

UNIVERSIDADE DE SÃO PAULO  
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE  
COMPUTAÇÃO

SSC0143  
PROGRAMAÇÃO CONCORRENTE - TURMA B

---

## Palíndromos

---

PROFESSOR DR. JULIO ESTRELLA

*Grupo 06:*

Bruno Junqueira Adami  
Lucas Junqueira Adami  
Lucas Lobosque

*Números USP:*

6878762  
6792496  
6792645

1 de maio de 2012

# 1 Introdução

O problema proposto é o de desenvolver 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. A execução do programa e seus resultados apresentam-se na figura 1.

### 2.1 Execução do parser

TODO.

### 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 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

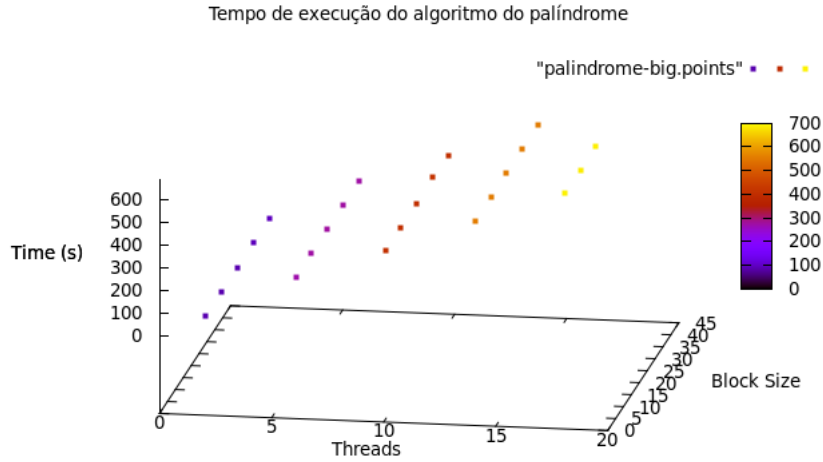


Figura 1: Resultados da execução do projeto MPI

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 2 e na figura 3. Os valores dos números de threads e o tamanho do bloco que cada thread executa foram baseados em valores estimados. Um cálculo do tamanho médio das palavras provenientes da leitura cega apontou que para o arquivo menor, essa média valia 40 e para o maior, 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 raiz do maior número que se deseja calcular. A complexidade do algoritmo é  $O(n \log(\log n))$ .

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.

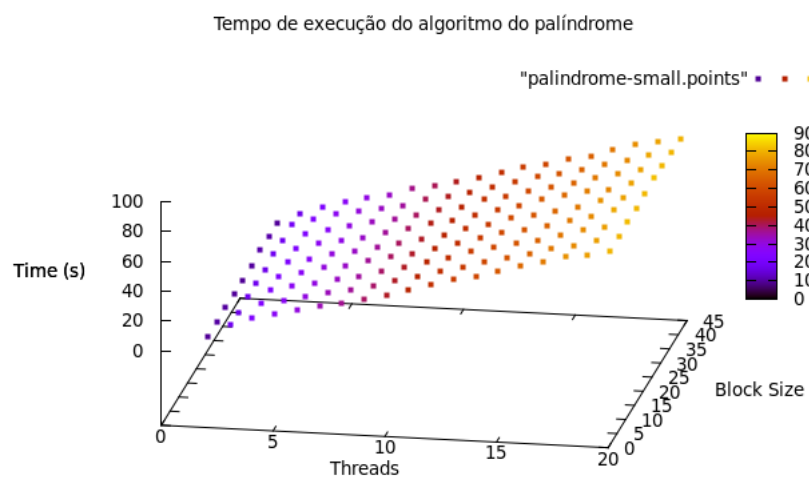


Figura 2: Resultados do algoritmo do palíndromo para o arquivo menor

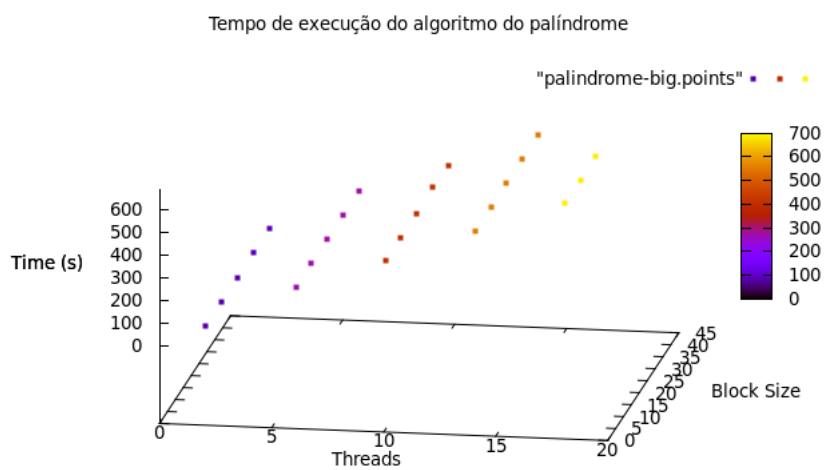


Figura 3: Resultados do algoritmo do palíndromo para o arquivo maior

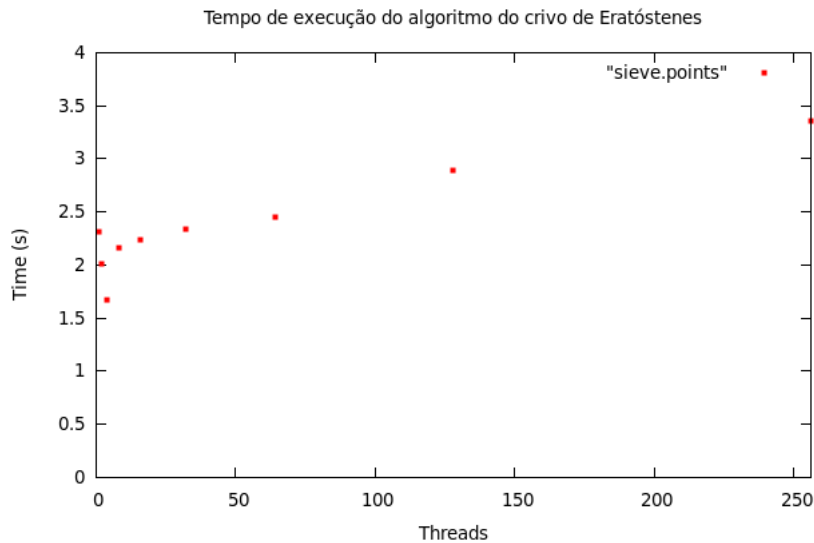


Figura 4: Tempos de execução do crivo usando OpenMP

A decomposição utilizada foi a de dados, pois foi paralelizada a demarcação dos múltiplos 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 4.

### 3 O projeto MPI

O projeto MPI foi desenvolvido seguindo a estrutura da figura 5. Todos os processos do programa têm suas chamadas principais executadas no início do programa. Existem três tipos de processos:

- Processos mestres principais.
- Processos mestres secundários.
- 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

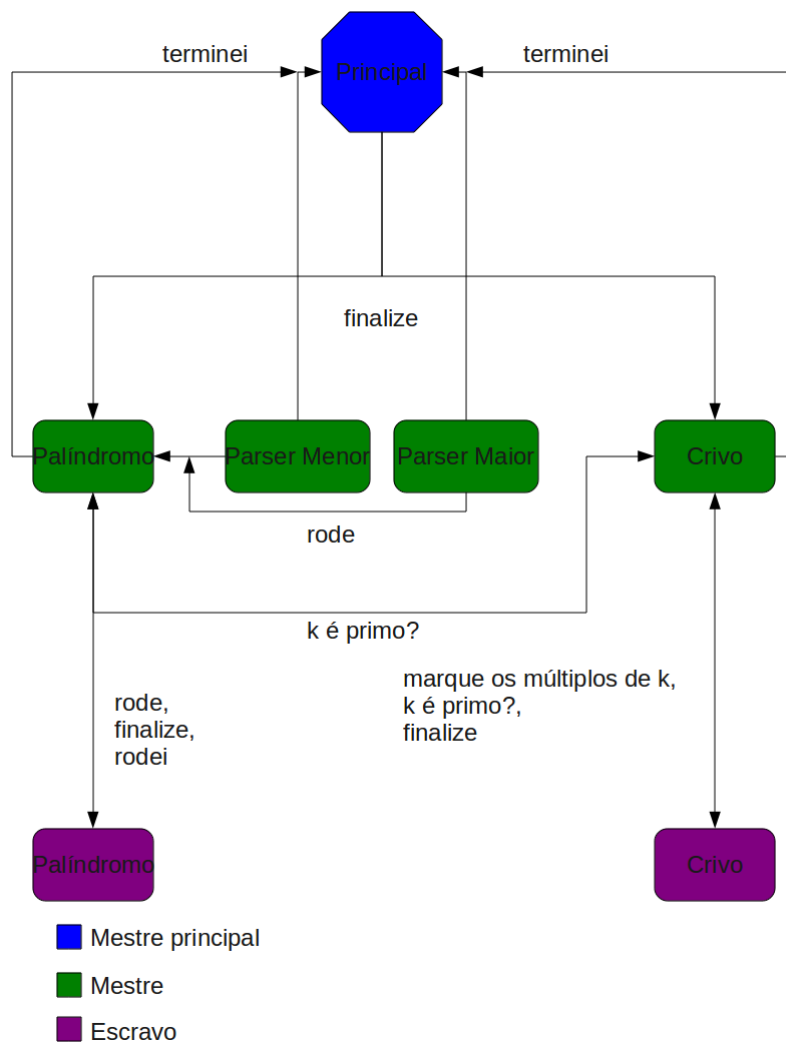


Figura 5: A estrutura do projeto MPI

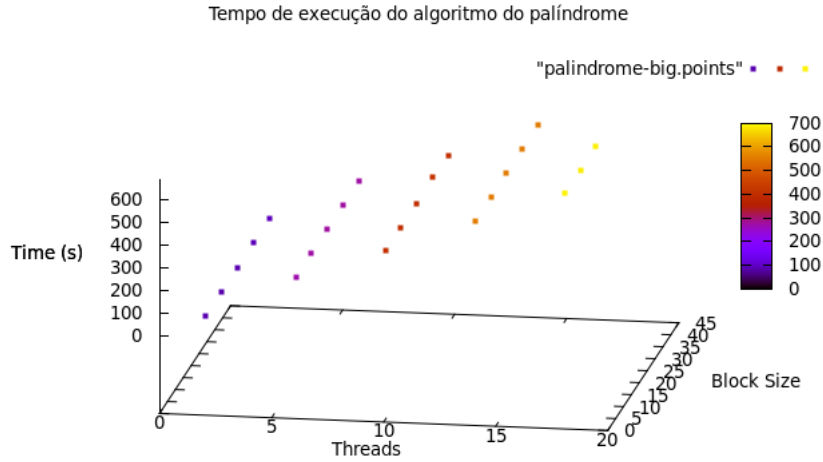


Figura 6: Resultados da execução do projeto MPI

em conjunto. Várias configurações de números de escravos foram utilizadas. Os resultados obtidos encontram-se na figura 6.

### 3.1 Execução do parser

TODO.

### 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 file 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.

### 3.3 Cálculo do crivo

O algoritmo para o cálculo é análogo para o projeto OpenMP. O cálculo aqui também paraleliza a marcação dos múltiplos dos primos. O processo mestre itera os números de 2 até a raiz do maior número e requisita aos 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

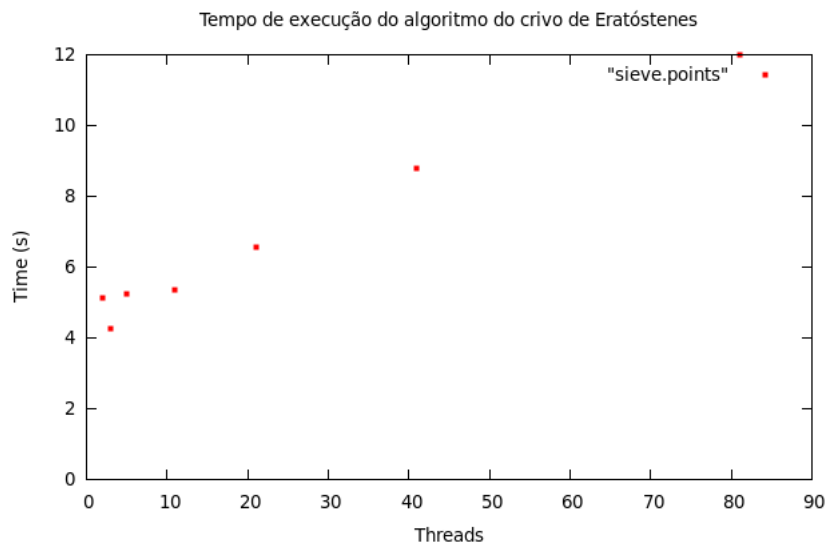


Figura 7: Tempos de execução do crivo usando MPI

a de dados, pelas mesmas razões citadas no projeto OpenMP.

Assim como anteriormente, testes de performance foram aplicados isoladamente do resto do programa. Os tempos obtidos encontram-se na figura 7.



## 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](http://en.wikipedia.org/wiki/Sieve_of_Eratosthenes)

[Crivo de Eratóstenes] [http://www.algorithmist.com/index.php/Prime\\_Sieve\\_of\\_Eratosthenes](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>