Arquitetura de Computadores 2017/18

Ficha 10

Tópicos: *Hierarquia de memória: Caches.*

Introdução

A matéria teórica ilustrada nesta aula prática corresponde às aulas teóricas sobre caches. A matéria encontra-se também no capítulo 6 do livro *Computer Systems: A Programmer´s Perspective 2nd Ed*.

Todo o código necessário para este trabalho é fornecido via CLIP. Para compilar tudo, ou de cada vez que alterar um dos ficheiros dos programas, basta dar o comando **`make`**. Este usa as definições em `Mafefile` para compilar os ficheiros necessários, e só esses. Em alternativa pode usar o comando **make *<prog>*** onde *<prog>* é um dos programas (por exemplo, **make cpuid**).

Caches de uma arquitetura real

Usando dois programas em C cujo código fonte está disponível no CLIP pretende-se que obtenha informações sobre a hierarquia de memória do processador do seu computador e das velocidades de acesso à memória que estão associados a cada um dos níveis de cache e memória central.

Informação disponibilizada pelo hardware

Verifique o código em *cpuid.c*. Tenha particular atenção à parte em que no código C aparece o uso de *assembly "inline"*. Esta é uma técnica muito comum para misturar código *assembly* em programas escritos em C, sem recorrer a ficheiros separados para o código *assembly.* Neste caso usa-se a instrução máquina **`cpuid`** para obter informação sobre as características das caches do CPU (assume-se que está a usar um CPU Intel recente ou equivalente). Os parâmetros da instrução **`cpuid`** e a interpretação do conteúdo dos registos eax, ebx, ecx e edx após a execução dessa instrução são descritos em detalhe no manual *Intel® 64 and IA-32 Architectures Software Developer's Manual,* disponível em:

http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html

Compile este programa e execute-o. O programa retorna informação sobre os vários níveis da cache presentes no CPU em que executa. Veja as características do seu CPU. Esta informação será também útil na análise na secção seguinte.

Memory Mountain

Vamos agora usar o programa **`mountain`** discutido no livro da bibliografia (ver sec. 6.6.1). O objetivo é avaliar a taxa de transferência CPU/memória (*memory throughput* ou *bandwidth*) do seu computador e avaliar os vários níveis de cache. Neste programa são medidas as transferências (MB/s) para vários padrões de acesso à memória, simulando diferentes níveis de localidade temporal e espacial. Para tal é medido o tempo que uma função demora a somar, repetidamente, todos os valores de um vetor de inteiros. Dois parâmetros ajustam o perfil dos acessos a memória: o tamanho do vetor e o intervalo entre acessos consecutivos (*stride*). Um vetor menor leva a mais acessos sempre na mesma zona, logo, melhor localidade temporal. Um menor *stride*, corresponde a acessos consecutivos mais próximos, logo, melhor localidade espacial; no limite, com *stride* de 1, temos acessos sequenciais.

Campus de Caparica
2829-516 CAPARICA
Tel: +351 212 948 536
Fax: +351 212 948 541
di.secretariado@fct.unl.pt
www.fct.unl.pt

Compile e execute o programa. Pode guardar os resultados num ficheiro mandando a saída do programa para esse ficheiro. Exemplo: `mountain > resultados.txt`

Este deve permitir obter as várias taxas de transferência para os diferentes parâmetros de tamanho do vetor e *stride*.

Pode, com uma folha de cálculo, ou outro programa, obter um gráfico (3D) que relacione estes valores, apresentando o aspecto semelhante ao de uma montanha (daí o nome do programa). Veja o exemplo na capa do livro da bibliografia ou na secção 6.6.1.

Procure responder às perguntas:

- Quantos níveis de cache tem o seu computador?
- Qual é (aprox.) a dimensão de cada nível?
- Qual é (aprox.) a dimensão das linhas das caches (blocos)?

Otimização de um programa fazer melhor uso das memórias cache do computador

Propõe-se agora que implemente um programa que calcula a matriz transposta de uma matriz quadrada de tamanho *NxN*. Esta matriz é inicializada com valores aleatórios entre 0 e 255.

O programa deverá calcular a matriz transporta através de dois métodos:

1. Versão A: O método "normal", que percorre as linhas da matriz original e as transpõe para as colunas na matriz destino.
2. Versão B: Um método "otimizado" que explora a localidade das memórias cache do computador, através de um processo de divisão da matriz original em blocos mais pequenos de *MxM* e transposição cada um desses blocos individualmente. Esta é a função que deve implementar.

O programa regista (e apresenta) o tempo que as duas transposições demoraram a executar.

No final, o programa verifica se as duas matrizes transpostas são iguais (e, portanto, se a implementação da Versão B está correta).

O programa aceitará um argumento da linha de comando que é a dimensão "*M*" dos blocos menores em que a matriz original deve ser subdividida.

No código fornecido via CLIP poderá encontrar um ficheiro de nome "`transposta.c`" que tem um programa que implementa todos os requisitos anteriores exceto a *Versão B*. Faça *download* do mesmo, compile-o, experimente-o, depois implemente a função "`versaoB`" e teste. A sua implementação desta função deverá ser consideravelmente mais rápida que a "`versaoA`" fornecida.

Experimente também aumentar a dimensão das matrizes (`DIM_MATRIZ`), ou compilar usando a opção de optimização do compilador (`-O2`).

Campus de Caparica
2829-516 CAPARICA
Tel: +351 212 948 536
Fax: +351 212 948 541
di.secretariado@fct.unl.pt
www.fct.unl.pt