UNIVERSIDADE FEDERAL DA GRANDE DOURADOS
Faculdade de Ciências Exatas e Tecnologia - FACET

IGOR ZANATTA SARAIVA

Implementação do Algoritmo de Identificação de Números Primos em Matrizes Utilizando MultiThreading em C++.

Dourados - MS 2023

IGOR ZANATTA SARAIVA

Relatório do Trabalho 1 de Sistemas Operacionais: Programação em Threads

Relatório dos objetivos, funções e resultados obtidos no trabalho 1 a respeito de programação em Threads na disciplina de Sistemas Operacionais ministrada pelo professor Dr. Marcos Paulo Moro.

Dourados - MS 2023

SUMÁRIO

1. INTRODUÇÃO

O objetivo deste trabalho é desenvolver um programa em C++, com o intuito de analisar a performance da busca de números primos em submatrizes de uma matriz utilizando programação multithread. Para isso, foi desenvolvido um programa em C++ que preenche uma matriz com números aleatórios e realiza a busca de números primos em submatrizes utilizando threads.

Este relatório faz descrições textuais e visuais das funções e lógicas de programação implementadas no algoritmo. Seu objetivo é documentar o processo de desenvolvimento do projeto.

2. INFORMAÇÕES DO SISTEMA

O hardware usado para fazer os testes do algoritmo foram RAM 16 GB, um Intel(R) Core(TM) i7-8750H CPU, possuindo:
-Número de núcleos: 6
-Número de threads: 12
-Frequência base: 2,2 GHz
-Frequência turbo máxima: 4,1 GHz
-Cache: 9 MB Intel Smart Cache.

O Sistema Operacional usado foi o Windows 11 HOME, o código foi inscrito no Visual Studio Code e em linguagem C++.

3. CÓDIGO

O programa possui um menu interativo que permite definir o tamanho da matriz, a semente de geração dos números aleatórios, preencher a matriz com números aleatórios, definir o número de submatrizes, definir o número de threads e executar a busca de números primos.

Para a busca de números primos em submatrizes, o programa utiliza threads para dividir a tarefa em paralelo. Para isso, é necessário definir o número de submatrizes e o número de threads a serem utilizados.

O programa possui uma função para gerar números aleatórios, uma função para verificar se um número é primo e uma função para preencher a matriz com números aleatórios.

Para analisar a performance do programa, foram realizados testes com diferentes tamanhos de matriz, sementes de geração de números aleatórios, números de submatrizes e números de threads.

3.1 MENU

Esta função é responsável pela interação do usuário com o programa. É exibido um menu com opções numeradas, que permitem ao usuário definir o tamanho da matriz, semente de geração de números aleatórios, preencher a matriz com números aleatórios, definir o número de submatrizes, definir o número de threads, executar o programa, visualizar o tempo de execução e quantidade de números primos, e encerrar o programa.

```
53  //menu de interacao
54  void menu(){
55      int menuint = 0, numThreads = 1, testevalida = -1, executado = 0;
56
57      do {
58          printf("-----------------------------------\n");
59          printf("---------------Menu----------------\n");
60          printf("-----------------------------------\n");
61          printf("1 - Defina o tamanho da matriz -\n");
62          printf("2 - Defina a semente de criacao da matriz - \n");
63          printf("3 - Preencher a matriz com numeros aleatorios -\n");
64          printf("4 - Definir o numero de Submatrizes -\n");
65          printf("5 - Definir o numero de Threads -\n");
66          printf("6 - Executar -\n");
67          printf("7 - Visualizar o tempo de execucao e quantidade de numeros primos -\n");
68          printf("8 - Encerrar -\n");
69          scanf("%d", &menuint);
70          printf("\n");
71
72          switch (menuint){
```

Ao iniciar a função, são definidas algumas variáveis, incluindo a variável "menuint", que é utilizada para armazenar a opção selecionada pelo usuário no menu.

Em seguida, um loop do-while é utilizado para exibir o menu e aguardar a entrada do usuário até que a opção de encerramento seja selecionada.

Dentro do loop, um switch-case é utilizado para tratar a opção selecionada pelo usuário.

A variável "testevalida" é utilizada para verificar se a matriz foi preenchida com números aleatórios antes de executar a opção de processamento. A variável "executado" é utilizada para verificar se o programa já foi executado antes de exibir o tempo de execução e a quantidade de números primos encontrados.

3.2 GERADOR DE NÚMEROS ALEATÓRIOS

```
151  void GeradorNum(long long int lines, long long int columns, int sementematriz){
152      // Inicializa o gerador de números aleatórios com a semente
153      std::srand(sementematriz);
154      // Gera números aleatórios entre 0 e 99999999 e os armazena na matriz
155      for(int i = 0; i < lines; i++){
156          for(int j = 0; j < columns; j++){
157              int random_number = std::rand() % 100000000;
158              matriz[i][j] = random_number;
159          }
160      }
161
162  }
```

Essa função é responsável por preencher uma matriz com números aleatórios gerados a partir de uma semente. Ela recebe como parâmetros o número de linhas (lines) e o número de colunas (columns) da matriz, bem como a semente (sementematriz) que será utilizada para a geração dos números aleatórios.

A função começa inicializando o gerador de números aleatórios com a semente fornecida. Em seguida, ela utiliza um loop for aninhado para percorrer cada posição da matriz e gerar um número aleatório entre 0 e 99999999 usando a função std::rand(). Esse número aleatório é armazenado na posição correspondente da matriz.

Ao final da execução da função, a matriz terá sido preenchida com números aleatórios gerados a partir da semente fornecida.

3.3 VALIDAR OS PRIMOS

```
162  //Mostra se o Numero e ou nao Primo
163  int isPrime(int number) {
164      if (number <= 1) {
165          return false;
166      }
167      for (int i = 2; i <= number/2; i++) {
168          if (number % i == 0) {
169              return false;
170          }
171      }
172      return true;
173  }
```

Essa função, chamada "isPrime", tem como objetivo verificar se um número é primo ou não. Ela recebe um inteiro "number" como argumento e retorna um valor booleano indicando se o número é ou não primo.

Primeiramente, a função verifica se o número é menor ou igual a 1, pois 1 não é considerado um número primo. Se o número for menor ou igual a 1, a função retorna "false" indicando que não é primo.

Caso o número seja maior que 1, a função itera um laço "for" que vai de 2 até a metade do valor do número. Dentro desse laço, a função verifica se o número é divisível por "i" (sendo "i" um valor de 2 até a metade do número). Se o número for divisível por "i", então ele não é primo e a função retorna "false".

Caso o laço termine e nenhum divisor tenha sido encontrado, a função retorna "true", indicando que o número é primo.

3.4 SUB MATRIZES

```
188  void preencherParametros(int numberThreads, long long int lines, long long int columns, long long int dividir) {
189      PARAMETRO addDataMatriz;
190      int totalNumbers = 0, totalEachThread = 0, firstLine = 0, lastLine = 0 , firstColumn = 0, lastColumn = 0, decider = 0;
191
192      totalNumbers = lines * columns;
193      totalEachThread = totalNumbers / dividir;
194      lastLine = totalEachThread;
195      lastColumn = totalEachThread;
196
197      if (lines >= columns) {
198          if ((lines * columns) % dividir == 0) {
199              for (int i = 0; i < dividir; i++) {
200                  addDataMatriz.id = i;
201                  addDataMatriz.startColumn = firstColumn;
202                  addDataMatriz.startLine = firstLine;
203                  addDataMatriz.endLine = lastLine;
204                  vetorParametro.push_back(addDataMatriz);
205
206                  for (int j = firstLine; j < lastLine; j++) {
207                      if (j % lines == 0 && j != 0) firstColumn++;
208                  }
209
210                  firstLine = lastLine;
211                  lastLine = lastLine + totalEachThread;
212              }
213          }
214          else {
215              for (int i = 0; i < dividir; i++) {
216                  addDataMatriz.id == i;
217                  if (i == (dividir - 1)) {
218                      addDataMatriz.startColumn = firstColumn;
```

Essa função preenche um vetor de estruturas PARAMETRO com informações de como dividir uma matriz em partes iguais para serem processadas em paralelo por threads.
Os parâmetros de entrada são:
- numberThreads: número de threads a serem criadas.
- lines: quantidade de linhas da matriz.
- columns: quantidade de colunas da matriz.
- dividir: quantidade de partes em que a matriz será dividida.

A função calcula o total de elementos na matriz e o total de elementos que cada thread deve processar. Em seguida, a matriz é dividida em partes iguais, sendo que cada parte é

identificada por um ID único, que é adicionado a uma estrutura PARAMETRO juntamente com as informações de onde começa e onde termina a parte correspondente na matriz.

A variável decider é usada para indicar se a divisão deve ser feita por linhas (decider = 1) ou por colunas (decider = 2).

Após o preenchimento do vetor de estruturas PARAMETRO, a função chama a função createThreads para criar as threads e processar cada parte da matriz em paralelo.

Por fim, a função chama esvaziarParametros para limpar o vetor de estruturas PARAMETRO.

3.5 ESVAZIAR PARÂMETROS

```
290  void esvaziarParametros() {
291      vetorParametro.swap(vetorParametro2);
292      vetorParametro2.clear();
293  }
```

A função esvaziarParametros() é responsável por limpar o vetor de parâmetros vetorParametro, transferindo seu conteúdo para um segundo vetor vetorParametro2 e então limpando o primeiro vetor.

Essa função é útil para preparar o programa para receber novos parâmetros sem manter os dados antigos do vetor.

3.6 ESVAZIA MATRIZ

```
176  void esvaziarMatriz() {
177      matriz.swap(matriz2);
178      matriz2.clear();
179      idThread.swap(idThread2);
180      idThread2.clear();
181
182      esvaziarParametros();
183  }
```

Essa função tem como objetivo esvaziar a matriz e os vetores de threads e parâmetros utilizados para realizar o processamento em paralelo.

A matriz original é substituída pela matriz2, que é uma matriz vazia. Além disso, o vetor de threads idThread também é substituído pelo vetor idThread2 vazio.

Em seguida, a função chama a função esvaziarParametros(), que foi descrita anteriormente e tem como objetivo esvaziar o vetor de parâmetros utilizado na criação das threads.

Dessa forma, a função esvaziarMatriz() garante que todas as estruturas utilizadas no processamento em paralelo sejam esvaziadas, permitindo que uma nova análise possa ser realizada sem interferências de dados antigos.

3.7 EXECUTANDO THREADS

```
334  void thread(void* matrizParametros){
335      PARAMETRO* dataMatriz;
336      dataMatriz = (PARAMETRO*)matrizParametros;
337
338      int travessiaLinha = 0, contadorPrimos = 0, travessiaColunas = 0;
339
340      travessiaColunas = dataMatriz->startColumn;
341      travessiaLinha = dataMatriz->startLine % lines;
342
343      for (int i = dataMatriz->startLine; i < dataMatriz->endLine; i++) {
344          if (i % lines == 0 && i != 0){ travessiaLinha = 0;}
345
346          if (travessiaLinha % lines == 0 && i != 0){travessiaColunas++;}
347
348          if (isPrime(matriz[travessiaLinha][travessiaColunas])){ contadorPrimos++;}
349
350          travessiaLinha++;
351      }
352
353      WaitForSingleObject(secaoCritica, INFINITE);
354
355      totalPrimos += contadorPrimos;
356
357      ReleaseMutex(secaoCritica);
358
359      _endthread();
360  }
```

```
362  void thread2(void* matrizParametros) {
363      PARAMETRO* dataMatriz;
364      dataMatriz = (PARAMETRO*)matrizParametros;
365
366      int travessiaLinha = 0, contadorPrimos = 0, travessiaColunas = 0;
367
368      travessiaColunas = dataMatriz->startColumn % columns;
369      travessiaLinha = dataMatriz->startLine;
370
371      for (int i = dataMatriz->startColumn; i < dataMatriz->endColumn; i++) {
372          if (i % columns == 0 && i != 0){ travessiaColunas = 0;}
373
374          if (travessiaColunas % columns == 0 && i != 0){travessiaLinha++;}
375
376          if (isPrime(matriz[travessiaLinha][travessiaColunas])) {contadorPrimos++;}
377
378          travessiaColunas++;
379
380      }
381
382      WaitForSingleObject(secaoCritica, INFINITE);
383
384      totalPrimos += contadorPrimos;
385
386      ReleaseMutex(secaoCritica);
387
388      _endthread();
389  }
```

Essa é uma função que será executada em uma thread e recebe um ponteiro para um objeto do tipo PARAMETRO. A função então realiza a contagem dos números primos em uma submatriz da matriz global "matriz", usando os limites especificados nos campos "startColumn", "endColumn", "startLine" e "endLine" do objeto PARAMETRO.

A função começa inicializando algumas variáveis, como as variáveis de contagem "contadorPrimos" e as variáveis de controle "travessiaLinha" e "travessiaColunas". Em seguida, ela entra em um loop “for” que percorre a submatriz especificada pelo objeto PARAMETRO. A cada iteração do loop, a função verifica se a posição atual contém um número primo usando a função "isPrime". Se a posição contiver um número primo, o contador de primos é incrementado.

Depois que o loop termina, a função bloqueia o objeto mutex "secaoCritica" para garantir a exclusão mútua ao atualizar a variável global "totalPrimos" com a contagem de primos da submatriz. Em seguida, ela libera o mutex e encerra a thread usando a função "_endthread()".

3.8 CRIANDO AS THREADS

```
293  void createThreads(int numberThreads, int switchThread, long long int dividir) {
294      if (switchThread == 1) {
295          for (int i = 0; i < dividir; i++) {
296              idThread.push_back(CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)thread, &vetorParametro[i], CREATE_SUSPENDED, NULL));
297          }
298          inicio = time(0);
299          for (int i = 0; i < numberThreads; i++) {
300              for (int j = 0; j < dividir; j++)
301              {
302                  ResumeThread(idThread[j]);
303              }
304          }
305
306          WaitForMultipleObjects(numberThreads, idThread.data(), TRUE, INFINITE);
307          fim = time(0);
308
309      }
310      else {
311          for (int i = 0; i < dividir; i++) {
312              idThread.push_back(CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)thread2, &vetorParametro[i], CREATE_SUSPENDED, NULL));
313          }
314          inicio = time(0);
315          for (int i = 0; i < numberThreads; i++) {
316              for (int j = 0; j < dividir; j++)
317              {
318                  ResumeThread(idThread[j]);
319              }
320          }
321          WaitForMultipleObjects(numberThreads, idThread.data(), TRUE, INFINITE);
322          fim = time(0);
323      }
324      tempoDeExecucao = fim - inicio;
325  }
```

A função createThreads cria e executa threads para processar uma matriz de inteiros em busca de números primos, utilizando os parâmetros especificados.

Os parâmetros numberThreads, switchThread e dividir indicam, respectivamente, o número de threads a serem criadas, qual função de thread será usada (1 para thread e 2 para thread2) e quantas partes a matriz será dividida para a execução paralela.

A função começa criando um vetor de identificadores de threads idThread e, dependendo do valor de switchThread, chama a função thread ou thread2 para cada parte da matriz especificada pelos parâmetros em vetorParametro.

Cada thread é criada com CreateThread, configurada para iniciar em modo suspenso com a flag CREATE_SUSPENDED, e seu identificador é adicionado ao vetor idThread. Em seguida, a função inicia as threads com ResumeThread.

Após o início das threads, a função aguarda o término de todas elas com WaitForMultipleObjects, passando o número de threads, o vetor de identificadores idThread e o valor TRUE para indicar que todas as threads devem terminar antes que a função prossiga.

Ao final, a função calcula o tempo total de execução em segundos, subtraindo o tempo inicial de startTime do tempo final de endTime. O resultado é armazenado em totalTime.

4. TESTES

Para analisar a performance do programa, foram realizados testes com diferentes tamanhos de matriz, sementes de geração de números aleatórios, números de submatrizes e números de threads.

Durante os diversos testes foram observados alguns detalhes sobre o uso de threads, e na diferença de um algoritmo sequencial, como por exemplo se exagerar na criação de threads as mesmas podem causar erro no resultado final do aplicativo, já outro erro é que se criar matrizes muito grande pode acontecer do código sequencial nem funcionar, mas os de thread funcionam ocorrendo erro apenas quando se tentava criar um número muito grande normalmente acima de cem submatrizes.

Já em um contexto geral da diferenciação dos dois códigos é que o multithreads se tem uma vantagem significativa sobre o sequencial quando se necessita da manipulação de muitos dados, como por exemplo matrizes gigantes como [10000]x[10000], já que o tempo de processamento normalmente demora metade do tempo que demoraria. Porém em uma quantidade de dados menores não se vê diferença entre um algoritmo sequencial e um algoritmo multi-threaded.

Na realização dos testes foi mantido como base a mesma semente, pois não se tinha uma diferença em mudar a semente toda hora e manter o número de linhas e colunas da matriz. Foram realizados diversos testes com tamanhos diferentes de matrizes, porém em tamanhos muito pequenos o tempo era insignificante, beirando zero, então foi utilizado os números grandes para se ter uma boa comparação.

4.1 TESTE 1

No teste 1, foi utilizado uma matriz de tamanho [2.500]x[2.500], e a semente de geração foi 19. Segue abaixo os resultados:

Resultado 1:
. Tempo: 14.
. Primos: 671139.
. Número de Threads: Sequencial.
. Número de Submatrizes: 100

Resultado 2:
. Tempo: 14.
. Primos: 671139.
. Número de Threads: Sequencial.
. Número de Submatrizes: 1

Resultado 3:
. Tempo: 14.
. Primos: 671139.
. Número de Threads: Sequencial.
. Número de Submatrizes: 10

Resultado 4:
. Tempo: 3.
. Primos: 671139.
. Número de Threads: 1.
. Número de Submatrizes: 100

Resultado 5:
. Tempo: 2.
. Primos: 671139.
. Número de Threads: 10.
. Número de Submatrizes: 100

Resultado 6:
. Tempo: 2.
. Primos: 671139.
. Número de Threads: 1.
. Número de Submatrizes: 50.

Resultado 7:
. Tempo: 3.
. Primos: 671139.
. Número de Threads: 4.
. Número de Submatrizes: 100

Resultado 8: ERRO
. Tempo: 2.
. Primos: 38290.
. Número de Threads: 100.
. Número de Submatrizes: 100

4.2 TESTE 2

No teste 2, foi utilizado uma matriz de tamanho [5.000]x[5.000], e a semente de geração foi 910. Segue abaixo os resultados:

Resultado 1:
. Tempo: 59.
. Primos: 2677619.
. Número de Threads: Sequencial.
. Número de Submatrizes: 100

Resultado 2:
. Tempo: 60.
. Primos: 2677619.
. Número de Threads: Sequencial.
. Número de Submatrizes: 1000

Resultado 3:
. Tempo: 11.
. Primos: 2677619.
. Número de Threads: 10.
. Número de Submatrizes: 100

Resultado 4:
. Tempo: 10.
. Primos: 2677619.
. Número de Threads: 1.
. Número de Submatrizes: 100

Resultado 5:
. Tempo: 11.
. Primos: 2677619.
. Número de Threads: 10.
. Número de Submatrizes: 50

Resultado 6:
. Tempo: 59.
. Primos: 2677619.
. Número de Threads: 1.
. Número de Submatrizes: 1

4.3 TESTE 3

No teste 3, foi utilizado uma matriz de tamanho [10.000]x[10.000], e a semente de geração foi 101. Segue abaixo os resultados:

Resultado 1:
. Tempo: 235.
. Primos: 10719248.
. Número de Threads: Sequencial.
. Número de Submatrizes: 100

Resultado 2:
. Tempo: 235.
. Primos: 10719248.
. Número de Threads: Sequencial.
. Número de Submatrizes: 1

Resultado 3:
. Tempo: 247.
. Primos: 10719248.
. Número de Threads:1.
. Número de Submatrizes: 1

Resultado 4:
. Tempo: 45.
. Primos: 10719248.
. Número de Threads: 10.
. Número de Submatrizes: 100

Resultado 5:
. Tempo: 45.
. Primos: 10719248.
. Número de Threads: 5.
. Número de Submatrizes: 25

Resultado 6:
. Tempo: 44.
. Primos: 10719248.
. Número de Threads: 1.
. Número de Submatrizes: 27

Resultado 7: ERRO
. Tempo: 19.
. Primos: 5840978.
. Número de Threads: 1.
. Número de Submatrizes: 1000

Resultado 8: ERRO
. Tempo: 1.
. Primos: 4480891.
. Número de Threads: 1000.
. Número de Submatrizes: 1000

4.4 TESTE 4

No teste 4, foi utilizado uma matriz de tamanho [15.000]x[15.000], e a semente de geração foi 37. Segue abaixo os resultados:

Resultado 1: ERRO
. Tempo: 0.
. Primos: 0.
. Número de Threads: Sequencial.
. Número de Submatrizes: 100

(A matriz desse tamanho não executava no sequencial, acima desse tamanho nem compilar o código.)

Resultado 2:
. Tempo: 100.
. Primos: 24116116.
. Número de Threads:10.
. Número de Submatrizes: 100

Resultado 3: ERRO
. Tempo: 90.
. Primos: 7958756.
. Número de Threads:1.
. Número de Submatrizes: 100

Resultado 4:
. Tempo: 99.
. Primos: 24116116.
. Número de Threads:6.

. Número de Submatrizes: 100
Resultado 5: ERRO
. Tempo: 63.
. Primos: 15314165.
. Número de Threads: 20.
. Número de Submatrizes: 200

Resultado 6:
. Tempo: 98.
. Primos: 24116116.
. Número de Threads: 8.
. Número de Submatrizes: 50

Resultado 7:
. Tempo: 101.
. Primos: 24116116.
. Número de Threads: 4.
. Número de Submatrizes: 100

5. CONCLUSÃO

Os resultados obtidos mostram que a utilização de threads para a busca de números primos em submatrizes de uma matriz aumenta significativamente a performance do programa. Além disso, o número de submatrizes e o número de threads utilizados afetam diretamente a performance do programa.

Em resumo, este relatório apresentou uma análise de performance da busca de números primos em submatrizes de uma matriz utilizando programação multithread. Os resultados obtidos mostraram que a utilização de threads aumentou significativamente a performance do programa, e que o número de submatrizes e o número de threads utilizados afetam diretamente a performance do programa.