Paper 196-2007

# Pipes and Threads: Performance Testing of Advanced Scalability Features in SAS® v9

Gerhardt Pohl, Fred Forst, Mario Widel, Thomas Burger
Eli Lilly and Company, Indianapolis, IN

## ABSTRACT

SAS v9 provides many new features of interest to users with large data sets. Many of the commonly used procedures are multi-treaded. This allows users to harness the full power of multi-processor architectures to enhance the efficiency of some tasks. In addition, SAS/Connect allows users to spawn multiple independent processes to execute custom parallel processing solutions. SAS/Connect also enables pipeline parallelism, i.e., piping of records from one DATA/PROC step to another between processes. This potentially relieves some of the I/O burden of the total SAS job, an important consideration when dealing with large data sets. We examined the real world speed of these three features using two different multiprocessor system architectures, a Sun Microsystems Sun Fire 3800 running Unix and an IBM 9672/Y36 running MVS. We investigated a typical usage pattern with a simple DATA step followed by a PROC TABULATE. Test data were from a simulated insurance claims file with test runs ranging from 10 million to approximately 322 million records. Our tests suggest that for an architecture based on traditional synchronous processing using the traditional method of the DATA step writing intermediate output to disk, SAS v9 is not substantially different than SAS v8. However, the enhanced features in SAS v9, piping and multi-threading, yield significantly shorter total run times on more powerful architectures. These performance gains were evident at all file sizes tested (1.2 to 38.4 GB). We noted also that the built-in multi-threading in PROC TABULATE performed better than a manual parallel processing solution. This is an important finding since it suggests that shorter run times can be achieved directly with the base product without the added expense of an additional complicated programming effort.

## INTRODUCTION

The introduction of SAS v9 marked a major upgrade in the SAS computing system with many features targeting scalability for large datasets. Many procedures have been enhanced with the ability to perform parallel processing. This includes some of the most frequently used procedures (MEANS, REPORT, SORT, SQL, SUMMARY, TABULATE, GLM, REG).

Introduced in V8, SAS/CONNECT gives you the ability to exploit SMP (Symmetric Multi-Processing) hardware as well as network resources to perform parallel processing and easily coordinate all the results into a single client SAS session.

In v9, SAS/CONNECT supports pipeline parallelism, which allows multiple DATA steps or procedures to execute in parallel and to pipe the output from one process as the input to the next process in a pipeline. Piping improves performance and reduces the demand for disk space.

## SYSTEM ARCHITECTURES

In our testing we used two basic architectures.

### UNIX

SAS version: SAS 8.2 (TS2MO) and 9.1.3 (TS1M3)
Operating System: SunOS 5.8
Sun Microsystems Sun Fire 3800
# of CPUs=8
Total memory available=16Gb
Each CPU's speed is 750Mhz
Data files and SAS work space were housed on Clariion CX700 storage configured as RAID 5 (4+1, 146 GB FC, 10,000 RPM drives), 225 GB device.
The application was housed on Symmetrix 8830 storage RAID 0+1 (Striped and Mirrored), 33 GB Meta devices. 4 x 146 GB SCSI drives (10,000 RPM), 2 Cylinder stripe size (960K), cache=32 GB with 1 Gbit front-end directors.

**MVS**

SAS version: V8.2 & V9.1.3 SP4
OS version: z/OS 1.4
IBM 9672/Y36
# of CPUs=3
Total memory available=7Gb
Each CPU's speed is 150MIPS (IBM publishes this as 575Mhz)
Storage Environment:
- ESS (SHARK) 2105 800
- Capacity is 12TB
- Average response time is 5ms
- Everything is fiber channel

## TEST METHOD

Tests were run at times of low load on the respective systems. Elapsed real time (wall clock time) was determined using the SAS function DATETIME before and after the task as shown. Reported times are based on several test runs of each scenario. Note that in parallel processing situations the total elapsed cpu time is often greater than the real time. This is precisely the efficiency that is hoped for. By spreading the load, more computing can be accomplished in less real time.

```
DATA _NULL_;
    T=DATETIME();
    CALL SYMPUT('START',T);
RUN;
```

The basic test task consists of a DATA step with some minor processing followed by a PROC TABULATE. DATA step reads a large simulated insurance claims data set with fairly small individual record size (128 bytes per record) and calculates out of pocket costs for the patient as the difference between the amount charged by the caregiver and the amount paid by the insurance company. The quarter of the year when the claim was transacted is added to the record based on a date field in the record. A two-way summary is created with rows being quarter (4 class levels) crossed by regions of the US (4 class levels) and columns being patient gender (2 class levels). A summary column for all patients regardless of gender is also created. Note that we did not investigate the impact of various table structures (number of levels, order of dimensions, etc.) on the efficiency of PROC TABULATE.

```
data testdata;
    set  savedata.simclaims;
    rec_id+1;
    out_of_pocket_costs=charge-paid;
    quarter=compress(put(from_dt,qtrr.))||' Q';
run;

proc tabulate data=testdata;
    class mem_sex region quarter;
    table quarter*region,mem_sex all;
run;
```

15 variations on this basic task were tested. Scenarios revolved around use of different levels of multi-threading in the tabulate step, use of pipeline parallelism to transfer data from a process executing the DATA step to a process running the PROC TABULATE, and manual partitioning the data to execute separately on portions of the PROC TABULATE. We call this last scenario manual parallelism. Test scenarios are summarized in the table below and some programming details of the various scenarios are discussed in later subsections.

Test Scenarios:

| Scenario | Name |
|---|---|
| 1 | v8 |
| 2 | v9 1 cpu no pipes |
| 3 | v9 2 cpu no pipes |
| 4 | v9 4 cpu no pipes |
| 5 | v9 8 cpu no pipes |
| 6 | v9 1cpu piped |
| 7 | v9 2 cpu piped |
| 8 | v9 4 cpu piped |
| 8 | v9 8 cpu piped |
| 10 | v9 2-way manual parallel no pipes |
| 11 | v9 4-way manual parallel no pipes |
| 12 | v9 8-way manual parallel no pipes |
| 13 | v9 2-way manual parallel piped |
| 14 | v9 4-way manual parallel piped |
| 15 | v9 8-way manual parallel piped |

Number of cpu's refers to the degree to which PROC TABULATE was provided cpu's with which to execute in threaded mode. System option THREADS was in effect in all runs, but was constrained by the number of cpu's specified in the CPUCOUNT= global option. Note, importantly, that the CPUCOUNT for the initiating client process was not constrained. Only the PROC TABULATE server process was constrained. Therefore, 1 cpu does not necessarily represent and architecture with only one physical cpu. Separate cpu's may have been used for the sending and the receiving processes of any pipe even if the receiving PROC TABULATE was constrained to be single threaded. Pipe refers to the use of a TCP socket to transfer records from the DATA step to the succeeding processing (in contrast to writing to the SAS work directory on disk). N-way manual parallel refers to the number of succeeding processes spawned to execute portions of the PROC TABULATE.

#### BASELINE: SCENARIOS 1 AND 2

Note that in SAS v8.2 pipeline parallelism and threading of PROC TABULATE are not available. Scenario 2 is simply the SAS v9.1.3 equivalent using a single cpu (non-threaded PROC TABULATE) with the intermediate work data set being written to disk.

#### MULTI-TREADING WITHOUT PIPES: SCENARIOS 3 - 5

No special coding required other than that the option CPUCOUNT must be set to an appropriate number higher than one and the THREADS option must be in effect for the PROC TABULATE. The intermediate work data set is written to disk.

#### PIPING WITHOUT MULTI-TREADING IN THE PROC TABULATE: SCENARIO 6

Pipeline parallelism requires initiation of at least two processes, one to write to the pipe and one to read from the pipe. The key feature is the LIBNAME statement which assigns the SASESOCK engine to a socket rather than an output disk file. In the example code, we demonstrate writing to a specific unused port number on our system. In these scenarios, the PROC TABULATE process does not need to wait for the DATA step process to complete. The PROC TABULATE reads records from the socket while the DATA step continues its work.

```
options comamid=tcp;
signon datatask sascmd “sas”;

signon task0 sascmd="sas -cpucount 1";

rsubmit process=datatask wait=no;
   libname savedata 'directory on disk system';
   libname outlib0 sasesock ':7000';

   data outlib0.q;
       set  savedata.simclaims;
       rec_id+1;
       out_of_pocket_costs=charge-paid;
       quarter=compress(put(from_dt,qtrr.))||' Q';
   run;
```

```
endrsubmit;

rsubmit process=task0 wait=no ;
   libname inlib0 sasesock ':7000';

   proc tabulate data=inlib0.q;
       class mem_sex region quarter;
       table  quarter*region,mem_sex all;
   run;
endrsubmit;
```

In our testing we initiated the overarching, controlling SAS session (client session) with CPUCOUNT=8. Therefore, when spawning server sessions on the same machine to act as either the DATA step task or the PROC TABULATE task, the server session had access to separate cpu's on the system. That is to say that specifying CPUCOUNT=1 for the DATA step process and CPUCOUNT=1 for the PROC TABULATE did not simulate the effect of having a system with only one cpu. This is an important distinction as it is not clear how efficient piping would be on a true single processor system in which the task writing to the pipe was potentially competing for cpu resources with the process that was reading from the pipe.

#### PIPING WITH MULTI-TREADING IN PROC TABULATE: SCENARIOS 7-9

This is similar to scenario 6 except that at SIGNON to the PROC TABULATE task multiple CPUCOUNT is set to an appropriate number higher than one and the THREADS option is in effect for the PROC TABULATE.

```
signon task0 sascmd="sas -cpucount 2 -threads";
```

#### MANUAL PARALLEL SOLUTION WITHOUT PIPING: SCENARIOS 10-12

In these scenarios, the DATA step splits the data set into a number of logical pieces. For example, in the 2-way scenario illustrated below, the DATA step performs an initial split to two files, one containing records for female patients and the other containing records for the male patients. (For the 4-way method, the split was based on the four quarters of the year; and, for the 8-way method the split was based on gender crossed with quarter.) Simultaneous processes are then initiated to perform a PROC TABULATE on each of the individual smaller files. These PROC TABULATE's are given access to all 8 cpu's, i.e., they utilize multi-threading.

```
options comamid=tcp;
signon datatask sascmd="sas –cpucount 1";
signon task0 sascmd=”sas -cpucount 8 -threads";
signon task1 sascmd=”sas -cpucount 8 -threads";

rsubmit process=datatask wait=no;
   libname savedata 'directory on disk system';

   data savedata.q0 savedata.q1;
       set  savedata.simclaims&size;
       rec_id+1;
       out_of_pocket_costs=charge-paid;
       quarter=compress(put(from_dt,qtrr.))||' Q';
       if mem_sex='M' then output savedata.q1;
       else output savedata.q0;
   run;
endrsubmit;

rsubmit process=task0 wait=no ;
   libname savedata 'directory on disk system';

   proc tabulate data=savedata.q0;
       class mem_sex region quarter;
       table  all, quarter*region,mem_sex all;
   run;
endrsubmit;
```

```
rsubmit process=task1 wait=no ;
   libname savedata 'directory on disk system';

   proc tabulate data=savedata.q1;
       class mem_sex region quarter;
       table  all, quarter*region,mem_sex all;
   run;
endrsubmit;
```

**MANUAL PARALLEL SOLUTION WITH PIPING: SCENARIOS 13-15**

These are similar to scenarios 10-12 except that the data is written to pipes rather than to disk files. Note that the initial DATA STEP writes to more than one pipe, with each pipe being read by one PROC TABULATE process.

```
options comamid=tcp;
signon datatask sascmd="sas –cpucount 1";
signon task0 sascmd=”sas -cpucount 8 -threads";
signon task1 sascmd=”sas -cpucount 8 -threads";

rsubmit process=datatask wait=no;
   libname savedata 'directory on disk system';
   libname outlib0 sasesock ':7000';
   libname outlib1 sasesock ':7001';

   data outlib0.q0 outlib1.q1;
       set savedata.simclaims&size;
       rec_id+1;
       out_of_pocket_costs=charge-paid;
       quarter=compress(put(from_dt,qtrr.))||' Q';
       if mem_sex='M' then output outlib1.q1;
       else output outlib0.q0;
   run;
endrsubmit;

rsubmit process=task0 wait=no ;
   libname inlib0 sasesock ':7000';

   proc tabulate data=inlib0.q0;
       class mem_sex region quarter;
       table all, quarter*region,mem_sex all;
   run;
endrsubmit;

rsubmit process=task1 wait=no ;
   libname inlib1 sasesock ':7001';

   proc tabulate data=inlib1.q1;
       class mem_sex region quarter;
       table all, quarter*region,mem_sex all;
   run;
endrsubmit;
```

## TEST RESULTS

Table 1: Unix Run Times (hh:mm:ss)

| | Number of Records | | | |
|---|---|---|---|---|
| | 10 million | 30 million | 100 million | 322 million |
| Scenarios | (1.2 GB) | (3.6 GB) | (11.9 GB) | (38.4 GB) |
| 1) v8 | 00:01:52 | 00:05:55 | 00:24:26 | 01:13:43 |
| 2) v9 1 cpu | 00:02:12 | 00:06:42 | 00:25:31 | 01:24:24 |
| 3) v9 2 cpu | 00:01:51 | 00:05:45 | 00:22:56 | 01:14:52 |
| 4) v9 4 cpu | 00:01:51 | 00:05:49 | 00:23:05 | 01:11:16 |
| 5) v9 8 cpu | 00:01:50 | 00:05:46 | 00:23:01 | 01:09:48 |
| 6) v9 1cpu piped | 00:01:38 | 00:04:49 | 00:17:21 | 00:56:43 |
| 7) v9 2 cpu piped | 00:01:15 | 00:04:09 | 00:13:58 | 00:48:22 |
| 8) v9 4 cpu piped | 00:01:15 | 00:04:09 | 00:13:47 | 00:48:04 |
| 9) v9 8 cpu piped | 00:01:15 | 00:04:09 | 00:13:54 | 00:48:09 |
| 10) v9 2-way manual parallel | 00:01:52 | 00:05:51 | 00:24:07 | 01:19:56 |
| 11) v9 4-way manual parallel | 00:01:56 | 00:05:39 | 00:23:48 | 01:18:41 |
| 12) v9 8-way manual parallel | 00:01:57 | 00:05:48 | 00:24:54 | 01:19:13 |
| 13) v9 2-way manual parallel piped | 00:01:19 | 00:04:18 | 00:14:17 | 00:47:20 |
| 14) v9 4-way manual parallel piped | 00:01:23 | 00:04:32 | 00:14:59 | 00:52:28 |
| 15) v9 8-way manual parallel piped | 00:01:30 | 00:04:47 | 00:15:56 | 00:54:16 |

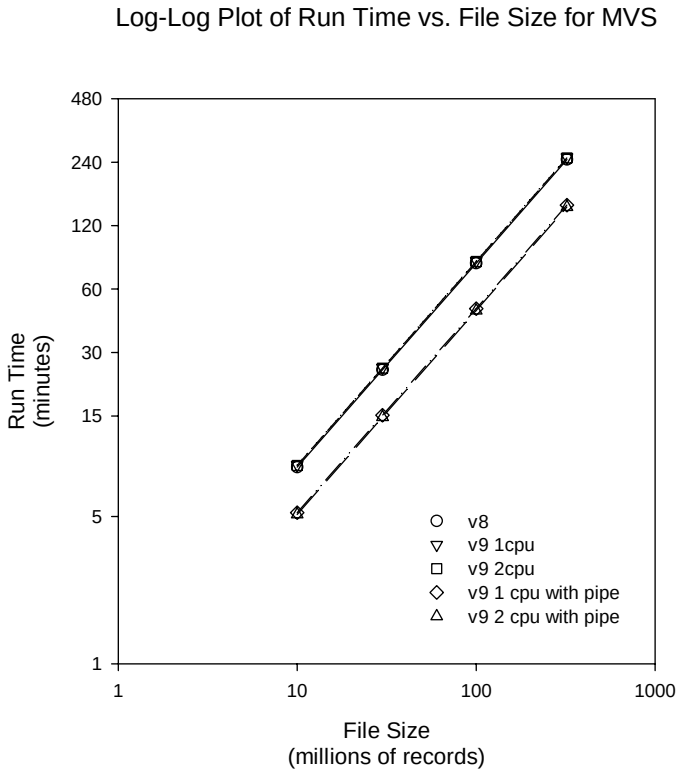
Table 2: MVS Run Times (hh:mm:ss)

| | Number of Records | | | |
|---|---|---|---|---|
| | 10 million | 30 million | 100 million | 322 million |
| Scenarios | (1.2 GB) | (3.6 GB) | (11.9 GB) | (38.4 GB) |
| 1) v8 | 00:08:48 | 00:26:30 | 01:26:06 | 03:28:31 |
| 2) v9 1 cpu | 00:08:41 | 00:23:40 | 01:20:00 | 04:40:00 |
| 3) v9 2 cpu | 00:08:06 | 00:27:18 | 01:20:42 | 04:12:00 |
| 6) v9 1 cpu piped | 00:05:10 | 00:14:51 | 00:49:39 | 02:30:21 |
| 7) v9 2 cpu piped | 00:05:19 | 00:14:09 | 00:46:54 | 02:29:27 |

Note not all scenarios possible on MVS since machine has only 3 cpu's.

The MVS architecture we tested was older and less powerful than the Unix architecture and not surprisingly yielded systematically slower times. However, on both architectures, each of the scenarios scaled linearly in the number of records, and the ratio of run times between scenarios was constant. We, therefore, summarized the relative efficiency of the scenarios by using PROC GLM to fit an analysis of covariance model (ANCOVA) regressing the log times against the log file sizes as shown, for example, in figure 1. The Unix scenarios were modeled similarly.

Figure 1:




The antilog of the difference in LSMEANS from the ANCOVA models yields estimates of the ratio of run times between scenarios as shown in table 3. We took scenario 2, v9 with 1 cpu and no pipe to be the baseline scenario for calculating efficiency relative speeds.

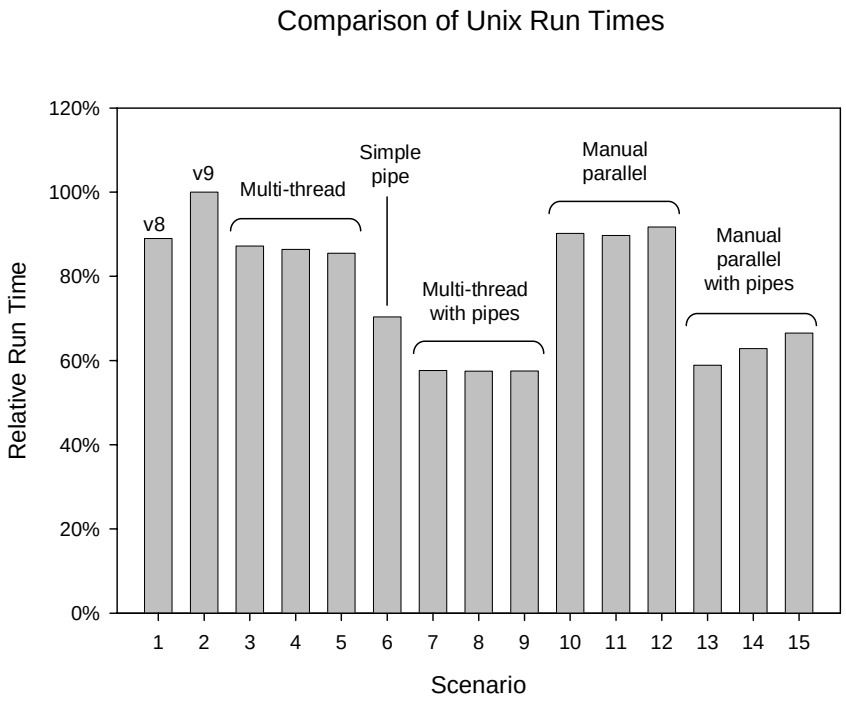
Table 3: Relative Run Times Between Scenarios

| Scenarios | Unix | MVS |
|---|---|---|
| 1) v8 | 89.0% | 97.7% |
| 2) v9 1 cpu | 100.0% | 100.0% |
| 3) v9 2 cpu | 87.2% | 99.4% |
| 4) v9 4 cpu | 86.4% | n/a |
| 5) v9 8 cpu | 85.5% | n/a |
| 6) v9 1cpu piped | 70.4% | 59.4% |
| 7) v9 2 cpu piped | 57.6% | 58.2% |
| 8) v9 4 cpu piped | 57.5% | n/a |
| 9) v9 8 cpu piped | 57.5% | n/a |
| 10) v9 2-way manual parallel | 90.2% | n/a |
| 11) v9 4-way manual parallel | 89.7% | n/a |
| 12) v9 8-way manual parallel | 91.7% | n/a |
| 13) v9 2-way manual parallel piped | 58.9% | n/a |
| 14) v9 4-way manual parallel piped | 62.8% | n/a |
| 15) v9 8-way manual parallel piped | 66.5% | n/a |

Note not all scenarios possible on MVS since machine has only 3 cpu's.

Relative speed of the various scenarios under Unix is depicted in figure 2.

Figure 2:




On Unix, SAS version 8 (single cpu and with a limitation to 2 GB RAM) was actually faster than SAS v9 operating with one cpu and access to all available RAM. Multi-threading in v9 brought the speeds down to slightly lower than v8. In this test, there was no benefit to multi-threading with more than 2 cpu's. Allowing the application access to 4 or 8 cpu's yielded similar run times to 2 cpu's. Piping showed a dramatic improvement in run time even with only one cpu. The benefit of piping was incremental to that of multi-threading and the best times achieved were with both multiple cpu's and piping between the DATA step and proc. Again there was no benefit to increasing the number of cpu's to more than 2 in the multi-thread with pipes scenarios.

The manual parallel processing yielded less improvement than the built-in multi-threading in PROC TABULATE. Run times increased as the number of parallel processes increased. This may represent contention for cpu's. Each of the up to eight PROC TABULATE processes potentially spawns eight light weight processes due to multi-threading. Increased overhead in coordinating communication between the processes might also play a role in increasing the run times. Given the large added effort required and programming complexity, this approach cannot be recommended for the simple task that was tested. Piping again showed a dramatic incremental improvement in the manual parallel processing scenarios as it did in the simpler scenarios.

A similar summary of run times under MVS is given in figure 3.

Figure 3:


Under MVS the situation was somewhat different than under Unix. There was no difference in the baseline one-cpu scenarios between SAS v8 and v9. Unlike in the Unix testing, built-in multi-threading in PROC TABULATE did not yield an improvement over using only one cpu. This could not be tested across higher numbers of cpu's due to the limited hardware we had available on MVS. Piping yielded even more substantial benefits than on the Unix system. Run times were reduced to only one-third of what they had been without piping.

## CONCLUSION

The multi-threading built into PROC TABULATE in v9 yielded minor improvements in run times in our simple test. More impressive was the speed up enabled by piping between the DATA step and the proc, using SAS/CONNECT. This eliminated a major I/O bottleneck in the processing of the large data sets. Reductions in run time to as low as 58% were observed. Scenarios using SAS/CONNECT to manually partition the problem and perform parallel processing failed to yield improvements sufficient to warrant their added complexity. Overall, SAS v 9 appears to represent a major enhancement allowing users to harness the power of large architectures to reduce run times particularly as regards piping to ease I/O burden.

## ACKNOWLEDGMENTS

Scott A. Iseminger, Edward M. Johnstone, Michael Kallin Carter.

## RECOMMENDED READING

Pipeline Parallelism Performance Practicalities
http://support.sas.com/rnd/papers/sugi30/pipeline.pdf

Pushing the Envelope: SAS System Considerations for Solaris/UNIX in Threaded, 64 bit Environments
http://www.sas.com/partners/directory/sun64bit.pdf

Performance Tuning & Sizing Guide for SAS Users and Sun System Administrators Updated Edition
http://www.sas.com/partners/directory/sun/sugi28SAS

**CONTACT INFORMATION**

Thomas H. Burger
Eli Lilly and Company
Lilly Corporate Center
Drop Code 1756
Indianapolis, IN 46285 USA
1-317-277-7266: