Trabalho de Programação Paralela Mandelbrot Set com OpenMP

Daniela Kuinchtner – 152064@upf.br

Implementação

Com base em uma análise prévia feita, foi descoberto que o desenvolvimento do algoritmo de conjuntos de números complexos está baseado, praticamente, no laço *while*, que se encontra dentro de dois laços *for* e requer um custo alto de tempo de processamento, mas como nesse laço há dependência de valores de iterações anteriores, o primeiro *for* foi paralelizado. O primeiro laço *for* percorre as linhas da matriz, o segundo percorre as colunas e o laço *while* realiza duas condições de parada, uma que o número complexo tem que ser menor que 2 e outra que atinja o número máximo de iterações, então realiza os cálculos para formar o desenho da saída, caso as condições de paradas forem satisfeitas, o caractere "#" é printado no arquivo de saída, senão ".".

Dificuldades encontradas

Entender a complexidade do algoritmo e especificar quais variáveis seriam privadas ou compartilhadas.

Recursos utilizados

Para paralelização foi utilizada a biblioteca *omp.h* que possibilita o uso de diversos recursos para a linguagem C/C++. Dentre esses recursos estão os utilizados:

- *#pragma omp parallel for*: indica que o bloco de código é para ser executado em paralelo, onde as iterações serão divididas entre a quantidade de *threads* definidas pelo programador. Esse comando foi utilizado para paralelizar o laço *for* das linhas da matriz.
- *num_threads(thread)*: recebe por parâmetro o número de *threads* que serão utilizadas nas regiões definidas como paralelas, sendo que, só é possível utilizar 2, 4, 8 e 16 *threads*
- *private(r,c)*: declara as variáveis que terão cópias independentes em cada uma das *threads*. No laço paralelizado foram declaradas as variáveis r e c como privadas.
- *schedule(runtime):* a decisão do *scheduling* - quebra do processamento para cada *thread* em grãos menores, deixando assim, o processamento mais balanceado entre as *threads* e com menor tempo ocioso entre as mesmas - é feita durante a execução do programa pela variável de ambiente OMP_SCHEDULE (o valor padrão para essa variável é "*static*, 0").

Resultados

Foi utilizado um computador com processador Intel® Core™ i7-2600 CPU @ 3.40GHz × 8, 4 núcleos físicos e 4 lógicos, 8 GB de memória RAM e o sistema operacional Ubuntu 16.04 LTS, 64-bit. O tempo resultado foi obtido após a realização de 5 execuções paralelas com a entrada de 1024 (linhas), 768 (colunas) e 18000 (iterações: n) para cada quantidade de *threads*: 2, 4, 8 e 16 (Anexo I). A Tabela I mostra a média do tempo dessas execuções, bem como, *speedup*, eficiência e custo.

Threads	Tempo (s)	Speedup	Eficiência	Custo (s)
1	582	1	100%	582
2	287,4	2,025	101,25%	574,8
4	158,3	3,676	91,9%	633,2
8	103	5,650	70,625%	824
16	104	5,596	69,95%	832

Tabela I – Resultados

Análise

Analisando a tabela acima, nota-se que houve um grande ganho de tempo durante a execução do código paralelizado, sendo seu ponto de saturação atingido com 8 *threads*. O motivo para isso é que ele pode utilizar ao máximo seus recursos, ou seja, todas as 8 *threads* (físicas e lógicas) disponíveis no computador estão realizando suas próprias tarefas. Com 16 *threads* o tempo é um pouco maior, pois as mesmas perdem tempo com comunicação.

Considerações finais

Para o desenvolvimento desse código o OpenMP demonstrou um resultado satisfatório, nota-se logo de início que paralelizar esse tipo de código consegue-se resultados muito melhores que com a execução do código em sequencial. Consequentemente, numa máquina de potencial maior, poderia obter-se um tempo de execução menor ainda.

Threads	Execução 1	Execução 2	Execução 3	Execução 4	Execução 5
2	4m47.448s	4m47.331s	4m47.457s	4m47.258s	4m49.966s
4	2m.40.084s	2m32.990s	2m32.161s	2m33.245s	2m32.251s
8	1m43.911s	1m43.856s	1m43.826s	1m43.844s	1m43.898s
16	1m44.381s	1m44.290s	1m44.195s	1m44.272s	1m44.661s

Anexo I – Tabela com tempos das 5 execuções paralelas