

**Universidade Federal do Rio de Janeiro**

**DCC - IM**

T1 - Sistemas Operacionais

Multiplicação de Matrizes

**Universidade Federal do Rio de Janeiro**

**Sistemas Operacionais**

**Leonardo Neves da Silva**

**Luan Cerqueira Martins**

**Rio de Janeiro**

**2015.1**

**SUMÁRIO**

1. **Grupo ………………………………... página 4**
2. **Definição do trabalho …………….. página 4**
3. **Códigos ……………………………... página 5**
   1. **processo único ....………….. página 5**
   2. **com subprocessos ………... página 10**
   3. **com threads ………………… página 15**
4. **Resultados ………………………….. página 22**
5. **Considerações finais ……………… página 23**

**Grupo**

* **Leonardo Neves da Silva**
* **Luan Cerqueira Martins**

[**https://github.com/sevenleo/MutiplicaMatrizes**](https://github.com/sevenleo/MutiplicaMatrizes)

**Definição do trabalho**

***O enunciado do trabalho que me passaram, pra quem ficou sem.***

***Realizar o produto de 2 matrizes***

* ***Criar subprocessos para multiplicar cada linha A e coluna B***
* ***Criar Threads para multiplicar cada linha A e coluna B***
  + ***Para ambos os casos medir os tempos execução***
  + ***Aumentar a ordem da matriz***
  + ***N = {3,10,50,100,200,500}***

***Fazer uma tabela de avaliação***

* ***Monitorar a ordem de execução dos processos e threads***
* ***imprimir: numero do processo, indice de R e resultado***

**Código - versão “com apenas um processo”**

|  |  |
| --- | --- |
|  | //Leonardo Neves da Silva DRE110155777 |
|  | //Luan Cerqueira Martins DRE111211704 |
|  | //T1 SO 2015.1 ProfValeria |
|  |  |
|  | #include <stdio.h> |
|  | #include <stdlib.h> |
|  | #include <time.h> |
|  | #include <sys/timeb.h> |
|  |  |
|  | #define BASE 100 |
|  |  |
|  | //parametros |
|  | typedef struct { |
|  | int \*\*mat1; |
|  | int \*\*mat2; |
|  | int coluna; |
|  | int linha; |
|  | int thread; |
|  | } Calculos ; |
|  |  |
|  |  |
|  | //PRE-DEFINIDOS |
|  | int dimensao; |
|  | int pai; |
|  | int temp=0; |
|  |  |
|  |  |
|  | //cria matrizes |
|  | int \*\*matrizA; |
|  | int \*\*matrizB; |
|  | int \*\*matrizResultado; |
|  |  |
|  | //funcoes |
|  | void gerasite(int \*\*matrizA,int \*\*matrizB); |
|  | void preenche(int \*\*matriz,int semente); |
|  | void imprime (int \*\*matriz); |
|  | void calcula(int t); |
|  | void multiplica(int \*\*mat1,int \*\*mat2); |
|  |  |
|  |  |
|  | ////////////////////////////////////////////////////// |
|  |  |
|  | int main( int argc, char \*argv[ ] ){ |
|  |  |
|  | //variaveis de tempo |
|  | struct timeb start, stop; |
|  | double elapsed; |
|  |  |
|  |  |
|  | int i; |
|  |  |
|  | if (argc > 1 ) { |
|  | dimensao = atoi(argv[1]); |
|  | } |
|  | else { |
|  | printf("\nQual a dimensao N (NxN) de suas matrizes?\n"); |
|  | scanf("%i",&dimensao); |
|  | } |
|  |  |
|  | //Cria matrizes |
|  | if (dimensao<=0 ) return -1; |
|  | matrizA = (int \*\*)malloc(dimensao\*sizeof(int \*)); |
|  | matrizB = (int \*\*)malloc(dimensao\*sizeof(int \*)); |
|  | matrizResultado = (int \*\*)malloc(dimensao\*sizeof(int \*)); |
|  | for (i=0;i<dimensao;i++) matrizA[i] = (int \*)malloc(dimensao\*sizeof(int)); |
|  | for (i=0;i<dimensao;i++) matrizB[i] = (int \*)malloc(dimensao\*sizeof(int)); |
|  | for (i=0;i<dimensao;i++) matrizResultado[i] = (int \*)malloc(dimensao\*sizeof(int)); |
|  |  |
|  |  |
|  | //PREENCHE MATRIZ |
|  | preenche(matrizA,time(NULL)); |
|  | preenche(matrizB,time(NULL)+1); |
|  |  |
|  | //THREADS CALCULAM A MULTIPLICACAO |
|  | ftime(&start); //inicia timer |
|  | multiplica(matrizA,matrizB); |
|  | ftime(&stop); //para timer |
|  | elapsed=((double) stop.time + ((double) stop.millitm \* 0.001)) - ((double) start.time + ((double) start.millitm \* 0.001)); |
|  |  |
|  | //IMPRIME |
|  | printf ("\nMatriz A:"); |
|  | imprime(matrizA); |
|  | printf ("\nMatriz B:"); |
|  | imprime(matrizB); |
|  | printf ("\n-Resultado:"); |
|  | imprime(matrizResultado); |
|  |  |
|  | //gera link |
|  | gerasite(matrizA,matrizB); |
|  |  |
|  | printf("\n[ProcessoUnico] Matrizes = %i x %i -> O tempo de execucao e de %.3lf\n", dimensao, dimensao,elapsed); |
|  |  |
|  | //exit(0); |
|  | return 0; |
|  | } |
|  |  |
|  | ////////////////////////////////////////////////////// |
|  |  |
|  |  |
|  | void preenche(int \*\*matriz,int semente){ |
|  | int i,j,temp; |
|  | srand(semente); |
|  |  |
|  | for (j=0;j<dimensao;j++){ |
|  | for (i=0;i<dimensao;i++) { |
|  | matriz[i][j]=rand()%BASE; |
|  |  |
|  | temp=rand()%2; |
|  | if (temp ==0) matriz[i][j] \*= (-1); |
|  | } |
|  | } |
|  | } |
|  |  |
|  | void imprime (int \*\*matriz){ |
|  | int i,j; |
|  | printf ("\n"); |
|  | for (j=0;j<dimensao;j++) { |
|  | for (i=0;i<dimensao;i++) { |
|  | printf ("%8i ",matriz[i][j]); |
|  | } |
|  | printf ("\n"); |
|  | } |
|  |  |
|  |  |
|  | } |
|  |  |
|  | void gerasite(int \*\*matrizA,int \*\*matrizB){ |
|  | int i,j; |
|  |  |
|  | printf ("\n google-chrome ' http://www.wolframalpha.com/input/?i="); |
|  | printf ("{"); |
|  | for (j=0;j<dimensao;j++) { |
|  | printf ("{"); |
|  | for (i=0;i<dimensao;i++) { |
|  | if (i==0) printf ("%i",matrizA[i][j]); |
|  | else printf (",%i",matrizA[i][j]); |
|  |  |
|  | } |
|  | if (j+1==dimensao) printf ("}"); |
|  | else printf ("},"); |
|  | } |
|  | printf ("}"); |
|  | printf (" \* "); |
|  | printf ("{"); |
|  | for (j=0;j<dimensao;j++) { |
|  | printf ("{"); |
|  | for (i=0;i<dimensao;i++) { |
|  | if (i==0) printf ("%i",matrizB[i][j]); |
|  | else printf (",%i",matrizB[i][j]); |
|  |  |
|  | } |
|  | if (j+1==dimensao) printf ("}"); |
|  | else printf ("},"); |
|  | } |
|  | printf ("}'\n"); |
|  | } |
|  |  |
|  |  |
|  | void calcula(int t){ |
|  |  |
|  | int linha,coluna,acumula,k; |
|  | linha=t; |
|  | for (coluna=0;coluna<dimensao;coluna++){ |
|  | acumula=0; |
|  | for (k=0;k<dimensao;k++){ |
|  | acumula=acumula+matrizA[k][coluna]\*matrizB[linha][k]; |
|  | } |
|  | matrizResultado[linha][coluna]=acumula; |
|  | } |
|  | } |
|  |  |
|  | void multiplica(int \*\*mat1,int \*\*mat2){ |
|  |  |
|  | int i,rc,acumula; |
|  | pthread\_t th[dimensao]; |
|  |  |
|  | for (i=0;i<dimensao;i++){ |
|  | calcula(i); |
|  | } |
|  | } |

**Código - versão “com apenas um processo”**

|  |
| --- |
| ./simples 4  Matriz A:  -35 5 -47 94  43 85 -44 70  63 -78 -11 -8  46 -57 36 83  Matriz B:  97 60 -60 -91  38 -69 64 67  40 -66 46 -99  -30 35 -73 -68  -Resultado:  -7905 3947 -6604 1781  3541 2069 -4274 1378  2947 9608 -8694 -9326  1246 7222 -10811 -17213  google-chrome ' http://www.wolframalpha.com/input/?i={{-35,5,-47,94},{43,85,-44,70},{63,-78,-11,-8},{46,-57,36,83}} \* {{97,60,-60,-91},{38,-69,64,67},{40,-66,46,-99},{-30,35,-73,-68}}'  [ProcessoUnico] Matrizes = 4 x 4 -> O tempo de execucao e de 0.000 |

**Codigo - versão “com subprocessos”**

|  |
| --- |
|  |
| //Luan Cerqueira Martins DRE111211704  //Leonardo Neves da Silva DRE110155777 |
| //T1 SO 2015.1 ProfValeria |
|  |
| #include <stdio.h> |
| #include <stdlib.h> |
| #include <time.h> |
| #include <sys/timeb.h> |
| #include <signal.h> |
| #define BASE 100 |
|  |
| typedef struct { |
| int linha; |
| int coluna; |
| int valor; |
| } Elemento; |
|  |
| //PRE-DEFINIDOS |
| int dimensao; |
| int pai; |
| int temp=0; |
|  |
|  |
| //cria matrizes |
| int \*\*matrizA; |
| int \*\*matrizB; |
| int \*\*matrizResultado; |
|  |
| //funcoes |
| pid\_t gettid( void ) ; |
| void gerasite(int \*\*matrizA,int \*\*matrizB); |
| void preenche(int \*\*matriz,int semente); |
| void imprime (int \*\*matriz); |
| Elemento calcula (int t); |
| void multiplica(int \*\*mat1,int \*\*mat2); |
|  |
|  |
| ////////////////////////////////////////////////////// |
|  |
| int main( int argc, char \*argv[ ] ){ |
|  |
| //variaveis de tempo |
| struct timeb start, stop; |
| double elapsed; |
|  |
|  |
| int i; |
| if (argc > 1 ) { |
| dimensao = atoi(argv[1]); |
| } |
| else { |
| printf("\nQual a dimensao N (NxN) de suas matrizes?\n"); |
| scanf("%i",&dimensao); |
| } |
|  |
|  |
|  |
| //Cria matrizes |
| if (dimensao<=0 ) return -1; |
| matrizA = (int \*\*)malloc(dimensao\*sizeof(int \*)); |
| matrizB = (int \*\*)malloc(dimensao\*sizeof(int \*)); |
| matrizResultado = (int \*\*)malloc(dimensao\*sizeof(int \*)); |
| for (i=0;i<dimensao;i++) matrizA[i] = (int \*)malloc(dimensao\*sizeof(int)); |
| for (i=0;i<dimensao;i++) matrizB[i] = (int \*)malloc(dimensao\*sizeof(int)); |
| for (i=0;i<dimensao;i++) matrizResultado[i] = (int \*)malloc(dimensao\*sizeof(int)); |
|  |
|  |
| //PREENCHE MATRIZ |
| preenche(matrizA,time(NULL)); |
| preenche(matrizB,time(NULL)+1); |
|  |
| //THREADS CALCULAM A MULTIPLICACAO |
| ftime(&start); //inicia timer |
| multiplica(matrizA,matrizB); |
| ftime(&stop); //para timer |
| elapsed=((double) stop.time + ((double) stop.millitm \* 0.001)) - ((double) start.time + ((double) start.millitm \* 0.001)); |
|  |
| //IMPRIME |
| printf ("\nMatriz A:"); |
| imprime(matrizA); |
| printf ("\nMatriz B:"); |
| imprime(matrizB); |
| printf ("\n-Resultado:"); |
| imprime(matrizResultado); |
|  |
| //gera link |
| gerasite(matrizA,matrizB); |
|  |
| printf("\n[Subprocessos=%i] Matrizes = %i x %i -> O tempo de execucao e de %.3lf\n",dimensao, dimensao, dimensao,elapsed); |
|  |
| //exit(0); |
| return 0; |
| } |
|  |
| ////////////////////////////////////////////////////// |
|  |
|  |
| void preenche(int \*\*matriz,int semente){ |
| int i,j,temp; |
| srand(semente); |
|  |
| for (j=0;j<dimensao;j++){ |
| for (i=0;i<dimensao;i++) { |
| matriz[i][j]=rand()%BASE; |
|  |
| temp=rand()%2; |
| if (temp ==0) matriz[i][j] \*= (-1); |
| } |
| } |
| } |
|  |
| void imprime (int \*\*matriz){ |
| int i,j; |
| printf ("\n"); |
| for (j=0;j<dimensao;j++) { |
| for (i=0;i<dimensao;i++) { |
| printf ("%8i ",matriz[i][j]); |
| } |
| printf ("\n"); |
| } |
|  |
|  |
| } |
|  |
| void gerasite(int \*\*matrizA,int \*\*matrizB){ |
| int i,j; |
|  |
| printf ("\n google-chrome ' http://www.wolframalpha.com/input/?i="); |
| printf ("{"); |
| for (j=0;j<dimensao;j++) { |
| printf ("{"); |
| for (i=0;i<dimensao;i++) { |
| if (i==0) printf ("%i",matrizA[i][j]); |
| else printf (",%i",matrizA[i][j]); |
|  |
| } |
| if (j+1==dimensao) printf ("}"); |
| else printf ("},"); |
| } |
| printf ("}"); |
| printf (" \* "); |
| printf ("{"); |
| for (j=0;j<dimensao;j++) { |
| printf ("{"); |
| for (i=0;i<dimensao;i++) { |
| if (i==0) printf ("%i",matrizB[i][j]); |
| else printf (",%i",matrizB[i][j]); |
|  |
| } |
| if (j+1==dimensao) printf ("}"); |
| else printf ("},"); |
| } |
| printf ("}'\n"); |
| } |
|  |
| Elemento calcula(int t){ |
| FILE \*arq; |
| Elemento resposta; |
| int linha,coluna,acumula,k; |
| linha=t; |
| for (coluna=0;coluna<dimensao;coluna++){ |
| acumula=0; |
| for (k=0;k<dimensao;k++){ |
| acumula=acumula+matrizA[k][coluna]\*matrizB[linha][k]; |
| } |
| //matrizResultado[linha][coluna]=acumula; |
|  |
| resposta.coluna=coluna; |
| resposta.linha=linha; |
| resposta.valor=acumula; |
|  |
| printf("\n ## Sub=%i PID=%i Encontrou=%i para a posicao [%i,%i]",t,(int)getpid(),acumula,linha,coluna); |
| arq = fopen("dadossubprocessos.txt", "a"); |
| fprintf(arq,"%i %i \t%i\n", resposta.linha,resposta.coluna,resposta.valor); |
| fclose (arq); |
|  |
| } |
| exit(0); |
| return resposta; |
| //exit(0); |
|  |
| } |
|  |
| void multiplica(int \*\*mat1,int \*\*mat2){ |
|  |
| int i,rc,acumula; |
| int dimensao2=dimensao\*dimensao; |
| int id=1; |
| Elemento x; |
| FILE \*arq; |
| arq = fopen("dadossubprocessos.txt", "w+"); |
| fclose (arq); |
|  |
| for (i=0;i<dimensao;i++){ |
| if (id!=0){ |
| id=fork(); |
| } |
| if (id==0){ |
| x=calcula(i); |
|  |
| } |
|  |
| } |
|  |
| if( id != 0 ) for (i=0;i<dimensao2;i++) wait(); |
| else kill(getpid(), SIGKILL); |
|  |
| //soh o pai acessa essa area de Codigo, os filhos ja retornaram ou morreram |
| arq=fopen("dadossubprocessos.txt","r"); |
| if (arq == NULL) { |
| printf ("Houve um erro ao abrir o arquivo.\n"); |
| exit(-1); |
| } |
| for (i=0;i<dimensao2;i++){ |
| fscanf(arq,"%i %i %i", &x.linha,&x.coluna,&x.valor); |
| //printf("%i %i \t%i\n", x.linha,x.coluna,x.valor); |
| matrizResultado[x.linha][x.coluna]=x.valor; |
| } |
| } |

**Codigo - versão “com subprocessos”**

**Aquivo de saída:**

A versão com subprocesscos gera um arquivos utilizado para a comunicação entre os processos filhos e o pai. Os filhos calculam os valores da matriz resultante e gravam no arquivo junto com a respectiva posição, passando assim as informações atualizadas que devem ser inseridas na matriz pelo processo pai.

./dadossubprocessos.txt

#linha #coluna #valor

0 0 2995  
0 1 -6364  
0 2 -743  
2 0 7039  
2 1 -11348  
2 2 -2087  
1 0 -176  
1 1 280  
1 2 3504

**Codigo - versão “com subprocessos”**

|  |
| --- |
| ./subprocessos 4  ## Sub=0 PID=12315 Encontrou=-3700 para a posicao [0,0]  ## Sub=0 PID=12315 Encontrou=3346 para a posicao [0,1]  ## Sub=0 PID=12315 Encontrou=-1826 para a posicao [0,2]  ## Sub=0 PID=12315 Encontrou=-4499 para a posicao [0,3]  ## Sub=3 PID=12318 Encontrou=6994 para a posicao [3,0]  ## Sub=3 PID=12318 Encontrou=-9673 para a posicao [3,1]  ## Sub=3 PID=12318 Encontrou=-10292 para a posicao [3,2]  ## Sub=3 PID=12318 Encontrou=-2608 para a posicao [3,3]  ## Sub=2 PID=12317 Encontrou=-4700 para a posicao [2,0]  ## Sub=2 PID=12317 Encontrou=606 para a posicao [2,1]  ## Sub=2 PID=12317 Encontrou=9378 para a posicao [2,2]  ## Sub=1 PID=12316 Encontrou=6787 para a posicao [1,0]  ## Sub=2 PID=12317 Encontrou=5321 para a posicao [2,3] ## Sub=1 PID=12316 Encontrou=-692 para a posicao [1,1]  ## Sub=1 PID=12316 Encontrou=-13577 para a posicao [1,2]  ## Sub=1 PID=12316 Encontrou=-3472 para a posicao [1,3]  Matriz A:  -5 -67 -17 -21  13 65 40 -73  -20 79 87 53  33 -15 70 2  Matriz B:  46 -12 58 -24  73 -62 33 -96  -70 -55 54 -47  -11 -78 61 17  -Resultado:  -3700 6787 -4700 6994  3346 -692 606 -9673  -1826 -13577 9378 -10292  -4499 -3472 5321 -2608  google-chrome ' http://www.wolframalpha.com/input/?i={{-5,-67,-17,-21},{13,65,40,-73},{-20,79,87,53},{33,-15,70,2}} \* {{46,-12,58,-24},{73,-62,33,-96},{-70,-55,54,-47},{-11,-78,61,17}}'  [Subprocessos=4] Matrizes = 4 x 4 -> O tempo de execucao e de 0.002 |

**Codigo - versão “com threads”**

|  |
| --- |
|  |
| **//Luan Cerqueira Martins DRE111211704** |
| **//Leonardo Neves da Silva DRE110155777**  **//T1 SO 2015.1 ProfValeria** |
|  |
| **#include <sys/syscall.h> ///////////USAR APENAS NO LINUX** |
| **#include <stdio.h>** |
| **#include <stdlib.h>** |
| **#include <time.h>** |
| **#include <sys/timeb.h>** |
| **#define BASE 100** |
|  |
| **//parametros** |
| **typedef struct {** |
| **int \*\*mat1;** |
| **int \*\*mat2;** |
| **int coluna;** |
| **int linha;** |
| **int thread;** |
| **} Calculos ;** |
|  |
|  |
| **//PRE-DEFINIDOS** |
| **int dimensao;** |
| **int pai;** |
| **int temp=0;** |
|  |
|  |
| **//cria matrizes** |
| **int \*\*matrizA;** |
| **int \*\*matrizB;** |
| **int \*\*matrizResultado;** |
|  |
| **//funcoes** |
| **pid\_t gettid( void ) ; ///////////USAR APENAS NO LINUX** |
| **void gerasite(int \*\*matrizA,int \*\*matrizB);** |
| **void preenche(int \*\*matriz,int semente);** |
| **void imprime (int \*\*matriz);** |
| **void\* calcula(void\* arg);** |
| **void multiplica(int \*\*mat1,int \*\*mat2);** |
|  |
|  |
| **//////////////////////////////////////////////////////** |
|  |
| **int main( int argc, char \*argv[ ] ){** |
|  |
| **//variaveis de tempo** |
| **struct timeb start, stop;** |
| **double elapsed;** |
|  |
|  |
| **int i;** |
| **if (argc > 1 ) {** |
| **dimensao = atoi(argv[1]);** |
| **}** |
| **else {** |
| **printf("\nQual a dimensao N (NxN) de suas matrizes?\n");** |
| **scanf("%i",&dimensao);** |
| **}** |
|  |
|  |
| **//Cria matrizes** |
| **if (dimensao<=0 ) return -1;** |
| **matrizA = (int \*\*)malloc(dimensao\*sizeof(int \*));** |
| **matrizB = (int \*\*)malloc(dimensao\*sizeof(int \*));** |
| **matrizResultado = (int \*\*)malloc(dimensao\*sizeof(int \*));** |
| **for (i=0;i<dimensao;i++) matrizA[i] = (int \*)malloc(dimensao\*sizeof(int));** |
| **for (i=0;i<dimensao;i++) matrizB[i] = (int \*)malloc(dimensao\*sizeof(int));** |
| **for (i=0;i<dimensao;i++) matrizResultado[i] = (int \*)malloc(dimensao\*sizeof(int));** |
|  |
|  |
| **//PREENCHE MATRIZ** |
| **preenche(matrizA,time(NULL));** |
| **preenche(matrizB,time(NULL)+1);** |
|  |
| **//THREADS CALCULAM A MULTIPLICACAO** |
| **ftime(&start); //inicia timer** |
| **multiplica(matrizA,matrizB);** |
| **ftime(&stop); //para timer** |
| **elapsed=((double) stop.time + ((double) stop.millitm \* 0.001)) - ((double) start.time + ((double) start.millitm \* 0.001));** |
|  |
| **//IMPRIME** |
| **printf ("\nMatriz A:");** |
| **imprime(matrizA);** |
| **printf ("\nMatriz B:");** |
| **imprime(matrizB);** |
| **printf ("\n-Resultado:");** |
| **imprime(matrizResultado);** |
|  |
| **//gera link** |
| **gerasite(matrizA,matrizB);** |
|  |
| **printf("\n[Threads=%i] Matrizes = %i x %i -> O tempo de execucao e de %.3lf\n",dimensao, dimensao, dimensao,elapsed);** |
|  |
|  |
| **//exit(0);** |
| **return 0;** |
| **}** |
|  |
| **//////////////////////////////////////////////////////** |
|  |
|  |
| **pid\_t gettid( void ) { return syscall( \_\_NR\_gettid );} ///////////USAR APENAS NO LINUX** |
|  |
| **void preenche(int \*\*matriz,int semente){** |
| **int i,j,temp;** |
| **srand(semente);** |
|  |
| **for (j=0;j<dimensao;j++){** |
| **for (i=0;i<dimensao;i++) {** |
| **matriz[i][j]=rand()%BASE;** |
|  |
| **temp=rand()%2;** |
| **if (temp ==0) matriz[i][j] \*= (-1);** |
| **}** |
| **}** |
| **}** |
|  |
| **void imprime (int \*\*matriz){** |
| **int i,j;** |
| **printf ("\n");** |
| **for (j=0;j<dimensao;j++) {** |
| **for (i=0;i<dimensao;i++) {** |
| **printf ("%8i ",matriz[i][j]);** |
| **}** |
| **printf ("\n");** |
| **}** |
|  |
|  |
| **}** |
|  |
| **void gerasite(int \*\*matrizA,int \*\*matrizB){** |
| **int i,j;** |
|  |
| **printf ("\n google-chrome ' http://www.wolframalpha.com/input/?i=");** |
| **printf ("{");** |
| **for (j=0;j<dimensao;j++) {** |
| **printf ("{");** |
| **for (i=0;i<dimensao;i++) {** |
| **if (i==0) printf ("%i",matrizA[i][j]);** |
| **else printf (",%i",matrizA[i][j]);** |
|  |
| **}** |
| **if (j+1==dimensao) printf ("}");** |
| **else printf ("},");** |
| **}** |
| **printf ("}");** |
| **printf (" \* ");** |
| **printf ("{");** |
| **for (j=0;j<dimensao;j++) {** |
| **printf ("{");** |
| **for (i=0;i<dimensao;i++) {** |
| **if (i==0) printf ("%i",matrizB[i][j]);** |
| **else printf (",%i",matrizB[i][j]);** |
|  |
| **}** |
| **if (j+1==dimensao) printf ("}");** |
| **else printf ("},");** |
| **}** |
| **printf ("}'\n");** |
| **}** |
|  |
|  |
| **void\* calcula(void\* arg){** |
| **int t = (int)arg;** |
| **int linha,coluna,acumula,k;** |
| **linha=t;** |
| **for (coluna=0;coluna<dimensao;coluna++){** |
| **acumula=0;** |
| **for (k=0;k<dimensao;k++){** |
| **acumula=acumula+matrizA[k][coluna]\*matrizB[linha][k];** |
| **}** |
| **matrizResultado[linha][coluna]=acumula;** |
|  |
|  |
| **printf("\n ## Thread=%i TID=%i Calculou=%i para a posicao [%i,%i]",t,(int)gettid(),acumula,coluna,linha);** |
| **}** |
| **}** |
|  |
| **void multiplica(int \*\*mat1,int \*\*mat2){** |
|  |
| **int i,rc,acumula;** |
| **pthread\_t th[dimensao];** |
|  |
| **for (i=0;i<dimensao;i++){** |
| **rc = pthread\_create(&th[i], NULL, calcula,(void\*)i);** |
| **if (rc) { printf("ERROR code is %d\n", rc); exit(-1); }** |
| **}** |
|  |
| **for (i=0;i<dimensao;i++){** |
| **if (pthread\_join(th[i], NULL)) {** |
| **printf("---------------ERRO: pthread\_join() ERRO NA THREAD=%i\n",i); exit(-1);** |
| **}** |
| **}** |
|  |
|  |
| **}** |

**Código - versão “com threads”**

|  |
| --- |
| ./threads 4  ## Thread=1 TID=12484 Calculou=1835 para a posicao [0,1]  ## Thread=1 TID=12484 Calculou=-10404 para a posicao [1,1]  ## Thread=1 TID=12484 Calculou=-235 para a posicao [2,1]  ## Thread=1 TID=12484 Calculou=-2211 para a posicao [3,1]  ## Thread=2 TID=12485 Calculou=2945 para a posicao [0,2]  ## Thread=0 TID=12483 Calculou=-4068 para a posicao [0,0]  ## Thread=0 TID=12483 Calculou=804 para a posicao [1,0]  ## Thread=0 TID=12483 Calculou=-1767 para a posicao [2,0]  ## Thread=0 TID=12483 Calculou=-9257 para a posicao [3,0]  ## Thread=2 TID=12485 Calculou=-11273 para a posicao [1,2]  ## Thread=2 TID=12485 Calculou=1269 para a posicao [2,2]  ## Thread=2 TID=12485 Calculou=4488 para a posicao [3,2]  ## Thread=3 TID=12486 Calculou=-1071 para a posicao [0,3]  ## Thread=3 TID=12486 Calculou=10750 para a posicao [1,3]  ## Thread=3 TID=12486 Calculou=487 para a posicao [2,3]  ## Thread=3 TID=12486 Calculou=1369 para a posicao [3,3]  Matriz A:  -18 -15 -54 43  84 -34 94 -29  -48 -7 51 23  -32 -89 -53 38  Matriz B:  10 -54 -92 71  72 65 -2 -49  9 -42 -48 48  -54 -10 -31 48  -Resultado:  -4068 1835 2945 -1071  804 -10404 -11273 10750  -1767 -235 1269 487  -9257 -2211 4488 1369  google-chrome ' http://www.wolframalpha.com/input/?i={{-18,-15,-54,43},{84,-34,94,-29},{-48,-7,51,23},{-32,-89,-53,38}} \* {{10,-54,-92,71},{72,65,-2,-49},{9,-42,-48,48},{-54,-10,-31,48}}'  [Threads=4] Matrizes = 4 x 4 -> O tempo de execucao e de 0.002 |

**Comparação de Resultados**

\*’0 arquivo’ foi quando contabilizamos o tempo sem incluir o tempo gasto para a leitura e a escrita no arquivo

\*\*Os tempos a seguir são relativos somente ao calculo da matriz, e não incluem os prints na tela.

[ProcessoUnico] Matrizes = 3 x 3 -> O tempo de execucao e de 0.000

[ProcessoUnico] Matrizes = 10 x 10 -> O tempo de execucao e de 0.000

[ProcessoUnico] Matrizes = 50 x 50 -> O tempo de execucao e de 0.001

[ProcessoUnico] Matrizes = 100 x 100 -> O tempo de execucao e de 0.012

[ProcessoUnico] Matrizes = 200 x 200 -> O tempo de execucao e de 0.069

[ProcessoUnico] Matrizes = 500 x 500 -> O tempo de execucao e de 1.181

[ProcessoUnico] Matrizes = 1000 x 1000 -> O tempo de execucao e de 16.024

[ProcessoUnico] Matrizes = 2000 x 2000 -> O tempo de execucao e de 140.281

[1 arquivo e Subprocessos =3] Matrizes = 3 x 3 -> O tempo de execucao e de 0.001

[1 arquivo e Subprocessos =10] Matrizes = 10 x 10 -> O tempo de execucao e de 0.002

[1 arquivo e Subprocessos =50] Matrizes = 50 x 50 -> O tempo de execucao e de 0.030

[1 arquivo e Subprocessos =100] Matrizes = 100 x 100 -> O tempo de execucao e de 0.109

[1 arquivo e Subprocessos =200] Matrizes = 200 x 200 -> O tempo de execucao e de 0.475

[1 arquivo e Subprocessos =500] Matrizes = 500 x 500 -> O tempo de execucao e de 3.242

[1 arquivo e Subprocessos =1000] Matrizes = 1000 x 1000 -> O tempo de execucao e de 20.034

[1 arquivo e Subprocessos =2000] Matrizes = 2000 x 2000 -> O tempo de execucao e de 135.416

[0\* arquivo e Subprocessos =3] Matrizes = 3 x 3 -> O tempo de execucao e de 0.001

[0\* arquivo e Subprocessos =10] Matrizes = 10 x 10 -> O tempo de execucao e de 0.001

[0\* arquivo e Subprocessos =50] Matrizes = 50 x 50 -> O tempo de execucao e de 0.011

[0\* arquivo e Subprocessos =100] Matrizes = 100 x 100 -> O tempo de execucao e de 0.021

[0\* arquivo e Subprocessos =200] Matrizes = 200 x 200 -> O tempo de execucao e de 0.080

[0\* arquivo e Subprocessos =500] Matrizes = 500 x 500 -> O tempo de execucao e de 0.893

[0\* arquivo e Subprocessos =1000] Matrizes = 1000 x 1000 -> O tempo de execucao e de 9.794

[0\* arquivo e Subprocessos =2000] Matrizes = 2000 x 2000 -> O tempo de execucao e de 81.975

[Threads=3] Matrizes = 3 x 3 -> O tempo de execucao e de 0.000

[Threads=10] Matrizes = 10 x 10 -> O tempo de execucao e de 0.000

[Threads=50] Matrizes = 50 x 50 -> O tempo de execucao e de 0.005

[Threads=100] Matrizes = 100 x 100 -> O tempo de execucao e de 0.008

[Threads=200] Matrizes = 200 x 200 -> O tempo de execucao e de 0.045

[Threads=500] Matrizes = 500 x 500 -> O tempo de execucao e de 0.655

[Threads=1000] Matrizes = 1000 x 1000 -> O tempo de execucao e de 8.446

[Threads=2000] Matrizes = 2000 x 2000 -> O tempo de execucao e de 52.285

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Metodo \ Dimensao** | **3** | **10** | **50** | **100** | **200** | **500** | **1000** | **2000** |
| **Processo unico** | 0.000 | 0.000 | 0.001 | 0.012 | 0.069 | 1.181 | 16.024 | 140.281 |
| **Subprocessos**  **com arquivo** | 0.001 | 0.002 | 0.030 | 0.109 | 0.475 | 3.242 | 20.034 | 135.416 |
| **Subprocessos**  **sem arquivo** | 0.001 | 0.001 | 0.011 | 0.021 | 0.080 | 0.893 | 9.794 | 81.975 |
| **Threads** | 0.000 | 0.000 | 0.005 | 0.008 | 0.045 | 0.655 | 8.446 | 52.285 |

**Conclusão**

Observando os resultados vemos que, para matrizes pequenas, não há muita diferença perceptível, inclusive separar um arquivo muito simples em vários **sub-processos** ou em várias **threads** pode acarretar na perda de desempenho, como visto nos tempos relativos as matrizes de dimensão inferior a 50 linhas. Mas para matrizes maiores e diferença passa a ser considerável.

O método com tempo de execução mais satisfatório foi utilizando **threads**, devido ao fato de utilizarem do mesmo contexto de software e compartilharem o mesmo endereçamento de memória do processo pai, causando um tempo menor de criação e escalonamento.

O método menos satisfatório nesse caso foi criando **sub-processos utilizando gravação por arquivos**, até porque além de gastar um tempo para criar sub-processos que pode não valer a pena se sua tarefa for muito simples e curta, no método utilizamos um arquivo para fazer a comunicação entre os processos filhos e o processo pai, essa constante tarefa de leitura/escrita no disco certamente prejudicou o desempenho em tempo de execução desse método. Com isso decidimos comentar as linhas em que o arquivo é acessado apenas para analisar o tempo gasto, assim conseguimos chegar mais perto do verdadeiro tempo de execução caso os processos se comunicassem diretamente.