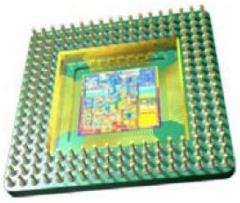
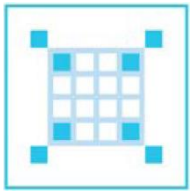


Arquitectura de Computadores

LICENCIATURA EM ENGª INFORMÁTICA
FACULDADE DE CIÊNCIA E TECNOLOGIA
UNIVERSIDADE DE COIMBRA



Lab 11 - Optimização de Código

Neste trabalho laboratorial pretende-se demonstrar como uma programação cuidada pode melhorar dramaticamente a eficiência computacional dos nossos programas. Pretende-se também verificar como a programação em *assembly* pode ajudar a melhorar os tempos de processamento de funções críticas que são executadas múltiplas vezes.

1. Introdução

O objetivo deste trabalho é perceber como podemos optimizar o código de funções complexas e que são executadas múltiplas vezes. Para isso, vamos recorrer à implementação de uma função de binarização de uma imagem, tal como foi feito no exercício 3 do trabalho anterior.

Uma imagem ***w x h*** pode ser guardada numa tabela unidimensional do tipo *'unsigned char*'. Se ***img*** é a referida tabela, então o byte ***img[i*w+j]***, em que ***0 <= i < h*** e ***0 <= j < w***, guarda o nível de cinzento do pixel situado na linha ***i*** e coluna ***j***. Pretende-se que desenvolva uma função, inicialmente escrita em C, que faça a binarização da imagem. O código deverá percorrer todos os píxeis da imagem, colocando a zero (preto) aqueles que estão abaixo de um determinado limiar predefinido, e escrevendo 255 nos restantes (branco). **Este limiar deverá ser calculado também na função como o valor médio dos pixéis contidos na imagem**.

Na função **`main`** (`main.c`) fornecida com este enunciado, pode verificar (a partir da linha 172) que a função de binarização vai ser chamada 500 vezes, para simular o que eventualmente poderia acontecer caso estivéssemos a processar uma sequência de vídeo em vez de uma única imagem. As múltiplas chamadas a esta função de binarização vão ser cronometradas e o valor gasto com a função binariza vai ser impresso no ecrã.

O resultado da função de binarização deverá ser semelhante ao apresentado na figura a seguir:

Fig. 1 – À esquerda a imagem original e à direita a imagem binarizada correspondente.

O código da função ***main*** lê uma imagem ***einstein.pgm***, chama a função de binarização, e devolve a imagem binarizada em ***output.pgm***. Se o executável final for **binariza**, então a chamada da linha de comandos deverá ser algo do género:

```
./binariza einstein.pgm output.pgm
```

2. Optimização de Código

a) Numa primeira abordagem implemente a função binariza em C (`bin_img_c`). Corra o programa utilizando esta função. Para isso, descomente a linha correspondente à chamada desta função no ficheiro `main.c`. Execute o programa e aponte o tempo de execução que obteve com a utilização desta função.

b) De seguida implemente a função binariza agora escrita directamente em *assembly* do MIPS (`bin_img.s`). Tal como no ponto anterior, corra agora o programa recorrendo a esta função no ficheiro `main.c`. Anote o tempo de execução obtido e compare com a versão anterior. Qual das versões teve um tempo de execução mais baixo?

c) Os compiladores modernos permitem a activação de modos de optimização de código que permitem gerar programas executáveis mais eficientes. No gcc a flag utilizada para este efeito é a flag -O que pode ainda conter um sufixo numérico que indica qual o modo de optimização a utilizar. Veja a seguinte página para obter uma explicação dos vários modos de optimização: http://www.rapidtables.com/code/linux/gcc/gcc-o.htm
Recorra novamente à versão em C que implementou no ponto a), mas desta feita utilize a flag de optimização na compilação. Execute o programa e verifique os tempos de execução obtidos com cada um dos modos de optimização. Compare estes tempos com o obtido pela sua própria implementação em *assembly*. Caso tenha curiosidade em perceber quais os tipos de optimizações realizadas, compile a função com a opção -S para gerar o código *assembly* correspondente.

d) Procure optimizar o seu código em assembly desenvolvido no ponto b) para ver até que ponto pode diminuir o tempo de execução da sua função. Bata o recorde da sua turma prática, ou pelo menos tente bater os tempos de execução da versão em C com as *flags* de optimização.

NOTAS:

1. *Para testar o programa no servidor MIPS, vai ter de transferir para a sua conta neste servidor os ficheiros de imagem de teste. Para visualizar as imagens com o resultado da execução do programa, terá de as transferir primeiro para o seu computador local e visualizá-las utilizando a ferramenta disponibilizada na pasta **Tools** (**OpenSeeIt.exe**) se estiver a utilizar o Windows. Para outros sistemas operativos poderá utilizar um visualizador de imagens compatível ou converte-las para um dos formatos de imagem mais usuais (.bmp, .jpg, etc...). Para isso poderá usar o conversor online disponível no endereço https://convertio.co/pt/.*

2. *Para poder controlar manualmente a utilização dos delayed slots terá que utilizar a seguinte directiva de compilação (o assembler deixa de fazer a reordenação do código e a ocupação automática dos delayed slots):*

```
.set   noreorder
```

*Note que ao fazer isto terá que preencher manualmente o delayed slot após cada instrução de salto (branches e jumps). Se não conseguir utilizar esse slot (por não ser capaz de reordenar o código) deverá utilizar a instrução **NOP**.*

A qualquer momento poderá instruir o assembler para voltar a controlar a reordenação do código fazendo:

```
.set   reorder
```