# Otimização do Programa Trabalho Final

**Desenvolvida por**: Alberto Azevedo Martinez, RA.: 211023337, Raul A. Gonzalez Augusto, RA.: 211023698

Objetivo: Use tudo o que você aprendeu nesta disciplina para otimizar o código fornecido. Você deve otimizar o programa todo, inclusive a parte de aumento e diminuição do brilho.

# Computadores Usados:

Avell High Performance A52 LIV

* Processador: Intel® Core™ i5-10300H CPU @ 2.50GHz
  + Cores: 4
  + Threads: 8
  + Frequência Máxima: 4.50 GHz
  + Cache: 8 MB Intel® Smart Cache
  + Bus Speed: 8 GT/s
* Placa de Vídeo: NVIDIA GeForce GTX 1650 Ti e Intel® UHD Graphics
* RAM: 16 GB DDR4-2666 Dual Channel
* Armazenamento: 500 GB nvme m.2 SSD 2GB/s E 1 Tera HDD 5400 rpm
* Sistema Operacional: Windows 10 pro 64bits

IdeaPad S145-15IGM

* Processador: Intel(R) Celeron(R) N4000 CPU @ 1.10GHz 1.10 GHz
* Placa de Vídeo: Integrada Intel® UHD Graphics 600
* RAM: 4 GB
* Armazenamento: 500 GB
* Sistema Operacional: Windows 10 pro 64bits

# Programas

Compilado no gcc em uma máquina virtual Linux.

# Executando o programa original

Tempos Raul:

51.403008

51.830897

51.771841

51.638618

51.758403

Media: 51.6805534

Tempos Alberto:

96,050790

96,287249

96,133111

96,314200

96,154142

Media: 96,1878984

# Executando o programa versão 1

Nessa versão substituímos funções da biblioteca math por expressões matemáticas.

Tempos Raul:

5.195901

5.209337

5.158869

5.284487

5.192458

Media: 5.2082104

Desempenho em relação ao original: speedup de 9.92x

Desempenho em relação ao original: melhoria de 89,92%

Tempos Alberto:

13,375718

13,470231

13,393704

13,462269

13,262371

Media: 13,3928586

Desempenho em relação ao original: speedup de 7,18x

Desempenho em relação ao original: melhoria de 86,07%

void processa(struct Pixel img[ALTU\_IMG][LARG\_IMG],

              struct Pixel imgSai[ALTU\_IMG][LARG\_IMG]) {

  struct Pixel corEtiqueta1 = {COR1};

  struct Pixel pixelPreto = {PRETO};

  int i, j;

  for (i = 0; i < ALTU\_IMG; i++) {

    for (j = 0; j < LARG\_IMG; j += 1) {

      struct Pixel pixelAtual = img[i][j];

      int dr = corEtiqueta1.r - pixelAtual.r;

      int dg = corEtiqueta1.g - pixelAtual.g;

      int db = corEtiqueta1.b - pixelAtual.b;

      if (dr \* dr + dg \* dg + db \* db < DIST\_COR1 \* DIST\_COR1) {

        pixelAtual.r = SATURA(pixelAtual.r \* 1.2);

        pixelAtual.g = SATURA(pixelAtual.g \* 1.2);

        pixelAtual.b = SATURA(pixelAtual.b \* 1.2);

        imgSai[i][j] = pixelAtual;

      } else {

        pixelAtual.r = SATURA(pixelAtual.r \* 0.3);

        pixelAtual.g = SATURA(pixelAtual.g \* 0.3);

        pixelAtual.b = SATURA(pixelAtual.b \* 0.3);

        imgSai[i][j] = pixelAtual;

      }

    }

  }

}

# Executando o programa versão 2

Nessa versão buscamos desenrolar os laços for.

Tempos Raul:

4.688014

4.799332

4.702356

4.759145

4.799332

Media: 4.7496358

Desempenho em relação ao original: speedup de 10.88x

Desempenho em relação ao original: melhoria de 90.80%

Tempos Alberto:

13,215919

13,107064

13,096351

13,041478

13,113989

Media: 13,1149602

Desempenho em relação ao original: speedup de 7,33x

Desempenho em relação ao original: melhoria de 86,37%

void processa(struct Pixel img[ALTU\_IMG][LARG\_IMG],

              struct Pixel imgSai[ALTU\_IMG][LARG\_IMG]) {

  struct Pixel corEtiqueta = {COR1};

  struct Pixel pixelPreto = {PRETO};

  int i, j;

  for (i = 0; i < ALTU\_IMG; i++) {

    for (j = 0; j < LARG\_IMG; j += 3) {

      struct Pixel pixelAtual = img[i][j];

      int dr = corEtiqueta.r - pixelAtual.r;

      int dg = corEtiqueta.g - pixelAtual.g;

      int db = corEtiqueta.b - pixelAtual.b;

      if (dr \* dr + dg \* dg + db \* db < DIST\_COR1 \* DIST\_COR1) {

        pixelAtual.r = SATURA(pixelAtual.r \* 1.2);

        pixelAtual.g = SATURA(pixelAtual.g \* 1.2);

        pixelAtual.b = SATURA(pixelAtual.b \* 1.2);

        imgSai[i][j] = pixelAtual;

      } else {

        pixelAtual.r = SATURA(pixelAtual.r \* 0.3);

        pixelAtual.g = SATURA(pixelAtual.g \* 0.3);

        pixelAtual.b = SATURA(pixelAtual.b \* 0.3);

        imgSai[i][j] = pixelAtual;

      }

      pixelAtual = img[i][j+1];

      dr = corEtiqueta.r - pixelAtual.r;

      dg = corEtiqueta.g - pixelAtual.g;

      db = corEtiqueta.b - pixelAtual.b;

      if (dr \* dr + dg \* dg + db \* db < DIST\_COR1 \* DIST\_COR1) {

        pixelAtual.r = SATURA(pixelAtual.r \* 1.2);

        pixelAtual.g = SATURA(pixelAtual.g \* 1.2);

        pixelAtual.b = SATURA(pixelAtual.b \* 1.2);

        imgSai[i][j+1] = pixelAtual;

      } else {

        pixelAtual.r = SATURA(pixelAtual.r \* 0.3);

        pixelAtual.g = SATURA(pixelAtual.g \* 0.3);

        pixelAtual.b = SATURA(pixelAtual.b \* 0.3);

        imgSai[i][j+1] = pixelAtual;

      }

      pixelAtual = img[i][j+2];

      dr = corEtiqueta.r - pixelAtual.r;

      dg = corEtiqueta.g - pixelAtual.g;

      db = corEtiqueta.b - pixelAtual.b;

      if (dr \* dr + dg \* dg + db \* db < DIST\_COR1 \* DIST\_COR1) {

        pixelAtual.r = SATURA(pixelAtual.r \* 1.2);

        pixelAtual.g = SATURA(pixelAtual.g \* 1.2);

        pixelAtual.b = SATURA(pixelAtual.b \* 1.2);

        imgSai[i][j+2] = pixelAtual;

      } else {

        pixelAtual.r = SATURA(pixelAtual.r \* 0.3);

        pixelAtual.g = SATURA(pixelAtual.g \* 0.3);

        pixelAtual.b = SATURA(pixelAtual.b \* 0.3);

        imgSai[i][j+2] = pixelAtual;

      }

    }

  }

}

# Executando o programa versão 3

Nessa versão transformamos os valores em inteiros

Tempos Raul:

4.556360

4.563521

4.585772

4.583445

4.595940

Media: 4.5770076

Desempenho em relação ao original: speedup de 11.29x

Desempenho em relação ao original: melhoria de 91.14%

Tempos Alberto:

Media:

Desempenho em relação ao original: speedup de x

Desempenho em relação ao original: melhoria de %

#define SATURA(x) x > 261120 ? 255 : (x < 0 ? 0 : x / 1024);

void processa(struct Pixel img[ALTU\_IMG][LARG\_IMG],

              struct Pixel imgSai[ALTU\_IMG][LARG\_IMG]) {

  struct Pixel corEtiqueta = {COR1};

  struct Pixel pixelPreto = {PRETO};

  int i, j;

  for (i = 0; i < ALTU\_IMG; i++) {

    for (j = 0; j < LARG\_IMG; j += 3) {

      struct Pixel pixelAtual = img[i][j];

      int dr = corEtiqueta.r - pixelAtual.r;

      int dg = corEtiqueta.g - pixelAtual.g;

      int db = corEtiqueta.b - pixelAtual.b;

      if (dr \* dr + dg \* dg + db \* db < DIST\_COR1 \* DIST\_COR1) {

        pixelAtual.r = SATURA(pixelAtual.r \* 1228);

        pixelAtual.g = SATURA(pixelAtual.g \* 1228);

        pixelAtual.b = SATURA(pixelAtual.b \* 1228);

        imgSai[i][j] = pixelAtual;

      } else {

        pixelAtual.r = SATURA(pixelAtual.r \* 307);

        pixelAtual.g = SATURA(pixelAtual.g \* 307);

        pixelAtual.b = SATURA(pixelAtual.b \* 307);

        imgSai[i][j] = pixelAtual;

      }

      pixelAtual = img[i][j+1];

      dr = corEtiqueta.r - pixelAtual.r;

      dg = corEtiqueta.g - pixelAtual.g;

      db = corEtiqueta.b - pixelAtual.b;

      if (dr \* dr + dg \* dg + db \* db < DIST\_COR1 \* DIST\_COR1) {

        pixelAtual.r = SATURA(pixelAtual.r \* 1228);

        pixelAtual.g = SATURA(pixelAtual.g \* 1228);

        pixelAtual.b = SATURA(pixelAtual.b \* 1228);

        imgSai[i][j+1] = pixelAtual;

      } else {

        pixelAtual.r = SATURA(pixelAtual.r \* 307);

        pixelAtual.g = SATURA(pixelAtual.g \* 307);

        pixelAtual.b = SATURA(pixelAtual.b \* 307);

        imgSai[i][j+1] = pixelAtual;

      }

      pixelAtual = img[i][j+2];

      dr = corEtiqueta.r - pixelAtual.r;

      dg = corEtiqueta.g - pixelAtual.g;

      db = corEtiqueta.b - pixelAtual.b;

      if (dr \* dr + dg \* dg + db \* db < DIST\_COR1 \* DIST\_COR1) {

        pixelAtual.r = SATURA(pixelAtual.r \* 1228);

        pixelAtual.g = SATURA(pixelAtual.g \* 1228);

        pixelAtual.b = SATURA(pixelAtual.b \* 1228);

        imgSai[i][j+2] = pixelAtual;

      } else {

        pixelAtual.r = SATURA(pixelAtual.r \* 307);

        pixelAtual.g = SATURA(pixelAtual.g \* 307);

        pixelAtual.b = SATURA(pixelAtual.b \* 307);

        imgSai[i][j+2] = pixelAtual;

      }

    }

  }

}

# PROGRAMA FINAL

#include <ctype.h>

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#define LARG\_IMG 2560

#define ALTU\_IMG 1600

struct Pixel {

  unsigned char r, g, b;

};

struct PixelHSI {

  unsigned char h, s, i;

};

struct Pixel imagem[ALTU\_IMG][LARG\_IMG];

struct Pixel imagemSai[ALTU\_IMG][LARG\_IMG];

char \*leLinha(FILE \*f) {

  static char str[1000];

  unsigned char c;

  int j = 0;

  c = fgetc(f);

  while (c == '#') {

    do {

      c = fgetc(f);

    } while (c != '\n');

    while (isspace(c = fgetc(f)))

      ;

  }

  do {

    str[j] = c;

    j++;

  } while (!isspace(c = fgetc(f)));

  str[j] = 0;

  return str;

}

void lePPm(char nomeArq[], struct Pixel img[ALTU\_IMG][LARG\_IMG]) {

  int larg, altu, cores, tam;

  char \*p;

  FILE \*f = fopen(nomeArq, "rb");

  if (!f) {

    printf("O arquivo da imagem não pode ser aberto para leitura.\n");

    exit(1);

  }

  if (strcmp(leLinha(f), "P6")) {

    printf("Não é um arquivo no formato PPM (P6) de imagem.\n");

    exit(1);

  }

  p = leLinha(f);

  larg = atoi(p);

  if (LARG\_IMG != larg) {

    printf("A imagem apresenta largura %d, mas o valor desejado e' %d.\n", larg,

           LARG\_IMG);

    exit(1);

  }

  p = leLinha(f);

  altu = atoi(p);

  if (ALTU\_IMG != altu) {

    printf("A imagem apresenta altura %d, mas o valor desejado e' %d.\n", altu,

           ALTU\_IMG);

    exit(1);

  };

  p = leLinha(f);

  cores = atoi(p);

  if (cores != 255) {

    printf("A imagem tem um valor máximo de %d, mas a valor de cor é de 255.\n",

           cores);

    exit(1);

  }

  tam = LARG\_IMG \* ALTU\_IMG;

  if (fread(&img[0][0], 3, tam, f) != tam) {

    printf("O arquivo da imagem parace nao estar completo.\n");

    exit(1);

  }

  fclose(f);

}

void salvaPPM(char nomeArq[], struct Pixel img[ALTU\_IMG][LARG\_IMG]) {

  int tam;

  FILE \*f = fopen(nomeArq, "wb");

  fprintf(f, "P6\n%d %d\n255\n", LARG\_IMG, ALTU\_IMG);

  tam = LARG\_IMG \* ALTU\_IMG;

  if (fwrite(&img[0][0], 3, tam, f) != tam) {

    printf("O arquivo de imagem nao foi gravado corretamente.\n");

    exit(1);

  }

  fclose(f);

}

#define PRETO 0, 0, 0

#define CINZA 128, 128, 128, 50

#define AZUL\_CLARO 0, 210, 250

#define DIST\_AZUL\_CLARO 180

#define AZUL 0, 40, 140

#define DIST\_AZUL 80

#define VERDE\_CLARO 100, 180, 0

#define DIST\_VERDE\_CLARO 60

#define VERDE 0, 110, 0

#define DIST\_VERDE 90

#define LARANJA 230, 100, 0

#define DIST\_LARANJA 30

#define AMARELO 240, 190, 0

#define DIST\_AMARELO 40

#define COR1 AZUL\_CLARO

#define DIST\_COR1 DIST\_AZUL\_CLARO

#define SATURA(x) x > 261120 ? 255 : (x < 0 ? 0 : x / 1024);

void processa(struct Pixel img[ALTU\_IMG][LARG\_IMG],

              struct Pixel imgSai[ALTU\_IMG][LARG\_IMG]) {

  struct Pixel corEtiqueta = {COR1};

  struct Pixel pixelPreto = {PRETO};

  int i, j;

  for (i = 0; i < ALTU\_IMG; i++) {

    for (j = 0; j < LARG\_IMG; j += 3) {

      struct Pixel pixelAtual = img[i][j];

      int dr = corEtiqueta.r - pixelAtual.r;

      int dg = corEtiqueta.g - pixelAtual.g;

      int db = corEtiqueta.b - pixelAtual.b;

      if (dr \* dr + dg \* dg + db \* db < DIST\_COR1 \* DIST\_COR1) {

        pixelAtual.r = SATURA(pixelAtual.r \* 1228);

        pixelAtual.g = SATURA(pixelAtual.g \* 1228);

        pixelAtual.b = SATURA(pixelAtual.b \* 1228);

        imgSai[i][j] = pixelAtual;

      } else {

        pixelAtual.r = SATURA(pixelAtual.r \* 307);

        pixelAtual.g = SATURA(pixelAtual.g \* 307);

        pixelAtual.b = SATURA(pixelAtual.b \* 307);

        imgSai[i][j] = pixelAtual;

      }

      pixelAtual = img[i][j+1];

      dr = corEtiqueta.r - pixelAtual.r;

      dg = corEtiqueta.g - pixelAtual.g;

      db = corEtiqueta.b - pixelAtual.b;

      if (dr \* dr + dg \* dg + db \* db < DIST\_COR1 \* DIST\_COR1) {

        pixelAtual.r = SATURA(pixelAtual.r \* 1228);

        pixelAtual.g = SATURA(pixelAtual.g \* 1228);

        pixelAtual.b = SATURA(pixelAtual.b \* 1228);

        imgSai[i][j+1] = pixelAtual;

      } else {

        pixelAtual.r = SATURA(pixelAtual.r \* 307);

        pixelAtual.g = SATURA(pixelAtual.g \* 307);

        pixelAtual.b = SATURA(pixelAtual.b \* 307);

        imgSai[i][j+1] = pixelAtual;

      }

      pixelAtual = img[i][j+2];

      dr = corEtiqueta.r - pixelAtual.r;

      dg = corEtiqueta.g - pixelAtual.g;

      db = corEtiqueta.b - pixelAtual.b;

      if (dr \* dr + dg \* dg + db \* db < DIST\_COR1 \* DIST\_COR1) {

        pixelAtual.r = SATURA(pixelAtual.r \* 1228);

        pixelAtual.g = SATURA(pixelAtual.g \* 1228);

        pixelAtual.b = SATURA(pixelAtual.b \* 1228);

        imgSai[i][j+2] = pixelAtual;

      } else {

        pixelAtual.r = SATURA(pixelAtual.r \* 307);

        pixelAtual.g = SATURA(pixelAtual.g \* 307);

        pixelAtual.b = SATURA(pixelAtual.b \* 307);

        imgSai[i][j+2] = pixelAtual;

      }

    }

  }

}

int main() {

  int j;

  clock\_t inicio, final;

  double duracao;

  lePPm("Lapis.ppm", imagem);

  printf("Iniciando processamento:\n");

  inicio = clock();

  for (j = 0; j < 120; j++)

    processa(imagem, imagemSai);

  final = clock();

  duracao = (double)(final - inicio) / CLOCKS\_PER\_SEC;

  printf("Tempo utilizado no processamento = %f\n", duracao);

  salvaPPM("LapisSai.ppm", imagemSai);

  return 0;

}