Universidade de São Paulo

Instituto de Ciências Matemáticas e de Computação

Desenvolvimento de Código Otimizado 2023.2

Atividade 1: Loop unrolling e interchange

Gabriel da Cunha Dertoni	11795717
Matheus Ventura de Sousa	11345541
Pedro Lucas de Moliner de Castro	11795784
Vitor Caetano Brustolin	11795589

Profa. Dr. Sarita Mazzini Bruschi

São Carlos, Setembro de 2023

1 Introdução

O objetivo da presente atividade é analisar a eficácia das técnicas de otimização “loop unrolling” e “loop interchange”, utilizando como base para comparação o algoritmo de multiplicação de matrizes e como métricas o tempo de execução, o número de *loads* e *misses* na cache L1, o número total de *branches* e o número de *branch misses* médios para cada técnica.

Além das diferentes técnicas de otimização, como um fator extra, será feita uma comparação entre as formas de alocação estática e dinâmica dos dados. Apesar dessa distinção não ter sido feita na descrição do exercício, o grupo identificou duas formas diferentes de executar a alocação dinâmica: a primeira, denominada unidimensional, achatando a matriz para representá-la como um vetor e fazendo uma alocação por matriz e a segunda, chamada de bidimensional, fazendo alocações aninhadas e usando ponteiros para ponteiros. Como essas abordagens resultam em *layouts* de memória distintos, ambas serão testadas separadamente.

2 Códigos e execução

Para cada estratégia de alocação de memória, dentro do diretório `src/`, existe um subdiretório equivalente: alocação estática é `static/`, dinâmica unidimensional é `unidim/` e dinâmica bidimensional é `bidim/`.

Dentro desses subdiretório, existem arquivos fonte para cada estratégia de otimização: `naive.c` é o código não otimizado, `unrolling.c` aplica o “loop unrolling” e `interchange.c` aplica “loop interchange”.

Além desses, headers com funções comuns estão no diretório `include/` e scripts auxiliares para profiling e cálculo das estatísticas estão dentro de `scripts/`. Para facilitar a compilação e execução dos códigos, um arquivo `Makefile` foi desenvolvido com os seguintes comandos:

```
# Compila todos arquivos .c e executa todos binários
make [SIZE=512] [REPEATS=1]
make run

# Instala as dependências dos scripts .py
make requirements

# Gera o .json de profilings e calcula as estatísticas
make profile [RUNS=15] [PROFILING=out/profiling.json]
make stats [PROFILING=out/profiling.json]

# Deleta os arquivos gerados
make clean
```

O parâmetro *SIZE* define o número de linhas e colunas das matrizes, *REPEATS* dita quantas vezes a multiplicação será feita a cada execução do código, *RUNS* define quantas vezes cada binário será executado e *PROFILING* é o caminho para o .json de *profilings*, usado como saída em `make profile` e como entrada em `make stats`.

3 Medição dos dados

Para isolar somente a execução do algoritmo e descontar o tempo de alocação e *output*, o tempo de resposta foi medido diretamente em código utilizando a função `clock()`. As demais medidas, relacionados a acessos a cache e *branches*, foram coletadas com a ferramenta `perf`.

4 Ambiente de execução

Os valores exatos obtidos nas medições podem variar muito ao alterar o ambiente de execução. Nos testes realizados, foi utilizado o compilador **gcc 11**, com o nível mais baixo de otimização **-O0** e usando o standard **C11**. As especificações da CPU do computador onde os códigos foram executados são as seguintes:

Nome do modelo	Intel(R) Core(TM) i5-9400F CPU @ 2.90GHz
Arquitetura	x86_64
Modos operacionais da CPU	32-bit, 64-bit
Ordem dos bytes	Little Endian
CPUs	6
Threads por núcleo	1
Núcleos por soquete	6
Soquetes	1
Núcleos de NUMA	1
Frequência máxima	4100 MHz
Frequência mínima	800 MHz
Cache de L1d	192 KiB (6 instâncias)
Cache de L1i	192 KiB (6 instâncias)
Cache de L2	1.5 MiB (6 instâncias)
Cache de L3	9 MiB (1 instância)

5 Análise de Resultados

Para a execução dos testes, foram utilizadas matrizes de tamanho 512×512, uma multiplicação de matriz é efetuada a cada execução e cada binário foi executado 15 vezes para se obter a média, a mediana e um intervalo de confiança com nível de significância de 95%.

5.1 Comparação do tempo de resposta

Alocação de memória	Otimização	Média (ms)	Mediana (ms)
Estática	Sem otimização	477.15 ± 5.52	479.33
	Loop unrolling	423.18 ± 5.19	422.59
	Loop interchange	313.42 ± 0.80	313.02
Dinâmica unidimensional	Sem otimização	584.82 ± 4.81	578.79
	Loop unrolling	580.75 ± 6.52	580.02
	Loop interchange	416.30 ± 0.80	416.91
Dinâmica bidimensional	Sem otimização	520.72 ± 1.98	521.45
	Loop unrolling	480.43 ± 1.49	480.32
	Loop interchange	416.99 ± 1.44	416.57

Fazendo uma análise inicial, pode-se notar que, comparando diferentes formas de alocação de memória com a mesma técnica de otimização, do menor tempo de resposta para o maior a ordem sempre é alocação estática, alocação dinâmica bidimensional e alocação dinâmica unidimensional, com exceção do caso com “loop interchange” que não apresentou diferença significativa entre as formas de alocação dinâmica.

De forma similar, mantendo a alocação de memória constante e alterando a técnica de otimização, parece haver um vencedor claro quanto ao tempo

de resposta, sendo do menor até o maior o “loop interchange”, em seguida o “loop unrolling” e, por fim, o código sem otimização. Mais uma vez, há apenas um caso onde não há diferença significativa: a comparação entre nenhuma otimização e “loop unrolling” utilizando alocação dinâmica unidimensional.

O fato de a alocação dinâmica bidimensional ter performado melhor que a unidimensional foi bastante surpreendente para o grupo, pois, antes de analisar o resultado, havia-se teorizado que a unidimensional levaria a execução mais veloz por apresentar menos indireções e melhor compactação dos dados, o que beneficiaria a localidade de cache.

5.2 Estatísticas de branches e cache loads

Tabela 1: Média, mediana e intervalo de confiança de instruções de *branch*

Alocação de memória	Otimização	Média	Mediana
Estática	Sem otimização	$146520549.20 \pm 40653.62$	146512760.00
	Loop unrolling	45897929.60 ± 72965.92	45868043.00
	Loop interchange	$146457081.20 \pm 29117.87$	146454782.00
Dinâmica unidimensional	Sem otimização	146017227.93 ± 4633.46	146015448.00
	Loop unrolling	45380074.13 ± 34673.84	45355001.00
	Loop interchange	145984466.00 ± 8402.40	145979214.00
Dinâmica bidimensional	Sem otimização	$146067490.20 \pm 21195.02$	146054432.00
	Loop unrolling	45381980.33 ± 7421.09	45379215.00
	Loop interchange	$146039129.20 \pm 17002.64$	146030740.00

Tabela 2: Média, mediana e intervalo de confiança de *branch misses*

Alocação de memória	Otimização	Média	Mediana
Estática	Sem otimização	294943.47 ± 999.96	294253.00
	Loop unrolling	296709.87 ± 3840.98	294656.00
	Loop interchange	293131.93 ± 801.89	292886.00
Dinâmica unidimensional	Sem otimização	295747.93 ± 397.79	295687.00
	Loop unrolling	296889.33 ± 1975.25	295839.00
	Loop interchange	294570.33 ± 757.56	294190.00
Dinâmica bidimensional	Sem otimização	294831.87 ± 1424.26	293889.00
	Loop unrolling	293827.67 ± 526.85	293536.00
	Loop interchange	293684.93 ± 1236.20	292980.00

Fazendo uma análise inicial simples, o único fator que parece apresentar uma redução significativa no número total de *branches* é a aplicação do "loop unroling", o que faz bastante sentido por ser a métrica que essa otimização ataca. No entanto, essa redução não parece ser refletida de forma relevante no número de *branch misses*, sendo que, nessa métrica, é o "loop interchange" que apresenta os menores valores de média e mediana, apesar de não ser uma diferença tão grande quando os intervalos de confiança são levados em consideração.

Tabela 3: Média, mediana e intervalo de confiança de *loads* na cache L1

Alocação de memória	Otimização	Média	Mediana
Estática	Sem otimização	$1760537774.93 \pm 67041.17$	1760520319.00
	Loop unrolling	$1559255818.60 \pm 111342.70$	1559228118.00
	Loop interchange	$1760433644.53 \pm 47751.99$	1760431460.00
Dinâmica unidimensional	Sem otimização	$2296695850.60 \pm 6928.30$	2296691333.00
	Loop unrolling	$2095411410.27 \pm 55341.41$	2095370700.00
	Loop interchange	$2296642585.33 \pm 13901.18$	2296632965.00
Dinâmica bidimensional	Sem otimização	$2834141624.07 \pm 35547.90$	2834118888.00
	Loop unrolling	$2632778800.53 \pm 12361.72$	2632774017.00
	Loop interchange	$2834094833.40 \pm 28721.45$	2834079594.00

Com base somente nos valores dessa tabela, já é possível notar uma clara crescente no número de *cache loads* quando alternando de alocação estática, para dinâmica unidimensional para dinâmica bidimensional. Isso provavelmente se dá devido ao aumento de indireções e acessos a ponteiros no código. Curiosamente, a técnica de "loop unrolling" apresentou menores médias e medianas em qualquer tipo de alocação, mas também apresentou o maior intervalo nos casos estático e dinâmico unidimensional.

Tabela 4: Média, mediana e intervalo de confiança de *load misses* na cache L1

Alocação de memória	**Otimização**	**Média**	**Mediana**
Estática	Sem otimização	$134735233.73 \pm 35239.56$	134721494.00
	Loop unrolling	134765401.67 ± 9950.16	134767672.00
	Loop interchange	8565159.87 ± 4249.72	8562516.00
Dinâmica unidimensional	Sem otimização	$134487381.00 \pm 44714.91$	134434610.00
	Loop unrolling	$134520132.20 \pm 46625.00$	134565836.00
	Loop interchange	8565902.47 ± 1935.95	8564808.00
Dinâmica bidimensional	Sem otimização	$151064002.20 \pm 120879.11$	151085444.00
	Loop unrolling	$143028283.00 \pm 107533.04$	143004629.00
	Loop interchange	8636143.73 ± 4762.56	8632453.00

Finalmente, com relação ao número de *cache misses*, ele parece ser um pouco maior com a alocação dinâmica bidimensional quando comparado às outras duas formas de alocação, no entanto, a disparidade mais significativa é, com certeza, a diminuição desses erros ao aplicar a técnica de "loop interchange". Novamente, essa redução parece óbvia ao considerar que esse é o fator que tal técnica de otimização tenta melhorar. Essa diminuição em até duas ordens de magnitude pode ser a causa dessa técnica ter resultado nos melhores tempos de resposta.

5.3 Análise da influência da variação dos fatores

Apesar de que a análise inicial dos valores de média e mediana colocados nas tabelas anteriores indica algumas tendências e levanta algumas suspeita de quais fatores são mais significantes para cada métrica, ainda é necessário provar de forma rigorosa a magnitude dessas influências. Para isso, serão realizados 4 testes com projeto fatorial 2^2, onde um dos fatores será sempre a utilização de memória estática ou uma das variações de memória dinâmicas e o outro a aplicação ou não de uma técnica de otimização.

Dado que as variações da alocação dinâmica não faziam parte da descrição original do exercício, e para manter a objetividade e simplicidade dessa atividade, nenhuma análise de influência comparando diretamente as duas formas de alocação dinâmica foi desenvolvida.

5.3.1 Aplicação ou não do "loop unrolling" com memória estática ou dinâmica unidimensional influenciando *branches*

Nessa análise, considera-se o fator A como sendo a utilização de memória dinâmica, onde -1 ou E significa a utilização de memória estática e 1 ou U representa a memória dinâmica unidimensional, e o fator B como a aplicação da otimização, onde -1 ou N é nenhuma otimização e 1 ou LU é a aplicação do "loop unrolling". Como variáveis de resposta, serão observados o tempo de resposta do programa, a quantidade de instruções de *branch* e a quantidade de *branch misses*.

Fatores		Variáveis de resposta		
A (Alocação)	**B (Otimização)**	**Tempo de resposta**	**Instruções de *branch***	***Branch misses***
$-1(E)$	$-1(N)$	477.15	146520549.20	294943.47
$1(U)$	$-1(N)$	584.82	146017227.93	295747.93
$-1(E)$	$1(LU)$	423.18	45897929.60	296709.87
$1(U)$	$1(LU)$	580.75	45380074.13	296889.33

Parâmetro	**Média estimada**		
	Tempo de resposta	**Instruções de *branch***	***Branch misses***
q_0	516.48	95953945.22	296072.65
q_A	66.31	-255294.18	245.98
q_B	-14.51	-50314943.35	726.95
q_{AB}	12.48	-3633.55	-156.25

Soma dos quadrados	**Variáveis de resposta**		
	Tempo de resposta	**Instruções de *branch***	***Branch misses***
SST	19052.73	$1.00 \cdot 10^{17}$	2453506.10
SSA	17588.06	$2.60 \cdot 10^{11}$	242024.64
SSB	842.16	$1.00 \cdot 10^{17}$	2113825.21
$SSAB$	622.50	$5.28 \cdot 10^{7}$	97656.25

Parâmetro	**Variação**		
	Tempo de resposta	**Instruções de *branch***	***Branch misses***
q_A	0.92	$2.57 \cdot 10^{-5}$	0.10
q_B	0.05	0.99	0.86
q_{AB}	0.03	$5.00 \cdot 10^{-9}$	0.04

De acordo com a tabela de variância, observa-se que a forma alocação de memória (fator A) é o mais influente no tempo de resposta, já a aplicação do “loop unrolling” (fator B) é muito influente tanto na quantidade total de instruções de *branch* quanto no número de *branch misses*. A interação dos fatores, entretanto, não apresenta resultado considerável.

5.3.2 Aplicação ou não do "loop unrolling" com memória estática ou dinâmica bidimensional influenciando *branches*

Para esse experimento, considera-se a utilização de memória dinâmica, onde -1 ou E significa a utilização de memória estática e 1 ou B representa a memória dinâmica bidimensional, como o fator A e a aplicação da otimização, onde -1 ou N é nenhuma otimização e 1 ou LU é a aplicação do "loop unrolling", como o fator B. As variáveis de resposta serão o tempo de resposta do programa, a quantidade de instruções de *branch* e a quantidade de *branch misses* novamente.

Fatores		**Variáveis de resposta**		
A (Alocação)	**B (Otimização)**	**Tempo de resposta**	**Instruções de *branch***	***Branch misses***
$-1(E)$	$-1(N)$	477.15	146520549.20	294943.47
$1(B)$	$-1(N)$	520.72	146067490.20	294831.87
$-1(E)$	$1(LU)$	423.18	45897929.60	296709.87
$1(B)$	$1(LU)$	480.43	45381980.33	293827.67

Parâmetro	**Média estimada**		
	Tempo de resposta	**Instruções de *branch***	***Branch misses***
q_0	475.37	95966987.33	295078.22
q_A	25.21	-242252.07	-748.45
q_B	-23.57	-50327032.37	190.55
q_{AB}	3.42	-15722.57	-692.65

Soma dos quadrados	**Variáveis de resposta**		
	Tempo de resposta	**Instruções de *branch***	***Branch misses***
SST	4809.19	$1.01 \cdot 10^{16}$	4305002.91
SSA	2541.17	$2.34 \cdot 10^{11}$	2240709.61
SSB	2221.24	$1.01 \cdot 10^{16}$	145237.21
$SSAB$	46.79	$9.89 \cdot 10^{8}$	1919056.09

Parâmetro	Variação		
	Tempo de resposta	**Instruções de *branch***	***Branch misses***
q_A	0.53	$2.30 \cdot 10^{-5}$	0.52
q_B	0.46	0.99	0.03
q_{AB}	0.01	$9.00 \cdot 10^{-8}$	0.45

Tendo em vista esses resultados, ambos fatores influenciam fortemente o tempo de resposta, mas sua combinação não apresenta grande relevância. Já o número de instruções de *branch* é definido quase inteiramente pela aplicação do "loop unrolling" (fator B), enquanto o número de *branch misses* é afetado de forma significativa tanto pela alocação de memória (fator A), quanto pela combinação dos fatores.

5.3.3 Aplicação ou não do "loop interchange" com memória estática ou dinâmica unidimensional influenciando *loads* na cache L1

Novamente, considera-se o fator A como sendo a utilização de memória dinâmica, onde -1 ou E significa a utilização de memória estática e 1 ou U representa a memória dinâmica unidimensional, já o fator B será a aplicação da otimização, onde -1 ou N é nenhuma otimização e 1 ou LI é a aplicação do "loop interchange". Como variáveis de resposta, aqui serão observados o tempo de resposta do programa e as quantidade de *loads* e *load misses* na cache L1.

Fatores		Variáveis de resposta		
A (Alocação)	**B (Otimização)**	**Tempo de resposta**	***Loads* na cache L1**	***Load misses* na cache L1**
$-1(E)$	$-1(N)$	477.15	1760537774.93	134735233.73
$1(U)$	$-1(N)$	584.82	2296695850.60	134487381.00
$-1(E)$	$1(LI)$	313.42	1760433644.53	8565159.87
$1(U)$	$1(LI)$	416.30	2296642585.33	8565902.47

Parâmetro	Média estimada		
	Tempo de resposta	*Loads* na cache L1	*Load misses* na cache L1
q_0	447.92	2028577464.00	71588419.27
q_A	52.64	268091754.10	-61777.53
q_B	-83.06	-39348.92	-63022888.10
q_{AB}	-1.20	12716.28	62148.83

Soma dos quadrados	Variáveis de resposta		
	Tempo de resposta	*Loads* na cache L1	*Load misses* na cache L1
SST	38686.08	$2.87 \cdot 10^{17}$	$1.59 \cdot 10^{16}$
SSA	11082.83	$2.87 \cdot 10^{17}$	$1.53 \cdot 10^{10}$
SSB	27597.52	$6.19 \cdot 10^{9}$	$1.59 \cdot 10^{16}$
$SSAB$	5.74	$6.47 \cdot 10^{8}$	$1.54 \cdot 10^{10}$

Parâmetro	Variação		
	Tempo de resposta	*Loads* na cache L1	*Load misses* na cache L1
q_A	0.29	0.99	0
q_B	0.71	$2.15 \cdot 10^{-8}$	1
q_{AB}	$1.50 \cdot 10^{-4}$	$2.25 \cdot 10^{-9}$	0

Considerando a tabela final, observa-se que o tempo de resposta é significativamente afetado pela otimização do “loop interchange” (fator B), mas também é influenciado pela forma de alocação de memória (fator A). O número de *loads* na cache L1 é influenciado majoritariamente pela alocação de memória (fator A) e o número de *load misses* pela otimização (fator B). A combinação dos fatores não apresenta diferença relevante em nenhuma das variáveis de resposta.

5.3.4 Aplicação ou não do “loop interchange” com memória estática ou dinâmica bidimensional influenciando *loads* na cache L1

Para a última avaliação, considera-se a utilização de memória dinâmica, onde -1 ou E significa a utilização de memória estática e 1 ou B representa a memória

dinâmica bidimensional, como o fator A e a aplicação da otimização, onde -1 ou N é nenhuma otimização e 1 ou LI é a aplicação do "loop interchange", como o fator B. As variáveis de resposta serão novamente o tempo de resposta do programa e as quantidades de *loads* e *load misses* na cache L1.

Fatores		**Variáveis de resposta**		
A (Alocação)	**B (Otimização)**	**Tempo de resposta**	***Loads* na cache L1**	***Load misses* na cache L1**
$-1(E)$	$-1(N)$	477.15	1760537774.93	134735233.73
$1(B)$	$-1(N)$	520.72	2834141624.07	151064002.20
$-1(E)$	$1(LI)$	313.42	1760433644.53	8565159.87
$1(B)$	$1(LI)$	416.99	2834094833.40	8636143.73

Parâmetro	**Média estimada**		
	Tempo de resposta	***Loads* na cache L1**	***Load misses* na cache L1**
q_0	432.07	2297301969.00	75750134.88
q_A	36.79	536816259.50	4099938.08
q_B	-66.87	-37730.27	-67149483.08
q_{AB}	15.00	14334.93	-4064446.15

Soma dos quadrados	**Variáveis de resposta**		
	Tempo de resposta	***Loads* na cache L1**	***Load misses* na cache L1**
SST	24196.26	$1.15 \cdot 10^{18}$	$1.82 \cdot 10^{16}$
SSA	5412.54	$1.15 \cdot 10^{18}$	$6.72 \cdot 10^{13}$
SSB	17883.71	$5.70 \cdot 10^{9}$	$1.80 \cdot 10^{16}$
$SSAB$	900.00	$8.22 \cdot 10^{8}$	$6.61 \cdot 10^{13}$

Parâmetro	**Variação**		
	Tempo de resposta	***Loads* na cache L1**	***Load misses* na cache L1**
q_A	0.22	0.99	$3.70 \cdot 10^{-3}$
q_B	0.74	$4.94 \cdot 10^{-9}$	0.99
q_{AB}	0.04	$7.13 \cdot 10^{-10}$	$3.64 \cdot 10^{-3}$

Observando a ultima tabela, conclui-se que o tempo de resposta é afetado principalmente pela aplicação do "loop interchange" (fator B), mas também é

influenciado pela alocação de memória (fator A). Novamente, o número de *loads* na cache L1 é explicado quase inteiramente pela alocação de memória (fator A), o número de *load misses* é explicado pela otimização (fator B) e a combinação dos fatores não tem influência relevante.

6 Conclusão

Como esperado, a otimização de “loop unrolling” reduz significativamente o número de instruções de *branch*, mas não está necessariamente ligada ao número de *branch misses*, tendo um caso onde essa variável é afetada pela alocação de memória e pela combinação dos fatores somente. A otimização de “loop interchange” provoca uma diminuição muito significativa no número de *load misses* na cache L1, porém o número total de *loads* solicitados nessa cache depende somente da forma de alocação de memória, sendo maior nos casos dinâmicos do que no estático.

Sobre o tempo de resposta, o fator que provocou uma diminuição mais significativa foi a aplicação do “loop interchange”, o “loop unrolling” só demonstrou relevância na comparação com a memória dinâmica bidimensional, mas foi irrelevante no caso com a unidimensional; além disso, em todos os casos a forma de alocação de memória foi importante, resultando em tempos menores com a alocação estática. Finalmente, foi possível verificar que, com exceção do experimento da memória dinâmica unidimensional com “loop unrolling”, os fatores da forma de alocação de memória e da aplicação ou não de otimizações são independentes, pois sua combinação não apresenta influência nas variáveis de resposta.