

The How and When of Efficient Programming

The How and When of Efficient Programming

Frank DiIorio
AIMS, Co.
Durham, North Carolina

Know What to Tune, and When

- ❄ Identify the constraints
- ❄ Implement in phases
- ❄ Test under realistic conditions

Recognize the Need for Program Tuning

- ❄ "But my program/data set is small."
- ❄ "I only get a 2% improvement when I use [technique x]."
- ❄ "But our server's huge. Everything runs fast."
- ❄ "This is a batch program that runs in the wee hours. Efficiency doesn't matter."

Reduce I/O!

- ❄ Reduce data set size as early in the program as possible. [Example 1](#)
- ❄ Use system options to control data set size and location.

The How and When of Efficient Programming

Reduce I/O! (cont.)

- ❄ Avoid sorting
 - Arrange steps to minimize sorts
 - Take advantage of grouping statements
 - Use the NOEQUALS option in SORT
 - Remember that SAS knows data order
 - Use Indexes (speed v. disk requirements)
 - Data Set Structure (grouped data, numbers as characters)

Tuning the DATA Step

- ❄ Know when to avoid a DATA step
- ❄ If you can't avoid it, you might want to ...
 - Use _NULL_ DATA steps when possible
 - Use temporary array elements
 - Combine data using MERGE alternatives
 - Use functions
 - Avoid list-style input directives
 - Minimize the number of statements executed

Identify and Use "Non-Obvious" Tuning Tools

- ❄ Take advantage of "passive"/background tuning
- ❄ Subset on the fly with data set options
- ❄ Assess the impact of the efficiency measures
 - System options
 - SAS Log
 - CONTENTS, DATASETS, dictionary views and tables

Tuning the DATA Step (cont.)

- ❄ Tuning techniques (cont.)
 - Reduce observation length
 - Store short codes
 - Use views
 - Always specify character variable lengths
 - Avoid consecutive, related IF statements
 - Use WHERE to filter observations

The How and When of Efficient Programming

Use Executable Code and Permanent Data Sets

- ❄ Use executable code
 - Use format libraries
 - Use compiled macros and the Stored Program Facility
- ❄ Use permanent data sets



Play to Procedures' Strengths

- ❄ Recognize procedure overlap
- ❄ Let procedures be the workhorses 
- ❄ Keep up with procedure enhancements



Go Outside the SAS "Box"

- ❄ When?
- ❄ During program development
- ❄ To create data 



Remember: PROCs are "Tuneable"

- ❄ Multiple outputs, actions per invocation 
- ❄ CLASS var(s); or BY var(s); ?



The How and When of Efficient Programming

Use Macros Effectively

- ❄ Create (and use) a library of utility macros
- ❄ Avoid nesting
- ❄ Manage memory reserved for macros
- ❄ Phase in compiled macros during development

End

Reduce I/O (Example 2)

Excessive Sorting

```
proc sort data=pass1;
  by dept;
run;

proc print data=pass1;
  by dept;
run;

proc sort data=pass1;
  by area;
run;

proc print data=pass1;
  by area;
run;

proc sort data=pass1;
  by dept;
run;

proc means data=pass1;
  by dept;
  var salchg;
run;
```

Minimal Sorting

```
proc sort data=pass1;
  by dept;
run;

proc print data=pass1;
  by dept;
run;

proc means data=pass1;
  by dept;
  var salchg;
run;

proc sort data=pass1;
  by area;
run;

proc print data=pass1;
  by area;
run;
```

Reduce I/O (Example 1)

Excessive I/O

```
data temp;
  set perm.master;
  other DATA step statements
run;
```

```
proc print data=temp;
  var variable list;
run;
```

```
proc report data=temp;
  REPORT procedure statements;
run;
```

Reduced I/O

```
data temp;
  set perm.master;
  other DATA step statements
  keep variables used in PRINT, REPORT, and other procedures;
run;
```

```
proc print data=temp;
  var variable list;
run;
```

```
proc report data=temp;
  REPORT procedure statements;
run;
```

Reduce I/O (Example 3)

Unnecessary Sort

```
proc sort data=large;
  by treatment;
run;
```

```
proc means data=large;
  class treatment;
  <other MEANS statements>
```

Effective Use of CLASS

```
proc means data=large;
  class treatment;
  <other MEANS statements>
```

Execute Without CLASS

```
proc sort data=large;
  by treatment;
run;
```

```
proc means data=large;
  by treatment;
  <other MEANS statements>
```

The How and When of Efficient Programming

Use Options for Tuning

```
Excessive I/O
data temp;
set perm.master;
if compound in ('DHK21', 'DHB23a') then do;
    _counter_ + 1;
    if _counter = 101 then stop;
end;
run;

proc print data=temp;
var compound age gender race rest_pulse bp_d bp_s;
run;

Reduced I/O
proc print data=perm.master
(wher= compound in ('DHK21', 'DHB23a')
keep=compound age gender race rest_pulse bp_d
bp_s
obs=100);
run;
```



Tuning the DATA Step (Example 2)

```
Acceptable
data demo;
infile '\\srv101\prod\oct2000.dat';
input id $4. loc $2. (m1-m100)(4.);
... other DATA step statements ...
if loc in ('P1', 'P3') then output;
run;

Alternative
data demo;
infile '\\srv101\prod\oct2000.dat';
input id $4. loc $2. @;
if loc in ('P1', 'P3') then do;
    input (m1-m100)(4.);
    ... other DATA step statements ...
    output;
end;
else input;
run;
```



Tuning the DATA Step (Example 1)

```
Unnecessary DATA Step
data _null_;
now = today();
call symput('asofdate', put(now, mmddyy8.));
run;

proc print data=one;
title "Data current as of &asofdate.";
run;

Alternative #1
proc print data=one;
title "Data current as of %sysfunc(today(), mmddyy8.)";
run;

Alternative #2
%let asofdate = %sysfunc(today(), mmddyy8.);
proc print data=one;
title "Data current as of &asofdate.";
run;
```



Tuning the DATA Step (Example 3)

```
Multiple IFs
if group = '1h' then rate = 1.0;
if group = '2a' then rate = 2.1;
if group = '1g' then rate = 1.2;

Implement with ELSE statements
if group = '1g' then rate = 1.2;
else if group = '1h' then rate = 1.0;
else if group = '2a' then rate = 2.1;

Implement with SELECT
select (group);
when ('1g') rate = 1.2;
when ('1h') rate = 1.0;
when ('2a') rate = 2.1;
otherwise;
end;
```



The How and When of Efficient Programming

Go Outside the SAS "Box"

During program development

```
grep tod *.sas
```

Creating data

Acceptable, but needless I/O

```
data _null_;
infile '$DPR0D/input/pdata000.dat' end=eof;
input;
if eof then call symput('start_at', put(_n_,7.));
run;
```

```
data last100;
infile '$DPR0D/input/pdata000.dat' firstobs=&start_at;
input <input directives go here>;
run;
```

Faster, uses OS utility

```
x 'tail -100 $DPR0D/input/pdata000.dat > $TMP/t.dat';
```

```
data last100;
infile '$TMP/t.dat';
input <input directives go here>;
run;
```



Effective PROC Usage

Ineffective

```
proc sql;
create table <etc. etc.> ;
quit;

proc sql;
create view <etc. etc.> ;
quit;
```

Better

```
proc sql;
create table <etc. etc.> ;
create view <etc. etc.> ;
quit;
```



Play to Procedures' Strengths

DATA Step Activity

Create a copy of a data set

Append data to a data set

Rename a series of data sets

Calculate univariate statistics

Compute a value's difference from the data set's or subsets mean.

Standardize a value around its mean.

Calculate rankings of a variable

Procedure Replacement

COPY

APPEND

DATASETS (AGE statement)

UNIVARIATE, MEANS, SUMMARY

SQL

STANDARD

RANK



Thanks!

Questions and comments are welcome, and should be directed to

fcd1@mindspring.com

