11º Exercício Prático
Desenvolvido no Laboratório

Objetivo

Comparar instruções semelhantes em linguagem de montagem e verificar sua influência no tempo de execução.

Materiais

1. Compilador GCC
2. Arquivo FiltroMediana.c
3. Imagem LapisRuido.ppm

Introdução

As instruções em linguagem de montagem de um processador podem ter os tempos de execução influenciados pela ordem que são executadas e se elas aproveitam bem o pipeline do processador.

O exercício de hoje consiste em aplicar o filtro de mediana em uma imagem que apresenta ruídos.

Um filtro de mediana [1] é um exemplo de filtro não linear para remover um tipo específico de ruido e, se projetado adequadamente, é muito bom para preservar os detalhes da imagem. Para executar um filtro de mediana:

1. Considere cada pixel na imagem
2. Classifique os pixels vizinhos em ordem com base em suas intensidades
3. Substitua o valor original do pixel pelo valor da mediana dos vizinhos.

[1]Jain Anil K. Fundamentals of digital image processing(Prentice Hall, Pearson Education, 1989).

Desenvolvimento

O programa a ser otimizado gera uma imagem filtrada utilizado o filtro da mediana. Cada pixel para ser filtrado utiliza os valores de cada canal dele e dos seus vizinhos, calcula a mediana desses valores e usa a mediana de cada canal na imagem de saída.

Executando o programa sem otimização

1. Compile o programa FiltroMediana.c
   - gcc -masm=intel -g -O2 FiltroMediana.c -o FiltroMediana
2. Execute o programa.
   - ./FiltroMediana

Qual foi o tempo observado na execução?

Tempo 1: _____ s

Otimização usando linguagem de montagem, tentativa 1

Nesta tentativa de otimização, a rotina mediana original foi substituída por uma rotina escrita em linguagem de montagem.

1. Abra o arquivo FiltroMediana.c em um editor de texto
2. Altere a função mediana() como código abaixo

```
int mediana(int v1, int v2, int v3, int v4, int v5) {
  asm("   cmp  %[v1],%[v2]     \n"
      "   jle  salta_12%=      \n"
      "   mov  r8d,%[v1]       \n"
      "   mov  %[v1],%[v2]     \n"
      "   mov  %[v2],r8d       \n"
      "salta_12%=:             \n"

      "   cmp  %[v1],%[v3]     \n"
      "   jle  salta_13%=      \n"
      "   mov  r8d,%[v1]       \n"
      "   mov  %[v1],%[v3]     \n"
      "   mov  %[v3], r8d      \n"
      "salta_13%=:             \n"

      "   cmp  %[v1],%[v4]     \n"
      "   jle  salta_14%=      \n"
      "   mov  r8d,%[v1]       \n"
      "   mov  %[v1],%[v4]     \n"
      "   mov  %[v4], r8d      \n"
      "salta_14%=:             \n"

      "   cmp  %[v1],%[v5]     \n"
      "   jle  salta_15%=      \n"
      "   mov  r8d,%[v1]       \n"
      "   mov  %[v1],%[v5]     \n"
      "   mov  %[v5], r8d      \n"
      "salta_15%=:             \n"
      //-----------------------------
      "   cmp  %[v2],%[v3]     \n"
      "   jle  salta_23%=      \n"
      "   mov  r8d,%[v2]       \n"
      "   mov  %[v2],%[v3]     \n"
      "   mov  %[v3], r8d      \n"
      "salta_23%=:             \n"

      "   cmp  %[v2],%[v4]     \n"
      "   jle  salta_24%=      \n"
      "   mov  r8d,%[v2]       \n"
      "   mov  %[v2],%[v4]     \n"
      "   mov  %[v4], r8d      \n"
      "salta_24%=:             \n"

      "   cmp  %[v2],%[v5]     \n"
      "   jle  salta_25%=      \n"
      "   mov  r8d,%[v2]       \n"
```

```
        "   mov  %[v2],%[v5]     \n"
        "   mov  %[v5], r8d      \n"
        "salta_25%=:             \n"
        //-----------------------------
        "   cmp  %[v3],%[v4]     \n"
        "   jle  salta1%=        \n"
        "   mov  %[v3],%[v4]     \n"
        "salta1%=:               \n"

        "   cmp  %[v3],%[v5]     \n"
        "   jle  salta2%=        \n"
        "   mov  %[v3],%[v5]     \n"
        "salta2%=:               \n"
        //-----------------------------
        : [v3] "+r"(v3)
        : [v1] "r"(v1), [v2] "r"(v2), [v4] "r"(v4), [v5] "r"(v5)
        :"r8");
  return v3;
}
```

Qual foi o tempo observado na execução?

Tempo 2: _____ s

Reduzindo o código anterior

Macros declaradas com *#define* podem receber *argumentos*, assim como funções verdadeiras. Como vários blocos semelhantes foram usados no programa anterior o objetivo agora é substituí-los.

1. Crie as duas macros e altere a função mediana() como código abaixo:

```
#define TESTA_TROCA(R1, R2, numRotulo) \
  "   cmp  " #R1 "," #R2 "  \n"        \
  "   jle  salta_" #numRotulo "%=\n"   \
  "   mov  r8d," #R1 "      \n"        \
  "   mov  " #R1 "," #R2 "  \n"        \
  "   mov  " #R2 ",r8d      \n"        \
  "salta_" #numRotulo "%=:  \n"

#define MENOR(R1, R2, R3)              \
  "   cmp  " #R1 "," #R2 "  \n"        \
  "   jle  salta1%=         \n"        \
  "   mov  " #R1 "," #R2 "  \n"        \
  "salta1%=:                \n"        \
                                       \
  "   cmp  " #R1 "," #R3 "  \n"        \
  "   jle  salta2%=         \n"        \
  "   mov  " #R1 "," #R3 "  \n"        \
  "salta2%=:                \n"
```

```
int mediana(int v1, int v2, int v3, int v4, int v5) {
  asm(TESTA_TROCA(%[v1], %[v2], 12) //
      TESTA_TROCA(%[v1], %[v3], 13) //
      TESTA_TROCA(%[v1], %[v4], 14) //
      TESTA_TROCA(%[v1], %[v5], 15) //
      //----------------------------
      TESTA_TROCA(%[v2], %[v3], 23) //
      TESTA_TROCA(%[v2], %[v4], 24) //
      TESTA_TROCA(%[v2], %[v5], 25) //
      //----------------------------
      MENOR(%[v3], %[v4], %[v5]) //
      //----------------------------
      : [v3] "+r"(v3)
      : [v1] "r"(v1), [v2] "r"(v2), [v4] "r"(v4), [v5] "r"(v5)
      :"r8");
  return v3;
}
```

Qual foi o tempo observado na execução?

Tempo 3: _____ s

Note que o tempo 3 é igual, ou muito semelhante, ao tempo 2 pois o código gerado é o mesmo. As duas macros declaradas com defines quando usadas, são substituídas pelas respectivas cadeias de caracteres com a substituição dos argumentos. Os argumentos precedidos por '#' são concatenados com as cadeias de caracteres onde eles se inserem.

Usando instrução específica para troca de dados, tentativa 2

Algumas instruções específicas podem ajudar na otimização:

1. Substitua a macro TESTA_TROCA como código abaixo:

```
#define TESTA_TROCA(R1, R2, numRotulo)  \
  "   cmp  " #R1 "," #R2 "  \n"         \
  "   jle  salta_" #numRotulo "%=\n"    \
  "   xchg " #R1 "," #R2 "  \n"         \
  "salta_" #numRotulo "%=:  \n"
```

Qual foi o tempo observado na execução?

Tempo 4: _____ s

A instrução xchg troca os valores de dois registradores, mas será que no seu computador essa troca foi efetiva na otimização? **Compare com o tempo 3**.

Otimizado sem usar desvios na mediana, tentativa 3

Algumas instruções pode se executadas condicionalmente sem utilizar desvios. No caso usou-se a instrução CMOVcc.

1. Substitua as macros TESTA_TROCA e MENOR como código abaixo:

```
#define TESTA_TROCA(R1, R2)             \
      "   cmp   " #R1 "," #R2 " \n"     \
      "   cmovg   r8d,  " #R2 " \n"     \
      "   cmovg " #R2 "," #R1 " \n"     \
      "   cmovg " #R1 ",  r8d   \n"

#define MENOR(R1, R2, R3)      \
      "   cmp   " #R1 "," #R2 " \n"     \
      "   cmovg " #R1 "," #R2 " \n"     \
      "   cmp   " #R1 "," #R3 " \n"     \
      "   cmovg " #R1 "," #R3 " \n"
```

2. Como foi retirado da macro TESTA_TROCA o argumento para diferenciar os rótulos de destinos de saltos, é necessário substituir também a função mediana(). Substitua a função mediana como código abaixo:

```
int mediana(int v1, int v2, int v3, int v4, int v5) {
  asm(
      TESTA_TROCA(%[v1], %[v2]) //
      TESTA_TROCA(%[v1], %[v3]) //
      TESTA_TROCA(%[v1], %[v4]) //
      TESTA_TROCA(%[v1], %[v5]) //
      //----------------------------
      TESTA_TROCA(%[v2], %[v3]) //
      TESTA_TROCA(%[v2], %[v4]) //
      TESTA_TROCA(%[v2], %[v5]) //
      //----------------------------
      MENOR(%[v3], %[v4], %[v5]) //
      //----------------------------
      : [v3] "+r"(v3)
      : [v1] "r"(v1), [v2] "r"(v2), [v4] "r"(v4), [v5] "r"(v5)
      :"r8");
  return v3;
}
```

   Qual foi o tempo observado na execução?

   Tempo 5: _____ s

A troca dos valores de dois registradores foi feita agora usando a instrução MOV condicional, Isto reduzirá os desvios aumentando a possibilidade de otimização, mas, novamente, será que no seu computador esta troca foi efetiva na otimização? **Compare com o tempo 4**.

Analisando os resultados

Envie a avaliação dos resultados como descrito no arquivo “Avaliacao Dos Resultados.pdf”. Lembre de comparar também os tempos 3 com 4 e os tempos 4 com 5.

Conclusão

Escolher as melhores instruções e registradores pode ter influência no tempo de execução de um programa.