UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
DEPARTAMENTO DE INFORMÁTICA APLICADA

INF01151 – SISTEMAS OPERACIONAIS II N

TRABALHO 1: MULTIPLICAÇÃO DE MATRIZES
PROCESSOS E THREADS

INTEGRANTES:
BRUNO GABRIEL FERNANDES
MARCELO DE OLIVEIRA ROSA PRATES
SUYÁ PEREIRA CASTILHOS

1. Ambiente de teste

Como ambiente de testes foi utilizado uma máquina com processador Pentium(R) Dual-Core CPU T4500 @ 2.30GHz com 2GB de memória RAM executando o sistema operacional Linux Mint 14 Nadia.

Como compilador, foi utilizado o GCC na versão 4.7.2.

2. Versões: processos e threads

Foram desenvolvidas duas versões de um módulo simples de multiplicação de matrizes. Ambas implementam programação concorrente, distinguindo-se uma da outra pelo uso de multiprocessos (versão 1) e multithreads (versão 2).

2.1. Versão processos

Esta versão se caracteriza pela distribuição da computação entre N processos UNIX, sua programação baseando-se nas primitivas' fork()' e 'wait()'. Além disso, em função da necessidade de uso de memória compartilhada, fez se uso da primitiva 'mmap()'.

Nesta versão, cada processo se responsabiliza por determinado conjunto de linhas da matriz 1, ficando encarregado de calcular os produtos escalares de cada uma destas linhas com cada coluna da matriz 2. As linhas são designadas para os processos seguindo a seguinte metodologia: sendo *N* o número total de processos, ao processo de índice *P* são designadas todas as linhas cujo índice *L* satisfaz a seguinte equação: $L \bmod N = P$.

Dessa maneira, a designação de linhas para processos pode ser considerada "justa" sob a óptica de que, numa situação com *N* processos e *N*k* linhas na matriz 1, cada processo

fica encarregado da computação de exatamente *k* linhas.

2.2. Versão threads

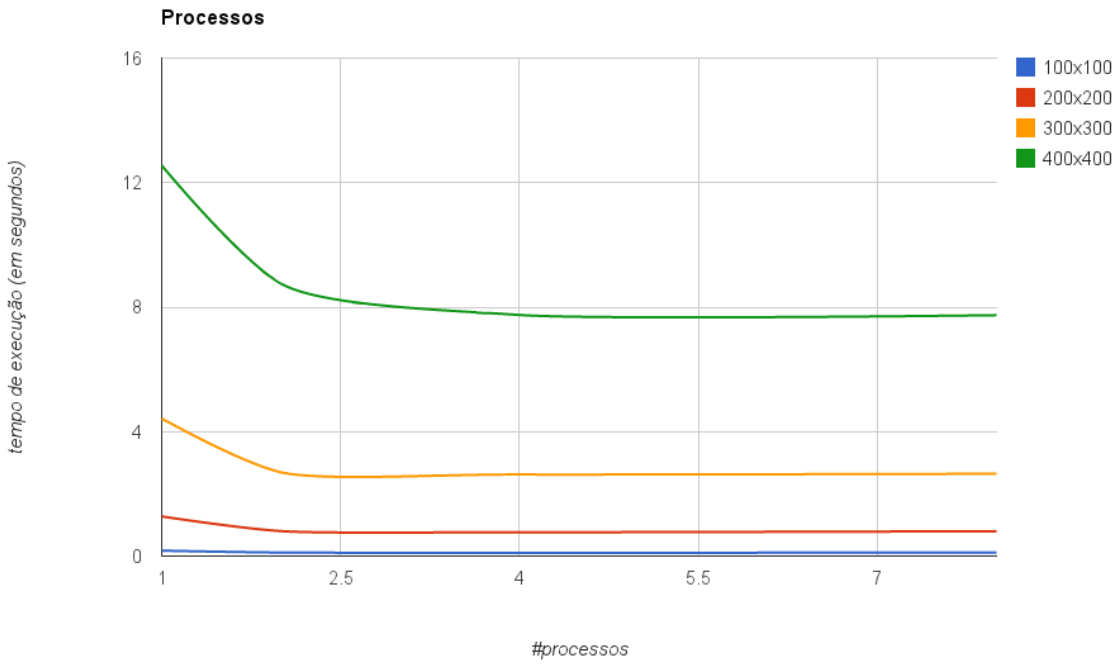
Esta versão se caracteriza pela distribuição da computação entre N threads POSIX (pthreads), sua programação baseando-se nas primitivas pthread_create() e pthread_join(). Como a versão processos, a versão thread fez uso da primitiva 'mmap()' para criação da seção de memória compartilhada.

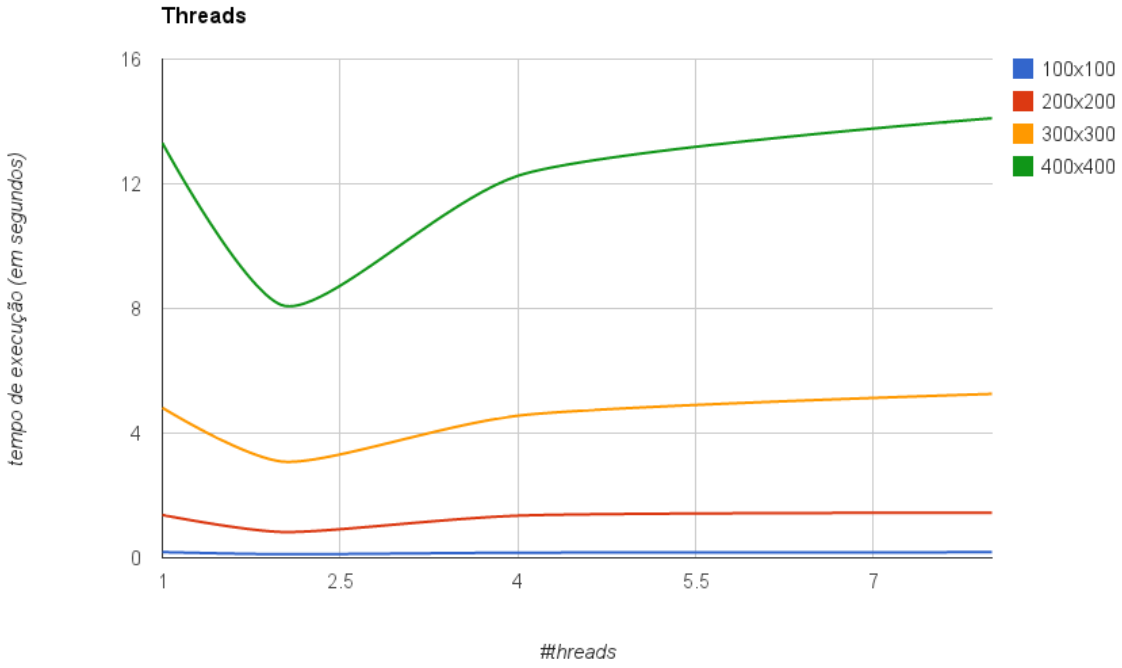
Fora isso, o cálculo da multiplicação de matrizes se dá de maneira idêntica ao da versão processos.

3. Resultados

Rodamos o programa, nas suas versões processos e threads, para diferentes combinações dos seguintes parâmetros: número de processos/threads e tamanho das matrizes de entrada.

Nos gráficos abaixo, cada curva representa um tamanho de matriz de entrada, suas curvaturas determinadas pela função que mede o tempo de execução do programa de acordo com o número de unidades concorrentes trabalhando no cálculo da multiplicação de matrizes.





4. Conclusões

Podemos observar, tanto nos resultados com processos quanto nos resultados com threads, que o tempo de execução do programa descresce com a transição da execução sequencial (um único processo/thread) para a execução paralela com exatamente dois processos/threads. Entretanto, contra-intuitivamente, a eficiência do programa nem sempre cresce de maneira monotônica com o aumento do grau de paralelismo. Pelo contrário, na versão threads, o tempo de execução tem um mínimo global (maior eficiência) em #threads = 2 e, a partir daí, cresce (perde eficiência) monotonicamente. Na versão processos a situação não é tão acentuada, com o tempo de execução diminuindo significativamente de #threads = 1 para #threads = 2 e de maneira pouco acentuada em seguida.

Uma possível explicação deste fenômeno leva em consideração o grau de paralelismo físico da máquina na qual os testes foram executados: uma decorrência intuitiva da natureza dual-core do processador é a de que, justamente, o melhor resultado possível será dado pela execução do programa com grau de paralelismo 2, dois sendo coincidentemente o número de núcleos do processador.

Um fenômeno a ser explicado é a diferença entre o comportamento das curvas nos gráficos relativos a processos e a threads. Uma hipótese é a de que o custo da criação e manutenção de N threads passa a não compensar o ganho de eficiência conferido pelo

paralelismo na computação para valores muito grandes de N. No nosso caso, em que o número de núcleos do processador é de 2, é intuitivo que a partir de N = 3 tenhamos eficiência prejudicada, como os gráficos indicam. O custo da manutenção dos processos pode não ser tão acentuado quanto o de threads, visto que o escalonamento de processos é executado pelo kernel do sistema operacional, ao passo que o escalonamento de threads depende da biblioteca POSIX, o que pode inserir um overhead na operação.