

Manuel Soares 96267

Eduardo Faustino 102298

1 Funcionalidades implementadas

Table 1: Funcionalidades implementadas

	Não implementada	Com falhas	Totalmente correta
ap-paralelo-simples			
Argumentos da linha de comando			x
Leitura do ficheiro img-process-list.txt			x
Criação de <i>threads</i>			x
Distribuição trabalho pelas <i>threads</i>			x
Verificação ficheiro existentes			x
Produção resultados			x
Estatísticas			x
ap-paralelo-complexo			
Argumentos da linha de comando			x
Leitura do ficheiro img-process-list.txt			x
Criação de threads			x
Distribuição trabalho pelas threads			x
Verificação ficheiro existentes			x
Produção resultados			x
Estatísticas			x

1.1 Descrição das funcionalidades com falhas

Apesar de não termos funcionalidades com falhas algumas das funcionalidades foram implementadas com alguns comportamentos que achamos que devem ser partilhados.

A produção de estatísticas é feita para um ficheiro chamado “stats.csv” criado no diretório onde se encontram as imagens.

A invocação do programa com um número de *threads* igual ou inferior a zero faz com que o programa tente utilizar um *thread* por imagem.

2 Estrutura do código

- [main.c] `void *process_image_set(void *args)` e outras começadas por `process_image_set` são as funções que as threads vão executar. Abre o ficheiro, carrega a imagem para memória, aplica as transformações e guarda os resultados,

repete enquanto ainda existirem imagens atribuídas ao thread. No modo complexo, há 3 funções pois as transformações são divididas por 3 threads.

- [filehandler.c] `void create_output_directories(char *output_path)`, cria os directorio de output Resize,Thumbnail e Watermark.
- [filehandler.c] `char *img_path_generator(char *imgs_path, char *subdirectory, char *img_name)`, gera o path final da imagem.
- [filehandler.c] `int list_pngs(char *imgs_path, char ***img_names)`, lista os ficheiros PNG dentro de um ficheiro “img-process-list.txt” e retorna quantos e quais ficheiros foram encontrados.
- [imagehandler.c] `gdImagePtr *create_image_array(int size)`, cria um array para guardar referencias para todas as imagens.
- [imagehandler.c] `image_set *create_image_set(char *imgs_path, char **array, gdImagePtr *image_array, unsigned int array_length, unsigned int start_index, unsigned int thread_count, gdImagePtr watermark, timer_data *thread_timers, timer_data *image_timers)`, cria um struct com todos os argumentos que vão ser passados para os threads.

3 Partição do trabalho

Nesta secção os alunos deverão descrever como é que as várias imagens são particionadas e distribuídas pelas várias *threads*

3.1 Algoritmo de partição

As imagens são atribuídas um índice entre 0 e o número total de imagens a processar, os threads são atribuídos um índice de 0 até ao número total de threads, cada thread é responsável pelas imagens cujo índice é dado por $(n \times \text{numero_threads} + \text{indice_thread})$

Este algoritmo não tem em conta qualquer previsão do tempo que cada thread vai demorar e certifica-se que nenhum thread recebem mais do que uma imagem a mais que todos os outros threads.

Os alunos deverão também preencher a seguinte tabela para o exemplo de um dataset com 10 imagens (Lisboa-1.png até Lisboa-10.png) a ser particionados por 3 *threads*:

Table 2: Distribuição de 10 imagens por 3 threads

	Imagens a ser processadas por cada <i>thread</i>			
	1 imagem	2 imagem	3 imagem	4 imagem
Thread_0	Lisboa-1.png	Lisboa-4.png	Lisboa-7.png	Lisboa-10.png
Thread_1	Lisboa-2.png	Lisboa-5.png	Lisboa-8.png	
Thread_2	Lisboa-3.png	Lisboa-6.png	Lisboa-9.png	

3.2 Envio de informação para as *threads*

Cada thread recebe um struct com os seus argumentos, os vetores de nomes de ficheiros, das imagens em memória e dos timers para estatísticas são únicos e partilhados por todos os threads, no entanto as colisões são evitadas porque em nenhum caso os threads leem ou escrevem para a mesma localização em memória.

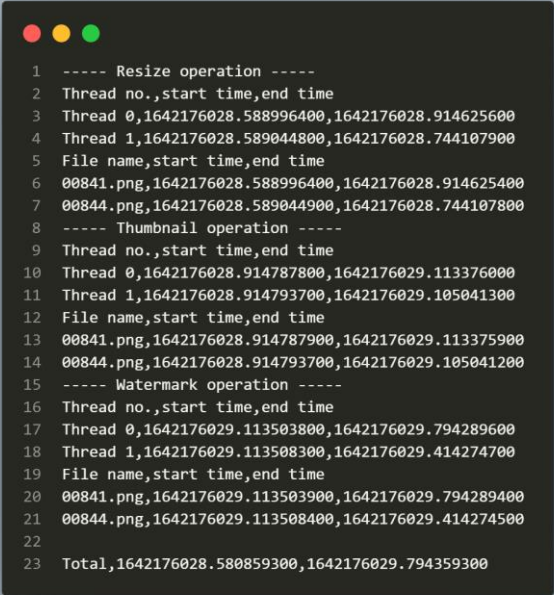
4 Geração de estatísticas

As estatísticas são geradas através da utilização da biblioteca `time.h`, e de *structs* `timer_data` que guardam dois structs `timespec` que guardam a informação do tempo de início e de tempo de fim de uma dada operação.

A variável `timer` guarda o timer relativo ao tempo total de execução, o seu valor `start` é capturado no início do `main`, e o seu valor `end` é capturado no valor da `main`

As variáveis `thread_timers` e `image_timers` guardam *vetores* de timers relativos ao tempo dos threads, e ao tempo de aplicação das transformações sobre as imagens. O thread timer captura o valor *start* no início da execução do thread e o seu valor de *end* é capturado quando o thread retorna, cada imagem processada pelo thread captura o seu tempo no `image_timers`, em modo complexo estes vetores são reescritos pelos threads das transformações subsequentes, mas não sem antes serem escritos para o `csv`.

As linhas `Thread x` representam o tempo de início e do fim do thread com `id x`
As linhas com o nome de um ficheiro representam o tempo de início e de fim da respetiva transformação nesse ficheiro.



```
1  ----- Resize operation -----
2  Thread no.,start time,end time
3  Thread 0,1642176028.588996400,1642176028.914625600
4  Thread 1,1642176028.589044800,1642176028.744107900
5  File name,start time,end time
6  00841.png,1642176028.588996400,1642176028.914625400
7  00844.png,1642176028.589044900,1642176028.744107800
8  ----- Thumbnail operation -----
9  Thread no.,start time,end time
10 Thread 0,1642176028.914787800,1642176029.113376000
11 Thread 1,1642176028.914793700,1642176029.105041300
12 File name,start time,end time
13 00841.png,1642176028.914787900,1642176029.113375900
14 00844.png,1642176028.914793700,1642176029.105041200
15 ----- Watermark operation -----
16 Thread no.,start time,end time
17 Thread 0,1642176029.113503800,1642176029.794289600
18 Thread 1,1642176029.113508300,1642176029.414274700
19 File name,start time,end time
20 00841.png,1642176029.113503900,1642176029.794289400
21 00844.png,1642176029.113508400,1642176029.414274500
22
23 Total,1642176028.580859300,1642176029.794359300
```

5 Resultados

Nesta secção os alunos deverão apresentar os resultados da execução dos dois programas desenvolvidos sobre os dois *datasets* fornecidos.

Nesta secção os resultados deverão corresponder ao processamento de todas as imagens de cada *dataset* (diretorias de destino completamente vazias).

5.1 Descrição do ambiente de execução

Foi utilizado o sigma

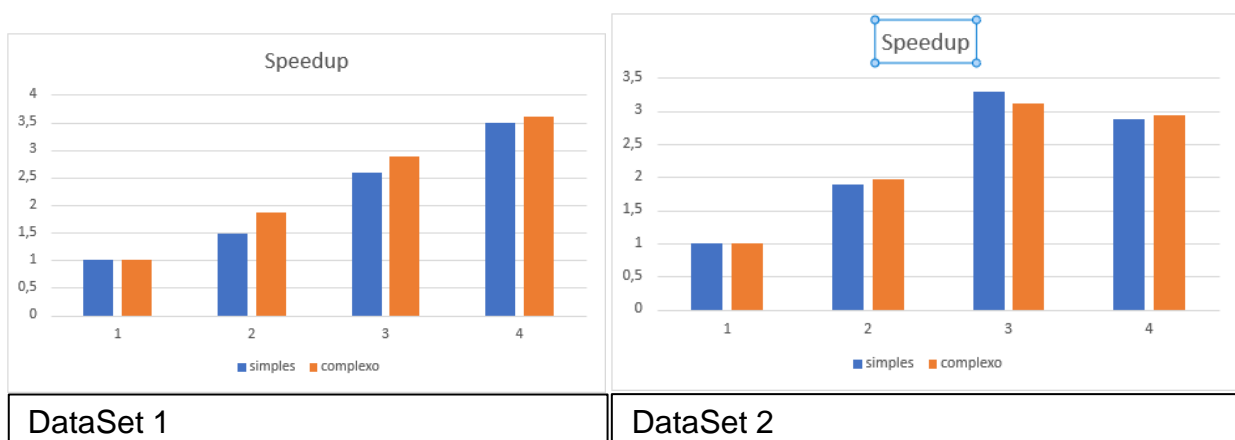
5.2 Resultados

Os alunos deverão preencher a seguinte tabela com os tempos de execução.

Table 3: Tempos de processamento (segundos)

		Número de <i>Threads</i>			
		1	2	4	8
Dataset1	ap-paralelo-simples	11,6101298	7,802659988	4,4921	3,33207011
	ap-paralelo-complexo	12,71671009	6,812620163	4,4214499	2,876140266
Dataset2	ap-paralelo-simples	50,77248001	26,80763006	15,43124986	17,70806003
	ap-paralelo-complexo	51,89135003	26,44145012	16,71674991	17,68817019

Os alunos deverão produzir dois gráficos (semelhantes aos seguintes) com os *speedups* calculados para os diversos programas, datasets e números de threads.



O speedup ideal para cada seria linear de acordo com o número de threads aumentadas. No entanto, o speedup real não é assim. Uma das razões para isto é estarmos a medir o tempo do programa inteiro, que inclui escrever no ficheiro. Além disso, poderemos também ter em atenção o uso do Sigma, que poderá ou não ser alto, e podemos também notar que temos as operações de leitura e escrita de ficheiros, que são demoradas.

No segundo dataset os resultados são ainda piores, e a razão para isso será o tamanho dos ficheiros e a sua distribuição: algumas imagens são muito maiores que outras. Isto faz com que haja threads que acabam o trabalho muito mais rapidamente que outras (segundo a divisão de trabalho aplicada - número de ficheiros, sem ter em conta o seu tamanho). Conclui-se que para este dataset, a divisão de trabalho não é boa.