UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO

SSC0143
Programação Concorrente - Turma B

Palíndromos

Professor Dr. Julio Estrella

Grupo 06:	*Números USP:*
Bruno Junqueira Adami	6878762
Lucas Junqueira Adami	6792496
Lucas Lobosque	6792645

6 de maio de 2012

1 Introdução

O problema proposto é o de devesenvolver versões de um programa paralelo para realizar as tarefas de determinar a ocorrência de palíndromos em dois textos especificados. Além disso, no texto maior, uma vez encontrado o palíndromo, é preciso determinar se a soma dos números correspondentes ao mapeamento do código ASCII de cada caracter da palavra é um número primo. Para calcular se o número é primo, o algoritmo de crivo de Erastótenes deve ser utilizado. As bibliotecas OpenMP e OpenMPI foram utilizadas para realizar o trabalho paralelo.

Para realizar o desenvolvimento da proposta, o projeto foi separado em três partes:

- Execução do algoritmo do crivo.
- Leitura dos arquivos de entrada.
- Cálculo dos palíndromos.

Todas as execuções citadas no documento, responsáveis pela geração dos gráficos, foram executados no cluster disponível.

2 O projeto OpenMP

Neste projeto, a paralelização do código foi feita nos loops do programa através das chamadas dos macros da biblioteca. O programa segue um fluxo contínuo e possui blocos de código executados em paralelo. Inicialmente, o cálculo do crivo é feito. Após esse passo, os arquivos são lidos e as palavras lidas são entregues ao verificador de palíndromos. Foi uma decisão de projeto em separar as execuções do arquivo pequeno e o grande. Isso significa que em cada execução, ou o arquivo menor ou o maior é processado. Os resultados dos testes apresentam-se na figura 1 e figura 2.

2.1 Execução do parser

O parser foi dividido em duas funções (sendo cada uma chamada em uma execução): uma determina a ocorrência de palíndromo apenas nas palavras (texto maior) e outra nas palavras e também nas frases (texto menor). Em ambos os casos, apenas caracteres alfanuméricos são considerados (a-z, A-Z, 0-9). Esta condição se fez necessária pois haviam vários caracteres nos textos que não são representados na tabela ASCII. No caso das frases, os espaços também são removidos.

2.2 Cálculo dos palíndromos

O algoritmo de cálculo dos palíndromos é simples. Sua complexidade é $O(n)$, pois baseia-se em apenas um loop para verificar a palavra. Além disso, ele já soma os valores ASCII dos caracteres para responder se a soma total é um número primo. Para a paralelização do algoritmo, o macro citado no código 1 foi utilizado. Neste passo, a decomposição de dados foi utilizada. O vetor que contém a palavra é dividido em sessões que serão computadas pela OpenMP

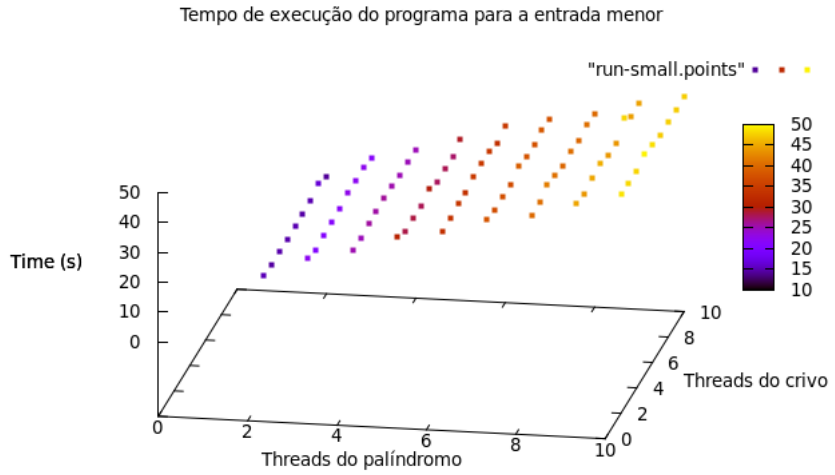

Figura 1: Resultados da execução do projeto MPI para o arquivo menor

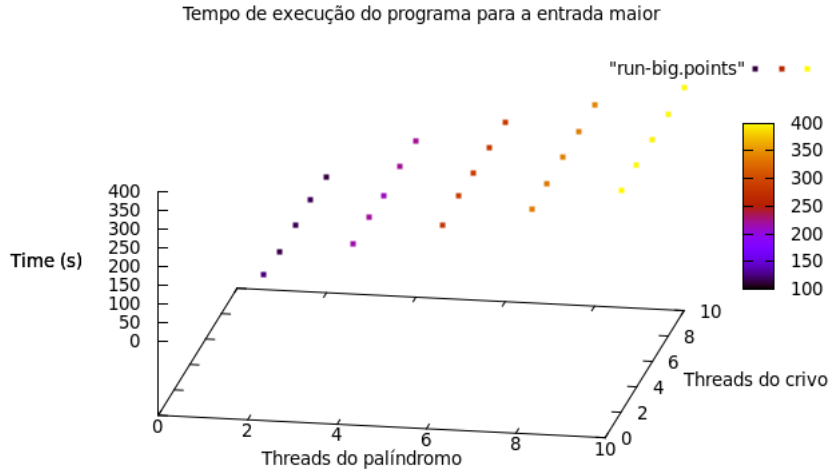

Figura 2: Resultados da execução do projeto MPI para o arquivo maior

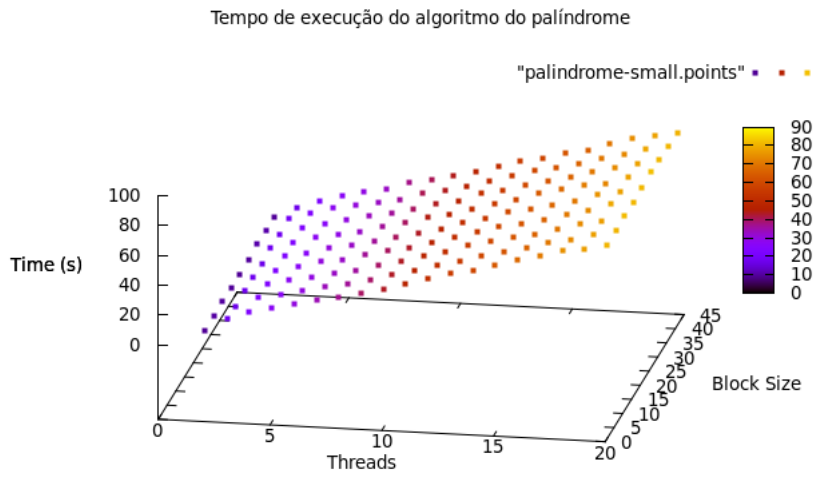

Figura 3: Resultados do algoritmo do palíndromo para o arquivo menor

através da chamada do macro. Cada partição executa o loop com suas iterações correspondentes em paralelo.

```
1 #pragma omp parallel for num_threads(PALINDROME_N_THREADS)
2         schedule(dynamic, PALINDROME_BLOCK_SIZE) reduction(+:sum)
3         reduction(&&:palindrome)
```

Código 1: Macro que paraleliza o algoritmo do palíndromo

As definições PALINDROME_N_THREADS e PALINDROME_BLOCK_SIZE são passadas ao programa através do arquivo makefile. A primeira diz quantas threads serão utilizadas na paralelização do loop. A segunda, quantas iterações cada thread irá realizar. A palavra dynamic define que não haverá uma ordem na distribuição das iterações para os loops. Neste macro também estão definidas as reduções da soma para verificar a primalidade e da validade do palíndromo.

Alguns testes foram aplicados isoladamente do resto do sistema para testar a performance do algoritmo e suas diferentes configurações. Uma leitura cega foi feita dos dois arquivos de entrada e os resultados obtidos estão presentes na figura 3 e na figura 4.

2.3 Cálculo do crivo

O crivo de Eratóstenes é um algoritmo e um método simples e prático para encontrar números primos até um certo valor limite. Segundo a tradição, foi criado pelo matemático grego Eratóstenes (c. 285-194 a.C.), o terceiro bibliotecário-chefe da biblioteca de Alexandria.

O algoritmo se baseia em marcar os múltiplos dos primos encontrados até agora. Os números não marcados serão os números primos. Deve-se iterar do número 2 em diante até a raíz do maior número que se deseja calcular. A complexidade do algoritmo é $O(n \log(\log n))$.

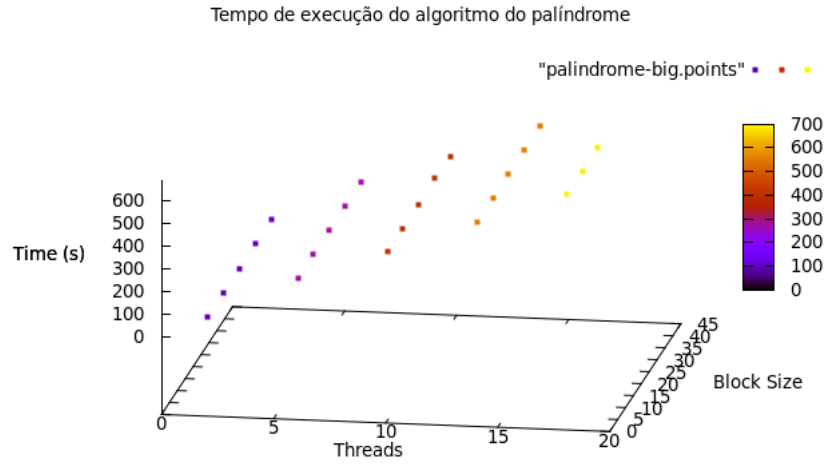

Figura 4: Resultados do algoritmo do palíndromo para o arquivo maior

Na implementação deste trabalho os números pares são ignorados para otimizar um pouco o método. A função isPrime testa se o número par é primo ou não. No caso, apenas o 2 será. Se o número for ímpar, esta função irá checar o vetor de marcação dos múltiplos.

A decomposição utilizada foi a de dados, pois foi paralelizada a demarcação dos mútiplos dos números primos, onde cada thread fica responsável por um trecho do vetor dos múltiplos. Para isso usou-se o código 2, que separa o for em vários blocos. O número de blocos é determinado pelo macro SIEVE_N_THREADS. A implementação em OpenMP paraleliza a parte que faz a marcação dos múltiplos do primo encontrado. No fim para saber se o número é primo ou não, basta apenas checar o vetor de marcação dos múltiplos.

```
1 #pragma omp parallel for num_threads(SIEVE_N_THREADS)
```

Código 2: Macro que paraleliza o algoritmo do palíndromo

Testes de performance foram aplicados isoladamente do resto do programa. Os resultados obtidos encontram-se na figura 5.

3 O projeto MPI

O projeto MPI foi desenvolvido seguindo a estrutura da figura 6. Todos os processos do programa têm suas chamadas principais executadas no início do programa. Neste projeto, a leitura dos arquivos pequeno e grande são feitas em conjunto, diferentemente do projeto anterior. Existem três tipos de processos:

- Processos mestres principais.
- Processos mestres secundários.

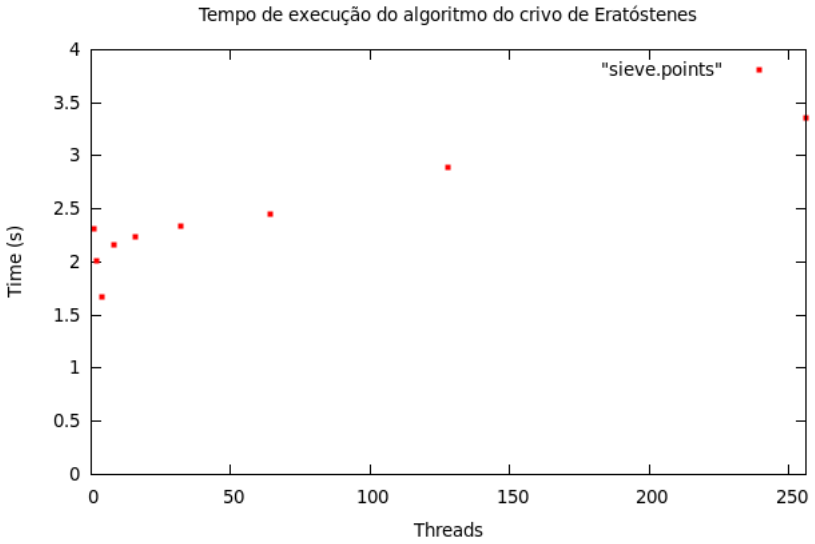

Figura 5: Tempos de execução do crivo usando OpenMP

- Processos escravos.

Há somente um processo mestre principal. O mestre principal é responsável por controlar os processos do segundo tipo, os processos mestres secundários. Ele envia mensagens de finalização para esses processos e recebe informações de término ao final da execução deles. Neste projeto, todo o sistema foi testado em conjunto. Várias configurações de números de escravos foram utilizadas. Os testes foram aplicados sem a leitura do arquivo maior, pois a leitura desse arquivo demorava muito. Os resultados obtidos encontram-se na figura 7.

3.1 Execução do parser

O parser segue as mesmas regras descritas no projeto do OpenMP. Porém, neste caso as leituras dos 2 arquivos são feitas paralelamente, e os dados são coletados pela leitura dos arquivos (small.in e big.in) e não pela entrada padrão.

3.2 Cálculo dos palíndromos

O algoritmo para o cálculo é o mesmo utilizado para o projeto OpenMP. Há um processo mestre que coordena os processos escravos. Ao receber uma palavra, o processo mestre envia a mensagem ao próximo escravo da fila (uma fila round-robin) e o escravo executa o algoritmo do palíndromo, enviando ao mestre o resultado. Aqui, a decomposição pode ser vista como uma decomposição pseudo-especulativa. Na verdade, cada requisição pode ser executada paralelamente, independentemente de qualquer fator, funcionando como filas paralelas de execução de trabalho. A tomada de decisão, que é característica dessa decomposição não existe. O mestre apenas tem que saber qual o próximo escravo a receber uma palavra.

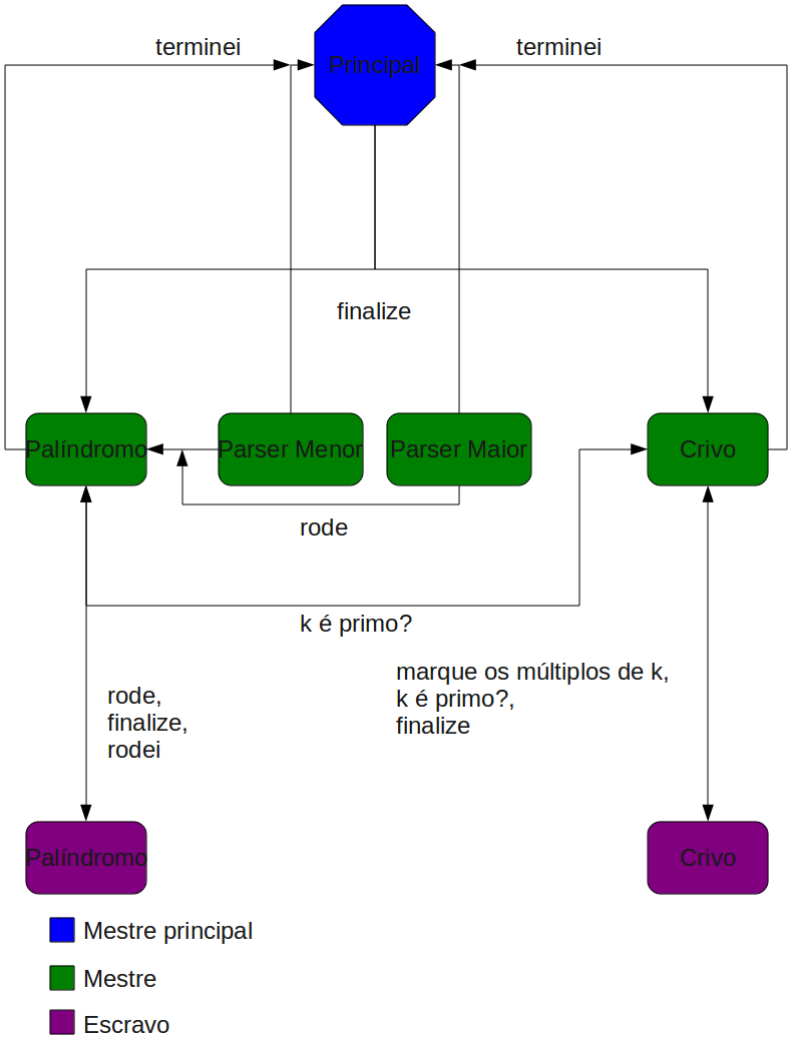

Figura 6: A estrutura do projeto MPI

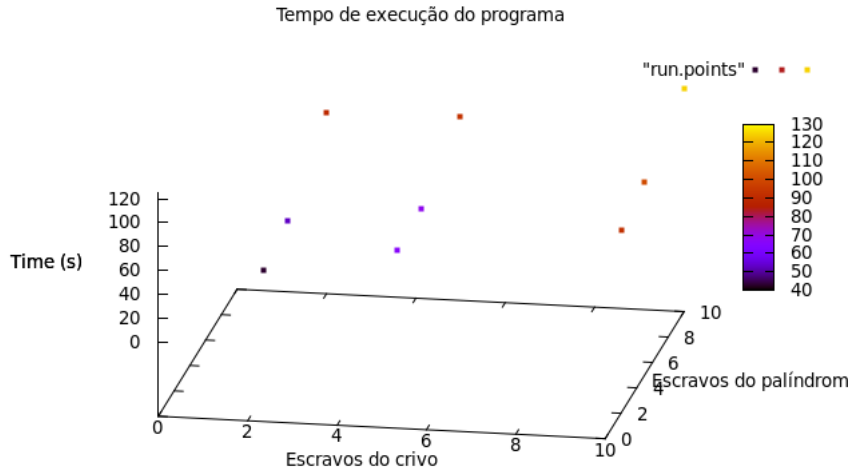

Figura 7: Resultados da execução do projeto MPI

Tabela 1: Resultados das execuções

Medida	OpenMP - Arquivo maior	OpenMP - Arquivo menor	MPI
Palíndromos	45475	302389	347864
Primos	0	190327	190327

3.3 Cálculo do crivo

O algoritmo para o cálculo é análogo para o projeto OpenMP. O cálculo aqui também paraleliza a marcação dos mútiplos dos primos. O processo meste itera os números de 2 até a raiz do maior número e requisita ao processos escravos que demarquem os múltiplos dos primos. Para encontrar se um número é primo, o processo mestre faz uma requisição ao escravo que contém o intervalo do número requisitado. O escravo então responde ao processo mestre que responde ao processo que fez a pergunta se o número é primo ou não. A decomposição é a de dados, pelas mesmas razões citadas no projeto OpenMP.

Para o crivo, testes de performance foram aplicados isoladamente do resto do programa. Os tempos obtidos encontram-se na figura 8.

4 Considerações finais

A execução do projeto MPI no cluster falhou. Os erros mostrados estão presentes no código 3. A causa desses erros não foi pesquisada a fundo. Para driblar esse problema, foi utilizada uma máquina local, com 2 CPUs e 4 GB de memória RAM.

No final das execuções, a quantidade de palíndromos e números primos, de acordo com o parser criado para os projetos, foram coletadas e estão mostradas na tabela 1.

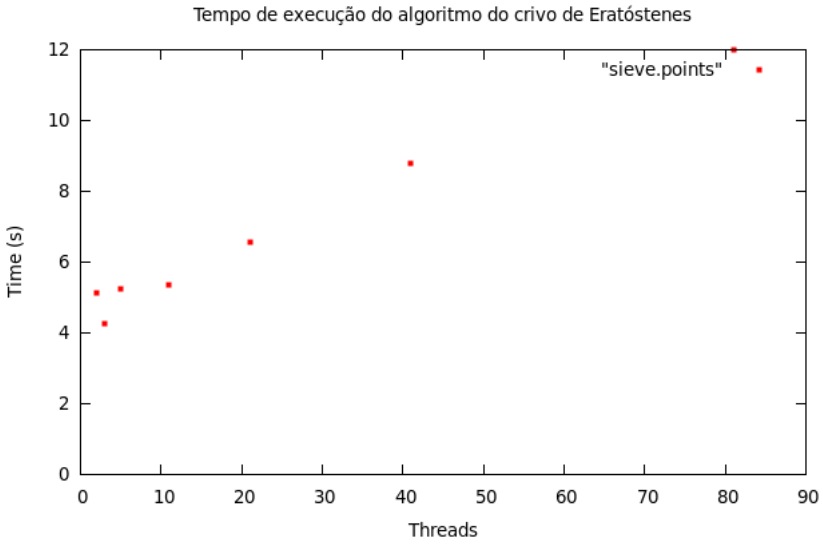

Figura 8: Tempos de execução do crivo usando MPI

```
1  mpirun noticed that job rank 0 with PID 21811 on node x-men exited on signal 15 (Terminated).
2  8 additional processes aborted (not shown)
3
4  mpirun noticed that job rank 0 with PID 21888 on node x-men exited on signal 15 (Terminated).
5  10 additional processes aborted (not shown)
6
7  [x-men][0,1,1][btl_sm_component.c:521:mca_btl_sm_component_progress]
8          SM faild to send message due to shortage of shared memory.
9
10 [x-men][0,1,5][btl_sm_component.c:521:mca_btl_sm_component_progress]
11         SM faild to send message due to shortage of shared memory.
12
13 [x-men][0,1,6][btl_sm_component.c:521:mca_btl_sm_component_progress]
14         SM faild to send message due to shortage of shared memory.
15
16 [x-men][0,1,7][btl_sm_component.c:521:mca_btl_sm_component_progress]
17         SM faild to send message due to shortage of shared memory.
18
19 [x-men][0,1,10][btl_sm_component.c:521:mca_btl_sm_component_progress]
20         SM faild to send message due to shortage of shared memory.
21
22 [x-men:21931] [0,0,0] ORTE_ERROR_LOG: Timeout in file base/pls_base_orted_cmds.c at line 275
23 [x-men:21931] [0,0,0] ORTE_ERROR_LOG: Timeout in file pls_rsh_module.c at line 1155
24 [x-men:21931] [0,0,0] ORTE_ERROR_LOG: Timeout in file errmgr_hnp.c at line 90
```

Código 3: Erros do MPI no cluster

Referências

[OpenMP] http://bisqwit.iki.fi/story/howto/openmp/

[OpenMP] http://openmp.org/wp/

[Gnuplot] http://www.duke.edu/ hpgavin/gnuplot.html

[Crivo de Eratóstenes] http://en.wikipedia.org/wiki/Sieve_of_Eratosthenes

[Crivo de Eratóstenes] http://www.algorithmist.com/index.php/Prime_Sieve_of_Eratosthenes

[MPI] http://www.slac.stanford.edu/comp/unix/farm/mpi.html

[MPI] http://www.eecis.udel.edu/~saunders/courses/372/01f/manual/manual.html