Universidade de São Paulo
Instituto de Ciências Matemáticas e Computação
Desenvolvimento de Código Otimizado - SSC 0951

Relatório Trabalho 3

Otimização do Compilador

1. Lucas G. Meneses Número: 13671615
2. Henrique S. Marques Número: 11815722
3. Carlos F. C. Lemos Número: 12542630

22 de novembro de 2023,
São Carlos, SP,
Brasil

Conteúdo

1 Introdução

Os compiladores desempenham um papel crucial na tradução do código-fonte de um programa para a linguagem de máquina executável, e ao longo das últimas décadas, têm evoluído significativamente para incorporar otimizações que melhoram o desempenho e a eficiência dos programas gerados. Entre os compiladores notáveis, destaca-se o GCC (*GNU Compiler Collection*), uma ferramenta de código aberto amplamente utilizada. As otimizações introduzidas por compiladores como o GCC visam aprimorar diversos aspectos do código, desde o uso eficiente dos recursos do processador até a redução do tempo de execução.

Uma das principais áreas de otimização reside na melhoria da eficiência do código gerado, buscando reduzir o número de instruções executadas sem comprometer a semântica do programa. O GCC utiliza uma variedade de técnicas, como *inlining* de funções, eliminação de código morto e propagação de constantes, para transformar o código-fonte em uma forma mais eficiente. Essas otimizações não apenas resultam em programas mais rápidos, mas também contribuem para a economia de recursos computacionais.

Além disso, o GCC incorpora estratégias avançadas de otimização durante a compilação, como a vetorização de *loops* e o agendamento de instruções, que exploram as capacidades específicas do *hardware* de destino. Isso permite que o compilador adapte o código gerado às características da arquitetura subjacente,

tirando proveito de recursos, como *pipelines* de instruções e unidades de execução paralela. Neste trabalho em específico, o objetivo foi a testagem de *flags* de otimização durante a compilação, com o objetivo de avaliar o impacto nas métricas de tempo de execução, tempo de compilação e tamanho do executável para dois programas selecionados do CLBG.

2 Metodologia

Para a realização de todos os experimentos, foi utilizado um computador rodando o Sistema Operacional Arch Linux, cuja CPU é um AMD Ryzen 7 5700X com frequência de *clock* de 3,4 GHz, 8 núcleos e 16 *threads*. Quanto à memória, o computador faz uso de 32 GB de RAM do tipo DDR4, cache L1 de 512 kB, L2 de 4 MB e L3 de 32 MB.

Nesse sentido, dois programas do CLBG (*Computer Language Benchmarks Game*) foram selecionados (nomearemos de *code1* e *code2*) para teste quanto ao tempo de compilação (Seção 3.1), tamanho do executável gerado (Seção 3.2) e tempo de execução (Seção 3.3). Tais códigos referem-se a dois modos diferentes de se calcular casas decimais do número pi (π), sendo adequados para testes de *benchmark*. Dessa maneira, foram compilados e executados 10 vezes, computamos a média aritmética e intervalo de confiança de cada métrica avaliada, para cada uma das seguintes *flags* de compilação:

- **O0** (sem otimizações): nenhuma otimização é ativada.
- **O1** (otimização leve): aplica otimizações que não aumentam significativamente o tempo de compilação, focadas em melhorar o desempenho sem comprometer muito o tempo de compilação.
- **O2** (otimização moderada): introduz otimizações adicionais, incluindo *inlining* de funções e reordenação de código, proporcionando melhorias substanciais no desempenho, mas com um aumento modesto no tempo de compilação.
- **O3** (otimização agressiva): aplica otimizações mais agressivas, como vetorização de *loops* e *inlining* mais agressivo, para maximizar o desempenho, embora isso possa aumentar significativamente o tempo de compilação.

- **Os** (otimização de tamanho): foca na redução do tamanho do executável, sacrificando potencialmente o desempenho. Isso é útil quando o tamanho do arquivo executável é crítico, por exemplo, em ambientes com restrições de armazenamento.

3 Resultados

3.1 Tempo de Compilação

A Figura 1 sumariza os resultados obtidos em relação ao tempo de compilação, variando a *flag* de otimização utilizada.

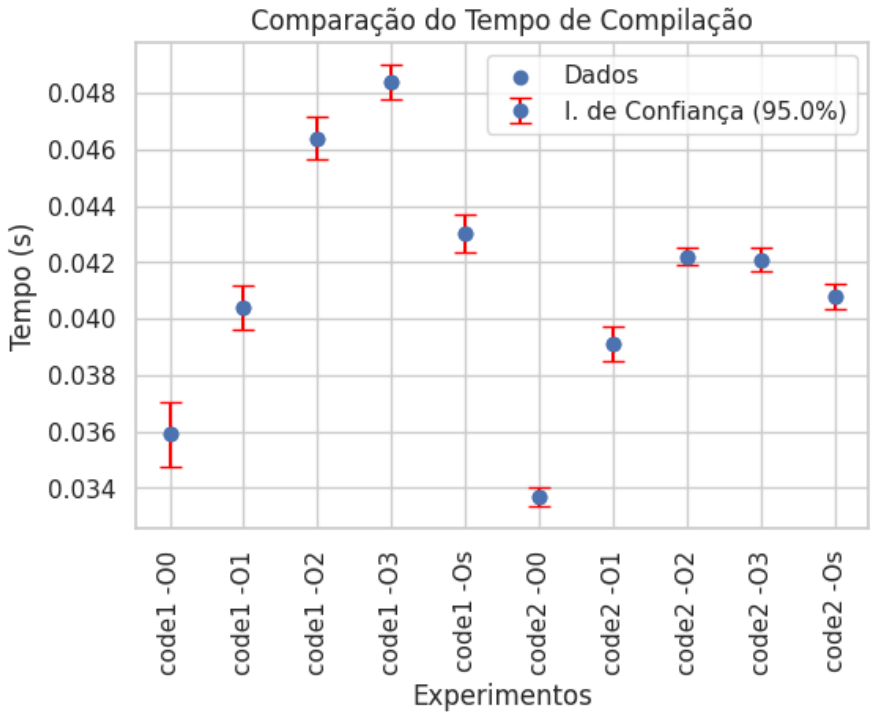

Figura 1: Tempos de compilação (s) em função dos códigos e das *flags* de otimização utilizadas. Os círculos azuis indicam a média dos dados obtidos para cada experimento e as barras vermelhas mostram o intervalo de confiança de 95 %.

Conforme esperado, os programas tiveram seu tempo de compilação aumentados ao se intensificar o nível de otimização exigido pelo compilador, em especial o *code1*,

cujo aumento foi mais significativo. No caso do *code2*, pode-se verificar que os tempos de compilação com as *flags* O2 e O3 foram estatísticamente equivalentes, apesar de maiores quando comparados às *flags* O0 e O1. Quanto à otimização de tamanho de executável (*flag* Os), obteve-se um tempo de compilação intermediário (entre O1 e O2) para ambos os programas.

3.2 Tamanho do Executável

A Figura 2 sumariza os resultados obtidos em relação ao tamanho do executável gerado, variando a *flag* de otimização utilizada.

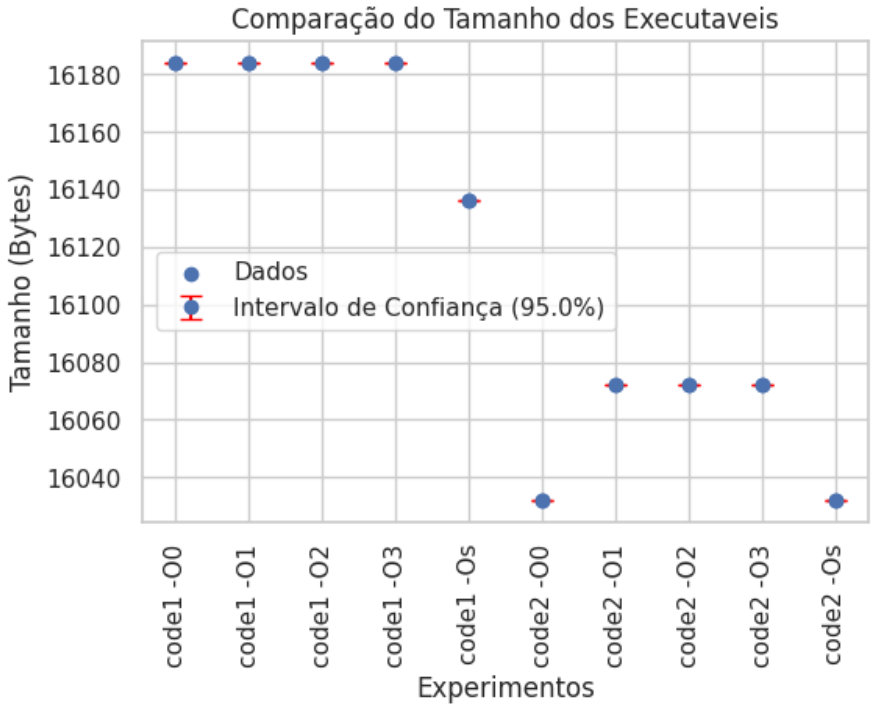

Figura 2: Tamanho dos executáveis gerados (kB) em função dos códigos e das *flags* de otimização utilizadas. Os círculos azuis indicam a média dos dados obtidos para cada experimento e as barras vermelhas mostram o intervalo de confiança de 95 %.

Em relação a esta métrica, as *flags* referentes à otimização de desempenho (O1, O2 e O3) não alteraram o tamanho dos executáveis em ambos os programas (aproximadamente 16,18 kB para *code1* e 16,07 kB para *code2*). Entretanto, tais *flags* geraram executáveis maiores para o *code2* quando comparadas à *flag* O0,

correspondente à nenhuma otimização, situação não verificada para *code1*, com mesmo tamanho de executável para as 4 *flags*. Por outro lado, a *flag* Os teve sucesso na redução do tamanho do executável para os programas, gerando os menores executáveis entre os experimentos (tamanho estatísticamente ”empatado”com O0, no *code2*).

3.3 Tempo de Execução

A Figura 3 sumariza os resultados obtidos em relação ao tempo de execução dos programas selecionados, variando a *flag* de otimização utilizada.

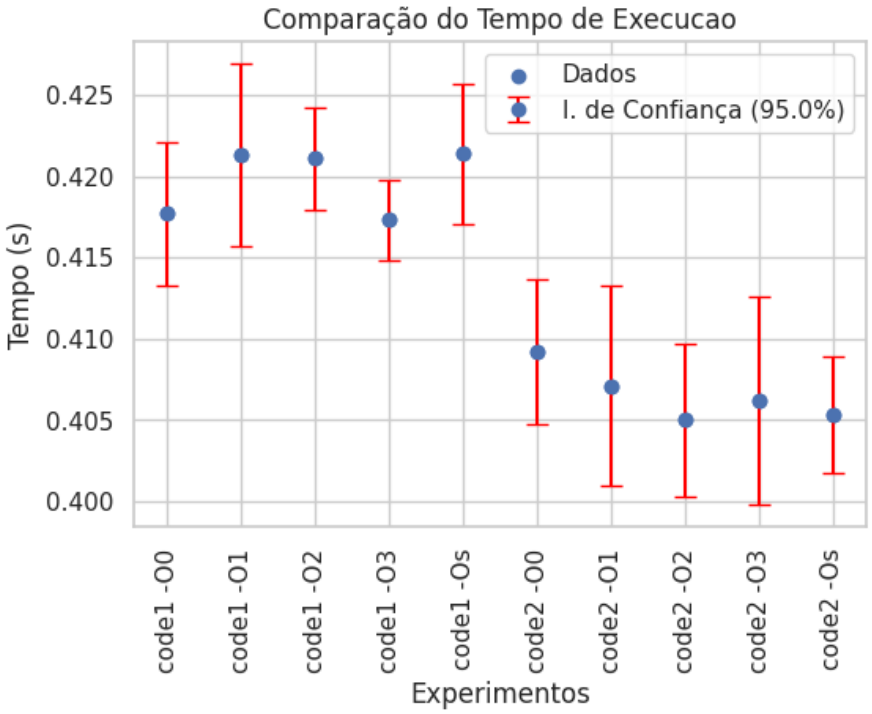

Figura 3: Tempos de execução (s) em função dos códigos e das *flags* de otimização utilizadas. Os círculos azuis indicam a média dos dados obtidos para cada experimento e as barras vermelhas mostram o intervalo de confiança de 95 %.

Ao se avaliar os intervalos de confiança obtidos com a execução repetida de cada um dos códigos, verificou-se que, para a arquitetura utilizada no experimento, as *flags* de otimização do compilador, bem como a *flag* O0 (sem otimização), não alteraram significativamente os tempos de execução de ambos os programas.

Apesar de contra-intuitivo, tal resultado pode estar relacionados à natureza robusta e eficiente do AMD Ryzen 7 5700X (processador empregado nos experimentos, conforme descrito na Seção 2) em lidar com uma variedade de cargas de trabalho. Sua arquitetura avançada, com 8 núcleos e 16 *threads*, combinada com caches L1, L2 e L3 generosas, pode mitigar a necessidade de otimizações agressivas. Se o código dos programas não explorar plenamente as capacidades da arquitetura, ou se os ganhos potenciais das otimizações forem ofuscados pela eficiência intrínseca da arquitetura, é possível que tais *flags* não tenham um impacto perceptível nos tempos de execução.

4 Conclusões

Os resultados obtidos neste trabalho fornecem *insights* valiosos sobre o impacto das *flags* de otimização do compilador GCC. Observou-se que, em relação ao tempo de compilação, o aumento do nível de otimização resultou em incrementos esperados nos tempos, sendo mais pronunciado para o código *code1*. Entretanto, em termos de tamanho do executável, as *flags* O1, O2 e O3 não influenciaram significativamente ambos os programas, enquanto a *flag* Os demonstrou eficácia na redução do tamanho do executável. Surpreendentemente, no que diz respeito aos tempos de execução, as *flags* de otimização, incluindo a *flag* O0 (sem otimização), não apresentaram alterações expressivas. Este resultado pode ser atribuído à eficiência intrínseca da arquitetura AMD Ryzen 7 5700X, cujas características robustas e avançadas podem reduzir a dependência de otimizações agressivas para alguns tipos de carga de trabalho, evidenciando a complexidade da relação entre otimizações de compilador e arquiteturas específicas.