Atividade 1 - Org. Arg. 1

Profa. Dra. Cíntia Borges Margi
(cintia@usp.br)

Guilherme S. Salustiano
(salustiano@usp.br)

Introdução

Nessa atividades realizaremos um brenchmark, analizando como a duração e IPC do problema mudam a partir do tamanho da entrade e numero de threads.

Requisitos

Essa atividade depende de funcionalidades únicas do Linux, recomendamos rodar linux nativamente. Caso não seja possível, é recomendado utilizar o WSL2 no Windows, ou via Máquina Virtual (com no mínimo 2 cores), em caso de duvidas contate o monitor.

Vamos precisar das seguintes dependências:

```
$ Ubuntu
$ sudo apt install build-essential # p/ gcc & make
$ sudo apt install linux-tools-generic # p/ perf
$ sudo apt install hwloc # p/ lstopo (Opcional)
```

Ou você também pode usar o docker com o script `run_by_docker.sh`.

Conhece quem tu(a CPU) és

Utilizaremos o `lscpu` para observar as especificações do CPU. No terminal rode:

```
$ lscpu
```

Observe no `CPU(s)` podemos observar o número de CPU da máquina, guarde esse número pois vamos precisar relacioná-lo a aplicação mais a frente. Também podemos observar outros termos já conhecidos, como a `Architecture`, a ordem dos bits e o tamanho dos caches.

Opcionalmente também podemos gerar uma imagem da arquitetura usando o `lstopo`.

```
$ lstopo architecture.png
```

Entrega

Gere um arquivo json `cpu_info.json` contendo as informações da sua CPU.

```
$ # a flag '-J' faz com que a saída seja em JSON
$ # o '>' redireciona a saída do programa para um arquivo
$ lscpu -J > cpu_info.json
```

Esse arquivo será submetido junto com o restante dos arquivos no final da tarefa.

A aplicação

Contexto

Você já ouviu falar do Conjunto de Mandelbrot? Seu descobridor foi Benoit Mandelbrot, que trabalhava na IBM durante a década de 1960 e foi um dos primeiros a usar computação gráfica para mostrar como a complexidade pode surgir a partir de regras simples. Benoit fez isso criando e visualizando imagens de geometria fractal.

Um desses fractais foi nomeado *Conjunto de Mandelbrot* pelo matemático Adrien Douady. O Conjunto de Mandelbrot pode ser informalmente definido como o conjunto dos números complexos c para os

quais a função $f_c(z) = z^2 + x$ não diverge quando é iterada começando em $z = 0$. Isto é, a sequência $f_c(0), f_c(f_c(0)), f_c(f_c(f_c(0))), \ldots$ é sempre limitada. Nas Figuras abaixo podemos ver algumas regiões do Conjunto de Mandelbrot.

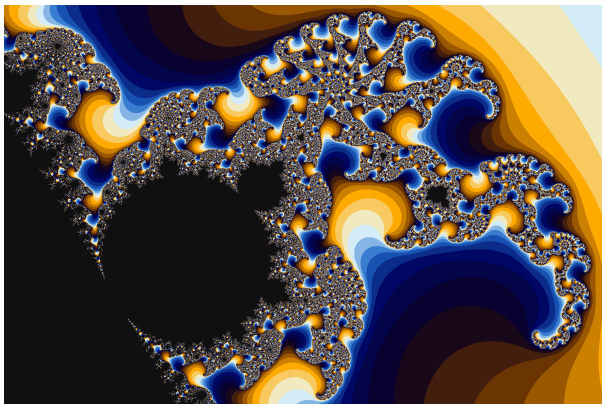

Figura 1: *Elephant Valley*

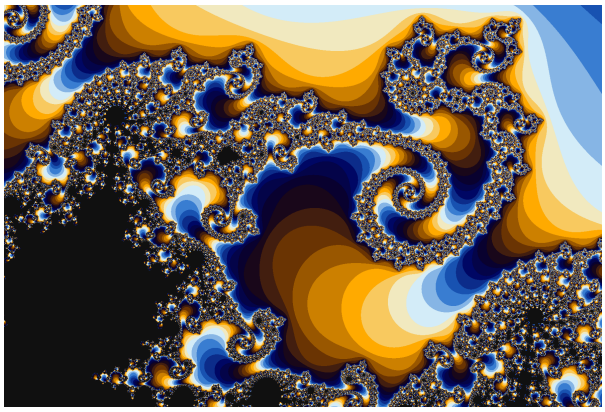

Figura 2: *Seahorse Valley*

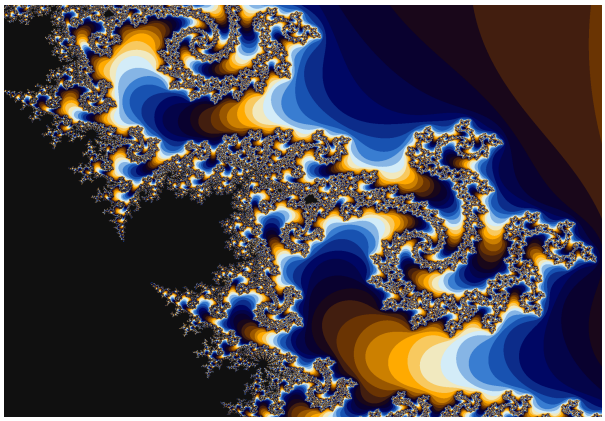

Figura 3: *Triple Spiral Valley*

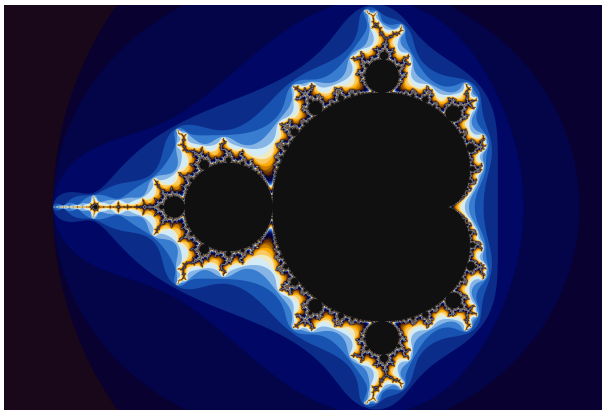

Figura 4: *Full Picture*

Executando o código

Baixe o código do EP e entre na pasta com a linha de comando, então basta executar os seguintes comandos

```
$ # Compila o código
$ make
$ # Roda o programa
$ OMP_NUM_THREADS=8 ./mandelbrot -2.5 1.5 -2.0 2.0 11500
```

O programa recebe 5 parâmetros, os primeiro quatro se referem a região que será calculada e o último diz respeito ao tamanho da imagem gerada. O número de threads usado é definido pela variável de ambiente `OMP_NUM_THREADS` passada no contexto do shell.

Ao final da execução, o programa gera o arquivo `mandelbrot.ppm` com o seu lindo conjunto de mandelbrot.

Entrega

Para a entrega dessa seção vamos realizar a medição performance utilizando o comando `perf`. Comece executando o seguinte comando:

```
$ OMP_NUM_THREADS=8 perf stat -r 10 -e cycles,instructions,duration_time,power/energy-cores/ ./mandelbrot -2.5 1.5 -2.0 2.0 11500 2048
```

Estamos usando o `perf-stat` que junta estatísticas sobre a execução do programa. O parâmetro `-r` significa o número de repetições da execução, utilizado para calcular o intervalo de confiança. No

parâmetro `-e` passamos a lista de eventos que queremos contar, podemos observar a lista inteira com o comando `perf list`. Por fim temos efetivamente o programa a ser executado.

Agora vamos rodar novamente o programa para uma entrada maior e salvar seu resultado para submissão.

```
$ OMP_NUM_THREADS=8 perf stat -r 10 -e cycles,instructions,duration_time,power/energy-cores/ ./mandelbrot -2.5 1.5 -2.0 2.0 11500 2> perf.txt
$ cat perf.txt
```

Brenchmark

Para cada uma das três versões do programa, vocês deverão realizar medições do tempo de execução para diferentes tamanhos de entrada. Nas versões paralelizadas vocês deverão também medir, para cada tamanho de entrada, o tempo de execução para diferentes números de *threads*.

Vocês devem fazer um número de medições e analisar a variação dos valores obtidos. Sugerimos 10 medições para cada experimento, e também que vocês usem a média e o intervalo de confiança das 10 medições nos seus gráficos. Caso observem variabilidade muito grande nas medições, resultando num intervalo de confiança muito grande, vocês podem realizar mais medições, sempre apresentando a média e o intervalo de confiança. **Não é recomendado** fazer menos de 10 medições.

A Tabela 1 lista os experimentos que devem ser feitos: os valores para o número de *threads* e de execuções, e os tamanhos de entrada. Cada experimento deverá ser repetido nas quatro regiões anteriormente apresentadas. As coordenadas para cada região podem ser obtidas executando no diretório `src`:

```
$ make mandelbrot_seq
gcc -o mandelbrot_seq -std=c11 mandelbrot_seq.c
$ ./mandelbrot_seq
usage: ./mandelbrot_seq c_x_min c_x_max c_y_min c_y_max image_size
examples with image_size = 11500:
  Full Picture:     ./mandelbrot_seq -2.5 1.5 -2.0 2.0 11500
  Seahorse Valley:  ./mandelbrot_seq -0.8 -0.7 0.05 0.15 11500
  Elephant Valley:  ./mandelbrot_seq 0.175 0.375 -0.1 0.1 11500
  Triple Spiral Valley: ./mandelbrot_seq -0.188 -0.012 0.554 0.754 11500
```

O número de iterações para o critério de convergência foi escolhido
de forma a produzir uma imagem interessante em diferentes níveis de magnificação, mas ter um tempo de execução razoável para tamanhos grandes de entrada.

Regiões	• *Triple Spiral* • *Elephant* • *Seahorse* • *Full*
Núm. de Threads	$2^0 \ldots 2^6$
Tamanho da Entrada	$2^4 \ldots 2^{13}$
Núm de Execuções	10

Tabela 1: Experimentos

Entrega

De forma a facilitar a execução dos parâmetros fornecemos o script `run_measure.py` que realiza a execução do programa (utilizando o `subprocess.run`), varia os parâmetros conforme o especificado e salva seus resultados na pasta `results` em formato JSON.

Para executar entre no diretório `src/` e execute o comando abaixo:

```
python run_measure.py
```

Atenção no teste de referencia executando em um processador i5 11° geração, o brenchmark rodou em 5 horas. Você pode interromper o script, se necessário, e ele voltará a executar do experimento que parou. Evite rodar coisas pesadas junto do experimento, como jogos, e em caso de notebook prefira rodar sempre enquanto estiver na tomada.

Após o término da execução você deverá submeter a pasta `results/` com os arquivos `json` criados.

Análise dos resultados

O script `run_measure.py` também gera gráficos na pasta `graphs/` (utilizando pandas e mathplotlib). Você pode alterá-los ou incrementá-los. se quiser.

Entrega

Vocês deverão analisar os resultados obtidos e tentar responder a algumas perguntas:

- Como e por que as três versões do programa se comportam com a variação:
  - Do tamanho da entrada?
  - Das regiões do Conjunto de Mandelbrot?
  - Do número de *threads*?
- Qual é o número de *threads* ideal teorico? O que podemos observar nesse ponto?
- Porque um IPC maior não necessariamente corresponde a uma menor duração?

Vocês conseguem pensar em mais perguntas interessantes?

Crie um documento `relatorio.pdf` com suas respostas e considerações, ele deverá ser submetido ao final com os demais arquivos.

Entrega final

Ao final gere um zip `atv1.zip` com os arquivos solicitados ao longo da atividade e submeta no tidia.

```
atv1.zip
├── cpu_info.json
├── perf.txt
├── results/*.json
└── relatorio.pdf
```

Bibliografia