Sistemas Operacionais II

Trabalho 01 - Multiplicação de Matrizes usando Processos e Threads

INF01151 – Sistemas Operacionais II N – Turma A – Prof. Alberto Egon Schaeffer Filho

Luiz Gustavo Frozi de Castro e Souza - Cartão 96957
Mário César Gasparoni Jr. - Cartão 151480

Outubro de 2013

Descrição do Ambiente de Teste

Sistema Operacional: Linux

- Distribuição: Debian
- Versão (kernel/distribuição): Debian 3.2.41-2+deb7u2

Configuração do computador:

- Processador: Intel Core I3
- Memória: 6GB
- Núcleos: 2
- Threads por núcleo: 2

Compilador: GNU C Compiler (gcc), versão gcc version 4.7.2 (Debian 4.7.2-5)

Descrição dos Casos de Teste Elaborados

Os casos de teste elaborados foram gerados a partir matrizes randomicas, aumentando-se as dimensões de cada matriz em cada caso. O valor máximo de matrizes para os testes foi fixado em 150x100 (ou 100x150) devido ao fato das limitações de memória/pilha/heap das máquinas aos quais foram executados. A dimensão máxima para cada matriz também foi fixada no programa, através da constante *MAX_DIMENSION*, sendo assim a dimensão máxima permitida para a matriz é de *MAX_DIMENSION X MAX_DIMENSION*. Além da limitação das dimensões, há também a limitação do número máximo de threads ou processos filhos a serem criados pelo programa. Nos testes, o valor máximo é 30, que corresponde também ao valor máximo fixado pelas constantes *MAX_PROCESSOS* e *MAX_THREADS* (a primeira correspondendo à versão do programa com processos filhos e a segunda, ao programa com threads)

Questionamentos

Execute a multiplicação de matrizes de forma seqüencial (n = 1), 10 (dez) vezes, e compare o tempo de execução com as versões que fazem o cálculo de forma concorrente. Há ganho de desempenho? JUSTIFIQUE, e apresente os dados coletados e uma análise dos resultados.

Sim, há ganho de desempenho. Isso ocorre devido ao fato de estarmos realizando concorrentemente (quando n>1) as tarefas que antes eram executadas serialmente (com n=1). A

concorrência permite diversas instâncias dos processos sendo executadas intercaladamente, disputando cada uma diferntes fatias de tempo de execução nos processadores da máquina em que executam. É importante destacar que multiplicação dematrizes é fortemente paralelizável, ou seja, podemos ter uma tarefa sendo executada a cada linha da matriz, processando o resultado independentemente de qualquer outra thread/processo que também esteja multiplicando uma outra linha, sem a necessidade de mecanismos de sincronização, como mutex ou semáforos, fato que torna essa operação ainda mais eficiente quando programamos concorrentemente. A figura abaixo ilustra o tempo de execução das duas versões dos programas de acordo com a variação do número de processos/threads filhas.

Medição a partir do número de processos/threads filhos

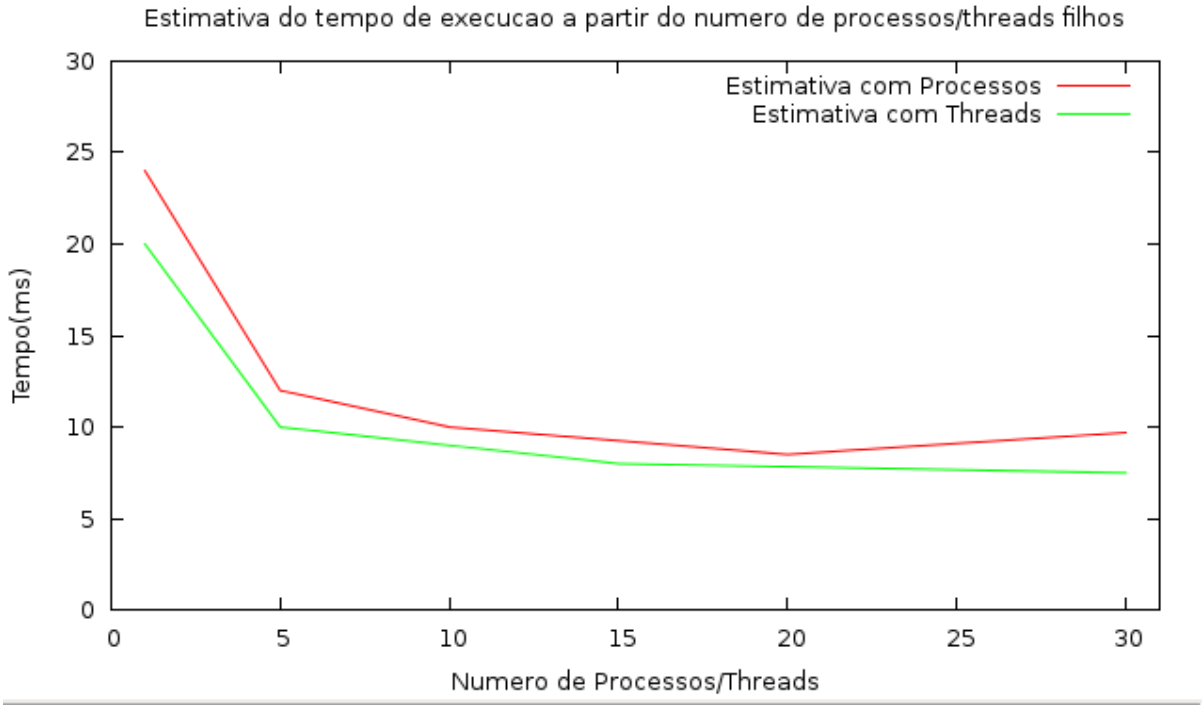

Figura 1: Estimativas de tempos para diferentes número de threads/processos filhos

Sem considerar ainda as diferenças entre a versão de threads e processos, podemos observar na figura 1 que o aumento do número (N) de processos/threads filhos reduz o tempo de execução total da multiplicação. Entretanto para valores de N altos, observamos que o ganho em desempenho diminui. Isso ocorre devido ao fato de que quando N é grande, aumentamos o grau de multiprogramação do sistema, de modo que o existem mais processos disputando pelas CPUs e , consequentemente , temos mais trocas de contexto entre processos/threads sendo realizadas, sendo esse o principal fator para a perda de desempenho quando N é muito alto.

Medição a partir do tamanho da matriz (quantidade de células)
(número de processos/threads filhos fixado em 3)

Há diferenças de tempo entre a versão com processo e a versão com threads para n = 2, 4 e 8? Se sim, em que casos? JUSTIFIQUE, e apresente os dados coletados e uma análise dos resultados.

Considerando agora as diferentes versões, uma com threads e outra com processos filhos, observamos na figura 1 que ocorrem sim diferenças nos tempos de execução para os programas executados, de modo que a versão com thread é a que possui um melhor desempenho durante a sua execução para os diferentes valores de N. Isso ocorre devido ao fato de thread, quando comparado com um processo filho, possuir uma estrutura mais leve, já que possui espaço de variáveis globais único e compartilhado entre todas as threads filhas do programa principal, além de outras estruturas de dados com tamanho reduzido em relação a um processo filho (como por exemplo Thread Control Block ,etc). Com os processos, entretanto, sabemos que cada processo filho é uma cópia exata do pai, ou seja, inclui-se em cada filho todo o espaço de enderaçamento do pai que, diferentemente das threads, acaba sendo replicado entre todos os filhos. Dessa forma, o comportamento esperado na teoria é o que ocorre na prática, como mostra o gráfico da figura, onde as threads, de maneira geral, obtém um desempenho superior ao da execução com processos.

Explique como a comunicação entre as unidades de execução (na implementação utilizando processos) foi implementada.

Para realizar o compartilhamento de memória entre todos os processos filhos, foi criado um segmento de memória compartilhado, através da função *shmget()*, que logo em seguida é vinculado/anexado ao processo através da função *shmat()*, sendo ambas chamadas executadas antes da criação dos processos filhos. Como cada filho é a cópia exata do pai, não é necessário vincular o segmento compartilhado novamente a cada processo filho criado, mas sim apenas usar diretamente o segmento já anexado pelo processo pai. Devido ao fato de não haver nenhuma seção crítica durante a multiplicação de matrizes (as matrizes de origem são apenas lidas e a de destino é acessada em áreas diferentes por cada thread/processo) , não foi necessário nenhum mecanismo para realizar a exclusão mútua entre cada thread/processo filho.