UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
INF01151 – SISTEMAS OPERACIONAIS II N
TURMA A

TIAGO MAZZAROLO DE PAULA

Trabalho Prático: comparativo do uso de threads x processos aplicado ao problema de multiplicação de matrizes

Prof. Alberto Egon Schaeffer Filho

Porto Alegre, 25 julho de 2017

1. Introdução

O trabalho apresenta uma comparação do tempo médio de processamento entre duas implementações distintas do problema de multiplicação de matrizes, uma das implementações faz uso dos processo UNIX e a outra utiliza *pthreads*.

Nas duas implementações foi utilizado um algoritmo simples, com laços aninhados, para fazer a multiplicação das matrizes. Ambas recebem como entrada dois arquivos com as matrizes A e B e um valor n que representa a quantidade de processos/threads que devem ser geradas, ao final do processamento grava um arquivo com a matriz resultante. A resolução concorrente do problema é feita nas linhas da matriz A. Sendo distribuídas LA / n linhas para cada processo, sendo LA a quantidade de linhas da matriz de entrada A e n o valor informado na entrada do programa. Caso a divisão não tenho resultado inteiro as linhas restante são atribuídas para o último processo/thread.

Abaixo são mostradas comparações entre o tempo médio de execução das duas implementações para matrizes de diferentes tamanhos com valores de entrada $n = 1,\ 2,\ 4$ e 8 .

2. Testes elaborados

2.1. Ambiente

As execuções foram realizadas em um máquina com as seguintes configurações: *Ubuntu 16.04.2 LTS - Xenial*, com memória principal disponível de 7,7 GiB e processador *Intel® Core™ i7-7500U CPU @ 2.70GHz×4 64-bit*. O compilador utilizado foi: *gcc 5.4.0*.

2.2 Casos de teste

Foram utilizados 6 pares de matrizes de tamanhos distintos. Cada caso de teste foi nomeado com os labels A, B, C, D, E e F, conforme tabela abaixo. Essa nomenclatura será utilizada nas próximas seções.

Matriz	in1.txt	int2.txt
A	32x64	64x128
B	64x128	128x256
C	128x256	256x512
D	256x512	512x1024
E	512x1024	1024x2048
F	1024x2048	2048x4096

Para cada entrada $n = 1, 2, 4$ e 8 e para cada versão da implementada (processo/thread) foram feitas 10 execuções consecutivas e os tempos de execução e o tempo médio foram gravados. A mensuração do tempo foi feita utilizando o comando *time*.

As matrizes foram geradas com números inteiros positivos aleatórios entre 0 e 10. Os testes foram automatizados e realizados utilizando os scripts definidos em */scripts* conforme estrutura mostrada no item 3.3.

3. Características da implementação

As duas implementações realizam as atividades periféricas do problema da multiplicação de matrizes de forma idêntica. A validação da entrada, a leitura e armazenamento das matrizes, o cálculo das linhas a serem atribuídas para cada processo/thread e parte do processo de gravação do arquivo resultado são funções compartilhadas.

O processo de cálculo de um conjunto de linhas atribuído a um processo filho ou uma thread também é semelhante nas duas implementações, a principal diferença está associada a forma como esse processo/thread filho deve retornar o resultado ao programa principal.

De forma geral a distribuição das linhas e a divisão do trabalho é feita da seguinte forma: as linhas que devem ser processadas por cada processo são armazenadas em um vetor onde a posição i indica as linhas que devem ser processadas pelo processo/thread i. Nesse cenário a comunicação entre as unidades do programa precisa ser feita de duas maneiras, o processo principal deve informar aos seus filhos qual o identificador i que eles possuem e cada filho ao final de sua execução deve retornar o resultado do processamento das linhas que lhe foram atribuídas.

Essa comunicação entre o programa principal e seus filhos foi resolvida de formas diferentes em cada implementação.

3.1 Implementação usando processos

Para comunicação entre o processos pai e seus filhos foi utilizado memórias compartilhadas. Uma memória compartilhada é criada para cada processo filho o qual grava, ao final de seu processamento, o resultado de seu processamento.

O programa principal mantém um contador (*myid*) para a quantidade de processos já criados (quantidade de *fork()* feitos), esse contador é incrementado no processo principal após cada *fork()*. Como o processo filho acessa uma cópia desse contador que está em uma versão anterior ao incremento, ele conhece o seu id e sabe quais as linhas deve processar consultando o vetor, que contém as linhas atribuídas para cada processo, na posição *myid*.

O programa principal fica aguardando (*busy waiting*) todos os processos filhos terminarem o processamento. Ao final o vetor contendo as memórias compartilhadas é lido e gravado no arquivo resultado.

3.2 Implementação usando threads

Diferente da implementação utilizando processos o retorno do processamento feito pelas unidades filhas não houve necessidade do uso de memória compartilhada. Como as *threads* compartilham a região de dados, uma matriz de resultado foi alocada bastando cada *thread* armazenar o resultado de seu processamento nas linhas que foram-lhe atribuídas.

Por outro lado, não havendo a cópia do dados, mas sim o compartilhamento deles, a mesma solução utilizada na implementação com processos para passar quais linhas cada *thread* filha deve processar não pode ser utilizada. Sendo assim, uma estrutura contendo a linha inicial e linha final atribuída para cada *thread* foi criada. Essa estrutura é passada para a *thread* em sua criação.

3.3 Organização do projeto

O diretório do projeto está organizado da seguinte maneira.

```
├──── bin
├──── data
│    ├──── m-32x64-64x128
│    .    ├──── in1.txt
│    .    └──── in2.txt
│    .
│    └──── m-1024x2048-2048x4096
│      ├──── in1.txt
│      └──── in2.txt
├──── inc
│    └──── util.h
├──── Makefile
├──── results
├──── scripts
│    ├──── all.sh
│    ├──── create.sh
│    └──── exec.sh
└──── src
       ├──── processos.c
       ├──── threads.c
       └──── util.c
```

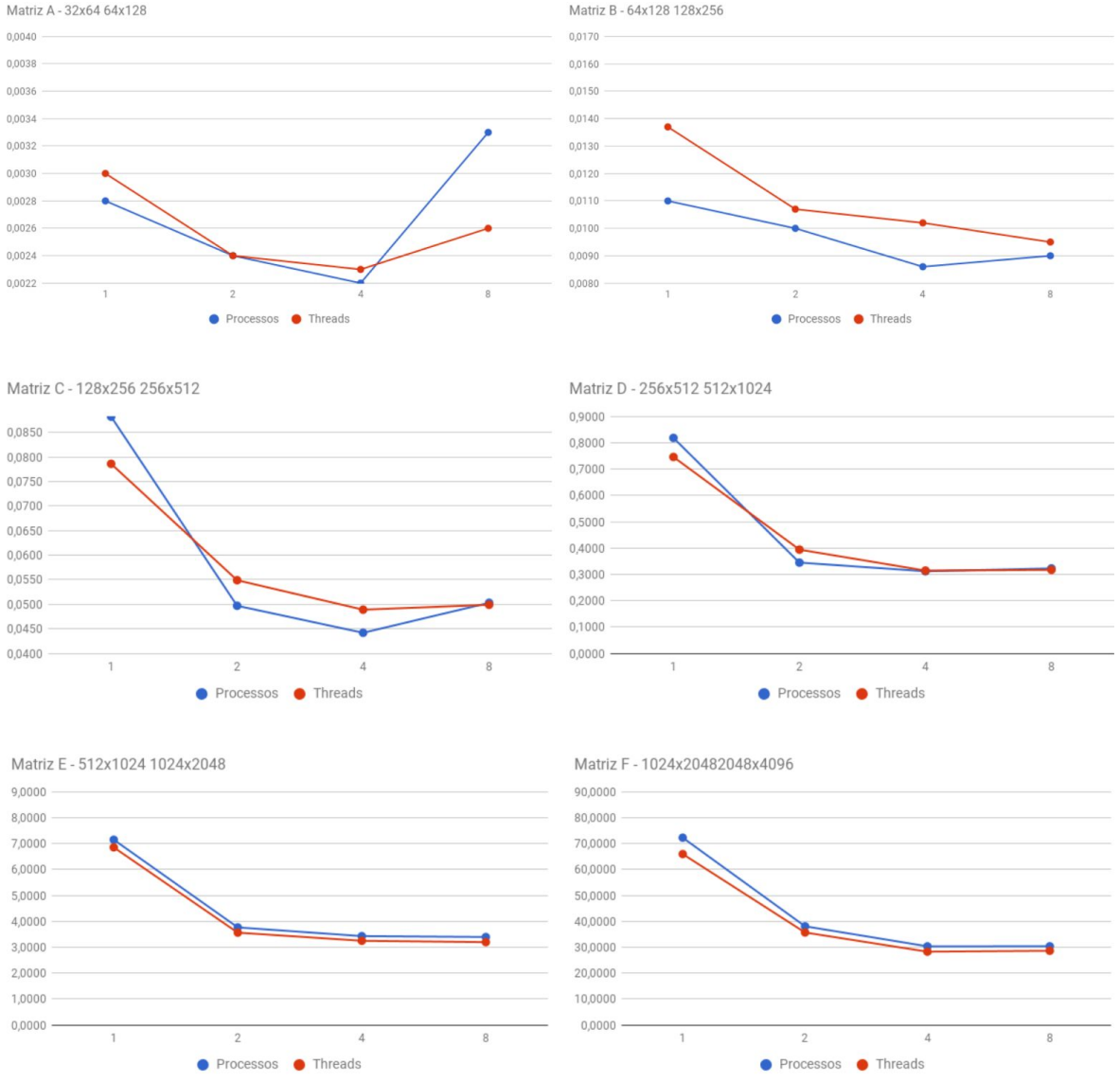
Sendo `/bin` diretório contendo os arquivo gerados após *make*, `/data` uma lista de diretórios cada um contendo o par de matrizes geradas para os testes, `/inc` diretório contendo

os arquivos *.h*. `/results` o diretório contendo os arquivos de resultado com os tempos de processamento de cada caso de teste e a média das 10 execuções. `/scripts` conjunto de scripts para geração das matrizes de teste e execução dos casos de teste para cada valor de n. `/src` códigos das implementações sendo *util.c* as funções compartilhadas entre as duas implementações.

4. Resultados

Abaixo são mostrados gráficos contendo o tempo de execução para cada caso de teste: A, B, C, D, E e F superpondo os tempos médios de execução avaliados para *threads* e para processos com as entradas $n = 1, 2, 4$ e 8. Todos os gráficos mostram a escala em segundos.

No Anexo I os dados dos tempos médios e do tempo de execução são mostrados de forma tabular.



5. Análise dos resultados

De forma geral, fica evidente o ganho de desempenho entre a execução sequencial e a execução concorrente, apesar de esse crescimento não se manter conforme *n* cresce. Em relação a comparação entre as implementações usando *threads* ou processos, essa diferença não é tão aparente. Abaixo alguns cenários são discutidos.

5.1 $n = 1$ *versus* $n = 2$

Nesse cenário o ganho de desempenho é o mais evidente, tanto para a implementação usando *threads* quanto para que utiliza processos. Em todos os casos de teste houve um ganho de desempenho significativo. Como pode ser observado na tabela abaixo, a taxa de redução chegou a quase ficou próximo aos 50% nos casos pares de matrizes maiores.

Caso de teste	Implementação	n=1	n=2	Redução	%Redução
A	Processos	0,0028	0,0024	0,0004	14,29%
A	Threads	0,0030	0,0024	0,0006	20,00%
B	Processos	0,0110	0,0100	0,0010	9,09%
B	Threads	0,0137	0,0107	0,0030	21,90%
C	Processos	0,0882	0,0497	0,0385	43,65%
C	Threads	0,0786	0,0549	0,0237	30,15%
D	Processos	0,8192	0,3452	0,4740	57,86%
D	Threads	0,7468	0,3946	0,3522	47,16%
E	Processos	7,1536	3,7674	3,3862	47,34%
E	Threads	6,8578	3,5643	3,2935	48,03%
F	Processos	72,3223	38,1041	34,2182	47,31%
F	Threads	65,9997	35,7106	30,2891	45,89%

Nos casos de teste menores o ganho não é tão evidente, provavelmente devido a pouco que o cálculo em si teve nesses cenários, onde o tempo de execução é tão é pequeno que as outras atividades periféricas ao processamento tem mais peso do que a realização do cálculo da multiplicação de matrizes.

Tal redução pode ser explicada pela quantidade de processadores disponível na máquina utilizada no teste. Utilizando dois processos/*threads* a execução dos programas

ocorreu de forma praticamente em paralela, uma vez que havia um núcleo de processamento para processo/*thread* do processo.

Para $n = 1$ nota-se uma diferença entre as implementações utilizando *threads* e utilizando processos. Nos casos de teste com matrizes menores, *threads* são mais rápida do que processos e conforme o par de matrizes de teste aumenta de tamanho essa diferença é reduzida e até mesmo invertida, essa diferença de tempo, provavelmente, está associada a forma como cada implementação armazena e grava os resultados parciais de processamento, uma vez que nos casos de teste com pares de matrizes maiores essa diferença é reduzida e parece tender a se igualar nos dois casos.

5.2 $n = 2, 4$ e 8

Primeiramente, em uma comparação geral $n = 1, 2, 4$ e 8, independente do tipo de implementação, nota-se como citado em 5.1 uma melhoria significativa de desempenho entre $n = 1$ e $n = 2$, com taxa de redução de até 50%, porém esse valor não se mantém com o aumento de n.

Esse comportamento é, de certa forma, esperado na comparação entre $n = 4$ e $n = 8$, uma vez que a máquina possui somente 4 núcleos de processamento e portanto existem mais processos/*threads* competindo pelos mesmo recursos, diferente no cenário com $n = 2$.

Já na comparação entre $n = 2$ e $n = 4$, podia-se esperar que a taxa de redução se mantivesse como ocorreu e foi explicado no cenário descrito em 5.1. Porém isso não aconteceu, provavelmente, devido ao ambiente onde os teste foram executados. Apesar da máquina ter 4 núcleos o programa sendo testado não tinha uso exclusivo da máquina e competia com outros processos externos.

Agora, comparando o desempenho das diferentes implementações nos diversos casos de testes apresentados, não há uma diferença evidente e uniforme. Nos casos de testes D, E e F *threads* e processos tem desempenho muito semelhante, havendo um pequeno gap que se mantém durante os valores de n.

Nos casos A, B e C pouco pode ser concluído considerando a baixa quantidade de dados e o ambiente de teste utilizado. No caso de teste A com $n = 8$ existe uma diferença visível entre a execução com *threads* e a execução com processos, é possível que essa esteja associada ao custo de criação de processos filhos que fica mais evidente nesse cenário devido à baixa quantidade de dados e diferença no custo de criação de uma *thread* e de um processo.

6. Conclusões

Fica evidente a melhoria de desempenho conforme ocorre o aumento de n e existem núcleos disponíveis. Quando o número de processo/*threads* fica próximo ao número núcleos disponíveis a melhoria de desempenho reduz. Claro que esse é também um comportamento atrelado à implementação feita para resolução do problema, nenhum acesso a periféricos é feito pelos processos/*threads* filhas, caso houvesse o melhoria no desempenho poderia ser melhor com o aumento de n mesmo que este fosse maior do que o número de núcleos da máquina.

Em relação as diferentes implementações, pouco pode ser concluído em relação ao desempenho no processamento, uma vez que não houve uma diferença evidente, conforme dados analisados.

O que fica claro relação às implementações feitas é a diferença entre as duas versões referente a simplicidade de código, controle dos processos/*threads* filhas e troca/compartilhamento de dados. Em todos esses itens a utilização de *pthreads* foi mais prático do que o uso de processos.

Anexo I

Dados brutos dos resultados obtidos

1. Tempo médio de processamento de cada caso de teste, para cada *n* nas duas implementações realizadas

Matriz	n	Processos	Threads
A	1	0,0028	0,0030
A	2	0,0024	0,0024
A	4	0,0022	0,0023
A	8	0,0033	0,0026
B	1	0,0110	0,0137
B	2	0,0100	0,0107
B	4	0,0086	0,0102
B	8	0,0090	0,0095
C	1	0,0882	0,0786
C	2	0,0497	0,0549
C	4	0,0442	0,0489
C	8	0,0503	0,0499
D	1	0,8192	0,7468
D	2	0,3452	0,3946
D	4	0,3124	0,3148
D	8	0,3231	0,3177
E	1	7,1536	6,8578
E	2	3,7674	3,5643
E	4	3,4307	3,2511
E	8	3,3968	3,1993
F	1	72,3223	65,9997
F	2	38,1041	35,7106
F	4	30,3679	28,2918
F	8	30,3868	28,6437

Anexo II

Links material complementar

1. Código implementações

   http://github.com/tmazza/multiplicacao-de-matrizes

2. Planilha e gráficos tempos médios

   https://docs.google.com/spreadsheets/d/1k8qltc9QFsut_mYOuxSm5SWDnbPZju2LsPNDvN158g8

3. Dados utilizados

   https://www.dropbox.com/s/i0hg2rjhw1hjhhy/dados-multiplicacao-de-matrizes.zip