Performance Analysis of an ad_OSS Molecular Dynamics Software

Soon-Heum "Jeff" Ko

National Supercomputing Centre, Linkoping University

0. Introduction

ad_OSS: program for modelling of water molecules

Quantum Chemistry, F77 based

(F77 -> F90, Parallelization, Co-array-Fortran?)

a) The "classical" molecular dynamics simulations are used inter-molecular potentials to describe interactions between molecules. We have previously developed a polarization bar model for water, which has an additional feature: it allows the dissociation of water molecules, so that the EXV. H_3O^+ and OH^-can be formed. This has over the years been used in a wide range of articles to describe EXV protonated water clusters of relevance to atmospheric and environmental chemistry. Potential model is implemented in the program "ad_OSS.f".

b) The program uses the dynamic equations for both positions / velocities and dipole moment, and use exv.vanliga matrix impact discussions and coupled double or triple loops of molecules in a variety of routines. Unfortunately, the program is fully serial, but we would of course wish that it was possible to run in parallel.

c) Fully serial, and therefore very poor performance on modern multicore / multiprocessor machines.

d) It is written in Fortran77. It would be good if dynamic allocation could be used.

1. Code performance benchmark

-- via vtune

1) For compilation: it uses BLAS and LAPACK.

%%%%% MKL BLAS/LAPACK

static linking of code myprog.f90 with Intel MKL sequential library.

```
ifort   myprog.f90  -shared-intel -Wl,--start-group
/global/apps/intel/mkl/10.2.0.013/lib/em64t/libmkl_intel_lp64.a
/global/apps/intel/mkl/10.2.0.013/lib/em64t/libmkl_sequential.a
/global/apps/intel/mkl/10.2.0.013/lib/em64t/libmkl_core.a -Wl,--end-group -lpthread
```

static linking of code myprog.f90 with Intel MKL parallel (threaded) library.

```
ifort   myprog.f90 -shared-intel -Wl,--start-group
/global/apps/intel/mkl/10.2.0.013/lib/em64t/libmkl_intel_lp64.a
/global/apps/intel/mkl/10.2.0.013/lib/em64t/libmkl_intel_thread.a
/global/apps/intel/mkl/10.2.0.013/lib/em64t/libmkl_core.a   -Wl,--end-group  -lguide -lpthread
```

dynamic linking of code myprog.f90 with Intel MKL parallel (threaded) library.

```
ifort   -shared-intel -Wl,-rpath,/global/apps/intel/mkl/10.2.0.013/lib/em64t -
L/global/apps/intel/mkl/10.2.0.013/lib/em64t  -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -lguide
-lpthread myprog.f90
```

In default Makefile,

```
LDFLAGS = -L/software/intel/composer_xe_2011_sp1.6.233/mkl/lib/intel64 -lmkl_intel_lp64 -
lmkl_sequential -lmkl_core
```

2) Running the code

Need to

```
export LD_LIBRARY_PATH="/software/intel/composer_xe_2011_sp1.6.233/mkl/lib/intel64"
```

unless the library had been linked statically

For running, type the command

```
./ad_OSS.x < adohaq128E0.i6
```

Or run the script

```
run.sh
```

To provide the profiling result:

```
amplxe-cl --result-dir ./Profile -c hotspots ./ad_OSS.x < adohaq128E0.i6
```

3) Profiling

```
amplxe-gui ./Profile
```

Elapsed Time: 1503.721

CPU Time: 1500.520

Top Hotspots

Function	CPU Time
dipolemoments_oss	360.150s
delocalmatrices_oss	215.904s
mkl_blas_xdsyr	167.960s
distances	141.264s
dipoleforce_oss	117.711s
[Others]	497.531s

1. Under dipolemoments

Line		Source		CPU Time
4976		A(nu,mu) = (alfafit*Sdd(i,j)/absr(i,j)**3) *		37.235s
4995		sumS = sumS + r(i,j,nusub)*q(j)/absr(i,j)**3 * Scd(i,j)		67.619s
5012		j = lpekO(int((mu+2)/3))		22.656s
5022		A(nu,mu) = A(nu,mu) +	! cT2	3.206s
5023	X	alfafit*DerSpbc(i,j)/absr(i,j)**4 *	! cT2	15.721s
5024	X	r(i,j,nusub)*r(i,j,musub)	! cT2	9.483s

a) change the loop ordering:

```
do j; do i
array(i,j)=f()
enddo; enddo
```

Line	Source	CPU Time
5008	call Ewald_dipolemoments(A_save,B_save)	181.945s

b) change the loop ordering:

2. Under delocalmatrices

Line	Source	CPU

		Time
4139	DerScd(j,i) = 0.0D0	46.810s
4143	DerSpbc(j,i) = 0.0D0	28.216s
4199	Spbc(i,j) = Ewald_Spbc(absr(i,j))	16.977s
4201	Scd(i,j) = Scd(i,j) + Spbc(i,j) - 1.0D0	34.662s
4205	DerSpbc(i,j) = Der_Ewald_Spbc(absr(i,j))	47.964s
4207	DerScd(i,j) = DerScd(i,j) + DerSpbc(i,j)	59.271s
4211	Der2Spbc(i,j) = Der2_Ewald_Spbc(absr(i,j))	19.844s

a) change the loop ordering:

Line	Source	CPU Time
4153	Scd(i,j) = Cutoff_Scd_OH_OSS(absr(i,j))	23.082s
4155	DerScd(i,j) = Der_Cutoff_Scd_OH_OSS(absr(i,j))	26.191s
4176	DerScd(i,j) = Der_Cutoff_Scd_OO_OSS(absr(i,j))	18.715s

b) Probably time taken for those function calls

3. Under dipoleforce

Line	Source	CPU Time
5364	Fuui(k) = 0.0D0	13.058s
5378	ujMrji = rji(1)*uj(1) + rji(2)*uj(2) + rji(3)*uj(3)	25.259s
5390	Fuqi(k) = -(Qi*ujMrji*DerScd(j,i)*rji(k)/rjiabs**4) +	32.300s
5391	X Scd(j,i)*	1.933s
5392	X (3*Qi*ujMrji*rji(k)/rjiabs**5 - Qi*uj(k)/rjiabs**3)	52.341s
5393	X +(Qj*uiMrji*DerScd(j,i)*rji(k)/rjiabs**4) - !	8.190s
5394	X Scd(j,i)* !	0.120s
5395	X (3*Qj*uiMrji*rji(k)/rjiabs**5 - Qj*ui(k)/rjiabs**3) !	4.590s

a) basically there is no much special technique to reduce this part

5367	rij(k) = r(i,j,k)	24.910s

b) minor recommendation: reordering r(i,j,k) array into r(k,i,j) as it runs k-loop

4. Under distances

Line	Source	CPU Time
3751	r(i,j,2) = (x(j,2) - x(i,2))/a2m	20.377s
3752	r(i,j,3) = (x(j,3) - x(i,3))/a2m	23.059s
3761	rc(k) = CelLVecStar(k,1) * r(i,j,1) +	13.682s
3771	& - CelLVec(2,k)*animage(2)	10.861s
3776	do j=i+1,natoms	54.169s
3780	absr(i,j) = dsqrt(r(i,j,1)**2 + r(i,j,2)**2 + r(i,j,3)**2)	17.613s

a) all are related with loop control + r array needs to be r(1:3,i,j)

5. Suggestion for tuning

a) Loop ordering

For example in line 4967, the 2-d loop generated as

```
do nu=1,3*noxygen          ! Calculate upper half, including diagonal
  i = lpekO( int((nu+2)/3) )
  nusub = mod(nu+2,3)+1
  do mu=nu,3*noxygen
    j = lpekO( int((mu+2)/3) )
    musub = mod(mu+2,3)+1
    if (i.eq.j) then
      A(nu,mu) = ijdelta(nu,mu)  ! idelta ersatt av en 3nOx3nO matris
    else
      A(nu,mu) = (alfafit*Sdd(i,j)/absr(i,j)**3) *
X              (ijdelta(nusub,musub) -
X               3*r(i,j,nusub)*r(i,j,musub)/absr(i,j)**2)
    endif
  enddo
enddo
```

Instead,

```
do mu=1,3*noxygen          ! Calculate upper half, including diagonal
   j = lpekO( int((mu+2)/3) )
   musub = mod(mu+2,3)+1
  do nu=1,mu
   i = lpekO( int((nu+2)/3) )
   nusub = mod(nu+2,3)+1
     …
  enddo
  enddo
```

will be helpful.

b) Looping indices

In line 4149,

```
  do l=1,nhydrogen
    do k=1,noxygen
      i = lpekO(k)
      j = lpekH(l)
      Scd(i,j) = Cutoff_Scd_OH_OSS(absr(i,j))
      Scd(j,i) = Scd(i,j)
      DerScd(i,j) = Der_Cutoff_Scd_OH_OSS(absr(i,j))
      DerScd(j,i) = DerScd(i,j)
    enddo
  enddo
```

It is a l-k loop while arrays(I,j) are updated. In that case, i- and j- increments might not be sequential/linear. The loop variables can be changed? Loop order inversing intends to increase the cache-hit ratio.

c) Data structure

For example, about r

Why r is constructed as r(natoms,natoms,1:3)? As the subroutine calls r(I,j,1:3) in sequence, it cannot gain the benefit of caching.

If data structure change is the major task, how about changing the calls as follows?

For example, from line 3776,

```
    do j=i+1,natoms
      r(j,i,1) = -r(i,j,1)
      r(j,i,2) = -r(i,j,2)
      r(j,i,3) = -r(i,j,3)

      absr(i,j) = dsqrt(r(i,j,1)**2 + r(i,j,2)**2 + r(i,j,3)**2)

      absr(j,i) = absr(i,j)

    enddo
```

Instead,

```
    r(i+1:natoms,i,1) = -r(i,i+1:natoms,1)

    r(i+1:natoms,i,2) = -r(i,i+1:natoms,2)

    r(i+1:natoms,i,3) = -r(i,i+1:natoms,3)

  ...
```

It will do the same thing as the do loop above, while it will complete the copy operation for i- and j-indices first, so can profit on cache hit.

6. Auto-parallelization

-- Intel compiler's auto-parallelization option turned on

1) Compilation

FFLAGS = -O3 -ip -parallel -par-report3

LDFLAGS = -L/software/intel/composer_xe_2011_sp1.6.233/mkl/lib/intel64 -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -liomp5 -lpthread

% libiomp5 and libpthread are incorporated in intel compiler lib. Its path is /software/intel/composer_xe_2011_sp1.10.319/compiler/lib/intel64

2) Running

The number of threads defaults to the total number of logical processor cores or hardware threads, but can be overwritten via the OMP_NUM_THREADS environment variable. Thus, all work identical on Kappa:

```
export OMP_NUM_THREADS=8; ./ad_OSS.x < adohaq128E0.i6
```

or

```
./ad_OSS.x < adohaq128E0.i6
```

3) References

For reference on auto-parallelization,

http://software.intel.com/en-us/articles/automatic-parallelization-with-intel-compilers/

More details at

http://software.intel.com/en-us/articles/intel-guide-for-developing-multithreaded-applications/

Also, check the compiler option guide + compiler optimization setup guide (PDF)

4) Analysis

Compared against the sequential run,

Elapsed Time: 1264.622

CPU Time: 9702.400

Top Hotspots

Function	CPU Time
__kmp_wait_sleep	4616.140s
__kmp_x86_pause	1819.653s
__kmp_execute_tasks	1198.843s
dipolemoments_oss	356.721s
__sched_yield	312.758s
[Others]	1398.284s

%%%%% Thread's waiting takes most time.

For actual functions,

dipolemoment: 356.721

delocalmatrices: 228.843

mkl_blas_xdsyr: 163.633

distances: 136.379

dipoleforce: 121.301

Auto-parallel W/O explicit OMP_NUM_THREAD and W OMP_NUM_THREAD provided identical results.

%%% Unfortunately the gain by threading was just 2.5 mins, while the full CPU has been consumed (through TOP). Potentially because the problem size was so small (so overhead to threading sets off the gain by multi-threading) or the inversed loop order (i.e., inefficient do looping) in this code is not suitable for threaded run.

7. Discussion point

-- Why don't using other open-source codes? They don't have this capability?

> GROMACS: Rossen Apostolov <rossen@kth.se>

One needs to not only completely redefine the properties of the generated ions and their topology, but also model reliably the transition state of the dissociative process.

It's possible to couple Gromacs with some QM software packages which would treat only a part of the system as QM (thus able to dissociate) and the rest classical.

> Dalton: Olav Vahtras <vahtras@pdc.kth.se>

-- Ability to make a user-defined function (library) and incorporate it on the open-source ones

> GROMACS:

Does not seem easy

> Dalton:

-- User community of this code? How much publication can be expected from this effort? How actively do you plan to use the parallelized code?

-- How much speed-up is hoped? If order of 1, how about OpenMP or accelerator (GPU / IntelMIC)?

> Co-array Fortran

-- How about opening a master's project? We can give the advising / supervising role.

> Or the small performance improvement by reordering loops and OpenMP programming.

8. Personal question

-- Ability to predict/describe Coulombinc force field on non-periodic field?

9. Further actions

Tuned

Elapsed Time: 1852.856

CPU Time: 1851.730

Top Hotspots

Function	CPU Time
erfc	509.308s
dipolemoments_oss	316.514s
delocalmatrices_oss	212.226s
mkl_blas_xdsyr	166.400s
exp.L	153.237s
[Others]	494.044s

%%%%% erfc: error function. probably some error during code modification.

Autopar_Tuned

Elapsed Time: 1734.746

CPU Time: 12591.900

Top Hotspots

Function	CPU Time
__kmp_wait_sleep	6088.513s
__kmp_x86_pause	2412.077s

__kmp_execute_tasks	1596.845s
erfc	477.012s
__sched_yield	408.553s
[Others]	1608.901s

%%%%% Thread's waiting takes most time.

For actual functions,

dipolemoment 319.733

delocalmatrices 274.888

mkl_blas_xdsyr 163.459

distances 124.078

dipoleforce 118.600

Tuned2

Elapsed Time: 1301.873

CPU Time: 1300.610

Top Hotspots

Function	CPU Time
dipolemoments_oss	311.978s
delocalmatrices_oss	183.734s
mkl_blas_xdsyr	165.581s
distances	124.771s
dipoleforce_oss	119.318s
[Others]	395.227s

Autopar_Tuned2

Elapsed Time: 1219.531

CPU Time: 9388.440

Top Hotspots

Function	CPU Time
__kmp_wait_sleep	4552.221s
__kmp_x86_pause	1789.439s
__kmp_execute_tasks	1196.434s
dipolemoments_oss	318.607s
__sched_yield	303.136s
[Others]	1228.603s

%%%%% Thread's waiting takes most time.

For actual functions,

```
dipolemoment: 318.607

delocalmatrices: 188.621

mkl_blas_xdsyr: 163.793

distances: 121.890

dipoleforce: 121.214
```

```
large400-triclinic.F(7673): (col. 23) remark: LOOP WAS AUTO-PARALLELIZED.
large400-triclinic.F(7675): (col. 23) remark: LOOP WAS AUTO-PARALLELIZED.
large400-triclinic.F(7677): (col. 23) remark: FUSED LOOP WAS AUTO-PARALLELIZED.
large400-triclinic.F(7682): (col. 18) remark: LOOP WAS AUTO-PARALLELIZED.
large400-triclinic.F(7680): (col. 18) remark: LOOP WAS AUTO-PARALLELIZED.
large400-triclinic.F(7684): (col. 15) remark: LOOP WAS AUTO-PARALLELIZED.
large400-triclinic.F(7684): (col. 15) remark: LOOP WAS AUTO-PARALLELIZED.
large400-triclinic.F(1902): (col. 10) remark: LOOP WAS AUTO-PARALLELIZED.
large400-triclinic.F(1923): (col. 7) remark: LOOP WAS AUTO-PARALLELIZED.
large400-triclinic.F(1357): (col. 21) remark: LOOP WAS AUTO-PARALLELIZED.
large400-triclinic.F(1360): (col. 21) remark: LOOP WAS AUTO-PARALLELIZED.
large400-triclinic.F(1363): (col. 21) remark: FUSED LOOP WAS AUTO-PARALLELIZED.
large400-triclinic.F(1382): (col. 18) remark: LOOP WAS AUTO-PARALLELIZED.
large400-triclinic.F(1387): (col. 15) remark: LOOP WAS AUTO-PARALLELIZED.
large400-triclinic.F(5761): (col. 11) remark: LOOP WAS AUTO-PARALLELIZED.
large400-triclinic.F(6510): (col. 13) remark: LOOP WAS AUTO-PARALLELIZED.
large400-triclinic.F(7359): (col. 13) remark: LOOP WAS AUTO-PARALLELIZED.
large400-triclinic.F(6368): (col. 7) remark: FUSED LOOP WAS AUTO-PARALLELIZED.
```

```
large400-triclinic.F(5043): (col. 7) remark: LOOP WAS AUTO-PARALLELIZED.

large400-triclinic.F(5106): (col. 15) remark: LOOP WAS AUTO-PARALLELIZED.

large400-triclinic.F(5222): (col. 9) remark: DISTRIBUTED LOOP WAS AUTO-PARALLELIZED.

large400-triclinic.F(4813): (col. 7) remark: LOOP WAS AUTO-PARALLELIZED.

large400-triclinic.F(4253): (col. 10) remark: LOOP WAS AUTO-PARALLELIZED.

large400-triclinic.F(4004): (col. 10) remark: LOOP WAS AUTO-PARALLELIZED.

large400-triclinic.F(1513): (col. 45) remark: LOOP WAS AUTO-PARALLELIZED.

large400-triclinic.F(1513): (col. 45) remark: LOOP WAS AUTO-PARALLELIZED.

large400-triclinic.F(6252): (col. 26) remark: LOOP WAS AUTO-PARALLELIZED.

large400-triclinic.F(7304): (col. 24) remark: LOOP WAS AUTO-PARALLELIZED.

large400-triclinic.F(7197): (col. 7) remark: FUSED LOOP WAS AUTO-PARALLELIZED.

large400-triclinic.F(1722): (col. 7) remark: LOOP WAS AUTO-PARALLELIZED.

large400-triclinic.F(7145): (col. 7) remark: LOOP WAS AUTO-PARALLELIZED.

large400-triclinic.F(1803): (col. 10) remark: LOOP WAS AUTO-PARALLELIZED.

large400-triclinic.F(1826): (col. 7) remark: LOOP WAS AUTO-PARALLELIZED.

large400-triclinic.F(7457): (col. 29) remark: LOOP WAS AUTO-PARALLELIZED.

large400-triclinic.F(4911): (col. 7) remark: LOOP WAS AUTO-PARALLELIZED.

large400-triclinic.F(4941): (col. 15) remark: LOOP WAS AUTO-PARALLELIZED.

large400-triclinic.F(6748): (col. 24) remark: LOOP WAS AUTO-PARALLELIZED.

large400-triclinic.F(6796): (col. 7) remark: LOOP WAS AUTO-PARALLELIZED.
```