A Peneira de Eratosthenes é um algoritmo clássico para encontrar os números primos ≤ N (um número inteiro positivo). O pseudo código sequencial abaixo ilustra o funcionamento da Peneira de Eratosthenes:

1. Crie um vetor de números inteiros 2, 3, 4, ..., N, sendo que nenhum item do vetor é marcado como primo (i.e., são iniciados com zero);
2. Determine K = 2, sendo K o primeiro número não marcado do vetor;
3. Repita
   (a) marque todos os múltiplos de K entre K^2 e N;
   (b) encontre no vetor o menor número maior que K que não está marcado e atribua a K este novo valor;
   até $K^2 > N$
4. Os números não marcados são todos números primos.

Exemplo do algoritmo Peneira de Eratosthenes para N = 60.
a) Criar a lista/o vetor de números naturais. Nenhum está marcado.

XX	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60

(b) K = 2 (marcar nrs: 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, … , 60)

XX	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60

(c) K = 3 (marcar nrs: 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57 e 60)

XX	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60

(d) K = 5 (marcar nrs: 25, 30, 35, 40, 45, 50, 55 e 60)

XX	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60

(e) K = 7 (marcar nrs: 49 e 56)

XX	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60

(f) como K = 11, onde **K^2** > **N**, então sai do **repita...até**.

Os números primos são os não marcados: 02, 03, 05, 07, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53 e 59.

Faça um projeto de algoritmo paralelo seguindo a metodologia PCAM para o problema da peneira de eratosthenes. Descreva, explicitamente, de forma textual e gráfica (se necessário), o particionamento, comunicação, aglomeração e mapeamento.

Considere que este algoritmo paralelo deve minimizar o tempo de resposta e que será carregado para execução em um cluster de computadores.

Considere que este projeto será encaminhado para outro grupo implementá-lo. Espera-se que o projeto seja suficientemente detalhado para que outros possam fazer a implementação adequadamente.

Particionamento
Versão 1: Particionando o vetor de nrs inteiros que representam os dados
Seguindo um particionamento por dados, o vetor de n-1 elementos é particionado em n-1 tarefas, cada uma responsável por marcar um elemento do vetor, caso este elemento seja múltiplo de k ou não a cada iteração (nova atribuição de k). Ao marcar um elemento do vetor, a tarefa não necessita mais continuar a computação. Enquanto não marcar, a tarefa continua a verificação do seu número no vetor em função de k. Outros critérios de parada também podem ser aplicados (como num < k^2).

Versão 2:particionando o k, valor que será buscado no vetor não particionado
Seguindo um particionamento da funcionalidade (ou de dados com interleaving) cada tarefa receberá um valor de k >= 2 e fará as buscas por múltiplos deste k a partir de k^2 até n.

Comunicação
Versão 1:
Ao iniciarem, as tarefas necessitam da posição do vetor (inicialmente marcadas como zero) e de k (inicialmente 2). A cada iteração um novo valor de k precisa ser calculado. O novo valor de k é o menor valor no vetor maior que o k atual e que não tenha sido marcado como múltiplo. Esse novo k deve ser difundido para todas as tarefas que ainda não marcaram seus números como múltiplos de um k anterior.
Essas atividades compreendem duas comunicações globais. Primeiro faz-se uma operação de redução para encontrar para o novo k o menor valor dentre os elementos do vetor ainda não marcados e que sejam maiores que o k atual. Depois esse novo k deve ser enviado para todas as tarefas ainda ativas (não marcadas como um número múltiplo).

Versão 2:
Há uma tarefa para cada k que se deseja encontrar os nrs primos no vetor que agora é compartilhado (não replicado, nem particionado) entre as diferentes tarefas. Há um valor de k global de modo que todas as tarefas sabem qual é o proximo k a ser buscado. Há um k local para k thread fazer as suas buscas sobre o vetor. O valor de k global é consultado sob demanda pela thread que deseja atualizar o seu k local.

Aglomeração (considerando um cluster com memória distribuída)

Neste tipo de plataforma tem-se como objetivo colocar o problema em termos de p processos. O valor de p é dependente do número de processadores, porém, não precisa ser necessariamente igual, i.e., pode ser maior. O que não é usual, é colocar o nr de processos em função da carga de trabalho, sem considerar o nr de processadores existentes.

Versão 1:
As tarefas devem ser aglomeradas formando p processos que marcam blocos de nrs de tamanho n/p, acertando-se as divisões não exatas. O valor exato de p pode ser >= a proc (nr de processadores).
Essa aglomeração acarreta em uma falta de balanceamento do nr de itens marcados por bloco, pois os processos responsáveis pela marcação dos nrs mais altos do vetor terão mais números para marcar, que os processos com blocos de nrs menores do vetor. Para tratar essa falta de balanceamento propõe-se usar p > proc, de modo que o mapeamento tente minimizar a falta de balanceamento.
Há também a análise do critério de parada (k^2 < n), quando esse valor de k não está presente apenas no primeiro processo. Neste caso, diferentes processos devem buscar o novo k, não apenas o primeiro.
O primeiro processo determina o novo valor de k se ele contiver todos os valores até K^2 > n.
Há uma comunicação global para difundir o novo k aos demais processos.

Versão 2
As tarefas geradas já formam os processos diretamente, o que dificulta melhorias no particionamento feito. O primeiro processo, com k=2, já tem metade da carga de trabalho, o que gera um grande desbalanceamento. Essa versão precisa ser refeita a partir do particionamento, reanalisando a divisão das atividades.

Mapeamento

Versão 1:
Os p processos são mapeados nos proc processadores disponíveis usando-se, no mínimo, uma fila circular. Assume-se que p >= proc.
Para minimizar a falta de balanceamento nesta versão, propõe-se usar mais processos que processadores, executando mais processos com nrs menores no mesmo processdor. Colocando-se p = 2 x proc, pode-se atribuir inicialmente um processo para cada processador e colocar na sequência os demais processos atéa metade do nr de processadores, como uma fila circular.

Versão 2:
O processo com k = 2 é mapeado em um processador, os demais nrs são mapeados em outro processador, enquanto a soma do tempo de resposta de todos os demais não ultrapassa o tempo de resposta do processo com k = 2. A partir desse limite, coloca-se os demais processos em um terceiro processador, lembrando que o processo com k = 2 não é dividido.