MAC 0219/5742 - Introdução à Computação Paralela e Distribuída

Mini EP 5 - Otimizando o uso de *cache* - parte 1

Alfredo Goldman e Vitor Terra

1. Introdução

A memória *cache* é uma memória de rápido acesso, localizada dentro dos processadores. Sua velocidade de acesso em comparação à RAM chega a ser da ordem de centenas de vezes mais rápida por: estar fisicamente muito mais próxima da CPU; e ser implementada usando SRAM (*static random-access memory*) que, embora tenha maior custo de produção por *byte*, permite a construção de memórias mais rápidas[1].

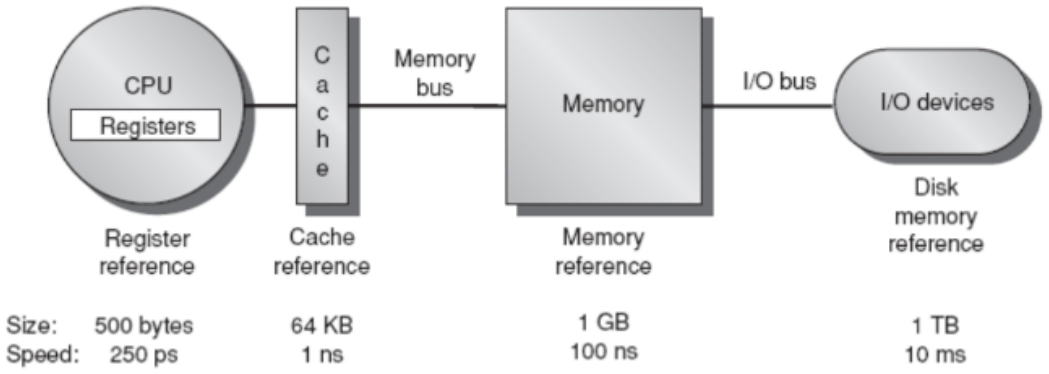

Figura 1: Comparação aproximada entre tamanho e tempo de acesso de diferentes formas de armazenamento de dados (registradores, cache, memória RAM e disco)[2].

Segundo o princípio de localidade espacial, programas tendem a acessar dados e instruções próximos àqueles recentemente utilizados (por exemplo, ao se iterar sobre os elementos de um vetor em um laço). Tendo isso em vista, ao se acessar um dado na memória principal, também se trazem os dados de endereços consecutivos para serem armazenados no *cache*.

Quando um dado solicitado pela CPU está presente no *cache* (*cache hit*), tal dado é obtido sem necessidade de trazê-lo da memória principal. Caso contrário, os dados presentes no cache são invalidados e substituídos por um novo bloco de dados obtido da memória principal (*cache miss*). De modo geral, quanto maior a taxa de *cache hits*, melhor o desempenho de um programa.

---

[1] What Every Programmer Should Know About Memory (freebsd.org)

[2] Hennessy, Patterson - Computer Architecture - A quantitative approach - cap. 5

Assim, o padrão de acesso aos dados em um programa tem impacto significativo no seu tempo de execução, o que será verificado na prática neste mini EP.

2. Problema

Considere o seguinte código (multiplicação de matrizes n × n), presente na função `matrix_dgemm_0` no arquivo fornecido `matrix.c`:

```
for (i = 0; i < n; ++i)
     for (j = 0; j < n; ++j) {
          double sum = 0;
          for (k = 0; k < n; ++k)
               sum += A(i, k)*B(k, j);
          C(i , j) = sum;
     }
```

Nesse contexto, DGEMM significa "Double Precision General Matrix Multiplication", nomenclatura utilizada em pacotes de álgebra linear de alto desempenho. Seu objetivo é utilizar os conhecimentos recém adquiridos a respeito da *cache* para otimizar o código fornecido.

Você deverá implementar na função `matrix_dgemm_1` uma versão mais rápida (e ainda correta) do código acima, usando apenas as noções de localidade de acesso à memória *cache* vistas em aula. Lembre-se de que a ordem de iteração sobre uma matriz (coluna depois linha ou linha depois coluna) pode acabar invalidando o *cache*. Assim, considerando a posição dos elementos das matrizes na memória, procure uma forma de garantir que a iteração sobre A e B, no código acima, possa melhor aproveitar o *cache*.

3. Código inicial

Foi fornecido no e-Disciplinas um programa base para a codificação do mini EP (arquivo `src_miniep5.zip`). Você deverá modificar apenas a função `matrix_dgemm_1` no arquivo `matrix.c` com a sua versão otimizada de `matrix_dgemm_0`. O programa fornecido contém testes de correção e de tempo de execução, nos quais a sua implementação de `matrix_dgemm_1` deverá passar.

Para executar os testes, use o comando

```
$ make test
```

no terminal, com seu *shell* aberto na pasta contendo o `Makefile`. Se o teste com a `matrix_dgemm_0` demorar muito, você pode diminuir o valor de `#define N` em `test.c` para que fique mais rápido (mantendo o N como potência de 2).

Para facilitar a coleta de amostras de tempo de execução, também é fornecido o código para geração de um binário `main` que recebe como entrada os seguintes argumentos:

```
$ ./main --matrix-size <N> --algorithm <ALGO>
```

onde `<N>` é a dimensão das matrizes quadradas (N = número de linhas = número de colunas) e `<ALGO>` é o número da implementação a ser executada (0 = dgemm_0, 1 = dgemm_1). Para compilá-lo, execute no terminal o comando

```
$ make
```

com seu *shell* aberto na pasta contendo o `Makefile`. O tempo de execução em segundos é mostrado na saída (`stdout`). Se desejar, você pode formatar a saída de forma conveniente para a análise dos dados, alterando a linha 118 do arquivo `main.c`. Para remover os arquivos binários gerados na compilação, use o comando

```
$ make clean
```

Observação: o código do mini EP, da maneira como foi fornecido, não é portável para Windows. A princípio, é possível contornar essa limitação com algumas alterações (por exemplo, no `Makefile` e no uso de `<sys/time.h>`), mas a recomendação para este mini EP (e possivelmente para os próximos também) é instalar o WSL (*Windows Subsystem for Linux*) para executar os experimentos.

4. Entrega

Você deverá elaborar um relatório breve, respondendo detalhadamente às seguintes questões:

1. Mostre, com embasamento estatístico, a variação do tempo de execução entre `matrix_dgemm_0` e sua implementação de `matrix_dgemm_1`. Houve melhora no tempo de execução? Explique o porquê.

2. Execute os mesmos experimentos em outra máquina (por exemplo, a de um colega) e verifique se houve mudança no tempo de execução. Eventuais diferenças podem ocorrer devido a diferenças nas especificações das máquinas (por exemplo, tamanho das linhas de *cache*).

Entregue no e-Disciplinas uma pasta compactada com o seu nome e sobrenome no seguinte formato: `miniep5_nome_sobrenome.zip`. Essa pasta deve ser comprimida em formato .zip e deve conter dois itens:

- O arquivo `matrix.c` modificado, contendo a sua implementação da função `matrix_dgemm_1`;

- O relatório em .txt ou .pdf com o seu nome e uma breve explicação sobre a sua solução e desafios encontrados, bem como os tópicos pedidos anteriormente. Imagens também podem ser inseridas no arquivo. Relatórios em .doc, .docx ou .odt não serão aceitos.

Em caso de dúvidas, use o fórum de discussão do e-Disciplinas ou entre em contato diretamente com o monitor (vitortterra@ime.usp.br) ou o professor (gold@ime.usp.br).

5. Agradecimentos

Aos monitores da disciplina em anos anteriores, Giuliano Belinassi e Matheus Tavares, pela elaboração do enunciado do mini EP no qual o presente exercício foi fortemente baseado.