

## Parallelize Join Operations

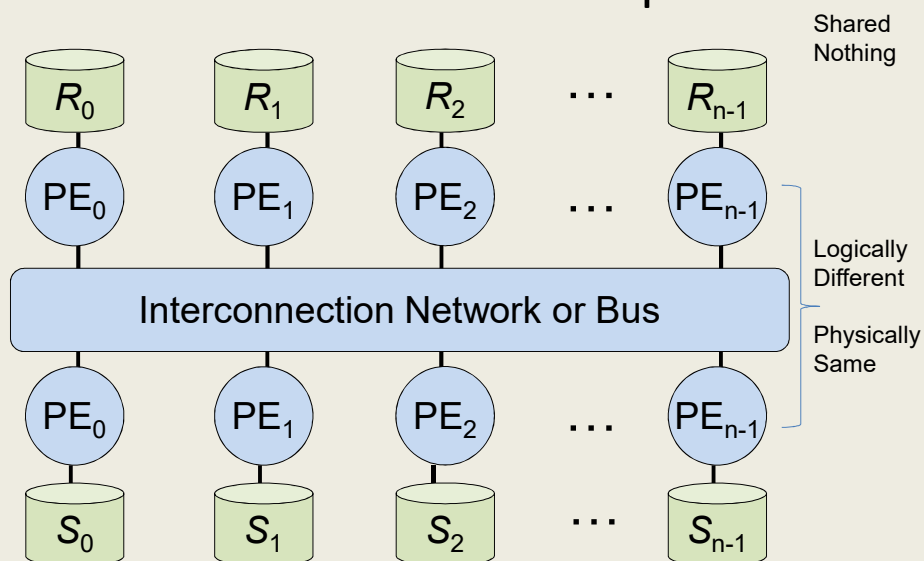
- Remember Join Algorithms
  - Nested Loop Join
  - Sort Merge Join
  - Hash Join
    - Simple Hash Join
    - GRACE Hash Join
    - Hybrid Hash Join
- Here, we consider how to parallelize these algorithms

2020/7/23

Advance Data Engineering (©H.Yokota)

232

## Parallelize Nested Loop Join



2020/7/23

Advance Data Engineering (©H.Yokota)

233

## Parallelize Nested Loop Join

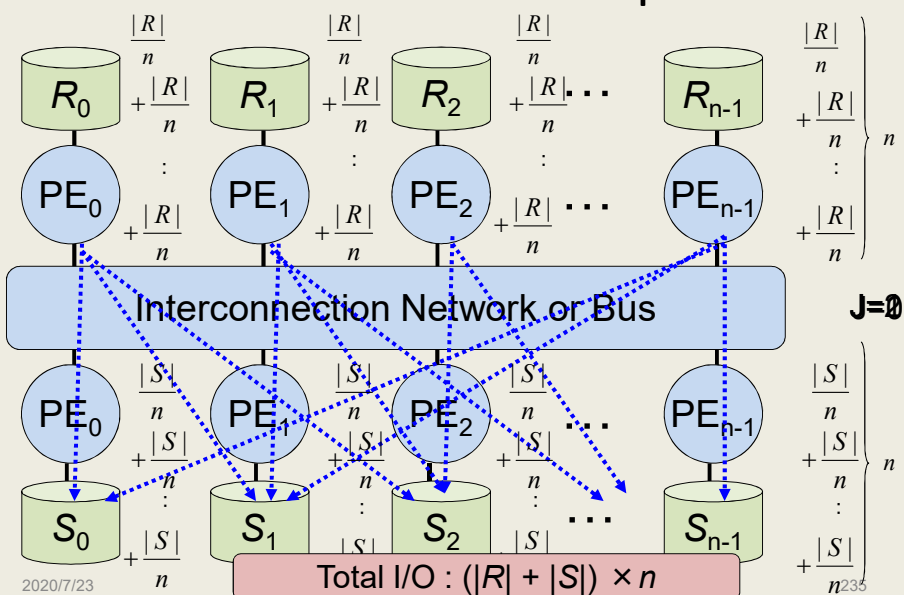
- Both relations are horizontally fragmented
  - $R_0, R_1, \dots, R_{n-1}$
  - $S_0, S_1, \dots, S_{n-1}$
  - independent of target attribute value
- Place both fragmented relations in each PE
  - $S_i$  and  $R_i$  are placed in  $i$ -th PE
- Outer Loop: ( $0 \leq j \leq n-1$ )
  - Send  $R_j$  to  $\text{mod}(i+j, n)$ 
    - 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

2020/7/23

Advance Data Engineering

## Parallelize Nested Loop Join



## Parallelize Sort Merge Join (1)

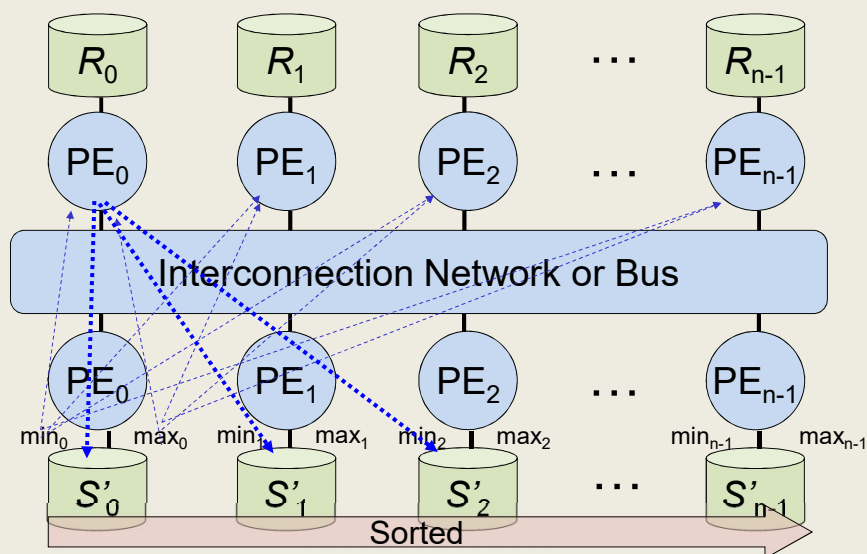
- Assumption: Both relations are fragmented
- Type 1
  - Sort one relation in parallel for the target attribute
    - $S'_0, S'_1, \dots, S'_{n-1}$
  - Broadcast maximum and minimum values in  $S'_i$
  - Send each tuple of  $R_i$  to a PE correspond to the value
    - Sort all received tuples in the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

2020/7/23

Advance Data Engineering (©H.Yokota)

236

## Parallelize Sort Merge Join



2020/7/23

Advance Data Engineering (©H.Yokota)

237

## Parallelize Sort Merge Join (2)

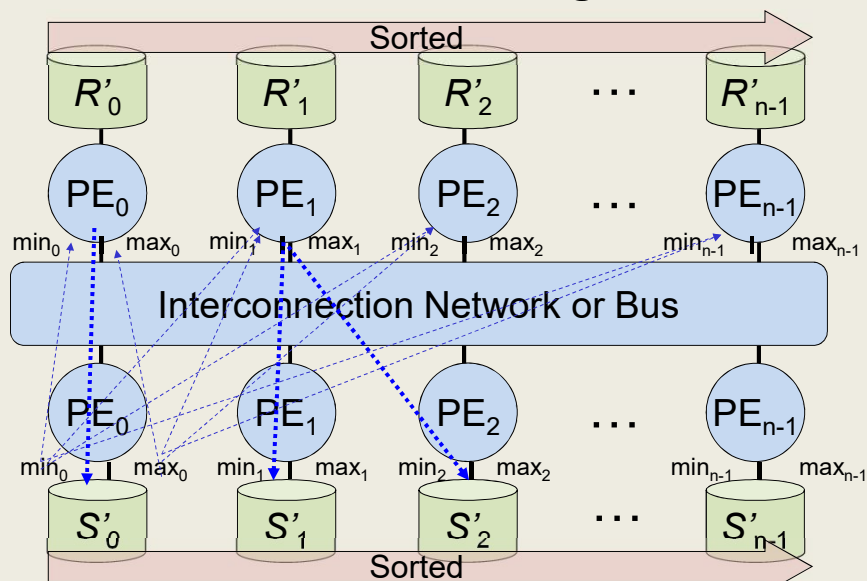
- Type 2
  - Sort both relations in parallel for the target attribute
    - $R'_0, R'_1, \dots, R'_{n-1}$
    - $S'_0, S'_1, \dots, S'_{n-1}$
  - Broadcast maximum and minimum values in  $S'_i$
  - Send each tuple of  $R'_j$  to a PE correspond to the value
    - $R'_j$  may be sent to multiple PEs
  - 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

2020/7/23

Advance Data Engineering (©H.Yokota)

238

## Parallelize Sort Merge Join

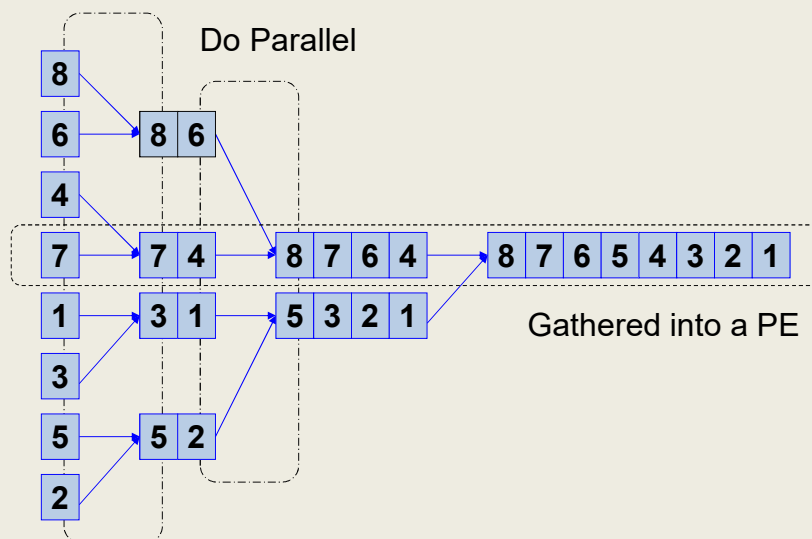


2020/7/23

Advance Data Engineering (©H.Yokota)

239

## Parallel Merge Sort (1)



2020/7/23

Advance Data Engineering (©H.Yokota)

240

## 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:  $2^m + 2^{m-1} + \dots + 2^1 = 2 \times (\{R\} - 1)$
  - Data transfer:  $m \times \{R\} = \{R\} \log_2 \{R\}$
  - Total comparison:  $((1+1-1) \times \{R\}/2) + ((2+2-1) \times \{R\}/2^2 + \dots + ((\{R\}/2 + \{R\}/2-1) \times \{R\}/2^m) = \{R\} \log_2 \{R\} - (\{R\} - 1)$
  - By parallel processing (time for corresponding):  $(2-1) + (2^2-1) + \dots + (2^m-1) = 2(\{R\} - 1) - \log_2 \{R\}$

2020/7/23

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:  $n \times \{R\} = \log_2 N \times \{R\}$ 
  - By parallel processing (time for corresponding)
 
$$(1/N + 2/N + \dots + 1/2) \times \{R\}$$

$$= (1 - (1/2)^{n+1}) / (1 - 1/2) \times \{R\}$$

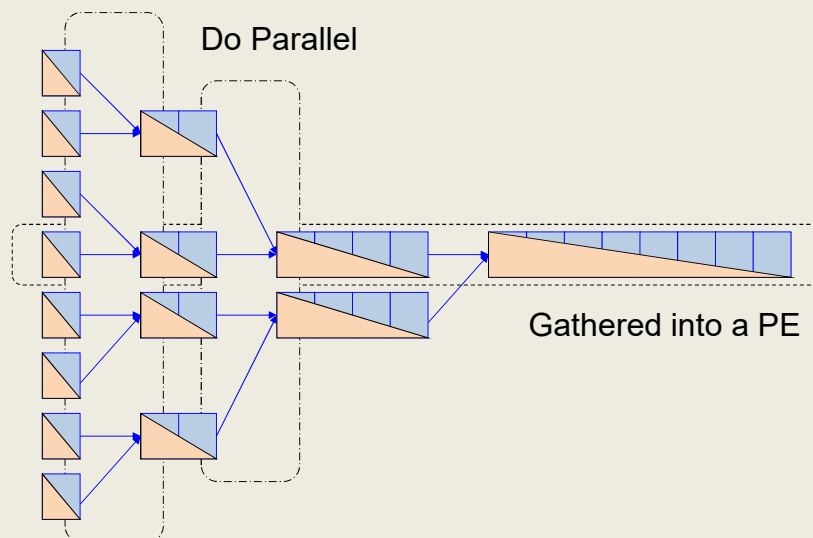
$$= (2 - 1/N) \times \{R\} \quad (\text{how many tuples are transferred})$$

2020/7/23

Advance Data Engineering (©H.Yokota)

242

## Parallel Block Merge Sort



2020/7/23

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)
 
$$\sum_{i=1}^n |R|/2^i + \sum_{i=1}^n |R|/2^{i-1}$$

$$= ((1-(1/2)^n)/(1-1/2)) \times |R| + 2((1-(1/2)^n)/(1-1/2)) \times |R|$$

$$= 6 \times (1 - 1/N) \times |R|$$
  - Expect Pipeline effect

2020/7/23

Advance Data Engineering (©H.Yokota)

244

## Question (9-1)

- Roughly estimate the execution time for parallel block 2way merge sort using 8 and 16 processors, where
  - Cardinality of a relation R: 100,000
  - Total length of a tuple: 100B
  - Disk transfer bandwidth: 10MB/s
  - Network (connection) bandwidth: 10MB/s
  - Ignore CPU costs, disk access latency, and network setup time

2020/7/23

Advance Data Engineering (©H.Yokota)

245

## Bitonic Sort (1)

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

2020/7/23

Advance Data Engineering (©H.Yokota)

246

## 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

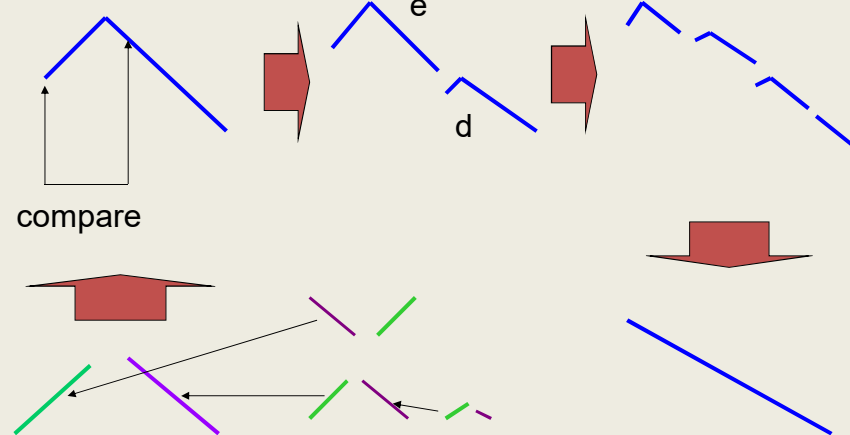
2020/7/23

Advance Data Engineering (©H.Yokota)

247

## Bitonic Sort (3)

Bitonic Sequence



2020/7/23

Advance Data Engineering (©H.Yokota)

248

## Parallel Bitonic Sort (1)

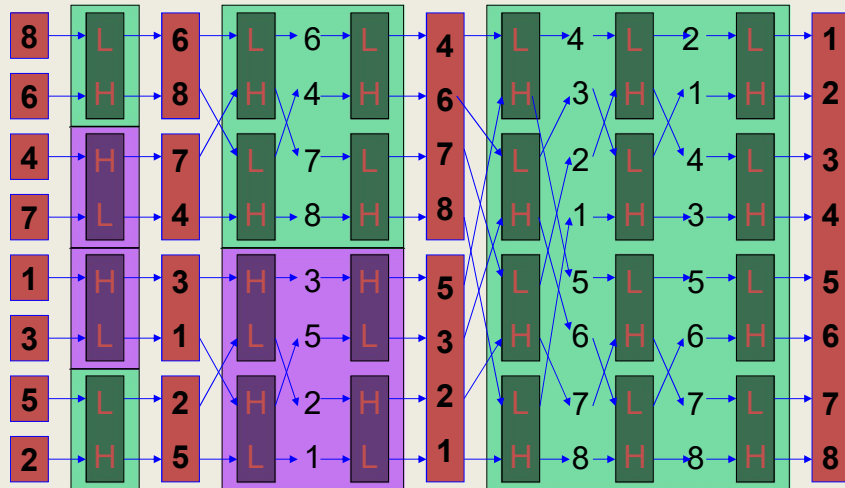
- A relation is horizontally fragmented
  - $R_0, R_1, \dots, R_{n-1}$
- Exchanging elements in a distance can be done in parallel
- Results are also fragmented
  - There is no bottleneck like parallel merge sort

2020/7/23

Advance Data Engineering (©H.Yokota)

249

## Parallel Bitonic Sort (2)



2020/7/23

Advance Data Engineering (©H.Yokota)

250

## Cost for Parallel Bitonic Sort

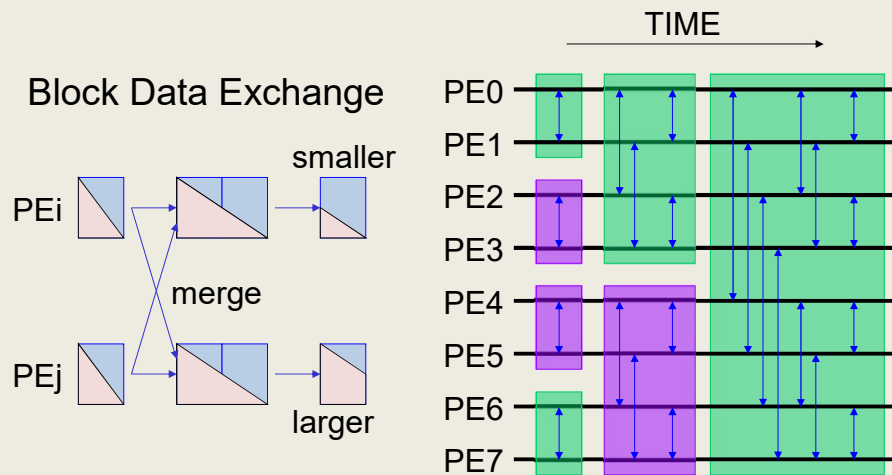
- The number of stages for generating bitonic sequence:
  - $\log_2 \{R\} = m$
- The number of stages for  $2^i$  length bitonic sequence:
  - $\log_2 2^i = i$
- Total stages:
  - $\sum_{i=1}^m \log_2 2^i = \sum_{i=1}^m i = m(m+1)/2 = (\log_2 \{R\}(\log_2 \{R\} + 1)) / 2$
- Total data transfer:
  - $\{R\}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

2020/7/23

Advance Data Engineering (©H.Yokota)

251

## Parallel Block Bitonic Sort



2020/7/23

Advance Data Engineering (©H.Yokota)

252

## 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:  $\sum_{i=1}^n i = n(n+1) / 2$
- Total data transfer:  $\{R\} \times n(n+1) / 2$
- By parallel processing (time for corresponding):
  - $\{R\} \times n(n+1) / 2N$
- Total I/O (read and write for each stage) :
  - $|R| \times n(n+1)$
- By parallel processing (time for corresponding):
  - $|R| / N \times n(n+1) = (\log_2 N (\log_2 N + 1)) / N \times |R|$

2020/7/23

Advance Data Engineering (©H.Yokota)

253

## Question (9-2)

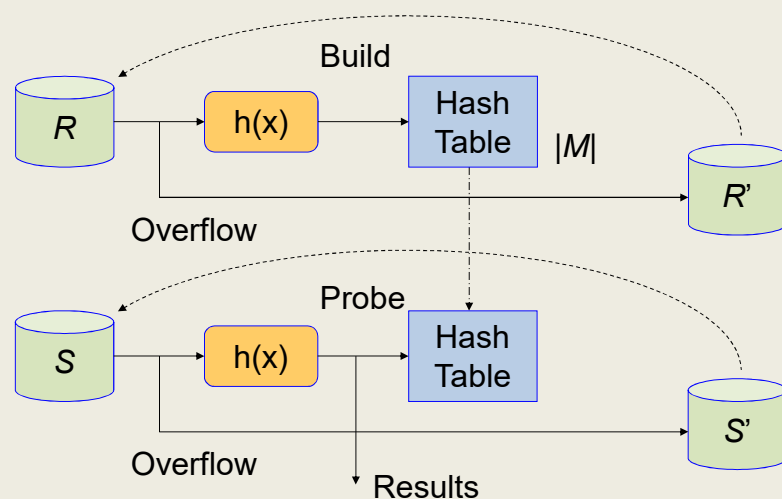
- Roughly estimate the execution time for parallel block bitonic sort using 8 and 16 processors, where
  - Cardinality of a relation R: 100,000
  - Total length of a tuple: 100B
  - Disk transfer bandwidth: 10MB/s
  - Network bandwidth: 10MB/s
  - Ignore CPU costs, disk access latency, and network setup time

2020/7/23

Advance Data Engineering (©H.Yokota)

254

## Illustration of Simple Hash Join



2020/7/23

Advance Data Engineering (©H.Yokota)

255

## Parallelize Simple Hash Join

- Where can we parallelize simple hash join?
- Loop ?
  - There are dependencies in iteration
    - (i-1)-th results are used for i-th iteration
- A step of the iteration ?
  - That IS Join
    - The Original Problem
- It is hard to parallelize simple hash join

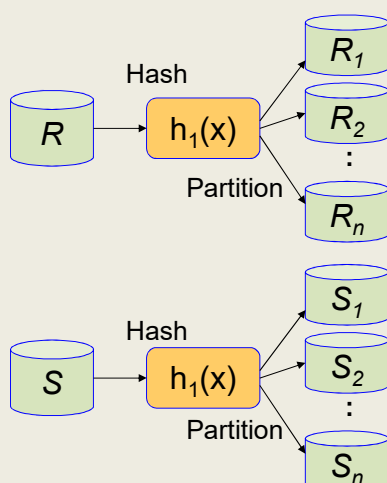
2020/7/23

Advance Data Engineering (©H.Yokota)

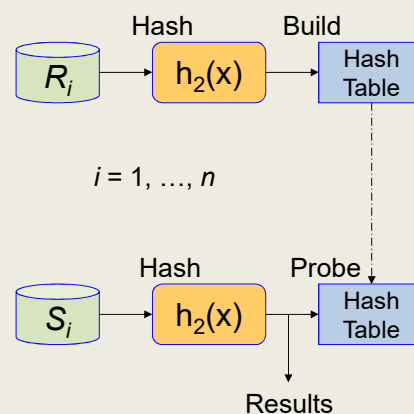
256

## Illustration of GRACE Hash Join

Partitioning Phase



Join Phase



2020/7/23

Advance Data Engineering (©H.Yokota)

257

## Parallelize GRACE Hash Join

- Parallelize Bucket Decomposition in Phase 1
  - Each relation is partitioned in advance
    - The average number of tuples in each disk  $\{R\}/N$  and  $\{S\}/N$
    - Selection operations are also executed in advance in parallel
  - Virtually divide connected disk into Read Disk and Write Disk
  - Assign each bucket to each processor
    - Send each tuple by its hash value
- Parallelize Join Operation in Phase 2
  - There is no communication within Phase 2

2020/7/23

Advance Data Engineering (©H.Yokota)

258

## Pseudo Code for Phase1

- Each PE has two threads:
  - Thread 1:
 

```

for ( $j = 1; j \leq \{R/N\}; j++$ ) {
  read  $j$ -th tuple  $t$  and attribute value  $v$  in  $t$ ;
   $x = h0(v)$ ; /* e.g.  $h(v) = \text{mod}(v, N)$  */
  send  $t$  to  $PE_x$ 
}
```
  - Thread 2
 

```

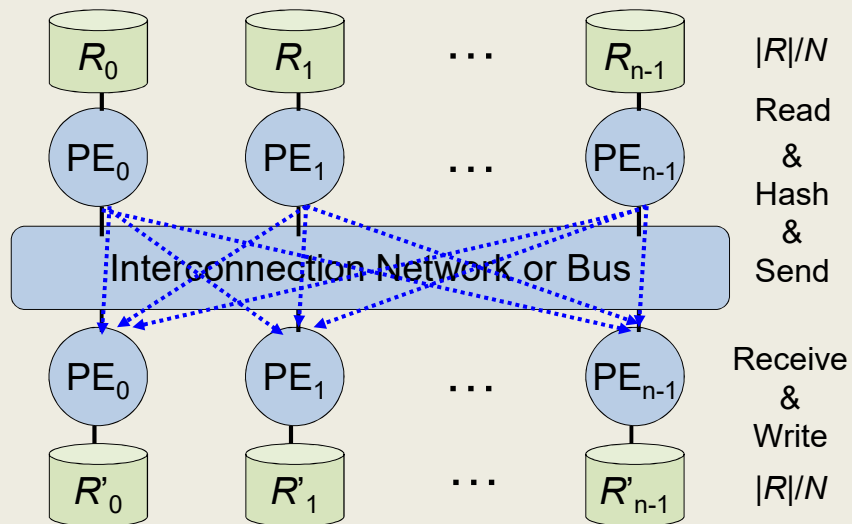
for (;;) {
  receive  $t$  and attribute value  $v$  in  $t$ ;
   $y = h1(v)$ ;
  write  $t$  into a file for  $y$ 
}
```
- It should be combined with the phase switch of all-to-all communication

2020/7/23

Advance Data Engineering (©H.Yokota)

259

## Parallel GRACE Hash Join (1)

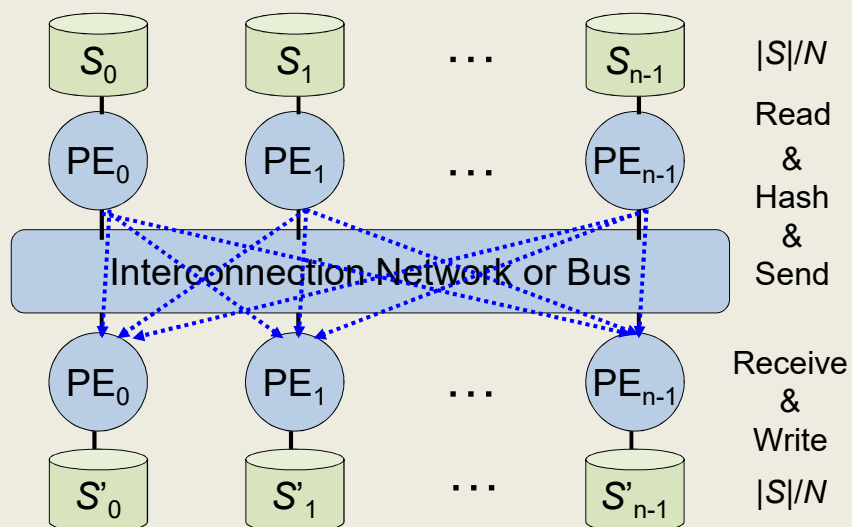


2020/7/23

Advance Data Engineering (©H.Yokota)

260

## Parallel GRACE Hash Join (2)



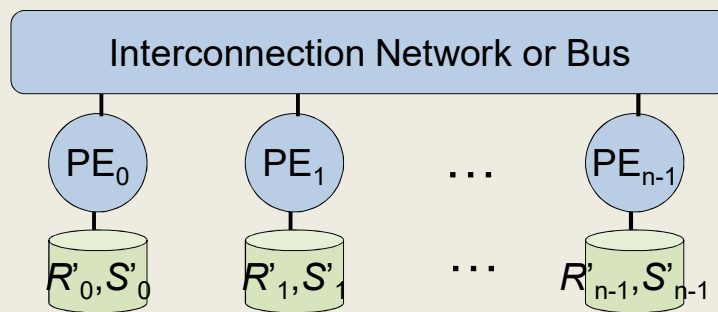
2020/7/23

Advance Data Engineering (©H.Yokota)

261

## Parallel GRACE Hash Join (3)

### Phase 2



Read & Local Join

$$(|R| + |S|) / N$$

2020/7/23

Advance Data Engineering (©H.Yokota)

262

## Cost of Parallel GRACE Hash Join

- Phase 1
  - $(|R| + |S|) / N$  read and  $(|R| + |S|) / N$  write in each PE
  - With all-to-all communication cost:  $\alpha$
- Phase 2
  - $(|R| + |S|) / N$  read in each PE
  - No communication
- Total I/O in each PE
  - $3 \times (|R| + |S|) / N$
  - It means  $(1/N + \alpha)$ , if there is no skew

2020/7/23

Advance Data Engineering (©H.Yokota)

263

## Estimate $\alpha$ (1/3)

- Bandwidth of each connection of network: 10MB/s
- Network setup time for each connection: 50  $\mu$ s
- $|R|$  and  $|S|$ : 64MB each
- The number of processors (N): 8
- Consider the cost for communication, assuming each processing element has enough large buffer space to keep  $|R|/N$  or  $|S|/N$
- Also consider the cost when each processing element has memory for two pages (8KB)

2020/7/23

Advance Data Engineering (©H.Yokota)

264

## Estimate $\alpha$ (2/3)

- When we have enough memory, we can reduce the number of communication into the number of processing elements N, i.e., 8.
- Each processing element has  $|R|/N$  and  $|S|/N$  data, i.e.,  $X = 64\text{MB}/8 = 8\text{ MB}$  for each relation..
- In each communication phase, each processing element send  $X/N$  data, i.e.  $Y = 8\text{MB}/8 = 1\text{MB}$
- Data transfer time is  $Z = Y/10\text{MB/s} = 100\text{ms}$
- Network setup time is 50  $\mu$ s which can negligible in this case.
- Since we have 8 phases for each relation, the cost for communication is  $8 \times 2 \times Z = 1.6\text{s}$

2020/7/23

Advance Data Engineering (©H.Yokota)

265

## Estimate $\alpha$ (3/3)

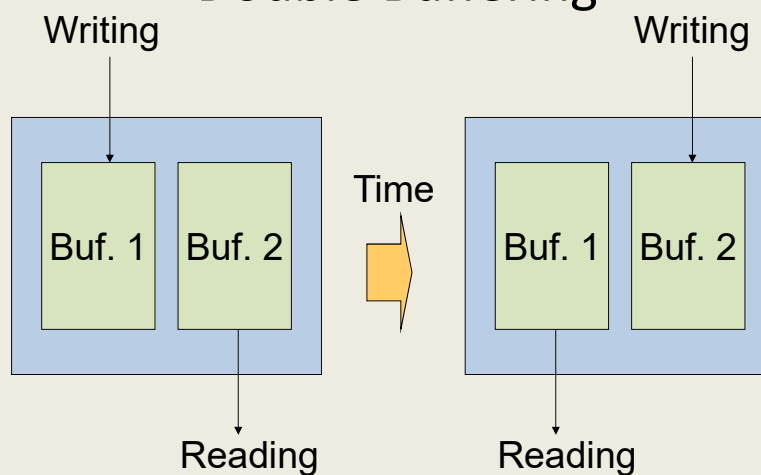
- When we have memory for only two pages (8KB), each processing element has to send page by page.
- We can apply the double buffering method
- Network setup is required for each buffer switch
- Since each processing element has to send 8MB data for each relation,  $8\text{MB}/4\text{KB} = 2\text{K}$  buffer switch occurs.
- The cost for network setup is  $50\ \mu\text{s} \times 2\text{K} = 100\text{ms}$  for each relation
- Total cost is  $1.6\text{s} + 0.2 = 1.8\text{s}$

2020/7/23

Advance Data Engineering (©H.Yokota)

266

## Double Buffering



- It enables simultaneous reading and writing

2020/7/23

Advance Data Engineering (©H.Yokota)

267

## Question (9-3)

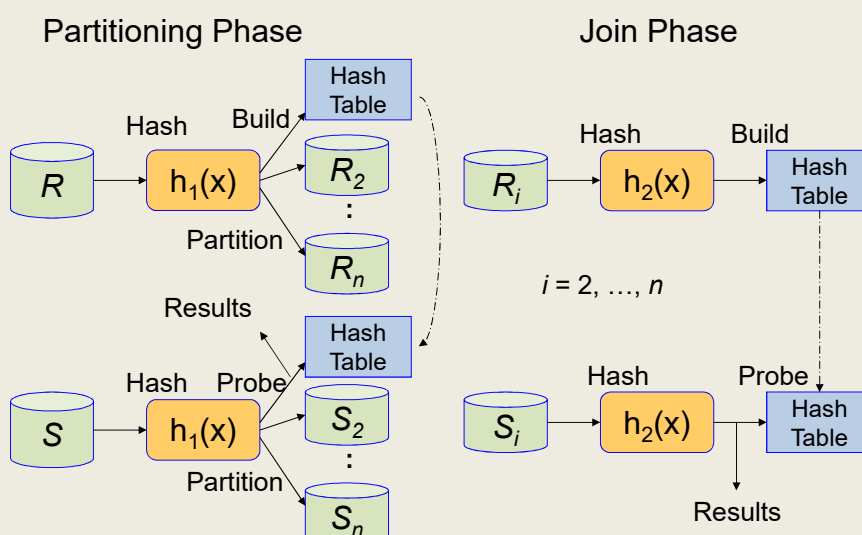
- Estimate the execution time for the GRACE Hash Join with considering the network communication cost under the assumption of:
  - Having enough memory
  - Using **8MB/s** bandwidth disks
  - Network bandwidth is 10MB/s
  - Ignoring seek time, rotational latency, network setup time
- Estimate the execution time for increasing the number of PEs from 8 to 16

2020/7/23

Advance Data Engineering (©H.Yokota)

268

## Illustration of Hybrid Hash Join



2020/7/23

Advance Data Engineering (©H.Yokota)

269

## Parallelize Hybrid Hash Join

- Consider how to parallelize the Hybrid hash join
- Pseudo Code

Thread 2

```
for (;;) {
    receive t and attribute value v in t;
    y = h1(v);
    if y is 0 then build (or probe) a hash table
    else write t into a file for y
}
```

2020/7/23

Advance Data Engineering (©H.Yokota)

270

## Multiple Joins

- There tend to be a number of join operations in a query
  - The query construct a query tree
  - The configuration of the tree deeply influence the performance of query processing
    - especially under parallel environment
- Parallel executions in a query tree
  - Independent executions
  - Pipeline executions
- Here, we assume hash join algorithm

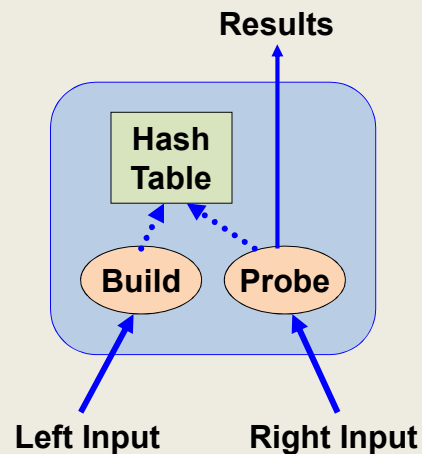
2020/7/23

Advance Data Engineering (©H.Yokota)

271

## Notation of a Hash Join Node

- Let left input is for **Build** and right for **Probe**
  - Hash Join cannot start Probe until the Build Phase is finished

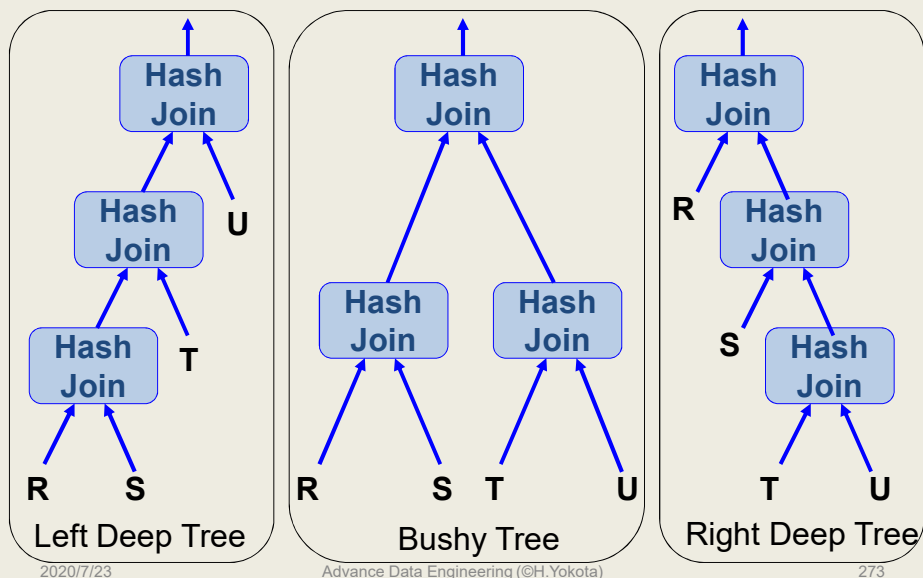


2020/7/23

Advance Data Engineering (©H.Yokota)

272

## Query Tree Configurations



2020/7/23

Advance Data Engineering (©H.Yokota)

273

## Parallel Multiple Join

- Parallel execution of multiple join is depend on the structure of the query tree
  - Left Deep Tree:
    - Sequential
  - Right Deep Tree:
    - Pipeline Execution (Parallel Build)
  - Bushy Tree:
    - Child node can be executed in parallel
    - Pipeline Execution can also be done