ELC139 - Programação Paralela

Paralelização do código Yet More Primes

Gabriel Di Domenico, Tiago Chagas

Hardware/Software - Gabriel

- O sistema operacional utilizado foi Linux Ubuntu 20.04.2
- Hardware utilizado foi um AMD Ryzen 5 5600X, com 4 núcleos e 8 threads

Hardware/Software - Tiago

- O sistema operacional utilizado foi Linux Ubuntu 20.04.2
- Hardware utilizado foi um Intel Core i7-8750H 8ª geração, com 6 núcleos e 12 threads

MPI - Funcionamento

- O MPI cria processos separados, isto é, toda a memória alocada de um processo é totalmente independente dos demais;
- É possível, através das funções do MPI, obter-se o ID do processo atual e o número total de processos.

O que paralelizar?

- O programa possui 2 laços de repetição:
  - O que realiza os cálculos e os armazena em um vetor;
  - O que pega os resultados deste vetor e gera os arquivos .bmp
- A paralelização será focada nestes laços.

Estratégia - Laços for

- A partir dos laços citados anteriormente, foi feita uma execução baseada em um offset, que divide o cálculo entre os processos:

```
for (int frame = procid*partition; frame < procid*partition+partition; frame++)
```

- Os frames começam a partir do “procid” (process ID) multiplicado pelo número total de frames dividido pelo número total de processos;

```
int partition = frames/numprocs;
```

- Esta separação divide igualmente o número de frames que serão calculados por cada processo.

Estratégia - “compartilhamento” do delta

- Ao realizar vários testes no programa, foi observado que a variável “delta” era o acumulador de resultados que garantia que o output tivesse a ordem correta de frames;
- Para essa variável mantivesse o resultado esperado entre processos de forma independente, foi criado um loop for auxiliar que incrementa o delta baseado no número do ID do processo, que se encontra antes do laço que realiza os cálculos:

```
for(int frame = 0; frame < procid*partition; frame++){
  delta *= 0.98;

}
```

- Por exemplo, no processo de ID 1, o valor de delta começará do jeito que o processo de ID 0 terminou.

Dificuldades

- Os processos são realizados em ordem aleatória, dependendo da disponibilidade das threads, o que em algumas soluções o resultado se torna não determinístico;
- O MPI não foi feito para realizar união de resultados por meio de acumuladores ou afins, pois, como falado anteriormente, a memória de cada processo é independente;
- Para realizar uma paralelização funcional, ou seja, para que cada processo não necessite esperar os resultados do anterior, é necessário alterar os valores por meio do número do ID do processo, juntamente com a quantidade total de processos.

Testes

- Cada teste foi realizado com 2,4 e 8 processos
- Os testes utilizados para o programa foram:
  - Teste 1: 256 32
  - Teste 2: 1024 32
  - Teste 3: 1024 64
- A medidas das saídas são em segundos.

Resultados - Gabriel

- Teste 1:
  - 2 proc: 1.0631 s
    - Speedup: 1.90819302041
    - Eficiência: 0.9540965102
  - 4 proc: 0.5514 s
    - Speedup: 3.67899891186
    - Eficiência: 0.91974972796
  - 8 proc: 0.3277 s
    - Speedup: 6.1904180653
    - Eficiência: 0.77380225816

Resultados - Gabriel

- Teste 2:
  - 2 proc: 16.9022 s
    - Speedup: 1.90393558235
    - Eficiência: 0.95196779117
  - 4 proc: 8.6272 s
    - Speedup: 3.7301441951
    - Eficiência: 0.93253604877
  - 8 proc: 4.6397 s
    - Speedup: 6.93594413432
    - Eficiência: 0.86699301679

Resultados - Gabriel

- Teste 3:
  - 2 proc: 32.1914 s
    - Speedup: 1.90780767534
    - Eficiência: 0.95390383767
  - 4 proc: 17.0229 s
    - Speedup: 3.6077871573
    - Eficiência: 0.90194678932
  - 8 proc: 8.9336 s
    - Speedup: 6.87460822065
    - Eficiência: 0.85932602758

Resultados - Tiago - Teste 1

- Teste 1:
  - 2 proc: 2.0751 s
    - Speedup: 0.985398294
    - Eficiência: 0.492699147
  - 4 proc: 1.0791 s
    - Speedup: 1.894912427
    - Eficiência: 0.473728107
  - 8 proc: *

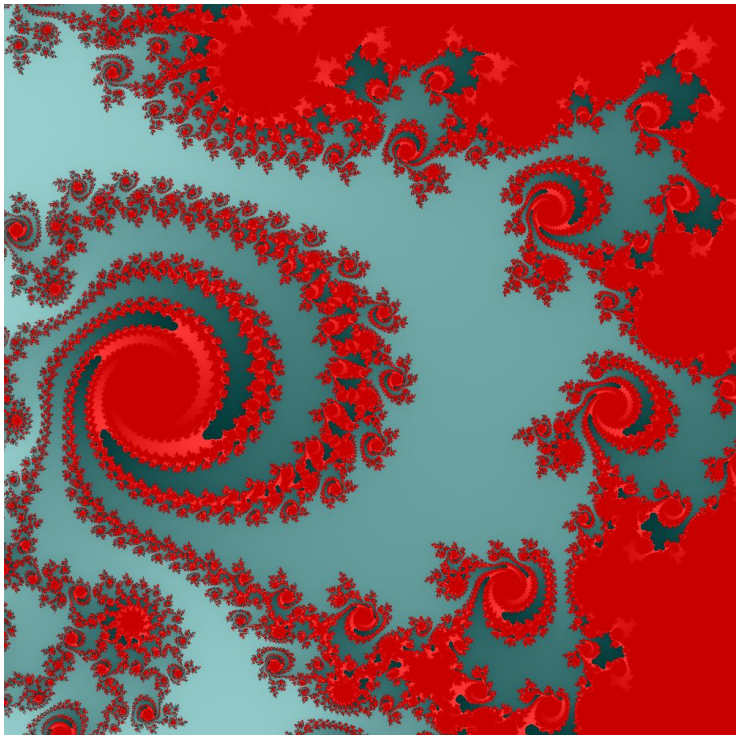
Resultados - Tiago - Teste 2

- Teste 2:
  - 2 proc: 27.6238 s
    - Speedup: 1.274495182
    - Eficiência: 0.637247591
  - 4 proc: 19.1837 s
    - Speedup: 1.835224696
    - Eficiência: 0.458806174
  - 8 proc: *

Resultados - Tiago - Teste 3

- Teste 3:
  - 2 proc: 54.6835 s
    - Speedup: 1.411224592
    - Eficiência: 0.705612296
  - 4 proc: 36.2556 s
    - Speedup: 2.128518077
    - Eficiência: 0.532129519
  - 8 proc: *

Animação personalizada



Bibliografia

- MPI v4.1.1 documentation:

  https://www.open-mpi.org/doc/current/