PharmaSUG2011 - Paper PO03

A Practical Approach to Process Improvement Using Parallel Processing

Viraj Kumbhakarna, Cognizant Technology Solutions Corporation, Lake Hiawatha, NJ

ABSTRACT

In applications which process huge volumes of data for analysis purposes, it is often essential to minimize processing time to increase efficiency. Large data volumes (in the author's experience, row counts of 1.5 million and up to 2500 columns) result in very long processing times of about 4 to 5 days. It is often required to reduce the execution time for such processes. This paper discusses the following generic steps for improving any process. (1) Identify areas for improvement (understand the process thoroughly; analyze logs to identify steps which take the longest amount of time). (2) Look for certain processes that can be executed in parallel (actual parallelization of independent processes executed one after another; automated virtual multithreading; executing multiple instances of jobs in UNIX). (3) Adopt alternative methods for performing tasks faster (sorting datasets using threads; use of SET statement and KEY= option; use of SYNCSORT® for merging in SAS; intelligent use of indexes for merging; use of user-defined formats). (4) Modularize complex steps and creating macros for performing repeated tasks. (5) Reduce code redundancy (removing unwanted code; use of functions and macros for performing repeated tasks). (6) Syntactic optimization and deletion of large intermediate data sets. The above techniques optimize the process to allow faster data delivery by reducing the execution time.

INTRODUCTION

This paper focuses on generic process improvement techniques that can be applied to any existing SAS process. It begins right from analyzing an existing process 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.

IDENTIFYING AREAS OF IMPROVEMENT

For performing any kind of process improvement, it is very important to understand the current process very well. In order to be able to understand the process thoroughly it is required to break out the process into DATA steps and PROC steps. We use the FULLSTIMER option while executing the SAS codes to check the execution time for each 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. Consider the following example of a very simple SAS program. The following program creates two data sets – Employee and Salary, sorts them and merges them together to obtain the salary information of the employees.

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

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

DATA Employee;
  Input Id $ Name $;
  datalines;
  101 Tim
  102 John
  103 Jane
  ;
RUN;

PROC SORT; by Id; RUN;
```

```
DATA Salary;
  Input Id $ Salary;
  datalines;
  101 60000
  102 60000
  103 80000
  ;
RUN;

PROC SORT; by Id; RUN;

DATA Employee_Salary;
  Merge Employee
        Salary;
  By Id;
RUN;
```

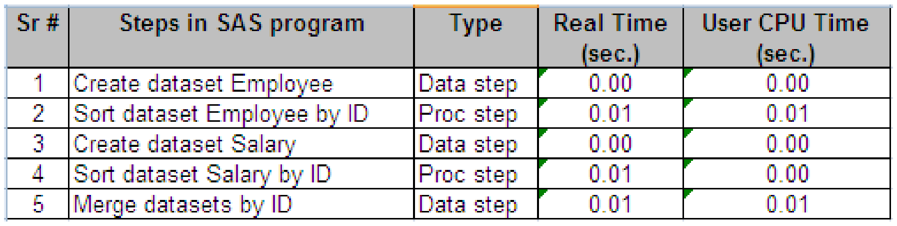
From the log file of the above program, the time required for execution of each step can be identified. We identify the steps which take longest amount of time to execute.

```
163  /*LOG TIME */
164  OPTIONS FULLSTIMER;
165  DATA Employee;
166    input Id $ Name $;
167    datalines;
NOTE: The data set WORK.EMPLOYEE has 3 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      user cpu time       0.00 seconds
      system cpu time     0.00 seconds
      Memory                            193k
      OS Memory                         16752k
      Timestamp            8/9/2010  12:37:58 AM
171    ;
172  RUN;

173  PROC SORT; by Id; RUN;

NOTE: There were 3 observations read from the data set WORK.EMPLOYEE.
NOTE: The data set WORK.EMPLOYEE has 3 observations and 2 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           0.01 seconds
      user cpu time       0.01 seconds
      system cpu time     0.00 seconds
      Memory                            94k
      OS Memory                         16752k
      Timestamp            8/9/2010  12:37:58 AM

174  DATA Salary;
175    input Id $ Salary;
176    datalines;

NOTE: The data set WORK.SALARY has 3 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      user cpu time       0.00 seconds
      system cpu time     0.00 seconds
      Memory                            193k
      OS Memory                         16752k
      Timestamp            8/9/2010  12:37:58 AM
180    ;
181  RUN;
```

```
182  PROC SORT; by Id; RUN;

NOTE: There were 3 observations read from the data set WORK.SALARY.
NOTE: The data set WORK.SALARY has 3 observations and 2 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           0.01 seconds
      user cpu time       0.00 seconds
      system cpu time     0.01 seconds
      Memory                            89k
      OS Memory                         16752k
      Timestamp            8/9/2010  12:37:58 AM
183  DATA Employee_Salary;
184    merge Employee
185          Salary;
186    by Id;
187  RUN;
NOTE: There were 3 observations read from the data set WORK.EMPLOYEE.
NOTE: There were 3 observations read from the data set WORK.SALARY.
NOTE: The data set WORK.EMPLOYEE_SALARY has 3 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      user cpu time       0.00 seconds
      system cpu time     0.01 seconds
      Memory                            238k
      OS Memory                         16752k
      Timestamp            8/9/2010  12:37:58 AM
```

Fig1.

Time required for executing each step in a process from Log file



Sr #	Steps in SAS program	Type	Real Time (sec.)	User CPU Time (sec.)
1	Create dataset Employee	Data step	0.00	0.00
2	Sort dataset Employee by ID	Proc step	0.01	0.01
3	Create dataset Salary	Data step	0.00	0.00
4	Sort dataset Salary by ID	Proc step	0.01	0.00
5	Merge datasets by ID	Data step	0.01	0.01

For more complex programs, it is suggested to list the time required for execution of all the steps. By comparing the Real time and User CPU times for each step, the user will be able to identify steps that require process improvement. Real Time represents the elapsed time or "wall clock" time. This is the time spent to execute a job or step. This is the time the user experiences in wait for the job/step to complete. Please refer the SAS support web site for more information on the Real Time and USER CPU times. As host system resources are heavily utilized the Real Time can go up significantly - representing a wait for various system resources to become available for the SAS job/step's usage. The User CPU Time is the time spent by the processor to execute user-written code. This is user-written from the perspective of the operating system and not the customer's language statements. 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.

* * * * * * * * * * * * * * *

PARALLEL EXECUTION

In the following section, we will consider the three different parallelization techniques that can be applied for processing the data in parallel.

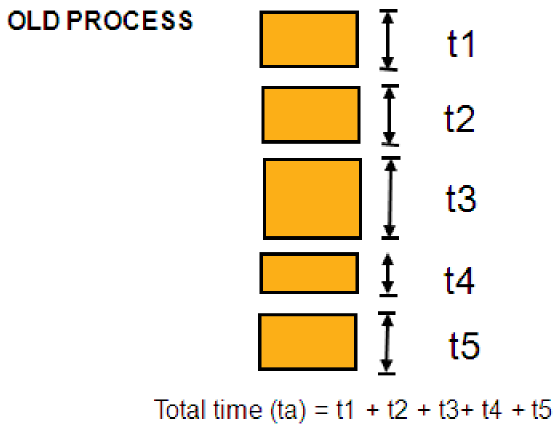
ACTUAL PARALLELIZATION

For being able to implement actual parallelization in the process, it is suggested to identify the serially executing independent steps in a SAS program. 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 the same example as stated above. Creation and sorting of the Employee data set and creation and sorting of the Salary data set are two independent processes. Currently the Time required for executing the above SAS program is equal to the sum of time required for executing each step. From the above example the Real time of execution of the above SAS program would be 0.01 + 0.01 + 0.01 = 0.03 sec.

Fig 2.

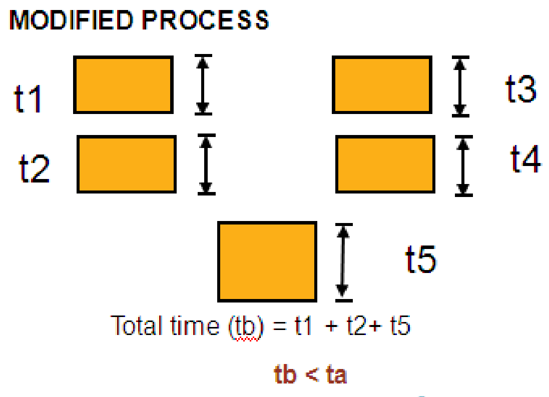
Process executing serially



Consider an approach when the two independent steps for creating and sorting employee data set and creating and sorting salary data set are broken out into two separate SAS programs. These programs would be executed in parallel and the output would be combined in a third program. Therefore, the 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 total time of execution of the above SAS program would now be reduced to 0.01 + 0.01 = 0.02 sec.

Fig 3.

Process executing in parallel



In the above scenario, 33.33% savings in processing time was obtained; since time of execution was reduced from 0.03 sec to 0.02 sec. Depending on the process a substantial saving in processing time can be obtained using the above technique of executing mutually independent steps in parallel.

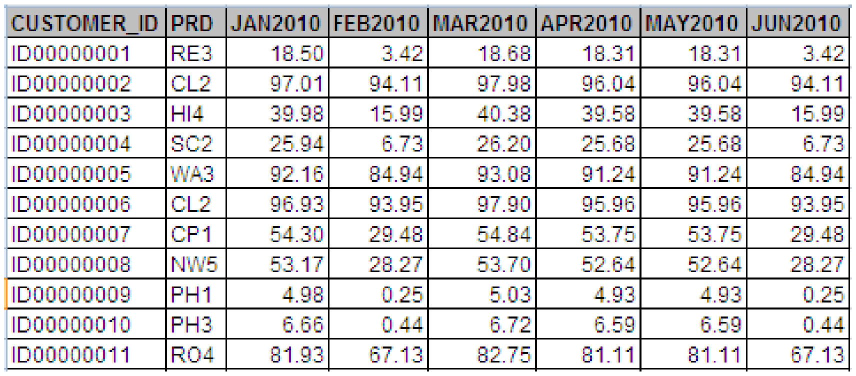
VIRTUAL MULTITHREADING

Virtual multithreading is a method that is introduced in the SAS SUGI 2007 paper, 036-2007 A Generic Method of Parallel Processing in Base SAS® 8 and 9. With the use of multithreading, a task which otherwise would have been executed serially can be executed in parallel. Multi-threading is faster as compared to serial processing since a task is split into number of partitions thereby executing each partition in parallel.

The technique suggested in Skosian's paper was applied to a hypothetical process for the sake of enumerating an example of virtual multi-threading technique. Consider two data sets from the database of an internet shopping website. The Larger data set is the Transaction dataset and the smaller data set is the Product Hierarchy data set. Transaction data set contains the monthly sales information by a customer for different products. Transaction dataset contains 15668100 observations and 8 variables. In the example considered, the product hierarchy dataset contains Department, Sub-department, Genre and Product details of a product for only two departments. Product hierarchy contains The requirement is to merge the transaction and the product hierarchy data sets on the PRD key variable and obtain the entire product hierarchy at the output data set for reporting semester sales. Please find the layout of the transaction data set, product hierarchy data set and the output data sets as below:

Fig 4.

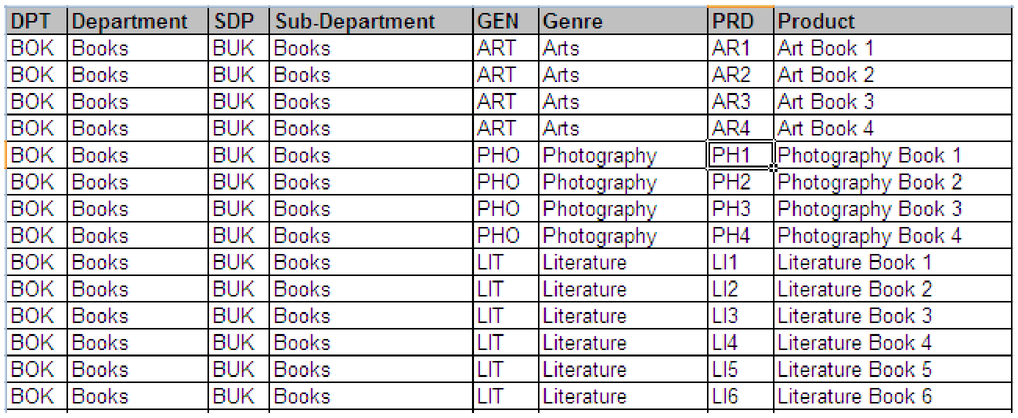
Transaction data set



CUSTOMER_ID	PRD	JAN2010	FEB2010	MAR2010	APR2010	MAY2010	JUN2010
ID00000001	RE3	18.50	3.42	18.68	18.31	18.31	3.42
ID00000002	CL2	97.01	94.11	97.98	96.04	96.04	94.11
ID00000003	HI4	39.98	15.99	40.38	39.58	39.58	15.99
ID00000004	SC2	25.94	6.73	26.20	25.68	25.68	6.73
ID00000005	WA3	92.16	84.94	93.08	91.24	91.24	84.94
ID00000006	CL2	96.93	93.95	97.90	95.96	95.96	93.95
ID00000007	CP1	54.30	29.48	54.84	53.75	53.75	29.48
ID00000008	NW5	53.17	28.27	53.70	52.64	52.64	28.27
ID00000009	PH1	4.98	0.25	5.03	4.93	4.93	0.25
ID00000010	PH3	6.66	0.44	6.72	6.59	6.59	0.44
ID00000011	RO4	81.93	67.13	82.75	81.11	81.11	67.13

Fig. 5

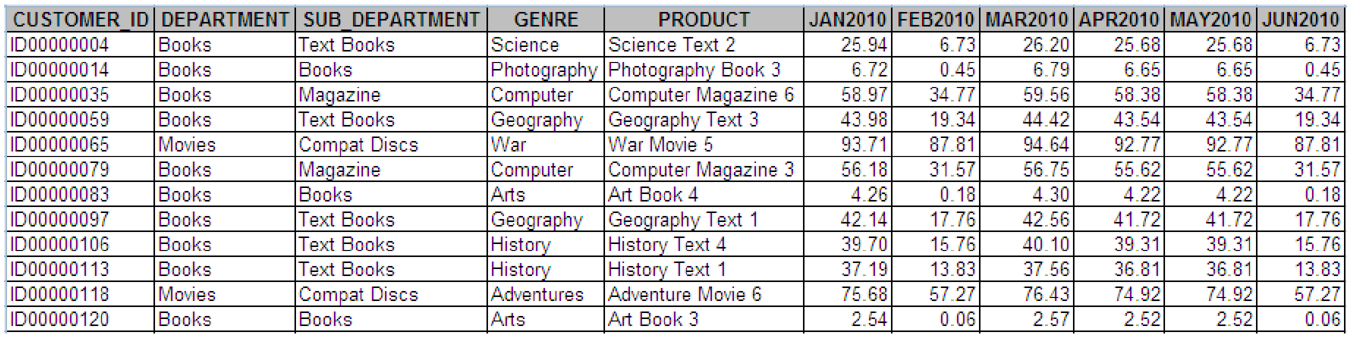
Product hierarchy data set



DPT	Department	SDP	Sub-Department	GEN	Genre	PRD	Product
BOK	Books	BUK	Books	ART	Arts	AR1	Art Book 1
BOK	Books	BUK	Books	ART	Arts	AR2	Art Book 2
BOK	Books	BUK	Books	ART	Arts	AR3	Art Book 3
BOK	Books	BUK	Books	ART	Arts	AR4	Art Book 4
BOK	Books	BUK	Books	PHO	Photography	PH1	Photography Book 1
BOK	Books	BUK	Books	PHO	Photography	PH2	Photography Book 2
BOK	Books	BUK	Books	PHO	Photography	PH3	Photography Book 3
BOK	Books	BUK	Books	PHO	Photography	PH4	Photography Book 4
BOK	Books	BUK	Books	LIT	Literature	LI1	Literature Book 1
BOK	Books	BUK	Books	LIT	Literature	LI2	Literature Book 2
BOK	Books	BUK	Books	LIT	Literature	LI3	Literature Book 3
BOK	Books	BUK	Books	LIT	Literature	LI4	Literature Book 4
BOK	Books	BUK	Books	LIT	Literature	LI5	Literature Book 5
BOK	Books	BUK	Books	LIT	Literature	LI6	Literature Book 6

Fig. 6

Output dataset obtained by merging Transaction and product hierarchy dataset



CUSTOMER_ID	DEPARTMENT	SUB_DEPARTMENT	GENRE	PRODUCT	JAN2010	FEB2010	MAR2010	APR2010	MAY2010	JUN2010
ID00000004	Books	Text Books	Science	Science Text 2	25.94	6.73	26.20	25.68	25.68	6.73
ID00000014	Books	Books	Photography	Photography Book 3	6.72	0.45	6.79	6.65	6.65	0.45
ID00000035	Books	Magazine	Computer	Computer Magazine 6	58.97	34.77	59.56	58.38	58.38	34.77
ID00000059	Books	Text Books	Geography	Geography Text 3	43.98	19.34	44.42	43.54	43.54	19.34
ID00000065	Movies	Compat Discs	War	War Movie 5	93.71	87.81	94.64	92.77	92.77	87.81
ID00000079	Books	Magazine	Computer	Computer Magazine 3	56.18	31.57	56.75	55.62	55.62	31.57
ID00000083	Books	Books	Arts	Art Book 4	4.26	0.18	4.30	4.22	4.22	0.18
ID00000097	Books	Text Books	Geography	Geography Text 1	42.14	17.76	42.56	41.72	41.72	17.76
ID00000106	Books	Text Books	History	History Text 4	39.70	15.76	40.10	39.31	39.31	15.76
ID00000113	Books	Text Books	History	History Text 1	37.19	13.83	37.56	36.81	36.81	13.83
ID00000118	Movies	Compat Discs	Adventures	Adventure Movie 6	75.68	57.27	76.43	74.92	74.92	57.27
ID00000120	Books	Books	Arts	Art Book 3	2.54	0.06	2.57	2.52	2.52	0.06

Virtual multi-threading technique was applied for the above datasets for creating the output dataset by merging the two input dataset together. Multi-threading was applied in the following three steps: a. splitting the input dataset into threads b. merge broken out datasets during multiple instances and c. combine broken out datasets to create output data set. Please find the code for the same as below:

```
%let dsn = transaction; ** Input dataset **;
libname in_lib "/analysis/NESUG/input"; ** Input library **;
libname out_lib "/analysis/NESUG/output"; ** Output library **;

%LET JOBNAME = MYJOB ;
*** Job name must be a unique name and should NOT have extensions **;
%LET NUM_PARTITIONS = 30; * Number of threads;
********* Randomly split the input dataset into equal subsets ****;
%MACRO RANDOM_SPLIT(INLIB=,OUTLIB=,DSN=,NUM_PIECES=);
DATA
%DO K=1 %TO &NUM_PIECES;
&OUTLIB..&DSN._&K
%END;;
SET &INLIB..&DSN;
RANDOM_NUMBER=RANUNI(0);
%DO K=1 %TO &NUM_PIECES;
%IF &K NE 1 %THEN ELSE; IF (&K-1)/&NUM_PIECES <= RANDOM_NUMBER <
&K/&NUM_PIECES THEN OUTPUT &OUTLIB..&DSN._&K;
%END;
;
DROP RANDOM_NUMBER;
RUN;
%MEND RANDOM_SPLIT;
%RANDOM_SPLIT(INLIB=IN_LIB,OUTLIB=OUT_LIB,DSN=&DSN,NUM_PIECES=&NUM_PARTITIONS);

OPTIONS OBS=0 NOSYNTAXCHECK FULLSTIMER SOURCE2;
*** Direct unnecessary portions of the log out of the main log file ***;
PROC PRINTTO LOG="./GARBAGE.LOG" NEW; RUN;

%MACRO PARTITION;
%DO PART_NUM=1 %TO &NUM_PARTITIONS;
FILENAME MPRINT "./&JOBNAME._PART&PART_NUM..sas" LRECL=170;
OPTIONS MPRINT MFILE;
%MACRO CODE;
OPTIONS FULLSTIMER SOURCE2;
LIBNAME IN_LIB "/analysis/NESUG/input/";
LIBNAME OUT_LIB "/analysis/NESUG/output/";

DATA OUT_LIB.OUTPUT_&JOBNAME._PART&PART_NUM (KEEP = CUSTOMER_ID DEPARTMENT
SUB_DEPARTMENT GENRE PRODUCT JAN2010 FEB2010 MAR2010 APR2010 MAY2010 JUN2010 JUL2010);
       RETAIN CUSTOMER_ID DEPARTMENT SUB_DEPARTMENT GENRE PRODUCT ;
       SET OUT_LIB.&DSN._&PART_NUM;
    SET IN_LIB.PRODHIER KEY = PRD / UNIQUE;
 RUN;
%MEND CODE;
%CODE
%END; ** num_partitions;
*** Redirect the log stream back to the main log file ***;
```

```
PROC PRINTTO LOG=LOG;RUN;
OPTIONS NOMFILE;
*** Create a shell script ***;
DATA _NULL_;
FILE "RUN_&JOBNAME..SH";
%DO PART_NUM = 1 %TO &NUM_PARTITIONS;
PUT "sas &JOBNAME._PART&PART_NUM..sas &";
%END;
PUT "wait";
RUN;
%MEND PARTITION;
%PARTITION

*** Execute the shell script ***;
X "chmod 755 RUN_&JOBNAME..SH";
X "RUN_&JOBNAME..SH";

*** Aggregate partitioned output datasets into one ***;
OPTIONS OBS=MAX PS=MAX NOCENTER;
%MACRO AGGREGATE;
DATA OUT_LIB.OUTPUT;
SET %DO PART_NUM = 1 %TO &NUM_PARTITIONS;
OUT_LIB.OUTPUT_&JOBNAME._PART&PART_NUM
%END;;
RUN;
%MEND AGGREGATE;
%AGGREGATE
```

Process was executed for splitting and merging a large transaction dataset with the smaller product hierarchy dataset to obtain the entire product hierarchy at the output. The transaction dataset was split into 40 smaller datasets and each dataset was processed separately to perform the merge step. Processing times and other statistics obtained from the log file for the overall job as shown below.

```
NOTE: The SAS System used:
      real time           2:26.51
      user cpu time       1:27.26
      system cpu time     23.92 seconds
      Memory                            10592k
      OS Memory                         14372k
      Timestamp            8/12/2010  2:53:41 AM
      Page Faults                       7338
      Page Reclaims                     10246
      Page Swaps                        0
      Voluntary Context Switches        231
      Involuntary Context Switches      14265
      Block Input Operations            0
      Block Output Operations           0
```

For comparison sake, let’s consider the example where the merge step was carried out by just using the set and key= option for creating the output. The log file below shows that Real time required was 2:57.60 sec. and the CPU time was 1:49.43 sec.

```
NOTE: The SAS System used:
      real time           2:57.60
      user cpu time       1:49.43
      system cpu time     14.01 seconds
      Memory                            3786k
      OS Memory                         4644k
      Timestamp            8/11/2010  1:25:07 AM
      Page Faults                       14331
      Page Reclaims                     3368
      Page Swaps                        0
      Voluntary Context Switches        36
      Involuntary Context Switches      7541
      Block Input Operations            0
      Block Output Operations           0
```

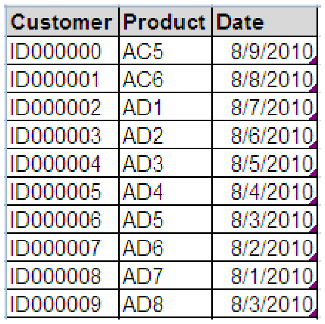
Therefore there has been a time savings of approx 18% with the use of virtual multithreading. Both the codes were executed one after another on the same AIX 64 UNIX server to ensure testing under similar memory, disk and CPU usage. During testing, it was observed that the time savings using the virtual multi-threading is a function of: a) amount of threads executed in parallel, b) current CPU available c) current memory available. Other concurrent processes may adversely affect the performance of the multi-threading technique.

EXECUTING MULTIPLE INSTANCE OF JOBS IN UNIX

Due to size limitations, we often come across a scenario, where the data is broken out by a recurrent parameter and certain processing is required to be done on the reoccurring files. Consider the following example which enumerates such a case. The transactions of purchases made by customers over the internet during a month are being obtained in Transaction data set. Transaction data set contains fields such as Customer ID, Product and Purchase date. Transaction data sets are broken out by departments for example; separate data sets for each product are listed. The transactions are at a product level, therefore there is product code. It is required to merge each of the market level transaction file with the Product Hierarchy to obtain the entire Hierarchy at the output. The Product Hierarchy data set contains information such as the Department code, Department name, Sub-Department code, Sub-Department name, Genre Code, Genre name, Product code and the Product name. Product code will be used to merge the Transaction data set and the about the product codes and product names. Please find the details of a transaction record from the Transaction data set and the details of the product hierarchy from the Product dataset below:

Fig 7.

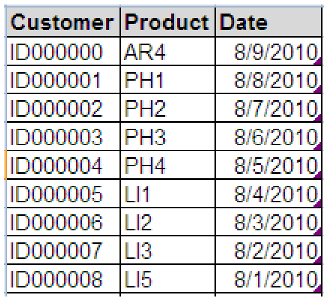
Transactions from Transaction_MOV.sas7bdat



Customer	Product	Date
ID000000	AC5	8/9/2010
ID000001	AC6	8/8/2010
ID000002	AD1	8/7/2010
ID000003	AD2	8/6/2010
ID000004	AD3	8/5/2010
ID000005	AD4	8/4/2010
ID000006	AD5	8/3/2010
ID000007	AD6	8/2/2010
ID000008	AD7	8/1/2010
ID000009	AD8	8/3/2010

Fig 8.

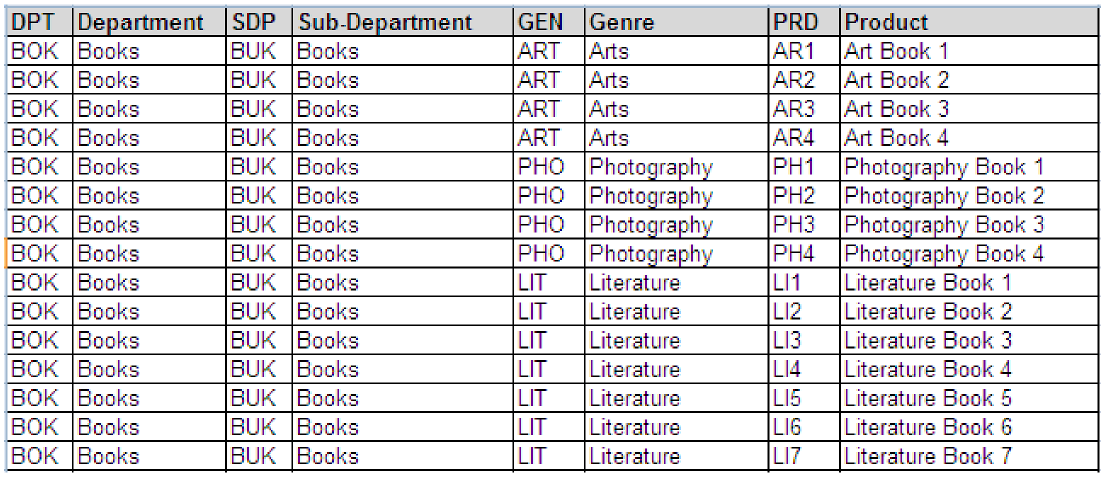
Transactions from Transaction_BOK.sas7bdat



Customer	Product	Date
ID000000	AR4	8/9/2010
ID000001	PH1	8/8/2010
ID000002	PH2	8/7/2010
ID000003	PH3	8/6/2010
ID000004	PH4	8/5/2010
ID000005	LI1	8/4/2010
ID000006	LI2	8/3/2010
ID000007	LI3	8/2/2010
ID000008	LI5	8/1/2010

Fig 9.

Sample records from the Product Hierarchy data set



DPT	Department	SDP	Sub-Department	GEN	Genre	PRD	Product
BOK	Books	BUK	Books	ART	Arts	AR1	Art Book 1
BOK	Books	BUK	Books	ART	Arts	AR2	Art Book 2
BOK	Books	BUK	Books	ART	Arts	AR3	Art Book 3
BOK	Books	BUK	Books	ART	Arts	AR4	Art Book 4
BOK	Books	BUK	Books	PHO	Photography	PH1	Photography Book 1
BOK	Books	BUK	Books	PHO	Photography	PH2	Photography Book 2
BOK	Books	BUK	Books	PHO	Photography	PH3	Photography Book 3
BOK	Books	BUK	Books	PHO	Photography	PH4	Photography Book 4
BOK	Books	BUK	Books	LIT	Literature	LI1	Literature Book 1
BOK	Books	BUK	Books	LIT	Literature	LI2	Literature Book 2
BOK	Books	BUK	Books	LIT	Literature	LI3	Literature Book 3
BOK	Books	BUK	Books	LIT	Literature	LI4	Literature Book 4
BOK	Books	BUK	Books	LIT	Literature	LI5	Literature Book 5
BOK	Books	BUK	Books	LIT	Literature	LI6	Literature Book 6
BOK	Books	BUK	Books	LIT	Literature	LI7	Literature Book 7

In the above example, it is required to merge the different department level transaction data sets with the product hierarchy data sets to obtain the entire product hierarchy for specified products at the output.

Instead of processing the job for merging and creating outputs by each Department, it is suggested to create a common job to perform the same process. A common code can be written to identify the various files at a department level and create a parameter file with the details of all the Departments for which the process is required to be executed. A common shell script can be created to execute the parameterized SAS program by concurrently executing the SAS program multiple times by passing the parameter value of Department one by one from the parameter file. Therefore, each department level file can be processed concurrently thereby reducing processing time. A shell script can be written to concurrently execute multiple instances of the same job in parallel.

ALTERNATIVE METHODS OF PERFORMING TASKS

In the following section, we will consider alternative ways which can be used to process the data and perform various tasks in a process. A simple case study was performed to assess the performance of merging two datasets using various techniques. Considering our previous example of merging the Transaction and Product Hierarchy dataset alternate methods of merging was utilized and their performance was analyzed. The same has been presented in the section below.

USE OF SET STATMENT AND KEY= OPTION

This method utilizes an index to join the two datasets together. It is required to create the indexes on the SAS data sets on the by variables, which would be used to merge two datasets. If the dataset is not already indexed by the variable on which one is sorting, one must create the index. An index can be created as follows: a. index in a data step, b. using PROC SQL or c. using PROC DATASETS index create option. One cannot explicitly tell the SAS System to use an index that has been created in processing. The SAS System will determine the most efficient way to process the merge step. Following code was utilized to merge the two datasets using set statement and key=option using index.

```
OPTIONS FULLSTIMER SOURCE2;

LIBNAME IN_LIB "/Info-One/adhoc/kumbhvi1/analysis/NESUG/input/";
LIBNAME OUT_LIB "/Info-One/adhoc/kumbhvi1/analysis/NESUG/output/";
DATA OUT_LIB.output (KEEP=CUSTOMER_ID DEPARTMENT SUB_DEPARTMENT GENRE PRODUCT
                    JAN2010 FEB2010 MAR2010 APR2010 MAY2010 JUN2010 JUL2010);
      RETAIN CUSTOMER_ID DEPARTMENT SUB_DEPARTMENT GENRE PRODUCT;
      SET IN_LIB.transaction;
      SET IN_LIB.PRODHIER KEY = PRD / UNIQUE;
RUN;
```

Performance statistics obtained from the log are as follows:

```
NOTE: SAS Institute Inc., SAS Campus Drive, Cary, NC USA 27513-2414
NOTE: The SAS System used:
      real time           2:57.60
      user cpu time       1:49.43
      system cpu time     14.01 seconds
      Memory                               3786k
```

```
OS Memory                                 4644k
Timestamp            8/11/2010  1:25:07 AM
Page Faults                               14331
Page Reclaims                             3368
Page Swaps                                0
Voluntary Context Switches                36
Involuntary Context Switches              7541
Block Input Operations                    0
Block Output Operations                   0
```

Performance statistics obtained from the logs state that the merge step by using index and key = option required a Real time of 2:57.60 sec and User CPU time of 1:49.43 sec for merging the two datasets.

USE OF SYNCSORT® FOR SORTING IN SAS

SYNCSORT® is a sort routine that can be purchased from Syncsort, Inc. One can use SYNCSORT® as an alternative sorting algorithm to that provided by the SAS System. To use SYNCSORT® with the SAS System, one is required to have the SYNCSORT® installed on the machine. Following test was carried out on an AIX 64 UNIX server, where SYNCSORT® was used for merging two flat files. In order to be able to use SYNCSORT®, it is required to have the source data in the form of flat files. In order to be able to process data using SYNCSORT®, the source datasets were first converted into flat files. Following technique can be used to create a SYNCSORT® program using SAS and execute it on the UNIX environment. In the technique described below, one can use the DATA _NULL_ step and file statement to create a SYNCSORT® file. All the commands and Please find the code for execution the merge step as follows:

```
/*TEST MERGING USING SYNCSORT */
OPTIONS COMPRESS=YES SORTPGM=HOST SORTANOM=BV;
FILENAME INSSY "/NESUG/SYN/SYNCMRG.SYN";

DATA _NULL_;
        FILE INSSY NEW LRECL=50000;
       PUT
        "/INFILE /NESUG/INPUT/TRANSACTION.TXT 45000 ""~"" " /
       "/JOINKEYS PRD" /
       "/INFILE /NESUG/INPUT/PRODUCTHIERARCHY.TXT 45000 ""~"" " /
       "/WORKSPACE /PROJ/SASTMP1" /
        "/WORKSPACE /PROJ/SASTMP2" /
       "/WORKSPACE /PROJ/SASTMP3" /
        "/WORKSPACE /PROJ/SASTMP4" /
       "/STATISTICS " /
       "/FIELDS       CUSTOMER_ID        1: -1:, " /
       "              PRD                2: -2:, " /
       "              JAN2010      3: -3:, " /
        "              FEB2010            4: -4:, " /
        "              MAR2010            5: -5:, " /
        "              APR2010            6: -6:, " /
       "              MAY2010      7: -7:, " /
        "              JUN2010            8: -8:, " /
       "              DPT          1: -1:, " /
       "              DEPARTMENT     2: -2:, " /
        "              SDP          3: -3:, " /
        "              SUB_DEPARTMENT     4: -4:, " /
        "              GEN          5: -5:, " /
        "              GENRE              6: -6:, " /
        "              PRD_0              7: -7:, " /
        "              PRODUCT            8: -8: " /
       "/JOINKEYS PRD_0 " /
        "/OUTFILE /NESUG/OUTPUT/OUTPUT_SYN.TXT 45000 OVERWRITE " /
        "/REFORMAT LEFTSIDE:  CUSTOMER_ID, "
       "RIGHTSIDE: "
       "DEPARTMENT , SUB_DEPARTMENT, GENRE, PRODUCT, " /
       "LEFTSIDE: "
       "JAN2010, FEB2010, MAR2010, APR2010, MAY2010, JUN2010"
       ;
       RUN;

       X "ECHO %NRSTR ('\012%%%%%%%%%%%%%%%%%%%') BEGINNING OF MERGE ROUTINE
       %NRSTR ('%%%%%%%%%%%%%%%%%%%\012') >> /NESUG/SYN/SYNCMRG_SYN.LOG";
```

```
        X "KSH /NESUG/SYN/TEMP1.SYN /NESUG/SYN/SYNCMRG.SYN
                                   2>> /NESUG/SYN/SYNCMRG_SYN.LOG";
```

Temp1.syn is used for invoking SYNCSORT®. It gives a call to the SYNCSORT® application and passes the source code to the script as a parameter for execution.

```
      #!/bin/ksh
      PATH=$PATH:/appl/syncsort/3.11/syncsort/bin
      export PATH
      # Start syncsort
      syncsort << endofsort
      /INSERT $1
      /END
      endofsort
```

Actual time taken for execution of the SYNCSORT® routine can be obtained from the log file of the SYNCSORT® program. From the log it is evident that a total of 1:26.98 seconds were required for sorting.

```
                                SyncSort statistics

Records read (left):          15,668,101  Data read (bytes) (left):   1,431,158,463
Records read (right):                136  Data read (bytes) (right):          8,006
Records paired (left):        15,668,101  Data paired (bytes) (left): 1,431,158,463
Records paired (right):              101  Data paired (bytes) (right):        5,616
Records unpaired (left):               0  Data unpaired (bytes) (left):           0
Records unpaired (right):             35  Data unpaired (bytes) (right):      2,390
Records output:               15,668,101  Data output (bytes):        2,003,103,999
Input record length (left):           92  Output record length:                 137
Input record length (right):          72
Work space used (bytes):     173,829,632

Elapsed time:                 0:01:26.98  CPU time:                      0:02:01.29
```

Statistics obtained from the log file of the driver SAS program show that the Real time required to execute the entire process right from outputting the SYNCSORT® program using put statement in SAS® to executing the SYNCSORT® code, required a total time of 1:28:74 sec.

```
      NOTE: SAS Institute Inc., SAS Campus Drive, Cary, NC USA 27513-2414
      NOTE: The SAS System used:
            real time           1:28.74
            user cpu time       0.02 seconds
            system cpu time     0.01 seconds
            Memory                            3238k
            OS Memory                         4132k
            Timestamp            8/11/2010  4:12:55 AM
            Page Faults                       0
            Page Reclaims                     2841
            Page Swaps                        0
            Voluntary Context Switches        46
            Involuntary Context Switches      19
            Block Input Operations            0
            Block Output Operations           0
```

It is required to explicitly state use of SYNCSORT® for sorting by specifying it in the SAS® system options. This can be achieved by inserting the following lines of code into one's program:

```
      OPTIONS sortpgm=host sortanom = bv;
```

Setting the sortpgm option to host specifies the use of host sort for sorting. Setting the SORTANOM option tells syncsort to run in multi-call mode instead of single-call mode. From the above example it can be observed that host sorts are generally faster than PROC SORT. Host sorts can be used with the propriety third party tool such as SYNCSORT® licensed. SYNCSORT® is uniformly faster than the native SAS® sort, and should always be used when available

USE OF USER-DEFINED FORMATS

Format procedure is generally used for creating one's own formats by redefining what value one wants to use for a given variable. Generally Format is used to assign certain values meaningful names. For example, in the data, one may refer the GENDER of a patient as M or F, but by using format it can be given a meaningful name of "MALE" or "FEMALE". But, alternatively it can also be used for merging two datasets. Alternatively, Format procedure can also be used to merge two data sets. In order to use format for merging, it is suggested to use a data step, and read in one of the datasets and set the value, variable type (character/numeric), and format name for the key variable. The CNTLIN option then builds the format using the specified dataset. In another data step, the second dataset is then set where the format is equal to the value assigned, returning only those records that meet specified selection criteria, thereby performing a merge. START is the value of the variable used to match, in this case PRD. LABEL represents the value START which will be assigned in the instance of a match, in this case, entire Product hierarchy. Please find the code used for merging the Transaction data set and the product hierarchy datasets for creation the output as follows:

```
LIBNAME INPUT "/INFO-ONE/ADHOC/KUMBHVI1/ANALYSIS/NESUG/INPUT";
LIBNAME OUTPUT "/INFO-ONE/ADHOC/KUMBHVI1/ANALYSIS/NESUG/OUTPUT";

DATA PRODUCTHIERARCHY;
SET INPUT.PRODUCTHIERARCHY(RENAME=(PRD=START));
LABEL='*';
FMTNAME='$CALL';
RUN;

PROC FORMAT CNTLIN=PRODUCTHIERARCHY;

DATA OUTPUT.OUTPUT_FMTMRG;
SET INPUT.TRANSACTION;
IF PUT(PRD,$CALL.) EQ '*';
RUN;
```

Please find the times of execution from the log file as below. From the log files, it can be seen that the merge step required a Real time of 1:01.54 sec and User CPU time of 48.02 sec. it is apparent that PROC FORMAT, is by far the fastest method used for merging, but it comes with its own set of limitations which are described below.

```
NOTE: SAS Institute Inc., SAS Campus Drive, Cary, NC USA 27513-2414
NOTE: The SAS System used:
      real time            1:01.54
      user cpu time        48.02 seconds
      system cpu time      9.15 seconds
      Memory                              4182k
      OS Memory                           4948k
      Timestamp             8/11/2010  2:22:53 AM
      Page Faults                         2948
      Page Reclaims                       3868
      Page Swaps                          0
      Voluntary Context Switches          28
      Involuntary Context Switches        3007
      Block Input Operations              0
      Block Output Operations             0
```

Though, it appears to be the fastest method amongst the ones compared above, there are a few limitations of the above method. First, the key variable used for merging has to be unique. Second, only matching records will be returned at the output. Third, only one variable from the format (smaller) dataset will be merged and obtained at the output. For example, only the PRD variable from the Product Hierarchy will appear at the output. This method is best used when one only need to merge on one variable from another data set. In our case, where product hierarchy data set contains four different attributes to be added to a master file, it would be required to create four CNTLIN data sets and four PROC FORMATS before executing four PUT functions.

THREADED SORT

Use of threaded sort for sorting data allows much faster processing and can significantly decrease the processing time. Specifying TAGSORT option can yield significant improvements in clock time, while typically increasing the CPU time. It causes the sort routine to store only the sort-keys and observation numbers in the SAS® temporary files. These keys and record numbers are the 'tags' of TAGSORT. Then, at the end of the sort, the tags are used to retrieve entire records from the original data set in sorted order. This has potential gains whenever the total length of the sort keys is small compared to the record length. Temporary disk usage is significantly decreased.

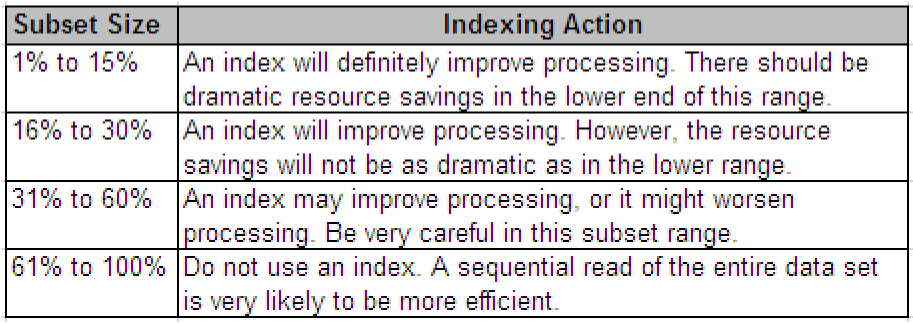
INTELLIGENT USE OF INDEXES

Large SAS indexes, especially those with small index page sizes, tend to have more index levels. The greater the number of levels an index has, the more I/Os are consumed during an index search and the longer it takes to complete the search. Conversely, indexes with fewer levels require fewer I/Os to traverse the index during an index search. Therefore while merging two datasets one big and small, it is always suggested to index smaller dataset while merging. This can improve processing time significantly.

Also, an index is most effective when it is used to access a small subset of observations from a large SAS data set. When this happens, the overhead of processing index pages and data set pages is lower than the overhead of reading the entire data set. As the size of the subset increases, the efficiency of using the index to read the data decreases.

Fig 11.

Indexing guidelines



Subset Size	Indexing Action
1% to 15%	An index will definitely improve processing. There should be dramatic resource savings in the lower end of this range.
16% to 30%	An index will improve processing. However, the resource savings will not be as dramatic as in the lower range.
31% to 60%	An index may improve processing, or it might worsen processing. Be very careful in this subset range.
61% to 100%	Do not use an index. A sequential read of the entire data set is very likely to be more efficient.

Using abovementioned guidelines for indexing, significant improvement in processing time was seen in the steps which used indexes for merging.

MODULARIZING COMPLEX STEPS

It is required to separate the independent steps and avoid writing huge lines of code for performing similar tasks. Creating macros for performing repeated tasks simplifies the process and makes the code easy to read. Moreover, it also reduces code redundancy and in case of a change to process, allows users to update code in only one location instead of many.

REDUCING CODE REDUNDANCY

Removing unwanted code; use of functions and macros for performing repetitive tasks makes the code much simpler and easier to read. Also any processes, differing in only a certain parameter or a set of parameters but having all the other processing logic similar, need to be converted to a macro. This would make the coded simpler and more readable.

SYNTACTIC OPTIMIZATION

Various forms of syntactic optimizations can be made to the code which can result in faster processing. Consider some of the examples for the same as follows:

USE WHERE INSTEAD OF IF

Use of where statement instead of an IF statement is more efficient while sub setting the data, since a sub setting IF statement requires all the observations to read from the dataset first, whereas the WHERE statement only preselects observations which qualify sub setting condition. Moreover a WHERE statement can be used in the DATA as well as PROC step, whereas an IF statement can only be used in a data step.

USE OF COMPRESS OPTION

System data step compress option or system compress option can be used to reduce the size of a dataset. Depending upon whether the dataset contains variables of numeric data type or character data type, appropriate compressing method needs to be selected. The compression should be changed to BINARY for efficiently compressing data sets having numeric variables data type. Compressing datasets typically save two times the amount of space that a data set might normally occupy

```
        OPTIONS COMPRESS=YES
```

DELETION OF LARGE INTERMEDIATE DATA SETS

Deleting large volume intermediate datasets after processing is complete, frees up disk space for other SAS processes executing in parallel thereby allowing faster processing of parallel jobs. Also, it is suggested to delete an existing dataset first, before overwriting the same. Whenever a SAS data set is updated or replaced with another SAS data set, SAS creates a temporary SAS data set to hold the new data until the DATA step or procedure successfully completes, thereby utilizing more disk space in the process.

CONCLUSIONS

In this section, I have summarized the findings from all the aforementioned techniques for process improvement.

Identifying areas of improvement gives the developer a precise idea of steps that are required to be worked upon. 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.

Actual parallelization technique can be applied to only certain 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

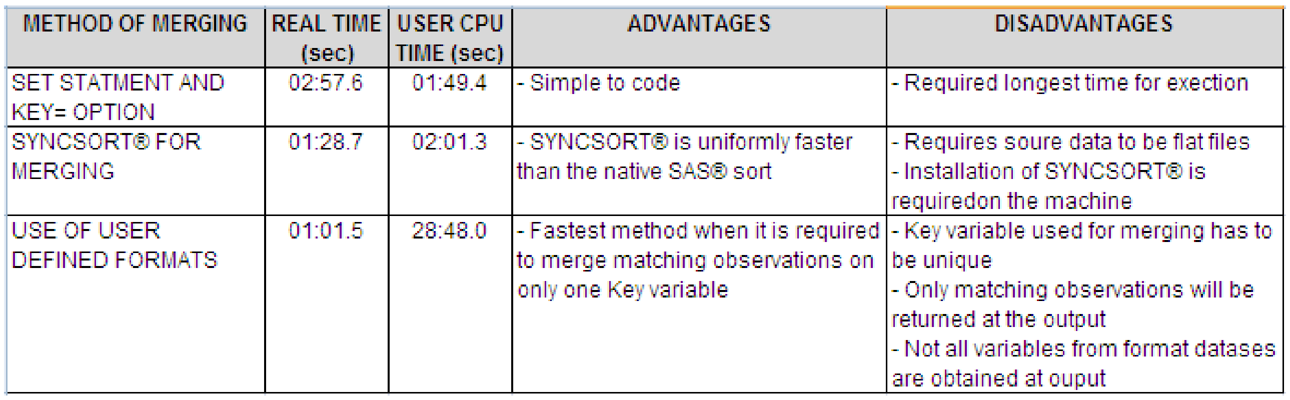
Virtual Multithreading is beneficial only in those situations where CPU time and the real time of a process are not far apart. It has been observed that for a system-intensive process, the CPU time and the Real time are not far apart, the ratio of CPU time and real time would be 1. Multi-threading is 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.

For, execution of multiple instances of jobs in UNIX, it is required that the nature of the data being processed should be such that one should be able to split it in order to process in parallel.

Results of the case study of the use of alternative methods for merging two datasets, one large and one small are listed in the table below, along with the advantages and the disadvantages of each method:

Fig 12.

Comparison of various methods of merging



METHOD OF MERGING	REAL TIME (sec)	USER CPU TIME (sec)	ADVANTAGES	DISADVANTAGES
SET STATMENT AND KEY= OPTION	02:57.6	01:49.4	- Simple to code	- Required longest time for exection
SYNCSORT® FOR MERGING	01:28.7	02:01.3	- SYNCSORT® is uniformly faster than the native SAS® sort	- Requires soure data to be flat files - Installation of SYNCSORT® is requiredon the machine
USE OF USER DEFINED FORMATS	01:01.5	28:48.0	- Fastest method when it is required to merge matching observations on only one Key variable	- Key variable used for merging has to be unique - Only matching observations will be returned at the output - Not all variables from format datases are obtained at ouput

Use of where statement instead of an IF statement is more efficient while sub setting the data

For use of indexes while merging two datasets one big and small, it is always suggested to index smaller data set

Deletion of large intermediate datasets: It is suggested to delete an existing data set first, before overwriting the same in order to free up any occupied disk space.

Using the above mentioned process improvement techniques; the time of a practical production process was reduced from 105 hours to 22 hours, thereby resulting in up to 74% process improvement. Above mentioned generic techniques can be applied to any system intensive processes.

REFERENCES:

Kosian, Sassoon. A Generic Method of Parallel Processing in Base SAS @ 8 and 9. SUGI 2007.

Ghent, Ian J. Faster results by multi-threading data steps. SUGI 2010.

Schuster, Kristie and Sipe, Lori. 50 WAYS TO MERGE YOUR DATA – INSTALLMENT 1… SUGI 26

Cassell, David L. A sort of a Mess - Sorting Large Datasets on Multiple Keys. SUGI 26

Raithel, Michael A. The Complete Guide to SAS® Indexes (Pg. 18,19,20,36 and 38 from Chapter 2 and table 2.1) Copyright 2006, SAS Institute Inc., Cary, NC, USA. All Rights Reserved. Reproduced with permission of SAS Institute Inc., Cary, NC

https://support.sas.com for details on the FULLSTIMER SAS option details. Same can be referenced at the following link http://support.sas.com/rnd/scalability/tools/fullstim/index.html

ACKNOWLEDGMENTS

I would like to profusely thank my clients James Maciaszek and BrahmanandaRao Nukala for being wonderful guides and mentors during my work at Novartis Pharmaceuticals Company. I would also like to thank my Cognizant managers Dhananjay Kelkar, Prakash Pothamsetti and Jayesh Soneji for encouraging my efforts to document my research and present a paper. I would also thank my company Cognizant Technology Solutions for providing me with an opportunity for attending the SAS SUGI 2009 which is when it dawned upon me to write a paper of my own. I would also like to mention the following authors: Sassoon Kosian, Ian J. Ghent, Lori Sipe, David L.Cassell and Michael A. Raithel who granted me their kind permissions to reference their work in my paper. Finally, I would also like to thank Christianna Williams, Howard Schreier and Neeraj Kumbhakarna for their invaluable review comments and tremendous patience with my work.

CONTACT INFORMATION

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

Viraj Kumbhakarna
Cognizant Technology Solutions Corporation
1 Health Plaza
East Hanover, NJ, 07936
973 220 2246
vkumbhakarna@gmail.com

****