Módulo 3 - Estendido

Avaliação de desempenho: contadores de hardware

1.a) e 1.b)

	-O0	-O3
#I	$3.1x10^6$	$1.0x10^6$
#CC	$2.0x10^6$	$1.4x10^6$
#CPI	**0.6**	**1.3**

Nota: deve converter-se os valores dos contadores para "double" para imprimir o CPI no ecrã.

1.c) e 1d)

	-O0	-O3
Texe (ms)	$2.0x10^6$ / $2.6x10^9$ = **0.769**	$1.4x10^6$/ $2.6x10^9$ = **0.538**
PAPI_usec	**0.930**	**0.694**

Nota: a diferença entre o estimado e o real é um pouco superior ao que seria de esperar

1.e) #I diminui em 2.0/1.4 = 3.1 vezes; CPI aumenta =1.3/0.6 = 2.2 vezes
Texe diminui com O3 porque #I * CPI diminui

1.f)

	-O0	-O3
CPE	$2,0x10^6$ / 256x256 = **30.5**	$1,4x10^6$/ 256x256 = **21.4**

2.a)

	TIPO	NºExecuções
pushl %ebp	SR	1
movl %esp, %ebp	outro	1
pushl %edi	SR	1
pushl %esi	SR	1
pushl %ebx	SR	1
subl $16, %esp	outro	1
cmpl $2, 16(%ebp)	LD	1
jle .L6	BR	1
movl 8(%ebp), %eax	LD	1
movl $1, %ebx	outro	1
movl $2, -24(%ebp)	SR	1
addl $4, %eax	outro	1
movl %eax, -28(%ebp)	SR	1
movl 16(%ebp), %eax	LD	1
sall $2, %eax	outro	1
movl %eax, -20(%ebp)	SR	1
.L3: # entre L3 e L6 executa W-2 = 254 vezes		
movl 20(%ebp), %eax	LD	254
testl %eax, %eax	outro	254
jle .L5	BR	254
movl -28(%ebp), %esi	LD	254
xorl %edi, %edi	outro	254
movl $0, -16(%ebp)	SR	254
.L4:		
movl 12(%ebp), %eax	LD	256*254
addl $1, -16(%ebp)	**LD+SR**	256*254
addl %edi, %eax	outro	256*254
movl (%eax,%edx,4), %ecx	LD	256*254
addl -4(%eax,%ebx,4), %ecx	LD	256*254
addl 4(%eax,%edx,4), %ecx	LD	256*254
movl $1431655766, %eax	outro	256*254
addl -20(%ebp), %edi	LD	256*254
imull %ecx	outro	256*254
sarl $31, %ecx	outro	256*254
subl %ecx, %edx	outro	256*254
movl %edx, (%esi)	SR	256*254
movl 20(%ebp), %eax	LD	256*254
addl -20(%ebp), %esi	LD	256*254
cmpl %eax, -16(%ebp)	LD	256*254
jne .L4	BR	256*254
.L5:		
addl $1, -24(%ebp)	LD+SR	254
addl $1, %ebx	outro	254
movl 16(%ebp), %eax	LD	254
addl $4, -28(%ebp)	LD+SR	254
cmpl %eax, -24(%ebp)	LD	254
jne .L3	BR	254
.L6:		
addl $16, %esp	outro	1
popl %ebx	LD	1
popl %esi	LD	1
popl %edi	LD	1
popl %ebp	LD	1
ret	LD	1

2.b) O bloco *hot spot* é aquele que começa em L4 e vai até ao início de L5.

Aproximação simples:

256/(256+1) = 99.6% (cada ciclo L4 executa 256x para cada L3 e L5)

A aproximação mais correta considera o #I de cada bloco:

16 x 254 x 256 / (**16** x 254 x 256 + **12** x 254 + **22**) = 99.7 %

2.c) #I = ((16*256)+12)*(256-2) +22 = **1043418**
#I_PAPI = **1046647**
} valores muito próximos

2.d) Nota: só usamos o ciclo mais interior (*hot spot*) porque os outros têm pouco peso.

%LD = 9/16 = 56% ← O PAPI conta uma das instruções como LD e SR
%SR = 2/16 = 13% ←
%BR = 1/16 = 6%
%Outras = 5/16 = 31%

%Instruções que acedem à memória = 10/16 = 62.5% → é diferente de %LD+%SR porque uma instrução é LD e SR

3.

	LD	SR	BR
Calculado (ver 2.d)	9*256*254 = $585x10^3$	2*256*254 = $130x10^3$	1*256*254 = $65x10^3$
Medido com PAPI	$587x10^3$	$131x10^3$	$66x10^3$

4.

Nota: ao compilar com opção **–O3** a rotina *kernel* fica embutida em *convolve3x1*

Activation record de *convolve3x1*:

20(%ebp) = H
16(%ebp) = w
12(%ebp) = inp = I
8(%ebp) = res = h

-16(%ebp) = y

Para otimizar o *hot spot* da rotina *convolve3x1*, será preciso introduzir as seguintes instruções antes de **L4**:

- `movl 12(%ebp), %r9` (r9=inp=I)
- `movl -16%ebp), %r10` (r10=y)
- `movl -20(%ebp), %r11`
- `movl 20(%ebp), %r12` (r12=H)

Deste modo, estes acessos à memória são executados apenas 1/256, em vez de 256/256. Este resultado poderia ainda ser melhorado se analisássemos o código anterior a L4.

Código original	Comentário	Novo código
.L4:		
movl 12(%ebp), %eax	EAX = I = inp	**movl %r9, %eax**
addl $1, -16(%ebp)	++y	**addl $1, %r10**
addl %edi, %eax	EAX = inp+y*W*4	igual
movl (%eax,%ebx,4), %ecx	ECX = I[y*W+x] <=> inp[ndx]	igual pq EAX varia (LD)
addl -4(%eax,%ebx,4), %ecx	ECX += inp[ndx]+inp[ndx-1]	igual pq EAX varia (LD)
addl 4(%eax,%ebx,4), %ecx	Inp[ndx+1]	igual pq EAX varia (LD)
movl $1431655766, %eax	ajuda a fazer DIV de ECX por 3	igual
addl -20(%ebp), %edi	EDI = (y+1)*W*4	**addl %r11, %edi**
imull %ecx	ajuda a fazer DIV por 3	igual
sarl $31, %ecx	ajuda a fazer DIV por 3	igual
subl %ecx, %edx	ajuda a fazer DIV por 3	igual
movl %edx, (%esi)	res[ndx]=(inp[ndx]+inp[ndx-1]+inp[ndx+1])/3	igual pq ESI varia (SR)
movl 20(%ebp), %eax	EAX = H	instrução desnecessária
addl -20(%ebp), %esi	ESI = ++&h[x+y*W])	**addl %r11, %esi**
cmpl %eax, -16(%ebp)	compara y com H (y-H)	**cmpl %r12, %r10**
jne .L4	salta para L4 se y != H	igual (BR)

4.a) #I = 15 ➔ **3** LD, **1** SR, **1** BR ➔ código do *hot spot* exige pelo menos 3 LD e 1 SR

4.b) considerando CPI_{global} = 1.3 (medido em 1.a) vem:
CPE = #I (para cada elemento) x CPI = 15x1.3 = 19.5

Nota: Na prática o CPI_{global} deve ser mais baixo pois temos menos acessos à memória do que quando medimos o CPI em 1.a).

```
# assembly completo da rotina convolve3x1 obtido com otimização –O3
pushl %ebp
movl  %esp, %ebp
pushl %edi
pushl %esi
pushl %ebx
subl  $16, %esp
cmpl  $2, 16(%ebp)            # compara W=256 com 2
jle   .L6                     # salta para L6 se W <=2 => não salta
movl  8(%ebp), %eax           # EAX=h
movl  $1, %ebx                # EBX = x = 1
movl  $2, -24(%ebp)           # M[EBP-24]=2
addl  $4, %eax                # EAX=h+4
movl  %eax, -28(%ebp)         # M[EBP-28]=h+4
movl  16(%ebp), %eax          # EAX=W
sall  $2, %eax                # EAX=W*4
movl  %eax, -20(%ebp)         # M[EBP-20]=W*4

.L3:
movl  20(%ebp), %eax          # EAX = H
testl %eax, %eax              # faz H&H=H ==> afeta as flags
jle   .L5                     # salta pata L5 se H<=0
movl  -28(%ebp), %esi         # ESI = h+4; h+2*4; ... h+x*4
xorl  %edi, %edi              # EDI = 0
movl  $0, -16(%ebp)           # M[EBP-16] = y = 0

.L4:
movl  12(%ebp), %eax          # EAX = I = inp
addl  $1, -16(%ebp)           # M[EBP-16] += 1 <=> ++y
addl  %edi, %eax              # EAX = inp+0; inp+4*W; ... <=> inp+y*W*4
movl  (%eax,%ebx,4), %ecx     # ECX=*(I+y*W*4+x*4) <=> I[y*W+x] <=> inp[ndx]
addl  -4(%eax,%ebx,4), %ecx   # ECX += *(I+y*W*4+x*4-4)= inp[ndx]+inp[ndx-1]
addl  4(%eax,%ebx,4), %ecx    # ECX += *(I+y*W*4+x*4+4)= inp[ndx]+inp[ndx-1]+inp[ndx+1]
movl  $1431655766, %eax       # ajuda a fazer a divisao por 3
addl  -20(%ebp), %edi         # EDI = 0+W*4; W*4+W*4; ... (y+1)*W*4
imull %ecx              # ajuda a fazer (inp[ndx]+inp[ndx-1]+inp[ndx+1])/3
sarl  $31, %ecx         # ajuda a fazer (inp[ndx]+inp[ndx-1]+inp[ndx+1])/3
subl  %ecx, %edx        # ajuda a fazer (inp[ndx]+inp[ndx-1]+inp[ndx+1])/3
movl  %edx, (%esi)      # *(h+x*4+y*W*4)=h[x+y*W] = res[ndx] =
                        #    = (inp[ndx]+inp[ndx-1]+inp[ndx+1])/3
movl  20(%ebp), %eax    # EAX = H
addl  -20(%ebp), %esi   # ESI=h+1*4+4*W; h+1*4+W*4+W*4;...h+x*4+y*W*4 <=> ++&h[x+y*W])
cmpl  %eax, -16(%ebp)   # compara y com H (y-H)
jne   .L4               # salta para L4 se y != H
.L5:
addl  $1, -24(%ebp)           # M[EBP-24]=2+1; 2+1+1; ... <=> 2+x
addl  $1, %ebx                # ++x
movl  16(%ebp), %eax          # EAX = W
addl  $4, -28(%ebp)           # M[EBP-28]=h+4+4; ...
cmpl  %eax, -24(%ebp)         # compara (x+2) com W  (x+2-W)
jne   .L3                     # salta para L3 enquanto x<=(W-2) <=> x<(W-1)
.L6:
addl  $16, %esp
popl  %ebx
popl  %esi
popl  %edi
popl  %ebp
ret
```