificação dos traços redundantes. Ao verificarmos que um traço com *x* instruções obteve *y* ocorrências, o número de instruções redudantes deste traço é de *x*y*.

Alguns *benchmarks* possui um par de resultados, visto que os mesmos possuem duas etapas, uma para codificação e outra para decodificação. Nos gráficos as palavras *encode* (codificação) e *decode* (decodificação) aparecem após os nomes dos *benchmarks*.

*Bolsista PBIP UFPel

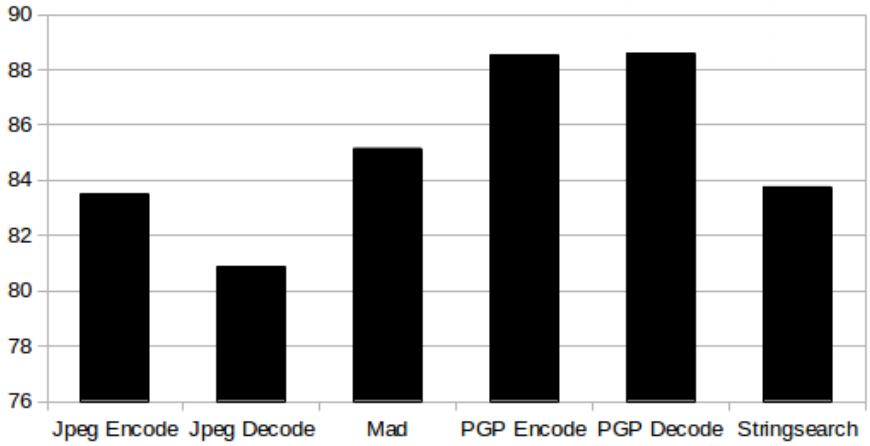

Figura 1: Percentual de instruções redudantes das aplicações simuladas

Os *benchmarks* simulados até o momento apresentam bons percentuais de instruções redundantes, sendo que os mesmos estão concentrados entre 80% e 90% do total de instruções. É importante observar que estes percentuais consideram que a tabela a qual armazena os traços possui tamanho ilimitado, visto que um traço é comparado com todos. Logo, pode-se ponderar que várias instruções redundantes não seriam reutilizadas de fato se fosse implementado a técnica de reuso de traços. Por outro lado, com a existência de um grande número de instruções redundantes, pode ser vantajoso explorar esta redundância com a implementação do reuso de traços, reduzindo a quantidade de instruções executadas.

Outra abordagem do trabalho diz respeito ao tamanho do traço redundante que é encontrado nas aplicações simuladas. O gráfico da Figura 2 exibe o percentual dos traços reutilizados com determinadas quantidades de instruções.

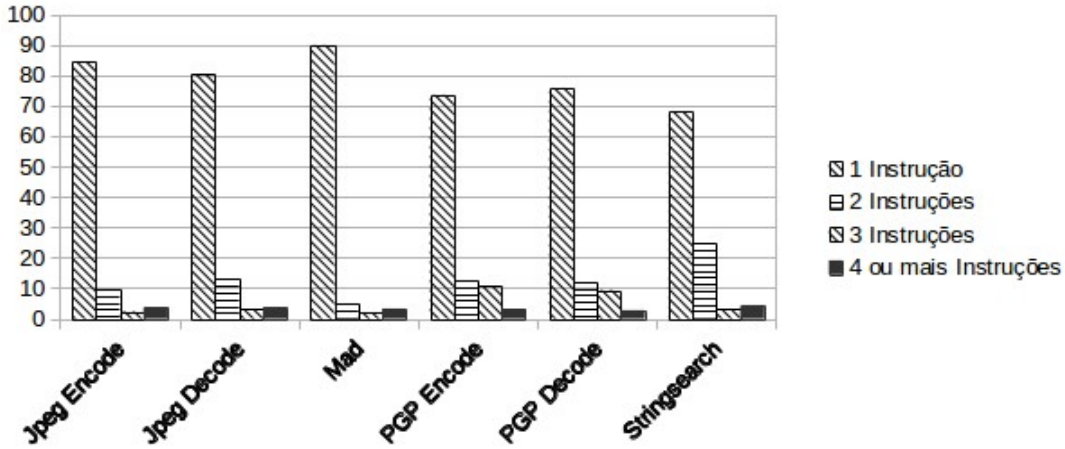

Figura 2: Percentual de traços reusados definido pelo número de instruções

Em todas as aplicações verificou-se que a maior quantidade de traços redundantes possui somente uma instrução, chegando a atingir 90% dos traços reusados na aplicação *mad*. Além disso, quando se aumenta o número de instruções dentro de um traço, menor torna-se a probabilidade do mesmo ser reusado. Isto é exibido nos resultados nas diferenças existentes entre os traços reusados de 1 e 2 instruções, assim como entre os de 2 e 3 instruções.

Isto indica que nestas aplicações não torna-se viável considerar traços com muitas instruções, visto que poucos seriam reutilizados, enquanto que traços com poucas instruções tenderiam a ser mais reusados.

*Bolsista PBIP UFPel

4. CONCLUSÕES

Os resultados referentes ao número de instruções redundantes mostraram-se bons, visto que todas as aplicações tiveram acima de 80% de código redundante. Isto sinaliza que a implementação da técnica de reuso de traços pode obter bons resultados, ainda que deva existir um limite no tamanho da tabela que guarda os traços.

Outra conclusão preliminar com a análise dos resultados mostrada no trabalho refere-se ao padrão no tamanho do traço reutilizado dos *benchmarks* simulados. A maioria dos traços reusados possuía entre 1 e 3 instruções, sendo que os traços com instrução única foram os mais reutilizados. Logo, na formação dos traços que podem ser reutilizados não é viável compô-los com muitas instruções, visto que uma pequena quantidade destes será reusada.

Como trabalhos futuros, pretende-se simular o restante das aplicações para verificar se os padrões encontrados nestes *benchmarks* é mantido. Além disso, outro trabalho futuro é utilizar a tabela de traços com uma política para armazenamento dos mesmos a fim de realizar uma análise mais realista na quantidade de traços reutilizados.

5. REFERÊNCIAS BIBLIOGRÁFICAS

RHYZHYK, L. **The ARM Architecture**. The University of South Wales, 2006. Acessado em 24 jul. 2014. Disponível em: http://www.cse.unsw.edu.au/~cs9244/06/seminars/08-leonidr.pdf

KNAGGS, P., WELSH, S. **ARM: Assembly Language Programming**. Bournemouth, Dorset, England: Bournemouth University, 2004.

TORRES, G., MOURA, R., PILLA, M. Estimativa de desempenho utilizando reuso de instruções em arquiteturas ARM. **XIV Simpósio em Sistemas Computacionais**. Porto de Galinhas, Pernambuco., p.218-221, 2013.

PILLA, M. **RST: Reuse through Speculation on Traces**. Junho de 2004. Tese (Doutorado em Ciência da Computação) – Programa de Pós-Graduação em Computação, Universidade Federal do Rio Grande do Sul.

MUDGE, T., AUSTIN, T., GRUNWALD, T. **The SimpleScalar ARM Power Modeling Project**. University of Michigan, Electrical Engineering and Computer Science Department, 2001. Acessado em 29 jul. 2013. Disponível em: http://web.eecs.umich.edu/~sim-panalyzer/

GUTHAUS, M., RINGENBERG, J., AUSTIN, T., MUDGE, T., BROWN, R. **MiBench Version 1.0**. University of Michigan, Electrical Engineering and Computer Science Department, 2001. Acessado em 24 jul. 2014. Disponível em: http://www.eecs.umich.edu/mibench/

GUTHAUS, M., RINGENBERG, J., ERNST, D., AUSTIN, T., MUDGE, T., BROWN, R. MiBench: A free, commercially representative embedded benchmark suite. **2001 IEEE International Workshop on Workload Characterization**. Austin, Texas., p.3-14, 2001.

*Bolsista PBIP UFPel