REUSO DE VALORES EM ARQUITETURAS ARM

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

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

1. INTRODUÇÃO

Atualmente, existe uma grande difusão de dispositivos portáteis os quais tem alta demanda por melhor desempenho. Muitos destes equipamentos possuem a arquitetura ARM em seu *hardware*, já que a mesma é popular no mercado atual de arquiteturas embarcadas (RYZHYK, 2006). Esta popularidade deve-se à sua principal característica: obter um bom desempenho com baixo consumo de energia. Existem também outras particularidades importantes da arquitetura ARM que a tornam popular:

- Um *core* ARM é mais simples se comparado a um de utilização geral. Isto implica em menor quantidade de transistores em um processador ARM, permitindo a inserção de mais componentes voltados para aplicações específicas;
- A arquitetura ARM é altamente modular. Somente o *pipeline* de inteiros é um componente obrigatório dentro de uma processador ARM. Assim, é possível construir processadores baseados em ARM com bastante flexibilidade.

Este trabalho foca na demanda por melhor desempenho dentro desta categoria de arquiteturas. Isto pode ser solucionado através de técnicas de reuso de código, como por exemplo a reutilização de instruções de um programa. Este método parte do princípio de que programas são constituídos em grande parte por computação redundante, logo pode ser reaproveitada. (PILLA, 2004) Na técnica de reuso de instruções, antes de executar uma instrução, seu contexto de entrada é comparado com os das execuções anteriores. Se os contextos forem iguais, isso signfiica que a instrução pode ser reaproveitada, sem a necessidade de executá-la.

Logo, o principal benefício que pode ser obtido da técnica de reuso de instruções é a não-execução de instruções redundantes dentro de uma aplicação e o consequente colapso de cadeias de dependências de dados em um único ciclo. Isto acarreta em menos trabalho para o processador e, consequentemente, diminui o consumo de energia.

Este artigo apresenta uma estimativa de possíveis ganhos em desempenho da técnica de reutilização de instruções em uma arquitetura ARM.

2. METODOLOGIA

Para que fosse possível extrair qualquer tipo de resultado, foi necessário utilizar um simulador para arquiteturas ARM. A ferramenta escolhida foi o Sim-Panalyzer (MUDGE; AUSTIN; GRUNWALD, 2001), já que ele possui facilidades quanto à medição de desempenho e apresenta o objetivo principal de criar um estimador adiantado de potência, permitindo avaliar ganhos e perdas na relação

FAPERGS/PqG (11/1065-1), PRONEX/FAPERGS/CNPq (10/0042-8)
[1]Bolsista FAPERGS/PROBIC

entre desempenho e dissipação de energia. Além disso, também é possível realizar a extração de *tracefiles* (arquivo que tem o registro de todos os estados internos de um programa) de aplicações executadas sobre o simulador.

Da mesma forma, foi essencial escolher um conjunto de programas para executar sobre o simulador e retirar resultados. Para isto, foi escolhido o conjunto de *benchmarks* (Aplicações desenvolvidas para simular programas reais com a finalidade de avaliar desempenho) MiBench (GUTHAUS et al., 2001). Este conjunto de *benchmarks* foi escolhido devido ao mesmo conter um bom número de aplicações comercialmente representativas de diversas áreas computacionais (Controle industrial, redes, segurança, automação, dentre outros) e também pelo fato do código-fonte de todos os programas possuir distribuição livre.

Para o trabalho ser realizado, foram executados os *benchmarks* do MiBench no Sim-Panalyzer. Com isto, é possível extrair os *tracefiles* dos programas executados, o que permite a análise das instruções executadas de cada aplicação. Os arquivos gerados pelo simulador devem ser verificados com a finalidade de procurar redundância nas instruções executadas, ou seja, buscar instruções que possuem os mesmos registradores de entrada, bem como o valor que está no registrador PC. Tendo estas informações, faz-se uma estimativa de quantas instruções poderiam ser reexecutadas em uma aplicação, o que é explorado por este trabalho.

Para que uma determinada instrução, com um valor de PC e registradores de entrada (são os utilizados pela instrução para gerar uma resultado em um registrador de saída) seja reexecutada, é necessário que esta instrução tenha sido executada uma vez, o que implica em redundância. Ao verificar um *tracefile* de uma aplicação, buscouse quantas vezes uma instrução foi reexecutada, além de contar quantas instruções foram executadas ao total.
Cruzando estes dados, foi possível estimar qual o potencial de reuso de instruções.

Foram desconsideradas instruções que envolvessem a utilização de memória e desvio , como por exemplo, *load*, *store*, *branch*, já que este gênero de instruções requerem um teste mais complexo para verificar a possibilidade de reuso das mesmas. O restante das instruções foi considerado neste trabalho, tais como instruções aritméticas (*add*, *sub*), lógicas (*and*, *or*) e de manipulação de dados (*mov*).

3. RESULTADOS E DISCUSSÃO

A Figura 1 mostra que a redundância existe nas aplicações simuladas. A porcentagem que é mostrada no gráfico foi calculada a partir da divisão entre o total de instruções executadas e a quantidade de instruções redundantes encontradas nos *tracefiles*. Ambas as quantidades de instruções somente contém instruções dentro do subconjunto estudado.

É importante ressaltar que, algumas aplicações do MiBench tiveram que ser simuladas duas vezes, por se tratarem de *benchmarks* que possuem etapas de codificação e decodificação. No gráfico da Figura 1, isto pode ser visto quando a aplicação está seguida da palavra *encode* (etapa de codificação) ou *decode* (etapa de decodificação).

FAPERGS/PqG (11/1065-1), PRONEX/FAPERGS/CNPq (10/0042-8)
[1]Bolsista FAPERGS/PROBIC

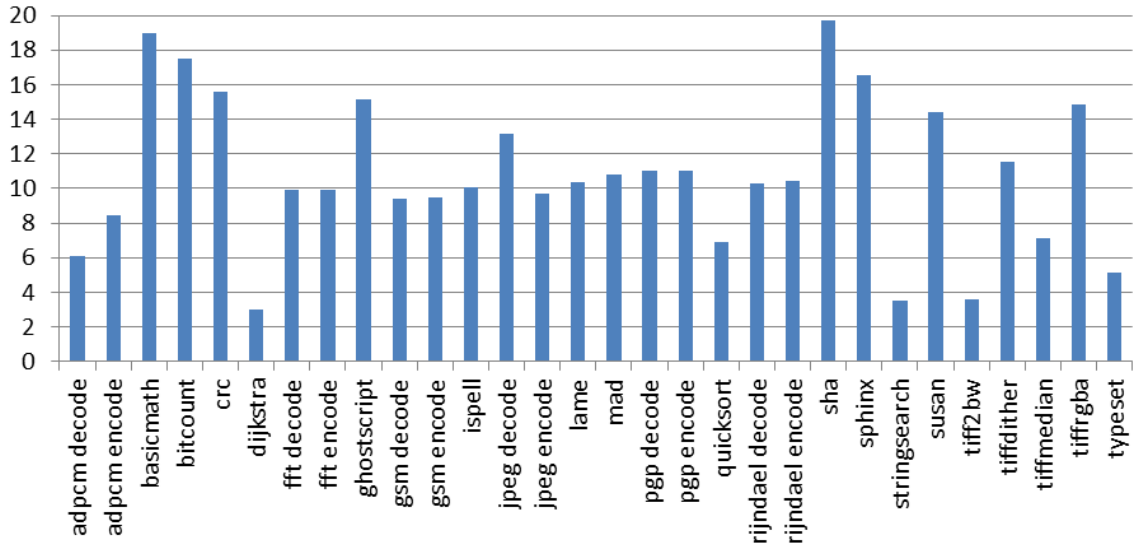

Figura 1: Percentual de redundância encontrada nos *benchmarks.*

Com os resultados obtidos, é possivel verificar um padrão em grande parte dos *benchmarks* simulados, os quais obtiveram percentuais de redundância razoável (Entre 8% e 12%), como pode ser visto, por exemplo, nas aplicações *fft*, *gsm* e *ispell*.

O restante dos *benchmarks* pode ser separado em dois grupos. O primeiro grupo engloba aplicações as quais obtiveram as melhores porcentagens de redundância. Como exemplos podem ser citados os *benchmarks bitcount*, *ghostscript*, *typeset, susan* e *tiffrgba*, os quais obtiveram redundância superior à 14%. Dentro deste grupo ainda pode-se destacar as aplicações que foram as únicas a ultrapassarem o percentual de 18% de redundância de instruções: *basicmath* e *sha*. Esta última obteve o melhor resultado encontrado nas simulações, chegando a quase 20%.

Em contrapartida, o outro grupo das aplicações obtiveram baixa quantidade de redundância. Neste grupo estão incluídos a etapa *decode* do *adpcm* e os *benchmarks dijkstra*, *quicksort, stringsearch*, *tiffmedian, tiff2bw* e *typeset*. O pior resultado encontrado foi no *dijkstra*, o qual obteve percentual de redundância de 3,3%. Estes *benchmarks* provavelmente obtiveram baixa redundância devido ao tipo de percurso que estas aplicações realizam em valores de entrada menos redundantes.

4. CONCLUSÕES

Os resultados mostrados neste artigo ratificam o princípio em que o trabalho corrente baseia-se – a existência de redundância nas aplicações. Porém, existem três divisões dentro do conjunto de *benchmarks* simulados. A primeira foi a existência de bons resultados em diversas aplicações, como visto nas aplicações *sha* e *basicmath*. A segunda divisão é a que abrange a maior quantidade de *benchmarks*, e obteve resultados razoáveis, com média de percentual de redundância por volta de 10%. O restante das aplicações simuladas obteve resultados com porcentagens inferiores à 8%, como pôde ser visto nos resultados do *dijkstra* e *typeset*. Estas aplicações obtiveram resultados abaixo do esperado, especialemente devido à simulações realizadas anteriormente com o conjunto de

FAPERGS/PqG (11/1065-1), PRONEX/FAPERGS/CNPq (10/0042-8)
[1]Bolsista FAPERGS/PROBIC

benchmarks SPEC CPU 2000 obtiveram melhores resultados (SODANI; SOHI, 1997).

Os baixos percentuais podem ser explicados devido à natureza dos *benchmarks* os quais fazem parte do MiBench, não contendo muita redundância. Outro fator possível para a explicação pode ser o subconjunto de instruções avaliado no artigo, o que significa que outros tipos de instruções, como instruções de acesso a memória e/ou desvio apresentem maior redundância. Além destes dois fatores, ainda é possível que a maneira com que os códigos das aplicações do MiBench são traduzidos para linguagem *assembly* de ARM influencie na redundância dos programas.

Como trabalhos futuros, pretende-se incrementar as informações coletadas. Para isto, mais aplicações do MiBench serão simuladas, com a finalidade de verificar se o padrão dos benchmarks é mantido. Depois, os *benchmarks* do MiBench serão recompilados para a arquitetura MIPS. Por fim, o MiBench será executado em uma arquitetura MIPS e será verificado se as aplicações possuem o mesmo comportamento relacionado à sua redundância, comparando com os resultados obtidos.

Além disso, o autor pretende expandir o conjunto de instruções estudado, verificando redundância em todo o tipo de instrução que é executada, para analisar se este fator influencia na redundância dos *benchmarks.*

5. REFERÊNCIAS BIBLIOGRÁFICAS

RYZHYK, L. **The ARM Architecture**. School of Computer Science and Engineering, The University of New South Wales. Acessado em 10 out. 2013. Disponível em: http://www.cse.unsw.edu.au/~cs9244/06/seminars/08-leonidr.pdf

PILLA, M. **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.** Electrical Engineering and Computer Science, University of Michigan. Acessado em 10 set. 2013. 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.** Electrical Engineering and Computer Science, University of Michigan. Acessado em 10 out. 2013. Disponível em: http://www.eecs.umich.edu/mibench/

SODANI, A., SOHI, G. Dynamic Instruction Reuse. **SIGARCH Comput. Archit. News.** New York, NY, USA. v.25, n.2. p.194-205, 1997.

FAPERGS/PqG (11/1065-1), PRONEX/FAPERGS/CNPq (10/0042-8)
[1]Bolsista FAPERGS/PROBIC