Universidade do Minho	**Módulo 3** **Avaliação de desempenho: contadores de hardware**	

Introdução

A complexidade crescente dos sistemas de computação torna o processo de otimização do tempo de execução das aplicações mais difícil. Para facilitar esta tarefa é necessário medir com exactidão vários aspectos da execução do programa. Neste sentido, os fabricantes de processadores foram introduzindo, ao longo dos últimos anos, contadores de eventos internos ao processador que podem ajudar neste processo de otimização. Alguns dos eventos mais frequentes incluem o número de instruções executadas (#I), o número de ciclos máquina (#CC) e o número de acessos à memória, entre outros.
A variedade e quantidade de eventos que podem ser medidos são específicas de cada arquitetura. Por exemplo, nos processadores Core2 existem 5 contadores para medição de eventos, mas 3 destes possuem uma funcionalidade fixa (contabilizam #I e #CC). Ou seja, esta arquitetura apenas apresenta dois contadores genéricos. Assim, quando se pretende medir mais do que 2 eventos é necessário repetir a execução do programa várias vezes.
A biblioteca PAPI (**P**erformance **A**pplication **P**rogramming **I**nterface) apresenta uma abstração sobre estes contadores de eventos, através de uma API que facilita a leitura de um conjunto uniforme de eventos nas diversas arquiteturas.
O comando "papi_avail" permite verificar quais os eventos disponíveis numa dada arquitetura. Por exemplo, o evento PAPI_TOT_INS contabiliza o número total de instruções executadas (#I). Note que o conjunto de eventos disponíveis pode variar com a arquitetura.

Medição de eventos com a biblioteca PAPI

Um programa para a medição de eventos com a biblioteca PAPI deve possuir a estrutura indicada abaixo. Neste exemplo específico são medidos dois eventos (`PAPI_TOT_INS` e `PAPI_TOT_CYC`) [1], recorrendo às funções `PAPI_start_counters()` e `PAPI_stop_counters()`:

```
// exemplo para dois contadores
#define NUMEVENTS 2
long long values[NUMEVENTS];

// exemplo para medir os contadores PAPI_TOT_INS e PAPI_TOT_CYC
int events[NUMEVENTS] = {PAPI_TOT_INS, PAPI_TOT_CYC};
char* events_names[NUMEVENTS] = {"#I", "#CC"}

// iniciar a contagem
if ((PAPI_start_counters(events, NUMEVENTS)) != PAPI_OK)
      PAPI_perror("PAPI_start_counters");

  << segmento de código a medir >>

// ler os valores dos contadores
if ((PAPI_stop_counters(values, NUMEVENTS)) != PAPI_OK)
      PAPI_perror("PAPI_stop_counters");

// mostrar os valores
for (int w=0; w<NUMEVENTS; w++)
     cout << events_names[w] << " " << values[w] << endl;
```

1 A sintaxe apresentada corresponde à versão 5.x do PAPI. No laboratório está instalada a versão 4.1.3, onde a sintaxe da função "PAPI_perror()" é ligeiramente diferente.

A constante `NUM_EVENTS` indica quantos eventos serão medidos, o vector `events` identifica os eventos a medir e o vector `values` guarda os valores lidos. Note que este último tem o tipo de dados `long long` que corresponde a inteiros de 64 bits (complemento para 2). Finalmente o vector `events_names` guarda o nome dos eventos (apenas para ajudar o utilizador do programa).

Exercícios:

1. Execute a comando "papi_avail" para verificar os eventos do PAPI que estão disponíveis nas máquinas do laboratório, quais os eventos suportados nativamente e quais os eventos calculados (i.e., derivados) a partir desses eventos nativos (nota: pode verificar os eventos específicos deste processador com o comando papi_native_avail).

2. O código desenvolvido no módulo 1 da disciplina já inclui o código PAPI necessário para efetuar a medição do tempo de execução (Texec). Altere agora o `main()` para que as duas funções exportadas do módulo `papi_inst.cpp` sejam invocadas (`startPAPI()` e `stopPAPI()`). A primeira função inicia a contagem dos eventos apropriados e a segunda termina esta contagem e apresenta os valores medidos. No módulo `main` pode verificar as chamadas a estas rotinas, antes e depois da chamada à rotina `convolve()`. Altere o módulo `papi_inst.cpp` por forma a que os contadores lidos correspondam ao total de ciclos (`PAPI_TOT_CYC`) e total de instruções executadas (`PAPI_TOT_INS`). Altere também a função `stopPAPI()` para que seja calculado e impresso no êcran o CPI (nota: atenção à conversão dos valores no vector `values` para `double` para que o CPI possa não ser um valor inteiro).
Anote o CPI, o número de instruções e o tempo de execução para as versões compiladas sem e com optimização (`-O0` e `-O3`, respectivamente) para a imagem `AC_images/abe_natsumi256.pgm`. Para a compilação, não esquecer de alterar a `Makefile` com o nível de optimização apropriado e de fazer `make clean` e `make`.

3. Anote a frequência do relógio do seu processador verificando o ficheiro `/proc/cpuinfo` mantido pelo Linux. (Nota; escreva `less /proc/cpuinfo` e procure a linha `cpu MHz`).

4. Calcule o tempo de execução da rotina `convolve`: `Texe = #I * CPI * Tcc`. Compare os valores estimados com os valores efectivamente medidos na alínea 2.

5. Altere a rotina `main()` de forma a calcular o CPE (*cycles per element*), onde cada elemento de dados é um pixel da imagem. Preencha o CPE, #I e Texec na tabela abaixo para as duas versões do código com diferentes ordens de travessia da imagem (*row-wise* e *column-wise*). Use a versão optimizada (`-O3`) e as imagens que vão desde `abe_natsumi32.pgm` a `abe_natsumi1024.pgm` para variar o número de elementos a processar.

6. Numa tentativa de explicar a grande diferença de CPE entre as duas ordens de travessia meça o total de instruções de acesso à memória para cada uma das versões e tamanho de dados. Meça os eventos `PAPI_LD_INS` (total de instruções de leitura de memória) e `PAPI_SR_INS` (total de instruções de leitura de memória). Reporte a soma de ambos (total de instruções de acesso à memória) e preencha a coluna correspondente na tabela abaixo.

7. O CPE para a ordem de travessia *row-wise* aumenta drasticamente quando o tamanho dos dados passa de 256 KByte (imagem 256x256) para 1 MByte (imagem 512x512) e 4 MByte (1024x1024). Vamos medir a *hit rate* da cache L2 para verificar o que se passará. Meça os eventos `PAPI_L2_TCH` (L2 *total cache hits*) e `PAPI_L2_TCA` (L2 *total cache acesses*) e reporte o rácio entre os dois (que corresponderá à *hit rate* na L2). O que acontece quando se passa de 1 MByte para 4 MByte? Sabendo que a cache L2 deste processador (Intel E5300) é de 2 MByte que conclui sobre a localidade de acesso aos dados desta ordem de travessia?