Programação Concorrente

2024/2025

Projeto – Parte A

O processamento computacional de grandes conjuntos de dados é normalmente composto por um conjunto de tarefas de elevada complexidade temporal. No entanto, muitas vezes pode-se tirar partido da existência de múltiplos CPUs ou *Cores* para reduzir o tempo total de processamento.

Um desses exemplos é a conversão e processamento de imagens. Normalmente em bancos de imagens, todas as imagens sofrem um conjunto de conversões igual: criação de uma cópia com menor resolução, criação de um *thumbnail*, ou mesmo aplicação de feitos, por exemplo. Estes passos são aplicados de forma igual a todas as imagens ou fotografias.

Neste projeto os alunos irão paralelizar um conjunto de tarefas (processamento de imagens) de modo a tirar partido dos vários *Cores* de um computador e reduzir o tempo total de processamento.

1 Descrição da parte A do projeto

Neste projeto os alunos deverão paralelizar a aplicação fornecida (**old-photo-serial.c**), de modo a reduzir o tempo de processamento de conjuntos de imagens.

A aplicação resultante deverá efetuar as 4 transformações a uma séria de imagens armazenadas numa diretoria para produzir imagens com efeito antigo. Esta aplicação chamar-se-á **old-photo-parallel-A**.

De modo a acelerar o processo e reduzir o tempo de processamento deverão ser usadas *threads*.

1.1 Funcionamento geral

Ao executar a aplicação desenvolvida, o utilizador indica na linha de comandos o nome da pasta onde se encontram as imagens a serem processadas. A aplicação, depois de saber que imagens se encontram na pasta indicada, seleciona as terminadas em **.jpeg** e, para cada uma destas imagens, produz uma imagem com efeito antigo. As imagens resultantes terão o nome da imagem original, mas serão armazenadas em diretorias específicas tal como descrito na Secção 2.2

1.2 Paralelização

A aplicação paralela a desenvolver (chamada **old-photo-parallel-A**) tem apenas um tipo de *threads. Cada thread* processa um subconjunto disjunto de imagens e para cada uma das imagens a si atribuídas gera a imagem envelhecida (exatamente como no **old-photo-serial.c**). O número de *threads* criadas é definido pelo utilizador através de um argumento da linha de comandos. O utilizador também define por que ordem as imagens são guardadas inicialmente na memória (alfabeticamente ou por tamanho crescente)

Depois de saber os nomes das imagens que se encontram na pasta e ordená-las (por tamanho ou alfabeticamente), o `main` cria um conjunto de threads e atribui-lhes um subconjunto do trabalho a realizar, como ilustra a Figura 1.

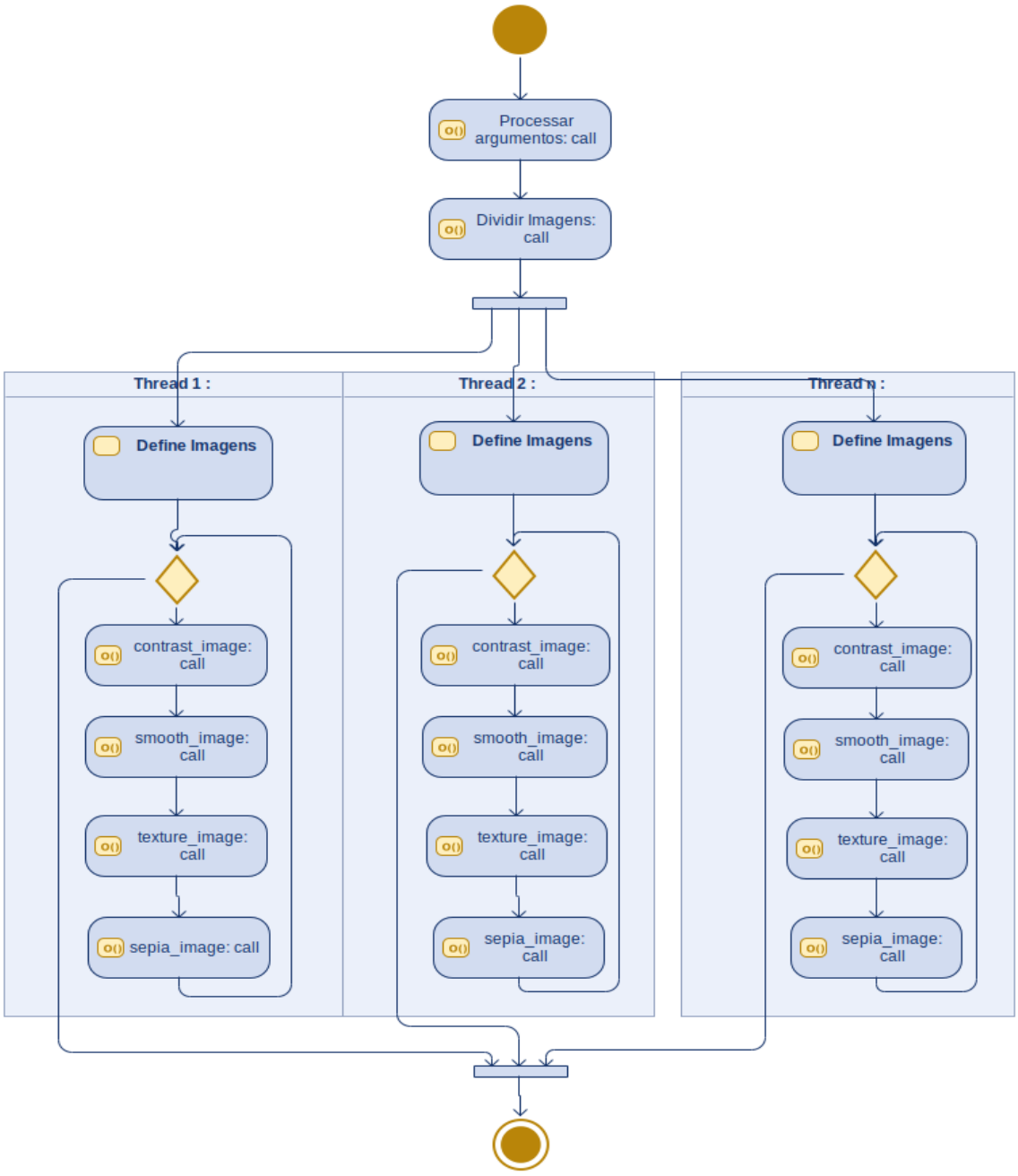

Figura 1: Fluxo de execução da paralelização (old-photo-parallel-A)

Cada *thread* executa as quatro transformação nas imagens que lhe foram atribuídas. Após a criação das *threads*, o *main* deverá esperar pela terminação de todas as *threads* antes de sair. A Figura 1 não identifica o local onde o programa fará a leitura e escrita dos ficheiros, devendo os alunos definir a localização no código dessas operações.

2 Funcionamento da aplicação paralela

A aplicação deverá ser desenvolvida em **C** e executará em Linux, WSL ou MAC OS X.

2.1 Argumentos da linha de comandos

Para a execução da aplicação **old-photo-parallel-A** o utilizador deverá sempre indicar através dos argumentos da linha de comandos o seguinte (pela ordem indicada):

- a diretoria onde se encontram as imagens
- número de *threads* a criar
- **-name** ou -**size** que indicam a ordem pela qual as imagens são organizadas antes de serem divididas pelas threads

como exemplificado de seguida.

```
./old-photo-parallel-A ./dir-1 4 -size
./old-photo-parallel-A ./dir-2 8 -name
./old-photo-parallel-A . 1 -name
```

A pasta onde se encontram as imagens pode ser uma relativa ao local onde o programa é executado (começando por **./**) ou absoluta se começar por **/** .

O número de *threads* deve ser um qualquer número inteiro positivo e indica quantas *threads* efetuarão o processamento das imagens. Se, por exemplo, o utilizador indicar **1** como o número de *threads* a criar, o programa utilizará apenas uma *thread* **para além** do `main()` para efetuar o processamento de todas as imagens.

O programa deverá processar todas as imagens cujo extensão é **,jpeg** que se encontrem na pasta indicada no primeiro argumento, ignorando todos os outros ficheiros.

Antes de criar a *threads* e dividir o trabalho por elas, o **main** deverá armazenar em memória o nome de todas as imagens **.jpeg** existente na pasta indicada. Os nomes dessas imagens deverão ser ordenados em memória por ordem alfabética (se o 3 argumento for **-name**) ou por tamanho (se o 3. argumento for **-size**). Só depois desta ordenação, se deverá proceder à divisão das imagens pelas *threads*. O algoritmo de divisão das imagens por *threads* deverá ser decidido pelos alunos.

Serão fornecidos conjuntos de imagens de modo que os alunos tenham dados variáveis. Os alunos podem naturalmente utilizar outras imagens durante o desenvolvimento e teste do projeto.

2.2 Resultados

A execução da aplicação produz um conjunto de novas imagens, correspondentes à transformação de cada uma das imagens iniciais.

O nome das novas imagens será o mesmo da imagem original, mas colocadas numa diretoria específica e relativa à pasta indicada na linha de comandos. Essa diretoria dever-se-á chamar **old_photo_PAR_A** e deverá ser criada pela aplicação a desenvolver na diretoria indicada na linha de comandos pelo utilizador onde se encontram as imagens originais.

A imagem seguinte exemplifica a estrutura dos ficheiros e pastas antes e depois de executar a aplicação com os seguintes argumentos:

```
./old-photo-parallel-A ./pasta-1 3 -size
```

2.3 Interrupção de execução

Se a aplicação for interrompida a meio do processamento das imagens, apenas parte dos resultados terão sido produzidos e guardados no disco.

Se o utilizador voltar a executar a aplicação, não deverá ser necessário voltar a produzir os ficheiros resultado já existentes. A aplicação só deverá processar e gastar tempo na criação dos ficheiros em falta.

Para verificar se um ficheiro existe os alunos poderão usar as funções *`access()`* como no seguinte exemplo:

```
if( access( nome_fich, F_OK ) != -1){
   printf("%s encontrado\n", nome_fich);
}else{
   printf("%s nao encontrado\n", nome_fich);
}
```

3 Avaliação de desempenho.

O relatório a produzir pelos alunos no final do trimestre deverá conter os resultados da avaliação de desempenho (tempos de execução e speedups) do programa e de cada uma das *threads.* Para tal, será necessário instrumentar o código para recolher os tempos de processamento do programa e todas as *threads*.

Os alunos deverão incluir no código do **old-photo-parallel-A.c** as funções de leitura de tempos (como no laboratório 3) e em cada execução produzir um ficheiro com esses resultados.

Este ficheiro dever-se-á chamar **timing_<n>-size.txt** ou **timing_<n>-name.txt** e deverá ser criado na pasta onde se encontram as imagens. O **n** deverá ser substituído pelo número de *threads* e **-size** ou **-name** dependerá do último argumento da linha de comandos.

4 Submissão do projeto

4.1 Prazo de submissão

O prazo para submissão da resolução da Parte A do projeto é dia **8 de Dezembro às 19h00** no FENIX.

Antes da submissão, os alunos devem criar grupos de dois alunos e registá-los no FENIX.

4.2 Ficheiros a submeter

Os alunos deverão submeter um ficheiro **.zip** contendo:

- todo o código da aplicação **old-photo-parallel-A**
- a `Makefile` para a compilação da aplicação

Não incluir no .zip as imagens dos *datasets* utilizadas.

5 Avaliação do projeto

A nota para esta parte do projeto será dada tendo em consideração o seguinte:

- Funcionalidades implementadas
- Modo de gestão das *threads* e recursos
- Estrutura e organização do código
- Tratamento de erros
- Comentários

Anexos

6 Código de exemplo de manipulação de imagens

Em anexo a este enunciado são fornecidas quatro aplicações que processam um conjunto de ficheiros (**apply_contrast.c**, **apply_sepia.c**, **apply_smooth.c** e **apply_texture.c**) na diretoria **gd-examples/**, e uma quinta aplicação (**old-photo-serial.c**) na diretoria **serial-version/** que implementa de forma sequencial (usando apenas um C*ore*/*CPU*) todo o processamento necessário para produzir fotografias com aspeto antigo.

Estas aplicações estão desenvolvidas em C e usam a biblioteca `libGD`[1] para leitura, manipulação e armazenamento das imagens.

As aplicações na pasta **gd-examples/** iteram sobre um vetor de strings (contendo os nomes dos ficheiros) e, para cada ficheiro, efetuam o seguinte:

- **apply_contrast.c**
  - leitura das imagens a serem processadas através da função **`read_jpeg_file()`**
  - criação de uma cópia de cada imagem com maior contraste através da chamada da função **`contrast_image()`**
  - gravação de todas as novas imagens na directoria **`./Contrast-dir/`** através da função **`write_jpeg_file()`**
- **apply_sepia.c**
  - leitura das imagens a serem processadas através da função **`read_jpeg_file()`**
  - criação de uma cópia de cada imagem mas amarelada/sépia através da chamada da função **`sepia_image()`**

1 https://libGD.github.io/

  - gravação de todas as novas imagens na directoria **`./Sepia-dir/`** através da função **`write_jpeg_file()`**
- **apply_smooth.c**
  - leitura das imagens a serem processadas através da função **`read_jpeg_file()`**
  - criação de uma cópia de cada imagem mas suavizada através da chamada da função **`smooth_image()`**
  - gravação de todas as novas imagens na directoria **`./Smooth-dir/`** através da função **`write_jpeg_file()`**
- **apply_texture.c**
  - leitura das imagens a serem processadas através da função **`read_jpeg_file()`**
  - criação de uma cópia de cada imagem com um efeito/textura de envelhecimento através da chamada da função **`texture_image()`** . Esta função rebe um png correspondente à textura a plica na image que no esxemplos se encontra no ficheiro **`paper-texture.png`** .
  - gravação de todas as novas imagens na directoria **`./Texture-dir/`** através da função **`write_jpeg_file()`**

A quinta aplicação (na pasta **old-photo-serial/**) processa de forma sequencial o mesmo conjunto de imagens e aplica a cada uma delas, em sequência, as seguintes transformações para conseguir criar novas imagens com efeito antigo:
contrast_image → smooth_image → texture_image → sepia_image

Como estas aplicações apenas processam um conjunto pré-definido de imagens (declarado no código numa variável), deverão apenas ser utilizadas para:

- **apply_contrast.c, apply_sepia.c, apply_smooth.c e apply_texture.c**
  - perceber os diversos processamentos e transformações a serem realizadas pelas versões paralelas
  - extrair e reutilizar as funções fundamentais para o processamento das imagens
- **old-photo-serial.c**

- referência para verificar o correto funcionamento da versão paralela
- referência para comparar os ganhos das versões paralelas

São fornecidos dois ficheiros auxiliares (**image-lib.c** e **image-lib.h**) que contêm todas as funções necessárias à manipulação e transformação das imagens. De modo a simplificar a compilação dos exemplos, são também disponibilizadas duas **Makefiles**, em cada uma das pastas.

Para executar as quatro aplicações fornecidas é necessário copiar os ficheiros dos *datasets* 1 e 2 para a diretoria onde se encontram as aplicações.

7 Instalação da biblioteca libGD

Para a execução dos programas fornecidos e realização do projeto é necessário instalar a biblioteca `libGD`.

Em Ubuntu/Debian basta executar o seguinte comando:

```
sudo apt install libgd-dev
```

Noutras distribuições de Linux o nome do pacote e comando são diferentes:

- centos / RedHat – `sudo dnf install gd-devel.x86_64`
- OpenSuse - `zypper install gd-devel`

Para MAC OS X a biblioteca terá de ser instalada através do comando brew:

```
brew install gd
```

como descrito em:

https://formulae.brew.sh/formula/gd

8 Compilação com libGD

De modo a compilar programas que usem a biblioteca libGD é necessário fazer o seguinte `include`:

```
#include <gd.h>
```

Aquando da compilação do programa final é necessário adicionar a opção **`-lgd`** para além de todas as opções normalmente usadas:

```
gcc -g programa.c -o programa -lgd
```

8.1 Descrição da biblioteca

A biblioteca libGD usa um tipo de dados (*`gdImagePtr`*) que permite armazenar e representar imagens de diversos formatos (jpg, tiff, png). As variáveis deste tipo são manipuladas por uma série de funções que permitem carregar as imagens para memória a partir do disco, manipulá-las, ou mesmo gravar as imagens de volta para o disco.

8.2 Leitura e escrita de imagens a partir do disco

A biblioteca `libGD` tem funções que leem imagens de diversos formatos com o apoio do *`fopen()`*: o programa abre o ficheiro com a imagem usando *`fopen()`*, e usa o *`FILE *`* retornado como argumento da função de leitura. Depois de lida a imagem para a variável do tipo *`gdImagePtr`*, o ficheiro pode ser fechado com *`fclose()`*.

A escrita é feita de forma semelhante, sendo necessário abrir o ficheiro em modo de escrita e a invocação de uma função específica.

Apresentam-se de seguida duas funções que encapsulam estes processos. A função de leitura (*`read_jpeg_file()`*) recebe um nome de um ficheiro do tipo `JPEG` e retorna a *`gdImagePtr`*. Esta função retorna *`NULL`* em caso de erro (impossível abrir o ficheiro ou ler o seu conteúdo).

A função de escrita (*`write_jpeg_file()`*) recebe um argumento do tipo *`gdImagePtr`* e um nome e grava essa imagem em disco num ficheiro com esse nome. Esta função retorna 1 em caso de sucesso.

A seguinte tabela apresenta pedaços de código que demonstram a leitura e escrita de imagens. Observe que se abre o ficheiro com a função *fopen()* e que o *FILE** retornado é depois usado para ler a imagem (*gdImageCreateFromJpeg()*) ou escrevê-la em disco (*gdImageJpeg()*).

```
gdImagePtr read_jpeg_file(
                char * file_name){
  FILE * read_fp;
  gdImagePtr read_img;
  fp = fopen(file_name, "rb");
  if (!fp) {
     return NULL;
  }
  read_img= gdImageCreateFromJpeg(fp);
  fclose(fp);
  if (!read_img) {
     return NULL;
  }
  return read_img;
}
```

```
int write_jpeg_file(
             gdImagePtr write_img,
             char * file_name){
   FILE * fp;
   fp = fopen(file_name, "wb");
   if (!fp) {
      return 0;
   }
   gdImageJpeg(write_img, fp);
   fclose(fp);
   return 1;
}
```

8.3 Funções de manipulação

As várias funções de manipulação das imagens recebem para além do argumento referente à imagem que irá ser manipulada outros parâmetros relevantes.

Foram desenvolvidas 4 funções que encapsulam as funções do libGD necessárias à realização do projeto:

```
gdImagePtr increase_contrast_image(gdImagePtr in_img)
```

Esta função recebe como argumento uma imagem (*gdImagePtr in_img*) e aumenta o contraste das suas cores. Esta função retorna uma nova imagem.

```
gdImagePtr smooth_image(gdImagePtr in_img)
```

Esta função recebe como argumento uma imagem (*gdImagePtr in_img*) e suaviza-a, reduzindo o detalhe da mesma. Esta função retorna uma nova imagem.

```
gdImagePtr apply_texture(gdImagePtr in_img, gdImagePtr texture_img)
```

Esta função recebe duas imagens e sobrepõe a imagem *texture_img* sobre a imagem *im_img*. Esta função retorna uma nova imagem.

```
ggdImagePtr sepia_image(gdImagePtr in_img)
```

Esta função recebe como argumento uma imagem (*gdImagePtr in_img*) e transforma a sua cor, amarelando-a como o papel antigo. Esta função retorna uma nova imagem.

8.4 Gestão de memória

As variáveis do tipo *gdImagePtr* apontam para uma estrutura que armazena toda a informação de uma imagem. A memória é alocada quando se lê do ficheiro ou quando se aplicam funções de transformação.

Quando deixam de ser necessárias, as imagens têm de ser libertadas. Para realizar essa operação deve ser usada a função *gdImageDestroy()*.

Como descrito atrás, todas as funções auxiliares criam e retornam uma nova imagem correspondente à transformação da imagem original. Assim é necessário que logo após as imagens deixem de ser necessárias, sejam destruídas com o *gdImageDestroy()*.

9 Listagem de ficheiros

Na pasta **dir-processing** é fornecida aos alunos uma sexta aplicação que lista o nome dos ficheiros de uma pasta e permite aceder ao seu tamanho a partir de código C.