Universidade Federal da Paraíba
Programa de Pós-Graduação em Informática
Arquitetura de Computadores

Redução de cache miss através da otimização de software

Autor: Frederico de Souza Guerra

Durante o desenvolvimento de softwares é comum o desenvolver ter um foco maior em garantir que a aplicação seja utilizável e entregue no prazo determinado. No entanto, a forma como o software é desenvolvido pode drasticamente impactar na sua usabilidade e desempenho. Quando o assunto é performance, a memória cache tem um papel crucial nisso e o seu desempenho pode ser definido como:

$$CPU_{tempo} = IC * (CPI_{executions} * (\frac{MemAccess}{Inst}) * (HT * MR * MP)) * CT \qquad (1)$$

onde:

- IC : Número de instruções
- CPI executions : ciclos por instrução
- Memacess : quantidade de instruções que acessam a memória
- HT: tempo para acessar dado que está em cache
- MR: taxa de Cache Miss
- MP: tempo perdido a cada Cache Miss
- CT: tempo de ciclo

Esse trabalho objetiva avaliar 4 diferentes técnicas de otimização de cache miss através de código: Mesclagem de arrays, Permuta de loops, Fusão de loop e Blocagem. Para avaliar cada uma dessas técnicas, inicialmente foi desenvolvido um código, simples e sem zelo por boas práticas, na linguagem C++ para multiplicar duas matrizes A e B quadradas de ordem N. Todos os elementos da Matriz A são elevados ao expoente 2.1, enquanto os elementos de B são iguais aos correspondentes de A elevados a 2.3 e divididos pela valor numérico da posição da coluna da matriz adicionado a 1.

```
int original(int N){

    double A[N][N], B[N][N], MULT[N][N];
    int i, j;

    for(j = 0; j < N; ++j){
        for(i = 0;i< N; ++i){
            MULT[i][j] = 0;
            A[i][j] = pow(i, 2.1);
            B[i][j] = pow(A[i][j], 2.3)/(j + 1);
            MULT[i][j] = A[i][j] * B[i][j];
```

```
        }
    }
    return 1;
}
```

A primeira técnica avaliada foi a Mesclagem de Arrays objetivando melhorar a localidade espacial das matrizes A e B. A principal alteração realizada foi na forma de criação das matrizes.

Versão inicial:

```
double A[N][N], B[N][N], MULT[N][N];
```

Versão alterada(Técnica 1):

```
struct DataStruct{
        double A, B, MULT;};
    DataStruct data[N][N];
```

A segunda técnica aplicada foi a alteração da ordem do laço aninhados. Originalmente o código varre todas as linhas i para cada coluna j, no entanto, após a alteração espera-se uma melhor localidade espacial.

Versão inicial:

```
for(j = 0; j < N; ++j){
        for(i=0;i<N;++i){...}}
```

Versão alterada(Técnica 2):

```
for(i = 0; i < N; ++i){
        for(j=0;j<N;++j){...}}
```

A terceira consiste em uma combinação das técnicas 1 e 2. Já a quarta é uma aversão à fusão de loops do código inicial, ou seja, possui 4 aninhamentos ao invés de um só. A quinta técnica é uma combinação das 1, 2 e 4.

Versão inicial:

```
for(j = 0; j < N; ++j){
        for(i = 0;i< N; ++i){
            MULT[i][j] = 0;
            A[i][j] = pow(i, 2.1);
            B[i][j] = pow(A[i][j], 2.3)/(j + 1);
            MULT[i][j] = A[i][j] * B[i][j];
        }
    }
```

Versão alterada(Técnica 4):

```
for(i = 0; i < N; ++i){
        for(j=0;j<N;++j){
            A[i][j] = pow(i,2.1);
        }
    }
for(i = 0; i < N; ++i){
```

```
    for(j=0;j<N;++j){
        B[i][j] = pow(A[i][j],2.3)/(j+1);
        }
    }
for(i = 0; i < N; ++i){
    for(j=0;j<N;++j){
        MULT[i][j] = 0;
    }
}
for(i = 0; i < N; ++i){
    for(j=0;j<N;++j){
        MULT[i][j] = A[i][j] * B[i][j];
    }
}
```

Foram realizados experimentos com tamanho amostral de 30 execuções para cada combinação de técnica avaliada e dimensão das matrizes de entrada para multiplicação. As dimensões consideradas das matrizes na avaliação foram 100, 250, 500, 1000, 5000 e 10000.

A tabela 1 mostra a média, desvio padrão e % de redução no tempo de execução em relação a versão original para cada técnica dadas as dimensões das matrizes A e B de 10000x10000.

Tabela 1. μ, σ e % de redução dos tempos de execução de cada técnica considerando as matrizes quadradas de entrada 10000x10000 e 30 repetições.

Técnica	**Código**	**Média (μ)**	**Desvio padrão (σ)**	**% Redução**
Técnica 0	Versão Original	5.08	0.30	-
Técnica 1	Mesclagem de Arrays	4.65	0.13	8.5
Técnica 2	Permuta de loops	4.43	0.11	12.8
Técnica 3	Mesclagem de Arrays + Permuta de loops	4.60	0.11	9.4
Técnica 4	Desagregação de loops	3.89	0.11	23.4
Técnica 5	Desagregação de loops + Mesclagem de Arrays + Permuta de loops	4.15	0.11	18.3

A figura 1 mostra o aumento na diferença entre os tempos de execução de cada técnica a medida que o tamanho da entrada cresce. Nota-se que a desfusão de loops foi a técnica que obteve os melhores resultados de desempenho no tempo de execução. Essa diferença é ainda mais notório a medida que o tamanho das matrizes de entrada aumenta.

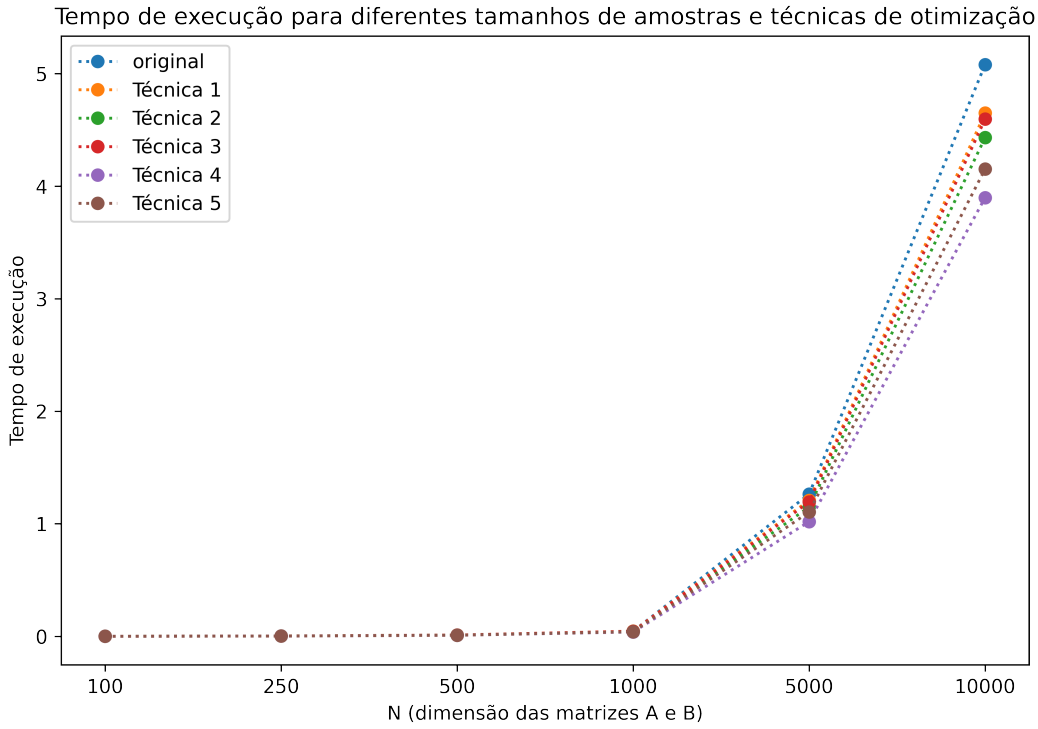

Figura 1. Comparação do tempo de execução do código original e após a aplicação das técnicas de otimização para diferentes tamanhos das matrizes A e B.

As figuras 2 e 3 detalham as distribuições dos tempos de execução para cada uma das técnicas, considerando N igual a 100 e 10000, respectivamente. Nota-se que para um N pequeno a mesclagem de array e permuta de loops não impactam positivamente no tempo de execução em relação à versão do código original. No entanto, quando N toma proporções maiores essas duas técnicas tornam o código mais eficiente, principalmente a permuta de loops.

Percebe-se também que, independente do tamanho das matrizes de entrada, a desagregação de loops tornou o código mais otimizado.

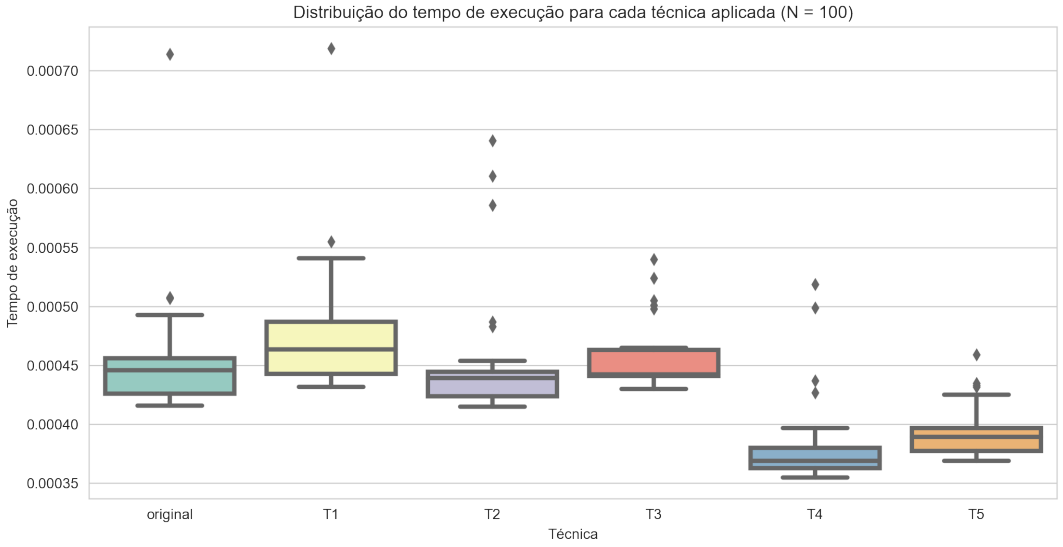

Figura 2. Comparação do tempo de execução do código original e após a aplicação das técnicas de otimização para matrizes 100x100.

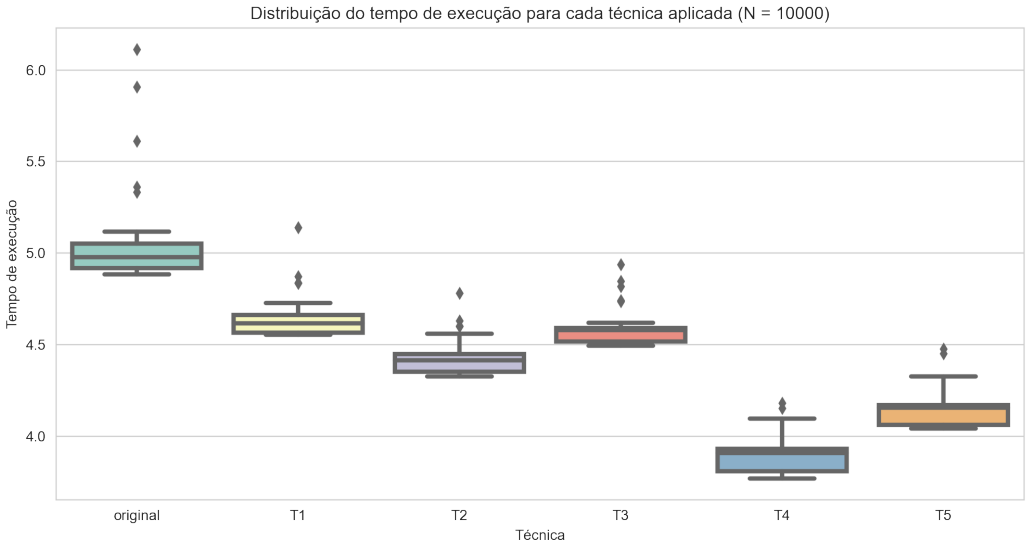

Figura 3. Comparação do tempo de execução do código original e após a aplicação das técnicas de otimização para matrizes 10000x10000

Referências

1. C. Pancratov, J. M. Kurzer, K. A. Shaw and M. L. Trawick, "Why Computer Architecture Matters: Memory Access," in Computing in Science Engineering, vol. 10, no. 4, pp. 71-75, July-Aug. 2008, doi: 10.1109/MCSE.2008.106.

2. C. Pancratov, K. Shaw, M. Trawick and J. Kurzer, "Why Computer Architecture Matters" in Computing in Science Engineering, vol. 10, no. 03, pp. 59-63, 2008. doi: 10.1109/MCSE.2008.87

3. C. Pancratov, J. M. Kurzer, K. A. Shaw and M. L. Trawick, "Why Computer Architecture Matters: Thinking through Trade-offs in Your Code," in Computing in Science Engineering, vol. 10, no. 5, pp. 74-79, Sept.-Oct. 2008, doi: 10.1109/MCSE.2008.126.