

Capítulo 7

Processamento Co-sequencial e Ordenação

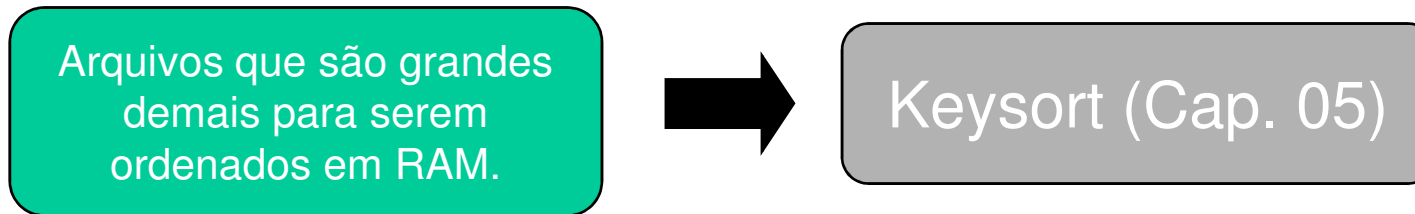
Parte 3



Objetivos

- Descrever uma atividade de processamento, freqüentemente utilizada, chamada de processamento co-sequencial.
- Apresentar um modelo genérico para implementar todas as variações de processamento co-sequencial.
- Ilustrar o uso do modelo para resolver diferentes problemas de processamento co-sequencial.
- Apresentar o algoritmo “K-way merge” (intercalação múltipla).
- Apresentar o heapsort como uma abordagem para sorting em RAM
- Mostrar como merging (intercalação) fornece a base para ordenação de arquivos grandes.

● ● ● | Sorting arquivos grandes



- Desvantagens:
 - Depois de ordenar as chaves, existe um custo substancial de seeking para ler e reescrever cada registro no arquivo novo.
 - O tamanho do arquivo é limitado pelo número de pares chave/ponteiro que pode ser armazenado na RAM. Inviável para arquivos realmente grandes.



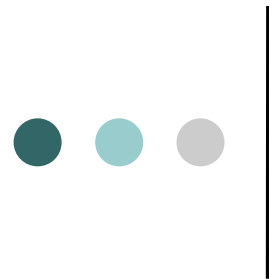
Sorting arquivos grandes

- Exemplo:
 - Características do arquivo a ser ordenado:
 - 8.000.000 de registros
 - Tamanho do reg: 100 bytes
 - Tamanho da chave: 10 bytes
 - Memória disponível para o trabalho: 10 MB (Sem contar memória para manter o programa, SO, buffer de I/O, etc.)
 - Tamanho total do arquivo: 800 MB
 - Número total de bytes para todas as chaves: 80 MB

Não é possível fazer Internal Sorting nem Keysorting.

● ● ● | Sorting arquivos grandes

- Solução => MergeSort
 - Criar sub-arquivos chamados “*runs*”:
 - trazer o máximo de registros possíveis para a memória, fazer o Internal Sorting e salvar num arquivo menor. Repetir este processo até que todos os registros do arquivo original tenham sido lidos.
 - Fazer o Multi-way Merge (Intercalação Múltipla) dos arquivos ordenados



Sorting arquivos grandes (MergeSort)

- No exemplo em questão, qual seria o tamanho de cada *run*?
 - Memória disponível = 10MB = 10.000.000 bytes
 - Tamanho do registro = 100 bytes
 - Número de registros que cabem na memória disponível = ???

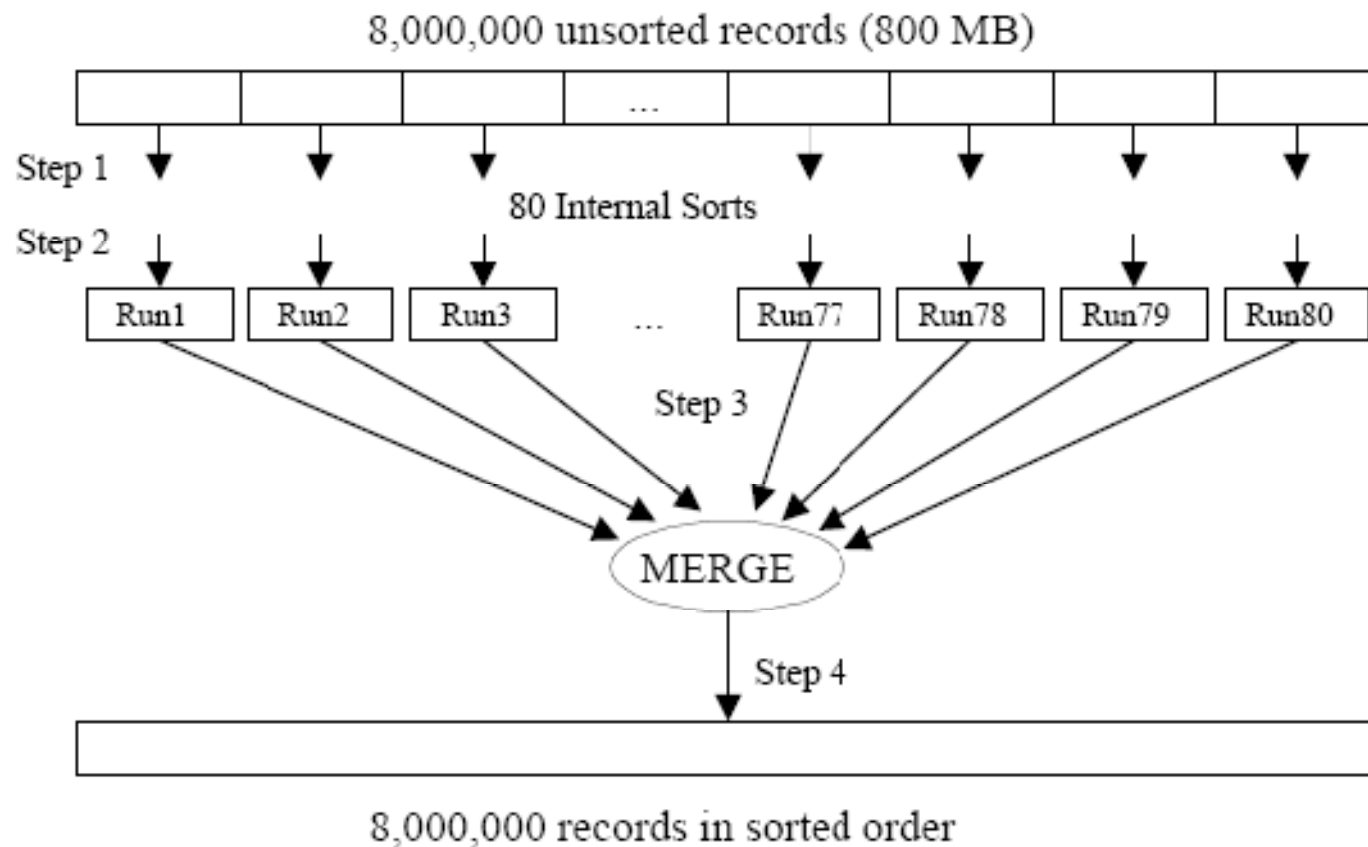
100.000 registros

- Se o número total de registros é 8.000.000, o número total de runs é = ???

80 runs



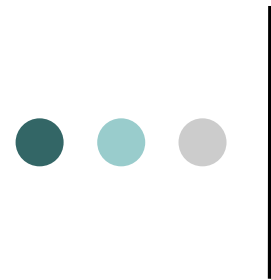
Sorting arquivos grandes (MergeSort)





Sorting arquivos grandes (MergeSort)

- A solução “criar runs” + “intercalação múltipla” tem as seguintes características:
 - Pode-se, de fato, ordenar arquivos grandes, e é extensível para arquivos de qualquer tamanho.
 - Leitura do arquivo input durante o passo de criação do run é sequencial.
 - A leitura do run durante o processo de merge e escrita dos reg ordenados também é sequencial. Acesso randômico é necessário somente quando alternamos de um run para outro durante o merge.



Sorting arquivos grandes (MergeSort)

- Operações de I/O são realizadas 4 vezes:
 - Durante a fase do Sort:
 1. Leitura de cada registro na memória para ordenar e criar os runs.
 2. Escrita dos runs ordenados no disco.
 - Durante a fase do Merge:
 3. Leitura dos runs ordenados na memória para realizar o merge.
 4. Escrita do arquivo ordenado no disco.



Sorting arquivos grandes (MergeSort)

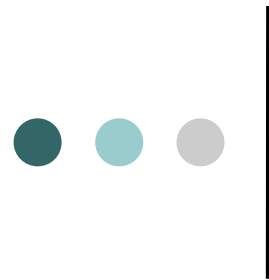
1. Leitura de cada registro na memória para ordenar e criar os runs.
2. Escrita dos runs ordenados no disco.

- Os passos 1 e 2 são feitos da seguinte forma:
 - Lê blocos de 10MB e escreve blocos de 10MB.
 - Repete isto 80 vezes.
 - Em termos de operações básicas em disco:
 - Para leitura: 80 seeks + tempo de transferência p/ 800 MB
 - Para escrita: idem.

● ● ● | Sorting arquivos grandes (MergeSort)

3. Leitura dos runs ordenados na memória para realizar o merge.

- O passo 3 é feito da seguinte forma:
 - Para minimizar o número de seeks, leia um bloco de cada run, ou seja, 80 blocos.
 - Uma vez que a memória disponível é 10MB cada bloco pode ter 10.000.000 bytes/80 runs = 125.000 bytes = 1.250 reg.
 - Quantos blocos serão lidos para cada run?
 - $\text{Tamanho do run/tamanho do bloco} = 10.000.000 / 125.000 = 80$
 - Número total de seeks = número total de blocos
 - Número total de blocos = nro runs x nro blocos por run
 - $80 \text{ runs} \times 80 \text{ blocos/run} = 80^2 \text{ blocos} = 6.400 \text{ seeks.}$



Sorting arquivos grandes (MergeSort)

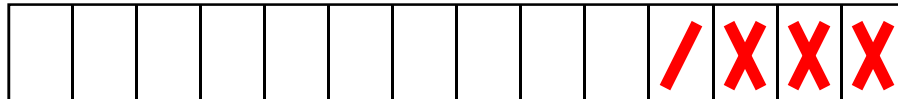
- Em geral para intercalação de K -runs, onde cada run ocupa o espaço disponível de RAM, o tamanho de buffer disponível para cada run é $(1/K)$ tamanho da RAM = $(1/K)$ tamanho do run
- São necessários portanto K seeks para ler cada run e existem K runs ao total, portanto são K^2 seeks. Essa operação é de complexidade $O(K^2)$.
- Ex: aumentar em N vezes o número de registros significa que aumentamos em N^2 o número de seeks.

● ● ● | Sorting arquivos grandes (MergeSort)

1º run = 80 blocos (80 acessos)

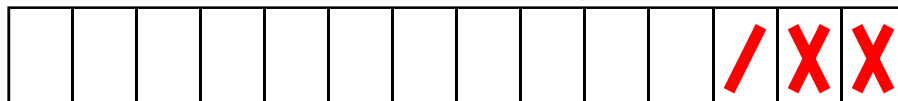


2º run = 80 blocos (80 acessos)



•
•
•

80º run = 80 blocos (80 acessos)



8.000.000 reg
ordenados



Sorting arquivos grandes (MergeSort)

4. Escrita do arquivo ordenado no disco.

- O passo 4:
 - Para escrever o arquivo ordenado no disco, o número de seeks depende do tamanho do buffer de saída (output buffer):
 - Bytes no arquivo / bytes no buffer de saída

Obs: O passo 3 domina o tempo de execução, ou seja, é o gargalo deste método.

● ● ● | Sorting arquivos grandes (MergeSort)

- Maneiras de reduzir o tempo do passo 3 (gargalo):
 1. Alocar mais hardware
 - ➔ 2. Realizar o merge em mais de um passo (reduz a ordem do merge e aumenta o tamanho dos runs.)
 - ➔ 3. Aumentar algoritmicamente o tamanho de cada run.
 4. Encontrar maneiras de sobrepor operações de I/O.



Sorting arquivos grandes (MergeSort)

1. Alocar mais hardware

- Aumentar a quantidade de memória RAM
 - Uma memória maior significa menos e maiores runs na fase do sort, e menos seek por run durante a fase do merge.
- Aumentar o número de discos
 - Se tivéssemos dois discos dedicados para o merge, poderíamos designar um para input e outro para output, portanto leitura e escrita poderia se sobrepor sempre que ocorressem simultaneamente.
- Aumentar o número de canais de I/O
 - Se existir um canal de I/O para cada disco, I/O pode-se sobrepor completamente.



Sorting arquivos grandes (MergeSort)

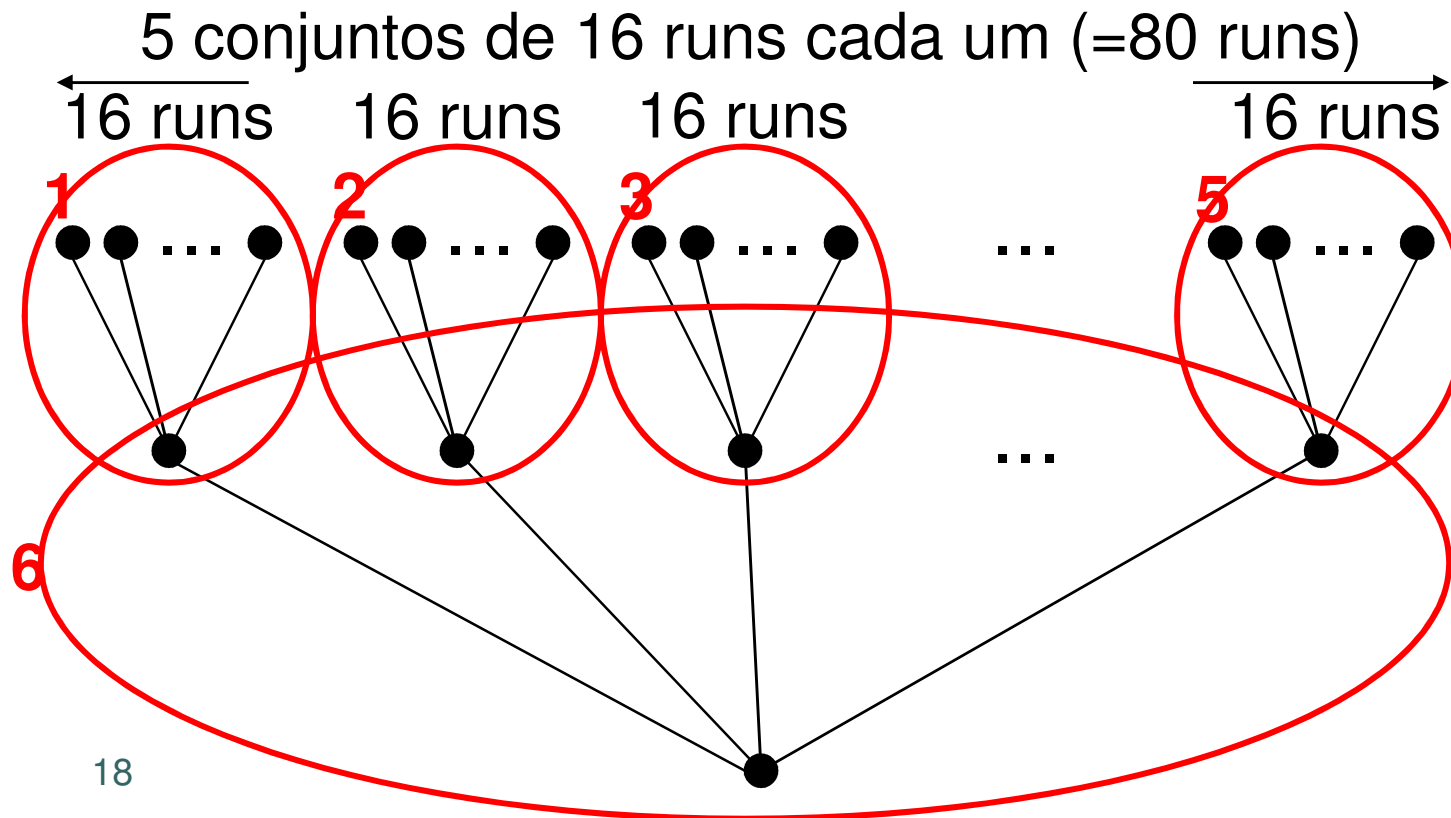
2. Realizar o merge em mais de um passo (reduz a ordem do merge e aumenta o tamanho dos runs.)
 - Uma das maneiras de reduzir seeking é reduzir o número de runs que nós temos que fazer o merge, portanto, dando para cada run uma parcela maior do espaço disponível do buffer.
 - **Multiple-step Merging**: quebramos o conjunto original de runs em grupos menores e fizemos o merge dos runs nestes grupos, separadamente.

Em cada um destes merges menores, mais espaço do buffer fica disponível para cada run, portanto menos seeks são necessário por run.



Sorting arquivos grandes (MergeSort)

2. Multiple-step Merging:





Sorting arquivos grandes (MergeSort)

2. Multiple-step Merging:

- Temos menos seeks na primeira passada, mas ainda há uma segunda passada.
- Existe vantagem em ler cada registro duas vezes?
 - 80-way merge original: 6.400 seeks.
 - Multiple-step merging:

Cabem
100.000
reg por
vez na
RAM
(5 blocos
de
20.000
reg
cada)

- 1ª passada: nro total de seeks = nro total de blocos
 - 16 runs x 16 blocos/run = 16^2 blocos = 256 seeks
 - 5 conjuntos x 256 = 1280 seeks.
- 2ª passada: temos 5 runs finais, então 1/5 do espaço do buffer é alocado para cada run. Cada run tem 1.600.000 reg, ou seja, teremos blocos de 20.000 registros (1/5) sendo carregados para o buffer cada vez. Isso equivale a 80 seeks por run. Portanto, na 2ª passada temos 80 seeks x 5 = 400 seeks.
- TOTAL: 1280 + 400 = 1680 seeks.

8.000.000 reg / 5

1.600.000reg/20.000reg



Sorting arquivos grandes (MergeSort)

2. Multiple-step Merging:

- Encontramos uma maneira de aumentar o espaço disponível no buffer para cada run.
- Trocamos passadas extras sobre os dados por uma diminuição significativa no acesso randômico.
- Se fizermos um merge em 3 passos, podemos obter resultados ainda melhores?
 - Talvez não, pois temos que considerar o tempo de transmissão dos dados. No merge original, os dados eram transmitidos apenas 2 vezes, no merge com 2 passos, são 4 vezes.

● ● ● | Sorting arquivos grandes (MergeSort)

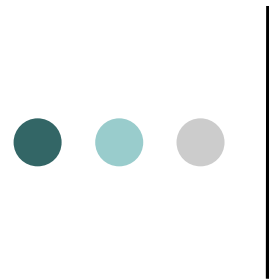
3. Aumentar algoritmicamente o tamanho de cada run.

Se pudermos, de alguma maneira,
aumentar o tamanho dos runs iniciais:



Diminuímos o trabalho necessário durante a passo do merge, durante o processo de ordenação:

- Runs iniciais maiores, significa menos runs no total, e um merge de baixa ordem.
- Conseqüentemente, buffer maiores, e menos seeks.



Sorting arquivos grandes (MergeSort)

- Sem a possibilidade de comprarmos o dobro de memória, como podemos criar runs iniciais que sejam duas vezes maiores do que o número de registros que nós podemos armazenar na RAM?

Replacement Selection



Sorting arquivos grandes (MergeSort)

REPLACEMENT SELECTION:

1. Ler uma coleção de registros e ordená-los usando o heapsort.
Isto cria uma heap de valores ordenados. Chame esta heap de *heap primária*.
2. Ao invés de escrever (output) toda a *heap primária* ordenada (como fazemos num heapsort normal), escreva (output) somente o registro cuja chave tem o menor valor.
3. Traga um novo valor e compare sua chave com aquela que acabara de ser escrita
 - a. Se a nova chave tem uma valor maior, insira o novo registro no local apropriado dentro da heap primária juntamente com os outros registros que estão sendo selecionados para output.
 - b. Se a nova chave tem um valor menor, coloque aquele registro numa *heap secundária* de registros cujas chaves tem valores menores do que aquela chave que já foi escrita (output).
4. Repita o passo 3 enquanto existirem registros na *heap primária* e registros para serem lidos. Quando a *heap primária* estiver vazia, faça a *heap secundária* ser a *heap primária* e repita os passos 2 e 3.

● ● ● | Sorting arquivos grandes (MergeSort)

- Exemplo do princípio básico da “replacement selection”

Input: 21, 67, 12, 5, 47, 16



Frente da string input

O que acontece se a 4ª chave for 2, ao invés de 12.

Input remanescente

Memória (P=3)

Output run

21, 67, 12	5	47	16	–
21, 67	12	47	16	5
21	67	47	16	12, 5
–	67	47	21	16, 12, 5
–	67	47	–	21, 16, 12, 5
–	67	–	–	47, 21, 16, 12, 5
–	–	–	–	67, 47, 21, 16, 12, 5

O processo produz uma lista de **seis** chaves usando apenas **três** locações de memória.

● ● ● | Sorting arquivos grandes (MergeSort)

- “Replacement selection” formando dois runs ordenados.

Input: 33, 18, 24, 58, 14, 17, 7, 21, 67, 12, 5, 47, 16

Frente da string input ————— ↑

Input remanescente	Memória (P=3)	Output run
33, 18, 24, 58, 14, 17, 7, 21, 67, 12	5 47 16	—
33, 18, 24, 58, 14, 17, 7, 21, 67	12 47 16	5
33, 18, 24, 58, 14, 17, 7, 21	67 47 16	12, 5
33, 18, 24, 58, 14, 17, 7	67 47 21	16, 12, 5
33, 18, 24, 58, 14, 17	67 47 (7)	21, 16, 12, 5
33, 18, 24, 58, 14	67 (17) (7)	47, 21, 16, 12, 5
33, 18, 24, 58	(14) (17) (7)	67, 47, 21, 16, 12, 5

● ● ● | Sorting arquivos grandes (MergeSort)

- “Replacement selection” formando dois runs ordenados.

Começa a construção do 2º. run...

Input remanescente	Memória (P=3)	Output run
33, 18, 24, 58	14 17 7	–
33, 18, 24	14 17 58	7
33, 18	24 17 58	14, 7
33	24 18 58	17, 14, 7
–	24 33 58	18, 17, 14, 7
–	– 33 58	24, 18, 17, 14, 7
–	– – 58	33, 24, 18, 17, 14, 7
–	– – –	58, 33, 24, 18, 17, 14, 7



Sorting arquivos grandes (MergeSort)

- Procedimento anterior:
 - Ler as chaves na memória
 - Ordená-las
 - Escrever (output) um run que é o tamanho do espaço em memória disponível para esse procedimento.
 - No exemplo das 13 chaves, teríamos 5 runs.
- Com “replacement selection”:
 - No exemplo das 13 chaves, temos apenas 2 runs utilizando o mesmo espaço de memória.

Utilizando o procedimento de “replacement selection” temos runs mais longos, diminuindo assim o número total de runs necessários.



Sorting arquivos grandes (MergeSort)

- Utilizando “Replacement Selection”:
 - Dadas P locações de memória, qual o tamanho médio do run que o procedimento produz?
 - Na média podemos esperar um de tamanho igual a $2P$ (Knuth’73) – o dobro do tamanho usado em merge sort com a mesma memória.
 - Quais são os custos de se utilizar “Replacement Selection”?
 - Sabemos que o custo de buscar cada registro individualmente no disco é proibitivo. Ao invés disto, temos que carregar um input num buffer, o que significa que não temos toda a memória para a operação de “Replacement Selection”. Parte da memória tem que ser usada para os buffer de input e output.



Sorting arquivos grandes(MergeSort)

- Para obter mais eficiência, podemos combinar:

“Replacement Selection”

+

Multi-step Merge



Sorting arquivos grandes (MergeSort)

- Ferramentas para melhorar a performance do *External Sorting*:
 1. Para o sorting em RAM, usar o heapsort formando as listas ordenadas de cada run.
 2. Usar o máximo de RAM possível. Fazer os runs mais longos e proporcionar mais, ou maiores, buffers durante a fase do merge.
 3. Se o nro de runs iniciais é muito grande, usar um multi-step merge. Pode diminuir o nro de seeks consideravelmente.
 4. Considerar a utilização de “replacement selection” para formar os runs iniciais, especialmente se existe a possibilidade que os runs estejam parcialmente ordenados.



Sorting arquivos grandes (MergeSort)

5. Usar mais do que um disco e canal de I/O de modo que leitura e escrita possam se sobrepôr. Isto é especialmente verdade se não existem outros usuários no sistema.
6. Ter em mente os elementos fundamentais para o External Sorting e seus custos. Buscar maneiras de tirar vantagem de novas arquiteturas e sistemas, como processamento paralelo e redes locais de alta velocidade.