Pré projeto de Pesquisa
Redução do Tempo de Instrumentação

Camilo S. Melgaço[1]

[1] Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte, MG - Brasil

1. Introdução

A depuração de programas é uma fase importante no desenvolvimento de software [Hailpern and Santhanam 2002]. Ela pode ser feita durante a codificação ou após a conclusão do desenvolvimento, quando a aplicação já está em formato para distribuição, chamado de binário da aplicação. Uma das técnicas de depuração é a instrumentação.

A instrumentação é o processo de inserir código novo no programa sem que seu comportamento geral seja alterado [Priyadarshan 2019]. A instrumentação dinâmica de binários, ou IDB (em inglês, *DBI*: *Dynamic Binary Instrumentation*), consiste em inserir instruções no binário durante sua execução. Essas instruções são então utilizadas como pontos de controle, gerando informações durante a execução do programa sobre essa própria execução. Várias análises podem ser feitas com IDB, como monitorar acessos a memória e reconstruir grafos de fluxo de controle, entre outros [Nethercote 2004].

Uma possibilidade de análise utilizando IDB é a detecção de código quente. Código quente é uma parte do programa que é executada proporcionalmente mais vezes quando comparada ao resto do programa. Essas partes podem ser pontos de interesse para otimização [Abeilice and Álvares 2022]. Para realizar a IDB, pode ser utilizado um *framework* IDB. Diferentes *frameworks* podem ter desempenhos distintos para uma mesma tarefa, e então são recomendados para aplicações diferentes. Uma ferramenta de contagem de instruções para detecção de código quente teve desempenho variado ao ser implementada em diferentes *frameworks* IDB [Abeilice and Álvares 2022], sendo eles VALGRIND[Nethercote and Seward 2007], PIN[Sousa and Tymburibá 2018] e DYNAMORIO[Bruening et al. 2003].

A análise de programas usando a instrumentação dinâmica não é uma tarefa simples e muitas vezes requer várias repetições, o que pode demandar muito tempo. Pensando nisso, este trabalho se propõe a buscar estratégias de diminuir o tempo gasto para realizar uma análise por instrumentação, estudando o efeito de instrumentar parcialmente o programa alvo. Isso será feito de duas maneiras: ignorando as bibliotecas compartilhadas durante a instrumentação, concentrando a análise apenas no código próprio do programa; e filtrando trechos do programa a serem instrumentados, realizando a análise apenas no código quente.

Essas reduções de código instrumentado podem não ser úteis para todos os tipos de instrumentação, como em usos de segurança onde todo o programa deve ser monitorado, mas podem ser utilizadas em casos de otimização, onde não é necessário instrumentar todo o programa para avaliar apenas parte do código. O objetivo é verificar qual o comportamento do tempo de execução do programa instrumentado quando se varia a porção do programa que é efetivamente instrumentada para análise.

2. Referencial Teórico

Esta pesquisa teve forte influência de [Sousa and Tymburibá 2018], onde os autores descrevem o que são ferramentas IDB e como utilizá-las, tendo como objeto de exemplo o *framework* IDB PIN. Além disso, o PINtem uma extensa API bem documentada, o que motiva esta pesquisa a tê-lo como primeira opção de *framework* IDB, assim como os autores de [Sousa and Tymburibá 2018] fizeram.

Na literatura, existem trabalhos que comparam diferentes *frameworks* IDB entre si, como [Abeilice and Álvares 2022] e [Rodríguez et al. 2014], mas todos eles consideram a instrumentação do programa completo, o que os diferem do objetivo deste pesquisa. Esta se propõe a verificar se limitar a faixa de instruções de um programa a ser analisada durante a instrumentação desse programa se traduz em uma melhora considerável de desempenho do processo de instrumentação como um todo.

Uma investigação mais próxima foi apresentada em [Álvares et al. 2021], onde os autores mostram que a maioria das instruções executadas no *SPEC CPU 2017* são visíveis, ou seja, foram instrumentadas, mas uma parcela relevante é invisível, ou seja, não foi instrumentada. Este assunto é tratado na primeira etapa proposta nessa pesquisa, que consiste em ignorar as bibliotecas compartilhadas durante a instrumentação, o que efetivamente se traduz em não instrumentar instruções invisíveis.

Assim, este pesquisa busca preencher uma lacuna no estudo do desempenho de IDBs, evidenciando com dados o impacto de se instrumentar parcialmente um programa.

3. Metodologia

Para se alcançar o objetivo da pesquisa, será necessário desenvolver uma ferramenta IDB em diferentes *frameworks* IDB que apresente sobrecarga de tempo de execução de um programa alvo considerável em relação à execução desse programa sem instrumentação. Após isso, devem ser implementadas técnicas para reduzir a porção de código instrumentada que representam situações reais do desenvolvimento de aplicações, como instrumentação de código quente. Estas técnicas devem ser reproduzidas nos diferentes *frameworks* para efeitos de comparação. Com as ferramentas prontas, faz-se necessária a definição de um conjunto de aplicações que serão utilizadas para testes, como por exemplo a suíte *SPEC CPU 2017*. Por fim, os dados de tempo de execução serão analisados possibilitando que conclusões sejam tomadas sobre a eficiência das técnicas desenvolvidas.

Assim, esta pesquisa será dividida nas seguintes etapas:

1. Revisão bibliográfica;
2. Desenvolvimento das ferramentas IDB;
3. Implementação das técnicas de redução de código analisado;
4. Definição dos programas de teste;
5. Aplicação dos testes;
6. Avaliação dos resultados.

A escrita do texto será feita ao longo do desenvolvimento da pesquisa.

4. Cronograma

Esta pesquisa deverá ser desenvolvida ao longo de 20 meses. As partes compostas de desenvolvimento demandam de bastante tempo, exigindo maior esforço intelectual, podendo apresentar percalços ao longo de sua realização. Já a parte de aplicação de testes demanda muito tempo devido à natureza da instrumentação, que aumenta substancialmente o tempo de execução de um programa. Assim, propõe-se o seguinte cronograma:

1. Revisão bibliográfica: 2 meses;
2. Desenvolvimento das ferramentas IDB: 5 meses;
3. Implementação das técnicas de redução de código analisado: 5 meses;
4. Definição dos programas de teste: 1 mês;
5. Aplicação dos testes: 4 meses;
6. Avaliação dos resultados: 1 mês;
7. Revisão do texto: 1 mês.

Totalizando 19 meses, reservando 1 mês (cerca de 5% do tempo total) para eventuais emergências e atrasos.

Referências

Abeilice, A. d. F. and Álvares, A. R. (2022). Comparativo entre instrumentadores dinâmicos de binários. *CEFET-MG - DECOM*.

Bruening, D., Garnett, T., and Amarasinghe, S. (2003). An infrastructure for adaptive dynamic optimization. In *International Symposium on Code Generation and Optimization, 2003. CGO 2003.*, pages 265–275. IEEE.

Hailpern, B. and Santhanam, P. (2002). Software debugging, testing, and verification. *IBM Systems Journal*, 41(1):4–12.

Nethercote, N. (2004). Dynamic binary analysis and instrumentation. Technical report, University of Cambridge, Computer Laboratory.

Nethercote, N. and Seward, J. (2007). Valgrind: a framework for heavyweight dynamic binary instrumentation. *ACM Sigplan notices*, 42(6):89–100.

Priyadarshan, S. (2019). A study of binary instrumentation techniques. *SBU SecLab Journal*.

Rodríguez, R. J., Artal, J. A., and Merseguer, J. (2014). Performance evaluation of dynamic binary instrumentation frameworks. *IEEE Latin America Transactions*, 12(8):1572–1580.

Sousa, H. and Tymburibá, M. (2018). Análise dinâmica de programas binários. *Sociedade Brasileira de Computação*.

Álvares, A. R., Amaral, J. N., and Pereira, F. M. Q. (2021). Instruction visibility in spec cpu2017. *Journal of Computer Languages*, 66:101062.