Algoritmos Paralelos Eficientes para Alguns Problemas em Processamento de Cadeias de Caracteres

Siang Wun Song

JAI/SBC 2007 - Rio de Janeiro
Parte 1

Agradecimentos

- O material deste curso é baseado em trabalhos publicados conjuntamente com os Professores Edson Norberto Cáceres e Carlos Eduardo Rodrigues Alves.
- As pesquisas realizadas contaram com o apoio do CNPq e FAPESP.
- Agradecimentos também são devidos aos organizadores da JAI e aos revisores anônimos.

Computação Paralela - Oportunidade

O que já temos ou que está no horizonte:

- Sistemas de computação maciçamente paralelos serão cada vez mais comuns.
- Clusters Beowulf baseados em arquiteturas abertas tornam o uso da Computação Paralela cada vez mais popular.
- Novas arquiteturas de processadors: multi-core: Intel já anunciou o lançamento de um chip com 80 processadores, um trilhão de operações aritméticas por segundo (1 TFLOPS).
- Médio ou longo prazo: Novas arquiteturas eficientes em energia e tecnologias que não são baseadas em silício.

Computação Paralela - Dificuldades

- Dificuldade de lidar com paralelismo: como projetar um programa eficiente para centenas ou milhares de processadores?
- Uma forma é o desenvolvimento de compiladores que fazem a paralelização automática a partir de um código sequencial.
  - É a forma ideal.
  - Mas temos muito pouco progresso.

Outra forma: para cada problema, um algoritmo paralelo

- Para cada problema com alta demanda computacional, projetar um programa paralelo eficiente.
  - O progresso também deixa a desejar.
  - Conseguimos lidar com aplicações trivialmente paralelizáveis.
  - Para muitas aplicações, levamos tempo demais para obter uma solução paralela eficiente.
  - Muitos algoritmos paralelos desenvolvidos não são escaláveis.

Computação Paralela - Desafios

Computação Paralela será regra e não exceção.

- Necessidade de formar profissionais capazes de projetar algoritmos paralelos eficientes e escaláveis.
- Descobrir novas aplicações com alta demanda computacional que podem tirar proveito do paralelismo maciço.
- Criar novos (?) modelos de computação, novos (?) paradigmas de programação paralela ou novas (?) linguagens de programação paralela.

Objetivos deste Mini-Curso

Este curso mostra:

- A importância de adotar modelos de computação paralela adequada.
- Um bom algoritmo paralelo requer um estudo cuidadoso das característica do problema.
- Vamos ilustrar isso mostrando como desenvolver algoritmos paralelos para dois importantes problemas em processamento de cadeias.

Os Dois Problemas

- Problema de Todas Subseqüências Maximais
  - Caso particular: Problema de Subseqüência Máxima
- Problema de Toda-Subcadeia Subseqüência Comum Mais Longa
  - Caso particular: Problema de Subseqüência Comum Mais Longa

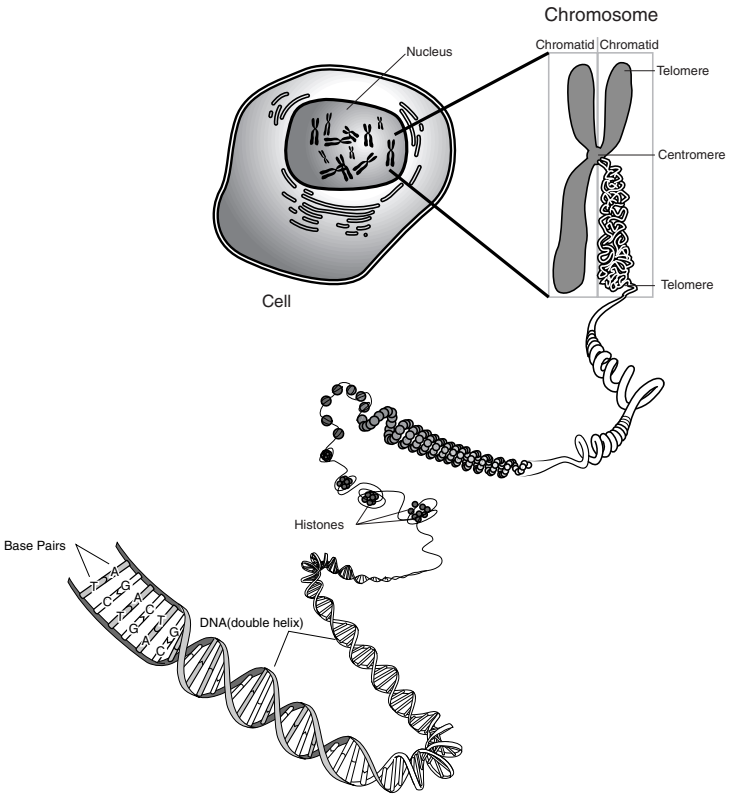
Vários Conceitos Importantes

- Antes de iniciar a discussão dos dois problemas e suas soluções paralelas vamos introduzir alguns conceitos preliminares.

Biologia Computacional

- Os problemas que vamos examinar lidam com seqüências (e.g. de DNA ou amino-ácidos) e têm importantes aplicações em Biologia Computacional.
- A seguir vamos dar uma introdução muito sucinta de conceitos como DNA, genes, etc.

Chromosome

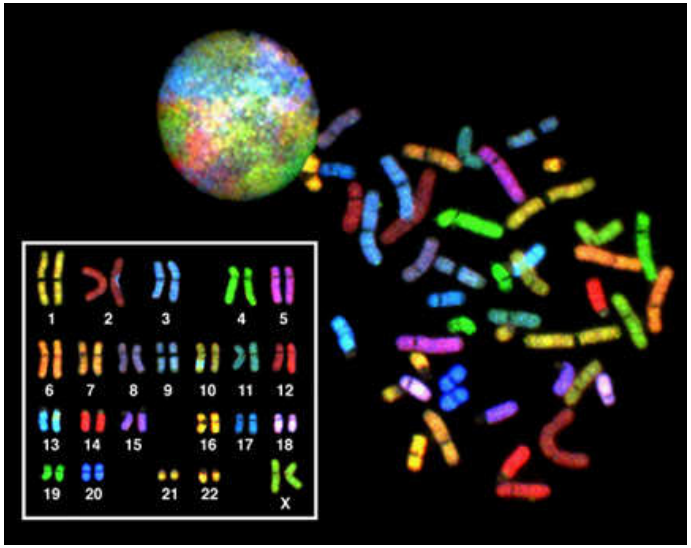


National Institutes of Health

National Human Genome Research Institute

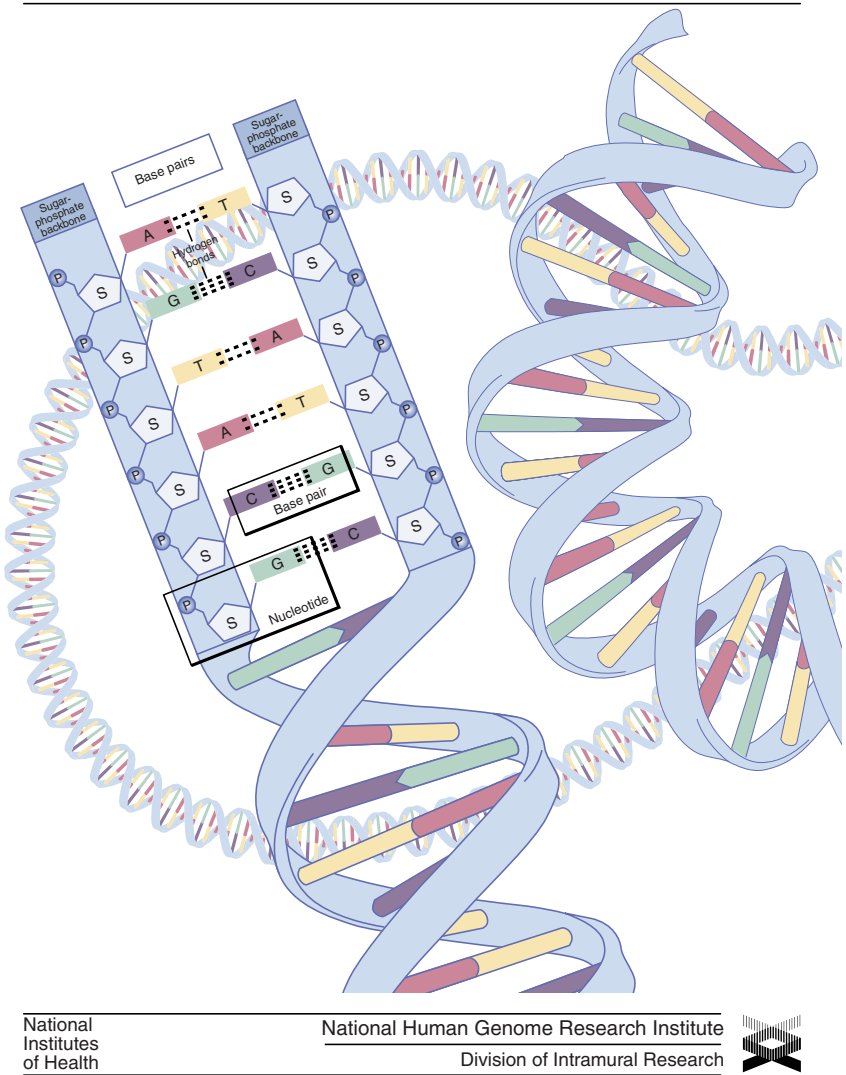
Division of Intramural Research

Uma pessoa tem trilhões de células. Em cada célula há 2 conjuntos de 23 cromosomos cada.



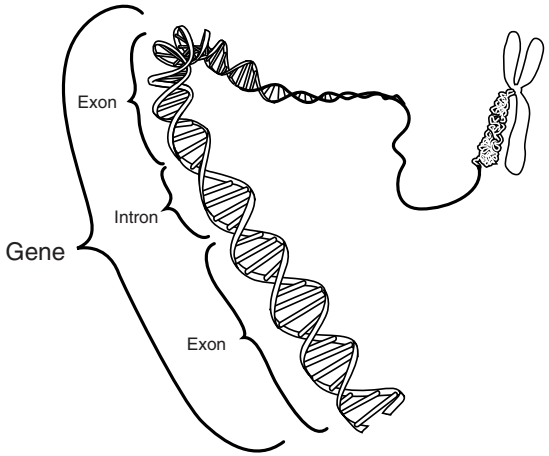
National Institutes of Health
National Human Genome Research Institute
Division of Intramural Research

O DNA completo (genoma) humano tem 3 bilhões de bases (representadas pelas letras A, T, C e G).



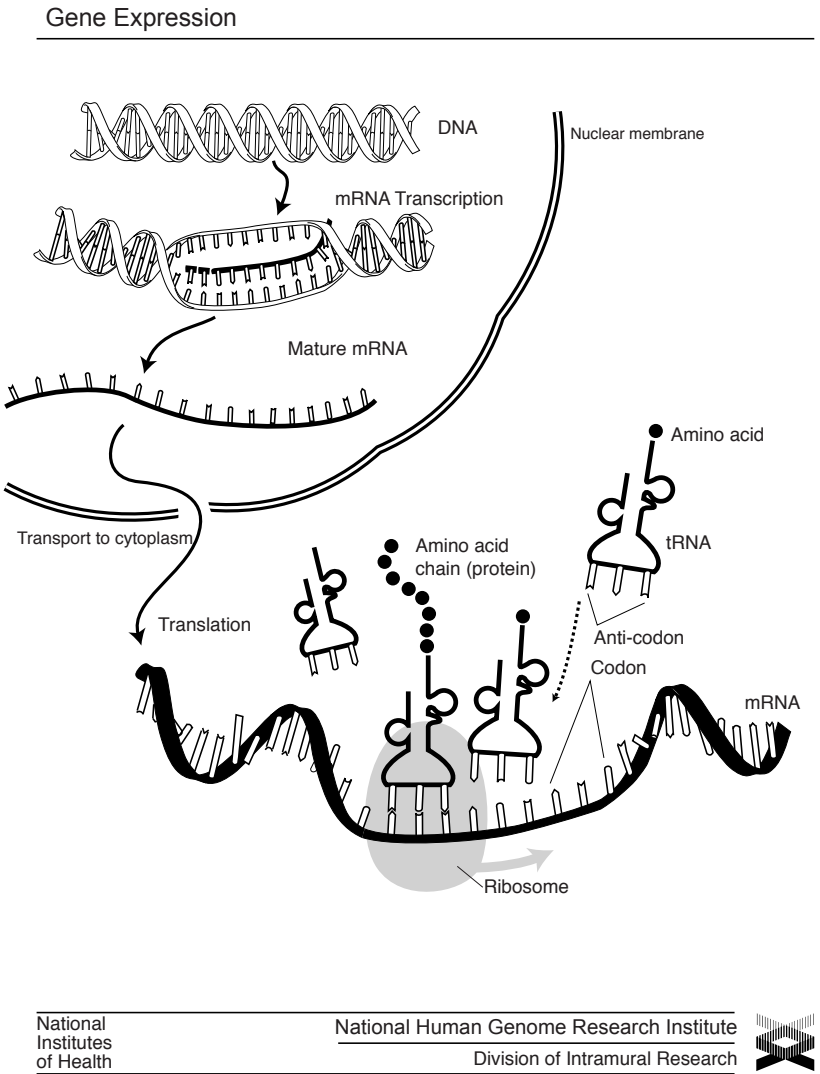
Todo o DNA no núcleo de uma célula mede mais que 1 m.

Gene



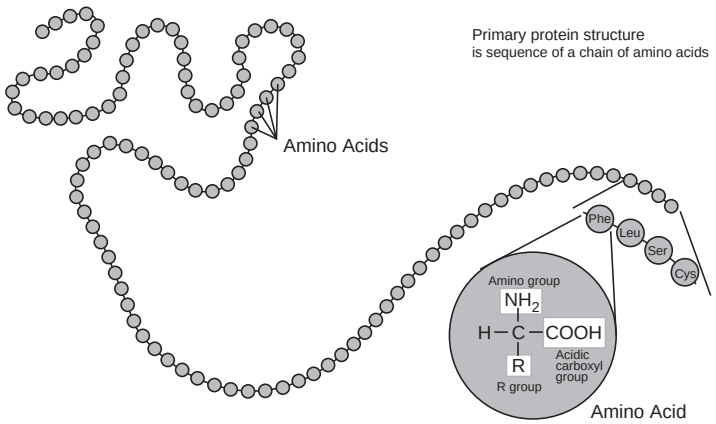
National Institutes of Health
National Human Genome Research Institute
Division of Intramural Research

Um gene é o código responsável pela síntese de proteína. Há cerca de 100 mil genes no ser humano.



Expressão gênica é o processo de converter uma seqüência de DNA de um gene em proteína, pela transcrição em RNA mensageiro.

Amino acid

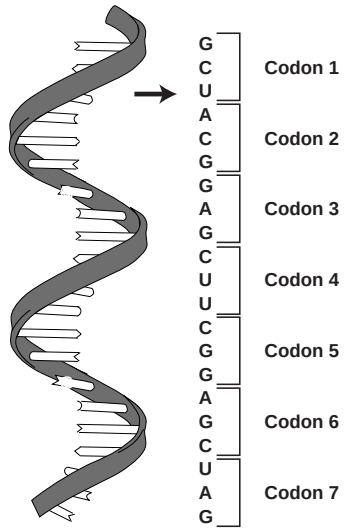


National Institutes of Health

National Human Genome Research Institute

Division of Intramural Research

A proteína é constituída de uma seqüência de amino-ácidos.



RNA

Ribonucleic acid

National Institutes of Health

National Human Genome Research Institute

Division of Intramural Research

Codon é o código genético formado por grupo de 3 bases que é transformado em amino-ácido.

Complexidade de um Algoritmo

A complexidade é a medida de eficiência de um algoritmo.

- A complexidade de um algoritmo é medida pelo número de operações que um algoritmo realiza para resolver um problema. Essa medida é melhor do que medir simplesmente o tempo absoluto, pois o tempo é dependente da máquina, além do algoritmo.
- A complexidade é expressa como uma função do tamanho da entrada, geralmente indicada por n.
- Tem especial interesse a complexidada assintótica, quando n é considerado muito grande, tendendo a infinito.
- Usa-se a notação O, por exemplo, $O(n)$ ou $O(n^2)$ para mostrar que a complexidade varia linearmente com n (isto é, proporcional a n), ou varia quadraticamente (com o quadrado de n).
- Quando a complexidade é constante, isto é, não depende de n, então dizemos que a complexidade é $O(1)$.

10000000 n é melhor que $1000n^2$ que é melhor que 2^n

- 5 algoritmos de complexidades $T_1(n)$ a $T_5(n)$ resolvem um mesmo problema.
- Supomos que uma operação leva 1 ms.
- A tabela dá o tempo usado por cada um dos algoritmos.

n	$T_1(n)$ n	$T_2(n)$ $n \log n$	$T_3(n)$ n^2	$T_4(n)$ n^3	$T_5(n)$ 2^n
16	0.016s	0.064s	0.256s	4s	1m 4s
32	0.032s	0.16s	1s	33s	46 dias
512	0.512s	9s	4m 22s	1 dia 13h	10^{137} séculos

10000000 n é melhor que $1000n^2$ que é melhor que 2^n

- 5 algoritmos de complexidades $T_1(n)$ a $T_5(n)$ resolvem um mesmo problema.
- Supomos que uma operação leva 1 ms.
- A tabela dá o tempo usado por cada um dos algoritmos.

n	$T_1(n)$ n	$T_2(n)$ $n \log n$	$T_3(n)$ n^2	$T_4(n)$ n^3	$T_5(n)$ 2^n
16	0.016s	0.064s	0.256s	4s	1m 4s
32	0.032s	0.16s	1s	33s	46 dias
512	0.512s	9s	4m 22s	1 dia 13h	10^{137} séculos

10000000 n é melhor que $1000n^2$ que é melhor que 2^n

- 5 algoritmos de complexidades $T_1(n)$ a $T_5(n)$ resolvem um mesmo problema.
- Supomos que uma operação leva 1 ms.
- A tabela dá o tempo usado por cada um dos algoritmos.

n	$T_1(n)$ n	$T_2(n)$ $n \log n$	$T_3(n)$ n^2	$T_4(n)$ n^3	$T_5(n)$ 2^n
16	0.016s	0.064s	0.256s	4s	1m 4s
32	0.032s	0.16s	1s	33s	46 dias
512	0.512s	9s	4m 22s	1 dia 13h	10^{137} séculos

10000000 n é melhor que $1000n^2$ que é melhor que 2^n

- 5 algoritmos de complexidades $T_1(n)$ a $T_5(n)$ resolvem um mesmo problema.
- Supomos que uma operação leva 1 ms.
- A tabela dá o tempo usado por cada um dos algoritmos.

n	$T_1(n)$ n	$T_2(n)$ $n \log n$	$T_3(n)$ n^2	$T_4(n)$ n^3	$T_5(n)$ 2^n
16	0.016s	0.064s	0.256s	4s	1m 4s
32	0.032s	0.16s	1s	33s	46 dias
512	0.512s	9s	4m 22s	1 dia 13h	10^{137} séculos

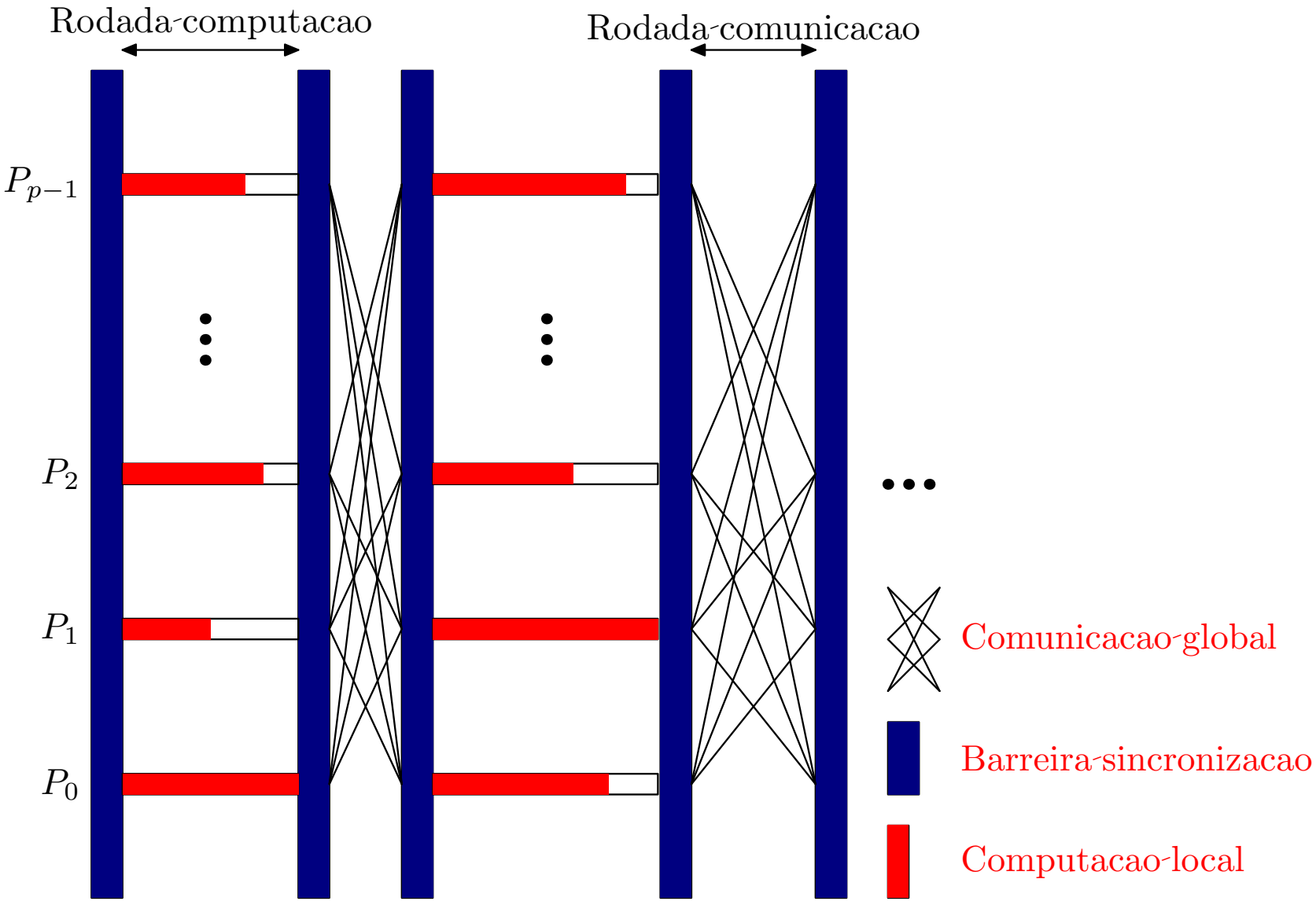
Complexidade de um algoritmo paralelo

Considere um dado problema.

- Suponha que um algoritmo seqüencial resolve o problema em tempo T_{seq}.
- Se um algoritmo paralelo usando p processadores resolve o problema em tempo T_{par}, então dizemos que houve um *speed-up* de T_{seq}/T_{par}.
- Um caso ideal é um algoritmo paralelo produzir um *speed-up* de $O(p)$.
- Por exmemplo tempo seqüencial = $O(n)$ e tempo paralelo = $O(n/p)$.

Modelo CGM - Coarse Grained Multicomputer

- Considere a entrada do problema de tamanho $O(n)$.
- Multicomputador com p processadores, cada um com memória local de tamanho $O(n/p)$.
- Um algoritmo CGM consiste numa alternância de duas rodadas:
  - Rodada de computação: cada processador computa independentemente dos demais.
  - Rodada de comunicação: cada processador pode enviar/receber $O(n/p)$ dados dos demais.
- A meta é minimizar o número de rodadas requeridas, a fim de obter um *speed-up* linear em p.


Rodada-computacao
Rodada-comunicacao
P_{p-1}
P_2
P_1
P_0
Comunicacao-global
Barreira-sincronizacao
Computacao-local