IDUG
Leading the DB2 User Community since 1988

IDUG Db2 Tech Conference NA
Philadelphia, PA | April 29 - May 3, 2018

#IDUGDb2

The Db2 Parallel Universe

John Hornibrook

IBM Canada

Session code: C19

05/dd/2018, HH:MM-HH:MM

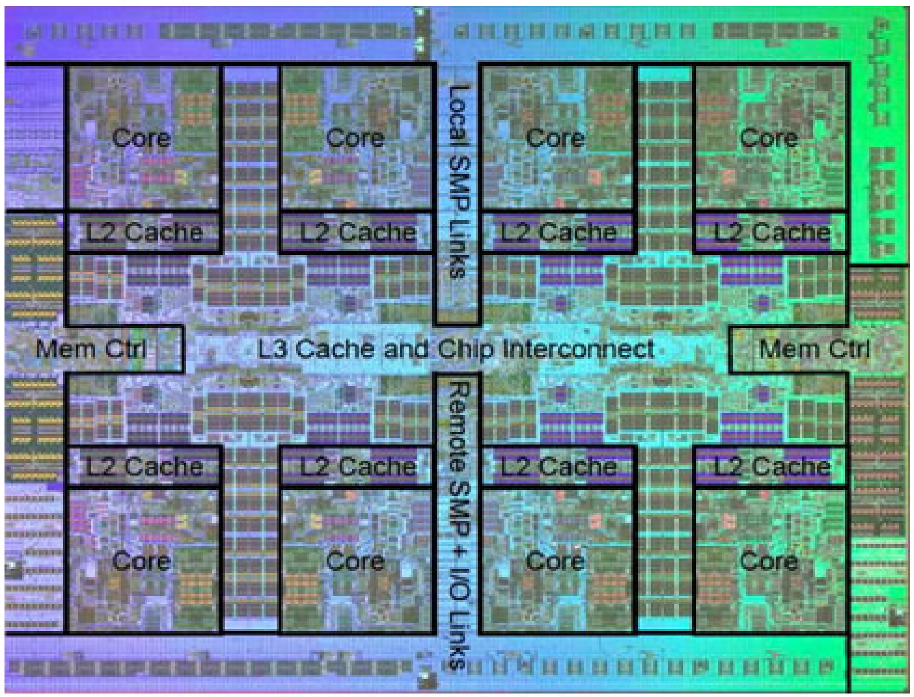
Db2

Agenda

- Query parallelism overview and concepts
- Row-organized query parallelism
- Column-organized query parallelism
- Query parallelism and the database partitioning feature (DPF)
- How the optimizer choose parallelization strategies
- Configuration
- Monitoring

Query Parallelism

SELECT..FROM.. WHERE | SELECT..FROM.. WHERE | SELECT..FROM.. WHERE | SELECT..FROM.. WHERE

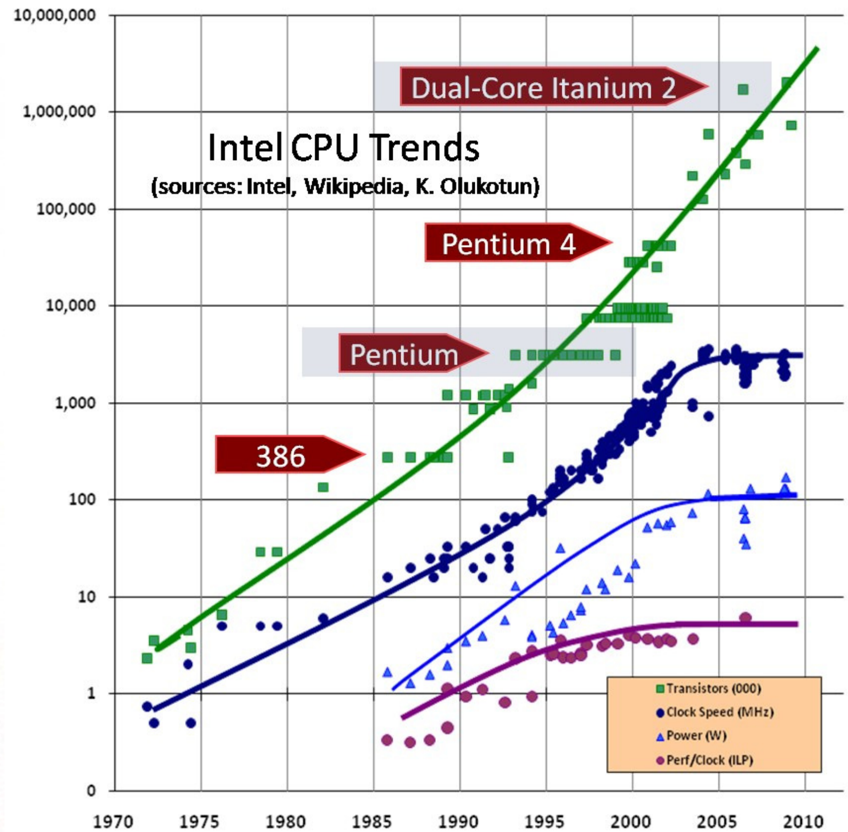


SELECT..FROM.. WHERE | SELECT..FROM.. WHERE | SELECT..FROM.. WHERE | SELECT..FROM.. WHERE

Query Parallelism

- Using multiple database subagents to execute a single SQL statement
- All modern systems have multiple cores (even your phone!)
- Increases in processor speed started to slow about 10 years ago
  - Limited by power consumption, heat dissipation and current leakage
  - Where are the 10GHz chips?
- So systems have more processors
- Modern SW must use parallelism to achieve performance improvements

CPUs aren't getting any faster…



CPU scaling showing transistor density, power consumption, and efficiency. Chart originally from [The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software]

2 Types of Query Parallelism in Db2 (1/2)

- Db2 parallelizes execution of a single SQL statement using 2 approaches:
  - **Inter**-partition parallelism
    - Occurs naturally with database (DB) partitioned tables
      - Db2 Database Partitioning Feature (DPF)
    - Statement must be executed on each of the table's DB partitions
    - Parallelism efficiency depends on:
      - Table's partitioning key
      - Other table's partitioning key (for joins)
      - Relational operations (joins, aggregation distinct, union, etc.)

2 Types of Query Parallelism in Db2 (2/2)

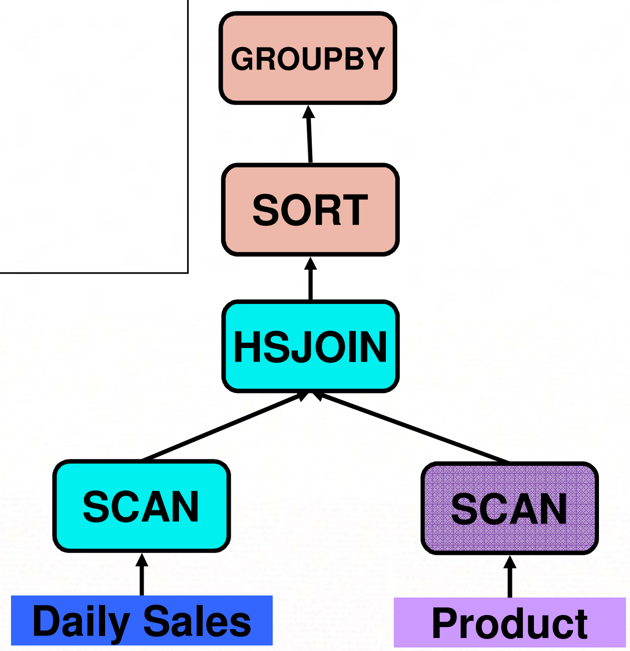
- Db2 parallelizes execution of a single SQL statement using 2 approaches:
  - **Inter**-partition parallelism
  - **Intra**-partition parallelism
    - Does not require tables to be partitioned (range or DB)
    - Can be used in combination with inter-partition parallelism
    - Used for both row and column-organized tables
    - Also known as 'multi-core parallelism'
  - Both approaches can be used together

Query Parallelization Techniques

- The **query** is sub-divided so that different pieces execute in parallel or are executed by multiple sub-agents
  - These 'pieces' are called **<u>subsections</u>**

Query Subdivision

```
SELECT SUM(S.QUANTITY_SOLD), P.CATEGORY
FROM
DAILY_SALES S,PRODUCT P
WHERE
S.PRODKEY=P.PRODKEY
GROUP BY P.CATEGORY
```



This query can be subdivided into 3 subsections:

1. Scan PRODUCT table and create hash table
2. Scan DAILY_SALES table and perform a hash join using the hash table from step 1.
3. Sort the result of the hash join and compute the SUM for each CATEGORY group.

Subsection 1 must complete before subsection 2 and 3.
Subsection 2 and 3 can execute in parallel.

Query Parallelization Techniques

- The **data** is sub-divided so that the same subsection can execute in parallel, on different pieces of the data
  - These 'pieces' of data are called **partitions**
  - Db2 partitions the data in a few ways:
    - Random
    - Hash
    - Range
- The data could also be replicated
  - Each subsection sees all the data
  - Will explain why this is useful later…

Data Subdivision

Data

1
1
2
3
3
4
5
6
7
8
8
10

Random

1	1	2	3
3	4	5	7
8	8	6	10

Hash

1	3	2	4
1	3	5	10
8	7	6	
8			

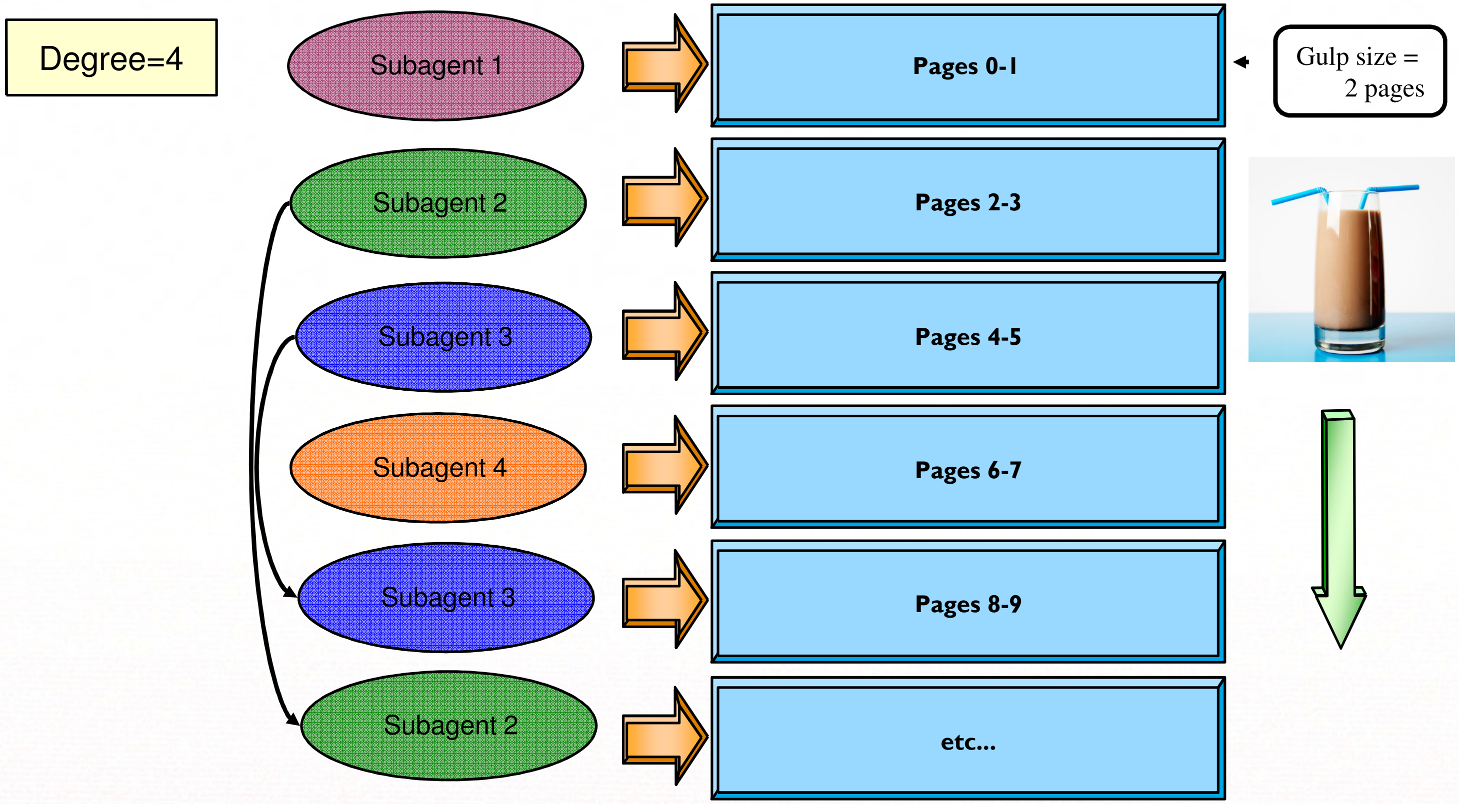
Range

1	3	5	8
1	3	6	8
2	4	7	10

Intra-Partition Query Parallelism Architecture

- Db2 doesn’t require tables to be pre-partitioned
- The data is dynamically partitioned when it is read from the table using a ‘straw scan’
  - Db2 determines optimal ‘gulp size’
  - Assigns gulps to subagents
    - Range of rows or pages
    - Multiple gulps for large tables to ensure load balance
    - Assign new range when range is consumed
    - Gulp size is adjusted dynamically to maintain load balance
  - Provides dynamic load balancing
  - Supports table and index scans
  - Used for row and column-organized tables

Dynamic Data Partitioning – “straw scans”



Dynamic Data Partitioning – "straw scans"

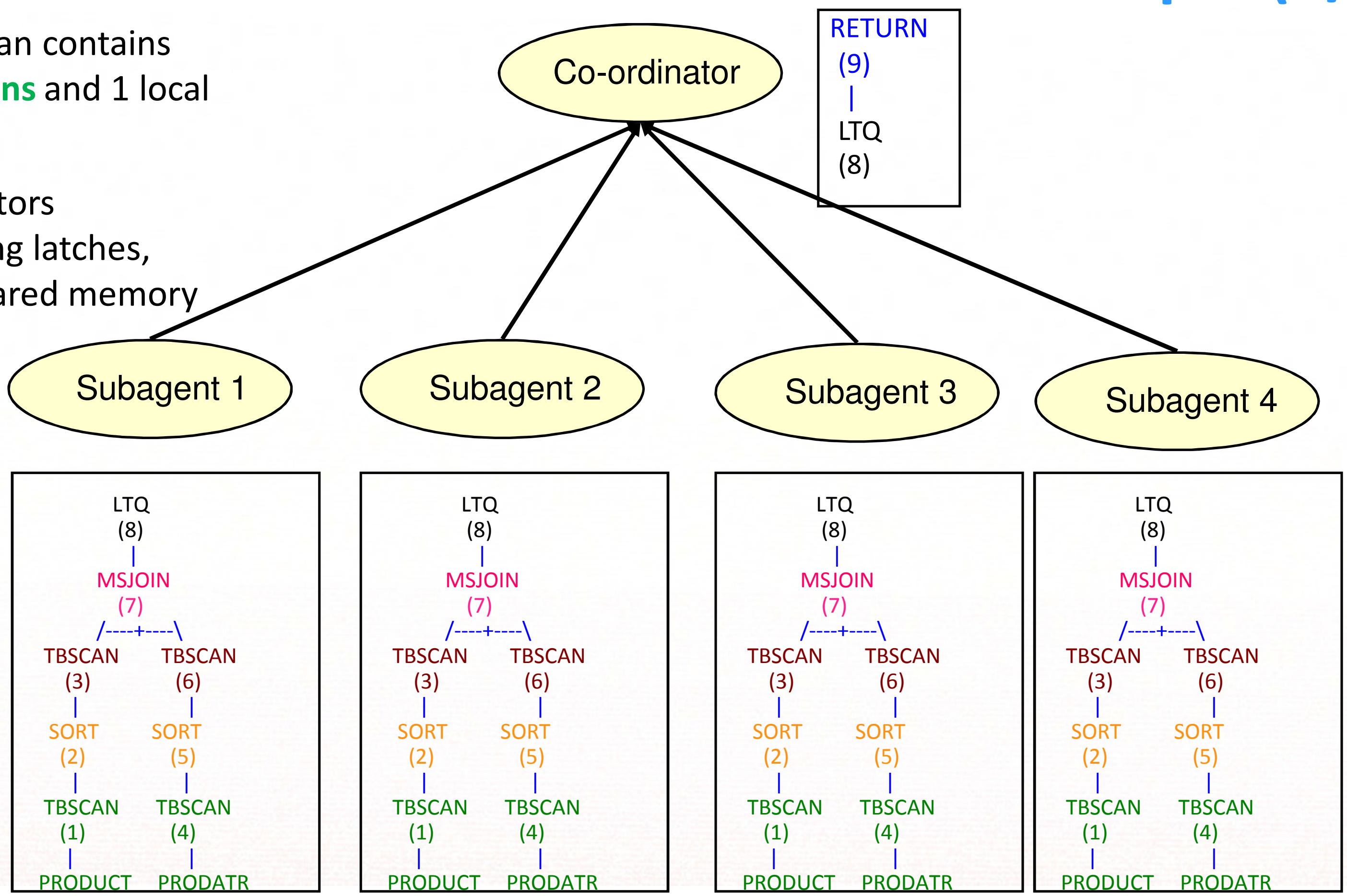
- Allows each subagent to get more work as needed
- Some subagents might get more "gulps" then others, but some gulps are easier to swallow
- The important thing is that all subagents (and cores) are busy and that each one does the same amount of work (load balance)

Row-organized Intra-Partition Parallelism Architecture

- Parallelize Query Execution
  - Query is processed in parallel by multiple subagents
  - Result set is returned to the co-ordinator agent
    - Via a “table queue”
    - A special Db2 pipe, with multiple writers (subagents) and 1 reader (co-ordinator agent)
    - There are many types of table queues (more later…)
  - Single co-ordinator agent services application requests

Row-organized Intra-Partition Parallelism Example (1/4)

- Every access plan contains **only 2 subsections** and 1 local table queue
- Runtime operators coordinated using latches, semaphores, shared memory controls blocks



Row-organized Intra-Partition Parallelism Example (2/4)

```
select p.name, p.prod_id, pa.attribute
from product p, prodatr pa
where p.prod_id = pa.prod_id;
```

```
              LTQ
              (8)
               |
            MSJOIN
              (7)
           /---+---\
      TBSCAN       TBSCAN
        (3)          (6)
         |            |
       SORT         SORT
        (2)          (5)
         |            |
      TBSCAN       TBSCAN
        (1)          (4)
         |            |
      PRODUCT      PRODATR
```

Results returned via shared memory table queue to co-ordinator agent

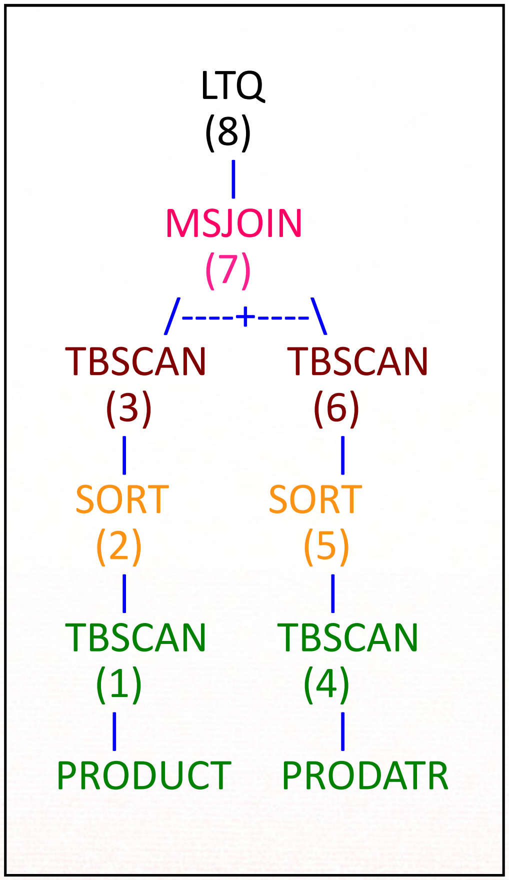
Join processed in parallel by each agent by joining corresponding partitions

Each agent scans a sort partition

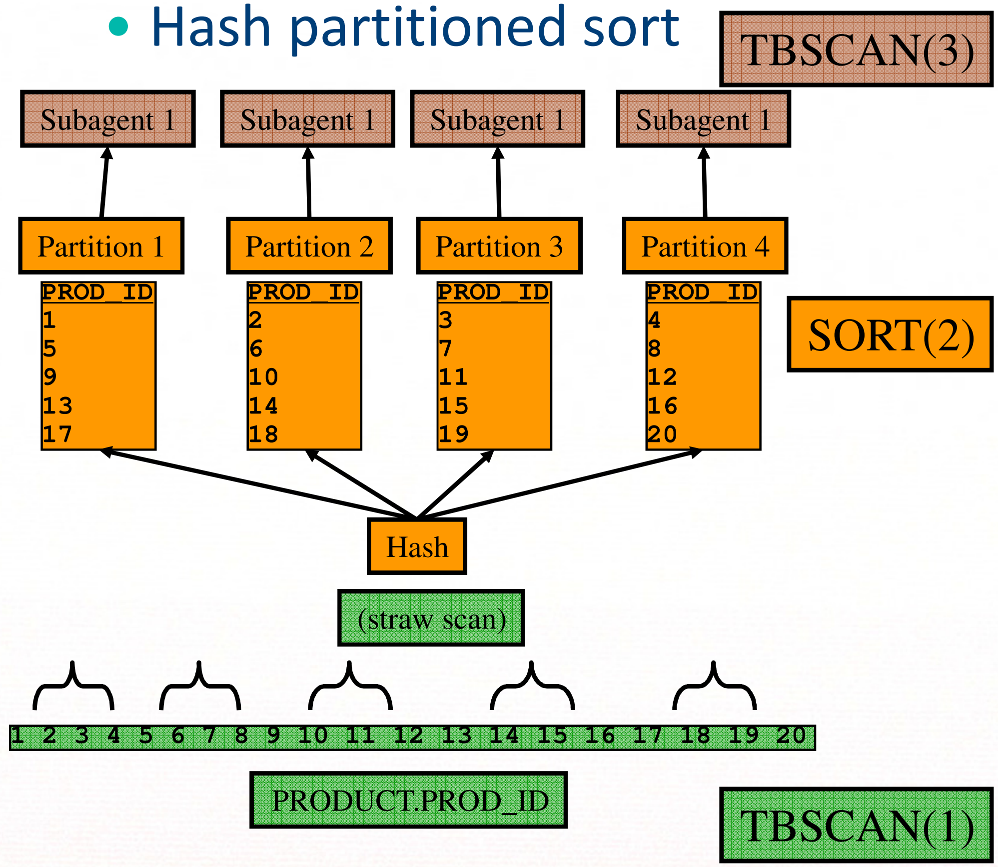
Hash partitioned SORTs on prod_id
- one partition per agent

Parallel table scans ("straw" scans)

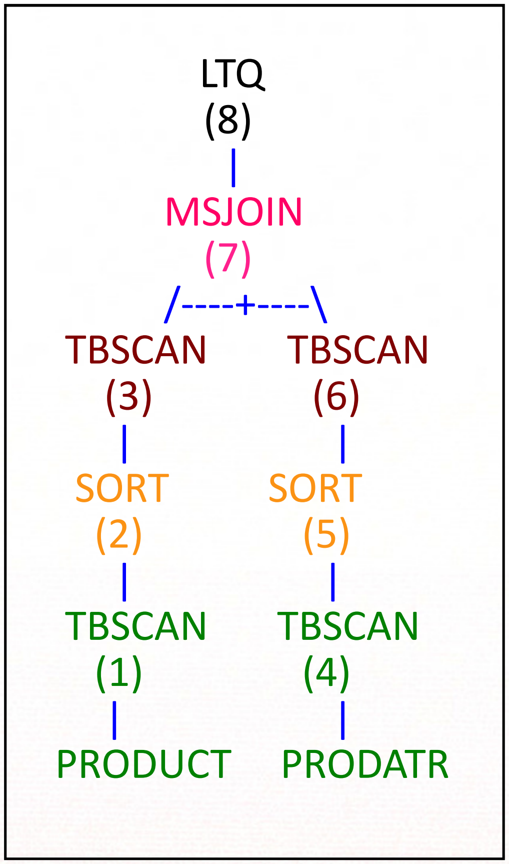
Row-organized Intra-Partition Parallelism Example (3/4)



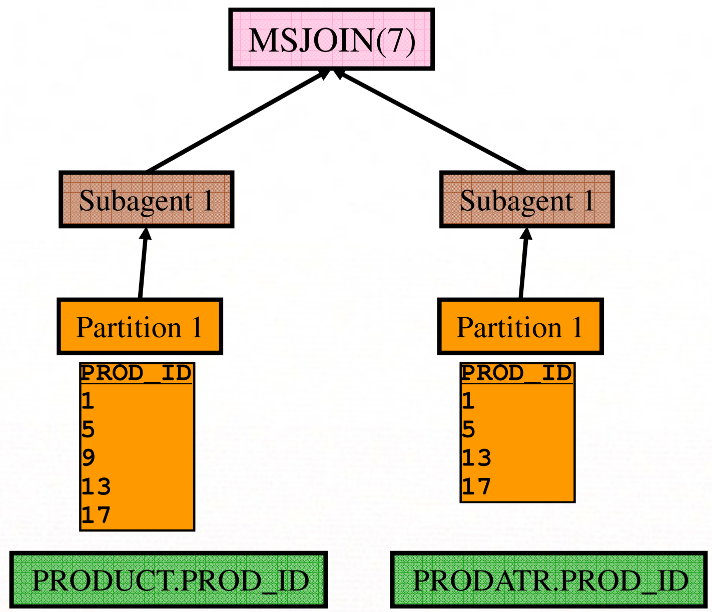
- Hash partitioned sort



Row-organized Intra-Partition Parallelism Example (4/4)



- Each sub-agent joins their corresponding sort partition



Row-organized Intra-Partition Query Parallelism Architecture

- Parallelization techniques
  - Scans dynamically partition data
    - 'Straw' partitioning (not hashed)
  - Sorts can be private, shared or partitioned
  - Temps can be private or shared
    - Temps are used to replicate streams

Row-organized Intra-Partition Query Parallelism Parallel Sorts (1/2)

- Private
  - Each subagent sorts and processes its own stream
  - Data is randomly partitioned
- Partitioned
  - Stream is hash-partitioned – provides ‘value’ partitioning
  - Most important for aggregation and distinct
- Shared
  - Each subagent inserts into the same sort, but the sort is read using a straw scan
  - Stream has random partitioning
  - Used for rebalancing a low cardinality stream
  - Higher contention on the single sort, but there are only a few rows

Row-organized Intra-Partition Query Parallelism Parallel Sorts (2/2)

- Round-robin
  - Multiple sort partitions are created
  - Subagents insert into each partition in round-robin fashion
  - Subagents assigned a single partition to read
  - Stream has random partitioning
  - Used to rebalance a high cardinality stream
  - Less contention => better parallelism
- Replicated
  - Each subagent inserts into the same sort
  - Each subagent reads the entire sort

Row-organized Intra-Partition Query Parallelism

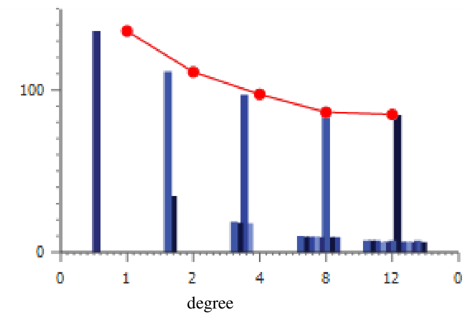
- Parallelization techniques
  - Joins
    - Hash join dynamically partitions data
      - Build and probe phases are each parallelized
    - Merge sort join relies on hash partitioned or replicated sorts
    - Nested loop join
      - Complex inners processed in parallel
      - Simple inners processed independently (privately)
      - Outer can be replicated or partitioned
  - Aggregation, distincting
    - Partial, final modes supported

Row-organized Intra-Partition Query Parallelism

- Load imbalance results in poor scalability
- REBAL operator redistributes rows to ensure all subagents do equal work
- Optimizer performs load balance analysis to determine REBAL placement

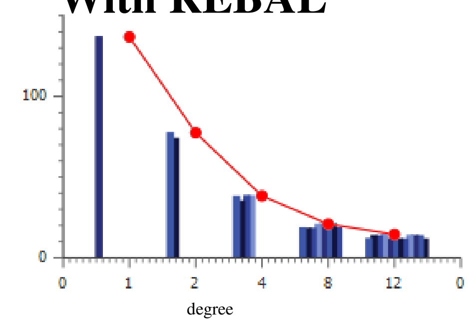
```
                     6.77122e+06
                       NLJOIN
                       (   6)
                       713706
                         63
             /---------+----------\
         292.2                     23173.3
         REBAL                      FETCH
        (   7)                     (   9)
        325.265                    2456.85
          11                          2
           |                     /---+----\
         292.2               23173.3    6.77122e+07
        TBSCAN               IXSCAN   TABLE: DB2USER
        (   8)               (  10)     DAILY_SALES
        325.265              1605.23        Q1
          11                    1
           |                    |
         2922              6.77122e+07
    TABLE: DB2USER        INDEX: SYSIBM
        PERIOD          SQL091218161022180
          Q2                   Q1
```

Without REBAL



Load balance - Comparison of CPU time between threads.

With REBAL



Load balance - Comparison of CPU time between threads.

Row-organized Intra-Partition Query Parallelism

- INSERT, UPDATE and DELETE operations are not executed in parallel
- A sub-select feeding an INSERT is parallelized, but not the INSERT
- SELECT over INSERT/UPDATE/DELETE is not parallelized

Query Parallelism and the Explain Facility

- Some operators are only used to support query parallelism
  - Table queues (TQ)
  - Rebalance operator (REBAL)
- Other operators can execute in serial or parallel
  - Extra explain arguments indicate the parallelization technique
- SORT
  - SORTTYPE (GLOBAL, PARTITIONED, ROUND ROBIN, REPLICATED, SHARED)
  - PARTCOLS
    - Partitioning columns when SORTTYPE=PARTITIONED

Query Parallelism and the Explain Facility

- TBSCAN, IXSCAN (row-organized processing only)
  - Optimizer determines these options for row-organized parallelism
  - Determined at runtime for column-organized parallelism
  - SCANGRAN (n): (Intra-Partition Parallelism Scan Granularity)
  - SCANTYPE: (Intra-Partition Parallelism Scan Type)
    - LOCAL PARALLEL
  - SCANUNIT: (Intra-Partition Parallelism Scan Unit)
    - PAGE | ROW
- TQ
  - TQ TYPE : (Table queue type)
    - LOCAL
  - TQDEGREE (n): (Degree of Intra-Partition parallelism)

Query Parallelism and the Explain Facility

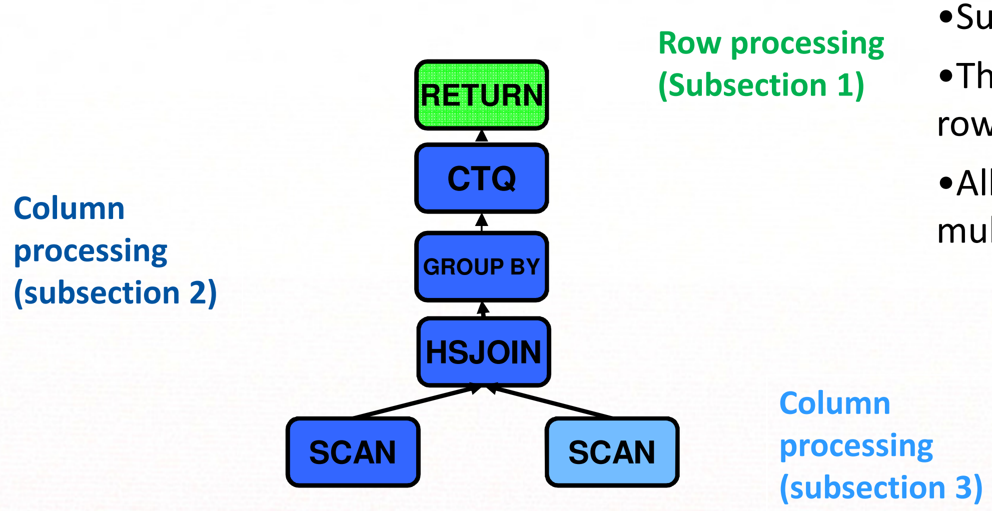
- TEMP (row-organized processing only)
  - SHARED (Temporary table is shared among subagents)
    - TRUE
  - SNGLPROD (Intra-partition parallelism SORT or TEMP produced by a single agent)
    - TRUE | FALSE

Column-organized Intra-Partition Query Parallelism

- Uses similar parallelization techniques as row-organized parallelism
  - Dynamic straw scans of column data
    - Straw size determined at runtime
  - Uses mostly hash or random partitioning
    - Limited use of range partitioning for certain types of OLAP functions
- There can be multiple column-organized subsections
  - Every individual table access is in a separate subsection
  - Subsections pass data through special runtime table queues
  - These aren't shown in explain because they are determined at runtime, not by the optimizer

Column-organized Intra-Partition Query Parallelism

- Column-organized operators execute in different subsections than row-organized operators
- Column-organized subsections are processed by different sets of subagents
- Data is transferred between row and column-organized subsections using a column-organized table queue (CTQ)
  - CTQ also performs row materialization



- Subsections run concurrently
- There can be multiple column or row processing subsections
- All subsections can be processed by multiple subagents

Column-organized Intra-Partition Query Parallelism

- BLU SORT uses PARADIS, a highly parallel in-place radix sort from IBM Watson
- BLU SORT can use range partitioning to improve parallelism and leverage ordered stream for multiple operations

```
Select  c1,
  c2,
  c3,
  max(c1) over (partition by c2),
  max(c1) over (partition by c2, c3)
from  tc1
```

```
   RETURN
   (   1)
     |
   LTQ
   (   2)
     |
   CTQ
   (   3)
     |
   TBSCAN
   (   4)
     |
   SORT
   (   5)
     |
   TBSCAN
   (   6)
     |
 CO-TABLE:
   TC1
```

- **Each sort output stream contains a range of values for C2**
- **The data is ordered on C3 within each distinct value of C2**
- **The data is not ordered on C2 within each stream**
- **C2 does not need to be ordered – just partitioned**
  - **Indicated by “R” (random order)**
- **This allows each MAX to be computed in parallel**

```
PARTCOLS: (Table partitioning columns)
            1: Q2.C2
SORTKEY : (Sort Key column)
            1: Q2.C2(R)
            2: Q2.C3(A)
SORTTYPE: (Intra-Partition parallelism sort type)
            PARTITIONED
```

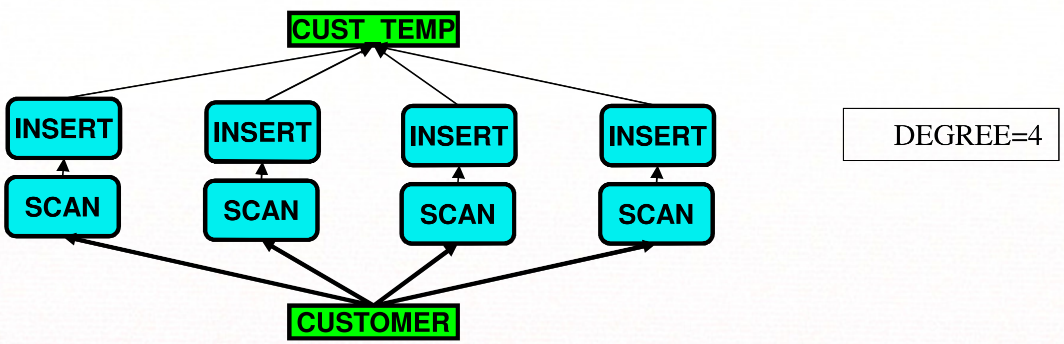
Column-organized Intra-Partition Parallelism

- Parallel INSERT is supported into BLU permanent and temporary tables (DGTTs)
  - Source can be a row or column-organized table
  - Applies to NOT LOGGED BLU DGTTs too
- Only used if a 'large' number of rows are being inserted
  - ~ 50K rows
- Each subagent puts data into a separate set of pages
- Parallelizing small inserts could result in a large amount of wasted space
- Ensure statistics are accurate so the optimizer can make the correct decision
- Enabled by default for BLU DGTTs in Db2 11.1
- Enabled by default for BLU permanent tables in Db2 11.1.2.2
- Some restrictions apply
  - See link to web page in speaker notes

Parallel Insert into NOT LOGGED BLU DGTT

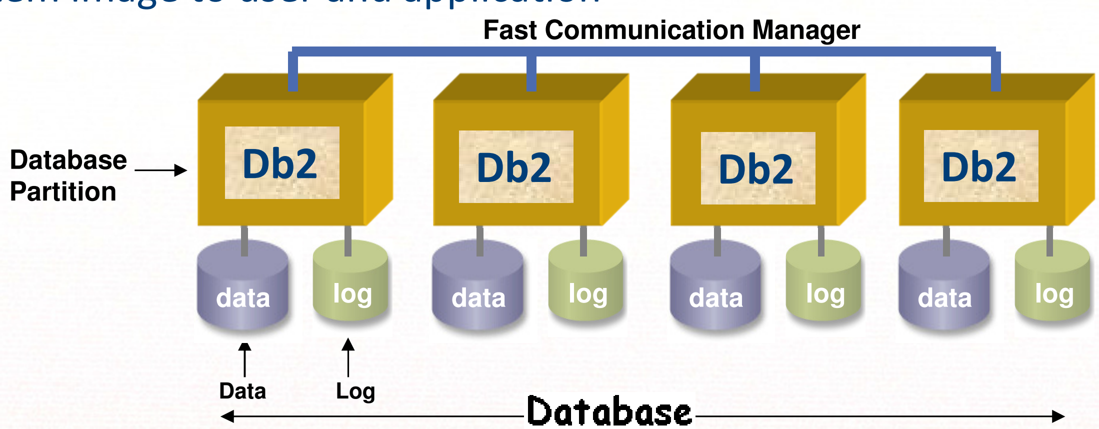
- Multiple DB agents can insert into a column-organized DGTT
  - Source must be a single table (regular or DGTT)
  - Source and target could be on a different number of DB partitions (MPP system)
  - Must be enough rows to make it worthwhile
    - about 50000 rows

```
INSERT INTO SESSION.CUST_TEMP SELECT * FROM DB2USER.CUSTOMER;
```



Inter-Partition Query Parallelism

- Shared nothing architectural model
- Partitioned database
  - Database is divided into multiple partitions
  - Database partitions can run on one or multiple machines
  - Each database partition has dedicated resources (engine, log manager, lock manager, bufferpool, etc.)
  - Parallel processing occurs on all partitions concurrently and is coordinated by the DBMS
  - Single system image to user and application



Inter-Partition Query Parallelism

- Tables are distributed across multiple DB partitions (hash or randomly partitioned)
- Queries are divided into subsections and executed across the DB partitions
- Query performance depends on how tables are partitioned and the relational operations
- Subsections are determined by table queue position

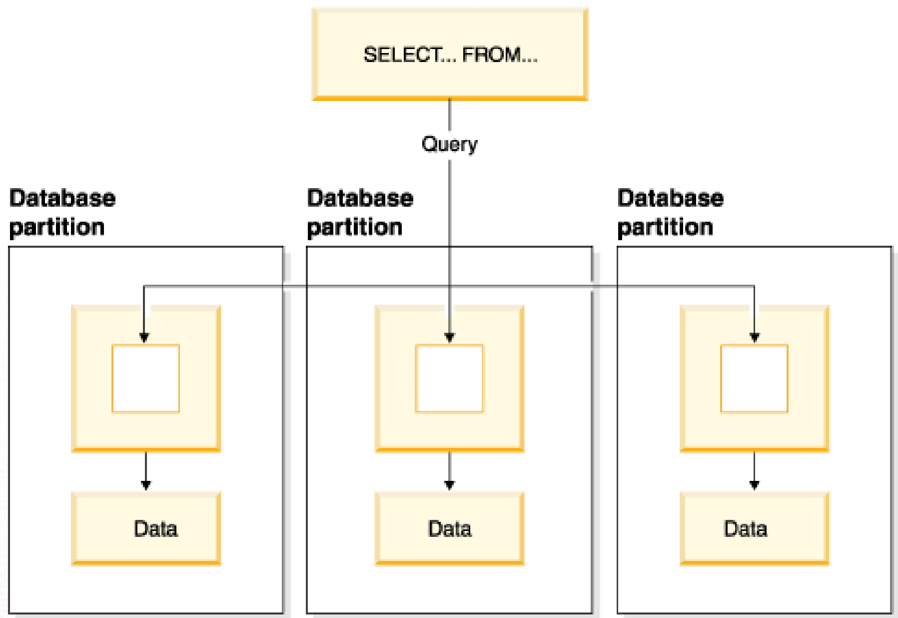


Table Queues (TQs)

- Table Queue represents communication between database partitions or subagents
- Subsection boundaries in DPF
- There are 5 types of TQs:
  - Merging TQ (MDTQ, MBTQ, LMTQ)
  - Broadcast TQ (BTQ, MBTQ)
  - Directed TQ (DTQ, MDTQ)
  - Local TQ (LTQ, LMTQ)
    - Intra-partition parallelism
  - Column-organized (CTQ, RCTQ)
    - Column->row or row->column

```
3) TQ    : (Table Queue)
    Arguments:
    ---------
    LISTENER: (Listener Table Queue type)
            FALSE
    SORTKEY : (Sort Key column)
            1: L_RETURNFLAG(A)
    SORTKEY : (Sort Key column)
            2: L_LINESTATUS(A)
    TQMERGE : (Merging Table Queue flag)
            TRUE
    TQREAD  : (Table Queue Read type)
            READ AHEAD
    TQSEND  : (Table Queue Write type)
            DIRECTED
    UNIQUE  : (Uniqueness required flag)
            FALSE
```

Inter-Partition Parallelism Join Strategies

Collocated join

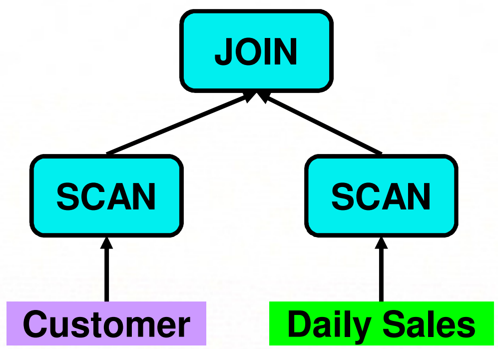
Partitioning keys:

CUSTOMER: CUSTKEY

DAILY_SALES: CUSTKEY

Join predicate:

CUSTOMER.CUSTKEY = DAILY_SALES.CUSTKEY



Equi-join predicate on each table's partitioning key
Tables must be in same DB partition group
Join column(s) data type must be partition compatible
No table queues (TQs) necessary

Directed join

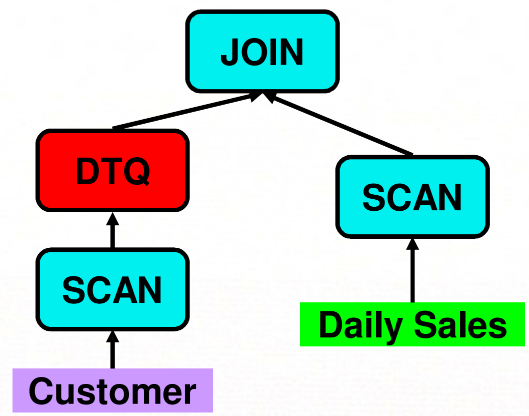
Partitioning keys:

CUSTOMER: CUST_NUMBER

DAILY_SALES: CUSTKEY

Join predicate:

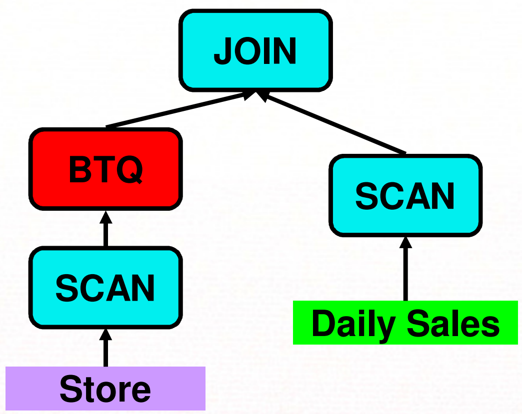
CUSTOMER.CUSTKEY = DAILY_SALES.CUSTKEY



Equi-join predicate on one table's partitioning key
Direct rows of one table to partitioning of the other

Inter-Partition Parallelism Join Strategies

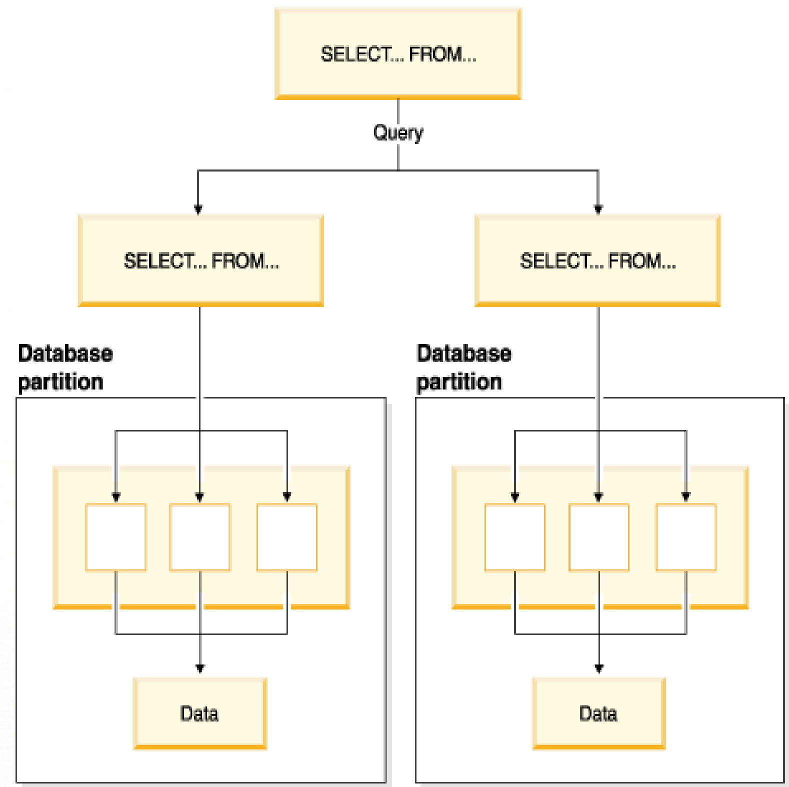
Broadcast join
Partitioning keys:
STORE: STOREKEY
DAILY_SALES: CUSTKEY
Join predicate:
STORE.STOREKEY = DAILY_SALES.STOREKEY



- No equi-join predicate on both table's partitioning key or no equi-join predicate.
- One table is much smaller than the other.
- Broadcast (replicate) smaller table to partitions of the larger table.

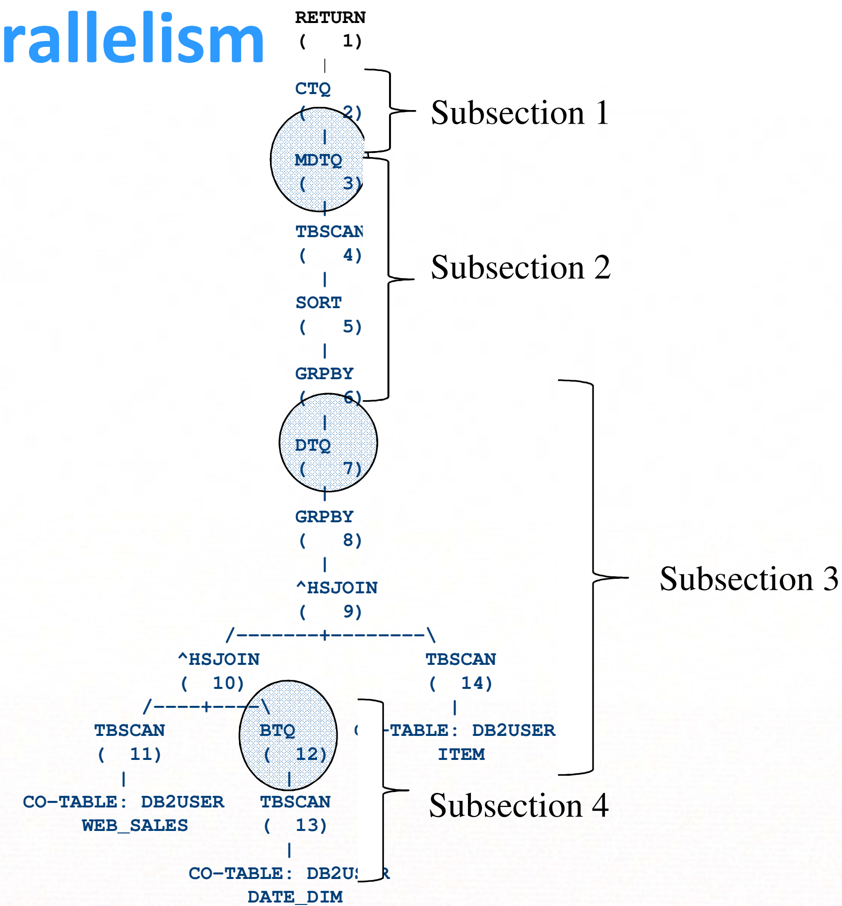
Inter + Intra Partition Parallelism

- Intra-partition parallelism can be enabled in a DPF system
- This allows each DPF subsection to be parallelized
- But… intra-parallel might add too much parallelism depending on the ratio of cores to DB partitions
- There might already be enough DPF subsections to drive parallelism
- BLU DPF systems should have more cores per DB partition, because BLU relies on intra-partition parallelism
  - 1 socket per DB partition
  - Typically 8-16 cores per DB partition
- Row-organized DPF systems could have 2-4 cores per DB partition



BLU Inter + Intra-Partition Parallelism

- BLU Table queues (TQ)
  - M = merging
  - D = directed
  - B = broadcast
- Flows encoded data
- A TQ delimits a DPF subsection
  - Each DPF subsection can use multiple subagents via intra-partition parallelism
  - Indicated by TQDEGREE explain argument
  - Recall that each DPF subsection could have multiple BLU subsections



Inter-Partition Parallel Aggregation

```
SELECT ITEM_DESC, SUM(PERCENT_DISCOUNT),
       SUM(EXTENDED_PRICE)
FROM PERIOD, DAILY_SALES, PRODUCT, STORE
WHERE PERIOD.PERKEY=DAILY_SALES.PERKEY AND
       PRODUCT.PRODKEY=DAILY_SALES.PRODKEY AND
       STORE.STOREKEY=DAILY_SALES.STOREKEY AND
       CALENDAR_DATE BETWEEN
       '01/01/2005' AND '04/28/2005' AND
       STORE_NUMBER='03' AND
       CATEGORY=72
GROUP BY ITEM_DESC ;
```

Row-organized processing example

Repartition stream

PARTCOLS: (Table partitioning columns)
1: Q6.ITEM_DESC
SORTKEY : (Sort Key column)
1: Q6.ITEM_DESC(A)

Local partial aggregation

AGGMODE : (Aggregration Mode)
INTERMEDIATE
GROUPBYR: (Group By requirement)
1: Q6.ITEM_DESC

Sort aggregation

AGGMODE : (Aggregration Mode)
PARTIAL
SORTKEY : (Sort Key column)
1: Q6.ITEM_DESC(A)

Final aggregation

AGGMODE : (Aggregration Mode)
FINAL
GROUPBYR: (Group By requirement)
1: Q6.ITEM_DESC(A)

```
   Rows
  RETURN
  (   1)
   Cost
    I/O
     |
   18136
  DTQ
  (   2)
  17791.7
  4639.2
     |
   4534
  GRPBY
  (   3)
  17778.2
  4639.2
     |
  17909.4
  MDTQ
  (   4)
   17773
  4639.2
     |
  17909.4
  GRPBY
  (   5)
  17748.9
  4639.2
     |
  17909.4
  TBSCAN
  (   6)
  17743.7
  4639.2
     |
  17909.4
  SORT
  (   7)
  17738.5
  4639.2
```

Inter-Partition Parallelism

- INSERT, UPDATE and DELETE are automatically parallelized for DB-partitioned tables

Intra-Partition Query Parallelism Configuration

- There are 2 types of controls:
  - Toggling intra-partition parallelism support
    - Db2 requires extra infrastructure to support query parallelism
      - Controlling access to shared data structures
      - Co-ordination of multiple subagent threads
    - This infrastructure adds 10-15% overhead to OLTP applications
  - Controlling the degree of parallelism
    - The number of subagents used to execute a query
    - Query parallelism is bad for OLTP SQL statements
    - Query parallelism is good for reporting queries on OLTP applications

Toggling Intra-partition Parallelism Support

- Instance level switch
  - INTRA_PARALLEL database manager configuration parameter
- Application level switch
  - SYSPROC.ADMIN_SET_INTRA_PARALLEL(YES | NO) stored procedure
  - Takes effect in the next transaction
  - Overrides the instance level switch
- Workload level switch
  - MAXIMUM DEGREE workload manager option
  - Value of 1 disables intra-partition parallelism
  - Value > 1 enables intra-partition parallelism and sets a cap on the degree
  - Overrides the instance and application level switches

Toggling Intra-Partition Query Parallelism

- WLM workload control:
  - An OLTP workload that doesn't use parallelism
    - =1 → INTRA_PARALLEL=NO

    ```
    CREATE WORKLOAD banking_wl APPLNAME ('banking') MAXIMUM DEGREE 1;
    ```
  - A complex query workload using parallelism
    - >1 → INTRA_PARALLEL=YES
    - Also specifies the degree upper limit
    - The application specifies the requested degree using existing external controls

    ```
    CREATE WORKLOAD report_wl APPLNAME ('cognos') MAXIMUM DEGREE 8;
    ALTER WORKLOAD report_wl MAXIMUM DEGREE 4;
    ```
- Application control:

  ```
  CALL SYSPROC.ADMIN_SET_INTRA_PARALLEL('YES')
  ```
- Toggles intra-partition parallelism at <u>transaction</u> boundaries
  - Must not have open cursors across transaction boundaries e.g. WITH HOLD cursors

Controlling the Degree of Parallelism

- There are 2 types of controls:
  - (1) Tell the optimizer what degree to use
    - CURRENT DEGREE special register (dynamic SQL)
    - DEGREE bind option (static SQL)
    - dft_degree database configuration parameter
      - Provides the default for special register or bind option
    - The degree can be a specific value
      - SET CURRENT DEGREE '16'
      - The query will use 16 subagents
    - The degree can be 'ANY'
      - SET CURRENT DEGREE 'ANY'
      - The optimizer will determine the degree
      - Runtime might choose to further reduce the degree based on system load

Controlling the Degree of Parallelism

- (2) Specify a degree limit at runtime
- MAX_QUERYDEGREE database manager configuration parameter
  - Instance level limit on the degree for any statement
  - Dynamic
  - The optimizer does NOT consider this parameter
  - ‘ANY’ or -1 means there is no limit
- SET RUNTIME DEGREE command
  - Allows specifying the maximum degree for a particular application using the application handle
  - SET RUNTIME DEGREE FOR (41408, 55458) TO 4
  - SET RUNTIME DEGREE FOR ALL TO 2

Controlling Intra-Partition Query Parallelism

- Intra-partition query parallelism requires **shared sort heap**
- Shared sort heap is always allocated when INTRA_PARALLEL DBM config parm = ON
  - SHEAPTHRES_SHR is automatic
- <u>Shared sort heap is not allocated when INTRA_PARALLEL=OFF and SHEAPTHRES > 0</u>
  - Must set SHEAPTHRES=0 in order to enable INTRA_PARALLEL at application or workload level
- If no shared sort heap available:
  - CALL SYSPROC.ADMIN_SET_INTRA_PARALLEL(‘YES’) will fail with SQL5192W
  - WLM MAXIMUM DEGREE > 1 setting will have no effect

Intra-Partition Query Parallelism Configuration

- Common Scenario: mixed workload support
  - Parallelize report queries in an OLTP system
  - Avoid parallel 'infrastructure' overhead on OLTP queries
    - There is a 10-15% impact just by setting INTRA_PARALLEL=ON
      - In ESE only. DPF unconditionally enables parallel infrastructure
    - Disable query parallelism at instance level
      - **update dbm cfg using intra_parallel off**
    - Enable parallelism for connections executing reporting queries
      - **SET CURRENT DEGREE 'ANY'**
      - Use Workload Manager (WLM) to toggle INTRA_PARALLEL and maximum DEGREE for a workload OR
      - Reporting connections issue "CALL ADMIN_SET_INTRA_PARALLEL(YES)"

Controlling the Degree of Parallelism

- The degree of parallelism is not limited by the number of cores/processors
  - It is completely independent, other than for degree = ‘ANY’
- Degree can exceed the number of cores
  - Query parallelism can be used on a single-processor machine
- Sometimes over parallelization can improve performance if system is I/O bound.

Intra-Partition Parallelism External Controls Summary

Switch
Switch + maximum
Runtime maximum
Compile time value

Parameter	Value	Default	Scope	DB2 10	Comment
INTRA_PARALLEL	NO,YES	NO	Instance	N	DBM configuration
ADMIN_SET_INTRA_PARALLEL	NO,YES	NO	Application	Y	Stored procedure. Switch INTRA_PARALLEL for a connection.
MAXIMUM DEGREE	1-32,767	DEFAULT	Workload	Y	WLM workload option. Controls both INTRA_PARALLEL and maximum runtime degree
MAX_QUERYDEGREE	ANY, 1-32,767	ANY	Instance	N	DBM configuration. Maximum runtime degree.
SET RUNTIME DEGREE	1-32,767	N/A	Application	N	CLP command. Maximum runtime degree for specific applications.
DFT_DEGREE	ANY, 1-32,767	1	Database	N	DB configuration. Default value for CURRENT DEGREE special register or package bind DEGREE option
CURRENT DEGREE	ANY, 1-32,767	DFT_DEGREE	Application	N	Special register. The degree of parallelism considered by the SQL compiler for dynamic SQL.
Bind DEGREE	ANY, 1-32,767	DFT_DEGREE	Package	N	DB2 bind option. The degree of parallelism considered by the SQL compiler for static SQL.

Monitoring Intra-Partition Query Parallelism

- Determine exactly what degree a statement used
- **intra_parallel_state**: YES/NO
- **effective_query_degree**: degree chosen by optimizer
- **query_actual_degree**: degree chosen at runtime if degree='ANY'

```
SELECT
    ac.application_handle, package_name, intra_parallel_state,
    effective_query_degree, query_actual_degree, stmt_text
    FROM
    TABLE( WLM_GET_WORKLOAD_OCCURRENCE_ACTIVITIES ( null, -1 )) ac,
    TABLE( MON_GET_ACTIVITY_DETAILS( ac.application_handle, ac.uow_id,
                                     ac.activity_id, -1)) ad,
    XMLTABLE (XMLNAMESPACES( DEFAULT 'http://www.ibm.com/xmlns/prod/db2/mon'),
              '$actmetrics/db2_activity_details'
              PASSING XMLPARSE( DOCUMENT AD.DETAILS ) AS "actmetrics"
              COLUMNS
              "PACKAGE_NAME" VARCHAR(128)      PATH 'package_name',
              "INTRA_PARALLEL_STATE" CHAR(3)   PATH 'intra_parallel_state',
              "EFFECTIVE_QUERY_DEGREE" BIGINT PATH 'effective_query_degree',
              "QUERY_ACTUAL_DEGREE" BIGINT     PATH 'query_actual_degree',
              "STMT_TEXT" VARCHAR(1024)        PATH 'stmt_text' ) am;
```

IDUG Leading the DB2 User Community since 1988

IDUG Db2 Tech Conference NA
Philadelphia, PA | April 29 - May 3, 2018

#IDUGDb2

John Hornibrook

IBM Canada

jhornibr@ca.ibm.com

Session code: C19

Please fill out your session evaluation before leaving!