Técnicas para Análise de Desempenho de Máquinas Virtuais

Candidato: Rafael Timbó Matos
Orientador: Prof. Dr. Edson Borin

Instituto de Computação
Universidade Estadual de Campinas

Resumo

A análise de desempenho de processadores tem sido realizada através da execução de conjuntos de aplicações denominados *benchmarks*. Da mesma forma, a comunidade científica tipicamente emprega o uso dos mesmos *benchmarks* para avaliar o desempenho de Máquinas Virtuais. Entretanto, resultados de pesquisas recentes [17] indicam que os *benchmarks* tipicamente utilizados para avaliação de processadores (SPEC CPU) podem mascarar a sobrecarga das técnicas de emulação quando utilizados para avaliar máquinas virtuais. Assim sendo, este trabalho propõe um levantamento de *benchmarks* voltados para a análise de máquinas virtuais e a construção de um banco de dados para armazenar resultados de experimentos. Tais *benchmarks* possibilitarão a avaliação e o desenvolvimento de técnicas de emulação em máquinas virtuais e o estudo de técnicas eficientes de interpretação, considerando adequadamente a sobrecarga das técnicas de emulação.

1 Introdução e Justificativa

Máquinas virtuais são programas de computador que emulam uma interface para execução de outros programas, compilados para a interface sendo emulada. Esta tecnologia está presente em diversos sistemas computacionais e é utilizada desde o suporte à linguagens de programação de alto nível, como na máquina virtual Java, até a implementação de processadores com projeto integrado de *hardware* e *software*, como é o caso do processador Efficeon [16] da Transmeta.

Este documento está organizado da seguinte forma: a Seção 1 introduz as principais técnicas de emulação em máquinas virtuais e apresenta as justificativas para o desenvolvimento deste trabalho.

A Seção 2 apresenta metodologia e as ferramentas utilizadas. Por fim, a Seção 3 contém o plano de trabalho e o cronograma da iniciação científica.

1.1 Interpretação

No contexto de máquinas virtuais, interpretação é uma técnica de emulação utilizada para execução de instruções de uma máquina hóspede. Um interpretador executa as instruções do programa hóspede uma a uma, em um ciclo que envolve o carregamento, a análise e a execução da instrução. A ferramenta Bochs [20] é um exemplo de máquina virtual que emula a arquitetura x86 utilizando interpretação.

O processo é relativamente simples e possibilita que o interpretador seja implementado geralmente em uma linguagem de alto nível, o que permite uma maior portabilidade. A simplicidade do procedimento também auxilia na depuração e validação do software. A execução de instruções uma a uma, em ordem, facilita o tratamento de interrupções e exceções durante a execução.

Apesar da simplicidade, este processo é geralmente lento, o que pode tornar inviável a utilização desta técnica em algumas máquinas virtuais. Segundo Smith e Nair [25], um dos principais motivos para a interpretação ser lenta é o fato de que a execução de cada instrução é acompanhada de uma análise - como, por exemplo, a decodificação - e da busca e execução de uma rotina para emular o comportamento da instrução. O interpretador da ferramenta FX!32, apesar de ser otimizado manualmente, executa em média 45 instruções da máquina Alpha para emular uma instrução x86 [11]. As técnicas de pré-decodificação e *direct threaded interpretation* [12, 15] podem ser usadas na prática para acelerar o processo de interpretação.

1.2 Tradução Dinâmica de Binários

Na interpretação, uma mesma rotina executa todas as instruções fontes do mesmo tipo. Por exemplo, toda instrução de subtração é executada por uma mesma rotina que realiza as operações de subtração. O desempenho do interpretador pode ser melhorado significativamente se cada instrução fonte for mapeada para a sua própria rotina, otimizada para execução da instrução. Este processo de converter as instruções do programa hóspede para instruções da máquina hospedeira é denominado tradução de binários [24].

Na tradução estática de binários, todo o código binário é traduzido antes da aplicação ser execu-

tada. Esta técnica nem sempre é possível, pois certas aplicações geram parte do seu código dinamicamente. Além disso, existem aplicações que misturam a área de código com a área de dados, tornando o processo de descoberta do código antes da execução mais difícil ou até mesmo impossível. Por outro lado, a tradução dinâmica de binários traduz o código hospedeiro à medida que ele é executado, isto é, à medida que o código é descoberto. Esta abordagem permite a tradução de código gerado dinamicamente. Adicionalmente, como apenas código executado é traduzido, o tradutor não precisa diferenciar código de dados.

A tradução dinâmica de binários envolve um custo de inicialização associado à tradução do programa hóspede durante a execução. No entanto, o código traduzido é geralmente armazenado em uma região de memória e reutilizado quando o mesmo trecho do programa hóspede precisa ser reexecutado. A reutilização do código traduzido amortiza o custo de inicialização e, nos casos onde a execução do código é frequente, a tradução dinâmica de binários provê uma execução mais rápida que a interpretação. Borin e Wu [6] mostram que o processo de tradução dinâmica da ferramenta StarDBT causa, em média, uma perda de desempenho de apenas 9% quando executa aplicações do benchmark SPEC CPU 2000 [14].

1.3 *Benchmarks* para Avaliação de Máquinas Virtuais

A tradução dinâmica de binários é uma técnica eficiente para execução de código frequente, também conhecido como código quente. Em trechos de código frequentemente executados, o tempo de inicialização, usado pela tradução, é apenas uma pequena fração do tempo de execução. Nesses casos, o bom desempenho do código traduzido amortiza o custo de inicialização, o que torna a tradução dinâmica de binários mais vantajosa do que a técnica de interpretação. Por outro lado, para trechos de código pouco executados, ou código frio, o custo de inicialização pode ser muito maior do que o tempo de execução do código traduzido.

Existem diversos tipos de aplicações onde o custo de inicialização pode tornar o uso de um tradutor de binários impraticável. Por exemplo, aplicações de tempo real, onde tarefas devem ser completadas em um determinado limite de tempo, podem ter suas restrições de tempo violadas por causa do tempo de inicialização da tradução de binários. Aplicações compostas por múltiplos programas de execução curta, onde a execução termina antes da execução do código otimizado amortizar o custo de inicialização, podem ter o seu tempo total de execução aumentado significativamente. O processo de inicialização do sistema operacional, frequente em plataformas como *laptops* e dispositivos móveis,

também é caracterizado pela execução de muito código frio, e o custo de inicialização pode tornar a tradução dinâmica inviável. Nesses casos, combinar interpretação com tradução dinâmica de binários é imprescindível.

Apesar dos exemplos mencionados acima, a maioria dos trabalhos relacionados [14, 18] só apresenta resultados de experimentos com *benchmarks* caracterizados por gastar a maior parte do tempo de execução em código quente, como o *benchmark* Spec CPU [14]. Este projeto de pesquisa levantará e desenvolverá novos *benchmarks* com características mais próximas ao uso típico de um computador móvel ou de mesa, permitindo uma avaliação mais realista do desempenho de máquinas virtuais. Estes *benchmarks* darão suporte a projetos correlacionados no laboratório de sistemas de computação (LSC), onde técnicas de tradução dinâmica de binários eficientes estão sendo desenvolvidas. Os *benchmarks* serão executados nativamente e emulados com máquinas virtuais e os resultados experimentais serão armazenados em um banco de dados, que servirá de base para o cruzamento de dados e repositório para as análises em outros projetos de pesquisa.

1.4 Justificativa Quanto ao Candidato e Orientador

O aluno candidato deste projeto estudou no Colégio Militar de Salvador. O candidato foi aprovado nos vestibulares de todas as universidades que prestou, dentre elas, Universidade Federal de Pernambuco e Universidade Federal da Bahia, sendo primeiro colocado nesta última no curso de Engenharia da Computação no ano de 2010. Na Universidade Estadual de Campinas, obteve mais do que 80% de rendimento em todas suas matérias de computação, mantendo-se em oitavo colocado na sua turma de Ciência da Computação no 1° semestre de 2012. O orientador deste trabalho tem experiência na área de tradução dinâmica de binários com diversas contribuições, incluindo novos algoritmos e suporte em *hardware* para implementação de máquinas virtuais eficientes. O pesquisador trabalhou quatro anos como cientista pesquisador no *Programming System Labs* da Intel, na Califórnia. Como parte do grupo BCT (*Binary Compilation Technology*), ele desenvolveu e publicou diversas técnicas para melhorar a tradução dinâmica de binários em diversos sistemas, incluindo novas técnicas de formação de regiões [8, 23, 9], técnicas de paralelização dinâmica [26], avaliação de desempenho de tradutores dinâmicos de binários [5, 7], e outros [27, 3, 4, 22]. Além destas publicações e outras patentes, listadas na súmula curricular, o pesquisador teve a oportunidade de trabalhar diretamente na infraestrutura do processador Efficeon [16][1], o estado da arte em tecnologia de máquinas virtuais de

[1] A infraestrutura do processador Efficeon foi licenciada pela empresa Intel.

sistema e de máquinas virtuais com o projeto integrado de *hardware* e *software*.

2 Materiais e Métodos

Os testes de desempenho serão realizados em na plataforma ARM i.MX53 Quick Start Board [19], fabricada pela Freescale e em computadores com processadores da família x86. A plataforma i.MX53 vem equipada com um *System-on-a-Chip* (SoC) com um processador Cortex-A8™ de 1GHz integrado com unidades de processamento de vídeo, de imagem e de gráficos 2D e 3D. Além disso, a plataforma possui 1GB de memória RAM e conexões ethernet, USB, RD 232, PATA, SATA II, e VGA. O sistema é fornecido com um cartão SD (de 4GB) com uma distribuição Linux Ubuntu Lucid Lynx 2.6.35.3 [1], incluindo o compilador gcc-4.4.3 [13]. Esta plataforma será responsável por executar *benchmarks*, como o SPEC CPU [14], de forma nativa e emulada. As máquinas virtuais responsáveis pela emulação desfrutarão de diversas técnicas de emulação, que poderão então ser testadas.

Como exemplo de máquinas virtuais usaremos as ferramentas QEMU [2] e DynamoRIO [10]. O QEMU é um tradutor dinâmico de binários flexível capaz de emular tanto um processo quanto um sistema operacional, o DynamoRIO é um tradutor dinâmico de binários eficiente desenvolvido pelo MIT. Pretendemos utilizar a ferramenta Bochs [20] para os testes de interpretação.

Com base em técnicas da literatura [12, 15], os *benchmarks* atuais serão adaptados para ponderar também código frio, simulando mais autenticamente uma máquina virtual. O objetivo é reproduzir com fidelidade os métodos utilizados em aplicações reais durante a avaliação dos interpretadores e tradutores dinâmicos de binários.

Um banco de dados será desenvolvido para armazenamento dos resultados de experimentos, que será implementado utilizando a ferramenta mysql [21]. O mesmo servirá de base para consolidar e armazenar resultados de experimentos provenientes deste e de outros projetos do laboratório de sistemas de computação do Instituto de Computação da Unicamp.

3 Plano de Trabalho

Inicialmente, nos primeiros meses, *benchmarks* atuais serão analisados para servir de base para a construção de um novo *benchmark*, sendo registradas as situações de interpretação não testadas. Em

seguida, novos *benchmarks* focados em máquinas virtuais serão desenvolvidos e testados, com posteriores correções e otimizações. Paralelo à isto, será desenvolvido o banco de dados.

3.1 Cronograma

	Mês											
Itens	1	2	3	4	5	6	7	8	9	10	11	12
Implementação do banco de dados	•	•	•									
Estudo e levantamento de *benchmarks*	•	•	•	•								
Testes do banco de dados				•	•							
Implementação dos *benchmarks*				•	•	•	•					
Experimentos de desempenho com os *benchmarks*							•	•	•	•	•	
Relatório final e artigos											•	•

Referências

[1] B. Al Housani, B. Mutrib, and H. Jaradi. The linux review - ubuntu desktop edition - version 8.10. In *Current Trends in Information Technology (CTIT), 2009 International Conference on the*, pages 1 –6, dec. 2009.

[2] Daniel Bartholomew. Qemu: a multihost, multitarget emulator. *Linux J.*, 2006:3–, May 2006.

[3] Edson Borin, Cheng Wang, Youfeng Wu, and Guido Araujo. Dynamic binary control-flow errors detection. *SIGARCH Comput. Archit. News*, 33:15–20, December 2005.

[4] Edson Borin, Cheng Wang, Youfeng Wu, and Guido Araujo. Software-based transparent and comprehensive control-flow error detection. In *Proceedings of the International Symposium on Code Generation and Optimization*, CGO '06, pages 333–345, Washington, DC, USA, 2006. IEEE Computer Society.

[5] Edson Borin and Youfeng Wu. Characterization of dynamic binary translation overhead. **best paper award**. In *Proceedings of the 1st Workshop on Architectural and Microarchitectural Support for Binary Translation (AMAS-BT '08)*, June 2008.

[6] Edson Borin and Youfeng Wu. Characterization of dbt overhead. In *Proceedings of the 2009 IEEE International Symposium on Workload Characterization*, volume 0 of *IISWC '09*, pages 178–187, Los Alamitos, CA, USA, 2009. IEEE Computer Society.

[7] Edson Borin and Youfeng Wu. Characterization of dbt overhead. In *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC '09)*, volume 0, pages 178–187, Los Alamitos, CA, USA, 2009. IEEE Computer Society.

[8] Edson Borin, Youfeng Wu, Mauricio Breternitz Jr., and Cheng Wang. Lar-cc: Large atomic regions with conditional commits. Accepted to be published at the International Symposium on Code Generation and Optimization (CGO '11), April 2011.

[9] Edson Borin, Youfeng Wu, Cheng Wang, Wei Liu, Mauricio Breternitz Jr., Shiliang Hu, Esfir Natanzon, Shai Rotem, and Roni Rosner. Tao: Two-level atomicity for dynamic binary optimizations. To appear at the Proceedings of the International Symposium on Code Generation and Optimization (CGO '10), April 2010.

[10] Derek Bruening, Timothy Garnett, and Saman Amarasinghe. An infrastructure for adaptive dynamic optimization. In *CGO '03: Proceedings of the international symposium on Code generation and optimization*, pages 265–275, Washington, DC, USA, 2003. IEEE Computer Society.

[11] Anton Chernoff, Mark Herdeg, Ray Hookway, Chris Reeve, Norman Rubin, Tony Tye, S. Bha radwaj Yadavalli, and John Yates. Fx!32: A profile-directed binary translator. *IEEE Micro*, 18(2):56–64, 1998.

[12] M. Anton Ertl and David Gregg. Optimizing indirect branch prediction accuracy in virtual machine interpreters. In *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation*, PLDI '03, pages 278–288, New York, NY, USA, 2003. ACM.

[13] Brian Gough, Foreword Richard, and M. Stallman. An introduction to gcc for the gnu compilers gcc and g++.

[14] John L. Henning. Spec cpu2000: Measuring cpu performance in the new millennium. *Computer*, 33:28–35, 2000.

[15] Paul Klint. Interpretation techniques. *Software - Practice & Experience*, 11(9):963–973, September 1981.

[16] Kevin Krewell. Transmeta gets more efficeon. *Microprocessor Report*, 17(10), 2003.

[17] Divino Lucas, Rafael Dalibera, Rafael Auler, Guido Araújo, Sandro Rigo, and Edson Borin. Hotness misprediction overhead on virtual machines. In *4th Workshop on Infrastructures for Software/Hardware co-design - WISH*, 2012.

[18] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. Pin: Building customized program analysis tools with dynamic

instrumentation. In *Proceedings of the ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation*, PLDI '05, pages 190–200, New York, NY, USA, 2005. ACM.

[19] Kinetis Microcontrollers and Kinetis Microcontrollers. Freescale solutions embedded control solutions based on arm ® technology.

[20] Darek Mihocka and Stanislav Shwartsman. Virtualization without direct execution or jitting: Designing a portable virtual machine infrastructure. In *Proceedings of the 1st Workshop on Architectural and Microarchitectural Support for Binary Translation*, AMAS-BT '08, June 2008.

[21] AB MySQL. Mysql reference manual. 2001.

[22] João Paulo Porto, Guido Araujo, Edson Borin, and Youfeng Wu. Trace execution automata in dynamic binary translation. In *Proceedings of the 3rd Workshop on Architectural and Microarchitectural Support for Binary Translation (AMAS-BT '10)*, June 2010.

[23] João Paulo Porto, Guido Araujo, Youfeng Wu, Edson Borin, and Cheng Wang. Compact trace trees in dynamic binary translators. In *Proceedings of the 1st Workshop on Architectural and Microarchitectural Support for Binary Translation (AMAS-BT '08)*, June 2008.

[24] Richard L. Sites, Anton Chernoff, Matthew B. Kirk, Maurice P. Marks, and Scott G. Robinson. Binary translation. *Communications ACM*, 36(2):69–81, 1993.

[25] Jim Smith and Ravi Nair. *Virtual Machines: Versatile Platforms for Systems and Processes (The Morgan Kaufmann Series in Computer Architecture and Design)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[26] Cheng Wang, Youfeng Wu, Edson Borin, Shiliang Hu, Wei Liu, Dave Sager, Tin-fook Ngai, and Jesse Fang. Dynamic parallelization of single-threaded binary programs using speculative slicing. In *Proceedings of the 23rd international conference on Supercomputing*, ICS '09, pages 158–168, New York, NY, USA, 2009. ACM.

[27] Youfeng Wu, Shiliang Hu, Edson Borin, , and Cheng Wang. A hw/sw co-designed multi-core virtual machine for energy-efficient general purpose computing. Accepted to be published at the International Symposium on Code Generation and Optimization (CGO '11), April 2011.