Supporting information for MSCALE: A General Utility for Multiscale Modeling

H. Lee Woodcock,[†,*] Benjamin T. Miller,[‡] Milan Hodoscek,[§] Asim Okur,[‡]
Joseph D. Larkin,[‡] Jay W. Ponder,[¶] Bernard R. Brooks[‡,*]

February 28, 2011

† Department of Chemistry, University of South Florida, 4202 E. Fowler Ave., CHE205, Tampa, FL 33620-5250
‡ Laboratory of Computational Biology, National Heart Lung and Blood Institute, National Institutes of Health, Bethesda, MD 20892.
§ Center for Molecular Modeling, National Institute of Chemistry, Hajdrihova 19, SI-1000 Ljubljana, Slovenia.
¶ Department of Chemistry and Molecular Biophysics, Washington University School of Medicine, 660 S. Euclid Ave., Box 8231, St. Louis, MO 63110
* hlw@usf.edu, brb@nih.gov

1 Example MSCAle input scripts

Three examples of MSCALE inputs are given below to demonstrate how to set up and run multiscale calculations using this tool. The first example shows how to set up a simple ONIOM QM/MM calculation. The script that runs the client is as follows:

1.1 Subtractive Multiscale: QM/MM

```
* MSCALE for QM/MM pentane, minimization
*

read rtf card name top_all35_ethers.rtf
read param card name par_all35_ethers.prm
read psf card name pentane.psf
read coor card name pentane.crd

! need to add a link atom between the C2 and C3 carbons, since we've broken
! that bond
addl qqh1 np 1 c3 np 1 c2
lonepair coli dist 0.0 scale -0.7261 sele type qqh1 end sele type c3 end sele type c2 end
coor shake

print coor

! set up the regions
define mmreg sele type c1 .or. type h11 .or. type h12 .or. type h13 .or. -
                  type c2 .or. type h21 .or. type h22 end
define qmreg sele .not. mmreg end

! MSCALE: We need 3 subsystems as follows:
! alllow: The full system at the MM level of theory
! modhigh: The model at the QM level of theory
! modlow: The model at the MM level of theory

mscale nsubs 3

subs alllow coef 1.00 prog "/home/charmm/bin/charmm" -
```

```
            outp "subsys/all_low-tt.out" inpu "subsys/all_low.inp" -
            sele .not. type qqh1 end

subs modhigh coef 1.00 prog "/home/charmm/bin/charmm" -
            outp "subsys/model_high-tt.out" inpu "subsys/model_high.inp" -
            sele qmreg .or. type qqh1 end

subs modlow coef -1.00 prog "/home/charmm/bin/charmm" -
            outp "subsys/model_low-tt.out" inpu "subsys/model_low.inp" -
            sele qmreg .or. type qqh1 end

sysd
end

skip all
coor copy comp

mini sd nstep 50 nprint 10
mini abnr nstep 20000 nprint 25 tolg 0.001

write coor card name oniom-moremini-tt.crd
* pentane minimized from TT
*

stop
```

This script starts out as normal, with the reading of topology and parameter files (which should include information for the link atom). The `MSCAle` command is then invoked and three subsystems are defined; one server will be launched for each of these subsystems using the executable specified by the `PROG` argument and the given inputs and outputs. Based on these subsystems and their given `COEFficients`, the final total potential energy will be given by $E^{total} = E^{MM}_{alllow} + E^{QM}_{modelhigh} - E^{MM}_{modellow}$. The `SYSDefinition` command is optional and makes CHARMM print out information on the subsystems that have been set up. The "skip all" line tells the client processor not to calculate any energy terms (as these will be computed in their entirety by the subsystems). The "work" of the calculation can then be performed as normal (in this case an energy minimization is performed).

The inputs for the three server runs are...
all_low.inp:

```
* whole system (pentane) at the low level of theory
*

read rtf card name top_all35_ethers.rtf
read param card name par_all35_ethers.prm
read psf card name pentane.psf
read coor card name pentane.crd

energy cdie cutnb 14.0 ctofnb 12.0 fshift vshift atom

server
stop
```

model_low.inp:

```
* model system at the low level of theory
*

read rtf card name top_all35_ethers.rtf
read param card name par_all35_ethers.prm
```

```
read psf card name 2c-model.psf
read coor card name 2c-model.crd

energy cdie cutnb 14.0 ctofnb 12.0 fshift vshift atom

server
stop
```

model_high.inp:

```
* model system at the low level of theory
*

read rtf card name top_all35_ethers.rtf
read param card name par_all35_ethers.prm
read psf card name 2c-model.psf
read coor card name 2c-model.crd

! Compute energy from Q-Chem
ENVI qchemcnt  "qchem_ctrl.inp"
ENVI qcheminp  "qchem.inp"
ENVI qchemexe  "qchem"
ENVI qchemout  "qchem.out"

qchem remove sele all end

print coor

energy

server
stop
```

All three of these scripts are similar in construction. The topologies and parameters are read in along with the structure and then the `SERVer` command is invoked. This command will go into a loop waiting for new coordinates from the client, calculating the energies and forces, and sending those back to the client. This loop will continue until the CHARMM script on the client terminates. Note that the nonbond list and (for calculations using periodic boundary conditions) periodic image data should be set up before the `SERVer` command (the client will transmit new unit cell sizes at each step, but the crystal must be defined and built and any image recentering commands issued before the `SERVer` command).

1.2 CHARMM / AMBER Free Energy Perturbation

The second example shows a free energy perturbation between the CHARMM22 and AMBER99SB force fields (as implemented in the CHARMM program). The client script is:

```
* MSCALE CHARMM/AMBER free-energy perturbation
*

! need this to avoid "block is untested w/ pert"
bomlev -2

! read topology, parameter, psf, and crd files
read rtf card name top_all22_prot.inp
read param card name par_all22_prot.inp
read psf card name alad.psf
read coor card name charmm-c7eq-nocut.crd

! initial energy
nbonds atom shift e14fac 1.0 wmin 1.5 cutnb 16. ctofnb 12. ctonnb 10.
energy

! remove the master processor's energy/force contribution
block 1
coef 1 1 0.0
end

! start up PERT, what we will do is start pert, then set up the MSCALE subsystems. Once this
! is done, we'll run dynamics, SKIPping all the energy contributed by the host system.
pert

mscale nsubs 2

! first subsystem is lambda, AMBER force field
subs lam1 lamb prog "/home/charmm/bin/charmm" -
     outp "subsys/lamb-amber.out" inpu "subsys/lamb-amber.inp" -
     nproc 2 sele all end

! second subsystem is 1 - lambda, CHARMM force field
subs lam0 mlamb prog "/home/charmm/bin/charmm" -
     outp "subsys/mlam-charmm.out" inpu "subsys/mlam-charmm.inp" -
     nproc 2 sele all end

sysd
end

! do Langevin dynamics for temp. coupling and solvent collision
open write unform unit 31 name nocut-300k.dcd
open read card unit 88 name tim2-short.punit

energy

! low friction
scalar fbeta set 1.0

dyna lang leap strt -
  timestep 0.001 nstep 168000000 nprint 100 -
  iuncrd 31 nsavc 100 inbfrq -1 -
```

```
  iseed 314159 wmin 1.0 -
  tbath 300.0 firstt 300.0 finalt 300.0 -
  iasors 1 iasvel 1 iscvel 1 -
  punit 88

pert off
stop
```

This script is similar to the client script for the QM/MM calculations, but in this case only two subsystems are used. Instead of being given a coefficient, these are given the `LAMBda` and `MLAMbda` areguments meaning their coefficient will be dynamically determined by the value of λ and $1 - \lambda$. Also, for this case the `BLOCK` command has been used instead of `SKIPe` to remove the client processor's energy contribution, The input files for the subsystems are...
lamb-amber.inp:

```
* amber FF
*

read rtf card name parm99_all-tim.rtf
read param card name parm99_all.prm
read psf card name amber.psf
read coor card name amber-c7eq-nocut.crd

nbonds atom switch cutnb 16. ctofnb 12. ctonnb 10.
energy
server
```

mlam-charmm:

```
* charmm FF
*

read rtf card name top_all22_prot.inp
read param card name par_all22_prot.inp
read psf card name alad.psf
read coor card name charmm-c7eq-nocut.crd

nbonds atom switch cutnb 16. ctofnb 12. ctonnb 10.
energy
server
```

1.3 SANDER Module

The final example demonstrates how to run AMBER's SANDER program through MSCALE.

```
* test amber with an alanine dipeptide
*

read rtf card name parm99_all.rtf
read param card name "parm99SB_all.prm"
read psf card name charmm-diala.psf
read coor card name charmm-diala.crd

mscale nsubs 1

subs test coef 1.0 prog "/home/charmm/bin/sander.MPI" -
            amber outp "ambersub/mdout" inpu "ambersub/mdin" -
            sele all end

sysd
end

! remove main processor contribution
skip all
shake bonh

! get AMBER energy

mini sd nstep 10
mini abnr nstep 100

open unit 20 write file name test2.dcd

dyna start nstep 1000 timestep 0.001 nprint 1 -
 firstt 50 finalt 100 tbath 100 teminc 5 ihtfrq 1000 -
 iuncrd 20 nsavc 1 -
 echeck 20.0

energy
coor force comp
print coor comp
print coor

stop
```

This script only contains one subsystem and the sander.MPI executable is used instead of a CHARMM executable. This requires the use of a special keyword (`AMBEr`) in the `SUBSystem` command. When MPI_Comm_spawn is called, this keyword will automatically add the correct command line arguments to put SANDER into MSCALE server mode. Although this example is very simple in that SANDER completely computes the energy and force terms, it is possible to only have SANDER work on part of the system by manipulating the atom selection.

These scripts can be run in the same way as a standard CHARMM parallel run using mpirun; no extra arguments need to be given to mpirun. However, the authors are unaware of any current MPI implementation that can change its processor allocation dynamically, so it is important to give mpirun a node or host file with sufficient processors to run all of the desired servers. Likewise, the resource request to the batch scheduler should specify enough processors to run **all** of the servers. Since the servers are spawned dynamically, however, the argument to the `-n` or `-np` option of mpirun should be the number of processors that the user wishes the client to have access to. Additionally, many MPI implementations do busy-waiting for communication, meaning that even processes that are not actively performing computations will use 100% of a CPU core. The authors have found that using MPICH2 with the ch3:sock device does not busy-wait, but this has not been explicitly tested with MSCALE.

2 Technical description of the MSCALE implementation

The MSCALE functionality is present in CHARMM version 35 and above. The various subroutines that provide its functionality are found in `source/mscale/mscale.src`. The calculations in this paper were done using development version 36, which contains bug fixes and feature enhancements. CHARMM must be compiled with the `MSCALE` keyword in pref.dat in order to use this feature. Within the `mscale.src` file, `SUBROUTINE MSCALE` handles the parsing of the commands related to MSCALE, including the `MSCAle` command itself along with `SUBSystem`, `SERVer`, etc. When the `MSCAle` command is issued, this subroutine sets a logical variable (`QMSCALE`) to true, indicating to CHARMM that results from subsystems should be folded into the total energy calculation (recall that the `MSCAle` command is only issued on the client side). The `MSCALE` subroutine also performs some basic sanity checking of the options provided and (on the client side) spawns the required servers.

When `QMSCALE` is true, CHARMM's main energy subroutine (`SUBROUTINE ENERGY` in `source/energy/energy.src` calls the `EMSCALE` subroutine twice: once at its start to broadcast information to the subsystems and again near the end to collect the energies, forces, etc. back. `EMSCALE` is told whether it should send data out or collect it back via a flag variable (`MSCINIT`). If `EMSCALE` is to send out data, it calls the routine `MAINBROADCAST`, which uses MPI_Bcast to broadcast the following information:

1. A control parameter (`NCONTROL`), saying whether the client is shutting down (in `MAINBROADCAST`, this is always 1, meaning that the calculation will continue).
2. If the `CRYSTal` flag is used, the periodic box unit cell data.
3. If the `QATOM` flag is used, the number of atoms in the system and their floating point format.
4. The X, Y, and Z coordinate arrays (after they have been filled in with only atoms relevant to the particular server).
5. The `QSECD` logical variable, telling the subsystem whether or not it should calculate second derivative terms.
6. If the `TORQue` flag is used, rotation matrices for torque centers.

If, on the other hand, `EMSCALE` is to receive information from the servers, it calls the routine `MAINRECEIVE` to handle this communication. This routine receives (again via MPI_Bcast) the following data from the server(s):

1. The X, Y, and Z force arrays.
2. The energy array.
3. If the `CRYSTal` flag is used, the virial and pressure terms calculated on the server.
4. If second derivatives were requested, the server's Hessian matrix.
5. If the `TORQue` flag is used, the torques on the centers.

All of the terms collected in this case are then added to those calculated directly on the client scaled by the user specified coefficient of the subsystem.

On the server side, once the `SERVer` command is processed by `SUBROUTINE MSCALE`, the `LOOPENE` subroutine is immediately called. This routine goes into a loop until the client program exits. The first routine called is `SERVERRECEIVE`, which posts MPI_Bcast calls corresponding to those in `MAINBROADCAST`. Once this is done, the new coordinates and unit cell information (if needed) is up to date and the `UPDECI` routine is then called to update the nonbond list if needed. If second derivatives are requested, the data structures for the Hessian matrix are then prepared and the `ENERGY` subroutine is called. Otherwise, the `GETE` routine is called, which is acceptable because the nonbond list has been updated. Once `GETE` or `ENERGY` have returned, the energies, forces, and (if desired) Hessian will have been calculated. At this point, the `SERVERBROADCAST` routine is called to broadcast these results back to the client in the same order as described above for `MAINRECEIVE`.

In order to make a code MSCALE-compatible, it is necessary to decide whether it will support running as a server or as both a server and a client. It is recommended to get server mode working first. In this case, it is only necessary to re-implement the `SERVERRECEIVE` and `SERVERBROADCAST` subroutines described above. The `SERVERRECEIVE` routine must receive the data in the order that it is sent by the client. It must also have some mechanism of determining whether optional terms such as unit cell data will be broadcast. This can be done via the server program's input file, special command line arguments, or some other method, however the use of a command line argument will require modification to the code in CHARMM that spawns the server program. The `SERVERRECEIVE` routine should be able to parse the data sent to it (including converting units from AKMA to the local code's units as needed) and update the code's own coordinate array etc. as necessary. Likewise, the `SERVERBROADCAST` routine must take the energy and force terms computed locally and put them in the form and units of CHARMM's energy and force

arrays. It must also package any other data (e.g virial terms) that are required, and send them back to the client in the order specified above. Once this process is working, client-side routines can be written in a similar fashion. In any case, it is highly recommended that the implementor study the routines in CHARMM and duplicate their structure as much as practicable to interface with the local code. Future plans include encapsulating the communications calls into an API, which will ease porting.

If it is too cumbersome to incorporate these MPI routines into the target code, it is possible to write a wrapper program that handles the communication. Examples of such wrappers for Gaussian03, Molpro, NWChem, and PSI3 can be found in the `source/mscale` directory of the CHARMM distribution. These wrappers implement the functionality of the `SERVERRECEIVE` and `SERVERBROADCAST` routines described above, but do the additional work of setting up all necessary input files for the target program and then executing it by a call to Fortran's `SYSTEM` routine. Additionally, the wrappers parse the output of the target program and extract the necessary information to be broadcast back to the client. This approach is less flexible than the method described in the previous paragraph, however, it has the advantage of not requiring any modification (or even access) to the source code of the target program (i.e. server). By use of a wrapper, practically any program that can compute and output energies and forces when given a set of coordinates can be interfaced with MSCALE.