Análise parcial preliminar do reuso de traços em arquiteturas ARM

Giovane de Oliveira Torres[1], **Rodrigo Costa de Moura**[1], **Maurício Lima Pilla**[1]

[1]Centro de Desenvolvimento Tecnológico – Universidade Federal de Pelotas (UFPel)
Rua Gomes Carneiro, 1 – 96.010-610 – Pelotas – RS – Brazil

{gdotorres,rcmoura,pilla}@inf.ufpel.br

Resumo. *O artigo efetua uma análise preliminar dos efeitos da técnica de reuso de traços sobre uma arquitetura ARM, a fim de verificar possíveis melhorias nas questões de desmpenho e consumo de energia. Para isto, utiliza-se um conjunto de benchmarks para executar sobre um simulador de arquiteturas ARM, buscando assim redundância nas instruções executadas em cada aplicação simulada.*

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 e Welsh 2004].

O presente trabalho procura efetuar uma melhoria no desempenho e consumo de energia nas arquiteturas ARM, através de uma técnica em *hardware* chamada de reuso de traços. Esta técnica tem como objetivo principal explorar redundância encontrada em sequência de instruções executadas dentro de uma aplicação. Logo, o objetivo deste trabalho é verificar como a reutilização de traços pode influenciar em uma arquitetura ARM, avaliando quantos dos traços executados nas aplicações simuladas podem ser reusados.

Os resultados deste trabalho mostram que existe bastante redundância dentro das aplicações simuladas, com percentual de instruções redundantes entre 80% e 90%. Além disso, a maioria dos traços reusados é composto de poucas instruções, sendo que em todos os *benchmarks* a maioria dos traços possui somente uma instrução.

O artigo é uma extensão de um trabalho desenvolvido anteriormente, o qual fez uma análise parcial da técnica de reuso de instruções na arquitetura ARM [Torres et al. 2013]. No trabalho atual, foi considerado todas as instruções do conjunto de ARM, além de expandir o reuso para conjuntos de instruções, não se restringindo somente à reutilização de instruções simples.

2. Reuso de traços

Esta técnica consiste no reuso de sequências dinâmicas de instruções (traços), os quais possuem um contexto de entrada e saída. Este mecanismo possui o objetivo de evitar a reexecução de um conjunto de instruções que estão dentro de um traço [Pilla et al. 2007].

O funcionamento do mecanismo de reuso de traços é caracterizado da seguinte forma: 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.

Com isso, é possível verificar algumas vantagens ao reusar traços. Uma das vantagens é a economia de recursos, onde alguns dos estágios do *pipeline* não são utilizados, o que pode acarretar em ganho tanto em desempenho como redução de energia [Tsai e Chen 2011] Isso ocorre porque a técnica de reuso pode ser implementada em paralelo ao *pipeline*, já que se não houver reuso basta simplesmente seguir o fluxo normal do *pipeline*. Além disso, pode ser destacado o fato de que os resultados podem ser gerados mais rapidamente, bem como a redução das dependências de dados, visto que instruções dependentes podem ser executadas em paralelo.

3. Metodologia

Para que o trabalho pudesse ser realizado, escolheu-se um *software* capaz de simular uma arquitetura ARM, sendo este o Sim-Panalyzer [Mudge et al. 2001]. Isto porque o simulador permite a extração de *tracefiles*, que são arquivos os quais contém todas as instruções do *assembly* de ARM 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 [Ringenberg et al. 2001], o qual é composto de 27 aplicações de diversas áreas computacionais, tais como redes, segurança e telecomunicações [Guthaus et al. 2001]. Além disso, o MiBench disponibiliza também todos os códigos dos *benchmarks* gratuitamente na linguagem C, bem como possui arquivos binários pré-compilados para ARM.

Com as ferramentas escolhidas, quatro aplicações do MiBench foram simuladas no Sim-Panalyzer: `jpeg`, `mad`, `pgp` e `stringSearch`, visto que foram as únicas aplicações que puderam ser analisadas até o presente momento. Isso se deu pelo volume de dados dos *tracefiles* gerados por estas aplicações que foi menor comparado às restantes. Para analisar os *tracefiles*, fez-se um *script* que buscou traços redundantes. A cada traço analisado, este é armazenado em uma tabela a qual fica disponível para comparação com outros traços, verificando assim a existência de traços redundantes. Verificou-se então qual o percentual de instruções reutilizáveis empregando a técnica de reuso de traços nas aplicações simuladas, além de analisar os tamanhos dos traços reutilizáveis.

4. Resultados

O gráfico mostrado na Figura 1 mostra a existência de redundância nas simulações das aplicações simuladas. O eixo vertical mostra o percentual das instruções que foram reusadas com reuso de traços, o qual foi calculado a partir da divisão da quantidade de instruções reutilizadas pelo total de instruções executadas, incluindo instruções fora do domínio de reuso. As aplicações `jpeg` e `pgp` tiveram dois resultados porque os *benchmarks* têm duas etapas, sendo uma de codificação e outra de decodificação, as quais são

indicadas no gráfico da Figura 1, como *encode* e *decode*, respectivamente.

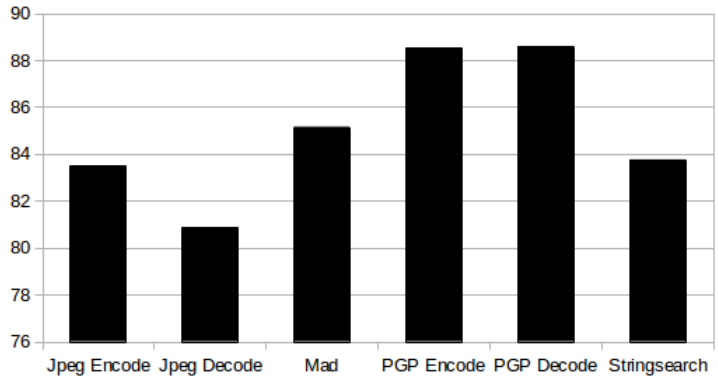

Figura 1. Percentual de redundância encontrado nos *benchmarks* simulados

Os *benchmarks* simulados até o momento apresentam bons percentuais de instruções redundantes, com valores entre 80% e 90% do total de instruções – esses valores aumentaram significamente comparado com o trabalho anterior de reuso de instruções, o qual obteve percentuais não maiores que 20%, e espera-se que o alto nível de redundância mantenha-se nos demais *benchmarks* que serão simulados. 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.

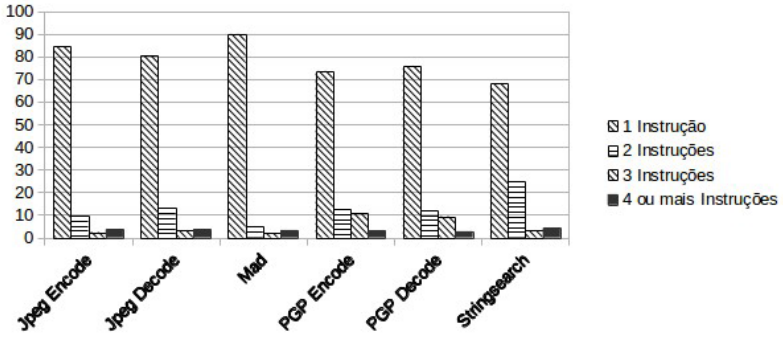

Figura 2. Percentual de redundância encontrado nos *benchmarks* simulados

Como pode ser visto, em todas as simulações a maioria dos traços reutilizados são formados por uma única instrução, onde a diferença entre a quantidade de traços com uma instrução e o número de traços compostos por múltiplas instruções é grande, a qual é bem evidenciada na aplicação `stringsearch`. Também pode-se afirmar que a medida que o traço vai aumentando de tamanho, menores são as chances do mesmo ser reutilizado.

5. Conclusões e trabalhos futuros

Os resultados apresentados indicam que a implantação da técnica de reuso de traços em arquiteturas ARM pode trazer bons resultados, já que foi encontrada uma boa quantidade de conjuntos de instruções redundantes ao longo da execução das aplicações (Acima de 80%). Porém, é importante ressaltar que muitas dessas instruções, as quais compõem traços, não serão de fato reutilizadas. Isto porque o *buffer* de armazenamento de traços deverá possuir uma determinada capacidade a qual limitará a reutilização dos mesmos.

Outra conclusão atingida com o desenvolvimento deste trabalho reside no tamanho dos traços. Conforme foi visto em trabalhos anteriores, a maior parte dos traços reutilizados é composto de poucas instruções, o que pôde ser visualizado nas aplicações simuladas no presente trabalho.

Como trabalhos futuros, pretende-se simular todas as aplicações do *benchmark* MiBench para verificar se os padrões encontrados nas aplicações simuladas é mantido. Além disso, intenciona-se impor um limite físico para o *buffer* de traços utilizando-se de alguma política para armazenamento dos mesmos, o que traria uma análise mais realista do potencial da técnica de reuso de traços, permitindo avaliar questões como *speedup* e consumo de energia. Por fim, em caso de maiores conclusões com aspecto postivo, torna-se fundamental a implementação dessa técnica para verificar se os bons resultados analisados teoricamente também são confirmados na prática.

Referências

Guthaus, M. R., Ringenberg, J. S., Ernst, D., Austin, T. M., Mudge, T., e Brown, R. B. (2001). Mibench: A free, commercially representative embedded benchmark suite. In *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, pages 3–14. IEEE.

Knaggs, P. e Welsh, S. (2004). Arm: Assembly language programming. *School of Design, Engineering & Computing, Bournemouth University*.

Mudge, T., Austin, T., e Grunwald, D. (2001). The simplescalar-arm power modeling project. Disponível em `<http://web.eecs.umich.edu/~panalyzer/>`. Acesso em: julho de 2013.

Pilla, M. L., Childers, B. R., Franca, F. M. G., e Da Costa, A. T. (2007). Limits for a feasible speculative trace reuse implementation. *International Journal of High Performance Systems Architecture*, 1(1):69–76.

Ringenberg, J., Guthaus, M., Austin, T., Mudge, T., e Brown, R. (2001). Mibench version 1.0. Disponível em `<http://www.eecs.umich.edu/mibench/>`. Acesso em: novembro de 2014.

Ryzhyk, L. (2006). The arm architecture. *University of New South Wales*.

Torres, G., Moura, R., e Pilla, M. (2013). Estimativa de desempenho utilizando reuso de instruções em arquiteturas arm. *Workshop de Iniciação Científica (WSCAD-WIC). Anais do XIV Simpósio em Sistemas Computacionais*, pages 218–221.

Tsai, Y.-Y. e Chen, C.-H. (2011). Energy-efficient trace reuse cache for embedded processors. *IEEE Transactions on Very Large Scale Integration (VLSI)*, 19(9):1681–1694.