シミュレーション研究におけるマルチスレッド化による時間短縮テクニック

土居 主尚
独立行政法人 放射線医学総合研究所 規制科学研究プログラム

Accelerate simulation studies by parallel computing

Kazutaka Doi
Regulatory Science Research Program,
National Institute of Radiological Sciences

要旨:

近年安価なPCでも複数コアのCPUを搭載しているが，その性能を全て活かすのは一般に困難である。そこで反復計算のシミュレーション研究の状況に限定されるが，複数コアを全て用いるテクニックを紹介する。

キーワード:シミュレーション研究，高速化，マルチスレッド

発表の内容

- 背景
  - PCの性能向上とマルチスレッド
- 全体の流れと準備
- 各プログラムの説明
  - シミュレーション全体に関わる設定
  - データ発生・プログラム生成
  - 生成したプログラムの実行と同期
  - 結果のまとめと生成したプログラムの消去
  - 全体の実行
- 結果
- 結論

背景

- コンピュータの性能向上
  - 実行ユニットを複数搭載したPC（マルチコア）が身近に
  - 最低でも2コア，多いもので6コア（サーバーは除く）
  - 複数のプログラムを同時に実行可能
- マルチスレッド対応プロシジャ（SAS9より）
  - 大幅な時間短縮
  - 理想的な状況ではコア数に比例した速度
  - 対応プロシジャ
    - BASE：MEANS，REPORT，SQL，SUMMARY，TABULATE
    - STAT：GLM，LOESS，REG，ROBUSTREG

マルチスレッド化

- 現在のコンピュータ高速化の主流
  - シングルスレッド高速化の限界
  - 単一の複雑なCPUより，単純なCPU（コア）を複数搭載
  - Athlon 64 X2（2005），Intel Core（2006），PS3 Cell（2006）
  - GPGPU（general purpose computing on graphics units）
  - 今後もこの流れは継続・加速
- 複数のCPU（コア）の活用
  - 複数のプログラムを同時に実行（マルチプロセス）
  - 単一プログラム中で同時実行可能な部分を抽出し，マルチスレッドで実行
  - 前の計算結果に後の処理が依存する際，同時実行は困難

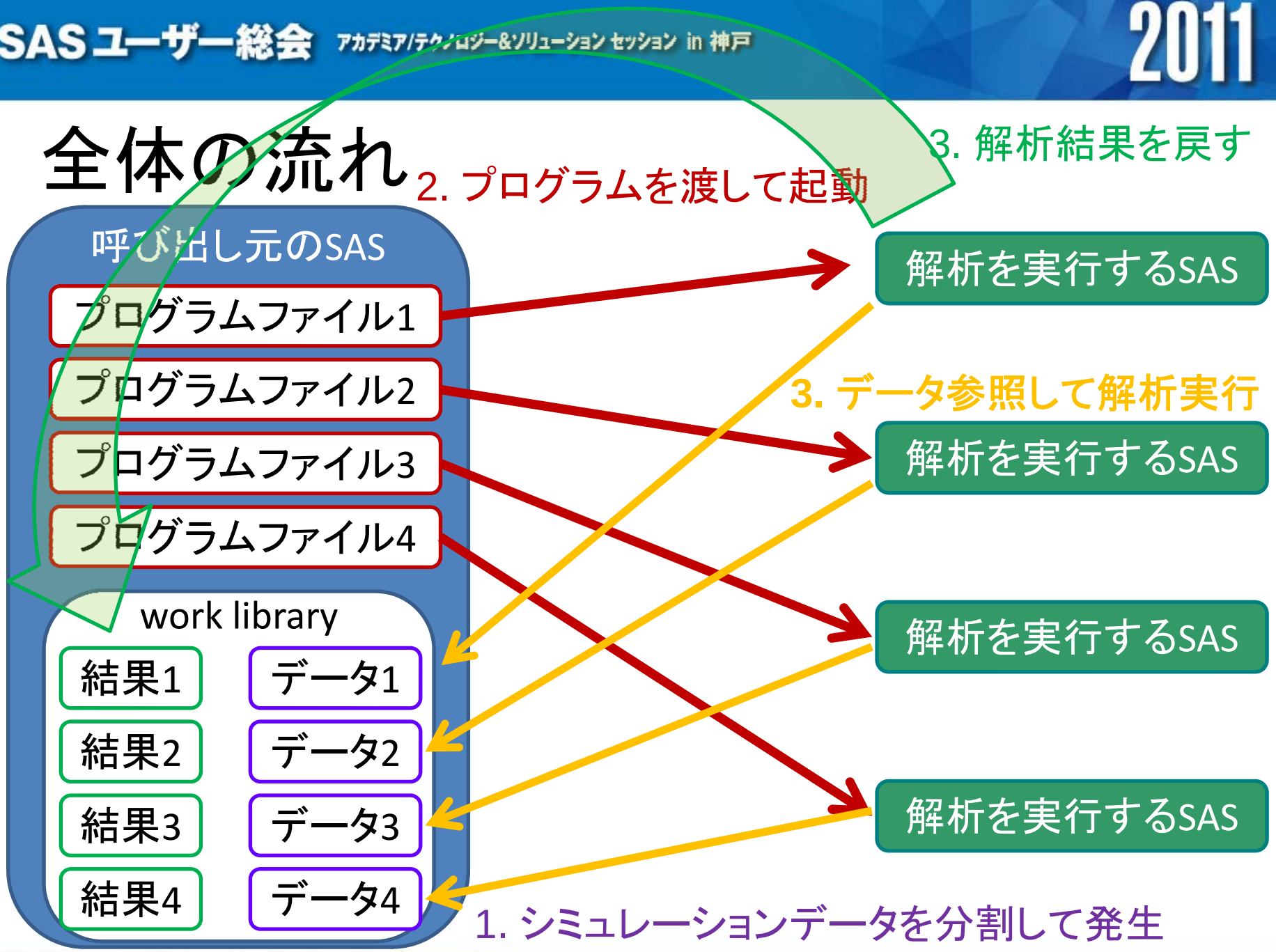
大規模シミュレーション研究

- 時間短縮の試み 1
  - 複数の設定で行う場合，設定ごとにプログラムを実行
  - 1つの結果が得られるまでの時間は変わらず
  - 試行錯誤に必要な時間短縮への寄与は少ない
- 時間短縮の試み 2
  - 1回のシミュレーションから同時実行可能な部分の抽出
  - 各反復ごとのデータ解析は他の反復に依存しない
  - 最も時間を要する解析部分の時間短縮
  - 1つの結果が得られるまでの時間が短縮

全体の流れ

- データ発生は一度に行う
  - 解析部分のためにデータを予め分割して準備
  - 乱数列の無規則性
    - １度データステップを抜けるとseedが初期化？
- 解析部分を同時実行
  - プログラムを生成し，別のSASを起動して実行させる
  - CPUと同じ数のSASを起動し，同時に解析を実行
  - データは起動元のwork libraryを参照
  - 結果も起動元のwork libraryに書き込む
- 結果をまとめる

全体の流れ


2. プログラムを渡して起動
3. 解析結果を戻す
呼び出し元のSAS
プログラムファイル1
プログラムファイル2
プログラムファイル3
プログラムファイル4
work library
結果1
結果2
結果3
結果4
データ1
データ2
データ3
データ4
解析を実行するSAS
解析を実行するSAS
解析を実行するSAS
解析を実行するSAS
3. データ参照して解析実行
1. シミュレーションデータを分割して発生

時間短縮のための準備

- SASプログラムからSASプログラムを生成
  - 複数のSASごとに異なるプログラム
    - 全く同一プログラムでは全く同じ結果
    - データセットの設定などの一部変更の必要性
  - 全ての起動したSASの実行終了後，結果を集計（同期処理）
    - 終了時にフラグファイルを作成し，その有無で終了判定
  - この技術は他の場面にも応用可能
- PCの準備
  - 64bit版のSAS（x64版）とメモリ容量の大きなPC
    - 大規模データを扱うためにメモリ制限がない方が望ましい
  - SSD等の高速なディスク（WORKに割り当て）

シミュレーション全体に関わる設定(1)

- マクロ変数で全体の環境設定

```
%LET WORKPATH=%SYSFUNC(GETOPTION(WORK));
%LET CPUCOUNT=%SYSCUNF(GETOPTION(CPUCOUNT));
%LET SASEXE="C:¥Program Files¥...（略）...¥sas.exe";
%LET CONFIG="C:¥Program Files¥... （略）...¥SASV9.CFG";
%LET DIRED=C:¥tmp¥SAS¥;
%LET ITER=1000; %LET SUBJECT=10000; %LET BETA=5; %LET SEED=4989;
```

- WORKPATHは呼び出す解析プログラムに現在のworkを知らせるため
- CONFIG中に解析時に起動されるSASのWORKを指定
- DIREDは一時ファイル置き場で，必ず「¥」までを指定
- ITERは反復回数，SUBJECTは対象者数，BETAはパラメタ，SEEDは乱数シード
- SASEXEの指定は必須ではない
- CPU・コア数は自動的に取得される（以後4つとする）

シミュレーション全体に関わる設定(2)

- 作成するプログラムファイル，フラグファイルの指定

```
%macro SetInitialState;
option nonotes;
data _null_;
   %do i=1 %to &CPUCOUNT.;
      %GLOBAL FILENAME&i.;  %GLOBAL FLAGFILE&i.;
      path1=cat("&DIRED.", "tmp&i..sas"); path2=cat("&DIRED.", "comp&i..sas");
      call symput("FILENAME&i.", trim(left(path1)));
      call symput("FLAGFILE&i.", trim(left(path2)));
   %end;
run;
%do i=1 %to &CPUCOUNT.;
   filename program&i. "&&filename&i..";
%end;
option notes;
%mend SetInitialState;
```

%DIREDで指定した場所にプログラム（tmp1.sas, tmp2.sas, tmp3.sas, tmp4.sas）と，実行が完了した際にフラグファイル（comp1.sas，comp2.sas, com3.sas, com4.sas）を作成することを指定（拡張子はsasである必要なし）

シミュレーション全体に関わる設定(3)

- 反復回数に応じたデータセットの分割指定

```
%macro SetIter;
option nonotes;
data _null_;
   end0=0;
   %do i=1 %to &CPUCOUNT.;
      %GLOBAL BEGIN&i.;
      %GLOBAL END&i.;
      BEGIN&i.=end%EVAL(&i.-1)+1;
      END&i.=ceil(&iter.*&i./&CPUCOUNT.);
      call symput("begin&i.", trim(left(BEGIN&i.)));
      call symput("end&i.", trim(left(END&i.)));
   %end;
run;
option notes;
%mend SetIter;
```

例えばCPUが4つで，反復が1000回の場合，

- 1 (BEGIN1) ～ 250 (END1)
- 251 (BEGIN2) ～ 500 (END2)
- 501 (BEGIN3) ～ 750 (END3)
- 751 (BEGIN4) ～ 1000 (END4)

に分割され，()内のマクロ変数に設定される

データ発生

```
%macro GenData;
%SetInitialState  %SetIter
option nonotes;
data %do i=1 %to &CPUCOUNT.; simdata_&i. %end;;
   call streaminit(&SEED.);
   do iter=1 to &ITER.;
      do i=1 to &SUBJECT.;
         x=RAND('NORMAL'); y=&BETA. * x + RAND('NORMAL');
         %do i=1 %to &CPUCOUNT.; if(&&BEGIN&i.. <= iter and iter <= &&END&i..)
            then output simdata_&i.;
         %end;
      end;
   end;
run;
option notes;
%mend GenData;
```

反復回数に応じて各データを
simdata_1からsimdata_4までに
分割して作成

プログラム生成 (1)

```
%macro GenerateProgram;
option nonotes;
%do i=1 %to &CPUCOUNT.;
data _null_;
  libstate=cat('libname workdata', ' "', "&WORKPATH.", '";');
  exitfile=cat('filename exitfile', ' "', "&&FLAGFILE&i..", '";');
  file program&i.; put libstate; put exitfile;
  put "ods exclude all;"; put "option nonotes;";
  put "ods output ParameterEstimates=workdata.est_&i.  ConvergenceStatus=workdata.cs_&i.;";
  put "proc nlmixed data=workdata.simdata_&i.;";
  put "    mu=beta*x;"; put "    model y ~ normal(mu, sigma);"; put "    by iter;"; put "run;";
  put "data _null_;";
  put "    file exitfile;";
  put '    put " ";';  put "run;";
run;
%end; option notes;
%mend GenerateProgram;
```

workと終了フラグファイルの指定

- 解析内容を変える際は，このプログラム中の前半のput文の解析部分を変更
- 解析内で読み込むデータはworkdata.simdata_&i.を指定し，また結果書き出しはworkdataに行う
- 終了フラグファイルは，何かファイルを作ればよい

プログラム生成 (2)

- 表現しづらい部分を拡大表示
  - (s) =「'」...single quote
    - 全てを文字列として扱う
  - (d) =「"」...double quote
    - マクロ変数を展開（&や%が意味を持つ）
  - 半角スペースを2倍にして強調

```
（途中略）
  libstate=cat('libname workdata', ' "', "&WORKPATH.", '";');
  libstate=cat((s)libname workdata(s), (s)  (d)(s), (d)&WORKPATH.(d), (s)(d);(s));
（途中略）
  exitfile=cat('filename exitfile', ' "', "&&FLAGFILE&i..", '";');
  exitfile=cat((s)filename exitfile(s), (s)  (d)(s), (d)&&FLAGFILE&i..(d), (s)(d);(s));
```

生成したプログラムの実行

```
%macro ExecuteProgram;
option NOXSYNC NOXWAIT;
data _null_;
   %do i=1 %to &CPUCOUNT.;
      x "  &SASEXE. &&FILENAME&i.. -nosyntaxcheck -nologo -config &CONFIG.";
   %end;
run;
option XSYNC XWAIT;
%mend ExecuteProgram;
```

- xコマンドの後に半角スペースは重要
  x "(半角スペース) &SASEXE.（以下略）"
- 非同期に複数のプログラムを走らせるためNOXSYNC，NOXWAITを指定して起動

解析プログラムの同期

```
%macro Syncronize;
data _null_;
   do until(fileexist("&FLAGFILE1.") =1 %do i=2 %to &CPUCOUNT.;
                                          and fileexist("&&FLAGFILE&i.") =1 %end;);
      call sleep(1,1);
   end;
run;
%mend Syncronize;
```

- 全ての終了フラグファイルが揃うまで待機（sleep(1,1)を繰り返す）

結果を一つのデータにまとめる

```
%macro CollectResult;
data est;
   set %do i=1 %to &CPUCOUNT.; est_&i. %end;;
run;
%mend CollectResult;
%macro Summarize;
proc sort data=est; by iter; run; proc sort data=cs; by iter; run;
data est;
   merge est cs;  by iter; if(Parameter ne "beta") then delete;  bias=estimate - &BETA.;
run;
ods output Summary=summ(keep=bias_Mean);
proc means data=est;  var bias; run;
data result;
   set result summ;
run;
%mend Summarize;
```

- CollectResultで複数のSASからの結果を一つにまとめる
- Summarizeで結果の集計を行う

生成したプログラムを消去

```
%macro DeleteProgram;
option NOXSYNC NOXWAIT;
%do i=1 %to &CPUCOUNT.;
data _null_;
   x "del &&FILENAME&i..";
   x "del &&FLAGFILE&i..";
run;
%end;
option XSYNC XWAIT;
%mend DeleteProgram;
```

- 作成したプログラムファイルや終了フラグファイルは消去しておく

シミュレーションのための準備

```
%macro Initall;
ods results=off;
data result;
   stop;
run;
%mend Initall;

%macro Init;
dm 'clear log';
dm 'clear output';
%mend Init;
```

```
data _null_;
   time1=datetime();
   put time1= DATETIME16.;
   call symput('t1', time1);
run;

data _null_;
   now=datetime();
   last=&t1;
   diff=now-last;
   put now= DATETIME16.;
   put diff= TIME12.2;
run;
```

- Initallはシミュレーションの最初に実行して結果の保存場所の生成など
- Initはlogやoutputを消去するマクロで，設定を変えて再度実行する際などに
- 右のマクロは時間計測用のマクロ

シミュレーション全体の実行

- 以下の順にマクロを実行

%Initall

%GenData

%GenerateProgram

%ExecuteProgram

%Syncronize

%CollectResult

%DeleteProgram

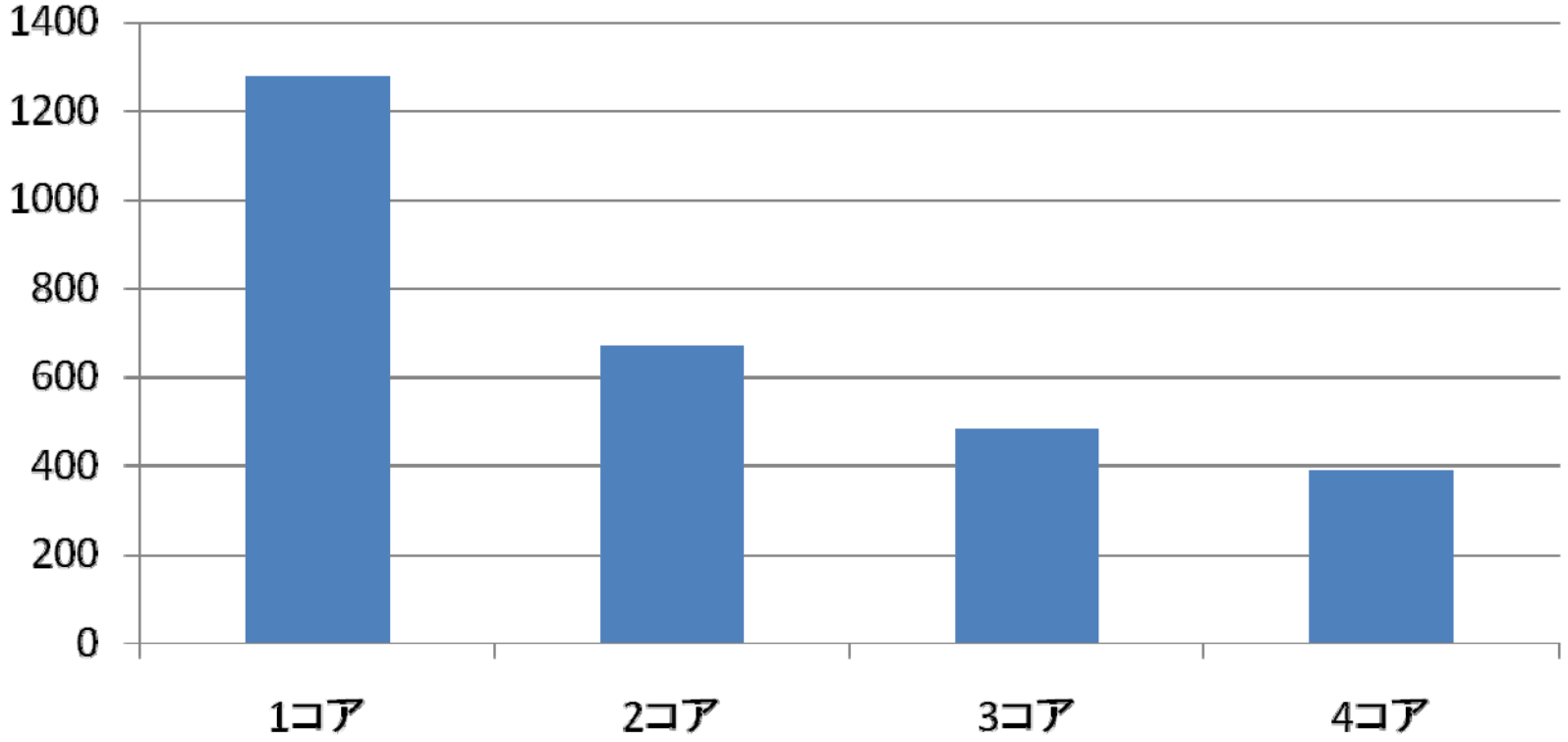
%Summarize

- 必要に応じて%GenDataの前で%LET BETA=2;のようにマクロ変数を変えることでシュミレーションの設定を変更する
- 今，どれだけCPU/ コアを使っているかを確認するには，Windowsではタスクマネージャを起動し（Ctrl + Alt + Del），CPU使用率を確認する
  - 4CPU/ コア搭載で1つしか使えない場合，25%前後になる

以上で手順は終了ですが…

- 恐らく
  - 実際に使い熟すまではかなり苦労すると思います
  - ただ，こんなマニアックなことをSASのマニュアルを調べながら1から構築するよりは遥かに楽に，プログラミングに時間もかけずに，シミュレーションの時間短縮が可能と思われます
- デバッグの際の参考に
  - option mprint; / option nomprint;
    マクロで生成されるプログラムを確認
  - %put _all_;
    マクロ変数の一覧を表示
  - option nonotesなどで出力は最低限にしておりますが，最初は正しくプログラムが動作する確認は必須です（念の為）

結果（使用コア数と実行時間（秒））



- Core2 Quad 2.66GHz，メモリ8GB，Windows Vista x64
- 対象者数10000，1000回の反復
- ポアソン回帰モデル（NLMIXEDプロシジャで尤度を最大化）
- 4通りの手法による解析

結論

- こんな面倒なことをもし0から構築するのであれば，時間短縮よりもプログラミングの手間の方が勝るためお勧めしません
- しかし，私が趣味で作ってしまいましたので，ここに広く公開いたします
- SASで大規模なシミュレーション研究を行う多くの方に，多少でもお役に立てば幸いです