ismaltx,ismsr 上の R について

目 次

1 ismaltx,ismsr 上の R の特徴

R はフリーの統計解析システムで，米国 Bell 研究所で開発された統計解析システム S(およびその商用版である S/S-Plus) と非常に高い互換性があります.

R は現在，R Core Team によって開発されており，主としてパーソナルな目的のために，Windows, Linux, MacOS X などのオペレーティングシステム用のバイナリが提供されており，また，そのソースコードも提供されています.

統計科学スーパーコンピュータシステム内の ismaltx,ismsx は共有分散メモリー型のスーパーコンピュータであり，主としてユーザが MPI や OpenMP などを利用して，Fortran または C(C++) で並列プログラムを書いて，実行することを目的としています．したがって，これまで統計解析システムは導入されていませんでしたし，また，現在のところ，並列計算を効果的に利用できる商用の統計解析システムはありません.

そこで，今回，R をこれらのシステムで利用できるようにするとともに，R から比較的簡単に，並列計算を行うことができるようにしました．実は，スーパーコンピュータとはいうものの，単体の CPU の性能は Windows の動いているパーソナルコンピュータのものとそれほど変わらないか，あるいは，たかだか数倍です．しかしながら ismaltx,ismsr 上で R を使うと以下のようなメリットがあります.

- 多くのメモリが利用できる.

  パーソナルコンピュータは，現在のところ，32bit CPU を利用しています (例えば Intel Pentium 4)．この CPU は最大 4GByte のメモリを利用できます．それに対して ismaltx は Intel Itanium2,ismsr は IBM PowerPC4+ という 64bit CPU を使用しており，利用可能なメモリ量は広大になります．実際には 1 CPU でも ismaltx では 32 GByte, ismsr では 64 GByte となります.

- 線形計算の自動並列化ができる.

  R の線形計算は線形計算ライブラリ BLAS を利用して行われるが，ismaltx 上では，BLAS の並列化バージョンである ScaLAPACK が利用できる．これにより，R のプログラムが自動的に並列実行できる.

- 並列プログラムが簡単に書け，また，実行できる.

  ismaltx,ismsr 上では，R の並列化機能 snow (Simple network of workstations) が利用できる．これに含まれる関数を使うと効果的な並列プログラムが，簡単に書ける．また，その実行も容易である.

2　簡単な使用法

2.1　ismaltx

ismaltx システムは 1 つのフロントエンド (ismaltx) と 4 つのバックエンド (ismaltx1〜ismaltx4) からなる．対話的に利用できるのは ismaltx のみであり，バックエンドはバッチ使用しかできない．

2.1.1　1CPU での利用

まず，フロントエンド ismaltx にログインする．
必要な環境変数を設定する：
bash の場合： source /usr/local/bin/env_local1.sh
csh, tcsh の場合： source /usr/local/bin/env_local1.csh
R (enter) で R が起動する．

2.1.2　複数の CPU での利用

環境変数も 1CPU と同様に設定します．
snow(SOCK) 以外は LAM-MPI を使うので,lamd をバックエンドで動かす必要があります．lamd の起動と停止 (lamboot,lamhalt) を mpiexec の-boot オプションを使うといっしょに行ってくれます．但し, 対話形式で複数のターミナルを使う場合,mpiexec -boot では複数走らせた場合に排他の概念が無いので, 開始時には lamboot, 終了時には lamhalt を行ったほうが良いでしょう．また,shell 等で連続して処理を行う場合は mpiexec 後すぐに mpiexec を行うと, 前のデーモンに対しての終了処理が行われず, すぐ次の処理が走ってしまい, 走ってから lamhalt(停止) される事になってしまいます.sleep を使うか,lamboot,lamhalt で挟んで連続した処理を行うとそういった事はありません．

snow(SOCK)　バックエンドの場合は BATCH になるので,
対話：R (enter) 又は R –no-save < hoge.R(enter)
バッチ：R CMD BATCH –no-save hoge.R
等とします. バッチの場合 –no-save をつけないと.RData に結果が書き込まれる．

snow(Rmpi)　-np にて並列数を決めます．
対話：mpiexec -boot -np 4 RMPISNOW –no-save < hoge.R
バッチ：mpiexec -boot -np 4 RMPISNOW CMD BATCH –no-save hoge.R
等とします．

2.2 ismsr

ismsr システムは 4 つのノード (ismsr,srnd01〜srnd04) からなる．ユーザはシステム管理者から割り当てられたノードにログインし，対話的に利用する．基本的には，ログインした一つのノードのみを利用し，すべてのノードを同時に使用することはできない．

2.2.1 1CPU での利用

まず，どれかのノードにログインする．
必要な環境変数を設定する：
bash の場合： source /usr/local/bin/env_local1.sh
csh, tcsh の場合： source /usr/local/bin/env_local1.csh
R (enter) で R が起動する．

2.2.2 複数の CPU での利用

snow(SOCK) のみ利用可能です．環境変数も 1CPU と同様に設定します．
バックエンドの場合は BATCH になるので,
対話：R (enter) 又は R –no-save < hoge.R(enter)
バッチ：R CMD BATCH –no-save hoge.R
等とします.–no-save をつけないと.RData に結果が書き込まれます.

snow(SOCK) バックエンドの場合は BATCH になるので,
対話：R (enter) 又は R –no-save < hoge.R(enter)
バッチ：R CMD BATCH –no-save hoge.R
等とします.–no-save をつけないと.RData に結果が書き込まれます.

3 環境

3.1 ISMALTX

3.1.1 環境変数

以下の設定を.bash_profile 等におこないます.

変数名		推奨値
		設定内容
PATH	必須	/usr/local1/gcc-3.4.2/bin:/usr/local1/bin:$PATH
		lam コマンドは/usr/local1/bin 以下にインストールされています.
	追加済	/opt/intel_cc_80/bin:/opt/intel_fc_80/bin:$PATH
		Intel Compiler 用のパスの追加
LD_LIBRARY_PATH	必須	/usr/local1/gcc-3.4.2/lib:/usr/local1/lib:$LD_LIBRARY_PATH
		lam library は/usr/local1/lib 以下にインストールされています.
	追加済	/opt/intel_cc_80/lib:/opt/intel_fc_80/lib:$LD_LIBRARY_PATH
		Intel Compiler のライブラリを追加します
LAM_MPI_SESSION_PREFIX	任意	/tmp
		通信用の共有テンポラリディレクトリ作成位置指定. 異なるマシン間では NFS マウントする必要があります. 無指定の場合は TMPDIR が使われます.

表 1: ISMALTX 環境変数設定一覧

変数名		推奨値
		設定内容
MANPATH	任意	/usr/local1/gcc-3.4.2/man:/usr/local1/man:$MANPATH
		lam マニュアルは/usr/local1/man 以下にインストールされています.
LAMCC	開発者	icc
		mpicc のコンパイラを指定します.
LAMCXX	開発者	icc
		mpic++ のコンパイラを指定します.
LAMF77	開発者	ifort
		mpif77 のコンパイラを指定します.

表 2: ISMALTX 用 LAM-MPI 向け環境変数設定一覧

留意事項　g77 利用時は R が-fno-second-underscore [1]で構築されている事に注意してください. g77 のデフォルト (f2c 互換) では SUBROUTINE foo_bar 等アンダースコアを含むシンボルは_(アンダースコア) ではなく__(ダブルアンダースコア) が関数名の後ろに付与されます.IntelFortran 等では_(アンダースコア) のみの付与となります.

ATLAS,LAM-MPI,BLACS,SCALAPACK は icc+ifort で構築してあります.

[1]hoge.so オブジェクトが gcc 版 R でも Intel 版 R でも使えるように

3.1.2 PBS

バッチ処理用のテンプレートとして
ismaltx:/home0/sample/ismaltx/template/template_R_para.sh
ismaltx:/home0/sample/ismaltx/template/template_R_para.csh
が用意されています.

template_R_para.sh

```
#!/bin/csh

##PBS -q q8
##PBS -q q16
##PBS -q q32
##PBS -q q8r
##PBS -q q16r
##PBS -q q32r
##PBS -q q32m
##PBS -q q64
##PBS -q q128
#PBS -l ncpus=XX
##PBS -N job_name
##PBS -m ae
##PBS -j oe

/usr/local1/bin/env_local1.sh

# snow(SOCK) or single
# R CMD BATCH --no-save script_file.R
#
# snow(Rmpi)
# mpiexec -boot -np XX RMPISNOW CMD BATCH --no-save script_file.R
#
# RScaLAPACK
# mpiexec -boot N R CMD BATCH --no-save script-file.R
```

template_R_para.csh

```
#!/bin/csh

##PBS -q q8
##PBS -q q16
##PBS -q q32
##PBS -q q8r
##PBS -q q16r
##PBS -q q32r
##PBS -q q32m
##PBS -q q64
##PBS -q q128
#PBS -l ncpus=XX
##PBS -N job_name
##PBS -m ae
##PBS -j oe

/usr/local1/bin/env_local1.csh

# snow(SOCK) or single
# R CMD BATCH --no-save script_file.R
#
# snow(Rmpi)
# mpiexec -boot -np XX RMPISNOW CMD BATCH --no-save script_file.R
#
# RScaLAPACK
# mpiexec -boot N R CMD BATCH --no-save script-file.R
```

3.1.3 R

BLAS の選択 /usr/local1/bin/R を実行すると $HOME/.Rconf を参照しそれぞれの R を実行するようにしました.

${HOME}/.Rconf の例

```
RVER=1.9.1              # R のバージョン
RCOMPILER=intel8        # intel8 compiler
#RCOMPILER=gcc          # gcc-3.4.2
#RBLAS=                 # normal BLAS ( gcc is netlib BLAS )
RBLAS=atlas             # ATLAS 3.6.0
#RBLAS=scs              # SGI SCSL
#RBLAS=goto             # libgoto 0.95
```

上記の例では,icc でコンパイル+ATLAS を利用した R が起動します.

R と BLAS の捕捉 R の構築は gcc では全く (遅いことを除いて) 問題が無いものの,Intel Compiler では精度に関するエラーがデフォルトでは出てしまいます.Intel Compiler の-mp オプションは IEEE に出来るだけ沿った形でコードを生成してくれます.

BLAS では計算順序や精度についての規則が無いため, メモリを経由しない演算や乗加算融合[2]による誤差が発生します.

ATLAS はソースからコンパイルが可能でしたので,Intel Compiler にて-mp (乗加算融合等を行わない) を指定して R にとって誤差の少ない BLAS を構築しています.

[2]乗算と加算を fma(a,b,c) 等と, 一度の処理で無限精度で処理を行う

SCSL は BLAS と LAPACK が融合されたライブラリの為,–without-lapack としても R の lapack.so は使用されず,BLAS としてロードした SCSL の LAPACK 関数が使われてしまいます. R では dgeev を rgeev に置き換える等の対策 (SUN 用) もとられていますが, SCSL では rgeev(lapack.so) から dhseqr(SCSL) と渡り (lapack.so とシンボルが重複し,libscs の関数を見にいってしまいます),pam.R の eigen でエラーコード 500 が帰って来てしまいます.

SCSL 使用時の問題

```
$ /usr/local1/R-1.9.1.gccscs/bin/R -q --no-save < pam.R
> library(cluster)
> set.seed(253)
> x <- rbind(cbind(rnorm(120, 0,8), rnorm(120, 0,8)),
+          cbind(rnorm(130,50,8), rnorm(130,10,8)))
> pdx <- pam(dist(x), 2)
> clusplot(pdx, dist=dist(x))
Error in eigen(Z, symmetric = FALSE, only.values = TRUE) :
        error code 500 from Lapack routine dgeev
Execution halted
```

GOTO では example(censboot) にてエラーとなりますが,unique で値を処理する時にメモリ (仮数部 53bit) を経由せず (仮数部 64bit) に乗加算融合を行いその誤差で集約出来ません.

GOTO 使用時の問題と対策

```
$ /usr/local1/R-1.9.1.gccgoto/bin/R -q --no-save <censboot.R
> library(survival)
> library(boot)
> data(melanoma, package="boot")
> library(splines)# for ns
> mel.cox <- coxph(Surv(time,status==1)~ns(thickness,df=4)+strata(ulcer),
+      data=melanoma)
> mel.surv <- survfit(mel.cox)
> agec <- cut(melanoma$age,c(0,39,49,59,69,100))
> mel.cens <- survfit(Surv(time-0.001*(status==1),status!=1)~
+      strata(agec),data=melanoma)
> mel.fun <- function(d) {
+      t1 <- ns(d$thickness,df=4)
+      cox <- coxph(Surv(d$time,d$status==1) ~ t1+strata(d$ulcer))
+      eta <- unique(cox$linear.predictors)
+      u <- unique(d$thickness)
+      sp <- smooth.spline(u,eta,df=20)
+      th <- seq(from=0.25,to=10,by=0.25)
+      predict(sp,th)$y
+ }
> mel.str<-cbind(melanoma$ulcer,agec)
> # this is slow!
> mel.mod <- censboot(melanoma,mel.fun,R=999,F.surv=mel.surv,
+      G.surv=mel.cens,cox=mel.cox,strata=mel.str,sim="model")
Error in xy.coords(x, y) : x and y lengths differ
Execution halted

# 以下のようにすれば処理可能
eta <- unique(signif(cox$linear.predictors,digit=11))
```

BLAS とコンパイラの組み合わせ　速度の遅い組み合わせから書いています.

COMPILER / BLAS	gcc	icc
netlib blas	良好	
R blas		良好
ATLAS(-mp)	良好	良好
SCSL	pam で問題あり	pam で問題あり
GOTO	censboot で問題あり	censboot で問題あり

表 3: ISMALTX の R と BLAS の組み合わせ

make check-all で表面化したケースのみです. 実数を直接 unique で集約する example が問題と考えれば icc+GOTO が最も高速ですし, あくまで IA32 と同じ結果を求めるなら icc+ATLAS が最良です.

3.2 ISMSR

BLAS は ATLAS を xlc+f90 でコンパイルしたものがあります.

BLAS \ COMPILER	xlc + f2c	xlc + f90
R blas	R2	R9
ATLAS		R9a

表 4: ISMSR の R と BLAS の組み合わせ

PATH に /usr/local1/bin を追加し,R(R9a),R2,R9,R9a のいずれかで R が起動します. デフォルト R(/usr/local1/bin/R) は f90+ATLAS 版です.

4 R 並列化使用例

4.1 snow(SOCK)

snow (http://www.stat.uiowa.edu/~luke/R/cluster/cluster.html) の SOCK では自ホストや自ホスト以外の R(要 snow) と分散処理が可能となります. Rmpi,RScaLAPACK の利用が出来ない ismsr での利用を想定します. 通常は rsh や ssh を介して slave となるプロセスを起動しますが, 単一ノードでは slave のプロセスの振り分けができないので rsh-fake 経由による並列化を行います.

snow(SOCK) の例

```
require(snow)
                                        # スレーブに渡す関数定義
foo1<-function(x) { mean(x) } ; foo2<-function(x) { min(x) }
foo3<-function(x) { max(x) }  ; foo<-list(foo1,foo2,foo3)
                                        # 関数の数を基にスレーブ数を決定
task<-length(foo)                       # タスク数
                                        # hostname は識別用に使用します
remotenodes <- paste("localhost",1:task,sep="")
calc <-function(nodes,n,fun)
{
   # Rmpi と同様に扱えるようホスト名でランク付け
   rank<-(1:length(nodes))[nodes==Sys.getenv("SOCKRANK")]
   set.seed(rank)
   x1<-array(runif(n*n),dim=c(n,n))
   x2<-x1%*%x1
   # write.table(x2,file=paste("hoge", rank, ".tsv", sep=""))
   fun[[rank]](x2)
}


                                        # プログラム指定
setDefaultClusterOptions(rprog="/usr/local1/R-1.9.1/bin/R")
                                        # snow スクリプト位置指定
setDefaultClusterOptions(scriptdir="/usr/local1/R-1.9.1/lib/R/library/snow")


                                        # slave 起動
cl <- makeSOCKcluster(remotenodes, rshcmd="rsh-fake")
                                        # 各スレーブにライブラリのロード
clusterEvalQ(cl,library(boot))
                                        # 仕事振り分け
rc <- clusterCall(cl, calc,remotenodes,100,foo)
sapply(rc,print)
```

青字は異なるマシン間で R の PATH が異なる場合利用します.

/usr/local1/bin/rsh-fake

```
#!/bin/sh

if [ $# -lt 1 ] ;then
        echo "usage: $0 -l user hostname [command [arg...]]"
        exit 1
fi

if [ $# -lt 4 ] ;then
        echo "usage: $0 -l user hostname [command [arg...]]"
        exit 1
fi

SOCKRANK=$3
export SOCKRANK

shift 3
exec $*
```

snow(SOCK) 実行の例

```
# 対話形式
$ R --no-save < ~/snowsock.R

# バッチ形式
$ R CMD BATCH --no-save ~/snowsock.R
```

snow(SOCK) 例

```
require(snow)

# データ
x <- rnorm(100000)
y <- rpois(100000, exp(1+x))
save(file="st1.img",list=c("x","y"))

# 普通の処理
sts<-proc.time()
glm(y ~x, family=quasi(var="mu", link="log"))$coefficients
glm(y ~x, family=poisson)$coefficients
glm(y ~x, family=quasi(var="mu^2", link="log"))$coefficients
eds<-proc.time()

print(eds-sts)

# 処理別
glmList<-list(
                expression(glm(y ~x, family=quasi(var="mu", link="log"))),
                expression(glm(y ~x, family=poisson)),
                expression(glm(y ~x, family=quasi(var="mu^2", link="log")))
                )
glmList

# 関数の数を基にスレーブ数を決定
task<-length(glmList)   # タスク数

# rsh-fake は hostname を参照しませんが識別用に使用します
remotenodes <- paste("localhost",1:task,sep="")

# スレーブで実行する関数
calc <-function(nodes,glmList)
{
   load(file="st1.img")
   # Rmpi と同様に扱えるようホスト名でランク付け
   rank<-(1:length(nodes))[nodes==Sys.getenv("SOCKRANK")]
   eval(glmList[[rank]])$coefficients
}

# slave 起動
cl <- makeSOCKcluster(remotenodes, rshcmd="rsh-fake")

# 仕事振り分け
stm<-proc.time()
rc <- clusterCall(cl, calc,remotenodes,glmList)
edm<-proc.time()
print(edm-stm)

sapply(rc,print)
```

SOCK は大きなデータのやり取りでは明らかに遅くなるので, データの部分的抽出や結果をファイルに出力する場合等に向いているのではないかと考察します.

4.2 snow(Rmpi)

http://www.stat.uiowa.edu/ luke/R/cluster/cluster.html

snow(Rmpi) の例

```
library(Rmpi);library(snow)

seed <-function()
{
   rank <- mpi.comm.rank(0)
   set.seed(rank)
   rank
}
calc <-function(n=1000)
{
   library("stats")
   x1<-array(runif(n*n),dim=c(n,n))
   x2<-x1%*%x1
   #write.table(x2,file=paste("hoge",mpi.comm.rank(0),".tsv",sep=""))
   mean(x2)
}
cl<-getMPIcluster()
clusterCall(cl, seed)
rc <- clusterCall(cl, calc,n=1000)
sapply(rc,print)
```

snow(Rmpi) 実行の例

```
# 対話形式
$ mpiexec -boot -np 4 RMPISNOW --no-save < ~/snowRmpi.R

# バッチ形式
$ mpiexec -boot -np 4 RMPISNOW CMD BATCH --no-save ~/snowRmpi.R
```

/usr/local1/bin/RMPISNOW 中で R_PROFILE=/usr/local1/etc/RMPISNOWprofile としています. したがって,–vanilla を付けた場合は sleve 側の処理が読み込めません.

snow(Rmpi) 例

```
require(snow)
require(Rmpi)

# データ
x <- rnorm(200000)
y <- rpois(200000, exp(1+x))
save(file="st1.img",list=c("x","y"))

# 普通の処理
sts<-proc.time()
glm(y ~x, family=quasi(var="mu", link="log"))
glm(y ~x, family=poisson)
glm(y ~x, family=quasi(var="mu^2", link="log"))
eds<-proc.time()

print(eds-sts)

# 処理別
glmList<-list(
                expression(glm(y ~x, family=quasi(var="mu", link="log"))),
                expression(glm(y ~x, family=poisson)),
                expression(glm(y ~x, family=quasi(var="mu^2", link="log")))
                )

#@# 関数の数を基にスレーブ数を決定
#@task<-length(glmList) # タスク数

# rsh-fake は hostname を参照しませんが識別用に使用します
#@remotenodes <- paste("localhost",1:task,sep="")

# スレーブで実行する関数
#@calc <-function(nodes,glmList)
calc <-function(glmList)
{
   require(stats)
   load(file="st1.img")
   #@# Rmpi と同様に扱えるようホスト名でランク付け
   #@rank<-(1:length(nodes))[nodes==Sys.getenv("SOCKRANK")]
   rank<-mpi.comm.rank(0)
   eval(glmList[[rank]])
}
# slave 起動
#@cl <- makeSOCKcluster(remotenodes, rshcmd="rsh-fake")
cl <- getMPIcluster()

# 仕事振り分け
stm<-proc.time()
rc <- clusterCall(cl, calc,glmList)
edm<-proc.time()
print(edm-stm)
sapply(rc,print)
```

SOCK に比べると通信性能は良好でそのまま結果を返すような処理でも実用に耐えます.

4.3 RScaLAPACK

RScaLAPACK は,intel V8 Compiler 版の R でのみ、実行可能です.
http://www.aspect-sdm.org/Parallel-R/RScaLAPACK/documentation.html

RScaLAPACK の例

```
.RscalaGrid<-c(2,4)
m=8000
set.seed(123)
library(RScaLAPACK)
A<-array(rnorm(m^2), dim=c(m,m))
B<-array(rnorm(m^2), dim=c(m,m))
AB<-A%*%B
sum(B)

st<-proc.time()
Z<-sla.solve(A,AB)
en<-proc.time()
sum(Z)
en-st

st<-proc.time()
Z<-solve(A,AB)
en<-proc.time()
sum(Z)
en-st
```

RScaLAPACK 実行の例

```
# 対話形式
$ mpiexec -boot N R --no-save < ~/rscalapack.R

# バッチ形式
$ mpiexec -boot N R CMD BATCH --no-save ~/rscalapack.R
```

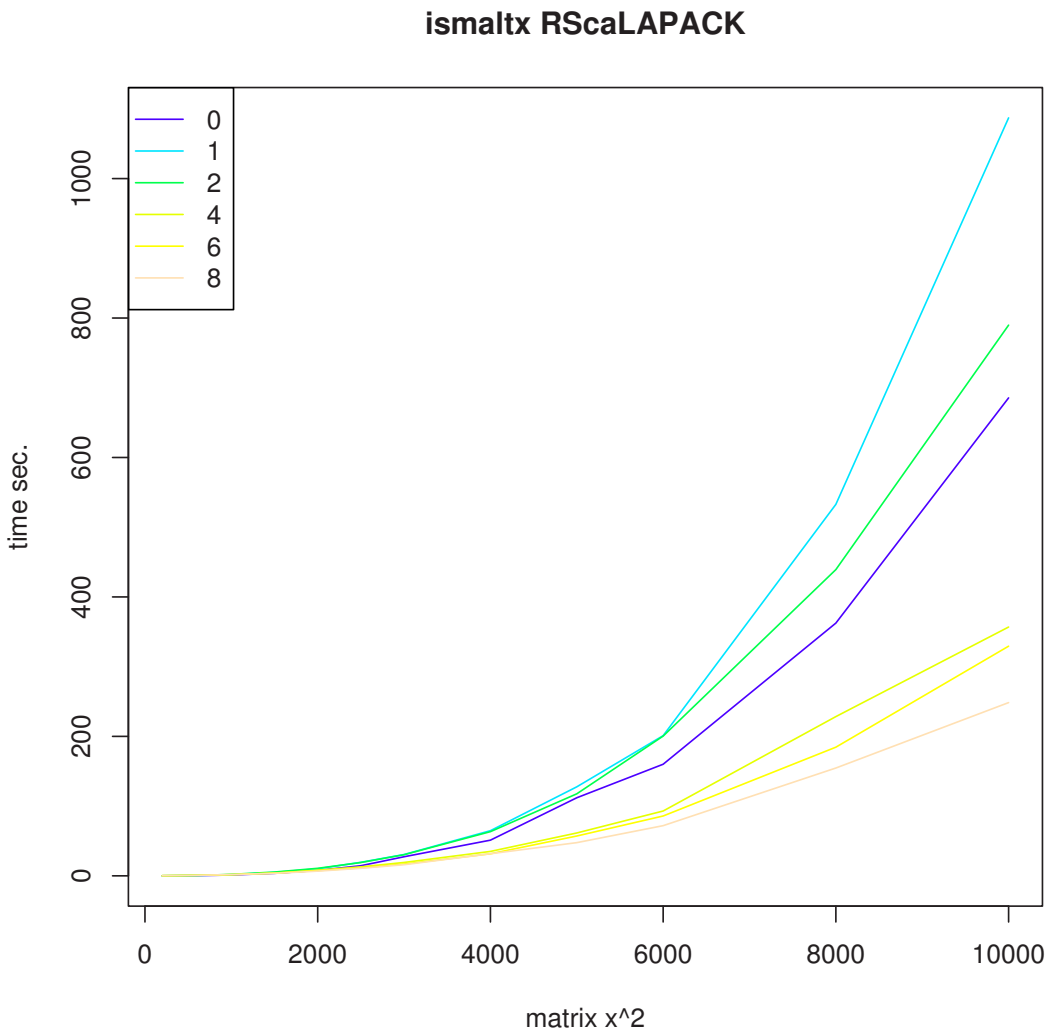

図 1: sla.solve の並列数と実行時間

5　追加パッケージのインストール

5.1　ディレクトリの作成

uname -r の結果を基にライブラリのパスを決定します. Altix では”ia64”,SR11000 では”001020004C00” が返ります. 利用する OS 毎に以下のコマンドを実行します.

```
$ mkdir -p  /.R-1.9.1/‘uname -m‘
```

5.2　R のライブラリ検索パスへの追加

“.Rprofile” に以下のように記述します.

.Rprofile

```
# .libPaths("~/.R-1.9.1/ia64") or .libPaths("~/.R-1.9.1/001020004C00")
.libPaths(paste(Sys.getenv("HOME"),
                .Platform$file.sep,                              # /
                ".R-",
                paste(version[c("major", "minor")],collapse="."), # 1.9.1
                .Platform$file.sep,                              # /
                as.vector(Sys.info()["machine"]),                # ia64
                sep=""))

options(CRAN="http://cran.md.tsukuba.ac.jp/")
```

–vanilla 等のオプションで.Rprofile を読み込まない場合は, スクリプトの先頭に,.libPaths を使い, パスの追加を行います.

5.3 追加パッケージのインストール

ライブラリのインストール先を指定してインストールを行います. 標準では CFLAGS 等不必要なオプションが入ってますので,

unset CFLAGS

unset FFLAGS

など, 必要にあわせてから, インストールを行います.

R から追加パッケージのインストール

```
# tree パッケージのインストール
# > install.packages("tree", lib="~/.R-1.9.1/ia64")
> install.packages("tree",
lib=
paste(Sys.getenv("HOME"),
      .Platform$file.sep,
              ".R-",
              paste(version[c("major", "minor")],collapse="."),
              .Platform$file.sep,
              as.vector(Sys.info()["machine"]),
              sep="")
                  )
```

シェル上からのインストールを行う場合, 一旦 tar.gz のソースパッケージをダウンロードして, 以下のようにしてインストールを行います.

Shell から追加パッケージのインストール

```
$ R_LIBS=~/.R-1.9.1/‘uname -m‘
$ R CMD INSTALL --library=$R_LIBS car_1.0-13.tar.gz
```

6 ESS

/usr/local1/share/emacs/site-lisp/ess にバイトコンパイルした物が格納してあります.

.emacs 設定例

```
(set-language-environment "Japanese")
(set-default-coding-systems 'euc-jp)
(set-terminal-coding-system 'euc-jp)
(set-keyboard-coding-system 'euc-jp)
(set-buffer-file-coding-system 'euc-jp)
(setq load-path
  (cons
    (expand-file-name "/usr/local1/share/emacs/site-lisp/ess")
   load-path
  )
)
(require 'ess-site)
(setq ess-ask-for-ess-directory nil)
(setq ess-pre-run-hook
  '((lambda () (setq S-directory default-directory)
               (setq default-process-coding-system '(euc-jp . euc-jp))
   )))
```