Módulo 3 - estendido

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. 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 otimização. 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 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). Ou seja, esta arquitetura apenas apresenta dois contadores genéricos. Assim, quando se pretende medir mais eventos, é necessário repetir a execução do programa.
A biblioteca PAPI apresenta uma abstração sobre estes contadores de eventos, através de uma API que utiliza 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 dos eventos podem não estar disponíveis numa dada arquitetura, enquanto outros eventos podem ser derivados, ou seja, calculados a partir de outros eventos.

Medição de eventos com a biblioteca PAPI

Um programa para a medição de eventos com a biblioteca PAPI deve possuir a seguinte estrutura, neste caso específico são medidos dois eventos (PAPI_TOT_INS e PAPI_TOT_CYC) [1]:

```
// 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;
```

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_perro" é ligeiramente diferente.

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 fornecido no módulo 1 da disciplina já inclui o código PAPI necessário para efetuar a medição do número de instruções executadas (#I) e do número de ciclos máquina necessários para executar o programa (#CC). Esse código encontra-se no módulo `papi_inst.cpp`. Este módulo exporta duas funções: `startPAPI()` e `stopPAPI()`. A primeira função inicia a contagem e a segunda termina a contagem e apresenta os valores medidos. No módulo `main` pode verificar as chamadas a estas rotinas, antes e depois da chamada à rotina `convolve3x1`.

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()` do módulo `papi_inst.cpp`. Note que os valores medidos estão armazenados no vetor `values` e que deve converter esses valores para o tipo `double`, de forma a que o valor do CPI global não seja convertido num valor inteiro. Compile o programa (`make`) e execute-o através da seguinte linha de comandos:

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

b) Compare o número de instruções executadas (i.e., #I) com dois níveis de optimização do compilador (O0 - sem optimização 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: `Texe = #I * CPI * Tcc`).

d) Meça o tempo de execução da rotina `convolve3x1` agora com a função `PAPI_get_real_usec()`. Esta função 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 rotina `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 optimização (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). Neste caso, pode obter uma aproximação razoável do CPE simplesmente dividindo o número total de ciclos necessários para executar a rotina `convolve3x1` pelo número de elementos processados.

3. Recorrendo à geração e visualização do *assembly* da rotina `convolve3x1` (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. Para tal deverá identificar no código *assembly* os blocos de instruções correspondentes a cada um dos dois ciclos da rotina `convolve3x1`. Cada bloco de instruções será executado um número de vezes igual ao do ciclo correspondente.

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 do "hot spot" 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.

4. 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*)

5. 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).