Módulo nº 7

Medição do desempenho: IA32

1. Introdução

Pretende-se com esta aula prática que os alunos se familiarizem com um mecanismo de medição do CPI na arquitectura IA32, e que avaliem o impacto das optimizações disponibilizadas pelos compiladores (gcc) no desempenho de um programa.

2. Medição do Desempenho

Medição do número de ciclos

A arquitectura IA32, dispõe, a partir do lançamento do Pentium, de um contador de 64 bits, que é incrementado em cada ciclo do relógio. O contador é iniciado a 0 quando é feito o *reset* do processador, sendo depois consecutivamente incrementado. Quando o maior valor possível de representar é atingido (64 *bits* a 1) o contador volta a 0. Num contador de 64 bits e com um relógio de 1 GHz, isto acontecerá de 570 em 570 anos.

Este contador pode ser lido pela instrução "*RDTSC – Read Time Stamp Counter*", que coloca os seus 32 bits mais significativos no registo EDX e os 32 bits menos significativos no registo EAX. Se forem ignorados os 32 bits mais significativos, é possível trabalhar com um contador de 32 bits apenas, tendo no entanto em consideração que o contador dará a volta (*wrap around*), isto é, volta a 0, de 4,3 em 4,3 segundos, numa máquina com um relógio de 1 GHz.

O número de ciclos decorridos durante a execução de um programa pode ser calculado lendo o contador no início (T1) e no fim (T2) da execução do mesmo. A diferença T2-T1 é igual ao número de ciclos, devendo-se verificar se T2>T1 e que o programa não demorou mais do que o tempo de *wrap around*.

O número de ciclos medido será sempre uma aproximação, pois o programa pode ser interrompido e podem ocorrer mudanças de contexto. Para que a leitura seja o mais precisa possível, mediremos apenas programas cujo tempo de execução seja muito inferior ao tempo de *wrap around* do contador de 32 *bits* e inferior à *time slice* dos processos no LINUX. Tentaremos portanto não ultrapassar os 5 milisegundos, minimizando também a influência das rotinas de atendimento de interrupção. Para mais detalhes sobre temporização de processos e utilização dos 64 bits do contador ver secções 9.3 e 9.4 do livro "Computer Systems: A Programmer's Perspective"; Bryant and Hallaron.

Variações nas medições

As medições obtidas para diferentes execuções do mesmo programa podem variar bastante, devido à influência da hierarquia de memória, interrupções e outros componentes da máquina. O valor nunca será, no entanto, menor do que o número de ciclos mínimo necessário para executar o programa em circunstâncias ideias: número mínimo de *cache misses*, não ocorrência de interrupções, etc.

Sugere-se que, para cada versão do programa, este seja executado várias vezes e o valor mínimo reportado seja o considerado. Para mais detalhes e métodos de temporização mais sofisticados, ver a bibliografia atrás referida.

Cálculo do CPI

Para calcular o **CPI** (ciclos por instrução) é necessário saber o número de instruções executadas (#I) além do número de ciclos decorridos. Este número (#I) pode ser estimado analisando o código *assembly* gerado pelo compilador. Note que esta estimativa não necessita de ser absolutamente exacta, dado o elevado número de instruções que são medidas.

3. Programa não optimizado

Considere o seguinte programa, escrito em C:

prog.c

```
#include <stdio.h>

#define ARR_SIZE 16384
int arr[ARR_SIZE];

int main ()
{
  int i, total, cycles;

  for (i=0; i<ARR_SIZE ; i++)
    arr[i] = i;
  total = 0;
  for (i=0; i<ARR_SIZE ; i++)
    total += arr[i];
  printf ("total=%d cycles=%u\n",total, cycles);
}
```

Compile-o usando o comando

gcc –S prog.c

A variável `cycles`, não utilizada pelo código em C, será a utilizada para contar o número de ciclos decorridos durante a execução do programa. Note que se trata de uma variável de 32 bits, pelo que usaremos apenas os 32 bits menos significativos, devolvidos no registo %EAX pela instrução RDTSC.

Questão 1 – Leia o contador de ciclos logo após a instrução de actualização do %EBP.

Questão 2 – Logo que possível, guarde o valor de %EAX, devolvido por RDTSC, no espaço de armazenamento reservado para `cycles`. Chamaremos a este valor T1.

Questão 3 – Pretende-se ler de novo o contador após a execução do último ciclo, antes de passar os parâmetros para o `printf()`. Chamaremos a este valor T2. Lembre-se que a leitura do contador altera os registos %EDX:%EAX, logo se estes contiverem valores necessários para a execução do resto do programa é necessário guardar esses valores em lugares apropriados. Vamos também verificar que o relógio não efectuou um *wrap around*, garantindo que T2-T1>0. Se tal não acontecer o programa deverá colocar `cycles` a 0. Sugere-se o seguinte código *assembly*, tomando em atenção que o parâmetro a passar para o `printf()`, deve ser o resultado que fica em %EAX e que onde aparece escrito `cycles`, se pretende referir o espaço de armazenamento reservado pelo compilador para esta variável.

```
        rdtsc
        subl cycles, %eax
        jg TSOK
        xorl %eax, %eax
TSOK:
```

Questão 4 – Calcule aproximadamente quantas instruções são executadas e medidas neste programa.

Questão 5 – Gere o executável usando o comando

`gcc –o progO0 prog.s`

Recorrendo à metodologia descrita anteriormente calcule o CPI para este programa. Se conseguir saber a frequência do relógio do seu computador calcule o tempo de execução, caso contrário use uma frequência de 1 GHz.

4. Compilação optimizada

Questão 6 – Repita todos os passos da Questão 1 à 5, mas agora usando o nível de optimização O3. Preste especial atenção ao local onde o valor de cycles é guardado e à possível escrita destrutiva de valores importantes em %EDX:%EAX na segunda leitura do contador. Use os seguintes comandos:

`gcc -S –O3 prog.c`
`gcc –O3 –o prog.O3 prog.s`

Questão 7 – Compare justificando o CPI obtido para cada uma das versões.

Questão 8 – Compare justificando o tempo de execução obtido para cada uma das versões.