A Benchmarking Technique to Identify Best Practices in SAS ®

Tony Pisegna, ICON Clinical Research, North Wales, PA

ABSTRACT

The term benchmarking is used widely across many industries and at a generic level it refers to the act of comparing a process or product against an accepted standard (a benchmark) with the intent of evaluating differences in measured outcomes in order to make informed decisions. In the context of SAS, the ideology of benchmarking can be used to compare procedures, functions and general SAS code on several measures of efficiency in order to identify and/or verify best practices. In this paper, an overview of the determinants of SAS program efficiency is provided and the design considerations of a benchmarking technique using SAS code are discussed. The steps and SAS code used in this technique are illustrated within the context of comparing several efficiency-improving SAS coding principles.

INTRODUCTION

The title of this paper refers to identifying 'best practices in SAS'. This phrase is subject to broad interpretation, for example, it can refer to cosmetic practices, such as indentation and spacing, to defensive-coding practices, such as coding in anticipation of future changes to data, or to procedural practices, such as clearing libraries between programs. For purposes of this paper, 'best practices' are ways of structuring and writing code that result in greater efficiency. Terms such as 'greater efficiency' or 'improved performance' can also be described in a number of ways, however this paper focuses on speed of processing.

Most experienced SAS users have learned efficiency-improving techniques from colleagues or from the vast array of documentation on the topic, but only the curious wonder how it was determined that a technique improves efficiency or ponder how much improvement occurs. In order to make these types of determinations, a comparison(s) must be made and outcomes must be measured, recorded and evaluated. This is the essence of the concept of benchmarking, as well as experimental design.

This paper introduces a benchmarking technique that compares SAS code, measures several indicators of efficiency and summarizes the results. The core of the technique is a method for extracting performance-related information from a program log into a data set so that it can be analyzed. Design considerations will be reviewed, the benchmarking technique will be described in detail, results will be displayed and conclusions will be presented.

Note that the benchmarking technique was developed and tested using SAS Version 9.1.3 on a multiple-user UNIX server. However, the technique can be adapted for use on various platforms and it will be noted where differences between platforms need to be considered.

DESIGN CONSIDERATIONS

The benchmarking technique is designed to address the question "Does coding method A result in faster processing than coding method B?" Fundamentally, the technique is a way for conducting an experiment, and as such, several elements have to be considered in the design. The primary considerations are to identify the variable(s) that will represent processing speed (outcome variables) and to identify and understand the factors that impact processing speed (independent variables).

OUTCOME VARIABLES

The outcome variable of primary interest is speed of processing. This is commonly defined as the run time (also referred to as real time or clock time), which is the amount of time between submission of code and the end of processing. However, speed of processing can also be defined by specific computer processes, such as total CPU time, amount of memory or the number of input/output processes needed to execute the code.

By default the SAS System is set so that basic information about processing is provided in the program log for each DATA step and procedure. For example, on a UNIX platform real time and CPU time are written to the program log. When the FULLSTIMER option is activated, more-detailed information about processing is written to the log. The following is an illustration of the information written to the log after execution of a DATA step when the FULLSTIMER option is activated on a UNIX platform (the amount of information provided differs between operating systems):

```
NOTE: There were 500000 observations read from the data set DAT._500000.
NOTE: The data set WORK.TEST has 500000 observations and 61 variables.
NOTE: DATA statement used (Total process time):
      real time           1:54.21
      user cpu time       1.88 seconds
      system cpu time     5.16 seconds
      Memory                           406k
      Page Faults                      14173
      Page Reclaims                    0
      Page Swaps                       0
      Voluntary Context Switches       21282
      Involuntary Context Switches     5089
      Block Input Operations           14173
      Block Output Operations          0
```

Each type of information can be useful for understanding and debugging performance and the benchmarking technique is capable of capturing and reporting all of these measures. However, for illustration purposes the outcomes chosen to represent speed of processing are real time and measures specific to computer processes, such as total CPU time (user CPU time + system CPU time), memory and total number of input/output processes (block input operations + block output operations). Table 1 gives a brief description of each outcome variable.

TABLE 1

Result (unit)	Brief description
Real Time (seconds)	The clock time between submission of code and the end of execution.
Total CPU Time (seconds)	Time spent by the central processing unit to execute code.
Memory (K)	The amount of memory allocated to execute code.
Total I/O Operations(n)	The I/O operations to read the data into memory and write out to files.

INDEPENDENT VARIABLES

There are many factors that independently impact processing speed. For purposes of discussion, they can be divided into categories, computer-related, data-related, user/task-related and situation-related.

- Computer-related
  - Speed/capacity of the central processing unit
  - System speed (how fast files can be moved)
  - Amount of memory
  - The SAS configuration (e.g., SAS version, administrative options and default settings)

- Data-related
  - Number of observations being processed
  - Number of variables in the data set(s) being processed
  - Length/storage requirements of variables (e.g., char vs. num)

- User/task-related
  - The type and number of SAS processes required (e.g., 2 sorts and 10 DATA steps vs. 10 sorts and 2 DATA steps).
  - Coding practices (e.g., processing only the required variables).
  - SAS options used (e.g., noprint, nosymbolgen, etc.)

- Situation-related
  - The number and type of processes/programs competing for CPU resources.

Recall that the primary research question is 'Does coding method A result in faster processing than coding method B?'. In order to allow for meaningful comparisons, the only factors that should be allowed to vary during the comparisons are coding practices (user/task-related factors), specifically the differences between method A

and B, while all other independent variables are held constant (to the greatest degree possible).

The computer-related factors can be held constant by conducting all comparisons on the same computer and/or in the same computing environment. The data-related factors can be controlled by using the exact same data when processing methods A and B. The situation-related variables are harder to control in real-world settings. For example, if a large job is competing for CPU time while processing Method A but not while processing Method B, then the results would be confounded by the situation and would not lead to meaningful conclusions. One way to work around this challenge is to conduct multiple comparisons and then evaluate the mean value of the outcome variables. This assumes that the multiple observations will represent a sampling of situations, such as peak versus non-peak usage, and will help to minimize the unpredictable effect of competing processes.

THE BENCHMARKING TECHNIQUE

The benchmarking technique is separated into 4 phases, Primary Setup, Multiple Trials, Data Preparation and Data Summarization.

PRIMARY SETUP PHASE

The steps involved in the Primary Setup Phase are:

1. Select a SAS programming environment that can be available at several time points. The ideal environment is one in which the influence of independent variables can be minimized. For example, if available, a PC used by one person would be preferable to a network with many users.

2. Choose or create one data set to be used for all comparisons. The data set should be similar to those used in real-world settings so that conclusions can be generalized and large enough (e.g., number of observation and variables) so that differences, if present, are detectable.

3. Create a SAS program that represents Method A (method_a.sas) and a program that represents Method B (method_b.sas). The only difference between the programs should be the coding techniques that are being compared. Within each program, activate the FULLSTIMER option and direct the log output to a permanent file. In addition, the NOTES/NONOTES option should be switched on and off so that the permanent log file contains the performance information only for the executions of interest (e.g., not interested in performance information from the PRINTTO procedure or other steps).

4. Create a shell script (runtrials.sh) that executes method_a.sas and method_b.sas 20 times in an alternating sequential fashion (method_a, method_b, method_a, method_b…). The intent of the sequential alternation is to minimize situation-related differences. Within method_a.sas and method_b.sas, the NEW option is not used on the PRINTTO procedure statement so that the permanent log file is appended with each additional execution. Note that the creation of the program to execute method_a.sas and method_b.sas may be dependent upon the operating system.

The following are code examples from a comparison between a method believed to be more efficient (method_a.sas – keep only the observations and variables needed for processing) and a method believed to be less efficient (method_b.sas – keep all observations and variables). The differences are shown in bold italics.

METHOD A

```
OPTIONS nonotes nofmterr fullstimer msglevel=n;

******** define library containing test data ;
LIBNAME dat "/public/benchmark/data sets";

******** reroute the log output ;
PROC PRINTTO LOG = "/public/benchmark/logs/method_a.log" ;
RUN;

******** execute test code ;
OPTIONS notes  ;

DATA test;
   LENGTH newvar $200.;
   SET dat._1000000 (where = (teevent = 1)
                     keep = teevent event prefterm bodcode strdt stopdt usubjid);
   newvar = trim(event)||" / "||trim(prefterm)||" / "||trim(bodcode);
   if nmiss(strdt, stopdt) = 0  then duration = stopdt - strdt + 1;
RUN;
```

METHOD B

```
OPTIONS nonotes nofmterr fullstimer msglevel=n;

******** define library containing test data ;
LIBNAME dat "/public/benchmark/data sets";

******** reroute the log output ;
PROC PRINTTO LOG = "/public/benchmark/logs/method_b.log" ;
run;

******** execute test code ;
OPTIONS notes  ;

DATA test;
   LENGTH newvar $200.;
   SET dat._1000000;
   newvar = trim(event)||" / "||trim(prefterm)||" / "||trim(bodcode);
   if nmiss(strdt, stopdt) = 0  then duration = stopdt - strdt + 1;
RUN;
```

MULTIPLE TRIALS PHASE

During the Multiple Trials Phase the following steps occur:

1. Execute runtrials.sas. After running the program once, two permanent log files will have been produced, one for Method A and one for Method B. Each log file will have 20 sets of performance outcome measures.

2. Execute runtrials.sas as many times as deemed necessary to collect adequate data. As the number of trials increases the effect of unpredictable situation-related variables decreases. The effects can be further minimized by executing runtrials.sas at different times of day on different days.

DATA PREPARATION PHASE

Before being able to analyze the data, it must be extracted from the permanent log files and put into data sets. The SAS System writes the performance information to the log file in a predictable format. That is, the information is written using the same text in the same order and unit of measure for every DATA step and procedure. Accordingly, the performance results can be programmatically extracted from the log files. The program (getresults.sas) performs the extraction and prepares a permanent data set by executing the following steps. Note that the LOGPARSE macro suite (available for download from http://support.sas.com/kb/34/301.html) can also be used to extract performance information from log files.

1. Read each line from the log file into a variable, compress the value and change the value to upper case.

2. Search each value (observation) for key words which indicate that a result needed for the outcome variables is present. Key words are USERCPU, SYSTEMCPU, REAL, MEMORY, BLOCKINPUT, BLOCKOUTPUT.

3. Strip unnecessary text from the string so that only the key word and the result are kept.

4. Convert results captured as HH:MM:SS:SS to seconds.

5. Assign a number to each trial.

6. Derive total CPU as USERCPU plus SYSTEMCPU and Total Input/Output Operations as BLOCKINPUT plus BLOCKOUTPUT.

7. Output a permanent data set for Method A and one for Method B.

The following is an example of getresults.sas for a comparison where Method A and B includes a single DATA step:

```
   ******** initialize macro to find text strings in log files and
            create a variable where the value represents the output
            parameter ;

   %MACRO findem (term);
      if INDEX(UPCASE(text), "&term") then param = "&term";
   %MEND findem;

   ******** initialize macro that
            retrieves the log file
            extracts the results for each outcome parameter
            assigns a trial number to each result
            outputs a permanent data set ;
```

Continued

```
%MACRO getresults (logname = );

   FILENAME getlog "/public/benchmark/logs/&logname..log";

   DATA getem (KEEP = param result trial method);
      INFILE getlog dlm='~' lrecl=2000;
      LENGTH  text text2 $2000. param method $60. sec2 $8.;
      INPUT text;  *** call macro to search for keywords;
      %findem(USERCPU);
      %findem(SYSTEMCPU);
      %findem(REAL);
      %findem(MEMORY);
      %findem(FAULTS);
      %findem(BLOCKINPUT);
      %findem(BLOCKOUTPUT);

    *** keep only obs that contain result data ;
       if param = '' then delete;

    *** strip unneeded text from string containing result ;
       text2 = COMPRESS(text, , "a");

    *** convert all times to seconds ;
       if param in ("REAL", "USERCPU", "SYSTEMCPU") then do;
         if index(text2, ":") then do;

           if length(text2) gt 8 then do;
             hour = input(scan(text2, 1, ":"), best.);
             min  = input(scan(text2, 2, ":"), best.);
             sec2 =       scan(text2, 3, ":");
             sec  = input(scan(sec2,  1, "."), best.);
             text2 = compress(put(
                (hour*3600) + (min*60) + sec, best.)||"."||scan(sec2, 2, "."));
           end;
           else if length(text2) le 8 then do;
             min  = input(scan(text2, 1, ":"), best.);
             sec2 =       scan(text2, 2, ":");
             sec  = input(scan(sec2,  1, "."), best.);
             text2 = compress(put((min*60) + sec, best.)||"."||scan(sec2, 2, "."));
           end;
         end;
       end;

    *** create trial number variable;
       if param = 'REAL' then trial + 1;

   result = INPUT(COMPRESS(text2), best.);
   method = "&logname";

   RUN;
```

Continued

```
******* create total CPU time and total I/O;

PROC TRANSPOSE
   DATA = getem
   OUT  = transem;
   ID param;
   VAR result;
   BY method trial;
RUN;

DATA makem;
   SET transem;
   TOT_CPU = usercpu + systemcpu;
   TOT_IO = blockinput + blockoutput;
RUN;

PROC TRANSPOSE
   DATA = makem
   OUT  =  out.&logname (RENAME = (_name_ = param)
                         KEEP = method result trial _name_);
   BY method trial;
RUN;

%mend getresults;

%getresults (logname = method_a);
%getresults (logname = method_b);
```

DATA SUMMARIZATION PHASE

The Data Summarization Phase consists of the execution of a program (dostats.sas), which calculates descriptive statistics using the MEANS procedure and creates a bar chart for each outcome measure by Method.

RESULTS

The results shown below are from comparisons that executed the benchmarking technique on SAS Version 9.1.3 on a UNIX platform within a multi-user network environment. The size of the data set used for all comparisons was 1136 megabytes and consisted of 1 million observations and 60 variables. The number of trials conducted for each comparison was 100 (executed as 5 runs at different time points at 20 trials each).

Comparison 1

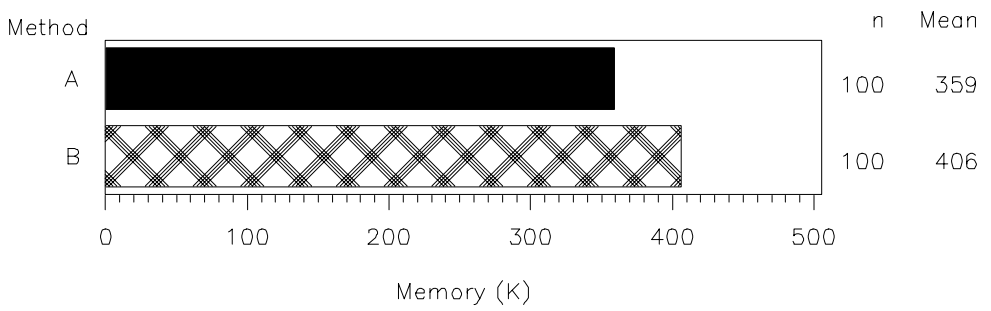
Method A = keep only the observations (using a WHERE clause on the SET statement) and variables (using a KEEP statement on the SET statement) needed for processing (872630 observations and 7 variables).

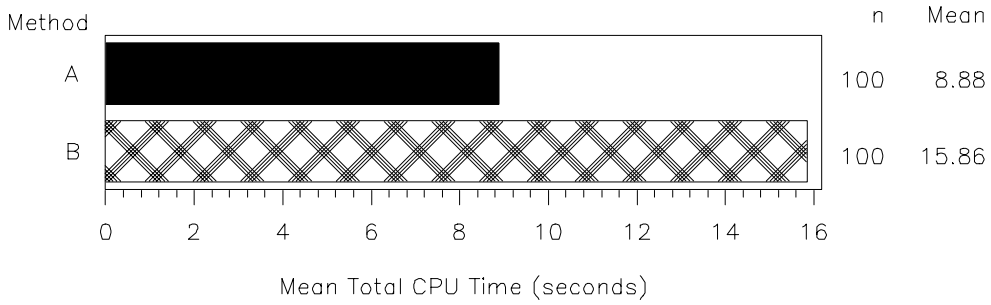
Method B = keep all observations and variables (1M observations and 60 variables).

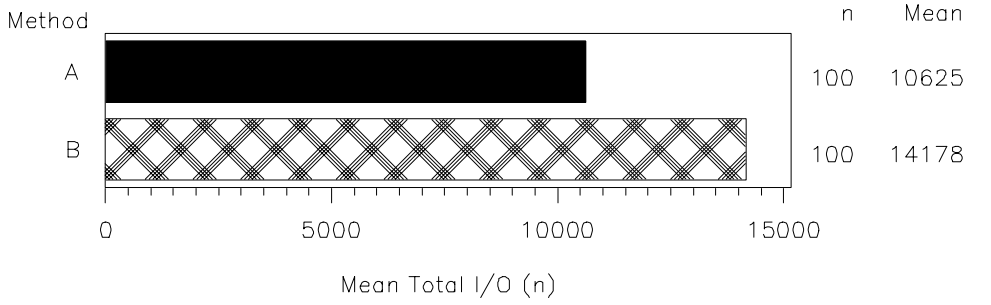
Comparison 1 Descriptive Statistics

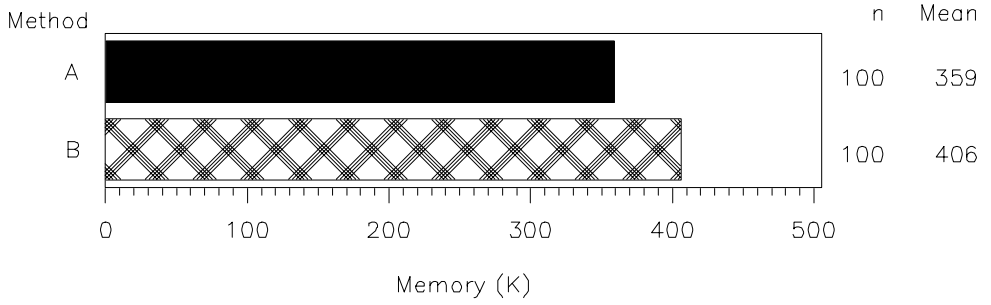
Parameter	Statistic	Method A	Method B
Real Time (seconds)	n	100	100
	mean	525.61	723.95
	median	555.41	816.45
	sd	110.883	191.803
	min	237.6	237.6
	max	702.0	851.7
Total CPU Time (seconds)	n	100	100
	mean	8.88	15.86
	median	8.24	15.73
	sd	1.796	1.288
	min	6.8	14.4
	max	15.4	25.8
Total I/O Operations	n	100	100
	mean	10624.64	14177.55
	median	3833.00	8566.00
	sd	18631.118	15275.061
	min	616.0	5007.0
	max	75781.0	76909.0
Memory (K)	n	100	100
	mean	359.00	406.00
	median	359.00	406.00
	sd	0.000	0.000
	min	359.0	406.0
	max	359.0	406.0

Comparison 1 Bar Charts









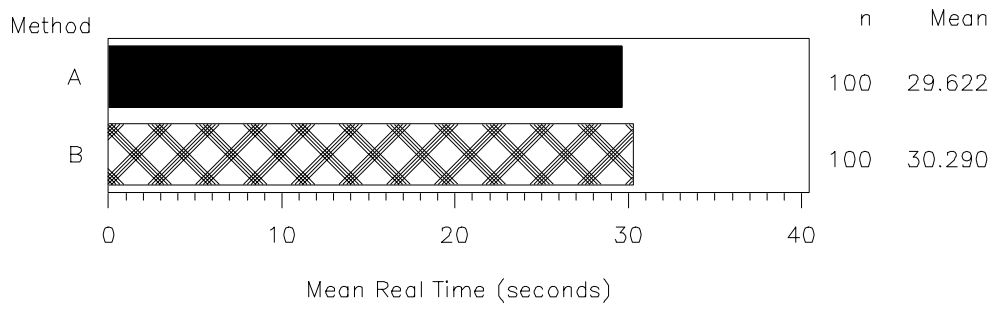
Comparison 2

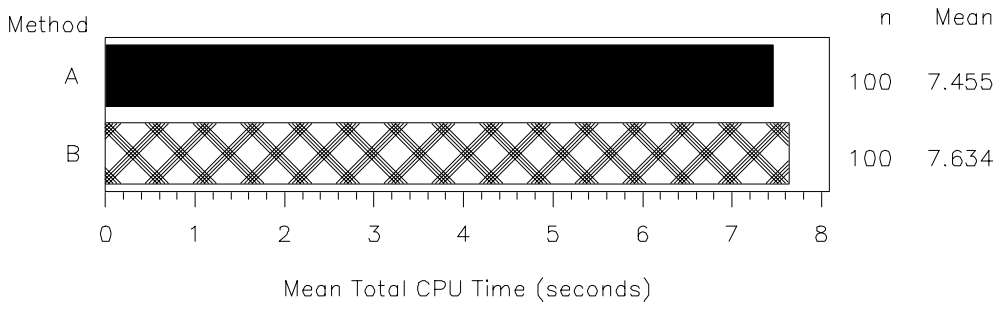
Method A = process 12 conditions using “if..then..else” processing, i.e., if x then y; else if q then r;…
Method B = process 12 conditions using “if…then” processing, i.e.., if x then y; if q then r;…

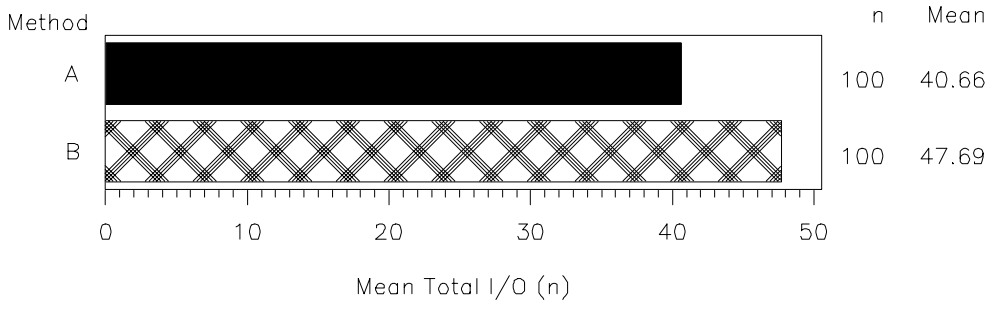
Comparison 2 Descriptive Statistics

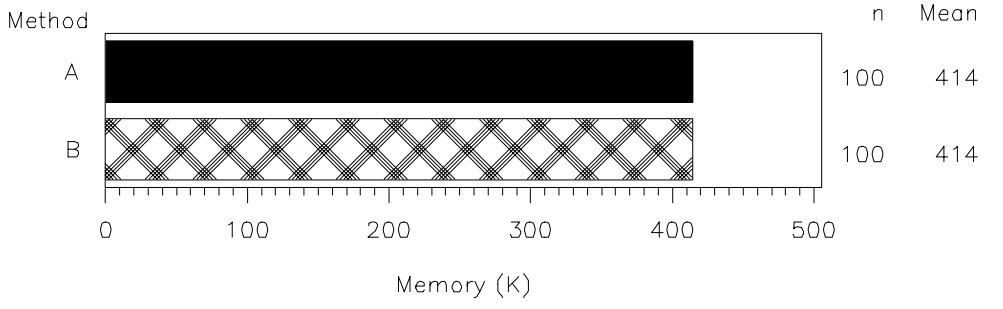
Parameter	Statistic	Method A	Method B
Real Time (seconds)	n	100	100
	mean	29.62	30.29
	median	29.68	29.91
	sd	9.435	10.416
	min	14.1	14.7
	max	79.0	74.1
Total CPU Time (seconds)	n	100	100
	mean	7.46	7.63
	median	7.40	7.51
	sd	0.396	0.429
	min	6.7	6.7
	max	8.5	9.3
Total I/O Operations	n	100	100
	mean	40.66	47.69
	median	7.00	8.00
	sd	123.203	162.148
	min	3.0	3.0
	max	909.0	1121.0
Memory (K)	n	100	100
	mean	414.00	414.00
	median	414.00	414.00
	sd	0.000	0.000
	min	414.0	414.0
	max	414.0	414.0

Comparison 2 Bar Charts









The benchmarking technique can also be adapted to compare methods that use multiple DATA steps and/or procedures. The results for Comparison 3 show differences between methods where Method A consisted of one procedure (the UNIVARIATE procedure) and Method B consisted of two procedures, a SORT and a UNIVARIATE. In order to make this comparison, the getresults.sas program is modified so that performance outcomes from 2 procedures in Method B can be combined.

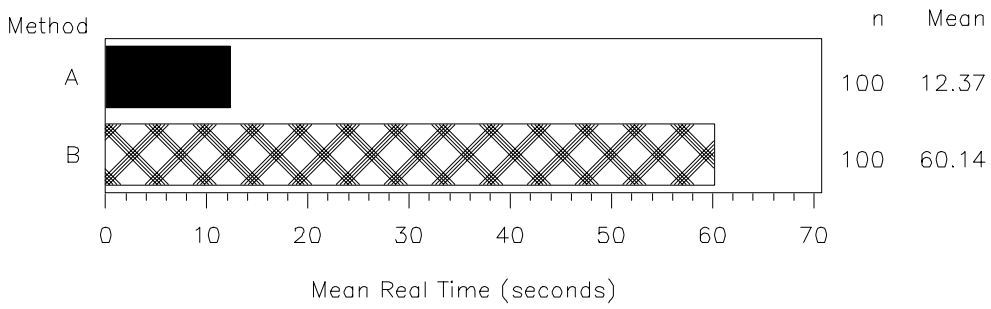
Comparison 3

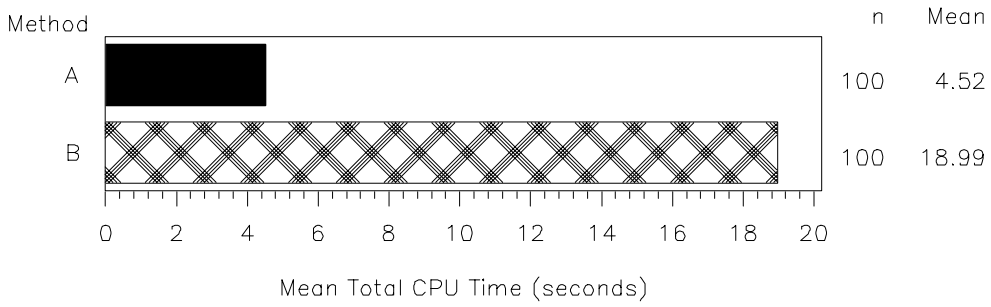
Method A = use a CLASS statement on procedures so that a pre-sort is not needed.
Method B = sort data set prior to procedure then execute procedure using BY statement.

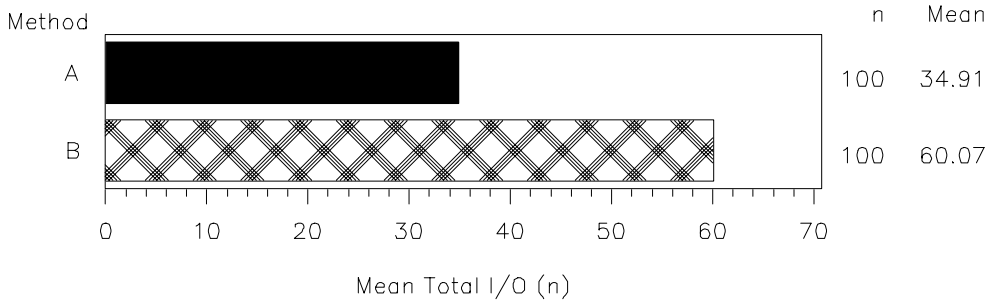
Comparison 3 Descriptive Statistics

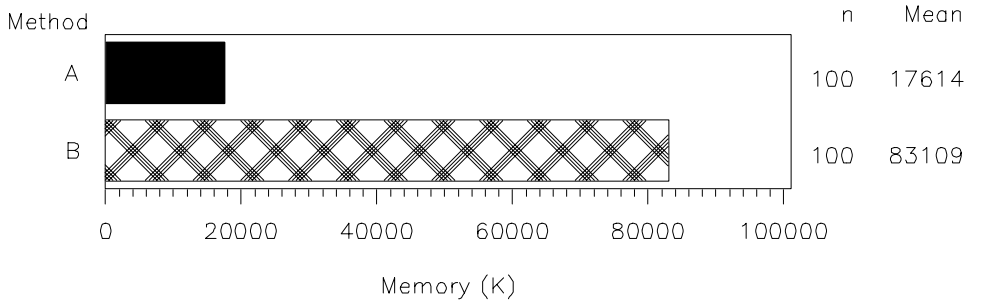
Parameter	Statistic	Method A	Method B
Real Time (seconds)	n	100	100
	mean	12.37	60.14
	median	12.09	60.74
	sd	1.099	9.833
	min	10.9	39.0
	max	17.7	80.7
Total CPU Time (seconds)	n	100	100
	mean	4.52	18.99
	median	4.48	19.03
	sd	0.374	1.187
	min	3.9	16.9
	max	5.7	22.2
Total I/O Operations	n	100	100
	mean	34.91	60.07
	median	20.50	56.50
	sd	35.509	29.952
	min	7.0	7.0
	max	243.0	223.0
Memory (K)	n	100	100
	mean	17614.00	83109.00
	median	17614.00	83109.00
	sd	0.000	0.000
	min	17614.0	83109.0
	max	17614.0	83109.0

Comparison 3 Bar Charts









CONCLUSIONS

The benchmarking technique described in this paper is a tool that can be used to analyze results from comparisons between one coding method and another. As demonstrated, the evaluation can determine or verify which of the two methods result in faster processing time, which can aid in determination of 'best practices' in SAS.

Beyond being a tool with specific purpose, the underlying concepts of experimental design allow for the technique to be modified and/or expanded upon in order to examine various comparisons and address a variety of questions.

- The definition of the outcome variable(s) can be changed. For example, rather than focusing on processing time, the emphasis could be a more in depth examination of the differences in I/O functions.

- Formal inferential statistics can be introduced into the design so that differences in outcomes can be evaluated for statistical significance.

- System administrators can adapt and use the technique to better understand and/or debug system problems by quantifying differences on outcomes. For example, they might examine the differences between Real Time and Total CPU Time across different peak periods of usage with the goal of fine tuning system performance.

- Examples of questions that can be evaluated are:

    - Does computer hardware A perform faster than computer hardware B?
    - How much does additional memory impact processing speed?
    - Does the degree of improvement in processing speed between Method A and Method B change as the size of data set changes? If so, how much?
    - How can SAS options affect processing speed?
    - To what degree is processing speed compromised as the number of competing processes increases?

The range of the benchmarking technique seems limited only by the imagination. The author invites and encourages the reader to utilize and expand upon the benchmarking technique to conduct tests on other methods and/or to invent and test new methods.

It's said that knowledge is power and the more a programmer knows about the factors that impact processing, the better he/she will be able to masterfully write efficient program code. The benchmarking technique described in this paper is a tool that can be used by programmers to gain this type of knowledge.

REFERENCES:

SAS Institute Inc., *SAS® Programming Tips: A Guide to Efficient SAS® Processing*, Cary, NC: SAS Institute Inc., 1990, 155 pp.

ACKNOWLEDGMENTS

CONTACT INFORMATION

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

Tony Pisegna
ICON Clinical Research
1700 Pennbrook Parkway
North Wales, PA 19454
Email: tony.pisegna@iconplc.com