Paper PA08

Parallel processing techniques for performance improvement for SAS® processes: Part II

Viraj R Kumbhakarna, JPMorgan Chase & Co., Columbus, OH

ABSTRACT

In this paper, we will discuss a host of new functionality within SAS® software version 9 related to parallel processing. Parallel processing refers to processing that is handled by multiple CPUs simultaneously. This technology takes advantage of hardware that has multiple CPUs, called SMP computers, and provides performance gains for two types of processes:

- threaded I/O
- threaded application processing

We also explore the use of newer, faster, practically applicable parallel processing techniques supported by SAS® software for version 9.2 and later versions that can be used for processing large volume of data in parallel on AIX UNIX as well as windows SAS® software environments. We further dwell on identifying ways to break the data to process in parallel, determining number of threads to process in parallel, using SAS MPCONNECT for multithreading and analyzing support for spawning and managing multiple threads.

In this paper, we also propose techniques to execute processes in parallel on AIX UNIX platform. We explore the use of Piping which is an extension of the MP connect functionality to address pipeline parallelism and discuss different techniques used to analyze the data, to analyze the current server configuration and use it to determine optimal number of threads to be submitted in parallel. In conclusion we compare and contrast benefits, cost overhead and return over investment (ROI) of implementing parallel processing for a statistical and analytical process in the form of a case study to process huge data volume (in the author's experience, over 5 million observations) and argue benefits of parallel processing in terms of improved performance, reduced processing times and reduced I/O.

INTRODUCTION

Processing times for complex applications processing huge volume of data are often very long, lasting for several days at a time.. Parallel processing can be used in order to improve processing speed and reduce processing time. SAS offers piping in MP connect in order to process serially independent steps in parallel. This paper focuses on generic process improvement techniques that can be applied to any existing process developed using SAS® software executing on Symmetric Multiprocessing (SMP) computers.

Paper considers methodologies to analyze existing processes to identify opportunity for parallel processing. It goes on to describe the techniques gathering information about various steps executing in a process, obtaining details such as longest executing steps in a process and differentiating between steps which require more time to execute and those which require lesser time, differentiating between steps which are more efficient and those which are less efficient. The paper further discusses how to break out the process into various sub-processes and how to work on the sub-processes which require more time to execute. It further discusses various techniques that can be applied to analyze the data and break larger data sets into smaller pieces to allow faster processing. It also discusses parallelization techniques to execute the smaller data sets in parallel and later combine them together. The paper goes on to discuss how a job can be broken out into multiple instances based on the source data to perform same task better and faster. Further, we compare and discuss alternate ways for performing same tasks for improving processing time.

PARALLEL PROCESSING METHODOLOGY

In this section, we will discuss different types of parallel processing methodologies; indentify hardware and software requirements for being able to implement parallel processing and make using of multithreading and analyze how best the existing SAS process can be modified to undergo parallel processing.

Parallel processing refers to processing a single process by multiple CPUs simultaneously such that maximum throughput can be obtained. SAS supports following two ways of parallel processing:

- threaded I/O
- threaded application processing

THREADED I/O

Threaded I/O is preferred for applications which are I/O bound. This situation occurs when the application's data processing capability is higher that the application's input output processing capability. When most of the application's time is spent in waiting for reading or writing data and not waiting for CPU (or processing resources) then the application is termed as I/O bound.

SAS provides supports for threaded I/O with the use of the Scalable Performance Data (SPD) engine. SPD engine is a propriety SAS engine that supports processing of data by boosting the performance of the applications which are I/O bound by breaking out the datasets into smaller partitions and processing the data in parallel. Partitioned datasets can reside on multiple disk drives on the servers but are treated as a single dataset by the SPD engine.

In order to obtain maximum throughput for multiprocessing using threaded I/O it is required to have at least one controller per CPU and at least two disks. Although, in order to obtain better performance, having multiple disk drives for every CPU in a multiple CPU environment in an SMP will result in faster results.

During the course of analysis it was found that the process on which case study is performed was CPU intensive and therefore paper limits the discussion to threaded application processing only and does not consider any examples for the threaded I/O. If you are interested in exploring threaded I/O, please see *SAS Scalable Performance Data Engine: Reference* for full details about this engine's capabilities.

THREADED APPLICATION PROCESSING

Threaded application processing is generally performed on applications that are CPU bound. This situation occurs when the application's data processing capability is lower that the application's input output processing capability. In such cases, most time is spent on waiting of the Central Processing Unit (CPU) to be made available to the process.

Generally when a process is submitted in SAS, single threaded processing occurs and 1 CPU is utilized per process when one process is running at a time. When multiple processes are executed at the same time on 1 CPU then time slicing occurs and wait times are increased due to thrashing, as multiple processes are contending for the use of one CPU. On an SMP multiprocessing system use of multithreading can be used to overcome this issue. Multithreading means having to modify the process to allow having to process multiple threads in parallel across multiple CPUs.

Some types of processes may not be suited for threaded application processing. One such example is a process that requires having to sort a dataset. If the dataset to be sorted is divided into multiple smaller datasets and then each of the smaller datasets are sorted in separate threads, when it is required to combine the sorted output individual datasets together, the need will arise to again sort the final combined output dataset. Multithreading best works on applications processing serially independent processes, for example applications which use statistical models for forecasting loan losses in banking. Such applications process every observation within the dataset separately and generally have the need to expand the data to create time series.

For SAS 9, certain procedures such as SORT and MEANS have been modified so that they can thread the processing through multiple CPUs, if they are available. In addition, threaded processing is being integrated into a variety of other SAS® software features in order to improve performance. For example, when you create an index, if sorting is required, SAS attempts to sort the data using the thread-enabled sort

ARCHITECTURE

In this section, we will discuss the hardware, software, data and process requirements in order to successfully implement threaded processing for applications developed using SAS® software. During this analysis we will consider a case study on a practical real life process developed using SAS® software, which was modified for multiprocessing. This is accomplished with the use of MP connect technology by spawning multiple SAS session to run multiple proc steps in parallel and pipe their output into another SAS session running another proc. This pipeline can be extended to include any number of steps and can even extend between different physical machines.

SYMMETRIC MUTIPROCESSING (SMP) ARCHITECTURE

Symmetric multiprocessing (SMP) architecture by definition is system having multiple processors connected to a shared memory, having shared I/O devices which are controlled with a single operating system. SMP architecture generally has multiple Central Processing Units (CPUs) and an operating system which can manage individual jobs running on the server.

SMP computers having multiple CPUs are most beneficial for threaded application processing. Using threaded application processes, an application developed using SAS® software can be broken into multiple threads, each of which is processed simultaneously on multiple CPUs in parallel. A thread is defined as a single instance of an independent flow of control through a program or within a process. Threading allows dividing serially independent processes into multiple sub-processes which can take advantage of a multiprocessor system.

During the course of this paper, the case study was performed on an UNIX AIX server environment having 40 shared CPUs.

SYMMETRIC MUTIPROCESSING (SMP) PROGRAMMING

Single processing programming and multi-processor programming are slightly different in the many ways. During uniprocessor programming a programmer will program his process to execute even the serially independent steps sequentially, where as in a multiprocessing environment, a programmer will program his process to execute the serially independent steps in parallel. It is possible that programs executing on an SMP environment will increase a performance increase even when they are written for a single processor environment since the hardware interrupts that usually suspend the execution of the program while the kernel handle's the threads can execute on an idle processor instead.

The effect of multiprocessor programming can be estimated as follows, the processing time of the process will reduce by the number of multiprocessors uses for processing. The improvement factor is nearly the same as the number of additional processors added for processing. Generally most of the popular commercial operating systems have support for multiprocessing SMP applications built in, such that multiple processors are utilized for performing SMP processes.

For SAS procedures that are thread-enabled, new SAS system options are introduced with SAS 9:

CPUCOUNT

Specifies how many CPUs can be used.

THREAD|NOTHREADS

Controls whether to use threads.

SYMMETRIC MULTIPROCESSING (SMP) PERFORMANCE

As compared to systems running single process, an SMP environment will be able to process many jobs in parallel thereby having a considerably better performance than a single processor system which processes only a single job at a time. Symmetric multiprocessing systems performance depends on the number of CPUs that are available for processing, number of user's accessing the same CPUs, type of processes processing threaded IO processing.

In an SMP system, sharing fewer CPUs, among many users, who are executing highly CPU intensive tasks, will surely provide lesser throughput as compared to a server sharing more CPUs, among fewer users who are performing less CPU intensive tasks. If multiple user's are executing highly CPU intensive processes for multiple threads then this results in contention for resources and lesser resources are available for users to process the data resulting in a lower throughput. In some experiments, we were able to see that jobs took over 3 times longer to process when multiple processes were processed in parallel larger than the number of available CPUs.

KNOWING THE HARDWARE ARCHITECTURE

The central processing unit (CPU) is the portion of a computer hardware that does the bulk of the processing of the computer programs, performs the analytical, logical and I/O operations of the system. A typical AIX UNIX server has multiple processors thereby allowing processing of multiple queries concurrently. Certain procedures in SAS® software version 9.2 have been modified to take advantage of multiple CPUs by threading the procedure processing. The Base SAS engine also uses threading to create indexes. The CPUCOUNT= option provides the information that is needed to make decisions about the allocation of threads.

As per the SAS® software version 9.2 installation recommendations for UNIX, to verify that you are you using a 64-bit Power processor, enter the following command:

$ lsconf |egrep 'Processor Type|CPU Type'

The response will be in the following format:

Processor Type: xxx

CPU Type: xxx

If the Processor Type field contains a reference to a Power CPU (for example, PowerPC_POWER4), the CPU Type must be 64-bit in order to use a Power CPU

SAS® software runs on AIX using either the 32 or 64-bit kernel; however, the 64-bit kernel is recommended so that system may be able to take advantage of the full functionality of SAS® software. It is suggested to use the 32-bit kernel only if using third-party software requires it. To determine what kernel type the AIX system is running on, perform the following command:

/usr/sbin/lsconf -k

If the kernel type is 32-bit, the output will be "Kernel Type: 32-bit". Similarly, if the kernel type is 64-bit, the output will be "Kernel Type: 64-bit".

The UNIX command lscfg displays configuration, diagnostic and vital product data (VPD) information about the system including CPU information. Processing speed of the CPU in hertz (Hz) on AIX 5.1 and subsequent releases can be found out from the following code:

lsattr -E -l proc0 | grep "Processor Speed"
Alternatively pmcycles command can be used to list the processor speed.

During the scope of this paper, the research and case study for modifying existing process to process in parallel has been carried out on a UNIX AIX® SMP server having the SAS® software version 9.2 package installed on it having the following configuration:

```
System Model: IBM,XXXX-XXX
Machine Serial Number: XXXXXXX
Processor Type: PowerPC_POWER7
Processor Implementation Mode: POWER 7
Processor Version: PV_7_Compat
Number Of Processors: 40
Processor Clock Speed: XXXX MHz
CPU Type: 64-bit
Kernel Type: 64-bit
Memory Size: XXXXXX MB
Good Memory Size: XXXXXX MB
Platform Firmware level: XXXXX_XXX
Firmware Version: IBM,XXXXX_XXX
Console Login: enable
Auto Restart: true
Full Core: true
```

MP CONNECT

The MP CONNECT feature within the SAS/CONNECT® software was introduced in SAS 8 to provide scalability and parallel processing capabilities. MP CONNECT enables users to overcome the limitations of a single threaded SAS 8 system and to multiply the scalability of a threaded SAS 9 system by allowing users to spawn multiple SAS sessions to run in parallel, each performing a portion of a larger application. One of the strongest features of MP CONNECT is its ability to not only to scale up to take advantage of SMP hardware but to also scale out and take advantage of an unlimited number of workstations across a network.

In SAS 9, the MP CONNECT feature of SAS/CONNECT® software has been enhanced with a new piping capability which allows overlapped execution of SAS data steps and/or certain SAS procedures. With this new support of pipeline parallelism, it is possible to feed the output of one SAS data step or procedure as input into the next SAS data step or procedure for overlapped execution. The net result is a reduction in the elapsed time necessary to complete the entire application.

The benefits of MP CONNECT include:

- distribute multiple independent units of work
- "divide and conquer" - repeatedly distribute small pieces of a problem to multiple processors until the whole problem is solved
- overlap SAS procedure and/or data step execution
- scale up and scale out

MP CONNECT provides a convenient syntax for spawning n SAS sessions to simultaneously execute n tasks as independent processes and coordinate the execution and results of all n tasks into the original SAS session. The n SAS sessions or processes can all execute on the same SMP machine with each session or process running on a separate processor. However, if the user doesn't have the horsepower of an SMP machine, MP CONNECT has the flexibility to spawn multiple SAS sessions to run on any number of remote machines across a network. The remote machines can have either single or multiprocessor capabilities. Single processor remote workstations are not only much less expensive than large SMP machines, but they each come equipped with their own CPU, disk drive(s), and I/O channel and can be unlimited in number!

MP CONNECT enables users to perform multiprocessing with SAS by establishing a connection between multiple SAS sessions and enabling each of the sessions to asynchronously execute tasks in parallel. Users can also merge the results of the asynchronous tasks into local execution stream at the appropriate time. In addition, establishing connections to processes on the same local computer has been greatly simplified. This enables the user to exploit SMP hardware as well as network resources to perform parallel processing and easily coordinate all the results into the client SAS session.

Users can use MP CONNECT to start any number of SAS processes that they want to perform in parallel. SAS processes that are started on a single multiprocessor computer are independent, unique processes just as they are if they are initiated on a remote host. For example, under Windows and UNIX, each SAS session is a separate process that has its own unique SAS WORK library. Each process also assumes the user context of the parent or of the user that invoked the original SAS session, and has all the rights and privileges that are associated with that parent. The client's SASHELP, SASMSG, SASAUTOS, and CONFIG allocations are passed to the new session as SAS option values.

MP CONNECT is implemented by executing an RSUBMIT statement and the CONNECTWAIT=NO option. This method causes SAS/CONNECT® software to submit a task to a server session for processing and return control immediately to the client session so that user’s can start other tasks in the client session or in other server sessions.

PIPING

Piping is an extension of the MP CONNECT functionality whose purpose is to address pipeline parallelism. Piping enables you to overlap the execution of SAS data steps and/or certain SAS procedures. This is accomplished by spawning one SAS session to run one data step or proc and pipes its output through a TCP/IP socket as input into another SAS session running another data step or proc. This pipeline can be extended to include any number of steps and can even extend between different physical machines. The benefits of piping include:

- overlapped execution of proc and/or data step
- eliminate intermediate write to disk
- improved performance
- reduced disk space requirements

PROCESS ANALYSIS

In order to get maximum throughput with the use of SAS MP Connect software, the user must analyze the SAS process to how best can it be modified to undergo multithreading using SAS MP Connect feature. For performing any kind of process improvement, it is suggested to analyze and examine the current process very well. In order to be able to analyze the process thoroughly an approach to break the process into DATA steps and PROC steps was found useful. The FULLSTIMER option within SAS software can be used while executing the SAS codes to check the execution time for each individual step from the SAS log. FULLSTIMER option in SAS provides step by step statistics for every step in a program. This option can help pinpoint performance problems down due to a specific step.

For the sake of discussion, let us broadly categorize user applications or processes to be optimized using SAS MP connect software’s capabilities into following two types:

a. Parallel SAS process

b. Parallel threads within a SAS process

PARALLEL SAS PROCESS

A SAS process consists of many pieces, including execution units, data structures, and resources. A process corresponds to an operating environment process. A process has a largely private address space. It is scheduled by the operating environment, and its resources are managed by the operating environment at the lowest level. Multiple SAS processes use multiple processors on an SMP computer, but they can also be run on multiple remote single or multiprocessor computers on a network. When running multiple SAS processes on an SMP computer, SAS does not schedule a specific process to a specific processor; scheduling is controlled by the operating environment. MP CONNECT provides the ability to run multiple SAS processes.

Consider a hypothetical example of multiple processes as shown below:

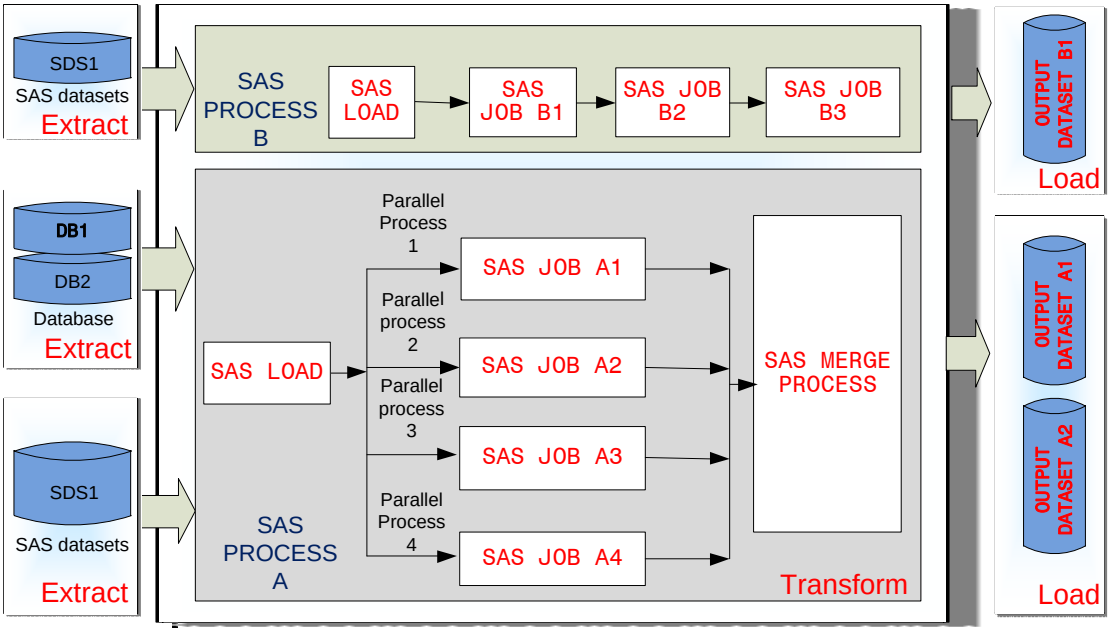


Figure 1. Multiple SAS processes executing in parallel

Consider a hypothetical example of multiple processes as shown above. It consists of two SAS processes. SAS process A and SAS process B. Both the SAS processes follow all of Extract-Transform-Load operations from to end to end but are completely independent of each other. Neither the data sources nor the transformations within the SAS processes A and B are dependent on each other. Performing process analysis on such completely independent processes it can be found that these processes are completely independent of each other and they can be submitted using RSUBMIT within the same SAS session.

MP CONNECT can be used to start multiple SAS sessions to execute independent units of work in parallel. The client session can synchronize the execution of the parallel tasks for subsequent processing. For this example, two SAS sessions would be started, and each session would perform one of the SAS processes.

PARALLEL THREADS WITHIN A SAS PROCESS

A process consists of one or more threads. A thread is also scheduled by the operating environment, but the running process might influence the behavior of threads by using synchronization techniques. All threads in a process share an address space and must cooperatively share the resources of the process. Multiple threads use multiple processors on an SMP computer but cannot be executed across computers. When running multiple threads within a SAS process, SAS does not schedule a specific thread to a specific processor; scheduling is controlled by the operating environment

Also identifying serially independent steps within a process can be helpful to determine whether a process can be scaled. If there are any such steps, it is recommended to modify the SAS program by creating separate programs for each step and executing them in parallel. This will reduce the wait time of the independent processes thereby allowing execution of independent steps in parallel. Consider a hypothetical example of a process which has 5 steps in total.

Step 1: Performs complex of analytical calculations to create dataset DataA

Step 2: Sorts the dataset DataA

Step 3: Performs another set of complex analytical calculations to create dataset DataB,

Step 4: Sorts the datasets DataB and

Step 5: Merges both the datasets DataA and DataB to create DataC.

```
/* PROGRAM TO ANALYSE TIME OF EXECUTION OF EACH PROCESS */

OPTIONS FULLSTIMER; /* FULLSTIMER option provides statistics in log*/

DATA dataA;
  <complex analytical calculations>;
RUN;

PROC SORT; by PrimaryKey; RUN;

DATA dataB;
  <complex analytical calculations>;
RUN;

PROC SORT; by PrimaryKey; RUN;

DATA dataC;
  Merge dataA
        dataB;
  By PrimaryKey;
RUN;
```

Currently the total ttime of execution for the above SAS program is equal to the sum of time required for executing each individual step. Total time (Ta) = t1 + t2 + t3 +t4 +t5

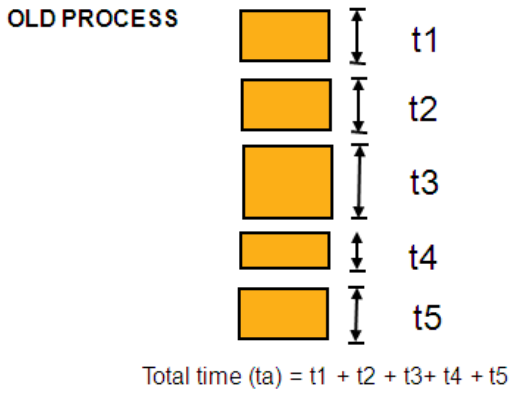


Figure 2. Process executing serially

Consider an approach when the two independent steps for creating and sorting dataset DataA and creating and sorting dataset DataB set are broken out into two separate threads. These threads would be executed in parallel and the output would be combined in a third program. Therefore, the total time required to execute the jobs in parallel would now be reduced to the time taken by the longest job executing in parallel in addition to any serial jobs thereafter. Therefore the time of execution of the above SAS program would now be reduced to Total Time (tb) = t1+t2+t5.

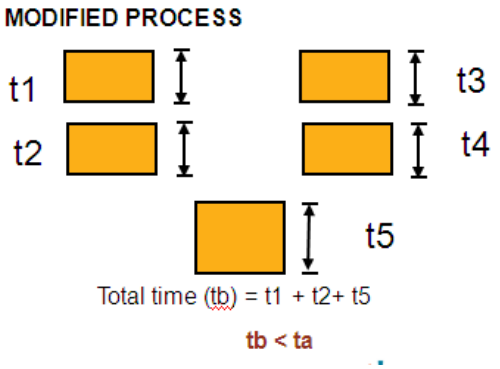


Figure 3. Process executing in parallel

CASE STUDY: PARALLEL PROCESSING

In the following section, we will consider a practical example of a process that was first executed serially and then apply parallel processing and multithreading techniques using the MP CONNECT feature of SAS/CONNECT® software to execute the same in parallel. Please note that the following case study was carried out on a UNIX AIX SMP server having 40 shared CPUs. A hypothetical process has been designed only to showcase step by step method to apply the parallel processing techniques and perform a practical benchmarking exercise to show the Return over Investment (ROI) of implementing multithreading technique using the MP CONNECT feature of SAS/CONNECT® software for this particular example.

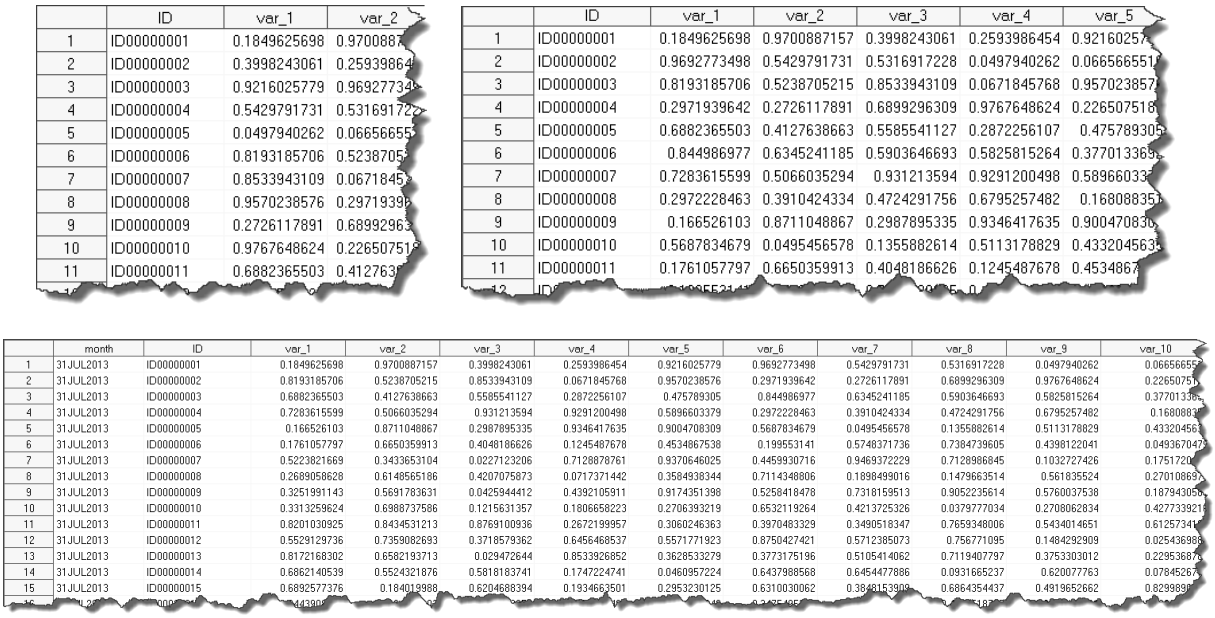
TEST DATA CREATION

Using the test data creation process we will create sample test datasets which is used for enumerating the case study for parallel processing. Sample datasets will be created for 3 different cases having increasing number for variables and same number of observations to analyze effect on longer datasets respectively. In order to create the test datasets for the process the following macro was used.

```
libname ip "/<input>/<directory>/<path>";

options nomprint nomlogic nosymbolgen;
%macro createdata(vars=,obs=);
  %if &obs < 1 %then %let &obs=1;
  %if &vars < 1 %then %let &vars=1;
  data ip.random_data_&vars._&obs.;
       length ID $10. ;
    do i=1 to &obs.;
      ID = "ID"||put(i,z8.);
      %do j=1 %to &vars.;
        var_&j.=ranuni(1);
      %end;
      output;
    end;
    drop i ;
  run;
%mend createdata;
%createdata(vars=2,obs=500000)
%createdata(vars=5,obs=500000)
%createdata(vars=10,obs=500000)
%createdata(vars=20,obs=500000)
```

The createdata macro above is a generic macro that was used to create all the input datasets used for testing. Two parameters are passed as macro variables to the macro, which specify the number of variables and the number of observations respectively to be contained in the generated datasets to be used as inputs. The macro makes use of the ranuni SAS® software function which is used to generate as many random variables specified in the vars parameter. An ID variable is created to uniquely identify every observation.



	ID	var_1	var_2
1	ID00000001	0.1849625698	0.9700887
2	ID00000002	0.3998243061	0.2593986
3	ID00000003	0.9216025779	0.9692773
4	ID00000004	0.5429791731	0.5316917
5	ID00000005	0.0497940262	0.0665665
6	ID00000006	0.8193185706	0.5238705
7	ID00000007	0.8533943109	0.0671845
8	ID00000008	0.9570238576	0.2971939
9	ID00000009	0.2726117891	0.6899296
10	ID00000010	0.9767648624	0.2265075
11	ID00000011	0.6882365503	0.4127638

	ID	var_1	var_2	var_3	var_4	var_5
1	ID00000001	0.1849625698	0.9700887157	0.3998243061	0.2593986454	0.9216025779
2	ID00000002	0.9692773498	0.5429791731	0.5316917228	0.0497940262	0.0665665551
3	ID00000003	0.8193185706	0.5238705215	0.8533943109	0.0671845768	0.9570238576
4	ID00000004	0.2971939642	0.2726117891	0.6899296309	0.9767648624	0.2265075181
5	ID00000005	0.6882365503	0.4127638663	0.5585541127	0.2872256107	0.4757893054
6	ID00000006	0.844986977	0.6345241185	0.5903646693	0.5825815264	0.3770133634
7	ID00000007	0.7283615599	0.5066035294	0.931213594	0.9291200498	0.5896603379
8	ID00000008	0.2972228463	0.3910424334	0.4724291756	0.6795257482	0.1680883511
9	ID00000009	0.166526103	0.8711048867	0.2987895335	0.9346417635	0.9004708309
10	ID00000010	0.5687834679	0.0495456578	0.1355882614	0.5113178829	0.4332045631
11	ID00000011	0.1761057797	0.6650359913	0.4048186626	0.1245487678	0.4534867538

	month	ID	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9	var_10
1	31JUL2013	ID00000001	0.1849625698	0.9700887157	0.3998243061	0.2593986454	0.9216025779	0.9692773498	0.5429791731	0.5316917228	0.0497940262	0.0665665551
2	31JUL2013	ID00000002	0.8193185706	0.5238705215	0.8533943109	0.0671845768	0.9570238576	0.2971939642	0.2726117891	0.6899296309	0.9767648624	0.2265075181
3	31JUL2013	ID00000003	0.6882365503	0.4127638663	0.5585541127	0.2872256107	0.475789305	0.844986977	0.6345241185	0.5903646693	0.5825815264	0.3770133634
4	31JUL2013	ID00000004	0.7283615599	0.5066035294	0.931213594	0.9291200498	0.5896603379	0.2972228463	0.3910424334	0.4724291756	0.6795257482	0.1680883511
5	31JUL2013	ID00000005	0.166526103	0.8711048867	0.2987895335	0.9346417635	0.9004708309	0.5687834679	0.0495456578	0.1355882614	0.5113178829	0.4332045631
6	31JUL2013	ID00000006	0.1761057797	0.6650359913	0.4048186626	0.1245487678	0.4534867538	0.199553141	0.5748371736	0.7384739605	0.4398122041	0.0493670473
7	31JUL2013	ID00000007	0.5223821669	0.3433653104	0.0227123206	0.7128878761	0.9370646025	0.4459930716	0.9469372229	0.7128986845	0.1032727426	0.1751720
8	31JUL2013	ID00000008	0.2689058628	0.6148565186	0.4207075873	0.0717371442	0.3584938344	0.7114348806	0.1898499016	0.1479663514	0.561835524	0.2701086
9	31JUL2013	ID00000009	0.3251991143	0.5691783631	0.0425944412	0.4392105911	0.9174351398	0.5258418478	0.7318159513	0.9052235614	0.5760037538	0.1879430
10	31JUL2013	ID00000010	0.3313259624	0.6988737586	0.1215631357	0.1806658223	0.2706393219	0.6532119264	0.4213725326	0.0379777034	0.2708062834	0.4277339
11	31JUL2013	ID00000011	0.8201030925	0.8434531213	0.8769100936	0.2672199957	0.3060246363	0.3970483329	0.3490518347	0.7659348006	0.5434014651	0.6125734
12	31JUL2013	ID00000012	0.5529129736	0.7359082693	0.3718579362	0.6456468537	0.5571771923	0.8750427421	0.5712385073	0.756771095	0.1484292909	0.0254369
13	31JUL2013	ID00000013	0.8172168302	0.6582193713	0.029472644	0.8533926852	0.3628533279	0.3773175196	0.5105414062	0.7119407797	0.3753303012	0.2295368
14	31JUL2013	ID00000014	0.6862140539	0.5524321876	0.5818183741	0.1747224741	0.0460957224	0.6437988568	0.6454477886	0.0931665237	0.620077763	0.0784526
15	31JUL2013	ID00000015	0.6892577376	0.184019988	0.6204688394	0.1934663501	0.2953230125	0.6310030062	0.3848153909	0.6864354437	0.4919652662	0.8299890

Figure 4. Input datasets random_data_2_500000, random_data_5_500000 and random_data_10_500000 respectively

Using the createdata above macro three different input datasets will be created in the input directory path specified in the libname statement. Each of these three datasets random_data_2_500000, random_data_5_500000 and random_data_10_500000 contain 2 variables and 500000 observations, 5 variables and 500000 observations and 10 variables and 500000 respectively. These datasets will be used as input datasets for testing the execution of the serial process and the parallel process respectively.

DATA PROCESSING

For the sake of enumeration the following statistical process was developed which was highly mathematical, calculation intensive and analytical in nature. We make use of a sample code above to generate random numbers and the proc univariate procedure within SAS® software to devise the process and perform a number of statistical calculations on the generated random dataset. Thereafter, we will execute the proc univariate procedure for every numeric variables in the input dataset one by one and process the data serially and create one output dataset for every variable in the input dataset containing one observation displaying the distributive statistics, quantile statistics, robust statistics and hypothesis testing statistics from the proc univariate SAS® software procedure. The time of execution of this process is recorded as the benchmark time to execute when the process is executed serially.

Thereafter, the MP CONNECT feature of the SAS® software was used and the process was modified such that the data was processed by spanning multiple threads in parallel by processing multiple variables simultaneously. One output dataset was created for every variable in the input dataset in multiple threads and the benchmark times were noted for executing the process in parallel. The study was concluded by comparing and contrasting the resource utilization and the performance times for executing each of the two methods.

Following statistical calculations were performed for every variable within the input dataset:

Statistics	Variable	Description
Descriptive Statistics	N	number of nonmissing values, var_<N>
Descriptive Statistics	MEAN	the mean, var_<N>
Descriptive Statistics	STD	the standard deviation, var_<N>
Descriptive Statistics	SKEWNESS	skewness, var_<N>
Descriptive Statistics	USS	the uncorrected sum of squares, var_<N>
Descriptive Statistics	CV	the coefficient of variation, var_<N>
Descriptive Statistics	SUMWGT	sum of the weights, var_<N>
Descriptive Statistics	SUM	the sum, var_<N>

Descriptive Statistics	VAR	the variance, var_<N>
Descriptive Statistics	KURTOSIS	kurtosis, var_<N>
Descriptive Statistics	CSS	the corrected sum of squares, var_<N>
Descriptive Statistics	STDMEAN	the standard error of the mean, var_<N>
Descriptive Statistics	NMISS	number of missing values, var_<N>
Descriptive Statistics	NOBS	number of observations, var_<N>
Descriptive Statistics	MAX	the largest value, var_<N>
Descriptive Statistics	MIN	the smallest value, var_<N>
Descriptive Statistics	RANGE	the range, var_<N>
Descriptive Statistics	MODE	the most frequent value, var_<N>
Hypothesis Testing Statistics	MSIGN	the sign statistic, var_<N>
Hypothesis Testing Statistics	SIGNRANK	the signed rank statistic, var_<N>
Hypothesis Testing Statistics	PROBT	p-value of the t statistic, var_<N>
Hypothesis Testing Statistics	PROBM	p-value of the sign statistic, var_<N>
Hypothesis Testing Statistics	PROBS	p-value of signed rank stat, var_<N>
Hypothesis Testing Statistics	NORMALTEST	the normality test statistic, var_<N>
Hypothesis Testing Statistics	PROBN	p-value of normality test stat, var_<N>
Quantile Statistics	P99	the 99th percentile, var_<N>
Quantile Statistics	P95	the 95th percentile, var_<N>
Quantile Statistics	P90	the 90th percentile, var_<N>
Quantile Statistics	Q3	the upper quartile, var_<N>
Quantile Statistics	MEDIAN	the median, var_<N>
Quantile Statistics	Q1	the lower quartile, var_<N>
Quantile Statistics	P10	the 10th percentile, var_<N>
Quantile Statistics	P5	the 5th percentile, var_<N>
Quantile Statistics	P1	the 1st percentile, var_<N>
Quantile Statistics	QRANGE	the interquartile range, var_<N>
Robust Statistics	GINI	Gini's mean difference, var_<N>
Robust Statistics	MAD	MAD, var_<N>
Robust Statistics	SN	Sn, var_<N>
Robust Statistics	QN	Qn, var_<N>
Robust Statistics	STD_QRANGE	standard deviation from QRANGE, var_<N>
Robust Statistics	STD_GINI	standard deviation from GINI, var_<N>
Robust Statistics	STD_MAD	standard deviation from MAD, var_<N>
Robust Statistics	STD_SN	standard deviation from SN, var_<N>
Robust Statistics	STD_QN	standard deviation from QN, var_<N>

Figure 5. Statistics calculated using proc univariate SAS procedure

SERIAL PROCESSING

The hypothetical process devised above was executed serially first for obtaining the statistical results using the proc univariate procedure for every numeric variable in the input dataset one by one and the data was processed serially to create one output dataset for every variable in the input dataset containing one observation displaying the distributive statistics, quantile statistics, robust statistics and hypothesis testing statistics from the proc univariate SAS software procedure.

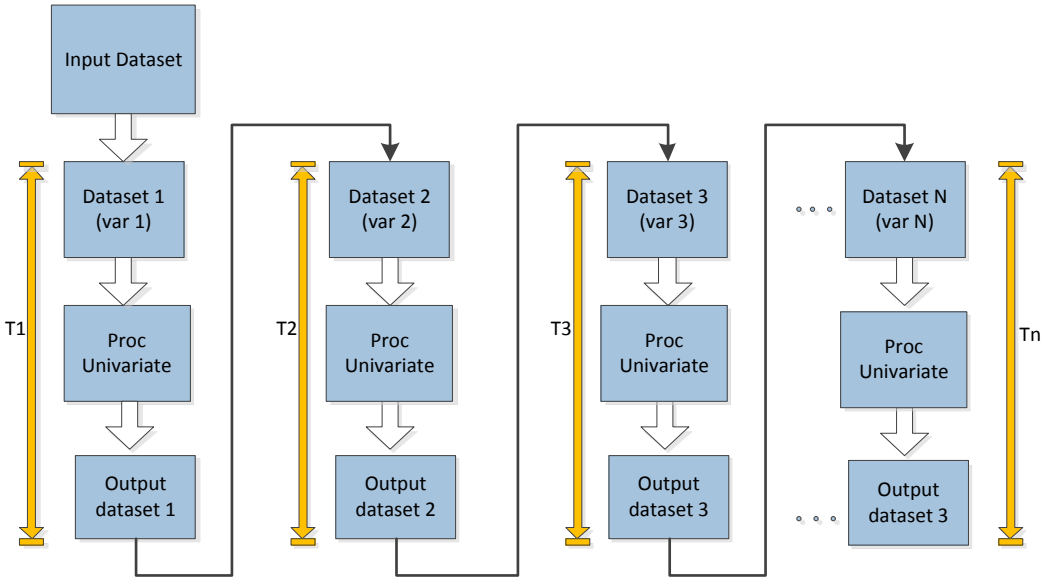


Figure 6. Process executing serially

As the data was processed serially one after other for every single variable in the input dataset, the total time required for processing was equal to the total time required for processing every single variables. For e.g. for processing the random_data_2_500000 input dataset containing 2 variables and 500000 observations, the total time required was equal to the time required for processing var_1 in addition to time required for processing var_2. Therefore total time T(T) = T1 + T2. Please see following extract from the log file which confirms the same. FULLSTIMER option was utilized to obtain the time required for every step in the process developed using SAS® software.

Process performance	Total	Var_1	Var_2
real time	1:03:26	0:34:09	0:29:16
user cpu time	0:38:12	0:19:02	0:19:10
system cpu time	0.08 seconds	0.04 seconds	0.03 seconds
memory	61027.18k	58212.43k	56332.93k
OS Memory	65092.00k	65092.00k	65092.00k
Timestamp	8/11/2013 12:36	8/11/2013 12:06	8/11/2013 12:36
Page Faults	0	0	0
Page Reclaims	57011	25128	23592
Page Swaps	0	0	0
Voluntary Context Switches	289	89	54
Involuntary Context Switches	146851	87071	59639
Block Input Operations	0	0	0
Block Output Operations	0	0	0

Figure 7. Process performance for process executing serially

Following code was written to read in the input dataset and execute the proc univariate on every variable in the input dataset to create one output dataset per variable containing the calculated statistics.

```
libname ip "/<input>/<directory>/<path>";
libname op "/<output>/<directory>/<path>";

options fullstimer;
/*UNIVARIATE */
%macro univariate(inlib=ip,indsn=,oplib=op,opdsn=,var=);
```

```
  %do i=1 %to &var;
    proc univariate data=&inlib..&indsn. noprint;
      var var_&i;
      output out=&oplib..&opdsn._&i.
      /* Descriptive Statistics */
      css=css cv=cv kurtosis=kurtosis max=max mean=mean
      min=min mode=mode n=n nmiss=nmiss nobs=nobs range=range
      skewness=skewness std=std  stdmean=stdmean sum=sum
      sumwgt=sumwgt uss=uss var=var
      /* quantile statistics */
      p1=p1 p5=p5 p10=p10 q1=q1 median=median q3=q3
       p90=p90 p95=p95 p99=p99 qrange=qrange
      /* robust statistics */
      gini=gini mad=mad qn=qn sn=sn std_gini=std_gini
      std_mad=std_mad std_qn=std_qn std_qrange=std_qrange
      std_sn=std_sn
      /* hypothesis testing statistics */
      msign=msign normaltest=normaltest signrank=signrank
      probm=probm probn=probn probs=probs probt=probt
      ;
    run;
  %end;
%mend univariate;
%univariate(inlib=ip,indsn=random_data_2_500000,
            oplib=op,opdsn=sp_uni_2v_500000o,var=2);
%univariate(inlib=ip,indsn=random_data_5_500000,
            oplib=op,opdsn=sp_uni_5v_500000o,var=5);
%univariate(inlib=ip,indsn=random_data_10_500000,
            oplib=op,opdsn=sp_uni_10v_500000o,var=10);
```

The ip and op libname statements will point to the input and output dataset directories respectively and the do loop will execute the process for the number of variables specified by the var parameter. The do loop will execute the process once for every variable specified by the vars parameter and create one output dataset per variable containing the statistics specified in the proc univariate procedure. The output datasets will create in the output directory specified by the op libname statement. For the case in question, two output datasets are created sp_uni_2v_500000o_1.sas7bdat and sp_uni_2v_500000o_2.sas7bdat. The layout of the output dataset is as shown below:

	N	MEAN	STD	SKEWNESS	USS	CV	SUMWGT	SUM	VAR	KURTOSIS	CSS
1	500000	0.5000913948	0.2886636003	-0.00128843	166708.95531	57.722169045	500000	250045.6974	0.0833266741	-1.197975707	41663.25

Figure 8. Sample output dataset (sp_uni_2v_500000o_1.sas7bdat) during serial processing

PARALLEL PROCESSING

The hypothetical process devised above was modified using the MP CONNECT feature of the SAS® software to be executed in parallel for obtaining the statistical results using the proc univariate procedure for every numeric variable in the input dataset in multiple threads and the data was processed in parallel to create one output dataset for every variable in the input dataset containing one observation displaying the distributive statistics, quantile statistics, robust statistics and hypothesis testing statistics from the proc univariate SAS® software procedure.

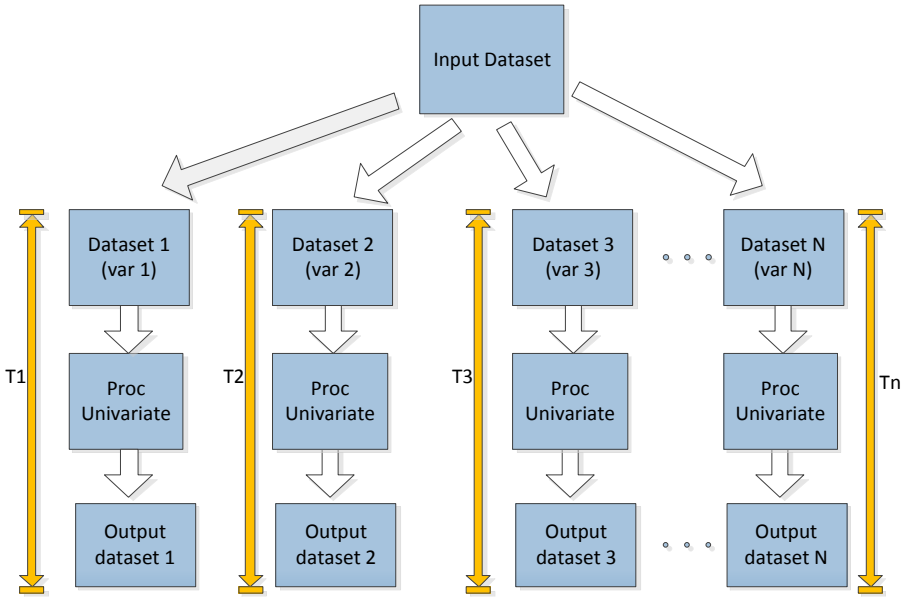


Figure 9. Process executing in parallel

As the data was processed in parallel threads for every variable in the input dataset, the total time required for processing was equal to the maximum time required for processing the variable which took the longest time. For e.g. for processing the random_data_2_500000 input dataset containing 2 variables and 500000 observations, the total time required was equal to the maximum time required for either processing var_1 or processing var_2. Therefore total time T(T) = MAX(T1 ,T2). Please see following extract from the log file which confirms the same. FULLSTIMER option was utilized to obtain the time required for every step in the SAS process.

Process performance	Total	Var_1	Var_2
real time	0:31:40	0:31:39	0:31:38
user cpu time	0.03 seconds	0:19:10	0:18:07
system cpu time	0.04 seconds	0.05 seconds	0.06 seconds
memory	4577.37k	61305.59k	61305.59k
OS Memory	5672.00k	65348.00k	65348.00k
Timestamp	8/11/2013 13:07	8/11/2013 13:07	8/11/2013 13:07
Page Faults	0	0	0
Page Reclaims	9529	35559	36269
Page Swaps	0	0	0
Voluntary Context Switches	1759	854	774
Involuntary Context Switches	175	50433	107646
Block Input Operations	0	0	0
Block Output Operations	0	0	0

Figure 10. Process performance for process executing in parallel

Following code was written to read in the input dataset and execute the proc univariate on every variable in the input dataset in parallel using the MP CONNECT feature of the SAS software in multiple threads to create one output dataset per variable containing the calculated statistics.

```
libname ip "/<input>/<directory>/<path>";
libname op "/<output>/<directory>/<path>";

%macro parallel_process(ilib=,idsn=,olib=op,odsn=);
  options fullstimer autosignon=yes sascmd="sas92 -nonews -threads";
  %global num_vars thread ;
  %let num_vars=%sysfunc(attrn(%sysfunc(open(&ilib..&idsn.,i)),nvars));

  %do thread = 1 %to (&num_vars-1);
```

```
    signon task&thread. wait=yes;
    %syslput thread         = &thread;
    %syslput ilib          = &ilib;
    %syslput idsn          = &idsn;
    %syslput olib          = &olib;
    %syslput odsn          = &odsn;

    rsubmit process=task&thread. wait=no sysrputsync=yes;
    libname ip "/<input>/<directory>/<path>";
    libname op "/<output>/<directory>/<path>";
    options fullstimer autosignon=yes sascmd="sas92 -nonews -threads";
    %macro univ_parallel;
        proc univariate data=&ilib..&idsn. noprint;
            var var_&thread.;
            output out=&olib..&odsn._&thread.
            /* Descriptive Statistics */
            CSS=CSS CV=CV KURTOSIS=KURTOSIS MAX=MAX MEAN=MEAN
            MIN=MIN MODE=MODE N=N NMISS=NMISS NOBS=NOBS RANGE=RANGE
            SKEWNESS=SKEWNESS STD=STD STDMEAN=STDMEAN SUM=SUM
            SUMWGT=SUMWGT USS=USS VAR=VAR
            /* Quantile Statistics */
            P1=P1 P5=P5 P10=P10 Q1=Q1 MEDIAN=MEDIAN Q3=Q3
            P90=P90 P95=P95 P99=P99 QRANGE=QRANGE
            /* Robust Statistics */
            GINI=GINI MAD=MAD QN=QN SN=SN STD_GINI=STD_GINI
              STD_MAD=STD_MAD STD_QN=STD_QN STD_QRANGE=STD_QRANGE STD_SN=STD_SN
            /* Hypothesis Testing Statistics */
            MSIGN=MSIGN NORMALTEST=NORMALTEST SIGNRANK=SIGNRANK
            PROBM=PROBM PROBN=PROBN PROBS=PROBS PROBT=PROBT
            ;
        run;
    %mend univ_parallel;
    %univ_parallel;
    endrsubmit;
%end;

waitfor _all_ %do thread = 1 %to (&num_vars-1);
                  task&thread
              %end;
;
%do thread = 1 %to (&num_vars-1);
  rget task&thread;
%end;
%do thread = 1 %to (&num_vars-1);
  signoff task&thread;
%end;
%mend parallel_process;
%parallel_process(ilib=ip,idsn=random_data_2_500000,
                  olib=op,odsn=pp_uni_2v_500000o);
%parallel_process(ilib=ip,idsn=random_data_5_500000,
                  olib=op,odsn=pp_uni_5v_500000o);
%parallel_process(ilib=ip,idsn=random_data_10_500000,
                  olib=op,odsn=pp_uni_10v_500000o);
```

The above code will be used to spawn as one less thread threads than the number of variables in the input dataset using MP CONNECT feature of the SAS® software to allow execution of proc univariate procedure for every variable except the ID variable in the input dataset. The ip and op libname statements point to the input and output directories respectively. The number of variables in the input dataset are counted and assigned to a macro variable num_vars using the file open function in the input mode to open the input SAS dataset and count the number of variables in the input dataset using the attrn SAS® software function.

The RSUBMIT statement specifies that the task is to run in a separate SAS® software session. The name of the task is specified by the task&thread_no (the name must have no more than 8 characters).WAIT=NO option is specified

such that the task processed asynchronously. The ENDRSUBMIT statement is added to end every task. SYSRPUTSYNC System Option sets %SYSRPUT macro variables in the client session when the %SYSRPUT statements are executed rather than when a synchronization point is encountered. The %SYSLPUT statement is a macro statement that is submitted in the client session to assign a value that is available in the client session to a macro variable that can be accessed from the server session. SAS must finish processing both tasks before announcing the jobs is complete. To ensure this, WAITFOR statement is added after the loop to ensure completion of all the threads.

Separate sessions are spawned in the do loops one for each variable in the input dataset. Since each task starts a separate SAS session, each task has its own WORK library. The data sets being created need to be processed in the "local" or parent SAS session. For the above case, two output datasets are created pp_uni_2v_500000o_1.sas7bdat and pp_uni_2v_500000o_2.sas7bdat simultaneously when the process is executed in parallel threads. The layout of the output dataset is as shown below:

	N	MEAN	STD	SKEWNESS	USS	CV	SUMWGT	SUM	VAR	KURTOSIS	CSS
1	500000	0.5000913948	0.2886636003	-0.00128843	166708.95531	57.722169045	500000	250045.6974	0.0833266741	-1.197975707	41663.25

Figure 11. Sample output dataset (pp_uni_2v_500000o_1.sas7bdat) during parallel processing

CONCLUSION

In conclusion it can be observed that the processing time for serially independent processes can be theoretically reduced by up to the number of CPUs available for processing using MP CONNECT feature of the SAS® software. The sample process was extrapolated to 2, 5,10 and 20 threads to execute in parallel an in each case it was observed that the processing time was reduced by ½, 1/5th,1/10th and 1/20th of the time required to execute the process serially. It was also observed that when the number of thread exceeded the number of available CPUs, the processing time reduced drastically due to thrashing that ensued after parallel threads were completing for resources.

Input dataset		Parallel Threads	Serial processing		Parallel processing		% Improvement	
Vars.	Obs.		Real Time (SP)	CPU Time (SP)	Real Time(PP)	CPU Time(PP)	Real Time	CPU Time
2	500000	2	1:03:26	0:38:12	0:31:40	0:00:03	50.08%	99.87%
5	500000	5	2:37:07	1:36:36	0:30:46	0:00:04	80.42%	99.93%
10	500000	10	5:05:35	3:13:29	0:29:20	0:00:04	90.40%	99.97%
20	500000	20	10:21:03	6:30:45	0:30:33	0:00:12	95.08%	99.95%

Figure 12. Comparison of processing time for parallel processing

From the above table it can be observed that on a 40 CPU UNIX AIX SMP server when the process was executed serially and in parallel for datasets having same number of observations (# 500000) and varying number of variables, it was observed that the processing time for the process was reduced by as many number of threads that were executed in parallel for the given case study. Percentage improvement up to 95% was observed for some cases.

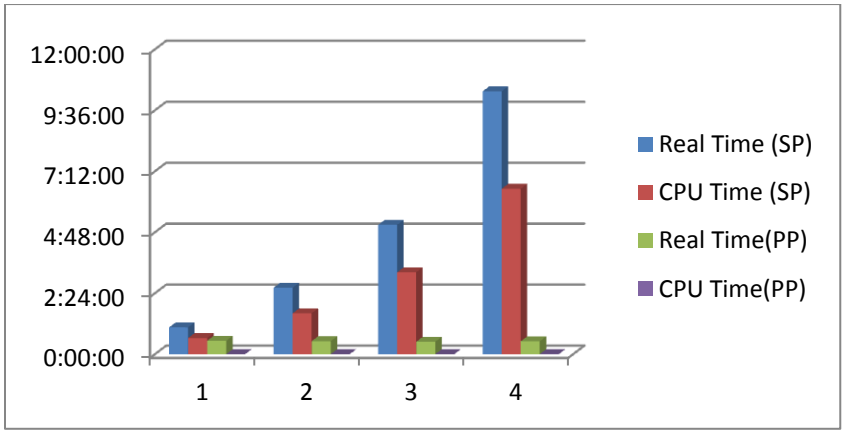


Figure 13. Comparison of processing time for parallel processing

Identifying areas of improvement gives the developer a precise idea of steps that are required to be worked upon and be modified to be executed in parallel. It is recommended to identify steps with high User CPU times and work on them, by using alternative methods to reduce overall time of the process. The developer can then focus on only analyzing and applying alternative strategies to further improve processing times of only such processes.

Parallelization technique yield better results when applied to best processes which have serially independent steps being executed one after the other. Careful planning and implementation would be required to indentify such steps and implement a solution around this to execute the same in parallel

Multithreading yields higher returns in situations where CPU time and the real time of a process are not far apart. It has been observed that for a system-intensive process, the CPU time and the Real time are not far apart, the ratio of CPU time and real time would be 1. Multi-threading is effective only for such system-intensive processes. In a production environment, current CPU available, current available memory, other heavy duty applications executing on the server, are some of the factors that govern the processing time of a multi-threaded process. It is suggested to execute the process multiple times during peak and off-peak hours to determine the average execution time of the process. Also, the number of threads in which the jobs should be broken out is an important factor which can be determined by executing the jobs multiple times and determining for among all the runs, one run for which run the difference between real time and CPU time was least during execution of the process.

REFERENCES

"SAS/CONNECT® 9.4 User's Guide" in SAS Online Doc, Version 9. Cary, NC: SAS Institute. *Copyright © 2013, SAS Institute Inc., Cary, NC, USA,*
http://support.sas.com/documentation/cdl/en/connref/64802/PDF/default/connref.pdf (September 25, 2011).

Shamlin, David. 2004. "Threads Unraveled: A Parallel Processing Primer". *The Twenty- Ninth Annual SAS Users Group International Conference Proceedings,* Montréal, Canada,
http://www2.sas.com/proceedings/sugi29/217-29.pdf (May 12, 2004).

Buchecker, Michelle M. 2005. "Parallel Processing Hands-On Workshop". *The Twenty-Ninth Annual SAS Users Group International Conference Proceedings*, Montréal, Canada,
http://www2.sas.com/proceedings/sugi29/124-29.pdf (May 12, 2004).

Kumbhakarna, Viraj. 2011. "A practical approach to parallel processing using SAS". *The 2011 MidWest SAS Users Group Annual Regional Conference Proceedings,* Kansas City, Kansas,
http://www.mwsug.org/proceedings/2011/posters/MWSUG-2011-PO06.pdf (September 25, 2011).

SAS Institute Inc. 2002. SAS OnlineDoc 9. Cary, NC: SAS Institute Inc.

ACKNOWLEDGMENTS

I would like to profusely thank Sam Zheng for being a wonderful guide and mentor during my work at JPMorgan Chase & Co. I would like to thank Luke Castellanos for encouraging me to write and publish a paper. I would also thank my employers JPMorgan Chase & Co. for providing me with an opportunity for attending the conference.

I would also like to mention the following authors: Shamlin, David and Buchecker, Michelle M who have granted me their kind permissions to reference their research in my paper. Finally, I would also like to thank my manager Zheng, Sam for his invaluable review comments and tremendous patience with my work.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name: Viraj R Kumbhakarna
Enterprise: JPMorgan Chase & Co.
Address: 8317 Falling Water Ln.
City, State ZIP: Columbus, OH, 43240
E-mail: vkumbhakarna@gmail.com

DISCLAIMER

The contents of the paper herein are solely the author's thoughts and opinions, which do not represent those of JPMorgan Chase & Co. JPMorgan Chase & Co. does not endorse, recommend, or promote any of the computing architectures, platforms, software, programming techniques or styles referenced in this paper.