Computação de Alto Desempenho - COC472

Task 1

Lucas Tavares Da Silva Ferreira – DRE 120152739

Engenharia de Computação e Informação – Universidade Federal do Rio de Janeiro (UFRJ)

Rio de Janeiro, 05 de maio de 2023

Repositório do trabalho: https://github.com/lucastavarex/hpc

Introdução

Neste primeiro trabalho da disciplina Computação de Alto Desempenho, serão desenvolvidos dois códigos em linguagens distintas com o intuito de computar o resultado da multiplicação de uma matriz quadrada por um vetor. As linguagens de implementação dos códigos serão: C e Fortran90. Em cada programa de cada linguagem, tem-se 2 formas de iteração entre os elementos do vetor e da matriz, ao passo que constataremos tempos distintos para realizar a execução dos programas, e para tal, iremos cronometrar o tempo de execução para eventualmente compará-los.

No código desenvolvido na linguagem C, implementaremos 2 funções a mais, uma para povoar a matriz e o vetor com números randomicos e outra para conferir a compatibiliadde dos valores dos 2 métodos. No código desenvolvido na linguagem Fortran90, o preenchimento é feito no módulo principal ao passo que não é feita nenhuma checagem dos valores após a operação de multiplicação.

Reservaremos os resultados obtidos do tempo de execução dos programas em um arquivo CSV para posteriori compararmos os resultados por meio de um programa simples, a qual a linguagem de programação ainda não foi definida no momento de escrita desta introdução, apenas para efeito de comparação e análise gráfica.

Tasks

Task 1

Estime o tamanho máximo dos arranjos A, x e b que podem ser alocados no seu sistema para realização da tarefa

Se utilizarmos variáveis de ponto flutuante com dupla precisão, seriam necessários $(n^2 + 2n) * 64$ bits para alocar as variáveis de dimensão n. Se considerarmos o meu notebook com 12GB de RAM $12 * 10^9 * 8\ bits$, o valor máximo de n seria determinado quando:

$$(n^2 + 2n) * 64 = 12 * 10^9 * 8$$

Resolvendo a equação do 2º grau e obtendo a raíz positiva, encontramos n = 38,7 para o meu computador.

É importante observar que esse valor máximo de n não será alcançado, uma vez que o sistema operacional e outras tarefas em execução também consomem parte dessa mesma memória.

Task 2

Os arranjos a serem utilizados durante as operações devem ser inicializados com números aleátorios (A e x no caso acima)

No código em C, foi implementada uma função para preencher a matriz e os vetores, enquanto no programa em Fortran90, o preenchimento foi realizado no módulo principal. O tempo gasto nessas operações, bem como na função criada para verificar a igualdade dos valores calculados pelos dois métodos, não são incluídos na medição de tempo.

Task 3

Comece com um problema de tamanho pequeno e tente chegar ao tamanho máximo estimado no item 1. Contabilize o tempo para realização das operações para todos os tamanhos de sistema e para ambas as ordens de execução dos loops

Os códigos são executados com base em uma lista de quantidades de RAM em GB passadas como parâmetro. Ao utilizar esse comando, é determinado o valor máximo de n para cada quantidade de RAM disponível, começando por exemplo com 1GB, depois 1.5GB e assim por diante. Esse método proporciona um maior controle sobre a quantidade de RAM utilizada nos programas, permitindo ajustes de acordo com as necessidades específicas.

Task 4

Apresente curvas mostrando o tempo de execução para cada dimensão do problema e relacione estas curvas à complexidade computacional do produto matriz - vetor (O(n2))

Aplicando a mesma estratégia descrita na task 3, foram gerados os resultados a seguir ao considerar uma lista de valores no intervalo [0.1, 8), com incrementos de 0.3GB em cada execução:

Img 1: Time spent in executing the code in C.

Img 2: Time spent in executing the code in Fortran.

Cabe destacar que o valor máximo utilizado foi obtido de forma empírica, isto é, testando diferentes valores até atingir o limite em que o computador começa a travar ou o sistema operacional “mata” o código.

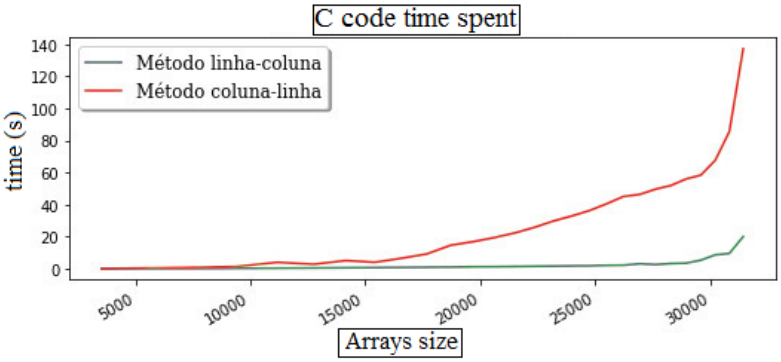
Task 5

Explique como o modo em que os arrays são armazenados nas duas linguagens afetam os resultados.

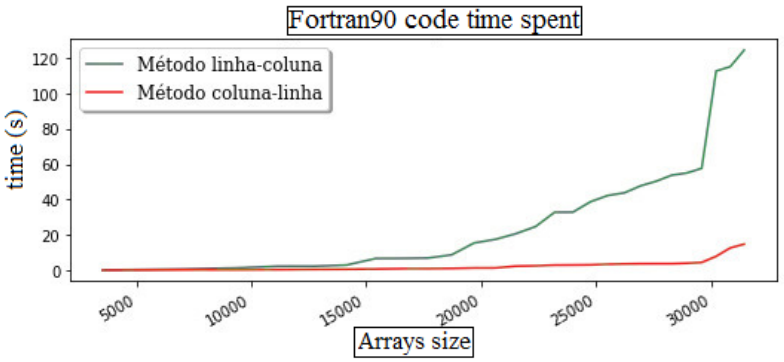
A linguagem Fortran e a linguagem C armazenam os dados de forma diferente. Enquanto o Fortran armazena os valores em colunas sequenciais na memória, permitindo um acesso mais rápido quando percorremos em loops do tipo "coluna-linha", o C armazena os valores das linhas sequencialmente na memória, o que possibilita um acesso mais eficiente quando percorremos em loops do tipo "linha-coluna". Essa diferença de armazenamento pode ter um impacto significativo no desempenho e eficiência do código, dependendo do tipo de operação realizada.

Img 3 e 4: Gráficos que utilizam a mesma quantidade de memória como base para comparação

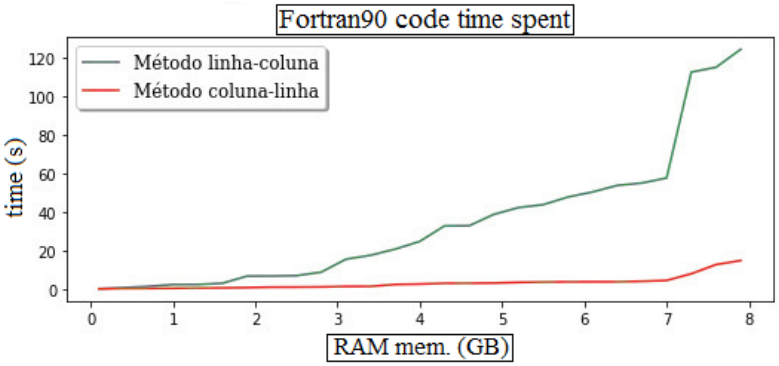
IMAGENS



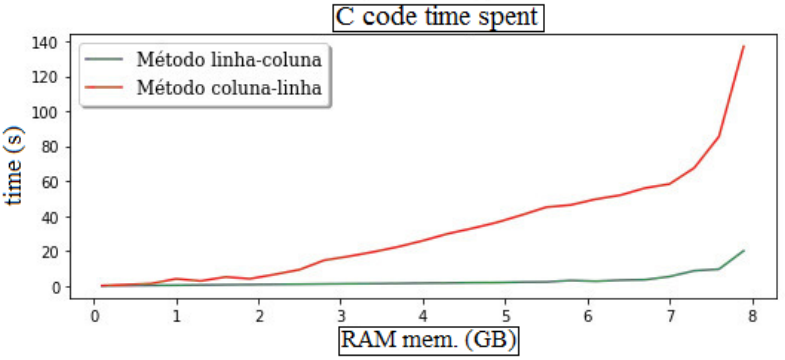
Img 1: Time spent in executing the code in C



Img 2: Time spent in executing the code in Fortran



Img 3: Time spent in executing the code in C



Img 4: Time spent in Executing the code in Fortran

CÓDIGOS

Código em C:

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

double matrix_lenght(double ramgb){

  double max = ramgb*pow(10,9)*8;
  double c = -1*max/64;
  double a, b, delta, x1, x2;
  a = 1.0;
  b = 2.0;
```

```
  delta = pow(b,2) - 4*a*c;
  if(delta >= 0){
    if(delta == 0){
      x1 = -b / (2 * a);
      return (int) floor(x1);
    }
    x1 = (-b - sqrt(delta)) / (2 * a);
    x2 = (-b + sqrt(delta)) / (2 * a);
    if(x1>x2){
    return (int) floor(x1);
    }
    return (int) floor(x2);

  }
  return 0;
}

double *limpaArray(int n) {
    double *vec = (double *)malloc(n * sizeof(double));
    for (int i = 0; i <= n; i++)
        vec[i] = 0;

    return vec;
}

double *multiply_matrix_v1(double **A, double *x, int n) {
    double *b = limpaArray(n);

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            b[i] += A[i][j] * x[j];
        }
    }

    return b;
}


double *multiply_matrix_v2(double **A, double *x, int n) {
    double *b = limpaArray(n);

    for (int j = 0; j < n; j++) {
        for (int i = 0; i < n; i++) {
            b[i] += A[i][j] * x[j];
```

```
        }
    }

    return b;
}


int main(int argc, char *argv[]) {
  system("clear");
  FILE *file = fopen("resultados.csv", "a");
  fprintf(file, "N, GB, t1, t2\n");
  printf("N, GB, t1, t2\n");
  for(int i=1;i<argc;++i){
    double ramgb = strtof(argv[i], NULL);

    int n = matrix_lenght(ramgb);

    srand(time(NULL));

    double **A = (double **)malloc(n * sizeof(double *));
    double *x = (double *)malloc(n * sizeof(double));
    double *b;
    double *b2;
    clock_t start1, finish1, start2, finish2;

    for (int i = 0; i < n; i++) {
      A[i] = (double *)malloc(n * sizeof(double));
      for (int j = 0; j < n; j++) {
        A[i][j] = (double)rand()/(double)(RAND_MAX);
      }
      x[i] = (double)rand()/(double)(RAND_MAX);
    };

    start1 = clock();
    b = multiply_matrix_v1(A, x, n);
    finish1 = clock();

    start2 = clock();
    b2 = multiply_matrix_v2(A, x, n);
    finish2 = clock();

    fprintf(file, "%d,%.2f, %.6f, %.6f\n", n, ramgb ,((double)(finish1 - start1))
/ CLOCKS_PER_SEC, ((double)(finish2 - start2)) / CLOCKS_PER_SEC);
    printf("%d,%.2f, %.6f, %.6f\n", n, ramgb ,((double)(finish1 - start1)) /
CLOCKS_PER_SEC, ((double)(finish2 - start2)) / CLOCKS_PER_SEC);
```

```
    free(x);
    free(b);
    free(b2);
    for (int i = 0; i < n; i++) {
      free(A[i]);
    }
  }
  return 0;
}
```

Código em Fortran:

```
PROGRAM Main
  use, intrinsic :: iso_fortran_env, only: dp=>real64
  implicit none
  real(dp)  :: u, i_b, f_b, i_b2, f_b2
  integer  :: i, j, count_, contador
  integer  ::  n
  real(dp), allocatable  :: A(:,:)
  real(dp), allocatable  ::  x(:), b(:), b2(:)
  CHARACTER(len=32)  ::  arg
  real(dp) :: ramgb_in

  print *, "N, GB, t1, t2"

  DO count_ = 1,30
    CALL get_command_argument(count_, arg)
    IF (LEN_TRIM(arg) == 0) EXIT
    read( arg, '(d10.0)' )  ramgb_in
      n = matrix_length(ramgb_in)
      allocate(A(n,n))
      allocate(x(n))
      allocate(b(n))
      allocate(b2(n))

    ! Matrix A
      do i = 1, n
        do j = 1, n
          call random_number(u)
          A(i,j) = u
        end do
      end do
```

```
      ! Vector x
      do i = 1, n
        call random_number(u)
        x(i) = u
      end do

    ! First method
      ! Vetor b
      b = 0.0
      call cpu_time(i_b)
      do i = 1, n
        do j = 1, n
          b(i) = b(i) + A(i,j) * x(j)
        end do
      end do
      call cpu_time(f_b)

    ! Second method
      ! Vector b2
      b2 = 0.0
      call cpu_time(i_b2)
      do i = 1, n
        do j = 1, n
          b2(j) = b2(j) + A(j,i) * x(i)
        end do
      end do
      call cpu_time(f_b2)

      print '(I0, ",", f10.6, ",", f10.6, ",", f10.6)', n, ramgb_in, f_b - i_b,
f_b2 - i_b2
      deallocate(A)
      deallocate(x)
      deallocate(b)
      deallocate(b2)
  END DO

contains

  real(dp) function matrix_length(ramgb)
    implicit none
    integer:: a, b, i
    real(dp) :: c, discriminant, x1, x2, ramgb

    a = 1
    b = 2
```

```
    matrix_length = 0

    c = ramgb*8
    c = c/64*(-1)
    c = c*1000000000

    discriminant = b**2 - 4*a*c

    if ( discriminant>0 ) then

      x1 = ( -b + sqrt(discriminant)) / (2 * a)
      x2 = ( -b - sqrt(discriminant)) / (2 * a)
      if (x1>x2) then
        matrix_length = x1
      else if (x2>x1) then
        matrix_length = x2
      end if
    else if ( discriminant==0 ) then
      x1 = - b / (2 * a)
      matrix_length = x1
    end if
  end function matrix_length

END PROGRAM Main
```