epiで並列Rを使う(2010/05/21)

2010年5月19日
18:12

ここではepi00にアカウント名shimoでログインして作業します.

次のようにして，アカウント名kanriのホームにあるtest20100517.tgzを自分のホームにコピーして，内容を展開してください.

```
shimo@epi00:~$ ls
dot.ssh  sample.txt  work  work00
shimo@epi00:~$ ls ~kanri
etcpack1              rup                VMwareTools-8.1.4-227600.tar.gz
make_known_hosts.sh   test20100517.tgz   vmware-tools-distrib
mytools               tmpdir.tgz
mytools-20100519.tgz  tmppass.tgz
shimo@epi00:~$ cp ~kanri/test20100517.tgz .
shimo@epi00:~$ tar xvfz test20100517.tgz
test20100517/
test20100517/lamhosts1
test20100517/test1.R
test20100517/test1.Rout
test20100517/test1.R~
test20100517/test1.ps
shimo@epi00:~$ ls
dot.ssh  sample.txt  test20100517  test20100517.tgz  work  work00
shimo@epi00:~$
```

画面の領域の取り込み 日時: 2010/05/19 18:17

展開してできたディレクトリtest20100517に移動します．test1.Rとlamhosts1をコピーしてtest2.Rとlamhosts2をつくります.

```
shimo@epi00:~$ ls
dot.ssh  sample.txt  test20100517  test20100517.tgz  work  work00
shimo@epi00:~$ cd test20100517
shimo@epi00:~/test20100517$ ls
lamhosts1  test1.ps  test1.R  test1.R~  test1.Rout
shimo@epi00:~/test20100517$ cp test1.R test2.R
shimo@epi00:~/test20100517$ cp lamhosts1 lamhosts2
shimo@epi00:~/test20100517$
```

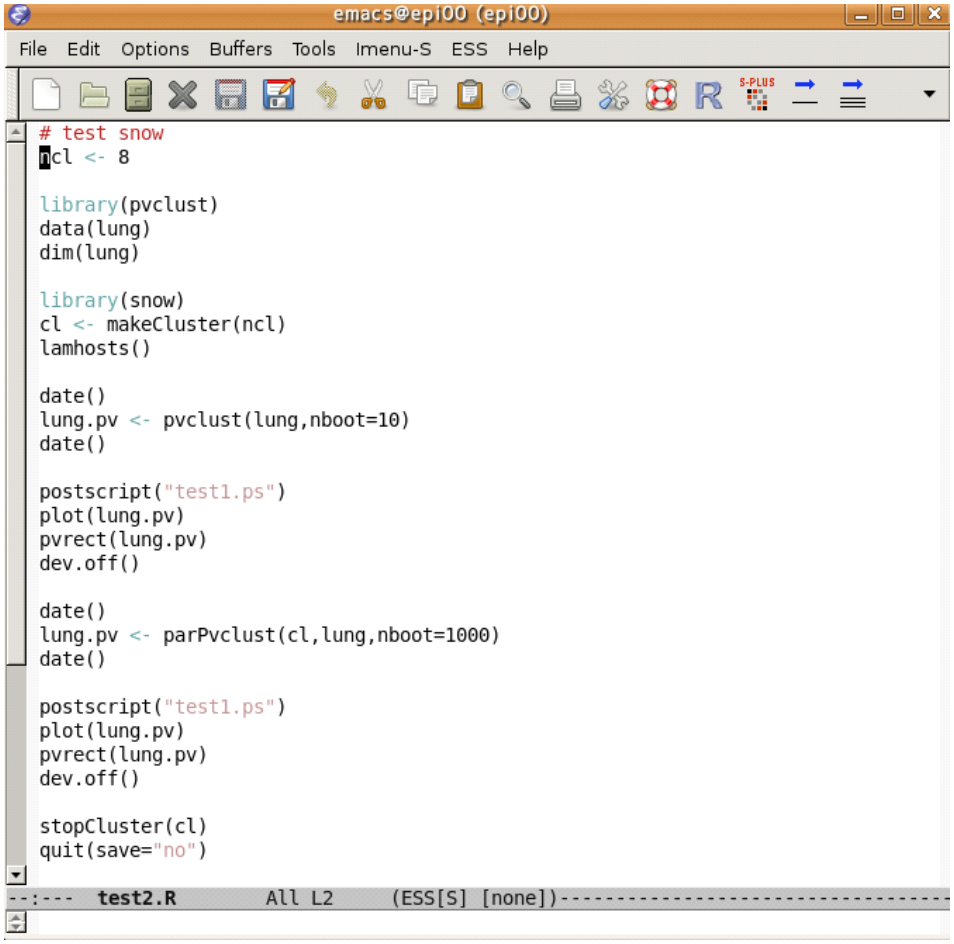
画面の領域の取り込み 日時: 2010/05/19 18:19

これをemacsで編集します．emacs lamhosts2を実行してください.

```
epi00 cpu=4
epi01 cpu=4
```

画面の領域の取り込み 日時: 2010/05/19 18:22

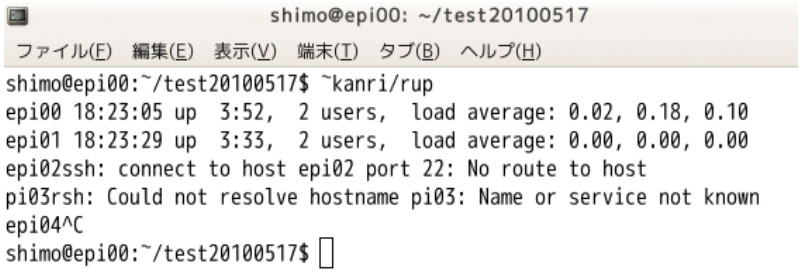
これはepi00はcpu=4個，epi01はcpu=4個という設定です．今回はこのままつかいます．変更するときは，Control-X Control-Sで保存してください．次にContorl-X Control-Fで別のファイルを開きます．test2.Rを開いてください.



画面の領域の取り込み 日時: 2010/05/19 18:24

最初の ncl <- 8が利用するcpu数の合計です．ここではこのままにしておきます．変更した場合，Control-X Control-Sで保存してください．Control-X Control-Cで終了です．

~kanri/bin/rupを実行すると，各ノードの稼働状況がわかります．（以下の説明中で~kanri/rupと記述してあるのは~kanri/bin/rupに読み替えてください）．途中でControl-Cで止めてください．ここではepi00とepi01がパワーオンしていてload averageが低く，他の人が計算してないのがわかります．



画面の領域の取り込み 日時: 2010/05/19 18:28

lamboot lamhosts2を実行します．

```
shimo@epi00:~/test20100517$ lamboot lamhosts2

LAM 7.1.2/MPI 2 C++/ROMIO - Indiana University

shimo@epi00:~/test20100517$
```

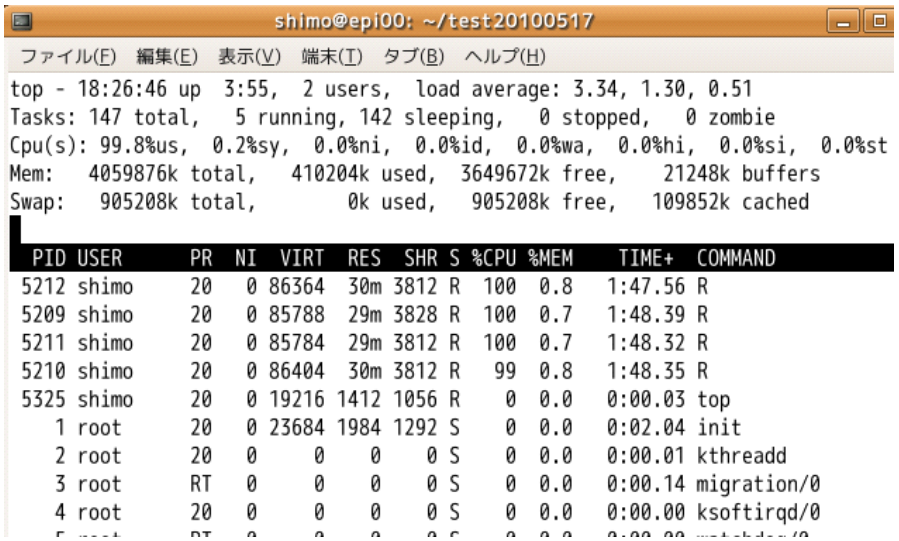
画面の領域の取り込み 日時: 2010/05/19 18:28

R CMD BATCH test2.R >& test2.log & とすると実行開始です．エラーなどはtest2.logに書き出されます．実行はバックグラウンドにしたいので，最後に&をつけています．こうすると，他の作業をしたりログアウトしたりしても，Rの計算は動き続けます．

```
shimo@epi00:~/test20100517$ R CMD BATCH test2.R >& test2.log &
[1] 5200
shimo@epi00:~/test20100517$
```

画面の領域の取り込み 日時: 2010/05/19 18:29

topとするとepi00での状況がわかります．load averageの時間平均（過去何分間かの平均値）がだんだん４にちかずきます．qで終了．



画面の領域の取り込み 日時: 2010/05/19 18:31

rupをみると，やはりepi00でもepi01でも４くらいです．

```
shimo@epi00:~/test20100517$ ~kanri/rup
epi00 18:27:38 up  3:56,  2 users,  load average: 3.87, 1.78, 0.72
epi01 18:28:02 up  3:37,  2 users,  load average: 3.79, 1.77, 0.68
epi02^C
shimo@epi00:~/test20100517$
```

画面の領域の取り込み 日時: 2010/05/19 18:32

rupというのは，shell scriptで書いた次のコマンドです．あとで自分のホームにbinというディレクトリをつくり，その中にコピーしてください．いちいち~kanri/bin/rupと打たずにrupだけで使えるようになります．（次にログインするとき，自動的に$HOME/binにpathがとおるように.profileが設定されています．）

```
shimo@epi00:~$ cat ~kanri/bin/rup
#!/bin/sh

for i in epi00 epi01 epi02 pi03 epi04 epi05 epi06 epi07 epi08 epi09 epi10 epi11 epi12 epi13 epi14 epi15 epi16
epi17 epi18 epi19 epi20  ; do
echo -n $i
rsh $i uptime
done
```

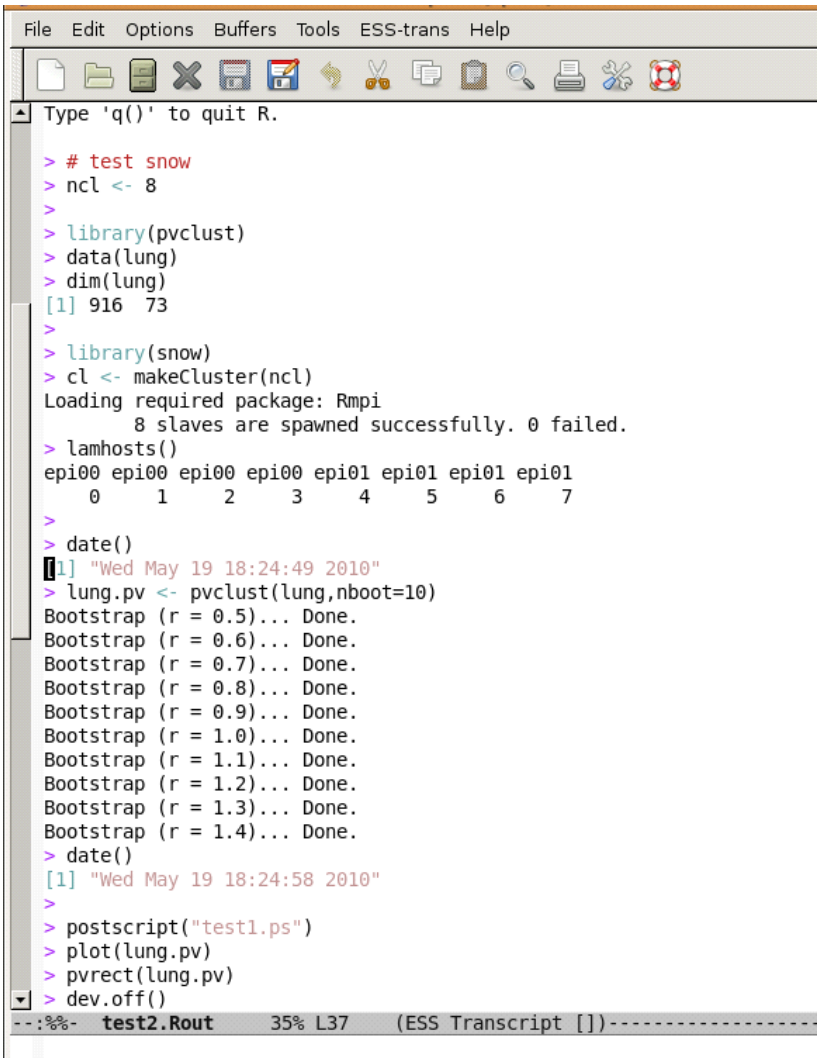
rsh epi00 uptime, rsh epi01 uptime, ..., rsh epi20 uptimeを順番に実行するだけです．自分の使いやすいように，あとで各自編集して工夫してください（本拠地」のホスト名を最初にして並べ替えるなどする）．epiではrshの部分はsshとしても同じ動作になるように設定してあります．

test2.Rの計算は数分で実行が終わります．できたファイルをみます．

```
shimo@epi00:~/test20100517$ ls -lt
total 44
-rw-r--r-- 1 shimo shimo  1842 2010-05-19 18:27 test2.Rout
-rw-r--r-- 1 shimo shimo 15920 2010-05-19 18:27 test1.ps
-rw-r--r-- 1 shimo shimo     0 2010-05-19 18:24 test2.log
-rw-r--r-- 1 shimo shimo    24 2010-05-19 18:13 lamhosts2
-rw-r--r-- 1 shimo shimo   376 2010-05-19 18:13 test2.R
-rw-r--r-- 1 shimo shimo  1842 2010-05-17 15:30 test1.Rout
-rw-r--r-- 1 shimo shimo   376 2010-05-17 15:28 test1.R
-rw-r--r-- 1 shimo shimo    47 2010-05-17 15:05 test1.R~
-rw-r--r-- 1 shimo shimo    24 2010-05-17 15:03 lamhosts1
[1]+  Done                    R CMD BATCH test2.R &>test2.log
shimo@epi00:~/test20100517$
```

画面の領域の取り込み 日時: 2010/05/19 18:34

test2.Routをみます．cat test2.Routとするかlv test2.Routかemacs test2.Routか，どれでも好きな方法で見てください．



画面の領域の取り込み 日時: 2010/05/19 18:37

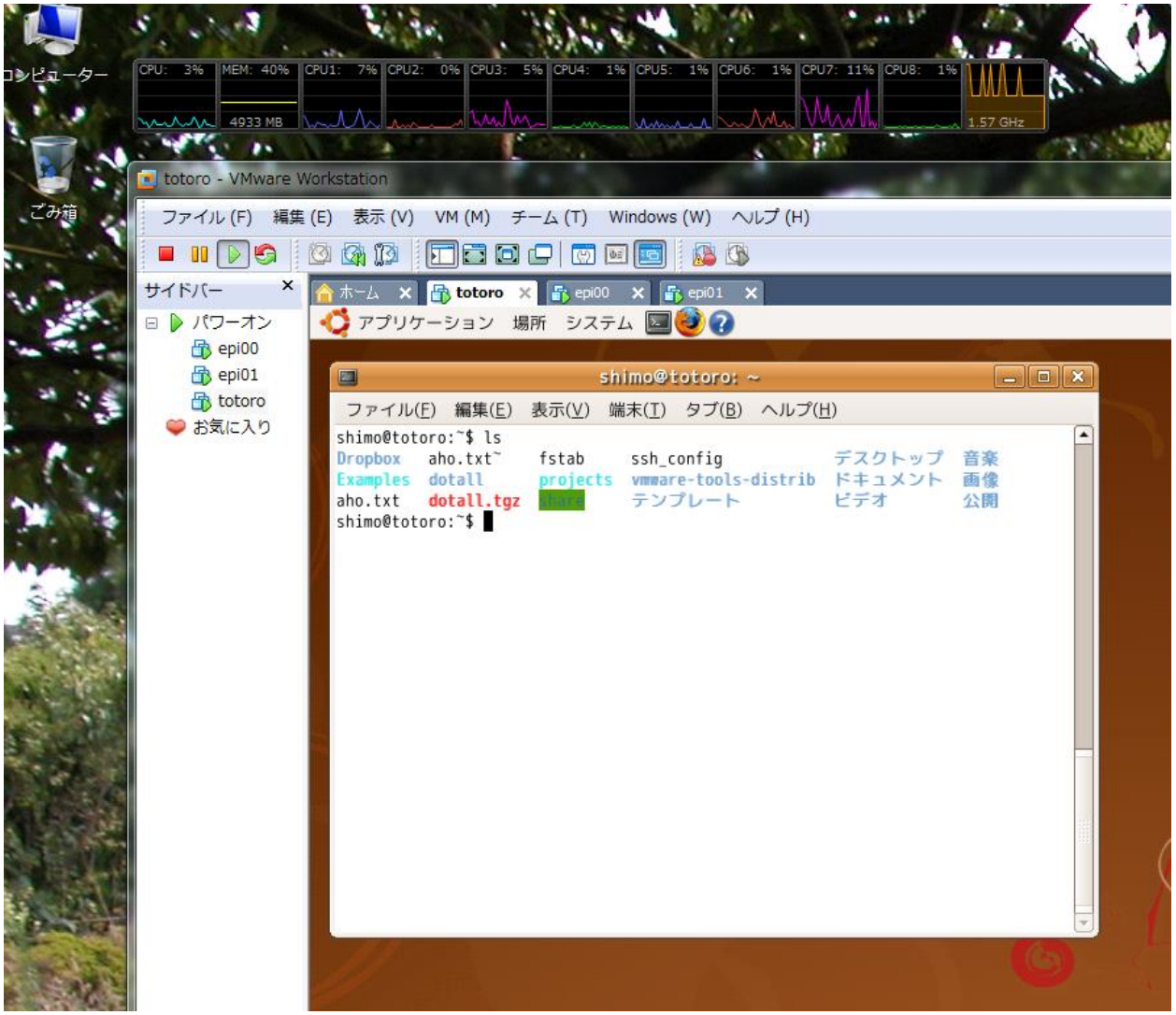
うえをみると，epi00で４個，epi01で４個のcpuが確保できたのがわかります．はじめはcpuを１個だけつかってpvclust（反復数=10)を実行です．時間をみると，9秒くらいですね．

```
Bootstrap (r = 1.3)... Done.
Bootstrap (r = 1.4)... Done.
> date()
[1] "Wed May 19 18:24:58 2010"
>
> postscript("test1.ps")
> plot(lung.pv)
> pvrect(lung.pv)
> dev.off()
null device
          1
>
> date()
[1] "Wed May 19 18:24:58 2010"
> lung.pv <- parPvclust(cl,lung,nboot=1000)
Multiscale bootstrap... Done.
> date()
[1] "Wed May 19 18:27:47 2010"
>
> postscript("test1.ps")
> plot(lung.pv)
> pvrect(lung.pv)
> dev.off()
null device
          1
>
> stopCluster(cl)
[1] 1
> quit(save="no")
> proc.time()
   user  system elapsed
  9.350   0.550 181.131
```

画面の領域の取り込み 日時: 2010/05/19 18:38

次に8cpuをつかった並列実行です．反復数=1000に増やしたので計算量はさっきの100倍です．計算時間は169秒なので169/9=18.8倍です．したがって，計算速度が100/(169/9)=5.3倍早くなったことになります．cpuを１個から8個にふやしたので8倍になってくれればうれしいですが，実際には多少下がることが多いです．なおcpuが１個のときの計算が8秒となっていますが誤差が1秒あります．もし並列化の効率を正確に測るならcpu=1のときの反復数=10をせめて100くらいにする必要があります．

実はepi00とepi01という２台の仮想マシンは １台のパソコンで動いている仮想マシンです（cpuも本当は１個でコア数が４つまりquad core ：quadは４ですよね．でも2倍してコア8としてあつかえるらしい）．見かけ上8cpuとしてあつかえるようなので，epi00は4cpu，epi01も4cpuの仮想マシンとしていますが，本当に8cpuあるわけではないです．なおここではvmware workstation 7という製品をつかっているので複数の仮想マシンが動かせます（忘れていましたが，本当はもう１個うごかしていて３個の仮想マシンが動いていました）．無料のvmware playerだと同時に１個しかパワーオンできないみたいです．vmware server （無料）なら複数の仮想マシンがパワーオンできるようですが，まだ試していません．



画面の領域の取り込み 日時: 2010/05/21 15:34

epi00でdmesgを実行してＣＰＵに関係する情報をみると

```
[    0.216371] CPU0: Intel(R) Core(TM) i7 CPU         960  @ 3.20GHz stepping 05
[    0.490750] CPU1: Intel(R) Core(TM) i7 CPU         960  @ 3.20GHz stepping 05
[    0.650269] CPU2: Intel(R) Core(TM) i7 CPU         960  @ 3.20GHz stepping 05
[    0.809787] CPU3: Intel(R) Core(TM) i7 CPU         960  @ 3.20GHz stepping 05
[    0.810051] Total of 4 processors activated (25654.12 BogoMIPS).
```

となっています．25654.12 BogoMIPSというのが，だいたいの計算速度を表します．

ちなみに，数日前に実行したtest1.Routのほうをみると下図のように115秒で実行されています．時間は115/9=12.8倍になったので，計算速度が100/(115/9)=7.8倍です．ほぼ８倍になっていて理想的です．

```
> date()
[1] "Mon May 17 15:28:22 2010"
> lung.pv <- parPvclust(cl,lung,nboot=1000)
Multiscale bootstrap... Done.
> date()
[1] "Mon May 17 15:30:17 2010"
```

画面の領域の取り込み 日時: 2010/05/19 18:47

このtest1.Rを実行したときは，epi00も4cpu, epi01も4cpuで今回と同じ設定なのですが，epi00は１台目のパソコン，epi01は２台目のパソコンで実行しました（ただしvmwareの優先順位をさげてCPUを半分だけつかう設定にしてありました．パソコンでメールを読んだり別の仕事の邪魔にならないように，これが標準の設定です）．それで仮想PCをつかう無駄がなくて，ほぼ理想的な結果でした．

計算がおわったら，最後にlamcleanを実行してください．

```
shimo@epi00:~/test20100517$ lamclean
shimo@epi00:~/test20100517$
```

画面の領域の取り込み 日時: 2010/05/19 18:54

これでepiでの並列Ｒの説明はおわりです.

番外編（その１）

本当はあとの番外編（その２）を最初にやったのだけど，その待ち時間にこの（その１）をやりました．．．

＊＊さっきの実験では１台のパソコン上でepi00,epi01,totoroという３台の仮想マシンが動作していた．totoroはアイドル状態だったけど，もしかしたら影響があるかもしれない．そこでtotoroをshutdownしてやりなおしてみる.

自宅からの作業ですが，実機がみえないせいで仮想マシンかどうかなんて全く分かりません．仮想マシンで動いてるtotoroのパワーオフもリモートから普通にできます.

さっきと同じように作業を進めます．epi00とepi01の２台の仮想マシンは１台のWindowsパソコン上で動いています.

```
shimo@epi00:~/test20100517$ lamclean
shimo@epi00:~/test20100517$ cat lamhosts2
epi00 cpu=4
epi01 cpu=4
shimo@epi00:~/test20100517$ lamboot -v lamhosts2

LAM 7.1.2/MPI 2 C++/ROMIO - Indiana University

n-1<7612> ssi:boot:base:linear: booting n0 (epi00)
n-1<7612> ssi:boot:base:linear: booting n1 (epi01)
n-1<7612> ssi:boot:base:linear: finished
shimo@epi00:~/test20100517$ R CMD BATCH test2b.R >& test2b.log &
[1] 7633
shimo@epi00:~/test20100517$
```

画面の領域の取り込み 日時: 2010/05/19 23:13

```
shimo@epi00: ~/test20100517
ファイル(F)  編集(E)  表示(V)  端末(T)  ヘルプ(H)
top - 23:02:51 up  8:31,  3 users,  load average: 3.76, 1.71, 0.65
Tasks: 150 total,   5 running, 145 sleeping,   0 stopped,   0 zombie
Cpu(s): 99.8%us,  0.2%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   4059876k total,   414312k used,  3645564k free,    23104k buffers
Swap:   905208k total,        0k used,   905208k free,   110168k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 7645 shimo     20   0 85360  29m 3812 R  100  0.7   2:37.46 R
 7642 shimo     20   0 85368  29m 3828 R  100  0.7   2:37.18 R
 7643 shimo     20   0 85360  29m 3812 R  100  0.7   2:37.51 R
 7644 shimo     20   0 86824  30m 3812 R  100  0.8   2:37.71 R
    1 root     20   0 23684 1984 1292 S    0  0.0   0:02.04 init
    2 root     20   0     0    0    0 S    0  0.0   0:00.01 kthreadd
    3 root     RT   0     0    0    0 S    0  0.0   0:00.18 migration/0
    4 root     20   0     0    0    0 S    0  0.0   0:00.00 ksoftirqd/0
    5 root     RT   0     0    0    0 S    0  0.0   0:00.00 watchdog/0
    6 root     RT   0     0    0    0 S    0  0.0   0:00.19 migration/1
```

画面の領域の取り込み 日時: 2010/05/19 23:13

```
> cl <- makeCluster(ncl)
Loading required package: Rmpi
        8 slaves are spawned successfully. 0 failed.
> lamhosts()
epi00 epi00 epi00 epi00 epi01 epi01 epi01 epi01
    0     1     2     3     4     5     6     7
>
> date()
[1] "Wed May 19 22:59:56 2010"
> lung.pv <- pvclust(lung,nboot=10)
Bootstrap (r = 0.5)... Done.
Bootstrap (r = 0.6)... Done.
Bootstrap (r = 0.7)... Done.
Bootstrap (r = 0.8)... Done.
Bootstrap (r = 0.9)... Done.
Bootstrap (r = 1.0)... Done.
Bootstrap (r = 1.1)... Done.
Bootstrap (r = 1.2)... Done.
Bootstrap (r = 1.3)... Done.
Bootstrap (r = 1.4)... Done.
> date()
[1] "Wed May 19 23:00:14 2010"
```

画面の領域の取り込み 日時: 2010/05/19 23:16

```
> date()
[1] "Wed May 19 23:00:14 2010"
> lung.pv <- parPvclust(cl,lung,nboot=1000)
Multiscale bootstrap... Done.
> date()
[1] "Wed May 19 23:03:01 2010"
>
```

画面の領域の取り込み 日時: 2010/05/19 23:17

結果は．．．cpu1個で8秒（さっきは9秒だけど，１秒差は測定誤差内）．cpu８個で167秒（さっきは169秒でたった２秒差）．したがって，アイドル中だったもう一台の仮想マシンtotoroの影響は実質的にほとんどなかったことになります．

＃＃＃＃　番外編（その２）

同じ計算を別のPCクラスターでやってみます．研究室で最初2003年6月頃に稼働させたものです（/etc/hostnameのタイムスタンプをみると，kumaは2003/06/02, fan08は2003/06/09)．当時の松岡研に在籍した山本君が下平研でも研究をすることになり，設定を全面的にやってくれました（感謝！）．20ノード(cpuが２個ずつ）あり，合計40cpuとして扱います． dmesgを実行して確認すると，

ホストノードのkumaとkabaは
CPU0: Intel(R) Xeon(TM) CPU 2.40GHz stepping 07
CPU1: Intel(R) Xeon(TM) CPU 2.40GHz stepping 07
Total of 2 processors activated (9555.14 BogoMIPS).

計算ノードfan00,...,fan17は
CPU0: AMD Athlon(tm) MP 2000+ stepping 02
CPU1: AMD Athlon(tm) Processor stepping 02
Total of 2 processors activated (6651.90 BogoMIPS).

となっています．epi00のときは
[0.810051] Total of 4 processors activated (25654.12 BogoMIPS).
だったので，fanに対するepiのコアあたりの速度比は(25654/4)/(6652/2)=1.9倍です．つまりlambootの指定で同じCPU数にした場合，epiクラスタのほうがkumaクラスタより２倍近く早いと予想できます．

並列Ｒを動かしてみます．まず計算ノードの空き状況をrupで調べます．

```
shimo@kuma:~/test20100517$ rup
kuma 20:53:19 up 160 days,  1:36,  1 user,  load average: 1.25, 1.06, 0.95
kabakaba: No route to host
fan00 20:36:47 up 205 days,  6:37,  0 users,  load average: 0.00, 0.00, 0.00
fan01 20:43:46 up 146 days,  3:21,  0 users,  load average: 0.00, 0.00, 0.00
fan02 20:34:01 up 205 days,  6:35,  0 users,  load average: 0.00, 0.00, 0.00
fan03 21:05:21 up 205 days,  6:38,  0 users,  load average: 0.00, 0.00, 0.00
fan04 20:25:03 up 205 days,  6:39,  0 users,  load average: 0.00, 0.00, 0.00
fan05 20:37:48 up 205 days,  6:39,  0 users,  load average: 0.00, 0.00, 0.00
fan06 20:35:22 up 205 days,  6:32,  0 users,  load average: 0.00, 0.00, 0.00
fan07 20:28:22 up 205 days,  6:38,  0 users,  load average: 0.00, 0.00, 0.00
fan08 20:37:21 up 205 days,  6:37,  0 users,  load average: 0.00, 0.00, 0.00
fan09 21:02:06 up 205 days,  6:37,  0 users,  load average: 0.00, 0.00, 0.00
fan10 21:05:42 up 205 days,  6:17,  0 users,  load average: 0.00, 0.00, 0.00
fan11 20:42:22 up 205 days,  6:38,  0 users,  load average: 0.00, 0.00, 0.00
fan12 20:44:56 up 205 days,  6:35,  0 users,  load average: 0.00, 0.00, 0.00
fan13 21:08:53 up 2 days,  3:13,  0 users,  load average: 0.00, 0.00, 0.00
fan14 21:11:47 up 205 days,  6:11,  0 users,  load average: 0.00, 0.00, 0.00
fan15 20:59:38 up 205 days,  6:34,  0 users,  load average: 0.00, 0.00, 0.00
fan16 20:46:21 up 205 days,  6:20,  0 users,  load average: 0.00, 0.00, 0.00
fan17 20:51:29 up 205 days,  6:31,  1 user,  load average: 0.00, 0.00, 0.00
shimo@kuma:~/test20100517$
```

画面の領域の取り込み 日時: 2010/05/19 20:53

空いてるみたいです（kabaは落ちてます）．fan08, fan09, fan10, fan11を各cpu=2として，合計8cpuでやってみます．

```
shimo@kuma:~/test20100517$ cat lamhosts3
fan08 cpu=2
fan09 cpu=2
fan10 cpu=2
fan11 cpu=2
shimo@kuma:~/test20100517$ rsh fan08
Last login: Wed May 19 20:40:16 2010 from kuma on ttyp0
Linux fan08 2.4.18 #1 SMP Thu Jun 5 12:13:58 JST 2003 i686 unknown

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
shimo@fan08:~$ cd test20100517
shimo@fan08:~/test20100517$ lamboot -v lamhosts3

LAM 7.1.1/MPI 2 C++/ROMIO - Indiana University

n-1<3639> ssi:boot:base:linear: booting n0 (fan08)
n-1<3639> ssi:boot:base:linear: booting n1 (fan09)
n-1<3639> ssi:boot:base:linear: booting n2 (fan10)
n-1<3639> ssi:boot:base:linear: booting n3 (fan11)
n-1<3639> ssi:boot:base:linear: finished
shimo@fan08:~/test20100517$
```

画面の領域の取り込み 日時: 2010/05/19 20:58

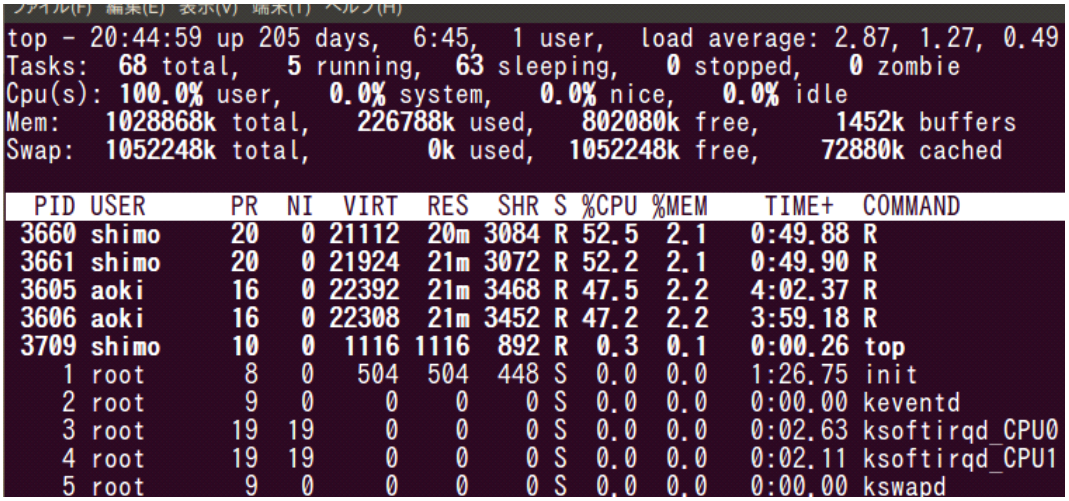
無事lambootができたので，Ｒプログラムを実行します．内容はtest2.Rと同一です．

R CMD BATCH test3.R >& test3.log &

```
shimo@fan08:~/test20100517$ R CMD BATCH test3.R >& test3.log &
[1] 3649
shimo@fan08:~/test20100517$
```

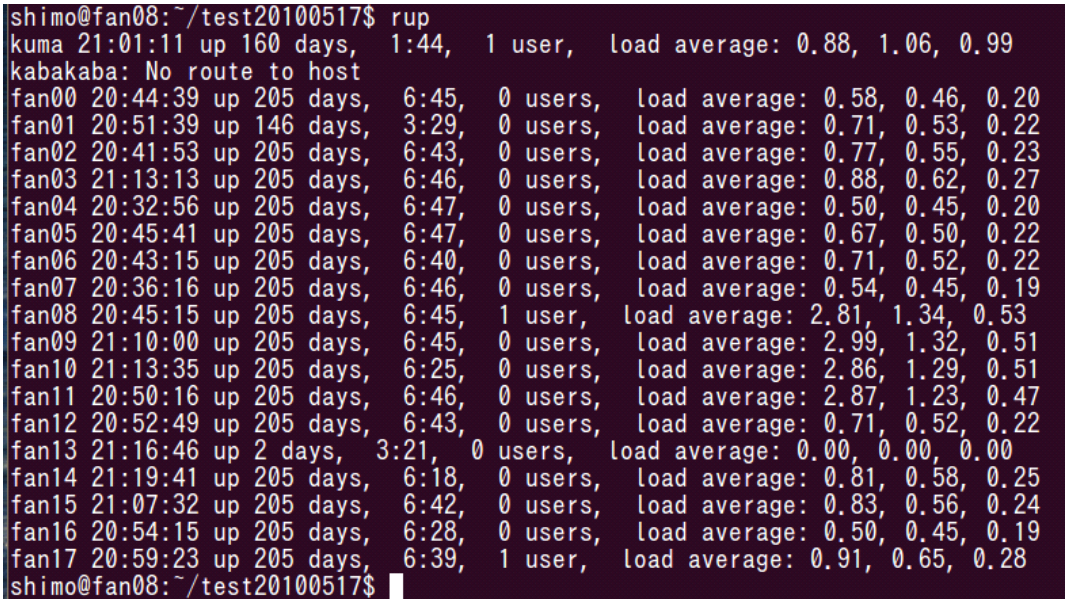
画面の領域の取り込み 日時: 2010/05/19 20:59

topコマンドでfan08の様子を見ます．



画面の領域の取り込み 日時: 2010/05/19 21:01

あれ，shimoのＲプロセスのほかに，aokiのＲプロセスが動いてる！
load averageも２を超えてる．rupで全体をみてみると，fan08, fan09, fan10, fan11のload averageが2をこえてる



画面の領域の取り込み 日時: 2010/05/19 21:01

ああ，aoki君も計算始めてしまったみたい．これでは計算速度が測定できません．というか，aoki君のほうが起動が一瞬はやかったじゃないか．私の方がaoki君の邪魔をしてるということです．一応，最後までやってみます．

```
> cl <- makeCluster(ncl)
Loading required package: Rmpi
        8 slaves are spawned successfully. 0 failed.
> lamhosts()
fan08 fan08 fan09 fan09 fan10 fan10 fan11 fan11
    0     1     2     3     4     5     6     7
>
> date()
[1] "Wed May 19 20:42:59 2010"
> lung.pv <- pvclust(lung,nboot=10)
Bootstrap (r = 0.5)... Done.
Bootstrap (r = 0.6)... Done.
Bootstrap (r = 0.7)... Done.
Bootstrap (r = 0.8)... Done.
Bootstrap (r = 0.9)... Done.
Bootstrap (r = 1.0)... Done.
Bootstrap (r = 1.1)... Done.
Bootstrap (r = 1.2)... Done.
Bootstrap (r = 1.3)... Done.
Bootstrap (r = 1.4)... Done.
> date()
[1] "Wed May 19 20:43:45 2010"
>
> postscript("test1.ps")
> plot(lung.pv)
```

画面の領域の取り込み 日時: 2010/05/19 21:11

```
>
> date()
[1] "Wed May 19 20:43:46 2010"
> lung.pv <- parPvclust(cl,lung,nboot=1000)
Multiscale bootstrap... Done.
> date()
[1] "Wed May 19 20:54:51 2010"
>
```

画面の領域の取り込み 日時: 2010/05/19 21:12

ＣＰＵが１個だと46秒， CPU８個だと665秒ですが，二人が計算を実行しているので，本来より遅くなったはずです．どのくらい影響があったか，はっきりしないので，もう一度やり直します．

朝起きてみると，aoki君の計算は終わっていました．並列Ｒを実行してみます．

```
ファイル(F)  編集(E)  表示(V)  端末(T)  ヘルプ(H)
shimo@fan08:~/test20100517$ rup
kuma 06:59:05 up 160 days, 11:42,  2 users,  load average: 0.00, 0.00, 0.00
kabakaba: No route to host
fan00 06:42:33 up 205 days, 16:43,  0 users,  load average: 0.00, 0.00, 0.00
fan01 06:49:33 up 146 days, 13:26,  0 users,  load average: 0.00, 0.00, 0.00
fan02 06:39:47 up 205 days, 16:41,  0 users,  load average: 0.00, 0.00, 0.00
fan03 07:11:07 up 205 days, 16:43,  0 users,  load average: 0.00, 0.00, 0.00
fan04 06:30:49 up 205 days, 16:44,  0 users,  load average: 0.00, 0.00, 0.00
fan05 06:43:34 up 205 days, 16:44,  0 users,  load average: 0.00, 0.00, 0.00
fan06 06:41:09 up 205 days, 16:38,  0 users,  load average: 0.00, 0.00, 0.00
fan07 06:34:09 up 205 days, 16:44,  0 users,  load average: 0.00, 0.00, 0.00
fan08 06:43:08 up 205 days, 16:43,  1 user,  load average: 0.00, 0.00, 0.00
fan09 07:07:53 up 205 days, 16:43,  0 users,  load average: 0.00, 0.00, 0.00
fan10 07:11:28 up 205 days, 16:23,  0 users,  load average: 0.00, 0.00, 0.00
fan11 06:48:09 up 205 days, 16:44,  0 users,  load average: 0.00, 0.00, 0.00
fan12 06:50:42 up 205 days, 16:40,  0 users,  load average: 0.00, 0.00, 0.00
fan13 07:14:39 up 2 days, 13:18,  0 users,  load average: 0.00, 0.00, 0.00
fan14 07:17:33 up 205 days, 16:16,  0 users,  load average: 0.00, 0.00, 0.00
fan15 07:05:25 up 205 days, 16:39,  0 users,  load average: 0.00, 0.00, 0.00
fan16 06:52:08 up 205 days, 16:25,  0 users,  load average: 0.00, 0.00, 0.00
fan17 06:57:16 up 205 days, 16:37,  1 user,  load average: 0.00, 0.00, 0.00
shimo@fan08:~/test20100517$
```

画面の領域の取り込み 日時: 2010/05/20 6:59

```
shimo@fan08:~/test20100517$ lamboot -v lamhosts3

LAM 7.1.1/MPI 2 C++/ROMIO - Indiana University

n-1<3970> ssi:boot:base:linear: booting n0 (fan08)
n-1<3970> ssi:boot:base:linear: booting n1 (fan09)
n-1<3970> ssi:boot:base:linear: booting n2 (fan10)
n-1<3970> ssi:boot:base:linear: booting n3 (fan11)
n-1<3970> ssi:boot:base:linear: finished
shimo@fan08:~/test20100517$ R CMD BATCH test3.R >& test3.log &
[1] 3980
shimo@fan08:~/test20100517$
```

画面の領域の取り込み 日時: 2010/05/20 7:00

```
> cl <- makeCluster(ncl)
Loading required package: Rmpi
        8 slaves are spawned successfully. 0 failed.
> lamhosts()
fan08 fan08 fan09 fan09 fan10 fan10 fan11 fan11
    0     1     2     3     4     5     6     7
>
> date()
[1] "Thu May 20 06:44:22 2010"
> lung.pv <- pvclust(lung,nboot=10)
Bootstrap (r = 0.5)... Done.
Bootstrap (r = 0.6)... Done.
Bootstrap (r = 0.7)... Done.
Bootstrap (r = 0.8)... Done.
Bootstrap (r = 0.9)... Done.
Bootstrap (r = 1.0)... Done.
Bootstrap (r = 1.1)... Done.
Bootstrap (r = 1.2)... Done.
Bootstrap (r = 1.3)... Done.
Bootstrap (r = 1.4)... Done.
> date()
[1] "Thu May 20 06:44:56 2010"
>
```

画面の領域の取り込み 日時: 2010/05/20 7:18

```
> date()
[1] "Thu May 20 06:44:56 2010"
> lung.pv <- parPvclust(cl,lung,nboot=1000)
Multiscale bootstrap... Done.
> date()
[1] "Thu May 20 06:51:52 2010"
>
```

画面の領域の取り込み 日時: 2010/05/20 7:19

CPU１個で34秒，CPU8個で416秒でした．CPUを１個から８個に増やして計算速度が100/(416/34)=8.2倍になったことがわかります．多少の誤差の範囲で，ほぼ理想的に効率が上がっています．

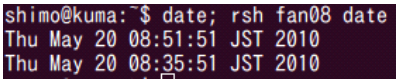
kumaクラスタとepiクラスタの計算速度を比較します．cpu８個の計算時間はfan08,...,fan11を使ったkumaクラスターは416秒．これにたいしてepi00+epi01(実はパソコン１台)でやったときは169秒だったので，同じＣＰＵ数をつかえば416/169=2.5倍くらいkumaよりepiのほうが速いことになります．

これは同じＣＰＵ数で比較した場合なので，もし40cpu全部使えばkumaは416*8/40=83.2秒くらいで計算が終わるはずなので，169/83.2=2倍くらい，epi00+epi01(実はパソコン１台）よりkuma全体のほうが速いことになります．つまり40cpuの古いクラスターは現在のパソコンの２台程度の能力です．当時の40cpuのクラスターと現在のパソコン２台の価格比は20倍くらいです．したがって，kumaからepiに乗り換えることによってコストパフォーマンスは２０倍程度向上します．

それに燃費(電力消費）とか，おき場所と冷房の問題とか，管理コストとか，いろんな問題を解決します．

kumaクラスターは，今年中に総入れ替え(再インストール）したいと思います．システム設定の練習用とか，まだそういう意味はあると思う．学部生，修士の学生はkumaクラスターを利用はしても，システム管理をやる人は皆無になってしまいました．vmwareの仮想PC上でクラスタを構成すれば，もっと身近になって，管理とかにも少し手を出す学生が出てくるかもしれない．そういう教育上の意義もあると思います．

追記：タイムスタンプがおかしいので気づいたがfanはntpとか時計合わせの設定してなかったみたい．kumaとfan08で比べると



画面の領域の取り込み 日時: 2010/05/20 8:54

となってます.