Programação concorrente

2021/2022

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 recursos e unidades computacionais para reduzir o tempo total de processamento.

Um desses exemplos é a conversão e processamentos de imagens: normalmente em bancos de imagens todas as imagens sofrem um conjunto de conversões igual: criação de uma copia com menor resolução, criação de um *thumbnail*, ou mesmo aplicação de marcas de água. Estes passos são aplicados de forma igual a todas as imagens ou fotografias.

Neste projeto os alunos irão paralelizar um programa que efetua esse processamento de modo a tirar partido dos vários *cores* de um computador e reduzir o tempo de processamento.

1 Versão sequencial

Em anexo a este enunciado, são fornecidas duas aplicação que processam um conjunto de ficheiros, aplicando a cada ficheiro as mesmas transformações para produzir novas imagens:

- redução da dimensão,
- criação de um *thumbnail* quadrado,
- aplicação de uma marca de água.

Como estas aplicações apenas processam um conjunto pré-definido de imagens, deverão apenas ser utilizadas para:

- perceber as diversas tarefas a serem realizadas pelas versões paralelas
- extração e reutilização da funções fundamentais de processamento dos dados
- referência para verificar o correto funcionamento das verões paralelas
- referência para comparar os ganhos das versões paralelas

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

Estas aplicações iteram sobre um vetor de **strings** contendo os nomes dos ficheiros e, para cada ficheiro, efetuam o seguinte:

- leem do disco imagens de entrada
- criam uma versão reduzida da imagem (chamando a função `resize_image`)
- criam um thumbnail da imagem (chamado a função `make_thumb`)
- aplicam o símbolo do IST sobre a imagem (chamado a função `add_watermark`)
- todas as transformações a cada uma das imagens são gravadas no disco

A grande diferença entre estas aplicações é o fluxo de processamento das imagens originais:

- a aplicação **serial-simples.c** tem apenas um ciclo (que itera sobre todos as imagens de entrada) e em cada iteração efetua as três conversões: redução da imagem, criação do *thumbnail* e aplicação da marca de água;
- A aplicação serial-complex.c tem três ciclos em que cada um deles efetua uma das transformações sobre todas as imagens (iterando sobre o vetor). Estes três ciclos são executados de forma sequencial (começando um clico após a terminação do anterior).

1 https://libGD.github.io/

1.1 Instalação da biblioteca libGD

Para execução do programa fornecido 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`

Em cygwin também existe um pacote que permite instalar a biblioteca:

- `libgd-devel`

1.2 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 programa.c -o programa -lgd
```

1.3 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.

1.3.1 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 *`fopen()`*.

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_png_file()`*) recebe um nome de um ficheiro do tipo `png` 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_png_file()`*) recebe um argumento to tipo *`gdImagePtr`* e um nome e grava essa imagem em disco. Esta função retorna 1 em caso de sucesso.

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

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

1.3.2 Funções de manipulação

As várias funções de manipulação da imagens recebem para além do argumento referente à imagem que irá ser manipulada, outros parâmetros relevantes. Foram desenvolvidas 3 funções que encapsulam as funções do libGD:

```
gdImagePtr  resize_image(gdImagePtr in_img, int new_width)
```

Este função recebe como argumento uma imagem (*`gdImagePtr in_img`*) e escala-a de modo a ficar com largura pretendida (*`new_width`*). Este função retorna uma nova imagem com tamanho diferente, mas com as mesmas proporções.

```
gdImagePtr make_thumb(gdImagePtr in_img, int size)
```

A função *`make_thumb()`* recebe uma imagem (*`in_img`*) e cria um thumbnail quadrado com largura *`size`*. Este função retorna uma nova imagem quadrada.

```
gdImagePtr  add_watermark(gdImagePtr in_img, gdImagePtr watermark)
```

A função *`add_watermark()`* recebe duas imagens e sobrepõe a imagem *`watermark`* no canto superior esquerdo da imagem *`im_img`*. É retornada uma nova imagem.

1.3.3 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()`*.

2 Descrição do projeto

Neste projeto os alunos deverão usar *threads* para implementar duas versões paralelas das aplicações fornecidas, de modo a reduzir o tempo de processamento de conjuntos de imagens. As duas aplicações a desenvolver recebem como argumento da linha de comando o nome duma diretoria onde se encontram as imagens e o número de *threads*

a usar. As aplicações efetuam as três transformações a cada uma das imagens armazenadas na diretoria (produzindo três novos conjuntos de imagens) e, de modo a acelerar o processo e reduzir o tempo de processamento usarão *threads* (no número definido pelo utilizador).

2.1 Paralelização simples

A primeira aplicação paralela (chamada **ap-paralelo-simples**) tem apenas um tipo de *threads. Cada thtread* processa um sub-conjunto das imagens e efetua as três transformações em sequência para essas imagens.

Como ilustra a Figura 1, o `main` divide o trabalho pelo número de *threads* definido pelo utilizador e cria-as. Cada *thread* executa as três transformação para os ficheiros que lhe foram atribuídas e, para esperar pelo fim da threads, o main faz *`pthread_join()`*.

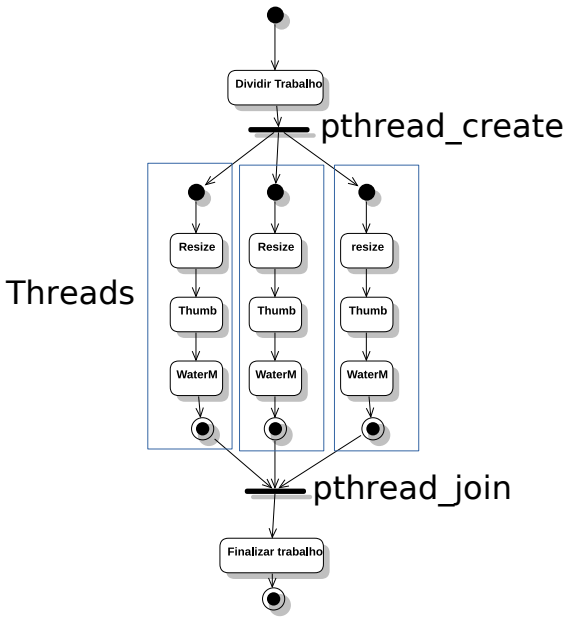


Figura 1: Paralelização simples

2.2 Paralelização complexa

Na segunda aplicação (chamada **ap-paralelo-complexo**), haverá três tipos de *threads*. Cada tipo de thread faz uma transformação específica (como ilustrado na Figura 2).

Também nesta aplicação cada thread processará um sub-conjunto das imagens originais definido pelo `main`.

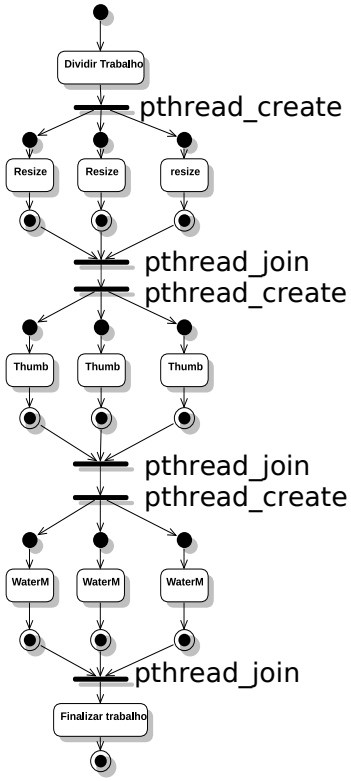


Figura 2: Paralelização complexa

Nesta solução apesar de haver *threads* de 3 tipos distintos, num determinado instante apenas executam em simultâneo *threads* de um tipo: o `main` lança as *threads* responsáveis pela ampliação/redução da imagens e só depois de todas estas *threads* terminarem (esperando com *`pthread_join()`*) lança as seguintes (para fazer os *thumbnails*). O último conjunto de *threads* (para o *watermark*) apenas é lançado depois de todas a *threads* do *thumbnail* terminarem

O `main` terá de esperar por cada conjunto de *threads* num ciclo de *`pthread_join()`*, específico.

Também nesta solução é necessário dividir o trabalho (imagens originais) pelo número de *threads*.

3 Funcionamento das aplicações

As aplicações serão desenvolvida em C e executarão em Linux (ou Cygwin).

Para cada execução, o utilizador deverá indicar através dos argumentos da linha de comando:

- a diretoria onde se encontram as imagens originais,
- o número de *threads.*

Os programas aplicarão as transformações atrás descritas, armazenando as imagens num conjunto de pastas pré-definido. Serão também produzidas estatísticas acerca do tempo de processamento.

3.1 Dados de entrada

A diretoria onde se encontram as imagens originais e o número de *threads* são definidos pelo utilizador na linha de comando aquando da execução das aplicações:

```
ap-paralelo-simples dir-1 4    //    ap-paralelo-complexo dir-2 8
```

O primeiro argumento corresponde à diretoria e o segundo ao número de *threads*.

Se, por exemplo, o utilizador executar o comando

```
app-parallel-simples . 1
```

a aplicação processará imagens na diretoria onde foi executada e utilizará apenas uma *thread* (para além do main).

A diretoria indicada pelo utilizador poderá conter mais imagens do que aquelas a serem processadas. O ficheiro `img-process-list.txt`, contém a lista das imagens a serem processadas (um nome de ficheiro por linha). As aplicações deverão ler este ficheiro e só as imagens aí listadas serão processadas.

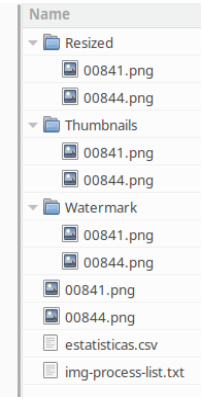
Apenas deverão ser processadas imagens do formato PNG.

Serão fornecidos conjuntos de imagens (com correspondente ficheiro `img-process-list.txt`) de modo a os alunos terem conjuntos de dados variáveis e comparáveis.

3.2 Resultados

A execução das aplicações produz um conjunto de imagens, correspondentes à transformações de cada uma das imagens iniciais. O nome das novas imagens será o mesmo da imagem original, mas cada imagem será colocada numa sub-diretoria específica:

- Resized – sub-diretoria que contém os resultado da redução das imagens
- *Thumbnails* – sub-diretoria que contém os thumbnail produzido
- Watermarks – sub-diretoria que contém a imagens com o watermark aplicado.



3.3 Interrupção de execução

Se as aplicações forem interrompidas a meio do processamento dos ficheiros, 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);
}
```

4 Estatísticas

Durante a execução das aplicações, deverão ser recolhidos os seguintes dados:

- instante de início e fim de cada thread
- instante de início e fim de processamento de cada imagem original (no caso da aplicação **ap-paralelo-simples**)
- instante de início e fim de processamento de cada transformação (no caso da aplicação **ap-paralelo-complexo**)
- instante de início e fim de execução da aplicação

Estas estatísticas deverão ser armazenadas num ficheiro chamado `estisticas.csv` na diretoria onde se encontram as imagens.

Este ficheiro deverá ser do tipo *Comma-separated values*[2].

5 Submissão do projeto

O prazo para submissão da resolução da parte A do projeto é dia **14 de Janeiro às 19h00** no FENIX.

Antes da submissão, os alunos devem criar grupos de 2 e registá-los no FENIX.

Os alunos deverão submeter um ficheiro **.zip** contendo o código de ambas as aplicações. O código para cada uma das aplicações deverá ser colocado numa das seguintes diretorias: **ap-paralelo-simples/** ou **ap-paralelo-complexo/**. Se possível, os alunos deverão entregar também uma `Makefile` para a compilação das aplicações.

Os alunos deverão submeter um pequeno documento/relatório (chamado **pconc-relatorio-1.pdf**). O modelo do relatório será fornecido posteriormente, mas deverá listar

2 https://pt.wikipedia.org/wiki/Comma-separated_values

as funcionalidades implementadas e apresentar os seguintes resultados produzidos com base no ficheiro `estatisticas.csv` para várias execuções:

- tempo total de execução da aplicação
- tempo médio de processamento de cada imagem
- speedup

Para calcular o *speedup*, os alunos devem apenas usar os dados armazenados no ficheiro `estatisticas.csv`.

Os alunos deverão executar as duas aplicações no computador **sigma** com as várias combinações de:

- dois conjuntos de dados diferentes (a ser fornecidos pelos docentes),
- 1, 2, 4, 8 número de threads.

6 Avaliação do projeto

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

- Número de aplicações e funcionalidades implementadas
- Modo de gestão das threads e recursos
- Estrutura e organização do código
- Tratamento de erros
- Comentários
- Relatório