Experiências com Ferramentas de Detecção de *Data Races* em Programas *Multithread*

Ricardo B. Gomes[1], Matheus R. de Figueiredo[1], Andrea S. Charão[1]
Cleber C. Sartorio[2], Dionatan K. Tietzmann[2]

[1] Universidade Federal de Santa Maria – UFSM

[2]Universidade Regional do Noroeste do Estado do Rio Grande do Sul – Unijuí

Resumo. *Neste trabalho, apresenta-se experiências com algumas ferramentas de detecção de* data races *disponíveis gratuitamente, para diferentes linguagens, reunindo subsídios que possam ser úteis a programadores confrontados com o problema em questão. Usou-se 3 ferramentas, cobrindo as linguagens C/C++ e Java, aplicadas a programas de pequeno porte. Os resultados confirmam a eficácia das ferramentas, mas evidenciam disparidades entre os resultados.*

1. Introdução

A programação paralela costuma ser considerada difícil, devido a características como o não-determinismo e a propensão a problemas de sincronização [Mühlenfeld 2008]. Um problema típico neste domínio são os chamados *data races*, em que acessos concorrentes a um dado compartilhado podem produzir um resultado incorreto. Programas em diferentes linguagens estão sujeitos a *data races*, à medida que utilizam bibliotecas ou outros recursos de programação *multithread*.

Para lidar com este tipo de problema, muitos pesquisadores se concentram na concepção de soluções para detecção de *data races*. Em alguns casos, as pesquisas dão origem a ferramentas que ficam disponíveis aos programadores, porém nem sempre são do conhecimento de quem desenvolve programas paralelos. Neste trabalho, apresenta-se experiências com algumas ferramentas de detecção de *data races* disponíveis gratuitamente, para diferentes linguagens. Tem-se como objetivo verificar sua eficácia e, ao mesmo tempo, reunir subsídios que possam ser úteis a programadores confrontados com o problema em questão.

2. Abordagens de Detecção de Data Races

Data races podem induzir comportamentos distintos para um mesmo programa concorrente. Portanto, tais situações devem ser identificadas e eliminadas do programa. Identificar com precisão a ocorrência de *data races* é uma tarefa complexa, tanto manual como automaticamente. Soluções automatizadas vem sendo propostas ao longo do tempo [Savage et al. 1997], cada uma com suas vantagens e limitações.

As abordagens para a detecção de *data races* geralmente se baseiam em estratégias de análise estática e/ou dinâmica do programa. A análise estática é realizada sobre o código-fonte, sem executá-lo, analisando os caminhos de execução possíveis [Netzer 1991]. Essa abordagem é pouco escalável e não leva em consideração a semântica dos programas. A análise dinâmica, por sua vez, ocorre durante a execução e concentra-se num caminho real percorrido pelas instruções do programa [Savage et al. 1997]. Ferramentas que usam uma abordagem dinâmica basicamente efetuam um acompanhamento do código ao ser executado, analisando-o instrução por instrução e registrando as ocorrências.

3. Ferramentas Utilizadas

As ferramentas escolhidas para este trabalho fazem uso de estratégias de análise dinâmica para detecção de *data races*. Selecionou-se ferramentas amplamente disponíveis e procurou-se cobrir linguagens bastante utilizadas: C/C++ e Java. Foram 3 as ferramentas escolhidas: Valgrind, Oracle Developer Studio e DRD.

3.1. Valgrind

Valgrind [Valgrind 2016] é um conjunto de ferramentas que possuem diversos usos para *debugging* e *profiling* de aplicações, disponível para Linux e vários outros sistemas operacionais. É capaz de trabalhar com diversas linguagens, incluindo Fortran, Java, Perl, Python, entre outras, embora originalmente seu nicho de aplicação seja mais voltado para programas em C/C++.

Suas ferramentas permitem a deteção de erros de gerenciamento de memória, *profiling* de *cache* e *heap*, depuração de programas *multithreads* e outros testes mais específicos. A interface com o usuário é feita através de comandos pelo próprio terminal do sistema, permitindo que seja feito o *profiling* ou *debugging* de uma aplicação já compilada. As informações obtidas são apresentadas em forma textual, contendo informações e erros detectados durante a execução, indicando a localização por meio da identificação da linha, processo e endereço de memória. Essas informações são apresentadas na tela do terminal ou podem ser redirecionadas para *logs* para visualização posterior.

3.2. Oracle Developer Studio

Oracle Developer Studio (ODS) [Oracle 2016] é um IDE completo para desenvolvimento de aplicações em C/C++. Dispõe de uma interface gráfica similar a outro IDE bastante conhecido, o NetBeans. Possui integrado a si um menu de configuração de *profiling*, onde é possível especificar o tipo de análise a ser executada. Em especial, é encontrado neste menu a opção de verificação de *data races*. As informações coletadas durante o *profiling* são organizadas e apresentadas em uma nova guia, na interface gráfica. Também pode ser encontrado um gráfico em tempo real contando o número de *threads* criadas pelo programa analisado.

Para a utilização do ODS, é necessário de computador baseado na arquitetura SPARC com sistema operacional Solaris, ou um computador baseado em x64 com sistema operacional Linux que suporte pacotes RPM nativamente. Também é necessário possuir instalado um *runtime* Java para que seja possível instalar e executar a ferramenta.

3.3. Data Race Detector (DRD)

Data Race Detector (DRD) [Tsitelov 2016] é uma ferramenta que visa a detecção de *data races* em programas construídos na linguagem Java. Trata-se de um Java Agent, o qual deve ser especificado como parâmetro no momento em que a aplicação a ser analisada é invocada, pela linha de comando. O DRD não possui interface gráfica e a exibição dos dados de análise se dá em formato textual e através dos arquivos de *logs* que são criados.

4. Experimentos e Resultados

Para os experimentos, decidiu-se focar em programas de pequeno porte, que fossem de fácil análise manual, formando um conjunto acessível a iniciantes. Tais programas foram

obtidos em fóruns de discussão e materiais instrucionais. Isso contrasta com outras abordagens utilizadas em trabalhos relacionados [Yu et al. 2016, Flanagan e Freund 2009], em que são usados *benchmarks* incluindo programas de grande porte, que testam os limites das ferramentas mas não favorecem a compreensão por iniciantes.

Ao todo, dispôs-se de 7 programas em C/C++ e 5 programas em Java. Primeiramente, foi feita uma análise visual nos códigos a fim de detectar a ocorrência de *data races*. Após essa etapa, foram então realizados testes utilizando as ferramentas Valgrind e ODS para os programas em C/C++ e DRD para os programas em Java.

A ferramenta Valgrind foi testada em um computador com sistema operacional Ubuntu 14.04 de 64 bits, enquanto as ferramentas ODS e DRD foram instaladas em um computador com sistema Fedora 24 64 bits. Ambos os computadores possuíam suporte nativo para *multithreading*, respectivamente com processadores dual core com suporte para 4 *threads* e octa-core com suporte para 16 *threads*.

Para cada programa, realizaram-se 3 execuções com as ferramentas escolhidas, utilizando-se suas configurações *default*. Os resultados obtidos com as ferramentas foram então comparados com aqueles obtidos pela análise manual dos programas. Nas tabelas 1 e 2, apresenta-se o número de *data races* detectados para os programas em C/C++ e Java.

Analisando-se as tabelas, nota-se que as ferramentas ODS e Valgrind produziram resultados diferentes entre si na maior parte dos casos. Porém, no geral, não ficaram distantes dos resultados da detecção manual, mesmo havendo alguns falso-positivos e também alguns casos não detectados. No caso da ferramenta DRD, nota-se que foram detectados muitos *data races* além daqueles identificados manualmente. Notou-se que o DRD analisa não apenas o programa em si, mas toda as bibliotecas do Java, o que explica o maior número de *races* identificados, que não puderam ser confirmados manualmente.

Tabela 1. *Data Races* detectados nos programas em C/C++

Programa/Ferramenta	Manual	ODS	Valgrind
Programa 1[1]	1	1	0
Programa 2[2]	1	1	1
Programa 3[3]	1	2	1
Programa 4[4]	0	0	1
Programa 5[5]	2	3	3
Programa 6[6]	3	7	4
Programa 7[7]	1	2	5

Tabela 2. *Data Races* detectados nos programas em Java

Programa/Ferramenta	Manual	DRD
Programa 8[8]	1	10
Programa 9[9]	1	15
Programa 10[10]	1	10
Programa 11[11]	1	0
Programa 12[12]	1	14

5. Conclusão

Neste trabalho, realizou-se um estudo empírico usando ferramentas de detecção de *data races*, aplicadas a programas de pequeno porte. Pôde-se evidenciar que as ferramentas, no geral, cumprem os objetivos a que se propõem, embora possam produzir resultados variados. Este estudo inicial pode ser continuado em trabalhos futuros, buscando aprofundar as explicações para as diferenças encontradas em cada caso e, também, ampliando a quantidade de ferramentas e programas analisados.

Referências

Flanagan, C. e Freund, S. N. (2009). Fasttrack: Efficient and precise dynamic race detection. In *Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '09, pages 121–133, New York, USA. ACM.

Mühlenfeld, A. (2008). *Runtime Data Race Detection in Multi-threaded Programs*. VDM Verlag, Saarbrücken, Germany.

Netzer, R. H. B. (1991). *Race condition detection for debugging shared-memory parallel programs*. PhD thesis, University of Wisconsin–Madison.

Oracle (2016). Oracle developer studio. Disponível em: http://www.oracle.com/technetwork/server-storage/developerstudio/overview/index.html.

Savage, S., Burrows, M., Nelson, G., Sobalvarro, P., e Anderson, T. (1997). Eraser: A dynamic data race detector for multithreaded programs. *ACM Trans. Comput. Syst.*, 15(4):391–411.

Tsitelov, D. (2016). Data race detector. Disponível em: https://code.devexperts.com/display/DRD.

Valgrind (2016). Valgrind home. Disponível em: http://valgrind.org/.

Yu, M., Park, S.-M., Chun, I., e Bae, D.-H. (2016). Experimental performance comparison of dynamic data race detection techniques. *ETRI Journal*.

[1]https://bartoszmilewski.com/2014/10/25/dealing-with-benign-data-races-the-c-way/

[2]http://thispointer.com/c11-multithreading-part-4-data-sharing-and-race-conditions/

[3]http://stackoverflow.com/questions/26998183/how-do-i-deal-with-a-data-race-in-openmp

[4]https://www.ecse.rpi.edu/Homepages/wrf/wiki/ParallelComputingSpring2014/openmp/Using-OpenMP-Examples-Distr/chapter7/

[5]https://docs.oracle.com/cd/E19205-01/820-0619/gdvwv/index.html

[6]https://docs.oracle.com/cd/E19205-01/820-0619/gdvwv/index.html

[7]https://www.ecse.rpi.edu/ wrf/wiki/ParallelComputingSpring2015/openmp/llnl/

[8]https://www.cs.umd.edu/class/fall2005/cmsc433/lectures/threadsPart2.pdf

[9]http://stackoverflow.com/questions/10309186/why-does-this-thread-data-race/

[10]http://jeremymanson.blogspot.com.br/2008/12/benign-data-races-in-java.html

[11]http://stackoverflow.com/questions/3836680/data-race-in-java-arraylist-class/

[12]http://stackoverflow.com/questions/10309186/why-does-this-thread-data-race