Thiago Amendola RA:148062

MC751 - Sistemas Distribuídos

Teste 1

Neste teste foi avaliada a performance de três aplicações de mesma finalidade, implementadas com execução sequencial, multithread e multi-processo.

Proposta

Os programas devem receber um inteiro N, alocar N * 2^{20} números inteiros de 64 bits e gerar valores aleatórios de 0 a 1000 para cada um deles. O programa então deve calcular e imprimir a média dos valores desta área alocada. Para as versões multithread e multi-processo, o programa deve receber um inteiro adicional K e dividir os trabalhos de escrita de números aleatórios e cálculo de média entre K threads/processos.

A versão multithread foi feita com o uso de Pthreads, enquanto a versão multi-processo utiliza POSIX Shared Memory. Nestas duas versões, foram necessários cuidados para que múltiplos processos/threads não façam acesso de escrita a mesma área de memória.

Como executar

Entre na pasta /teste1 e execute o seguinte comando:

```
make run
```

Resultados

Os resultados foram gerados para N = 64, assumindo valores variados para K:

K	seq	thread	process
2	1.12s	19.10s	0.57s
4	1.09s	12.51s	0.49s
8	1.09s	12.22s	0.50s

Discussão

Os resultados obtidos para execução sequencial e multi-processo estão dentro do esperado. A versão multi-processo mostra-se cerca de 2 vezes mais rápida que a versão sequencial, devido ao seu fator de paralelismo. No entanto, nota-se que, com o acréscimo no número de processos paralelos, os ganhos tornam-se pequenos ou nulos devido ao overhead da separação do programa em múltiplos processos.

A versão multithread apresentou um tempo mais alto do que o esperado, sendo cerca de 15 vezes mais lento que a aplicação sequencial. Após uma análise refinada da execução desta versão do programa, foi possível notar que a maior parte do tempo de execução da versão multithread era gasto na atribuição e escrita dos valores aleatórios à memória alocada. Como as outras versões do programa são capazes de realizar esta etapa do algoritmo em tempos suficientemente rápidos sem maiores alterações no código, é possível concluir que a biblioteca utilizada para gerenciamento de threads bloqueie automaticamente o espaço alocado restringindo o acesso simultâneo de múltiplas threads ao vetor, mesmo que em partes diferentes do mesmo. Este evento explicaria a demora inesperada neste trecho do algoritmo. Vale ressaltar que etapas que fazem a leitura do vetor não sofrem com tempos altos de execução, reforçando a hipótese da execução de exclusão mútua internamente a biblioteca de threads apenas em casos de acesso a escrita.