2º ano

Módulo 3

Hierarquia de Memória: Medição do Desempenho

Localidade Espacial

A hierarquia de memória é eficaz na redução dos tempos de acesso à memória quando os padrões de acesso à mesma (dados e instruções) exibem localidade. A localidade espacial caracteriza-se por acessos consecutivos à memória endereçarem células de memória contíguas.

Exercício 1 − Em C as matrizes são armazenadas em memória em *row major order*, isto é elementos consecutivos da mesma linha são armazenados em posições contíguas de memória.

A função de multiplicação de matrizes usada na última sessão, gemm1 (), utiliza o seguinte algoritmo para calcular C = A * B, sendo A, B e C matrizes quadradas com N linhas (e colunas):

Indique quais as matrizes cujos acessos exibem localidade espacial e quais as que não exibem.

Copie o ficheiro /share/acomp/GEMM-P03.zip para a sua directoria, faça "unzip GEMM-P03.zip" e verifique que foi criada a directoria P3. É nesta que deverá trabalhar ao longo deste módulo. Note que o código aqui disponibilizado corresponde à conclusão com sucesso do módulo anterior, pelo que pode usar o seu próprio código se preferir.

Exercício 2 — A ordem de aninhamento dos 3 ciclos for deste código pode ser alterada, mantendo a correcção funcional do programa; no entanto, essa alteração tem um impacto significativo no desempenho. Edite o ficheiro gemm.c e copie a função gemm1 () criando uma nova versão da função gemm2 (). Nesta última altere a ordem dos ciclos:

Indique de novo quais as matrizes cujos acessos exibem localidade espacial e quais as que não exibem. Compare a sua resposta com a do exercício anterior!

E localidade temporal? Consegue identificar uma matriz em que o mesmo elemento seja acedido repetidamente em iterações consecutivas?

Construa o executável (make), verificando na Makefile que estão a ser usadas as opções de optimização: CCFLAGS = -02 -march=ivybridge

2º ano

Exercício 3 – Execute a multiplicação de matrizes para 512 e 1024 linhas, usando as versões 1 e 2. Por exemplo:

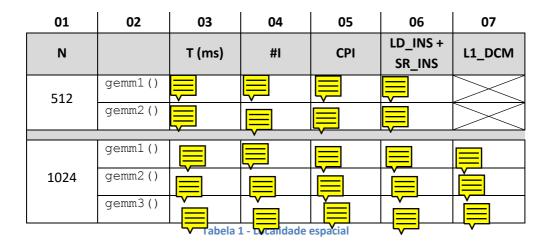
```
> sbatch gemm.sh 1024 1
```

usa gemm1 () para processar uma matriz de 1024x1024 elementos;

```
> sbatch gemm.sh 512 2
```

usa gemm2 () para processar uma matriz de 512x512 elementos;

Preencha as colunas 03 a 06 da Tabela 1 para as linhas correspondentes a gemm1 () e gemm2 () na folha de cálculo GEMM-results.



Exercício 4 — Considerando os valores registados na Tabela 1 é fácil concluir que o melhor desempenho da versão 2 se deve a uma redução abrupta do CPI e apenas marginalmente à redução no número de instruções executadas. No entanto, o número de instruções de acesso à memória não apresenta variações significativas! A que se deverá então a redução do CPI?

Exercício 5 – Altere o ficheiro main.c para que o número de data misses na cache L1 seja contabilizado:

```
#define NUM_EVENTS 5
int Events[NUM_EVENTS] = { PAPI_TOT_CYC, PAPI_TOT_INS, PAPI_LD_INS, PAPI_SR_INS, PAPI_L1_DCM};
```

Preencha a coluna 08 da Tabela 1 para gemm1 () e gemm2 () e N=1024. Observa alguma diferença no número de *misses* na cache de dados? Consegue explicar a que se deve o ganho no CPI?

Localidade Temporal

No exercício 2 foi-lhe pedido que identificasse a matriz cujo padrão de acesso exibe localidade temporal, isto é, o mesmo elemento é acedido repetidamente em iterações consecutivas do ciclo.

Tal acontece com a matriz A e a hierarquia de memória permite aumentar o desempenho explorando esse facto.

O compilador pode ele próprio explorar a localidade temporal, copiando para um registo o elemento a [i] [k] e evitando leituras da memória (o tempo de acesso aos registos é inferior ao tempo de acesso à cache L1). Esta foi aliás a principal optimização estudada em Sistemas de Computação.

No entanto, na função <code>gemm2()</code> o compilador não pode copiar <code>a[i][k]</code> para um registo, devido a um bloqueador de optimização designado por *aliasing*. Na verdade é possível que as matrizes A e C sejam as mesmas, isto é que os apontadores *a e *c apontem para o mesmo espaço de memória ou para espaços que se intersectam. Como o compilador não verifica se tal acontece, então lê <code>a[i][k]</code> de memória sempre que lhe acede.

Exercício 6 – Edite o ficheiro gemm.c e copie a função gemm2 () criando uma nova versão da função gemm3 (). Nesta última use uma variável local:

Execute gemm3 () para N=1024. Que variações detecta em PAPI_LD_INS e PAPI_SR_INS? Justifique estas diferenças. Preencha a linha correspondente a gemm3 () na Tabela 1 e na folha de cálculo GEMM-results.