MOLCAS for advanced users

Valera Veryazov
Overview

- How to run Molcas in an efficient way?
- Code development in molcas environment
Quantum chemistry has specific demands to hardware!

- more (!) RAM
- fast CPU/GPU
- large cache and bus speed
- fast HDD, or better Solid State Disk
- network and intercommunication
## Cluster/SMP/Multicore

<table>
<thead>
<tr>
<th></th>
<th>RAM access</th>
<th>Disk</th>
<th>Hardware cost</th>
<th>Coding cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster</td>
<td>distributed</td>
<td>distributed</td>
<td>cheap</td>
<td>high</td>
</tr>
<tr>
<td>SMP</td>
<td>shared</td>
<td>external</td>
<td>expensive</td>
<td>low</td>
</tr>
<tr>
<td>Multicore</td>
<td>shared</td>
<td>shared</td>
<td>average</td>
<td>average</td>
</tr>
</tbody>
</table>

If intercommunication is low - cluster is the best solution! Molcas parallelization made for clusters.
across nodes..

Timings for parallelization over $n$ nodes

- SEWARD
- Cho-CASSCF
- RASPT2
- CASSCF
- Cho-SEWARD
- CASPT2
- Cho-CASPT2

Speed-up vs. $n$ (nodes)
across cores..
## Compilers (for Linux)

<table>
<thead>
<tr>
<th>Name</th>
<th>free?</th>
<th>stable?</th>
<th>performance</th>
<th>parallel</th>
</tr>
</thead>
<tbody>
<tr>
<td>g77</td>
<td>Yes</td>
<td>Yes</td>
<td>average</td>
<td>Yes</td>
</tr>
<tr>
<td>gfortran</td>
<td>Yes</td>
<td>Yes</td>
<td>good</td>
<td>Yes</td>
</tr>
<tr>
<td>g95</td>
<td>Yes</td>
<td>Yes</td>
<td>average</td>
<td>not tested</td>
</tr>
<tr>
<td>Intel</td>
<td>No</td>
<td>yes</td>
<td>excellent</td>
<td>Yes</td>
</tr>
<tr>
<td>SunStudio</td>
<td>Yes</td>
<td>yes</td>
<td>good</td>
<td>not tested</td>
</tr>
<tr>
<td>NAG</td>
<td>No</td>
<td>yes</td>
<td>good</td>
<td>some problems</td>
</tr>
<tr>
<td>Portland</td>
<td>No</td>
<td>yes</td>
<td>good</td>
<td>some problems</td>
</tr>
</tbody>
</table>
BLAS libraries

- 60 – 80% in RASSCF, CASPT2, CCSD codes
- ’MOLCAS’ BLAS and LAPACK
- ’Linux’ package
- GotoBLAS
- Atlas
- Intel MKL
- AMD ACML
Memory issues..

- MOLCASMEM - max allocation of dynamic memory in Mb
- only 64-bit installation allows > 2000 Mb
- do not overload MOLCASMEM!
- How to allocate memory for a parallel job?
Overclocking and benchmarking

- try `-speed fast`
- don’t expect too much from overclocking!
- always use local filesystem
- check `hdparm -tT`
- use BLAS libraries
- Always verify!

- `molcas verify performance` - running performance tests
- `molcas timing` - generate report
- `molcas timing -all` - generate report for any test
Standard verification fails....

Reasons for verification failure:

- numerical instability,
- compiler optimization problems,
- platform dependent bugs.

Ways to solve the problem:

- Reconfigure molcas with low optimization flags.
- use *snooper* script to locate routines, which are required low optimization.
Debug alternatives:

- `-speed debug`: decrease optimization level
- `$_MOLCAS_PRINT`: increase print level
- `-debug`: even more verbose
- `-trace`: print tracing info
- Run the input via debugger
- Run the input via tracing tool, e.g. valgrind
How to use gdb

• use `-g -ggdb` flags, or `-speed debug`: you get information about lines, not subroutines
• molcas $MOLCAS_DEBUGGER$=gdb input
• useful gdb commands: run, where, quit
• $MOLCAS_BOMB = YES$ : to generate exception.
• you can also use ddd (gdb with GUI)
• for parallel debugging - see the manual
Molcas programming guide

- Directory structure
- Building Molcas (in details)
- Patch system
- Verification
- Documentation
- Use of utilities
- Tools for development
- Coding rules
A new code in Molcas

- `molcas install`: interactive script to install a new code
- `configure` automatically creates Makefile
- Check out list:
  - return code: to communicate with other codes
  - prgm: to define file names, and attributes
  - documentation: to be included into the manual
  - XML documentation: to be used in help, and in GUI
Molcas has functions to access data from binary files, e.g. RunFile, and to perform ‘standard’ operations. Documented functions: `molcas help src`
An example

```fortran
subroutine my(ireturn)
  Real*8 COO(3,100)
  Call Get_Natoms_All(iAtom)
  write (6,* ) iAtom
  Call Get_Coord_All(COO,iAtom)
  Do J=1,iAtom
    write(6,'(3F10.3)') (COO(I,J),I=1,3)
  EndDo
end
```
subroutine my(ireturn)
include 'WrkSpc.inc'
Call Get_Natoms_All(iAtom)
write (6,* ) iAtom
Call Allocate_Work(ipCoo,iAtom*3)
Call Get_Coord_All(Work(ipCoo),iAtom)
write(6,'(3F10.3)')
   *(Work(ipCoo+i),i=0,iAtom*3-1)
Call Free_Work(ipCoo)
ireturn=0
end
More convenient use of dynamic memory

```fortran
subroutine my(ireturn)
  include 'WrkSpc.inc'
  Call Get_Natoms_All(iAtom)
  Call Allocate_Work(ipCoo,iAtom*3)
  Call Get_Coord_All(Work(ipCoo),iAtom)
  Call PrintCOORD(Work(ipCoo),iAtom)
  Call Free_Work(ipCoo)
end

subroutine PrintCOORD(COO,iAtom)
  Real*8 Coo(3,iAtom)
  Do J=1,iAtom
      write(6,'(3F10.3)') (COO(I,J),I=1,3)
  EndDo
end
```
Return codes

subroutine my(ireturn)
#include 'warnings.fh'

... Call Print_COORD(iRC,
... if(iRC.eq.0) then
    ireturn=_RC_ALL_IS_WELL_
else
    ireturn=_RC_INVOKED_OTHER_MODULE_
end
Files

subroutine my(ireturn)
LUnit=33
LUnit=isfreeunit(LUnit)
call molcas_open(LUnit, 'COORD')
end

data/my.prgm
(prgm) "$MOLCAS/bin/my.exe" executable
(file) COORD "$WorkDir/$Project.xyz" rwsg

s- save the file to $CurrDir,
g - register the file for GUI