Parallel DelPhi95 v5.1 User Guide

Parallel DelPhi95 (pdelphi95) is a new variation of the well-known DelPhi program software package. It utilizes Message Passing Interface (MPI) library to achieve parallel computing on multiple CPUs and produce numerical solution of the Poisson-Boltzmann Equation (PBE). The achieved parallel solution tracks the serial solution obtained by DelPhi95 to double precision. This manual provides the users how to compile and run pdelphi95 on individual environment. Additional information about DelPhi program can be found at http://compbio.clemson.edu/delphi.php.

Authors:

Parallel DelPhi95 package is developed and maintained by Chuan Li from DelPhi team:

Email: chuanli@clemson.edu

References:

The following references should be quoted if the use of the Parallel DelPhi95 package results to a publication.

- Li C, Petukh M, Li L, Alexov E, Continuous development of schemes for parallel computing of the electrostatics in biological systems: Implementation in DelPhi, J Comput Chem. 2013 Jun 4.
- Li C, Li L, Zhang J, Alexov E, Highly efficient and exact method for parallelization of grid-based algorithms and its implementation in DelPhi, J Comput Chem. 2012 Sep 15;33(24):1960-6.

Table of Contents

1. PACKAGE CONTENT

```
/pdelphi95 --- makefile (makefile template to compile and run the program)
           --- mkpbs (bash script to generate a running directory for the examples
                      included in the folder of /examples)
           --- PBStemplate (template to submit jobs using PBS)
           --- /src (source code)
           --- /examples
               --- /1C54 (protein 1C54)
               --- /1VSZ (protein 1VSZ)
               --- /3KIC  (protein 3KIC)
               --- /3LZM (protein 3LZM)
               --- /barstar6 (protein barstar with scale = 6.0)
               --- /example4 (example4 from our website)
               --- /mmobj (a simple sample with multiple media)
               --- /robot (Clemson robot created by ProNOI)
```

2. ENVIRONMENT REQUIREMENT

Unlike the regular DelPhi program which has multiple versions that users can compile and run on various operating system (Windows, Linux and Mac OS), pdelphi95 is intended to be compiled and run on clusters which are equipped with UNIX/Linux operating system and multiple computer nodes to achieve the best performance.

In order to successfully compile the source code of pdelphi95, the following software packages are essentially required:

- GNU Compiler gfortran version 4.4.5 20110214 or later
- mpich2 version 1.4/openmpi version 1.4.3 or later

In addition to above packages, it is recommended to have the following job queening system installed on the computing cluster in order to submit and run computational jobs:

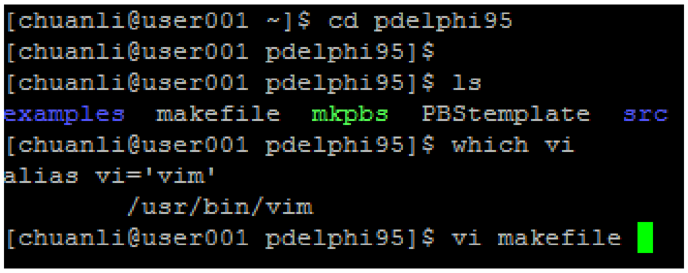
- PBS Professional 11.1

The source code has been extensively tested on the computing cluster Palmetto at Clemson University, which is equipped with above software packages and Myrinet 10G network interconnect (http://desktop2petascale.org/resources/175/download/Palmetto.Cluster.Users.Guide.pdf).

3. COMPILE THE PROGRAM

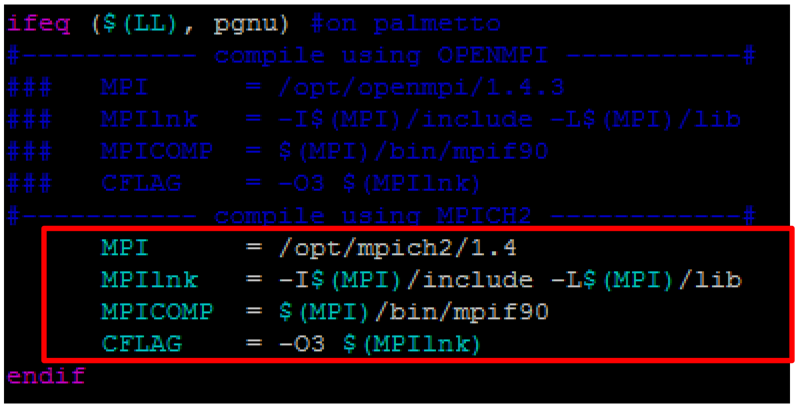
It is suggested to use the included makefile to compile and run the program. However, users need to make necessary changes in the makefile if they decide to compile and run the program on their own clusters.

Users can modify the makefile using editor vi/vim.



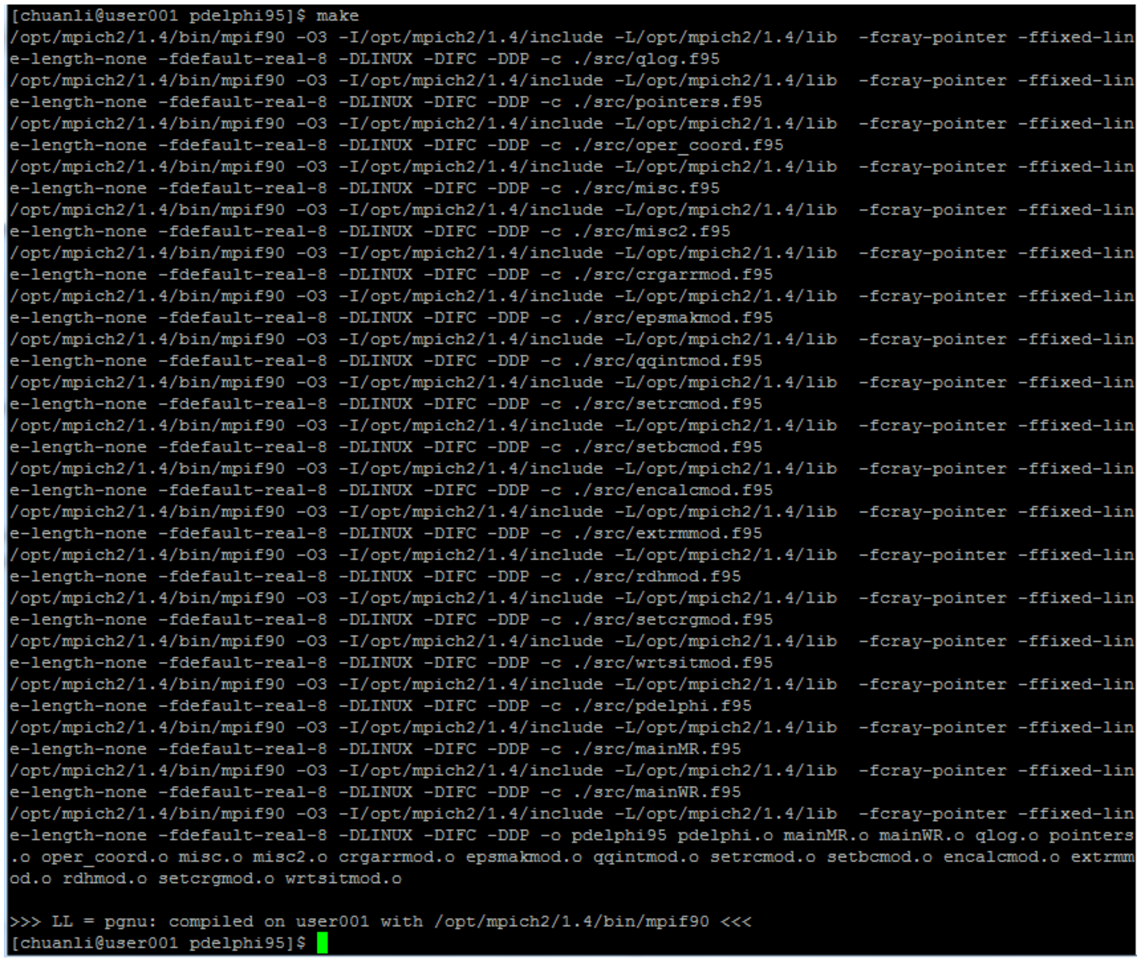
```
[chuanli@user001 ~]$ cd pdelphi95
[chuanli@user001 pdelphi95]$
[chuanli@user001 pdelphi95]$ ls
examples  makefile  mkpbs  PBStemplate  src
[chuanli@user001 pdelphi95]$ which vi
alias vi='vim'
        /usr/bin/vim
[chuanli@user001 pdelphi95]$ vi makefile
```

The first item users need to modify in the makefile is the variable “LL”. “LL = pgnu” is set by default and represents the path where mpich2 MPI library is installed on Palmetto. Users can either modify the lines in red box or give new values to LL to specify correct path of the MPI library in their own environment.



```
ifeq ($(LL), pgnu) #on palmetto
#------------ compile using OPENMPI ------------#
###   MPI      = /opt/openmpi/1.4.3
###   MPIlnk   = -I$(MPI)/include -L$(MPI)/lib
###   MPICOMP  = $(MPI)/bin/mpif90
###   CFLAG    = -O3 $(MPIlnk)
#------------ compile using MPICH2 -------------#
      MPI      = /opt/mpich2/1.4
      MPIlnk   = -I$(MPI)/include -L$(MPI)/lib
      MPICOMP  = $(MPI)/bin/mpif90
      CFLAG    = -O3 $(MPIlnk)
endif
```

While mpich2 is configured to use GNU compiler, users now can type “make” to start compiling the source code and generate the executable “pdelphi95” (set by the variable “CODE” in makefile by default).

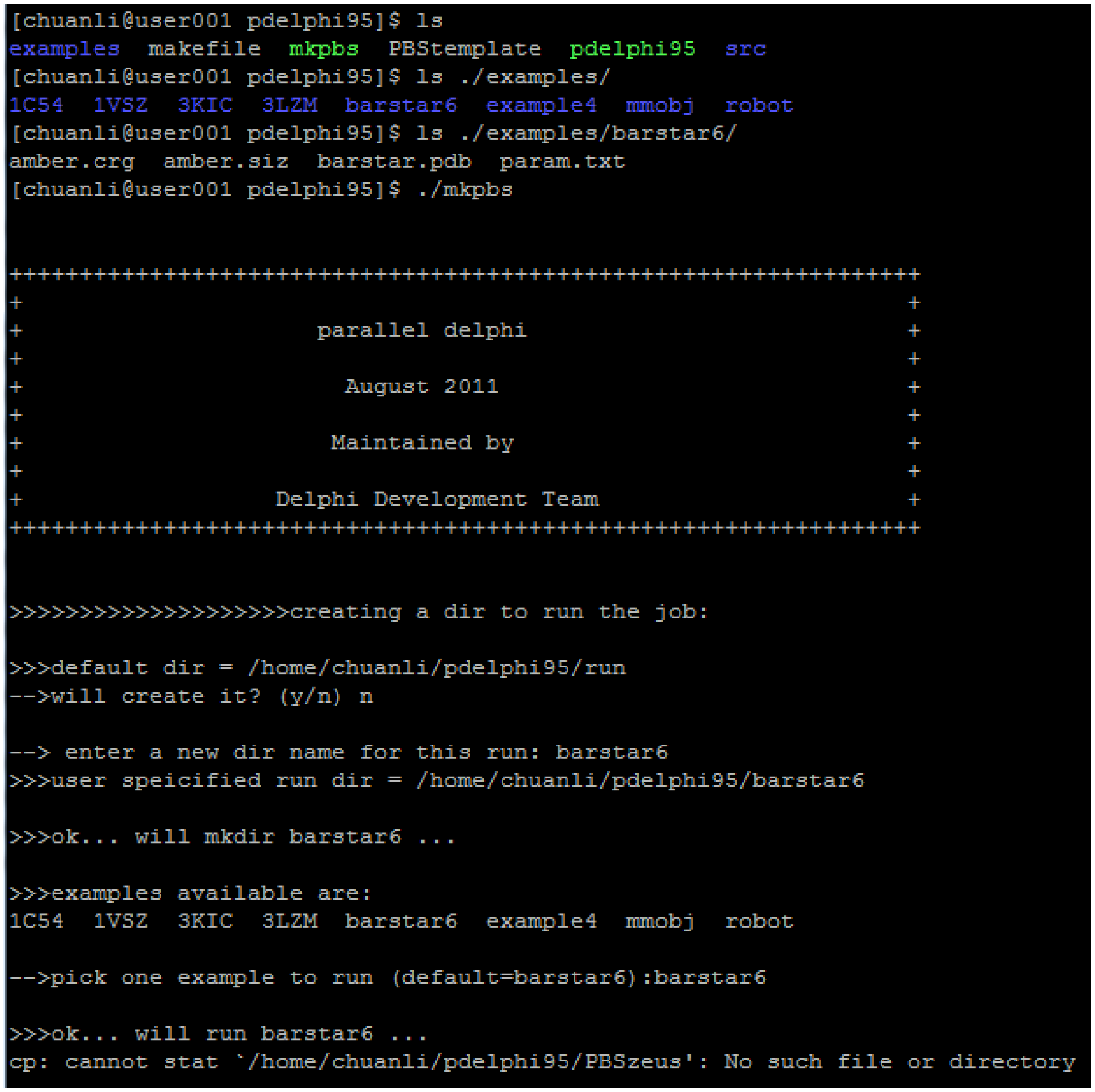


```
[chuanli@user001 pdelphi95]$ make
/opt/mpich2/1.4/bin/mpif90 -O3 -I/opt/mpich2/1.4/include -L/opt/mpich2/1.4/lib  -fcray-pointer -ffixed-lin
e-length-none -fdefault-real-8 -DLINUX -DIFC -DDP -c ./src/qlog.f95
/opt/mpich2/1.4/bin/mpif90 -O3 -I/opt/mpich2/1.4/include -L/opt/mpich2/1.4/lib  -fcray-pointer -ffixed-lin
e-length-none -fdefault-real-8 -DLINUX -DIFC -DDP -c ./src/pointers.f95
/opt/mpich2/1.4/bin/mpif90 -O3 -I/opt/mpich2/1.4/include -L/opt/mpich2/1.4/lib  -fcray-pointer -ffixed-lin
e-length-none -fdefault-real-8 -DLINUX -DIFC -DDP -c ./src/oper_coord.f95
/opt/mpich2/1.4/bin/mpif90 -O3 -I/opt/mpich2/1.4/include -L/opt/mpich2/1.4/lib  -fcray-pointer -ffixed-lin
e-length-none -fdefault-real-8 -DLINUX -DIFC -DDP -c ./src/misc.f95
/opt/mpich2/1.4/bin/mpif90 -O3 -I/opt/mpich2/1.4/include -L/opt/mpich2/1.4/lib  -fcray-pointer -ffixed-lin
e-length-none -fdefault-real-8 -DLINUX -DIFC -DDP -c ./src/misc2.f95
/opt/mpich2/1.4/bin/mpif90 -O3 -I/opt/mpich2/1.4/include -L/opt/mpich2/1.4/lib  -fcray-pointer -ffixed-lin
e-length-none -fdefault-real-8 -DLINUX -DIFC -DDP -c ./src/crgarrmod.f95
/opt/mpich2/1.4/bin/mpif90 -O3 -I/opt/mpich2/1.4/include -L/opt/mpich2/1.4/lib  -fcray-pointer -ffixed-lin
e-length-none -fdefault-real-8 -DLINUX -DIFC -DDP -c ./src/epsmakmod.f95
/opt/mpich2/1.4/bin/mpif90 -O3 -I/opt/mpich2/1.4/include -L/opt/mpich2/1.4/lib  -fcray-pointer -ffixed-lin
e-length-none -fdefault-real-8 -DLINUX -DIFC -DDP -c ./src/qqintmod.f95
/opt/mpich2/1.4/bin/mpif90 -O3 -I/opt/mpich2/1.4/include -L/opt/mpich2/1.4/lib  -fcray-pointer -ffixed-lin
e-length-none -fdefault-real-8 -DLINUX -DIFC -DDP -c ./src/setrcmod.f95
/opt/mpich2/1.4/bin/mpif90 -O3 -I/opt/mpich2/1.4/include -L/opt/mpich2/1.4/lib  -fcray-pointer -ffixed-lin
e-length-none -fdefault-real-8 -DLINUX -DIFC -DDP -c ./src/setbcmod.f95
/opt/mpich2/1.4/bin/mpif90 -O3 -I/opt/mpich2/1.4/include -L/opt/mpich2/1.4/lib  -fcray-pointer -ffixed-lin
e-length-none -fdefault-real-8 -DLINUX -DIFC -DDP -c ./src/encalcmod.f95
/opt/mpich2/1.4/bin/mpif90 -O3 -I/opt/mpich2/1.4/include -L/opt/mpich2/1.4/lib  -fcray-pointer -ffixed-lin
e-length-none -fdefault-real-8 -DLINUX -DIFC -DDP -c ./src/extrmmod.f95
/opt/mpich2/1.4/bin/mpif90 -O3 -I/opt/mpich2/1.4/include -L/opt/mpich2/1.4/lib  -fcray-pointer -ffixed-lin
e-length-none -fdefault-real-8 -DLINUX -DIFC -DDP -c ./src/rdhmod.f95
/opt/mpich2/1.4/bin/mpif90 -O3 -I/opt/mpich2/1.4/include -L/opt/mpich2/1.4/lib  -fcray-pointer -ffixed-lin
e-length-none -fdefault-real-8 -DLINUX -DIFC -DDP -c ./src/setcrgmod.f95
/opt/mpich2/1.4/bin/mpif90 -O3 -I/opt/mpich2/1.4/include -L/opt/mpich2/1.4/lib  -fcray-pointer -ffixed-lin
e-length-none -fdefault-real-8 -DLINUX -DIFC -DDP -c ./src/wrtsitmod.f95
/opt/mpich2/1.4/bin/mpif90 -O3 -I/opt/mpich2/1.4/include -L/opt/mpich2/1.4/lib  -fcray-pointer -ffixed-lin
e-length-none -fdefault-real-8 -DLINUX -DIFC -DDP -c ./src/pdelphi.f95
/opt/mpich2/1.4/bin/mpif90 -O3 -I/opt/mpich2/1.4/include -L/opt/mpich2/1.4/lib  -fcray-pointer -ffixed-lin
e-length-none -fdefault-real-8 -DLINUX -DIFC -DDP -c ./src/mainMR.f95
/opt/mpich2/1.4/bin/mpif90 -O3 -I/opt/mpich2/1.4/include -L/opt/mpich2/1.4/lib  -fcray-pointer -ffixed-lin
e-length-none -fdefault-real-8 -DLINUX -DIFC -DDP -c ./src/mainWR.f95
/opt/mpich2/1.4/bin/mpif90 -O3 -I/opt/mpich2/1.4/include -L/opt/mpich2/1.4/lib  -fcray-pointer -ffixed-lin
e-length-none -fdefault-real-8 -DLINUX -DIFC -DDP -o pdelphi95 pdelphi.o mainMR.o mainWR.o qlog.o pointers
.o oper_coord.o misc.o misc2.o crgarrmod.o epsmakmod.o qqintmod.o setrcmod.o setbcmod.o encalcmod.o extrmm
od.o rdhmod.o setcrgmod.o wrtsitmod.o

>>> LL = pgnu: compiled on user001 with /opt/mpich2/1.4/bin/mpif90 <<<
[chuanli@user001 pdelphi95]$
```

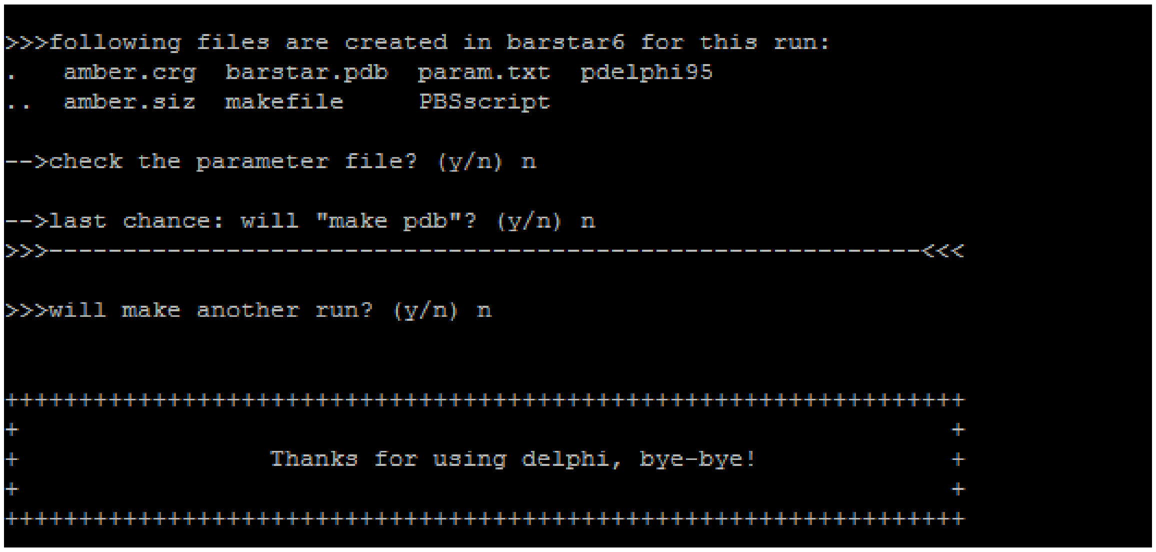
A batch of objective files (*.o) and an executable file (pdelphi95) are generated in the home directory after successfully compiling the source code. The executable is the only one file useful for most users. Other generated objective files can be removed easily by typing “make clean”. Typing “make swipe” removes all generated files for a run.

4. RUN THE PROGRAM

Taking bastar6 included in the folder of /examples as an instance, we demonstrate how to make a new run using the included bash script *mkpbs* and PBS script template *PBStemplate* to make a new run using PBS queening system on the cluster *Palmetto*.



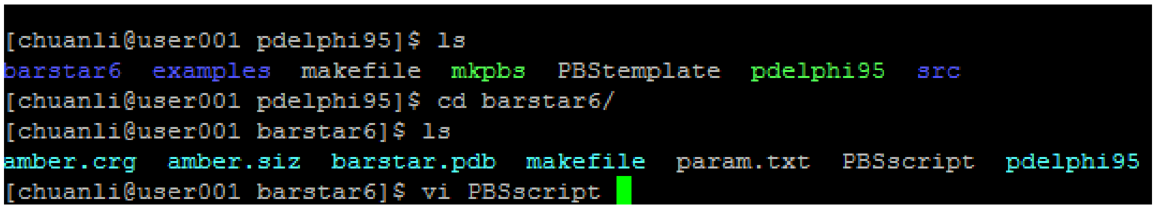
```
[chuanli@user001 pdelphi95]$ ls
examples  makefile  mkpbs  PBStemplate  pdelphi95  src
[chuanli@user001 pdelphi95]$ ls ./examples/
1C54  1VSZ  3KIC  3LZM  barstar6  example4  mmobj  robot
[chuanli@user001 pdelphi95]$ ls ./examples/barstar6/
amber.crg  amber.siz  barstar.pdb  param.txt
[chuanli@user001 pdelphi95]$ ./mkpbs


+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+                                                               +
+                     parallel delphi                           +
+                                                               +
+                       August 2011                             +
+                                                               +
+                      Maintained by                            +
+                                                               +
+                  Delphi Development Team                      +
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


>>>>>>>>>>>>>>>>>>>>creating a dir to run the job:

>>>default dir = /home/chuanli/pdelphi95/run
-->will create it? (y/n) n

--> enter a new dir name for this run: barstar6
>>>user speicified run dir = /home/chuanli/pdelphi95/barstar6

>>>ok... will mkdir barstar6 ...

>>>examples available are:
1C54  1VSZ  3KIC  3LZM  barstar6  example4  mmobj  robot

-->pick one example to run (default=barstar6):barstar6

>>>ok... will run barstar6 ...
cp: cannot stat `/home/chuanli/pdelphi95/PBSzeus': No such file or directory
```



```
>>>following files are created in barstar6 for this run:
.   amber.crg  barstar.pdb  param.txt  pdelphi95
..  amber.siz  makefile     PBSscript

-->check the parameter file? (y/n) n

-->last chance: will "make pdb"? (y/n) n
>>>-----------------------------------------------------------<<<

>>>will make another run? (y/n) n


+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+                                                               +
+                 Thanks for using delphi, bye-bye!             +
+                                                               +
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

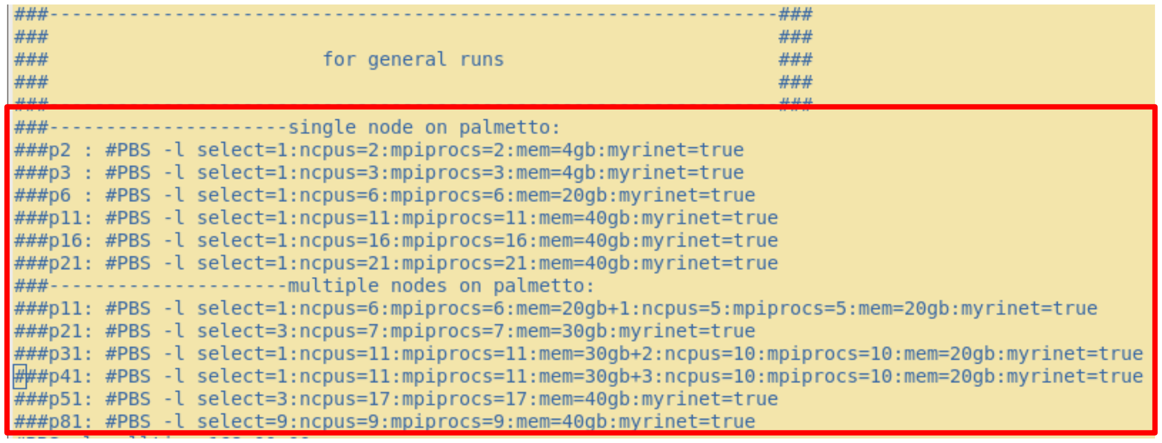
A new directory /barstar6 is created with appropriate symbolic links created and files copied in this directory for a new run of the example barstar6 after running the bash script *mkpbs*.

Entering the directory and editing the PBS script *PBSscript* before submitting a new job to PBS.



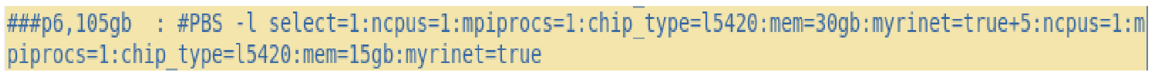
```
[chuanli@user001 pdelphi95]$ ls
barstar6  examples  makefile  mkpbs  PBStemplate  pdelphi95  src
[chuanli@user001 pdelphi95]$ cd barstar6/
[chuanli@user001 barstar6]$ ls
amber.crg  amber.siz  barstar.pdb  makefile  param.txt  PBSscript  pdelphi95
[chuanli@user001 barstar6]$ vi PBSscript
```

The computing nodes on Palmetto cluster have various configurations (see the Palmetto cluster user guide at above link for more detail) which give the users great flexibility to specify where your PBS jobs are submitted to. For most users who do not care about which nodes to be used, you can simply pick one line in the below red box and let PBS scheduler decides which nodes to use.



```
###---------------------------------------------------------------###
###                                                               ###
###                        for general runs                       ###
###                                                               ###
###                                                               ###
###--------------------single node on palmetto:
###p2 : #PBS -l select=1:ncpus=2:mpiprocs=2:mem=4gb:myrinet=true
###p3 : #PBS -l select=1:ncpus=3:mpiprocs=3:mem=4gb:myrinet=true
###p6 : #PBS -l select=1:ncpus=6:mpiprocs=6:mem=20gb:myrinet=true
###p11: #PBS -l select=1:ncpus=11:mpiprocs=11:mem=40gb:myrinet=true
###p16: #PBS -l select=1:ncpus=16:mpiprocs=16:mem=40gb:myrinet=true
###p21: #PBS -l select=1:ncpus=21:mpiprocs=21:mem=40gb:myrinet=true
###--------------------multiple nodes on palmetto:
###p11: #PBS -l select=1:ncpus=6:mpiprocs=6:mem=20gb+1:ncpus=5:mpiprocs=5:mem=20gb:myrinet=true
###p21: #PBS -l select=3:ncpus=7:mpiprocs=7:mem=30gb:myrinet=true
###p31: #PBS -l select=1:ncpus=11:mpiprocs=11:mem=30gb+2:ncpus=10:mpiprocs=10:mem=20gb:myrinet=true
###p41: #PBS -l select=1:ncpus=11:mpiprocs=11:mem=30gb+3:ncpus=10:mpiprocs=10:mem=20gb:myrinet=true
###p51: #PBS -l select=3:ncpus=17:mpiprocs=17:mem=40gb:myrinet=true
###p81: #PBS -l select=9:ncpus=9:mpiprocs=9:mem=40gb:myrinet=true
```

PBSscript also allows advanced users to submit the PBS jobs to a particular group of nodes. For instance, we can choose to use 6 nodes of the same chip type (Intel Xeon L5420) and request 30 GB memory on one node serving as the master node and 15 GB memory on the rest 5 nodes serving as the slave nodes by the following line

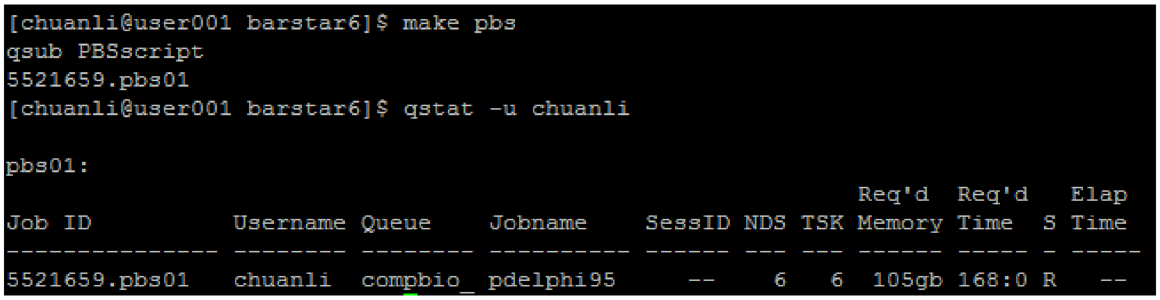


```
###p6,105gb  : #PBS -l select=1:ncpus=1:mpiprocs=1:chip_type=l5420:mem=30gb:myrinet=true+5:ncpus=1:m
piprocs=1:chip_type=l5420:mem=15gb:myrinet=true
```

After the decision is made, users can press enter and make a new line to validate the selection.

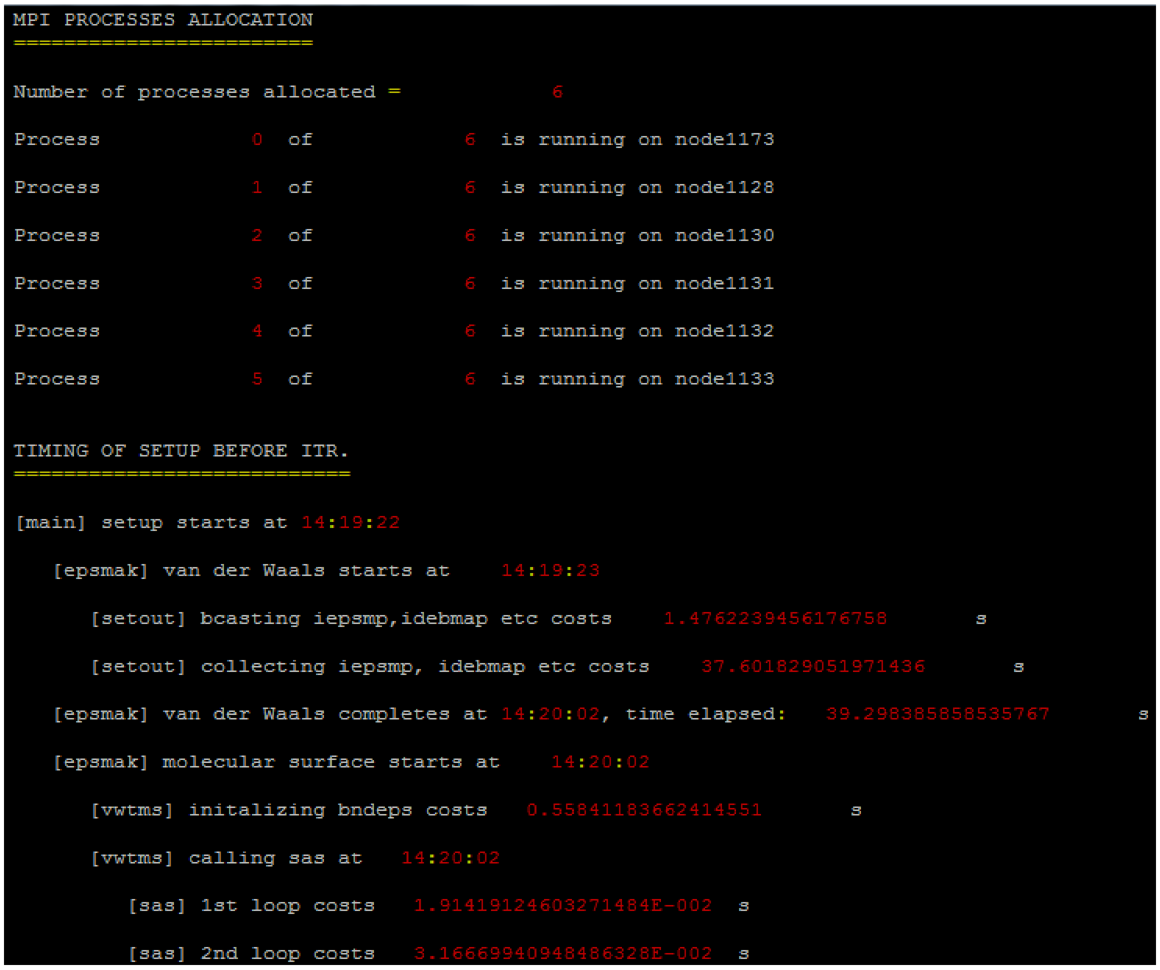
```
###p6,105gb  :
#PBS -l select=1:ncpus=1:mpiprocs=1:chip_type=l5420:mem=30gb:myrinet=true+5:ncpus=1:mpiprocs=1:chip_
type=l5420:mem=15gb:myrinet=true
```

Save the changes made in the PBSscript file and submit the job to PBS by typing "make pbs".



```
[chuanli@user001 barstar6]$ make pbs
qsub PBSscript
5521659.pbs01
[chuanli@user001 barstar6]$ qstat -u chuanli

pbs01:
                                                            Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
5521659.pbs01   chuanli  compbio_ pdelphi95    --     6   6  105gb 168:0 R   --
```

Two log files, out95.run and time.run, will be generated after the job is finished. The out95.run is of the same as the regular DelPhi output file, while the time.run contains detailed information about the parallel computing, such as which nodes are used and how much time cost by major subroutines.

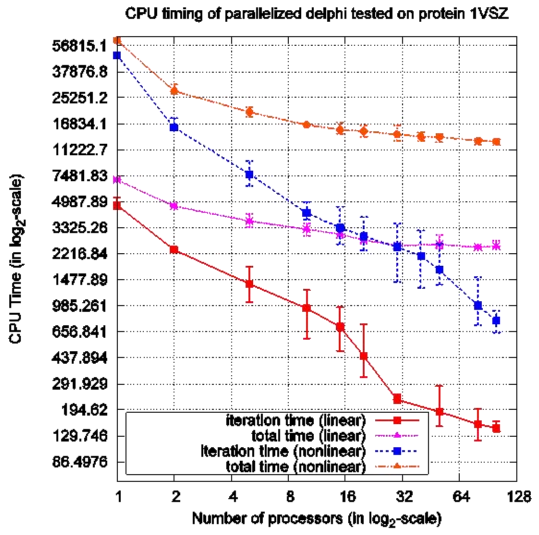


```
MPI PROCESSES ALLOCATION
========================

Number of processes allocated =            6

Process            0  of           6  is running on node1173

Process            1  of           6  is running on node1128

Process            2  of           6  is running on node1130

Process            3  of           6  is running on node1131

Process            4  of           6  is running on node1132

Process            5  of           6  is running on node1133


TIMING OF SETUP BEFORE ITR.
===========================

[main] setup starts at 14:19:22

   [epsmak] van der Waals starts at    14:19:23

      [setout] bcasting iepsmp,idebmap etc costs    1.4762239456176758       s

      [setout] collecting iepsmp, idebmap etc costs    37.601829051971436       s

   [epsmak] van der Waals completes at 14:20:02, time elapsed:   39.298385858535767       s

   [epsmak] molecular surface starts at    14:20:02

      [vwtms] initalizing bndeps costs   0.55841183662414551       s

      [vwtms] calling sas at   14:20:02

         [sas] 1st loop costs   1.91419124603271484E-002  s

         [sas] 2nd loop costs   3.16669940948486328E-002  s
```

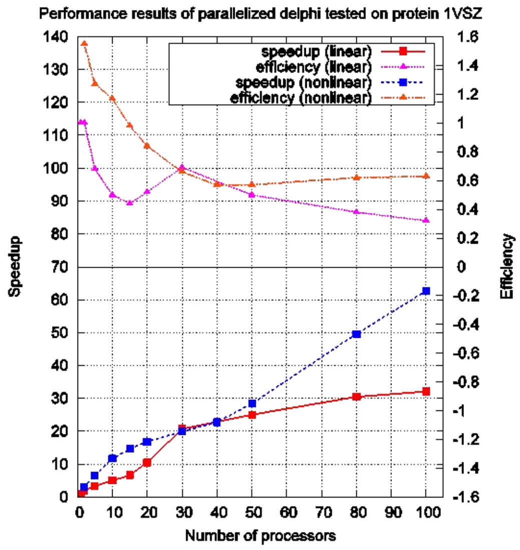
5. SAMPLE RESULTS OBTAINED BY THE PROGRAM

We will present the results obtained on real large proteins by running pdelphi95 in this section. The results have been reported in the reference papers mentioned above.

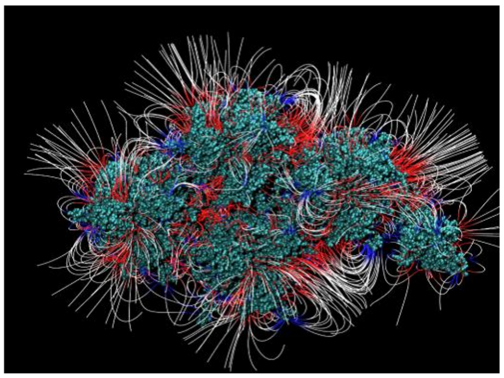
The first series of experiments, require solving linear and nonlinear PB equations, were performed on a fraction of the protein of human adenovirus 1VSZ downloaded from the Protein Data Bank (PDB) and protonated by TINKER. Amber force field was used. Scale = 2.0 and 70% filling of the box domain were set in the parameter file resulting in the dimensions of the box domain $\approx 407\,\text{Å}\times 407\,\text{Å}\times 407\,\text{Å}$ and $815\times 815\times 815$ mesh points in total. The CPU time achieved by solving the linear and nonlinear PBE as a function of increasing number of processors is shown in Figure 1(a) with vertical bars indicating variations of five runs. To compare their performance, log-scale plots of speedup and efficiency are shown in Figure 1(b). The resulting potential and electrostatic field are plotted by VMD and demonstrated in Figures 1(c) and 1(d).



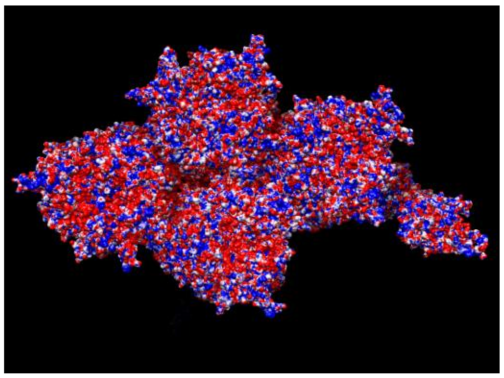
(a)



(b)



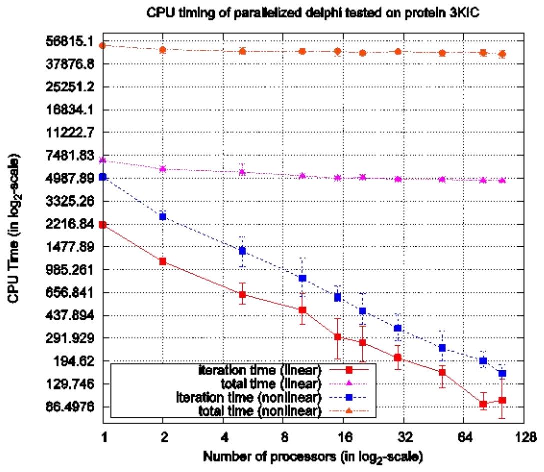
(c)



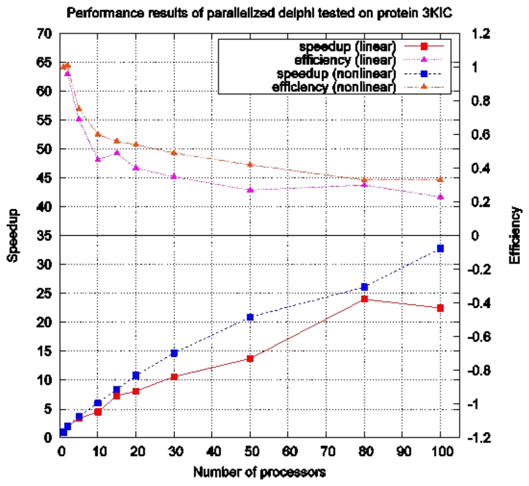
(d)

Figure 1. Performance results and electrostatic properties of 1VSZ. (a) Execution (purple) and iteration (red) time for solving the linear PBE, compared to execution (orange) and iteration (blue) time for solving the nonlinear PBE. (b) Speedup (red) and efficiency (purple) achieved by solving the linear PBE, compared to speedup (blue) and efficiency (red) obtained by solving the nonlinear PBE. (c) Resulting electrostatic field. (d) Resulting electrostatic potential.

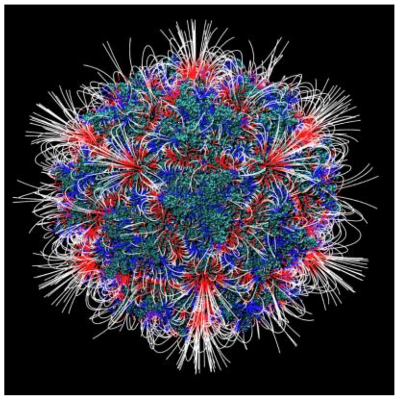
The next series of experiments were performed on the protein of adeno-associated virus 3KIC with the same Amber force filed and parameter setups. This case has significantly more atoms (atoms in a pdb file of size 25.4 MB) than those of 1VSZ (atoms in a pdb file of size 9.5 MB). The results are presented in Figure 2(a)-(d).



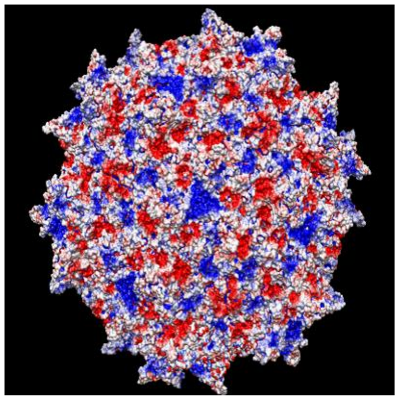
(a)



(b)



(c)



(d)

Figure 2. Performance results and electrostatic properties of 3KIC. (a) Execution (purple) and iteration (red) time for solving the linear PBE, compared to execution (orange) and iteration (blue) time for solving the nonlinear PBE. (b) Speedup (red) and efficiency (purple) achieved by solving the linear PBE, compared to speedup (blue) and efficiency (red) obtained by solving the nonlinear PBE. (c) Resulting electrostatic field. (d) Resulting electrostatic potential.

The 3rd series of experiments were performed on the bovine mitochondrial supercomplex 2YBB. Scale = 1.0 to 2.0 with increment 0.2 and 80% filling of the box domain were set in the parameter file. The resulting CPU time to solve the PBE (linear and nonlinear) for the protein 2YBB on various scales using the sequential DelPhi95 program is shown in Figure 3.

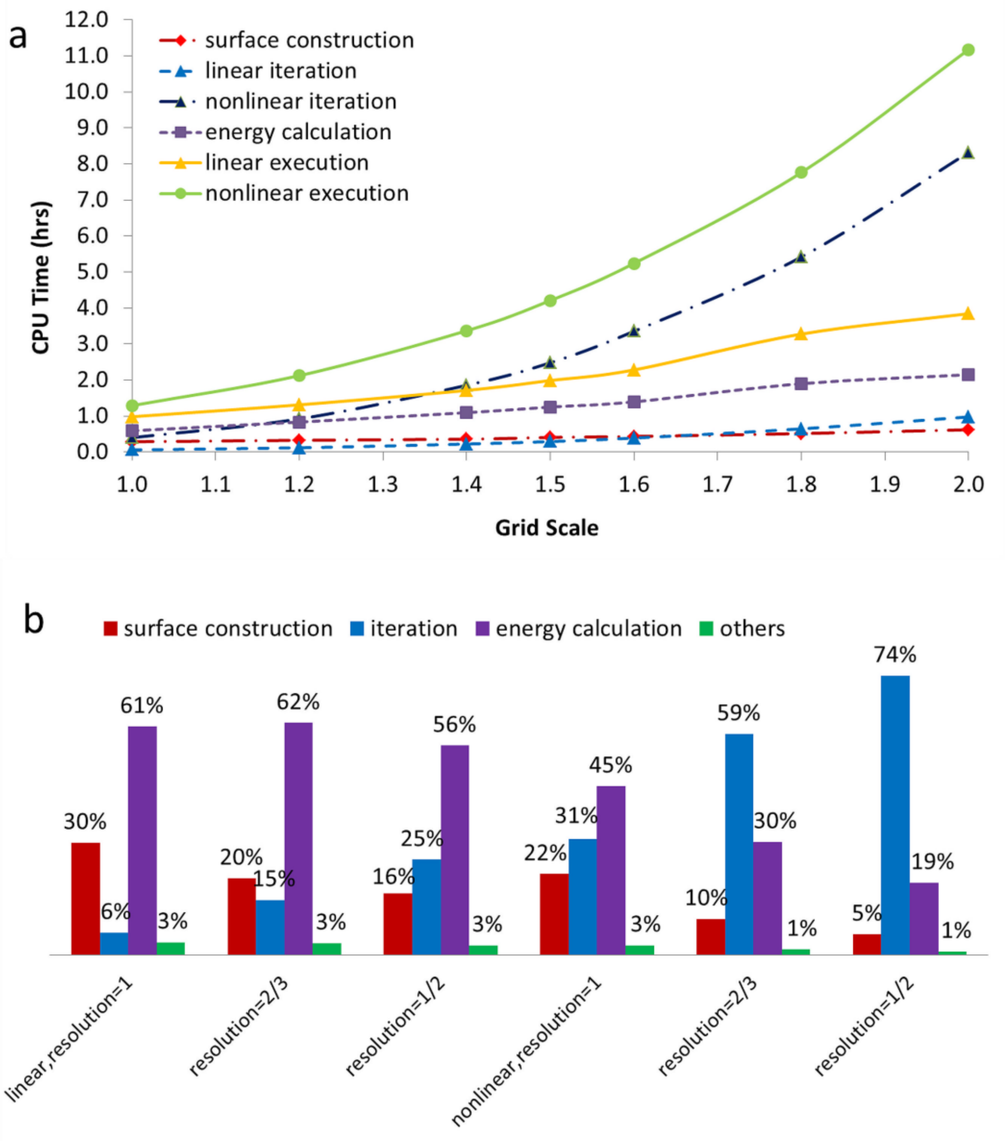


Figure 3. Increasing CPU time of the DelPhi program when calculating electrostatic potential and energies for protein 2YBB. a) CPU time cost by individual calculations and overall execution time at various grid resolutions b) Individual percentage contributed by each parallel calculation when solving linear and nonlinear PBE at 3 selected grid scale=1.0, 1.5 , and 2.0 grid/Å.

The results obtained by corresponding parallel computing experiments are shown in Figure 4.

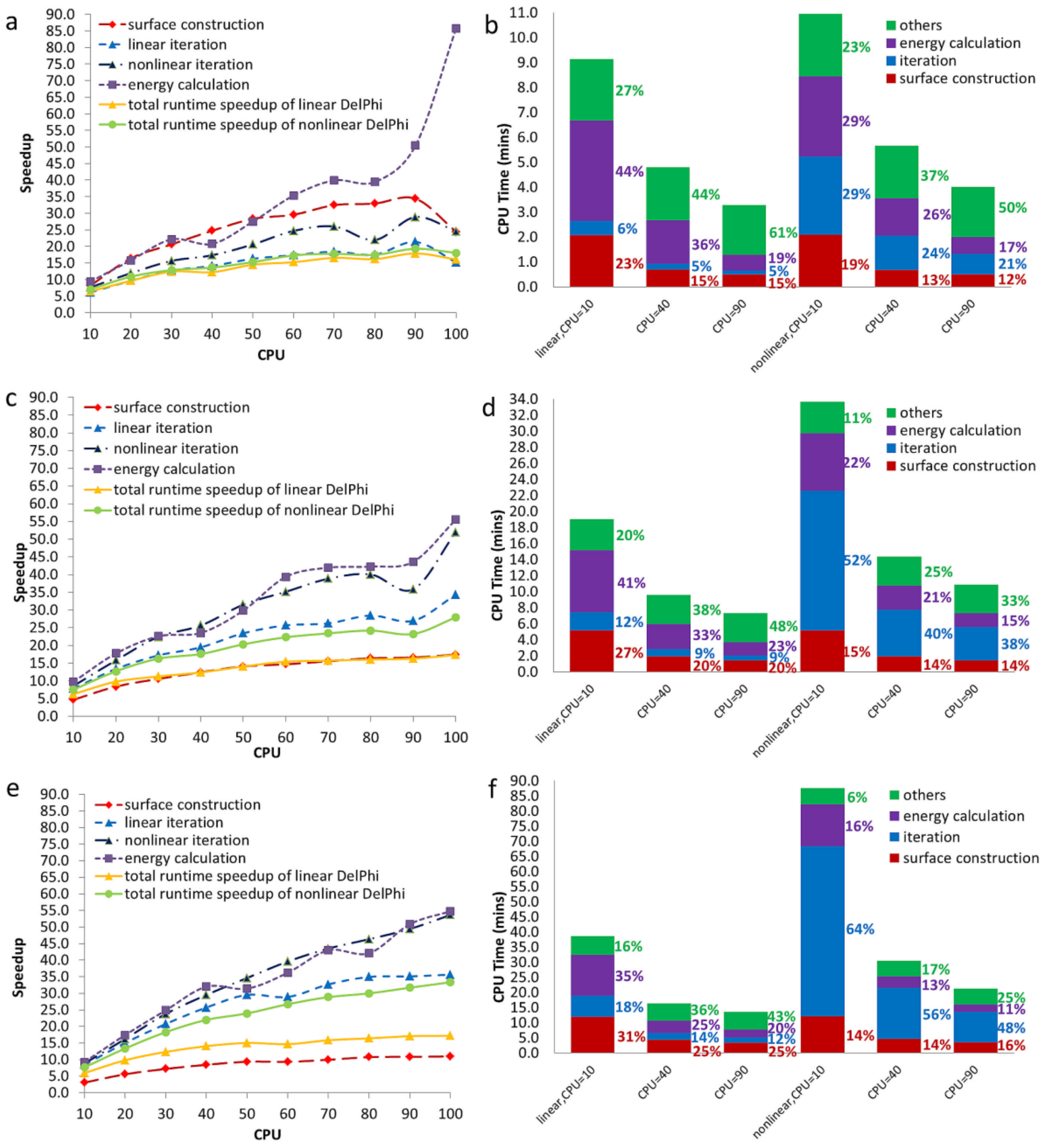


Figure 4. Speedups obtained by the MLIPB method at scale =1.0, 1.5, and 2.0 grid/Å and results obtained for selected examples at CPU=10, 40, 90. a)-b) Results obtained at grid resolution=1 Å/grid. c)-d) Results obtained at grid scale=1.5 Å/grid. e)-f) Results obtained at grid scale=2.0 grid/Å/grid).

The resulting electrostatic field and potential maps are shown in Figure 5.

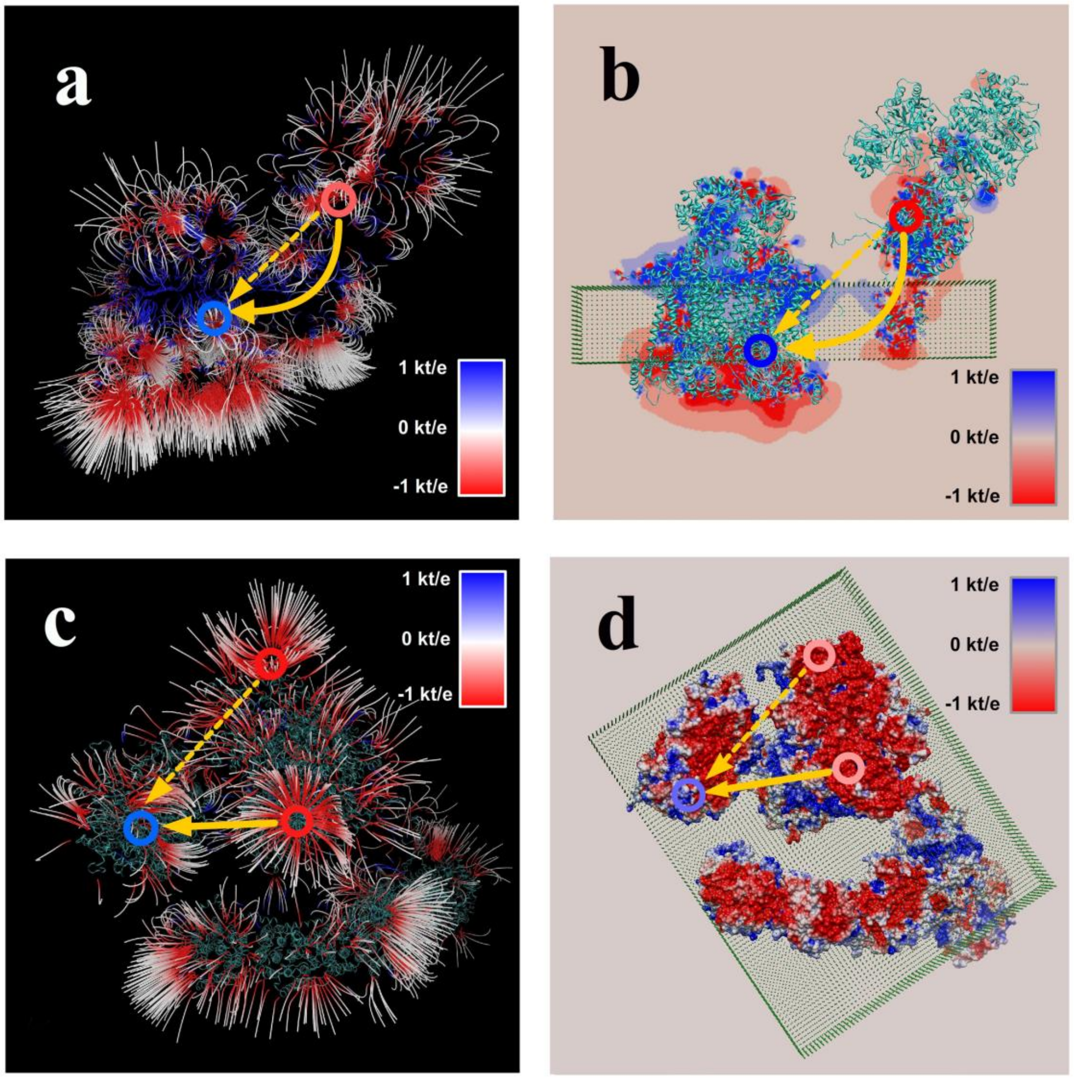


Figure 5. Resulting electrostatic field and potential maps of the bovine mitochondrial supercomplex. Membrane is shown as a slab made of pseudo atoms. (a) Electrostatic field distribution in case of side-view; (b) Potential distribution in a plane at the center of the supercomplex, side-view. The protein moiety is shown as well; (c) Electrostatic field distribution in case of membrane-view; and (d) Electrostatic potential mapped onto molecular surface of the supercomplex, membrane-view. The protein moiety is shown as well.