Arquitectura de Computadores
Licenciatura em Engenharia Informática
Luís Paulo Peixoto dos Santos
Dep.de Informática - Universidade do Minho

Exercícios Propostos (2º teste)
Dezembro, 2024

4 – SuperEscalaridade

1. Considere o programa abaixo:

```
I1: pop  %eax
I2: add  $20, %eax
I3: push %eax
I4: pop  %ecx
I5: add  %esi, %eax
I6: mul  %ebx, %ecx
I7: mov  %edx, 20(%ebx)
I8: push %ecx
```

a) Considere um processador com uma única unidade funcional. As instruções de acesso à memória têm CPI=2, as restantes CPI =1. Qual o CPI para a execução deste programa?

b) Preencha a tabela abaixo considerando um processador com 2 unidades funcionais de acesso à memória (UF1 e UF2) e 2 unidades funcionais de cálculo de inteiros (UF3 e UF4). UF1 e UF2 têm CPI=2, UF3 e UF4 têm CPI=1. Considere escalonamento *in-order*. Qual o CPI?

cycle	MEM		INT	
	UF1	UF2	UF3	UF4
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				

c) Preencha a tabela abaixo considerando que o processador da alínea anterior foi melhorado para fazer escalonamento *out-of-order.* Qual o CPI?

cycle	MEM		INT	
	UF1	UF2	UF3	UF4
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				

2. Considere o programa abaixo:

```
I1: mov  (%ebx, %ecx), %esi
I2: add  $20, %esi
I3: mov  %esi, (%ebx, %ecx)
I4: dec  %ecx
I5: jnz  I1:
```

Considere um processador superescalar *out-of-order* com 2 unidades funcionais de acesso à memória (UF1 e UF2) e 2 unidades funcionais de operações sobre inteiros e saltos (UF3 e UF4). Estas unidades funcionais têm CPI=1. Os saltos condicionais são sempre previstos como tomados.

a) Preencha a tabela abaixo, apresentando o escalonamento das primeiras duas iterações deste programa. Etiquete a execução da cada instrução com um sufixo com o número da iteração. Por exemplo: `I1` executada na 1ª iteração escreve-se `I1-1` e na 2ª iteração escreve-se `I1-2`. Assuma que o valor de `%ecx` não atinge 0 nestas duas iterações. Qual o CPI?

cycle	MEM		INT	
	UF1	UF2	UF3	UF4
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				

b) Reescreva o código acima aplicando *loop unrolling* (2 elementos do vector por iteração).

c) Preencha a tabela abaixo, apresentando o escalonamento da primeira iteração do código *unrolled.* Qual o CPI?

cycle	MEM		INT	
	UF1	UF2	UF3	UF4
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				

5 – Vectorização

1. Para cada um dos excertos de código baixo indique, justificando, se vectoriza:

```
float a[SIZE], b[SIZE];

j = 0;
for (i=2 ; i < SIZE-2 ; i++) {
  a[i] = a[j] / b[i+2];
  j++;
}
```

```
float a[SIZE], b[SIZE];

for (i=1 ; i <SIZE-1 ; i++)
  a[i] = a[i+1] * b[i-1];
```

```
float a[SIZE], b[SIZE];

for (i= SIZE-2 ; i >= 0 ; i--)
  a[i] = a[i+1] * b[i-1];
```

2. O excerto de código abaixo vectoriza? Pode escrever uma nova versão que vectorize parcialmente?

```
float a[SIZE];

for (i=1 ; i < SIZE-2 ; i++) {
  a[i] = a[i-1]/ (a[i]*a[i+2]) + 10. / (a[i]*a[i]*a[i+1]);
}
```

3. Considere o excerto de um programa em C apresentado abaixo, bem como o código *assembly* do corpo do ciclo gerado por um compilador para as versões escalares e vectoriais. Note que:

i) as instruções `vmovuss, vaddss, vmulss` (isto é, com sufixo `ss`) são instruções escalares;

ii) os registos `%xmm1` e `%ymm1` estão inicializados com o valor 20.

```
#define SIZE 40000
float a[SIZE];
...
for (i=0 ; i <SIZE ; i++)
  a[i] = a[i]*20. + 20./a[i];
```

```
ciclo:
  vmovuss %ecx(%ebx), %xmm0
  vmulss  %xmm2, %xmm0, %xmm1
  vdivss  %xmm3, %xmm1, %xmm0
  vaddss %xmm0, %xmm2, %xmm3
  vmovuss %xmm0, %ecx(%ebx)
  incl %ecx
  cmpl SIZE, %ecx
  jl ciclo
```

```
ciclo:
  vmovaps %ecx(%ebx), %ymm0
  vmulps %ymm2, %ymm0, %ymm2
  vdivps %ymm3, %ymm1, %ymm0
  vaddps %ymm0, %ymm2, %ymm3
  vmovaps %ymm0, %ecx(%ebx)
  addl $8, %ecx
  cmpl SIZE, %ecx
  jl ciclo
```

Dados os valores médios de CPI para cada tipo de instrução apresentados abaixo, e com a frequência do relógio igual a 2 GHz, qual o tempo de execução para cada uma das versões?

Instruções Escalares		**Instruções Vectoriais**	
Tipo	**CPI**	**Tipo**	**CPI**
Acesso à memória	2	Acesso à memória	3
Operações lógicas e aritméticas	1	Operações lógicas e aritméticas	1.5
Saltos	1	---	---

6 – Paralelismo *multi-core*

1. Para o código abaixo proponha uma implementação que explore *Thread Level Parallelism* usando o OpenMP. Justifique as suas opções; em particular justifique cada uma das directivas e/ou cláusulas OpenMP utilizadas.

```
#define S 1000000
float a[S];
int i ,j;
for (i=0 ; i < S ; i++) {
  a[i] = 0.;
  for (j=0 ; j < i ; j++) {
    a[i] += (float)j;
} }
```

2. O código apresentado abaixo pretende calcular o valor mínimo de uma matriz. Este código tem 2 erros semânticos associados ao OpenMP, que fazem com que o respectivo resultado num ambiente paralelo seja indeterminado. Identifique esses erros (**justificando**) e apresente uma versão correcta do código.

```
#define W 40000
int a[W][W];
int min, i, j;
...
#pragma omp parallel
{
  min = a[0][0];
  #pragma omp for
  for (i=0 ; i < W ; i++) {
    for (j=0 ; j < W ; j++) {
      if (a[i][j] < min) min = a[i][j];
}}}
```

3. Considere o código C + OpenMP apresentado abaixo:

```
int x=10;
#pragma omp parallel num_threads(3) {
  #pragma omp master
       printf ("Aqui vamos nós…\n");
  #pragma omp atomic
         x += 2;
  printf ("T%d: x+1 = %d\n", omp_get_thread_num(), x);
}
printf ("That's all, folks!\n");
```

Indique qual dos seguintes *outputs* é possível.

	T1: x+1 = 12
	T0: x+1 = 14
☐	Aqui vamos nós…
	T2: x+1 = 16
	That's all, folks!

	Aqui vamos nós…
	T1: x+1 = 16
☐	T2: x+1 = 16
	T0: x+1 = 16
	That's all, folks!

	T1: x+1 = 16
	Aqui vamos nós…
☐	T2: x+1 = 16
	T0: x+1 = 16
	That's all, folks!

	T1: x+1 = 10
	Aqui vamos nós…
☐	T0: x+1 = 12
	T2: x+1 = 14
	That's all, folks!