# 2021_2 - FUNDAMENTOS DE SISTEMAS PARALELOS E DISTRIBUÍDOS - TM



# Primeiro exercício de programação: Mandelbrot paralelizado

**Prazo de entrega: confira o moodle, no link de entrega**

Atenção: os parâmetros para execução do programa foram alterados em relação à especificação inicial, confira no primeiro parágrafo da seção "Requisitos".

## Introdução

O conjunto de Mandelbrot é definido no espaço dos números complexos, por uma expressão que pode ou não convergir quando executada iterativamente. Os números para os quais a expressão converge constituem o conjunto. Na representação gráfica desse problema, os pontos que divergem são coloridos em função de quantas iterações foram necessárias para se confirmar que o ponto divergiu. Como uma imagem fractal, o conceito tem interesse para a matemática em a computação gráfica, entre outros.

Apesar da imagem básica ser bem conhecida, os detalhes exibidos dependem da resolução usada para a sua geração: ao focar em regiões pequenas da imagem mais comum, fora da área do conjunto propriamente dito (a parte preta, normalmente), pode ser preciso cada vez mais iterações para se determinar a solução para cada ponto. A imagem pode então ir se tornando cada vez mais complexa, o que levou os especialistas inclusive a definir nomes para cada tipo de estrutura principal que pode se encontrada nesses casos.

Uma animação de um processo de zoom numa região do espaço, pode mostrar a natureza fractal do problema, onde as estruturas vão mudando e às vezes se repetindo à medida que se escolhe cada vez uma área menor para ser exibida (fonte: Wikipedia).

## O que é fornecido

O arquivo fornecido para este exercício contém uma implementação da visualização do conjunto implementada em C, no Linux, com a biblioteca libgraphic, uma implementação simples da biblioteca gráfica do Turbo C, uma das mais antigas. Escolhida pela sua simplicidade, ela é bem limitada, tanto em termos do tamanho da janela gráfica quanto da paleta de cores disponíveis (apenas 4 bits). Seu objetivo é mais para mostrar que o código fornecido para o cálculo da imagem funciona corretamente - **a parte gráfica deverá ser removida da sua implementação, pois a libgraphic é incompatível com a biblioteca de Pthreads**. Detalhes sobre a biblioteca serão fornecidos para quem quiser executar o programa localmente; vou pedir para o CRC ver se é possível instalá-la nas máquinas do laboratório de graduação.

O programa, como apresentado, recebe como parâmetro um arquivo de entrada (e, opcionalmente, um número entre 0 e 9, que muda a forma como a paleta de cores é calculada - números menores têm efeito melhor, em geral). O arquivo de entrada contém uma ou mais linhas, cada uma da forma:

xgr_min ygr_min xgr_max ygr_max real_min img_min real_max img_max

Os quatro primeiros valores são inteiros e representam uma janela no espaço entre (0,0) e (640,480) (exclusive). Os quatro valores seguintes são valores de ponto flutuante (double) e representam uma janela no espaço dos números complexos, sendo que a parte real varia de real_min a real_max e a parte imaginária, de img_min a img_max. Por exemplo, para exibir o conjunto mais conhecido, os valores seriam:

0 0 640 480 -1.5 -1.0 0.5 1.0

Uma janela bem mais ampliada seria, por exemplo (a melhor paleta seria a 5 nesse caso):

0 0 640 480 0.27092045 0.00474965 0.27092055 0.00474975

A janela gráfica é gerenciada pela libgraphic e tem extensão limitada, o que limita os blocos que podem ser representados no arquivo de entrada para essa versão.

Alguns arquivos de teste são fornecidos junto com o programa. Os arquivos a-i contêm cada um o descritor de uma imagem de 640x480 pixels, relativas à sequência de zoom apresentada na aula do dia 22/11. O arquivo z tem um conjunto de 64 tarefas que montam a mesma figura do arquivo h, como 64 blocos separados.

## Objetivo

Seu objetivo será criar um programa em C/C++, com Pthreads, que calcula janelas dentro da imagem do fractal, de forma paralela (pense em um servidor de imagens, que recebe a descrição da imagem e calcula o bloco de pixels correspondente). Por limitações do ambiente, os blocos serão calculados, mas não exibidos. A estrutura final do programa deve conter pelo menos dois tipos de threads: uma thread que controla a entrada de dados e outro tipo de thread que calcula um bloco de imagem (pelo menos quatro devem ser criadas). Se realmente estivéssemos gerando imagens, provavelmente precisaríamos de um outro tipo de thread para receber as imagens geradas e dar um destino para elas, mas não vamos precisar disso nesse caso.

## Detalhamento

O princípio de funcionamento do programa será basicamente no modelo mestre-trabalhadores: a thread de entrada lê descrições de tarefas a serem executadas e as envia para os trabalhadores, que pegam uma tarefa por vez para ser executada. Quando as tarefas terminarem, o mestre deve enviar um registro especial que indique o fim das tarefas (em muitos sistemas esse registro é chamado de EOW, "end-of-work"). Não há retorno de informação dos trabalhadores para o mestre, vamos apenas coletar algumas estatísticas de execução, que poderão ser exibidas pelo programa principal, depois que as threads terminarem.

A comunicação entre a thread de entrada e as threads de geração das imagens deve ser feita por uma fila de elementos do tipo fractal_param_t que deve ter espaço para, no máximo, 4 vezes o número de threads de cálculo criadas. Você pode implementar uma fila circular simples como a mostrada em alguns dos slides, ou usar um container de C++, mas a sincronização deve ser obrigatoriamente controlada com mutex e variáveis de condição para os casos limite. O controle da fila mapeia exatamente para o problema do jantar dos selvagens: cada vez que a thread de entrada executa ela enche a fila e é suspensa; ela só deve ser acordada/liberada para colocar mais elementos na fila quando o número de tarefas restantes na fila (depois de uma thread retirar sua próxima tarefa) for igual ao número de threads trabalhadoras. O registro de EOW pode ser criado fazendo todos os valores do descritor de tarefa iguais a zero - basta testar se o tamanho da janela em pixels é zero.

A primeira thread deve ler o arquivo de entrada, transformar cada linha em um registro de descrição de tarefa (do tipo fractal_param_t) e colocá-lo na fila. Ao encontrar o fim de arquivo, deve colocar tantos registros de EOW quantas forem as threads trabalhadoras criadas (esse número pode ser alterado por um parâmetro). Cada thread trabalhadora deve tirar um registro descritor de tarefa da fila, executar a função de geração da imagem do fractal de mandelbrot (sem a parte da exibição na tela, apenas) e retornar em busca de novas tarefas.

Estatísticas

As seguintes estatísticas devem ser apresentadas ao final da execução, usando os formatos indicados a seguir. Os nomes de variáveis não precisam ser os mesmos.

- Total de tarefas, média e desvio padrão do número de tarefas executadas por thread:

- `printf("Tarefas: total = %d;  média por trabalhador = %f(%f)\n", total_tarefas, media_tarefas_pt, desvio_tarefas_pt);`

- Tempo médio (e desvio padrão) do tempo de execução de cada tarefa em milissegundos:
  - `printf("Tempo médio por tarefa: %.6f (%.6f) ms\n", t_medio, t_desvio);`

- Número de vezes em que os trabalhadores encontraram a fila vazia
  - `printf("Fila estava vazia: %d vezes\n", conta_fila_vazia);`

A princípio, essas três linhas são as únicas que devem ser exibidas pelo programa final. Caso você inclua mensagens de acompanhamento e depuração, certifique-se de que elas sejam comentadas ou bloqueadas com um flag de execução na versão submetida.

Cada aluno é responsável por gerar o código necessário para o cálculo dessas estatísticas e garantir que as operações sejam executadas com exclusão mútua, quando necessário. Para medidas de tempo, estão disponíveis as funções `gettimeofday` e `clock gettime`. Escolha a que considerar mais adequada.

## Sobre a execução do programa:

Na máquina virtual usada para desenvolver o programa original distribuído para este exercício, a versão sem a biblioteca gráfica, usando lendo o arquivo t fornecido junto com o programa, gasta em média 10 segundos para executar. É bastante provável quem em outras máquinas esse tempo seja muito pequeno coletar estatísticas válidas para o programa. Nesse caso, há três ações que podem ser adotadas para obter um arquivo de entrada mais adequado:

1. incluir tarefas que contenham preferencialmente regiões internas ao conjunto de Mandelbrot (áreas em preto); por exempo, a região `-0.5 -0.5 0.0 0.5`;
2. aumentar a área em pixels das imagens a serem geradas. Por exemplo, os blocos no arquivo t têm cada um 80x60 pixels; aumentar essa área algumas vezes pode ser suficiente - escolher o fator de aumento adequado para cada caso fica a cargo de cada aluno (como não vamos usar a biblioteca gráfica, isso não terá impacto visual);

3. aumentar o número de taefas em um arquivo de testes. Uma forma fácil de fazer isso pode ser simplesmente repetir tarefas já definidas no arquivo t, por exemplo.

O arquivo final de testes utilizado na correção do exercício será gerado usando as três operações. Você pode também incluir o arquivo utilizado para os testes do seu programa na entrega e ele será também considerado na avaliação.

## Requisitos (alterados em relação à especificação inicial):

O programa deve receber como **primeiro parâmetro** de execução o nome do arquivo de entrada contendo os blocos a serem calculados. **Além disso**, o programa deve aceitar um segundo parâmetro, **opcional**, que indicará o número de threads trabalhadoras que devem ser criadas. Certifique-se de que seu programa seja padronizado para se ajustar a esse valor, por exemplo, para definir o tamanho da fila de execução e as condições em que a fila será preenchida com base nesse número. Caso seu programa precise de um limite superior, considere que esse número nunca será maior que 20. Nenhum outro parâmetro deve ser necessário.

O código deve usar apenas C/C++ padrão, sem bibliotecas além das consideradas padrão. O paralelismo de threads deve ser implementado usando POSIX threads (Pthreads) apenas. Em particular, não será aceito o uso da biblioteca de threads do C++, nem é permitido o encapsulamento de Pthreads para simular aquela API, ou qualquer outra.

O material desenvolvido por você deve executar sem erros nas máquinas linux do laboratório de graduação. A correção será feita naquelas máquinas e programas que não compilarem, não seguirem as determinações quanto a nomes, parâmetros de entrada e formato da saída, ou apresentarem erros durante a execução, serão desconsiderados.

## O que deve ser entregue:

Você deve entregar um arquivo .zip contendo os seguintes elementos:

- código fonte do programa final produzido, devidamente comentado para destacar as decisões de implementação necessárias para a implementação das threads e da coleta de estatísticas;

- makefile com as seguintes regras: `clean` (remove todos os executáveis, .o e outros temporários); `build` (complila e gera o executável); e `run` (gera o executável, se necessário, e executa o programa, passando a variável de ambiente $ARGS para o executável);
- relatório em PDF.

Para a execução do `make run` com argumentos, basta considerar que ARGS será uma variável de ambiente. Assim, se o programa se chamar "prog", o comando no make run pode ser algo como `./prog ($ARGS)` e a execução do make pode ser algo como `make run ARGS="nome_do_arquivo"`.

O relatório não precisa ser longo, mas deve documentar as principais decisões de projeto consideradas - em particular, qual a implementação adotada para a fila de tarefas e como foi feita a coleta das estatísticas - e uma análise do tempo de execução. Essa documentação deve apresentar a informação em alto nível, como um complemento para a informação contida nos comentários do próprio programa. A análise de tempo deve comparar a execução com uma thread trabalhadora apenas e com threads em número igual ao número de cores disponíveis na sua máquina.

Preste atenção nos prazos: entregas com atraso não serão aceitas.

## Dúvidas?

Usem o fórum criado especialmente para esse exercício de programação para enviar suas dúvidas. Entretanto, **não é permitido publicar código no fórum**! Se você tem uma dúvida que envolve explicitamente um trecho de código, envie mensagem por e-mail diretamente para o professor.

◄ Recursos de apoio

Seguir para...

Material para o primeiro exercício de programação ►