Uma Análise da Facilidade de Emulação de Binários RISC-V

Leandro Lupori[1], **Vanderson Martins do Rosario**[1], **Edson Borin**[1]

[1]Instituto de Computação – UNICAMP Campinas – SP – Brasil

leandro.lupori@gmail.com, {vanderson.rosario, edson}@ic.unicamp.br

Resumo. *Cada vez mais a heterogeneidade nos ambientes como a nuvem, dispositivos móveis e Internet das Coisas, tem tornado importante a pesquisa em tradução de binários para ISAs diferentes. Grande parte das sobrecargas computacionais vem da qualidade da tradução e estão ligados com a ISA sendo traduzida. O OpenISA já se demonstrou ser fácil de emular e obteve bons desempenhos de desempenho. Nesse trabalho apresentamos resultados que indicam que o RISC-V também pode ser emulável com baixa sobrecarga de desempenho, menos de 30%.*

1. Introdução

RISC-V é uma arquitetura de conjunto de instruções, ou ISA[1], livre e aberto, inicialmente desenvolvido na divisão de Ciência da Computação do departamento EECS da Universidade da Califórnia, Berkeley. O RISC-V tem o objetivo de ser pequeno, rápido, eficiente e implementável em processadores reais, seguindo a filosofia RISC [Waterman et al. 2014]. Hoje, o projeto é mantido pela fundação *RISC-V Foundation* [RISCV.org 2018], com participação e suporte de diversas empresas como AMD, Google, HP, IBM, Nvidia, Qualcomm, Microsoft, Western Digital, entre outros.

Uma ISA define a interface entre a pilha de *software* e a pilha de *hardware*, de forma que programas compilados para uma ISA não podem ser executados em outras, se não por meio de emulação [Smith and Nair 2005]. Portanto, com o surgimento de novas ISAs, como o RISC-V, e com a heterogeneidade entre a computação nas nuvens, móvel e da Internet das Coisas, a pesquisa de melhores técnicas de emulação de ISAs para a redução do custo do desenvolvimento e implantação de *software* tem se tornado cada vez mais importante [Zhang et al. 2015]. Principalmente porque, apesar da emulação de programas na mesma ISA terem sobrecarga (empecilho na obtenção de desempenho igual ao nativo) pequena, como demonstrado pelo StarDBT da Intel [Borin and Wu 2009], a emulação de diferentes ISAs ainda possui um tempo de execução maior que duas vezes o da execução nativa mesmo com a aplicação de técnicas sofisticadas como a tradução dinâmica de binários [Zhang et al. 2015, Hong et al. 2012]. Essa diferença de sobrecarga nos dois cenários demonstra a importância da escolha da ISA no desempenho da emulação e implica que grande parte da sobrecarga vem da qualidade da tradução. Além disso, atualmente, as duas ISAs mais comuns do mercado, x86 e ARM, possuem características que dificultam a emulação [Borin and Wu 2009, Shen et al. 2012]. Tendo tudo isso em vista, o OpenISA foi introduzido recentemente como uma ISA desenvolvida com o objetivo de ser fácil e eficientemente emulável [Auler and Borin 2017]. De fato, os resultados apresentados por Auler e Borin indicam que é possível traduzir código OpenISA para ARM e x86 e obter um código altamente eficiente, com menos de 20% de sobrecarga.

[1]do inglês: *Instruction Set Architecture*

Devido à crescente importância do RISC-V e do interesse em emulação de aplicativos em ambientes heterogêneos, nesse trabalho buscamos entender o quão difícil é a emulação de binários RISC-V e, para isso, comparamos as suas características com as do OpenISA e apresentamos resultados preliminares de qualidade de código obtidas por meio de experimentos com nosso Tradutor Estático de Binários (TEB) de RISC-V para x86 nos *benchmarks* do *MiBench* [Guthaus et al. 2001].

2. Emulação de ISAs

Existem dois métodos bastante utilizados na implementação de um emulador, a interpretação e a tradução dinâmica de binários. Ambas permitem emular código compilado para uma ISA convidada (do inglês, "*guest*") em um sistema com uma ISA diferente, o anfitrião (do inglês, "*host*") [Smith and Nair 2005]. A interpretação é uma técnica que se baseia no laço busca-decodificação-execução para imitar o comportamento de um processador. A técnica é simples e permite a implementação de emuladores compactos e bastante portáteis. Contudo, normalmente a interpretação utiliza centenas de instruções nativas para emular uma instrução da ISA convidada.

Por outro lado, a tradução dinâmica de binários traduz e mapeia regiões de código do binário convidado para um código equivalente e nativo para a ISA do anfitrião. O código traduzido é guardado em uma *cache* para ser reutilizado posteriormente. A qualidade do código produzido por um tradutor dinâmico de binários (TDB) varia. No entanto, seu desempenho é normalmente muito melhor do que a obtida por interpretadores e em alguns casos chega próximo à execução nativa do binário compilado diretamente para a ISA do anfitrião, como mostra a análise do StarDBT apresentado por Wu e Borin [Borin and Wu 2009], que demonstra também que a maior parte da sobrecarga está ligada à qualidade do código traduzido. Muitos tradutores dinâmicos aliam a interpretação em código frio e a tradução em código quente para obtenção de melhor desempenho. Contudo, apesar das vantagens de um TDB, o tempo de execução da tradução dinâmica inclui o tempo de compilação (tradução) das regiões e, assim, é importante que todo código traduzido seja frequentemente executado (código quente) para que os benefícios introduzidos pela tradução sejam maiores do que o custo da tradução por si só.

Por fim, também é possível emular uma ISA traduzindo, de uma só vez, todo o binário de forma estática, conhecido como tradução estática. Esse método possui diversas limitações na emulação de aplicações reais, devido à dificuldade de se traduzir determinadas construções de código sem executar o mesmo, diferentemente da tradução dinâmica. Entretanto, no caso de um TDB existem três fatores que podem afetar a qualidade do código gerado: *a técnica de seleção de regiões quentes*; *o compilador, incluindo as otimizações*; e *as características da ISA convidada*. Como nosso objetivo se concentra na análise da terceira característica, um Tradutor Estático de Binários (TEB) facilita esta análise ao separar a etapa de tradução da execução do binário. Para isso, escolhemos a compilação por método e o LLVM 7.0, pois consideramos serem as melhores para mitigar o impacto da tradução, na análise das características da ISA convidada. Ainda, implementamos o TEB com duas opções de contexto (valores dos registradores emulados da ISA convidada): manter dentro das funções como variáveis locais, ou como variáveis globais.

3. RISCV vs OpenISA

Para compararmos com o OpenISA, escolhemos o RISC-V 32 MFD, que é a versão 32 bits do RISC-V (OpenISA também é 32 bits), composta pelo conjunto de instruções da

base inteira I (obrigatórias), extensões padrão M (multiplicação e divisão), F e D (operações com ponto-flutuante de precisão simples e dupla), por serem os módulos mais usados e maduros. No trabalho do OpenISA [Auler and Borin 2017] são apresentadas diversas características que podem facilitar ou dificultar a emulação de uma ISA. Uma importante característica citada por Auler e Borin que dificulta a tradução do binário é a presença de um registrador de *status*. Apesar do OpenISA não conter um registrador de *status*, o RISC-V contêm, mas este é utilizado apenas por instruções de ponto-flutuante no modo de arredondamento dinâmico. Uma característica negativa no RISC-V comparado com o OpenISA é o uso compartilhado do banco de registradores entre instruções de ponto flutuante de precisão simples e dupla. Entretanto, ambas ISAs se destacam pela *Simplicidade das Instruções*, algo muito positivo para emulação, que o RISC-V herda da sua filosofia RISC, possuindo apenas 107 instruções, sendo 71 (66%) equivalentes às 139 do OpenISA, facilitando a implementação de um tradutor de binários. Portanto, a baixa complexidade, limitado número de instruções e uso não frequente do registrador de *status*, indicam que o RISC-V é uma arquitetura que provavelmente terá um bom desempenho em emulação. Note que emular uma ISA mais simples em cima de uma arquitetura mais complexa é sempre mais fácil do que o contrário. Portanto, a simplicidade do RISC-V comparado com ARM e x86 são bons indicadores.

4. Resultados Experimentais do Tradutor Estático

Para medir a sobrecarga de desempenho do código traduzido de RISC-V e OpenISA, quando comparado com a execução do código nativo, ou seja, compilado diretamente para a arquitetura nativa, cada *benchmark* foi executado 10 vezes e foi calculada a média aritmética dos tempos de cada execução, de modo a minimizar os efeitos do sistema operacional e outros programas sobre o tempo medido. Cada *benchmark* foi traduzido tanto em modo global quanto local, por nosso TEB. Os experimentos foram executados em uma máquina Debian 9, Kernel 4.9.0-6-amd64, Intel Core i7-6700K em 4,00 GHz. Como os resultados do OpenISA [Auler and Borin 2017] foram reproduzidos utilizando a versão 3.7 do LLVM e nossos experimentos utilizaram a versão 7.0, compilamos os *benchmarks* com ambas as versões e comparamos as médias geométricas das sobrecargas obtidas por cada versão do LLVM. As diferenças foram pequenas, menos de 2%, demonstrando que os resultados que obtivemos não foram causados por melhorias no LLVM.

A Figura 1 mostra a sobrecarga de desempenho do código traduzido de RISC-V e OpenISA; quanto maior o valor na figura, pior o resultado e menor o desempenho alcançado. Os resultados, média de 27% de sobrecarga para RISC-V e 15% para OpenISA com contexto local, indicam que, pelo menos para x86, podemos obter bons resultados de desempenho na tradução de RISC-V. Apenas o resultado do crc32 com contexto local foi ruim. Contudo, analisando essa aplicação descobrimos que grande parte desse resultado vem da frequente chamada a função getc(). Atualmente, no TEB em modo local, os valores dos registradores emulados são sempre sincronizados em chamadas de funções, o que não ocorre no TEB OpenISA para funções de bibliotecas externas, o que mostra, portanto, que o desempenho ruim não veio de características ruins do RISC-V, mas de uma otimização ainda não implementada no TEB RISC-V. Além disso, o LLBT [Shen et al. 2012], um dos tradutores estáticos com melhores resultados da literatura, também baseado no LLVM, obteve resultados piores, 66% de sobrecarga na tradução de ARM para Intel Atom, demonstrando novamente a facilidade de emulação do RISC-V.

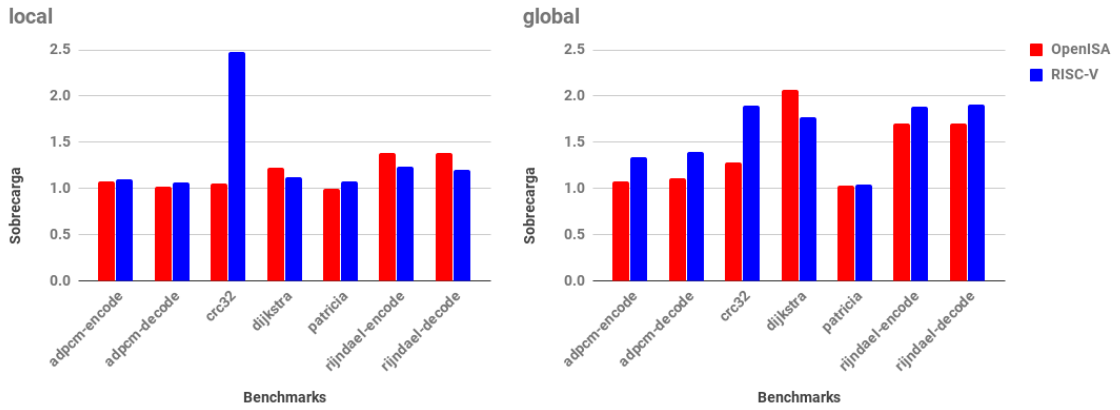

Figura 1. Sobrecarga no desempenho de benchmarks traduzidos.

5. Conclusões

Os resultados da nossa análise sobre o conjunto de instruções RISC-V 32 MFD, e de nosso TEB, indicam que o RISC-V é uma ISA de fácil emulação, ou seja, é simples e sem características que dificultem a tradução de códigos com alta qualidade. Em futuros trabalhos devemos estender o nosso TEB para suportar mais *benchmarks* e testá-lo em mais arquiteturas anfitriãs, como a ARM.

Agradecimentos

Este trabalho foi possível graças ao apoio da CAPES e ao suporte oferecido pela equipe do Laboratório Multidisciplinar de Computação de Alto Desempenho da Unicamp.

Referências

Auler, R. and Borin, E. (2017). The case for flexible isas: unleashing hardware and software. In *SBAC-PAD*, pages 65–72. IEEE.

Borin, E. and Wu, Y. (2009). Characterization of dbt overhead. In *IISWC 2009. IEEE International Symposium on*, pages 178–187. IEEE.

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

Hong, D.-Y., Hsu, C.-C., Yew, P.-C., Wu, J.-J., Hsu, W.-C., Liu, P., Wang, C.-M., and Chung, Y.-C. (2012). Hqemu: a multi-threaded and retargetable dynamic binary translator on multicores. In *CGO*, pages 104–113. ACM.

RISCV.org (2018). Risc-v foundation. *RISC-V Foundation. Url: https://riscv.org/. Acessado em 25 de Fevereiro de 2018.*

Shen, B.-Y., Chen, J.-Y., Hsu, W.-C., and Yang, W. (2012). Llbt: an llvm-based static binary translator. In *Proceedings of the 2012 ESWEEK*, pages 51–60. ACM.

Smith, J. and Nair, R. (2005). *Virtual machines: versatile platforms for systems and processes*. Elsevier.

Waterman, A., Lee, Y., Patterson, D., and Asanovic, K. (2014). The risc-v instruction set manual. *volume I: User-level ISA, version 2.0, EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-54*.

Zhang, X., Guo, Q., Chen, Y., Chen, T., and Hu, W. (2015). Hermes: A fast cross-isa binary translator with post-optimization. In *CGO, 2015 IEEE/ACM International Symposium on*, pages 246–256. IEEE.