
 Universidade do Minho	<p>Módulo 3 - estendido</p> <p>Avaliação de desempenho: contadores de hardware</p>	
--	--	---

Introdução

A complexidade crescente dos sistemas de computação torna o processo de optimização do tempo de execução das aplicações mais difícil. Com o intuito facilitar essa tarefa, os fabricantes de processadores foram introduzindo, ao longo dos últimos anos, contadores de eventos internos ao processador que podem ajudar neste processo de optimização. Alguns dos eventos mais frequentes são: número de instruções executadas (#I); número de ciclos máquina (#CC); acessos à memória.

A variedade e quantidade de eventos que podem ser medidos em simultâneo é específica 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).

A biblioteca PAPI apresenta uma abstracção sobre estes contadores de eventos, apresentando uma API que permite a utilização 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 alguns desse eventos podem ser derivados, ou seja, calculados a partir de outros eventos.

Medição de eventos com a biblioteca PAPI

Um programa para a medição deve possuir a seguinte estrutura:

```
// 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};

// 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 << values[w] << endl;
```

Pode ser necessário executar o bloco de código múltiplas vezes, sempre que o número de contadores disponíveis for inferior aos eventos a medir. Para isso devem ser efetuadas várias chamadas a `start_counters` e `stop_counters` indicando, em cada chamada, os contadores pretendidos.

Exercícios:

1. O código fornecido no módulo 1 da disciplina já inclui a medição do número de instruções executadas (#I) e do número de ciclos máquina necessários para executar o programa (#CC).

- a) Altere o programa para apresentar o número médio de ciclos necessário para executar cada instrução (i.e., CPI global). Inclua esse código na rotina `stopPAPI()` no módulo `papi_inst.cpp`. Compile o programa (`make`) e execute-o através da seguinte linha de comandos, para obter os valores para a execução da rotina `convolve3x1`:

```
./convolve AC_images/abe_natsumi256.pgm result.ppm 1
```

- b) Compare o número de instruções executadas (i.e., #I) e CPI obtidos com dois níveis de otimização do compilador (O0 - sem otimização e O3). Para tal edite a linha `CCFLAGS` na "Makefile", faça `make clean` e `make` para recompilar todo o código e reexecute o programa para cada uma destas opções. Que conclusão pode retirar?
- c) Calcule o tempo de execução da rotina `convolve3x1` sabendo que a máquina executa a uma frequência de relógio de 2,6 GHz (nota: $\text{Texe} = \#I * \text{CPI} * \text{Tcc}$).
- d) Meça o tempo de execução da rotina com a função `PAPI_get_real_usec()`. Esta função que mede o tempo (em microssegundos) decorridos desde um determinado instante (frequentemente, desde que a máquina foi ligada). Para medir o tempo de execução de um segmento de código com esta função deverá utilizar o esqueleto apresentado de seguida. Neste caso específico, este código deverá ser introduzido na chamada à função `convolve3x1` no módulo `main.cpp`.

```
long long PAPI_start, PAPI_stop;
PAPI_start = PAPI_get_real_usec();
    << segmento de código a medir >>
PAPI_stop = PAPI_get_real_usec();
printf("Time in microseconds: %lld\n", PAPI_stop - PAPI_start);
```

- e) Compare agora os valores de #I, CPI e Texe obtidos com os dois níveis de otimização (O0 e O3). Que conclusão pode retirar?
- f) A imagem processada possui uma dimensão de 256 x 256. Calcule valor aproximado do número de ciclos necessários para processar cada elemento da imagem (i.e., CPE).

2. Recorrendo à geração e visualização do *assembly* da rotina `convolve3x1`, com o comando `g++ -O3 -S convolve3x1.cpp`, indique, assumindo que se processa uma imagem com uma dimensão de 256 x 256:

- a) O número de vezes que cada instrução é executada.
- b) O bloco de instruções que tem maior impacto no desempenho (ou seja o "hot spot"). Calcule a percentagem do tempo total de execução desse bloco de código, assumindo que as instruções têm todas o mesmo CPI (i.é., o CPI global).
- c) O número total de instruções executadas. Note que pode obter uma boa aproximação considerando apenas o "hot spot". Compare esse valor com o medido pelo PAPI.
- d) Classifique cada uma das instruções em: acessos à memória (leitura e escrita); controlo de fluxo; aritméticas/movimento de dados e calcule a percentagem de instruções correspondentes a cada uma das classes. Calcule também a percentagem de instruções que efetuam acessos à memória.

3. Utilize o PAPI para contar o número de instruções executadas das classes: i) a acessos à memória para leitura, ii) acessos à memória para escrita e iii) controlo de fluxo, alterando o módulo `papi_inst.cpp`. Utilize os seguintes eventos (nota: na arquitetura Core2 apenas se pode, simultaneamente, medir dois destes contadores):

PAPI_LD_INS (*LoaDs*)

PAPI_SR_INS (*StoRes*)

PAPI_BR_INS (*BRanches*)

4. Suponha que se pretende desenvolver uma versão otimizada da rotina `convolve3x1`. Observando a implementação da rotina em *assembly* indique:

- a) O número mínimo de instruções de cada classe que pensa ser necessário para implementar a rotina. Considere que a arquitetura possui um número adicional de registos (e.g., r9, r10, r11, etc.). Note que basta concentrar-se no “hot-spot” da rotina.
- b) O valor aproximado do número de ciclos necessários para processar cada elemento da imagem (i.e., CPE), assumindo que as instruções têm todas o mesmo CPI (i.é., o CPI global).