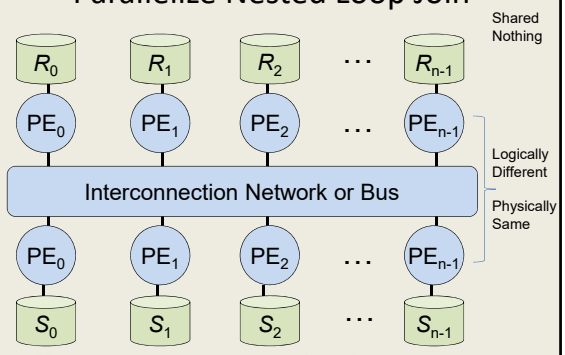
Parallelize Join Operations

- Remember Join Algorithms
  - Nested Loop Join
  - Sort Merge Join
  - Hash Join
    - Simple Hash Join
    - GRACE Hash Join
    - Hybrid Hash Join
- Here, we consider how to parallelize these algorithms

Parallelize Nested Loop Join

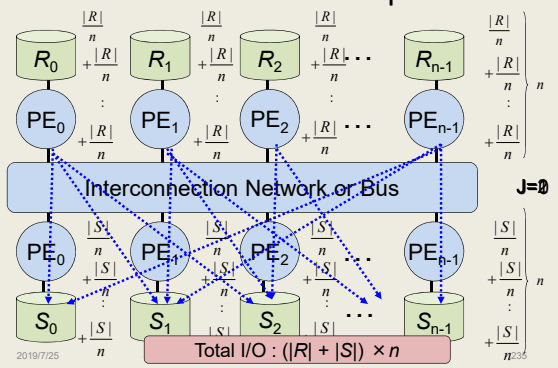


Parallelize Nested Loop Join

- Both relations are horizontally fragmented
  - $R_0, R_1, \ldots, R_{n-1}$
  - $S_0, S_1, \ldots, S_{n-1}$
  - independent of target attribute value
- Place both fragmented relations in each PE
  - S_i and R_i are placed in i-th PE
- Outer Loop: ($0 \leq j \leq n-1$)
  - Send R_i to *mod(i+j, n)*
    - Read $|R|/n$ page n times in each PE: $|R|$
- Inner Loop:
  - Do Join Operation between S_i and received relation
    - If each PE has enough memory
    - Read $|S|/n$ page n times in each PE: $|S|$
- Total I/O : $(|R| + |S|) \times n$

Enlarge size increase total costs
Parallel processing has no effects

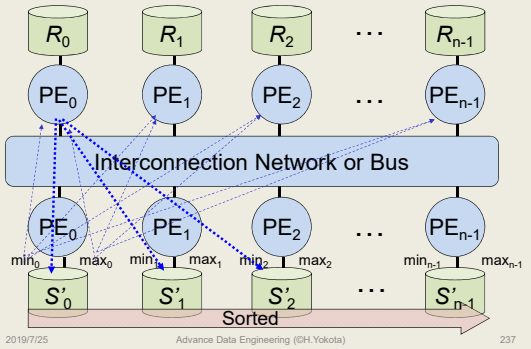
Parallelize Nested Loop Join



Parallelize Sort Merge Join (1)

- Assumption: Both relations are fragmented
- Type 1
  - Sort one relation in parallel for the target attribute
    - $S'_0, S'_1, \ldots, S'_{n-1}$
  - Broadcast maximum and minimum values in S'_i
  - Send each tuple of R_i to a PE correspond to the value
    - Sort all received tuples in the PE
  - Do Sort-Merge Join in each PE

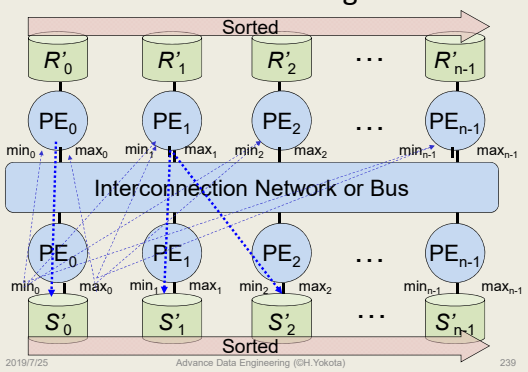
Parallelize Sort Merge Join



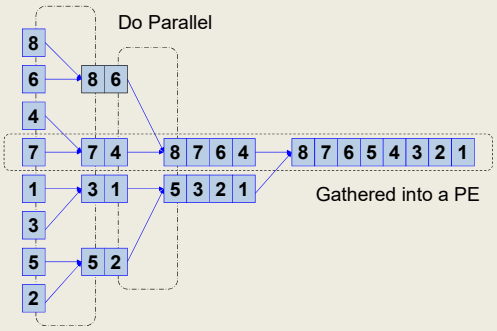
Parallelize Sort Merge Join (2)

- Type 2
  - Sort both relations in parallel for the target attribute
    - $R'_0, R'_1, ..., R'_{n-1}$
    - $S'_0, S'_1, ..., S'_{n-1}$
  - Broadcast maximum and minimum values in S'_i
  - Send each tuple of R'_j to a PE correspond to the value
    - R'_j may be sent to multiple PEs
  - Do Sort-Merge Join in each PE
  - Disk I/O: |R|/n + |S|/n, Total Disk I/O |R|+|S|
- Parallel Sort Algorithm
  - There are so many parallel sort algorithm

Parallelize Sort Merge Join



Parallel Merge Sort (1)



Parallel Merge Sort (2)

- Construct *log*({*R*}) stages for sorting a stream
  - Use {R}/2 PEs at first
  - Finally, all tuples are gathered into a PE
  - Comparisons in each stage can be done in parallel
- Cost for sorting {R} = 2^m tuples
  - Communication paths: $2^m + 2^{m-1} + \ldots + 2^1 = 2 \times (\{R\} - 1)$
  - Data transfer: $m \times \{R\} = \{R\} log_2 \{R\}$
  - Total comparison: $((1+1-1) \times \{R\}/2) + ((2+2-1) \times \{R\}/2^2 + \ldots + (((\{R\}/2+\{R\}/2-1) \times \{R\}/2^m) = \{R\}\log_2\{R\} - (\{R\} - 1)$
  - By parallel processing (time for corresponding): $(2-1) + (2^2-1) + \ldots + (2^m-1) = 2(\{R\} - 1) - \log_2\{R\}$

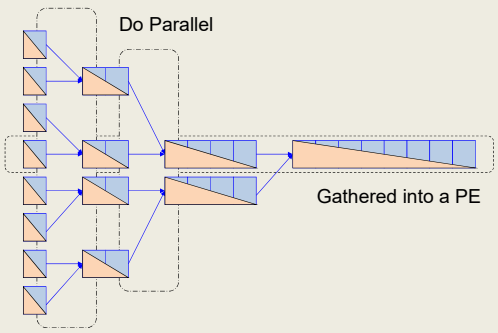
2019/7/25 Advance Data Engineering (©H.Yokota) 241

Costs for Parallel Block Merge Sort

- For sorting tuples larger than the number of PEs
  - Using $N = 2^n$ PEs, $n \geq 1$ ($N \geq 2$)
- A comparison operation is replaced by a stream merge
- Total Data transfer: n x $\{R\} = log_2 N \times \{R\}$
  - By parallel processing (time for corresponding)
    $(1/N + 2/N + \ldots + 1/2) \times \{R\}$
    $= (1-(1/2)^{n+1})/(1-1/2) \times \{R\}$
    $=(2- 1/N) \times \{R\}$ (how many tuples are transferred)

2019/7/25 Advance Data Engineering (©H.Yokota) 242

Parallel Block Merge Sort



2019/7/25 Advance Data Engineering (©H.Yokota) 243

Parallel Block Merge Sort (2)

- If disks are used in each stage
  - Total I/O : $2n \times |R| = 2 \log_2 N \times |R|$
  - the i-th stage: read $|R|/2^{n-(i-1)}$, write $|R|/2^{n-i}$
  - By parallel processing (time for corresponding)
    $\Sigma^n_{i=1} |R|/2^i + \Sigma^n_{i=1} |R|/2^{i-1}$
    $= ((1-(1/2)^n)/(1-1/2)) \times |R| + 2((1-(1/2)^n)/(1-1/2)) \times |R|$
    $= 6 \times (1 - 1/N) \times |R|$
  - Expect Pipeline effect

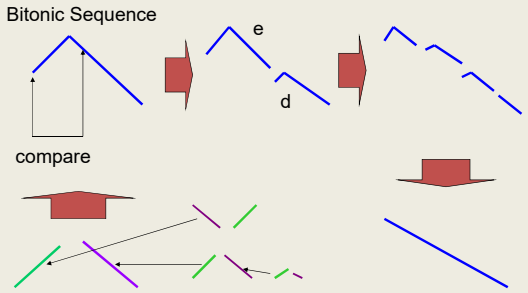
Bitonic Sort (1)

- Bitonic Sequence
  - There exist $1 \leq j \leq 2n$ which satisfies
    - $a_1 \leq a_2 \leq \ldots \leq a_j \geq a_{j+1} \geq \ldots \geq a_{2n}$
- For a bitonic sequence $a_1, a_2, \ldots, a_{2n}$
  - Let $d_i = \min(a_i, a_{n+i})$ and $e_i = \max(a_i, a_{n+i})$ where $1 \leq i \leq n$
  - Then $d_1, d_2, \ldots, d_n$ and $e_1, e_2, \ldots, e_n$ are also bitonic sequences
  - And $\max(d_1, d_2, \ldots, d_n) \leq \min(e_1, e_2, \ldots, e_n)$

Bitonic Sort (2)

- If there is a bitonic sequence, ordered small (half size) bitonic sequence can be generated by exchanging elements in a distance
  - Finally, the small bitonic sequence becomes an element
  - That is the result of the sort operation
- A Problem: How to generate the first bitonic sequence
  - Answer: Concatenation of two half size sorted sequence
  - Recursively continue until an element

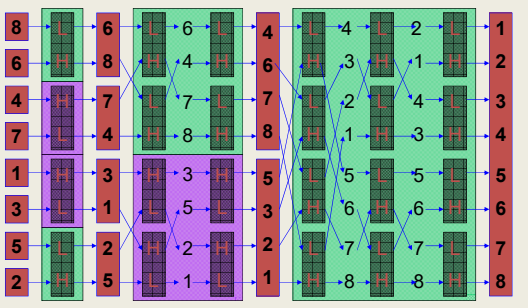
Bitonic Sort (3)



Parallel Bitonic Sort (1)

- A relation is horizontally fragmented
  - $R_0, R_1, \ldots, R_{n-1}$
- Exchanging elements in a distance can be done in parallel
- Results are also fragmented
  - There is no bottleneck like parallel merge sort

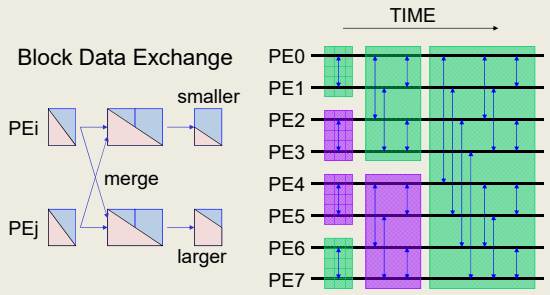
Parallel Bitonic Sort (2)



Cost for Parallel Bitonic Sort

- The number of stages for generating bitonic sequence:
  - $log_2\{R\}=m$
- The number of stages for 2^i length bitonic sequence:
  - $log_2\ 2^i = i$
- Total stages:
  - $\Sigma^m_{i=1} log_2\ 2^i = \Sigma^m_{i=1} i = m(m+1)/2 = (log_2\{R\}(log_2\{R\}+1)) / 2$
- Total data transfer:
  - $\{R\} \times ((log_2\{R\}(log_2\{R\}+1)) / 2)$
- Total data exchanges:
  - $(1/2) \times \{R\} \times ((log_2\{R\}(log_2\{R\}+1)) / 2)$
- By parallel processing (time for corresponding):
  - $(log_2\{R\}(log_2\{R\}+1)) / 2$

Parallel Block Bitonic Sort



Cost for Parallel Block Bitonic Sort

- For sorting tuples larger than the number of PEs ($N = 2^n$)
- Merge two streams and leave smaller (larger) part
- Total stages: $\Sigma^n_{i=1} i = n(n+1) / 2$
- Total data transfer: $\{R\}$ x $n(n+1) / 2$
- By parallel processing (time for corresponding):
  - $\{R\}$ x $n(n+1) / 2N$
- Total I/O (read and write for each stage) :
  - $|R|$ x $n(n+1)$
- By parallel processing (time for corresponding):
  - $|R|/N$ x $n(n+1) = (log_2N(log_2N+1)) / N\}$ x $|R|$