



## SAS® Scalable Performance Data Server® 4.3 TSM1:

*Parallel Join with Enhanced GROUP BY Processing*



---

## Table of Contents

<b>Introduction</b> .....	<b>1</b>
<b>Parallel Join Coverage</b> .....	<b>1</b>
Parallel Join Execution .....	1
<b>Parallel Join Requirements</b> .....	<b>5</b>
Tables Types Supported .....	5
Column Types .....	5
Parallel Join and Parallel GROUP BY Methods.....	5
Types of Join Syntax Supported .....	6
Inner Joins .....	6
Left and Right Join Support .....	7
Outer Joins .....	8
<b>Parallel Join Execution Methods</b> .....	<b>8</b>
Controlling the Parallel Join .....	9
<b>Performance and Tuning</b> .....	<b>10</b>
Hardware.....	10
Data Tables .....	11
SQL Queries .....	11
Demonstration of Benefits.....	12
Parallel Sort Usage.....	13
Parallel Join With GROUP BY Enhancement .....	13
Concurrency .....	13
Ordered Join Keys.....	14
<b>Conclusion</b> .....	<b>15</b>



---

## Introduction

SAS Scalable Performance Data Server (SAS SPD Server) is designed to meet the storage and performance needs for processing large amounts of data in applications that use SAS software. As the size of the data grows, the demands to process that data quickly increase, and storage architecture must change to keep pace with business needs. Hundreds of sites worldwide use the SAS SPD Server, which hosts data warehouses in excess of 30 terabytes. The SAS SPD Server provides high-performance data storage for customers from industries such as banking, credit card, insurance, telecommunications, healthcare, pharmaceutical, transportation, and brokerage and government agencies.

This paper discusses the coverage, restrictions, tuning, and performance benefits of the Parallel Join Facility and the enhanced GROUP BY method in the SAS SPD Server 4.3 TSM1 release. This paper assumes a basic understanding of SAS SPD Server tables and concepts of data partitioning, symmetric multiprocessing, and configuring disk for I/O scalability. Knowledge of SQL and joins is also recommended. The SAS SPD Server performs best with multiple CPUs and parallel I/O, where the hardware is set up to provide a maximum scalable environment.

This paper is divided into four sections:

1. Parallel Join Coverage
2. Parallel Join Requirements
3. Parallel Join Execution Methods
4. Performance and Tuning

---

## Parallel Join Coverage

An *SQL join* is a method that combines rows from two tables using processes such as sorting and match-merge processing to create final output. A *parallel join* is a method that combines rows from two tables using multi-threading for sorting and match merging to create final output, the goal of which is to reduce the total time required to complete the task.

The SAS SPD Server 4.3 TSM1 release provides a new facility that executes SQL joins in parallel, thereby reducing the total time required to complete the query. This new facility provides three different parallel join methods, and it leverages the SAS SPD Server's parallel GROUP BY method to provide further enhancement of join performance when a query includes a GROUP BY clause.

## Parallel Join Execution

The Parallel Join Facility is a new feature of the SAS SPD Server SQL planner, which decreases the run time of a query by executing joins of tables in parallel. For SQL queries that include joins

between pairs of tables, the planner will determine if any of the pairs of tables can be handled by the Parallel Join Facility. The planner first searches for pairs of SAS SPD Server source tables where a join is being performed between two tables. The planner checks the join syntax for that pair to determine if it meets all the requirements of the Parallel Join Facility. When the syntax meets the requirements, the pair of tables will be joined by the Parallel Join Facility.

Not every join can be executed by the Parallel Join Facility. This section provides a high-level view of what pairs of tables are chosen by the facility for parallel execution and how to understand when parallel execution has taken place. Other sections of this paper discuss the SQL syntax requirements that are necessary for a query to be executed by the Parallel Join Facility.

The first case is a simple join between two tables. Both tables must be stored as SAS SPD Server tables under the same server. They can be in the same domain or in separate domains. However, if the two tables are in separate domains, there is a restriction as to the type of parallel join the facility can perform. This restriction is explained later in the section “Parallel Join Requirements”.

The following example is a basic SQL query that will be executed by the Parallel Join Facility. The bolded section of the query shows the two tables that will be joined in parallel by the facility.

```
%let spdssqlr=_method
    plljoin;

proc sql;
  create table junk as
  select *
  from path1.dansjunk1 a,
  path1.dansjunk2 b
  where a.i = b.i;
quit;
```

When a job is executed that uses the Parallel Join Facility, the application prints a note from the log that indicates which parallel join method was used. When you use the reset option `_METHOD`, the application prints a tree that shows all SQL methods that were used to execute the query. The previous program outputs the following log section:

```
SPDS_NOTE: SQL execution methods chosen are:
<0x000000010052CF28> sqxslct
<0x000000010052CDA8> sqxjp11
SPDS_NOTE: Parallel Merge Join Method
Selected
```

The method **sqxjp11** is one of two execution methods the Parallel Join Facility uses to join two tables in parallel. For this SQL query, the method chosen to perform the join was the parallel sort-merge method.

The second case is a join with more than two tables. Unless the join is executed by the Star-Join Facility, the planner only joins two tables at a time in the execution phase, which, in this case, is called a *pair-wise join execution*. Before the execution phase, the planner looks at all of the tables in the collection and generates an ordered list of the tables to be joined. The planner then joins the first two tables in the list and produces an intermediate result set. Then, the planner selects the next table in the sequence to join with the intermediate result, and it continues iteratively until all tables in the collection are joined.

For the pair-wise execution model, the Parallel Join Facility can only perform a parallel join on the first pair of tables that are joined (assuming that they meet the requirements for parallel joining). All other pairs of joins are single threaded. The following example shows a section of the SAS log with the method tree showing the node **sqxjpl1**, which indicates that two of the tables were using a parallel join, and the third table was using a hash join. When three tables are joined, the method tree does not indicate which two tables were chosen by the Parallel Join Facility for the parallel join, so no tables are highlighted in the program. For more complex three-table joins, the planner can modify the order of the tables that will be joined.

```
%let spdsSQLr=_method
      plljoin;

proc sql;
  create table junk as
  select *
    from path1.dansjunk1 a,
         path1.dansjunk2 b,
         path1.dansjunk3 c
  where   a.i = b.i
         and b.i = c.i;
quit ;

SPDS_NOTE: SQL execution methods chosen are:
<0x0000000100E90448>   sqxslct
<0x0000000100E902C8>       sqxjhsh
<0x0000000100EC25E0>           sqxjpl1
<0x0000000100E8E728>       sqxsrc
SPDS_NOTE: Parallel Merge Join Method
                Selected
```

Support for parallel-join execution includes any nested join in a query that meets the parallel-join requirements. When a query contains multiple nested joins, each of which meet the parallel-join criteria, then each nested join is executed in parallel. The following example shows an SQL query with a nested join that is executed in parallel. The highlighted sections indicate nested sections that will be executed with the Parallel Join Facility. The example's log shows the method tree and a message stating which parallel join method was used.

```
proc sql;
  create table junk as
  select *
    from path1.dansjunk1 a,
         (select *
           from path1.dansjunk2 b,
              path1.dansjunk3 c
           where b.i = c.i) tbl1
  where a.i = tbl1.i;
quit;

SPDS_NOTE: SQL execution methods chosen are:
<0x00000001004E5D68>   sqxcrtA
<0x00000001004E5B28>       sqxjhsh
<0x0000000100EC0FD8>           sqxjpl1
<0x00000001006F84B0>       sqxsrc
SPDS_NOTE: Parallel Merge Join Method
                Selected
```

Support for parallel joins also includes cases where a union of two joins occurs, as long as at least one of the joins meets the Parallel Join Facility requirements. Each side of the union is evaluated for parallel joins individually. For each subjoin that qualifies, the joins will be executed in parallel. In the following example, two individual parallel joins are executed. In the final table, the result will be the union of the two parallel joins. As in the previous example, the highlighted statements show which sections are executed in parallel. The example's log shows that two parallel joins were performed. The log messages indicate that parallel sort merges were used for both joins.

```

%let spdssqlr=_method
    plljoin;

proc sql;
  create table junk as
  select *
    from path1.table1 a,
         path1.table2 b
   where a.i = b.i
 union
  select *
    from path1.dansjunk3 c,
         path1.dansjunk4 d
   where c.i = d.i;
quit;

SPDS_NOTE: SQL execution methods chosen are:
<0x0000000100E95D88>   sqxslct
<0x0000000100E95CC8>       sqxuniq
<0x00000001006737F0>       sqxuall
<0x0000000100E93B28>                               sqxjp11
<0x0000000100E95A88>                               sqxjp11
SPDS_NOTE: Parallel Merge Join Method Selected
SPDS_NOTE: Parallel Merge Join Method Selected

```

The Parallel Join Facility also includes enhancements for data summarization that use the SAS SPD Server's parallel GROUP BY technology. The following example shows the combined use of both the parallel join and parallel GROUP BY methods.

```

%let spdssqlr=_method
    plljoin;

proc sql;
  create table junk as
  select a.junk1,
         b.junk2,
         sum (a.amt) as sum_amt
  from path1.dansjunk1 a,
       path1.dansjunk2 b
   where a.i = b.i
   group by a.junk1,
            b.junk2;
quit;

SPDS_NOTE: SQL execution methods chosen are:
<0x00000001006FB1F8>   sqxslct
<0x00000001004E7918>       sqxxpjn
SPDS_NOTE: Parallel Merge Join with Group By
            Method Selected

```

The log shows that the Parallel Join Facility used the SPD Server method `sqxxpjn`, which combines a parallel join with the enhanced parallel GROUP BY technology.

---

## Parallel Join Requirements

This section discusses the query syntax restrictions that determine whether a pair of tables will be accepted or rejected by the Parallel Join Facility. The section covers restrictions, acceptable syntax, and allowable summary functions.

### Tables Types Supported

Pairs of tables that are candidates for execution by the Parallel Join Facility must be SAS SPD Server tables that were used as source input to a join. To qualify as source input for a join via the Parallel Join Facility, tables must be SAS SPD Server. The tables can be standard SAS SPD Server tables, dynamic cluster tables, or a combination of the two. In addition, the tables must be stored on the same SAS SPD server, although they can be stored in separate domains on that server.

### Column Types

The Parallel Join Facility handles multicolumn character columns, numeric columns, or a combination of character and numeric columns that are joined between pairs of tables. For joins that involve numeric columns, the column widths do not have to match. For example, a table with a numeric column of 4 bytes for the join key can be joined to a table with a numeric column that has a width of 6 bytes for the join key. Joins that involve character columns must have identical column widths to take advantage of the parallel join method. For example, the Parallel Join Facility will not be used if a character column in one table has a width of 6 bytes and the corresponding character column in the other table has a width of 5 bytes.

Columns that are involved in a join cannot be derived from the CASE statement or from functions such as SUBSTR, YEAR, MONTH, DAY, and TRIM. The following example shows columns that are derived from SQL functions, and, as such, are not supported in the Parallel Join Facility.

```
substr (junk1,1,1) as just_j
year (a.date_test) as what_year
month (a.date_test) as what_mth
qtr (a.date_test) as what_qtr
```

### Parallel Join and Parallel GROUP BY Methods

One of the more important features of the Parallel Join Facility is the use of the Parallel GROUP BY summarization technology that accelerates SQL joins that have GROUP BY clauses. The GROUP BY method uses SQL queries to provide rapid, parallel summarization of single tables.

Within the scope of the Parallel Join Facility, the restriction of using the Parallel GROUP BY method only on single tables has been removed.

The GROUP BY enhancement includes support for the following functions: COUNT, AVG, MEAN, MAX, MIN, NMISS, STD, SUM, and VAR. This support is identical to that for the Parallel GROUP BY method. Support is limited to those functions with single columns, such as shown in this example:

```
sum(amount) as tot_amt.
count(distinct column_name) as
  dist_values
```

Calculations using two columns are not supported such as the following code example:

```
sum (amount - discount) as
  tot_less_discount
```

## Types of Join Syntax Supported

Four types of SQL joins are supported by the Parallel Join Facility: inner, outer, left, and right. Join types not supported by the facility are cross and natural. Parallel join coverage for the UNION operator was discussed previously in the section “Parallel Join Execution”.

### Inner Joins

*Inner joins*, also called *equijoins*, are joins in which a value of the join column in one table is matched to the identical value of the join column in another table. When a row from either table does not have a corresponding matching value from the other table on the join key table, that row is discarded. Two forms of the syntax are supported for inner joins. The first form specifies an inner join by use of a WHERE clause. The following program illustrates the use of the WHERE clause:

```
proc sql;
  select *
  from table1 a,
       table2 b
  where a.join_key = b.join_key;
quit;
```

The second form of syntax uses an INNER JOIN operator. This example illustrates the use of the INNER JOIN operator.

```
proc sql;
  select *
  from table1 a
  inner join table2 b
  on a.join_key = b.join_key;
quit;
```

## Left and Right Join Support

*Right* and *left* joins specify that all the rows of one table involved in the join (right or left) should have all of the selected rows output in the join. If you use the RIGHT or LEFT operators, the planner will execute the query, keeping all rows from one of the tables and adding blank or missing values to columns in the other table with no row matches. Right and left tables are determined by the order in which they appear in the SQL program submitted. The first table specified in the SQL syntax `from table-name` is the left table. The right table is specified second. If the join is a left join, all rows from the left table are kept, and columns from the rows that do not have matching rows from the right table are populated with blank or missing values. If the join is a right join, all rows from the right table are kept, and columns from the rows that do not have a matching row from the left table are populated with blank or missing values.

The Parallel Join Facility supports both the right and left join syntax. In the following examples, the highlighted sections show which operations will be executed by the Parallel Join Facility.

### Example 1: Left Join on Numeric Columns

```
proc sql;
  create table junk as
  select *
    from path1.dansjunk1 a
   left join path1.dansjunk2 b
     on a.i = b.i;
quit;
```

SPDS\_NOTE: SQL execution methods chosen are:  
 <0x0000000100EC40F8> sqxslct  
 <0x0000000100EC4038> sqxjpl1  
 SPDS\_NOTE: Parallel Modulo Join Method  
 Selected

The method tree in this log output shows that the parallel join was performed, and the messages indicate that a parallel modulo join was used. When performing left and right joins on numeric columns, the Parallel Join Facility performs a numeric hash join by default. For character join columns, the facility uses a parallel sort-merge join, as shown here in Example 2.

### Example 2: Right Join on Character Columns

```
proc sql;
  create table junk as
  select *
    from path1.dansjunk1 a
   right join path1.dansjunk2 b
     on a.ic = b.ic;
quit;
```

SPDS\_NOTE: SQL execution methods chosen are:  
 <0x00000001004E7318> sqxslct  
 <0x00000001004E7258> sqxjpl1  
 SPDS\_NOTE: Parallel Merge Join Method  
 Selected

## Outer Joins

An *outer join* keeps all rows from both tables and sets columns that would normally be excluded from the other table to blank or to missing values when there are no corresponding rows to match. In this example, the bolded statements show sections that the Parallel Join Facility will execute.

```
proc sql;
  create table junk as
  select *
  from path1.dansjunk1 a
  full join path1.dansjunk2 b
  on a.i = b.i;
quit;
```

```
SPDS_NOTE: SQL execution methods chosen are:
<0x0000000100EC0D98> sqxslct
<0x0000000100EC0CD8> sqxjpl1
SPDS_NOTE: Parallel Modulo Join Method
Selected
```

The method tree shows that the parallel join was performed, and the log indicates that the parallel modulo join was used. As with the right and left joins, the Parallel Join Facility performs a numeric hash join by default for numeric join columns and a parallel sort-merge join for character join columns.

---

## Parallel Join Execution Methods

The Parallel Join Facility supports three execution methods (sort-merge, range, and numeric hash), all of which are variations of a parallel sort-merge join. The difference among the three parallel join methods is in how the SQL planner distributes work among the threads during the execution phase of the join. The planner determines the work distribution based on knowledge about the join key values and the location of those values in the table.

The *sort-merge join method* employs the SAS SPD Server parallel sort to order the data and then merge the tables in parallel. During the join, the facility concurrently joins multiple rows from one table with rows in the other table. You can use this general-purpose method to execute any join that meets the requirements for parallel join.

The *range join method* indexes each of the tables to be joined. The range index divides the two tables into a specified number of near-equal parts based on matching values between the join columns. The facility then joins a range of rows that contain matching values between the join columns. The Parallel Join Facility recognizes which ranges to match, and it uses concurrent join threads to join the rows in parallel. To use a range join, both tables must be in the same domain. The best scenario for range joins is a join on tables that have been presorted or that are highly ordered on the join keys.

The *numeric hash join method* uses a divide-and-conquer approach to joining the tables. The hash join method first hashes the tables into sections using the join key values; then it joins matching sections together. After the two join-candidate tables have been hashed into individual sections, the facility concurrently joins, by multiple threads, sections that had identical modulo

factors. The numeric hash join must involve a numeric column. The table that is hashed must have enough different values of the join key to support splitting rows into multiple parts. For example, if all values in the join column are identical, then the column can be hashed into only one section because the hash function will return the same value for every column.

## Controlling the Parallel Join

The Parallel Join Facility has a set of default behaviors that are used to control the execution methods, which control the degree of parallelism employed. A full set of options are available to turn the Parallel Join Facility on or off, to turn the Parallel GROUP BY feature on or off, and to control the types of parallel join.

Four options control the Parallel Join Facility. The options can be set in the SPDSSERV.PARM file, by using either the macro variable SPDSSQLR for implicit pass-through or using an EXECUTE command for explicit pass-through. The following examples illustrate how these options are set:

### Example 1: SPDSSERV.PARM

```
splopts="reset plljoin concurrency=4
        pllmagic=0";
```

### Example 2: Macro Variable

```
%let spdssqlr=plljoin
    concurrency=4
    pllmagic=0;
```

### Example 3: Explicit Passthru

```
execute (reset plljoin
        concurrency=4
        pllmagic=0) by sasspds;
```

You can globally set the main toggle option for the Parallel Join Facility, PLLJOIN/NOPLLJOIN, for the server in the SPDSSERV.PARM file by using the SQLOPTS option. You can also set it for individual queries by using the reset options mentioned previously.

By default, the Parallel Join Facility sets concurrency to a value equal to half the number of processors on the machine. To set the concurrency to another value, use the SAS SPD Server SQL reset option CONCURRENCY=. This option controls the degree of parallelism used by the Parallel Join Facility. SAS recommends not increasing the concurrency setting higher than the Parallel Join Facility default settings. Increasing concurrency can degrade performance because competing parallel join threads compete for resources.

The PLLJMAGIC= option controls the execution methods that are available to the planner. By default, there is a hierarchy of default parallel join execution methods chosen. The first default method performs a range join provided that the range join index has been created. If the hierarchy cannot choose a range join, the second method is a parallel sort-merge method.

You can override these defaults by using the SAS SPD Server SQL reset option `PLLJMAGIC=`. When this option is set to 0, the Parallel Join Facility selects the default settings. The option accepts a three-digit integer that controls the use of the parallel range join and parallel sort-merge join plus a third range of values, which controls the use of the parallel modulo join (including its concurrency setting). Valid values are the integers 100, 200, and a range from 300 to 364.

A value of 100 forces a parallel range join provided that a range index is present. A value of 200 forces a parallel sort-merge join.

Values between 300 and 364 force the use of a parallel modulo join. The two trailing digits control the modulo factor that the server will use. When the value 300 (last two digits are “00”) is used, the modulo factor sets the value of the hash partitioning to the value of concurrency, whether it is the facility default or a manual setting. When the value is between 301 and 364, the modulo factor is the integer represented by the last two digits, or, equivalently, is found by subtracting 300 from the value. For example, a parallel join with `PLLJMAGIC=304` will have a modulo hash partition factor computed as  $304 - 300 = 4$ .

SAS recommends using the default values. The best performance is usually obtained with the parallel range join, provided that the join keys are ordered or reasonably ordered on the join key. The next best performance is typically found with the parallel sort-merge join. The parallel numeric hash join was developed to facilitate future development, and it should not be used in a production system currently.

---

## Performance and Tuning

To see the impact of the Parallel Join Facility on query performance, SAS constructed a test environment that compares run times of the same queries with and without the facility turned on. A description follows of the test bed and the results of the performance testing.

### Hardware

Because the SAS SPD Server Parallel Join Facility requires various tasks, processing, and I/O to occur simultaneously, the parallel scalability of the hardware plays a critical role in the overall performance of the server in answering queries. The demonstrations cover the behavior of the SAS SPD Server on a UNIX server that has 16 gigabytes of RAM and 8 processors. The server has 7 independent file systems: one for indexes and metadata, one for temporary utility files, and five for actual data (rows of the tables). No file system attached to the machine shares a disk drive with any other file system.

Even with this modest server configuration, the parallel join execution of the queries was able to drive all 8 CPUs to near 100 percent utilization. This behavior is due to the Parallel Join Facility's use of the threaded parallel sort, which enables it to concurrently join rows between the two tables. In addition, once the rows are joined, they can be directly summarized using the enhanced GROUP BY processing that is included in the Parallel Join Facility.

## Data Tables

The tests use a pair of standard SAS SPD Server tables, each with 132 million rows and a column width of 112 bytes. One table represents the sales for the year 2004 and the other table represents sales for the year 2005. The table contains 12 months of example data that represents sales to customers of a financial corporation. Each month contains 11 million rows of data. The values for the join key column are unique across the entire table. The rows in the table are sorted by date—oldest data first.

Each table has two columns that can be used for performing joins between the tables. The two columns contain values that create one-to-one matches of rows from one table to other. The first of the set of join columns is ordered by the join key values. The second set of join columns contain the same values as the first set, but the values are randomly distributed within each month of data loaded into the table.

To demonstrate the performance benefits of having the enhanced GROUP BY method in the facility, additional columns of data will be grouped and summarized in the queries

## SQL Queries

All test queries executed will perform inner joins. Every join performed for the demonstration creates an output table. To isolate the benefits of the Parallel Join Facility, no subsetting of the tables will occur. The queries fall into four main categories.

The first two categories are inner joins that keep all rows from both tables; one case is when the join key is ordered in both tables, and the second case is identical but with the join key in each table being randomly ordered. Optimizations in the software can detect when tables are already ordered, which avoids performing unnecessary sorts.

The second two categories are similar to the first two cases, but they include additional grouping and summarization of columns from the joined tables on a column or columns other than those used to join the two tables. The purpose of demonstrating these two inner joins is to show the enhanced GROUP BY performance that is included in the Parallel Join Facility.

The basic SQL query syntax used is shown here:

### Category 1:

```
proc sql;
  create table table-name as
  select column-list
  from table1 a,
       table2 b
  where a.join-key = b.join-key;
quit;
```

Category 2:

```

proc sql;
  create table table-name as
  select column-list
  from table1 a,
        table2 b
  where a.random-join-key=
        b.random-join-key;
quit;

```

Category 3:

```

proc sql ;
  create table table-name as
  select column-list
  from table1 a,
        table2 b
  where a.join-key = b.join-key
  group by column-list;
quit;

```

Category 4:

```

proc sql;
  create table table-name as
  select column-list
  from table1 a,
        table2 b
  where a.random-join-key=
        b.random-join-key
  group by column-list;
quit;

```

## Demonstration of Benefits

We will present relative run times rather than actual run times because the specific performance of SAS SPD Server is highly dependent on the hardware configuration. What makes this technology compelling is a comparison of the relative improvement of parallel join against not having parallel join. Performance improvement can be realized across a wide variety of hardware configurations. The benefits presented here demonstrate the two areas where the Parallel Join Facility provides the greatest benefits.

The numbers presented are based on run-time reduction using the Parallel Join Facility compared to a base line without the use of the facility. To accomplish the measurements, identical jobs were run against identical tables using technology prior to the Parallel Join Facility and then using the Parallel Join Facility.

The time required for a job to complete without the Parallel Join Facility is assigned a baseline number of 100. The run time of the query with parallel join will be presented as a percent of the baseline. For example, if a job without parallel join took 10 minutes to complete and the job with parallel join took 2 minutes to finish, the results will be presented as 100 percent and 20 percent. The 20 percent represents the percentage of time compared to the baseline that the query required to complete without the Parallel Join Facility.

The performance demonstrations are broken into four sections:

1. improvements due to parallel sort
2. improvements due to enhanced summarization
3. effects of changing concurrency settings
4. impact of having tables stored sorted or unsorted.

## Parallel Sort Usage

The SAS SPD Server parallel sort, a multithreaded sort that can perform scalable I/O, can be leveraged by parallel join. The facility will use parallel sort when it determines that in a pair of tables being joined, one or both of the tables require sorting. When one table in the pair is already sorted by the join key, the facility will bypass the sort.

For this demonstration, the query performs an inner join using the parallel sort-merge method. The same query will be executed without the use of the Parallel Join Facility. The query kept all columns of both tables and did not subset the tables. Because the join key between the data tables has a one-to-one match in the other table, the number of rows output was equal to the number of rows input.

The results show that the Parallel Join Facility with the parallel sort reduced the run time of the query by 90 percent.

## Parallel Join With GROUP BY Enhancement

Next we demonstrated the parallel join GROUP BY method by running two identical queries that used parallel join: one query had the enhanced join GROUP BY method turned off (using the reset option NOGRPSEL), and the other query had the enhanced join GROUP BY method turned on. Both jobs used the parallel sort-merge method with a concurrency of 4. The query that included the enhanced GROUP BY method ran 88 percent faster than the query without the GROUP BY method. This example demonstrates the significant benefit of using the GROUP BY method when executing queries with the Parallel Join Facility..

## Concurrency

Next, we tested the same query with different concurrency settings for each parallel join method. We performed runs with the concurrency set to 1, 2, 4, 6, 8, and 16. The columns that we used to join the tables had randomly distributed keys. The join matched 132 million rows and returned five columns (three from the first table, two from the second). No summarization of the joined tables occurred.

The following table shows the results from the multiple runs using each available method of the Parallel Join Facility. The column labeled “Without Parallel Join” provides a baseline of 100 to compare the previous SAS SPD Server SQL join with joins that use Parallel Join Facility. All of the other results are a percentage of the baseline. Note that there are no values for Concurrency=1 because that situation represents a degenerate case of performing parallel sort-merge with extraneous overhead.

	Without Parallel Join	Sort-Merge Join	Range Join	Numeric Hash Join
Concurrency=1	100	17.1	N/A	N/A
Concurrency=2	100	15.1	15.1	16.8
Concurrency=4	100	12.5	12.8	18.4
Concurrency=8	100	10.5	10.7	27.0
Concurrency=16	100	9.4	9.6	44.1

### Ordered Join Keys

Finally, we demonstrated how a table sort in order of join keys will benefit parallel join performance. The best sort is one that is performed only one time. When a table is frequently joined on the same join columns, it makes sense to optimize the join by loading the pair of tables ordered by the join key. Presorting the tables enables the Parallel Join Facility to avoid costly overhead because it

1. bypasses the sort process for pairs of tables in order to join them
2. uses the optimum parallel join method (the sort-merge method) for ordered join keys, but without the sorting.

The following table illustrates the benefit of joining presorted tables. We set concurrency to 16 for this example. The first column shows a baseline from an identical join that uses unsorted tables and that was not performed with the Parallel Join Facility. The second and third columns show the results using the best parallel sort method, the parallel sort-merge join. The second column illustrates performance using unsorted tables. The third column illustrates performance using sorted tables.

	Without Parallel Join Unsorted Tables	Parallel Sort- Merge Join with Unsorted Tables	Parallel Join with Presorted Tables
Run time Factor	100	9.4	7.9
CPU Utilization	15%	95%+	15%

This example shows a 16 percent performance gain between runs on the unsorted and the sorted tables. This gain may not seem as significant as the gain obtained in the example comparing runs with no parallel sort and parallel sort. However, the second row of the table (CPU Utilization) shows the true success of this method. Using sorted tables enables the server to process the query faster to use 1/6<sup>th</sup> the amount of CPU as the same query with unsorted tables. More efficient use of CPU resources permits more queries and more users on the server than would be possible if the tables were not sorted.

---

## Conclusion

The Parallel Join Facility provides two important benefits to join performance:

- use of the SAS SPD Server parallel sort
- use of the Parallel GROUP BY method in a join.

These benefits can greatly improve the performance of table joins, and, thus query performance. Such improvements make SAS SPD Server an effective solution for large data storage for business and analytic intelligence.



World Headquarters  
and SAS Americas  
SAS Campus Drive  
Cary, NC 27513 USA  
Tel: (919) 677 8000  
Fax: (919) 677 4444  
U.S. & Canada sales:  
(800) 727 0025

SAS International  
PO Box 10 53 40  
Neuenheimer Landstr. 28-30  
D-69043 Heidelberg, Germany  
Tel: (49) 6221 4160  
Fax: (49) 6221 474850  
**[www.sas.com](http://www.sas.com)**