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

Relatório Trabalho 1

Profilling: Loop Interchange e Loop Unrolling

1. Lucas G. Meneses Número: 13671615
2. Henrique S. Marques Número: 11815722
3. Carlos F. C. Lemos Número: 12542630
4. Eduardo S. Rocha Número: 11218692

16 de setembro de 2023,
São Carlos, SP,
Brasil

Conteúdo

1 Introdução

A atividade 1 da disciplina "Desenvolvimento de Código Otimizado" teve como objetivo introduzir conceitos relacionados à *Profilling* utilizando a ferramenta perf, bem como a análises estatísticas referentes a desempenho de código e duas técnicas de otimização de laços: *loop interchange* e *loop unrolling*, em cenários com alocação estática (AE) e dinâmica (AD).

O *perf* é uma ferramenta que se destaca por sua capacidade de aproveitar os contadores de desempenho disponíveis em CPUs modernas para realizar análises detalhadas do desempenho de aplicações. Permite que administradores e desenvolvedores coletem dados referentes a eventos de hardware e software, bem como tempos de execução de funções. Uma das aplicações mais notáveis do *perf* é o *profilling* de aplicações, permitindo a identificação de gargalos de desempenho, ineficiências de código e otimizações críticas. Através da captura precisa de eventos de desempenho, como ciclos de CPU, *branch predictions* e cache *misses*, o *perf* oferece uma visão profunda do comportamento do sistema, facilitando a melhoria da eficiência e a resolução de problemas de desempenho.

Quanto às técnicas de otimização de laços, o *loop interchange* e *loop unrolling* são amplamente utilizadas em programação para melhorar o desempenho de código. O *loop interchange* envolve a reorganização das iterações de um *loop*, muitas vezes para melhorar o acesso à memória e maximizar a eficiência da cache. Isso pode reduzir os tempos de espera por dados e melhorar significativamente o desempenho.

Por outro lado, o *loop unrolling* busca diminuir a sobrecarga de um *loop*, onde várias iterações são combinadas em uma única iteração maior. Isso pode reduzir instruções de controle do *loop* e melhorar o paralelismo, tornando o código mais eficiente. Ambas as técnicas são valiosas para otimizar laços em programas, adaptando-se a diferentes cenários e requisitos de desempenho.

2 Metodologia

Para a realização de todos os experimentos, foi utilizado um computador rodando o Sistema Operacional Ubuntu, cuja CPU é um Intel Core i5 6600k com frequência de *clock* de 3,5 GHz, 4 núcleos e 4 *threads*. Quanto à memória, o computador faz uso de 16 GB de RAM do tipo DDR4, cache L1 de dados e de instruções ambas 8-*way* com 4×32 kB de capacidade, L2 4-*way* 4×256 kB e L3 12-*way* de 6 MB.

Ademais, 6 seções de experimentos foram realizadas, todas utilizando multiplicação de matrizes quadradas de tamanho 800:

- Sem otimização, utilizando AE
- Sem otimização, utilizando AD
- Com *loop interchange*, utilizando AE
- Com *loop interchange*, utilizando AD
- Com *loop unrolling*, utilizando AE
- Com *loop unrolling*, utilizando AD

Em todas as seções, o código foi executado 10 vezes utilizando *perf* e as métricas observadas foram: (i) tempo de resposta; (ii) quantidade absoluta de cache *loads*; (iii) quantidade absoluta de cache *misses*; (iv) quantidade absoluta de instruções tipo *branch*; (v) quantidade absoluta de *branch-misses*. Dessa forma, a análise dos resultados (Seção 3) obtidos foi subdivida em duas partes distintas, a primeira destinada apenas aos tempos de resposta (Seção 3.1) e outra à análise de influência de fatores (Seção 3.2).

Para verificação dos tempos de resposta, as médias aritméticas das 10 amostras de cada seção foram calculadas juntamente de seus invervalos de confiança (95 %) para efeitos de comparação. Quanto à influência de fatores, todos os cálculos foram realizados, bem como os *plots* gerados, utilizando o *script* em *R* fornecido via e-Disciplinas.

3 Resultados

3.1 Tempos de Resposta

Como pode ser observado na Figura 1, a técnica de otimização *loop interchange* foi a que levou aos melhores resultados em relação ao tempo de resposta (tempo total de execução do programa), tanto com AD quanto com AE, sendo aproximadamente meio segundo mais rápido que a execução sem otimização com AE, a terceira com melhor desempenho. Neste caso, o tipo de alocação não trouxe mudanças significativas no tempo e, também, os valores permaneceram similares ao longo das 10 amostragens, fenômeno observado pelos intervalos de confiança pequenos.

Diferentemente do *loop interchange*, o *loop unrolling* apresentou diferenças significativas ao se alterar o método de alocação, tendo melhor desempenho com AE. O mesmo pôde ser observado quando nenhuma técnica de otimização foi aplicada. Nesse sentido, a melhora de desempenho da AE em comparação à AD é explicada pela melhor localidade espacial da AE, tendo em vista que o endereçamento é conhecido em tempo de compilação e, assim, há vantagens quanto ao aproveitamento da cache. Esta observação foi confirmada através da análise de influência de fatores discutida na Seção 3.2.

Ademais, a Figura 1 também indica que a técnica *loop unrolling* não trouxe mudanças relevantes nos tempos de resposta, em especial pela equivalência com o desempenho dos experimentos sem técnicas de otimização (claro, quando utilizado o mesmo modo de alocação), tendo em conta os intervalos de confiança sobrepostos.

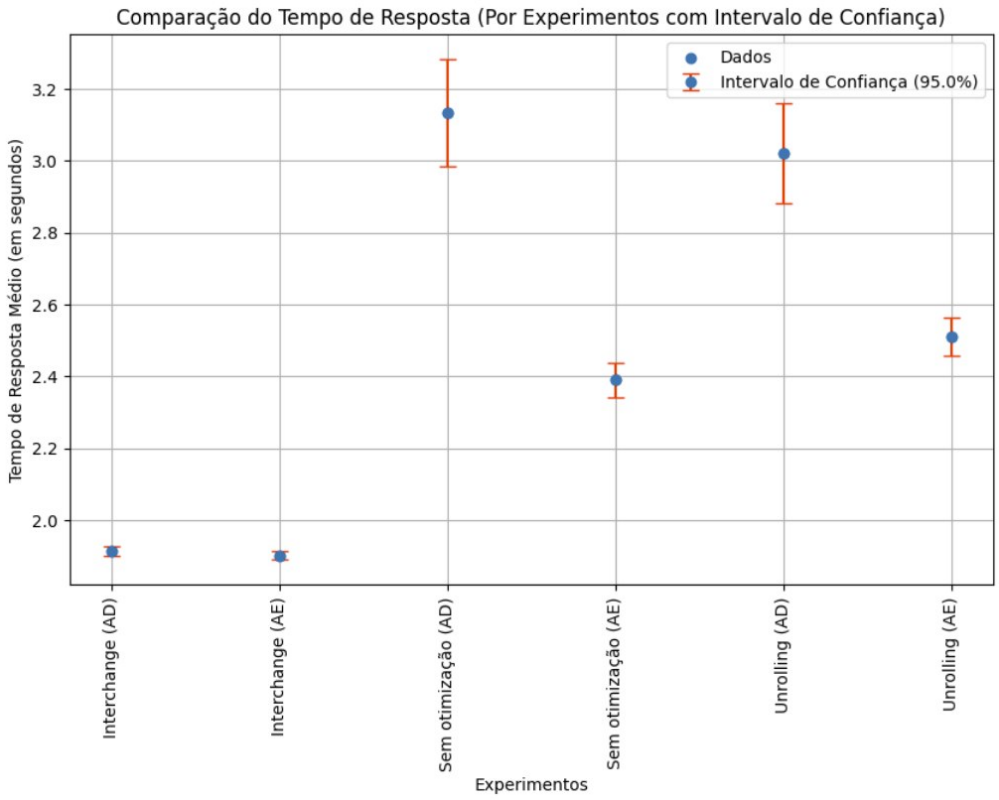


Figura 1: Tempos de resposta (s) em função do experimento realizado. Os círculos azuis indicam a média dos dados obtidos para cada experimento e as barras vermelhas mostram o intervalo de confiança de 95 %.

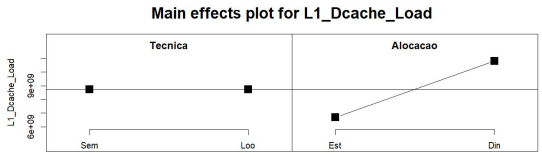
3.2 Análise de Influência de Fatores

Como pode ser visto pela Figura 2, a influência da técnica de otimização e do método de alocação varia de acordo com o parâmetro utilizado como referência. Nesse sentido, os gráficos podem ser facilmente interpretados ao avaliar a posição relativa entre os pontos: quanto mais similares as alturas, menos alterações a técnica de otimização ou de alocação causam no parâmetro observado.

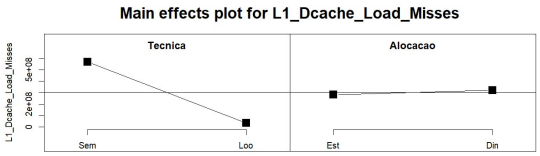
Pelo Gráfico 2a, vemos que não há diferenças significativas no número absoluto de cache *loads* quando o *loop interchange* é utilizado ou não. Entretanto, a AE apresentou um menor número de cache *loads*, por motivos previamente discutidos na Seção 3.1. Já em relação ao número absoluto de cache *misses* (Figura 2b), o uso do *loop interchange* foi responsável por diminuí-lo, enquanto a técnica de alocação

pouco influenciou. Este resultado pode explicar, majoritariamente, o motivo pelo qual esta técnica de otimização obteve os melhores tempos de resposta.

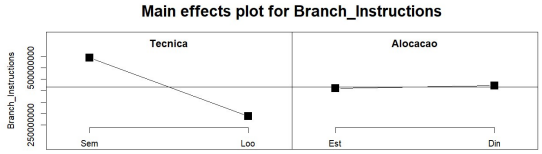
Em relação às *branches* (Figuras 2c e 2d), o número absoluto de instruções deste tipo diminuiu consideravelmente com o uso do *loop unrolling*, fenômeno completamente esperado tendo em vista o principal objetivo da técnica: diminuir a quantidade de iterações e, consequentemente, a quantidade de *branches*. Por outro lado, a quantidade de *branch misses* permaneceu constante mesmo com a utilização do *unrolling*, o que pode explicar, em partes, o porquê desta técnica não ter melhorado o desempenho do código de maneira significativa. Por fim, o modo de alocação foi influente no número de *branch misses* (cerca de 3000 *branch misses* a mais, em média, com uso de AD).



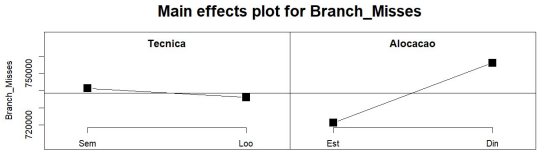
(a) Influência do fator cache *load* (*loop interchange*).



(b) Influência do fator cache *misses* (*loop interchange*).



(c) Influência do fator instruções *branch* (*loop unrolling*).



(d) Influência do fator *branch misses* (*loop unrolling*).

Figura 2: Análise de Influência de Fatores. Em cada figura da matriz, avaliamos se a técnica de otimização e/ou o modo de alocação interferiram nos fatores avaliados pelo estudo.

4 Conclusões

Neste estudo, investigamos o desempenho de técnicas de otimização de código, especificamente *loop interchange* e *loop unrolling*, em conjunto de diferentes métodos de alocação de memória, no contexto de tempo de resposta e de outros marcadores

de desempenho. Os resultados obtidos proporcionam alguns *insights* sobre o impacto dessas técnicas em vários aspectos.

A técnica de otimização *loop interchange* demonstrou ser altamente eficaz na melhoria do tempo de resposta do programa. Através da reorganização das estruturas de *loops*, conseguimos reduzir significativamente o tempo total de execução em relação à execução sem otimização, obtendo uma economia de aproximadamente meio segundo. Além disso, a alocação de memória, seja dinâmica (AD) ou estática (AE), não exerceu uma influência significativa no desempenho quando esta técnica foi aplicada, como evidenciado pela similaridade dos resultados ao longo das 10 amostragens. Além disso, a análise de influência de fatores revelou que o *loop interchange* teve um impacto positivo no número absoluto de cache *misses*, contribuindo para a melhoria do tempo de resposta.

Em contraste, o *loop unrolling* apresentou um impacto menos pronunciado no tempo de resposta do programa. Embora tenha reduzido significativamente o número absoluto de instruções *branch*, não conseguiu melhorar substancialmente o desempenho geral. O número de *branch misses* permaneceu praticamente inalterado, independentemente de ser aplicada ou não esta técnica.

Em resumo, este estudo ressalta a importância de escolher as técnicas de otimização apropriadas para um determinado contexto e considerar cuidadosamente o impacto do método de alocação de memória. Para programas sensíveis ao desempenho, a aplicação de técnicas como o *loop interchange* pode resultar em melhorias notáveis nos tempos de resposta, enquanto o *loop unrolling* pode ser mais adequado quando a redução no número absoluto de instruções *branch* é de suma importância.