Otimização por Random-Mutation Hill Climbing

Rodrigo Tsuhako

***Resumo*—Os compiladores exercem um papel importante na qualidade de um programa. Entretanto, devido a quantidade de possibilidades de otimizações nem sempre o compilador gerará um código de boa qualidade. Para auxiliar na escolha de um bom conjunto de otimizações é possível aplicar heurísticas de busca. No presente trabalho será feito a análise do impacto que as versões do algoritmo Random-Mutation Hill Climbing provoca no desempenho do compilador CLANG. Tal análise mostrou que o Random-Mutation Hill Climbing com mutação de remoção tem maior taxa de melhoria.**

***Index Terms*—compilador, otimização, Hill Climbing .**

I. Introdução

ATUALMENTE códigos são escritos em linguagem de programação de alto nível e são convertidos em linguagem de máquina, [1] sem modificar a semântica do programa, por meio de um compilador. Dessa forma, seu papel influencia diretamente no desempenho final da tarefa realizada.

Entretanto, os códigos gerados pelos compiladores modernos nem sempre possuem a melhor qualidade devido às diversas possibilidades de otimização. Nesse contexto, para que o compilador gere um código de qualidade é necessário um bom conjunto de otimizações e para isso é possível aplicar heurísticas para buscar um conjunto de otimizações que visam melhorar o desempenho de um programa.

O presente trabalho tem como objetivo aplicar o algoritmo Random-Mutation Hill Climbing para encontrar um bom conjunto de otimizações para o compilador CLANG e analisar o desempenho alcançado em dez programas.

II. Desenvolvimento

Os experimentos foram realizados em uma máquina com sistema operacional Ubuntu 18.04.3 LTS com processador Intel Core I5-4210, 1,7 GHz e 8G de memória RAM. Versão do CLANG 6.0.0 e versão do LLVM 6.0.0. Foram utilizados dez programas do Benchmark Test Framework (TF):ASCI-Purple-logs, enc-md5, espresso, evalloop, Fhourstones, fib2, misr, n-body, RSBench e uudecode. Cada programa foi executado três vezes para cada versão de mutação do algoritmo Random-Mutation Hill Climbing, os resultados apresentados são a média entre 3 execuções para cada uma das versões.

O algoritmo Random-Mutation Hill Climbing é uma técnica de busca local iterativa que tem como objetivo encontrar soluções boas para um determinado problema. No presente trabalho a implementação do algoritmo foi baseada na abordagem de [2] Jisheng Zhao.

1) Escolher conjunto inicial como melhor avaliado.
2) Aplicar mutação de característica aleatória no conjunto.
3) Avaliar novo conjunto gerado. Se o conjunto for pior ele é ignorado, se for melhor ele se torna o novo melhor conjunto.
4) Se o algoritmo atingiu o critério de parada, retorne o melhor conjunto. Caso contrário, volte a etapa 2.

Como conjunto inicial foi selecionado o conjunto de otimizações O3.

O algoritmo implementado possui quatro mutações diferentes com características aleatórias: mutação de remoção do conjunto O3, mutação de inserção no conjunto O3, mutação de troca na ordem dos elementos de O3, mutação de troca entre elementos de O3 com o domínio de O3.

O critério de parada foi definido como oitocentas iterações.

A principal métrica avaliada foi o desempenho provocado em cada programa após a aplicação do algoritmo Random-Mutation Hill Climbing, para isso foi necessário extrair o tempo de execução do conjunto inicial e o melhor tempo de execução atingido.

III. Análise e Resultados

Para analisar o desempenho que a otimização proporcionou, foi calculada a taxa de melhoria representada pela seguinte [3] fórmula: melhoria (porcentagem) = (speedup - 1)*100. Sendo o speedup a razão entre o tempo de execução sem otimização com o tempo de execução com otimização.

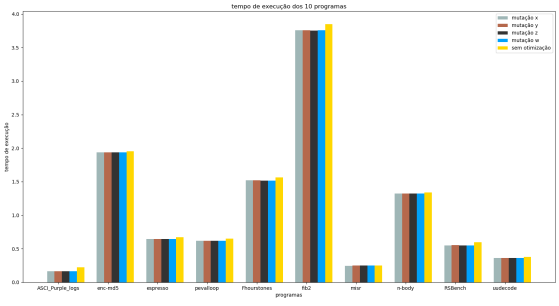

Figura 1. Tempo de execução.

Na figura 1 é possível observar que após a aplicação do algoritmo Random-Mutation Hill Climbing, o tempo de execução dos programas melhorou. Tais resultados indicam que independente da mutação escolhida, o algoritmo irá encontrar um conjunto de parâmetros dentro das otimizações O3 que reduz o tempo de execução do programa.

Universidade Estadual de Maringá - Departamento de Informática
Outubro 06, 2019;

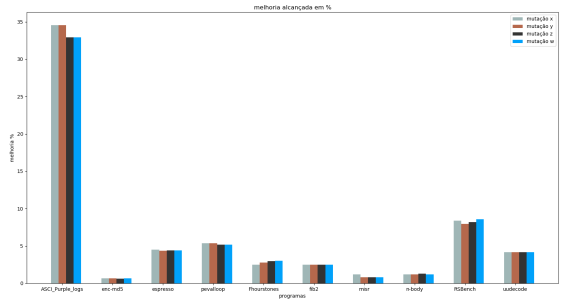

Figura 2. Melhoria.

A figura 2 indica a porcentagem de desempenho provocado, e os resultados indicam que o algoritmo provocou melhor desempenho no programa ASCI-Purple-logs. Mostrando que é possível otimizar programas que já tenham tempo de execução baixos.

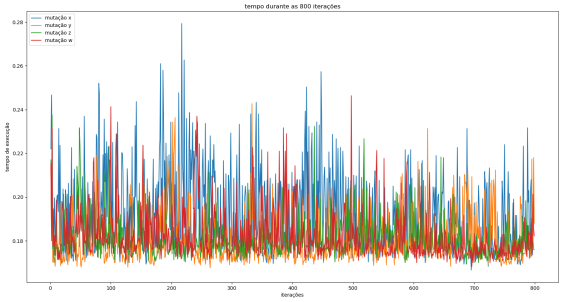

Figura 3. Iterações.

Na figura 3 é possível observar o tempo de execução do programa ASCI-Purple-logs. Os resultados mostram a aleatoriedade do algoritmo em escolher as otimizações, já que no gráfico não existe um crescimento contínuo. Portanto, para provocar um bom desempenho é necessário uma quantidade alta de iterações, de forma que aumentem as chances do algoritmo escolher um conjunto bom de otimizações.

IV. Conclusões

Devido as diversas otimizações possíveis, escolher um conjunto de otimizações que melhora a qualidade dos programas não é uma tarefa fácil. Desta forma, é possível aplicar heurísticas de busca para encontrar um bom conjunto de otimizações.

O presente trabalho mostrou que dentro do conjunto de otimizações O3, o algoritmo Random-Mutation Hill Climbing provoca melhor desempenho quando parâmetros são removidos, mostrando que a quantidade não traz qualidade ao programa.

Referências

[1] SEBESTA, R. W. Concepts of Programming Languages, 9th ed. AddisonWesley Publishing Company, USA, 2009.

[2] Zhao, J. JIKES RVM ADAPTIVE OPTIMIZATION SYSTEM WITH INTELLIGENT ALGORITHMS, Department of Computer Science, September 2004

[3] Santos, R. R. O IMPACTO DAS OTIMIZAÇÕES APLICADAS POR UM COMPILADOR OTIMIZANTE, Universidade Estadual de Maringá / Centro de Tecnologia, 2017