SUMÁRIO

INTRODUÇÃO

Este trabalho é constituído de 3 programas que foram desenvolvidos de formas diferentes com o intuito de analisar os desempenhos e explorar a Programação Concorrente através das Threads.

A primeira parte do trabalho é composta de 1 programa que preenche uma Matriz de dimensões escolhidas pelo usuário, após o preenchimento o mesmo calcula o Produto Interno de cada linha e ao final soma todos os valores constituindo o Produto Interno da Matriz. Em seguida o programa informa ao usuário o valor máximo e o valor mínimo da Matriz assim como as suas posições. Todo esse processo é feito de forma sequencial, assim o programa é constituído de um único processo.

A segunda parte do trabalho contém mais 1 programa, este tem a mesma função do primeiro, porém a forma como ele executa a sua função será diferente. Esse programa é composto de Threads, e o seu trabalho é dividido pelas mesmas. Como no primeiro o usuário escolherá o numero de linhas e de colunas que a Matriz terá e em seguida o programa apresentará a Matriz preenchida. Diferente do programa anterior no segundo programa serão usadas Threads para calcular o Produto Interno de cada linha da Matriz, assim **n** Threads calcularão o Produto Interno de **n** linhas. Desta forma buscamos um melhor desempenho do segundo programa em relação ao tempo do primeiro.

Na terceira parte do trabalho teremos finalmente a última versão do programa. Nessa versão também buscamos um melhor desempenho em relação às versões anteriores, e para isso diminuiremos o numero de Threads a serem criadas. No calculo do Produto Interno serão criadas apenas 4 Threads e o trabalho será dividido entre as mesma. Assim caso seja criada um Matriz de 45 linhas as 3 primeiras Threads calcularão o Produto Interno de 11 linhas e a última Thread calculará das 12 linhas restantes.

Através deste trabalhos procuramos não só apresentar o uso da Computação Concorrente através de Threads mas também as vantagem da utilização da mesma, bem como a maneira mais eficiente para sua utilização.

DESENVOLVIMENTO

Foram criadas diversas versões para a solução de um mesmo problema proposto para este trabalho. Como já foi dito a principal diferença nas várias versões é a forma com o trabalho é dividido nos diversos fluxos.

Na primeira versão existe apenas um fluxo, e o programa é totalmente sequencial. Pudemos observar que à medida que o número de linhas da matriz vai aumentando o tempo de execução do programa acompanha esse crescimento.

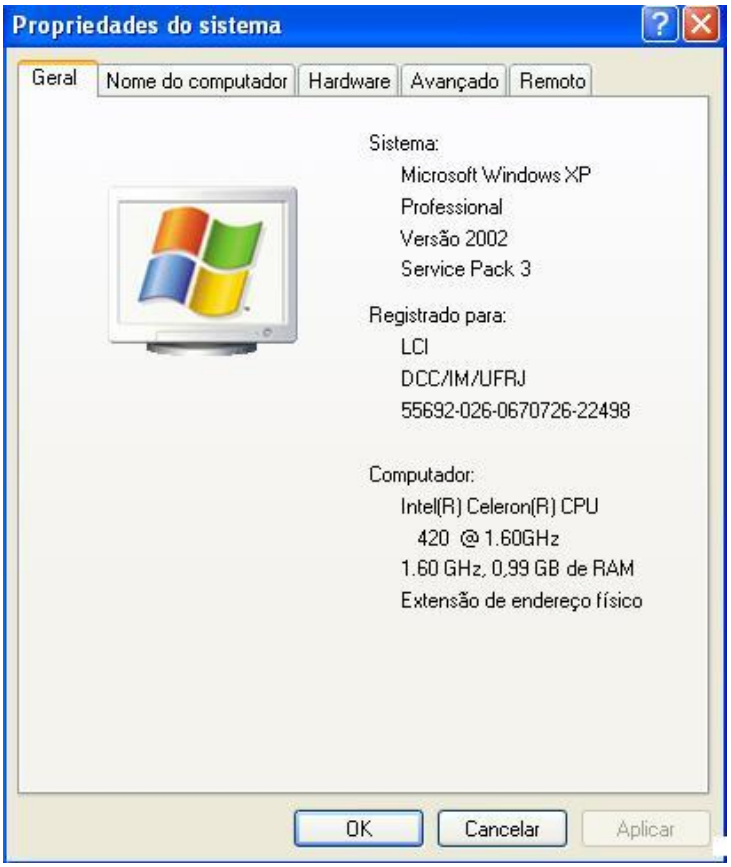
Na segunda existem o número de fluxos é proporcional ao número de linha da matriz. Ao executar essa versão também observamos um crescimento proporcional ao número de linhas, porém também observamos que o tempo de execução da segunda versão é maior do que a primeira.

A terceira versão o número de fluxos sempre será quatro, usamos quatro pois esse é número usual de threads que rodam simultaneamente nos computadores atuais, e quanto mais próximo do número de núcleos que há no computador melhor é o desempenho. Também verificamos um crescimento no tempo de execução acompanhando o crescimento da matriz, mas desta vez o tempo de execução ficar entre os tempos da primeira e da segunda versão.

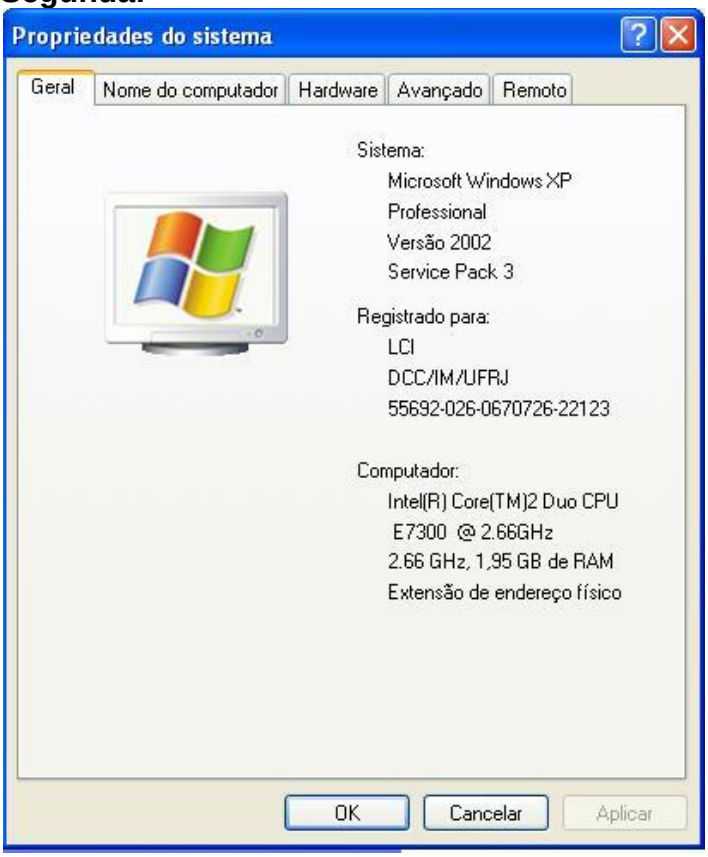
Na tabela a seguir podemos observar o tempo de execução de cada um dos programas com diferentes dimensões da Matriz. Essa tabela contém o tempo das três versões do trabalho anterior (**1.1**, **1.2**, **1.3**) feito através de sub-processos, e também do atual trabalho onde implementamos as Threads (**2.1**, **2.2**, **2.3**).

Os programas foram executados em duas maquinas com configurações diferentes. A primeira máquina possui o seguinte processador: **Intel Celeron.** A segunda máquina possui um processador **Core 2 Duo.**

Primeira:



Segunda:



A tabela a seguir mostra o desempenho a cada acréscimo de linha e coluna, em milissegundos.

Celeron

Trabalho	10x10	20x20	50x50	100x100	380x380	500x500	1000x1000
1.1	0	0,001	0,011	0,023	0,294	0,49	1,907
1.2	0,012	0,27	0,069	0,121	0,721	0,983	3,728
1.3	0,008	0,013	0,043	0,081	0,779	1,564	12,483
2.1	0	0	0,011	0,025	0,286	0,501	2,308
2.2	0,001	0,003	0,012	0,039	0,334	*Error	*Error
2.3	0	0,001	0,012	0,025	0,329	0,535	2,225

***Error** = o sistema só administra 382 threads por processo

Core 2 Duo

Trabalho	10x10	20x20	50x50	100x100	380x380	500x500	1000x1000
1.1	0	0,001	0,011	0,016	0,115	0,201	0,768
1.2	0,009	0,015	0,041	0,84	0,480	0,604	1,709
1.3	0,006	0,009	0,024	0,047	0,277	0,445	3,104
2.1	0	0,001	0,011	0,016	0,124	0,212	0,790
2.2	0	0,001	0,012	0,018	0,138	*Error	*Error
2.3	0	0,001	0,010	0,016	0,135	0,204	0,854

***Error** = o sistema só administra 382 threads por processo

Na segunda versão deste trabalho tivemos dois erros na execução do programa como pode ser observado na tabela anterior. Este erro aconteceu ao tentarmos criar uma Matriz com mais de 382 linhas acontece um erro na criação das Threads.

Eis um exemplo de uma execução com a criação de mais de 382 threads:

“Digite o número de matriz.colunas: Digite o número de linhas: In main: criando thread numero 0
In main: criando thread numero 1
In main: criando thread numero 2
In main: criando thread numero 3
In main: criando thread numero 4
...
In main: criando thread numero 380
In main: criando thread numero 381
In main: criando thread numero 382
ERROR code is 11”

CONSIDERAÇÕES IMPORTANTES

***** Pudemos verificar que o resultado nos tempos de execução do segundo teste, feitos com o processador **Dual Core,** foram muito menores comparados aos valores do primeiro teste. Isso acontece devido à configuração dos processadores.

***** O tempo de execução dos programas onde foram implementadas Threads é menor do que o tempo dos programas onde foram usados sub-processos.

* No primeiro teste os valores da terceira versão do segundo trabalho ficaram entre a primeira e a segundo (2.1**<**2.3**<**2.2). Já no segundo teste os valores variam, em alguns momentos essa ordem se mantém, em outros momento (como na matriz **50X50**) o tempo de execução da terceira versão é menor.

CONCLUSÃO

Os resultados do primeiro teste em que usamos uma máquina **Celereron(R)** foram piores em termos de desempenho do que o caso em que usamos uma máquina **Dual Core,** isso aconteceu devido ao número de núcleos que o segundo processador possui. Com um numero maior de núcleos o processamento junto torna-se mais rápido

A principal causa dos processos com thread obterem o tempo melhor do que os que utilizam sub-processos é que quando utilizamos sub-processos a alocação dos valores resultados do cálculo do produto interno foi feita por meio de um arquivo, e novamente ocorreu a leitura do arquivo para recuperar esses valores e mostrar o resultado.

Programas concorrentes que utilizam múltiplos threads são mais rápidos do que implementados como múltiplos (sub)processos. Como os threads compartilham os recursos do processo, as operações de criação, troca de contexto e eliminação dos threads geram um ganho de desempenho. Como todos os threads em um processo compartilham o mesmo espaço de endereçamento, a comunicação entre os threads pode ser feita utilizando o compartilhamento de memória de forma rápida e eficiente.