MOLCAS for advanced users

Valera Veryazov

Overview

- How to run Molcas in an efficient way?
- Code development in molcas environment

Best computer for Molcas

Quantum chemistry has specific demands to hardware!

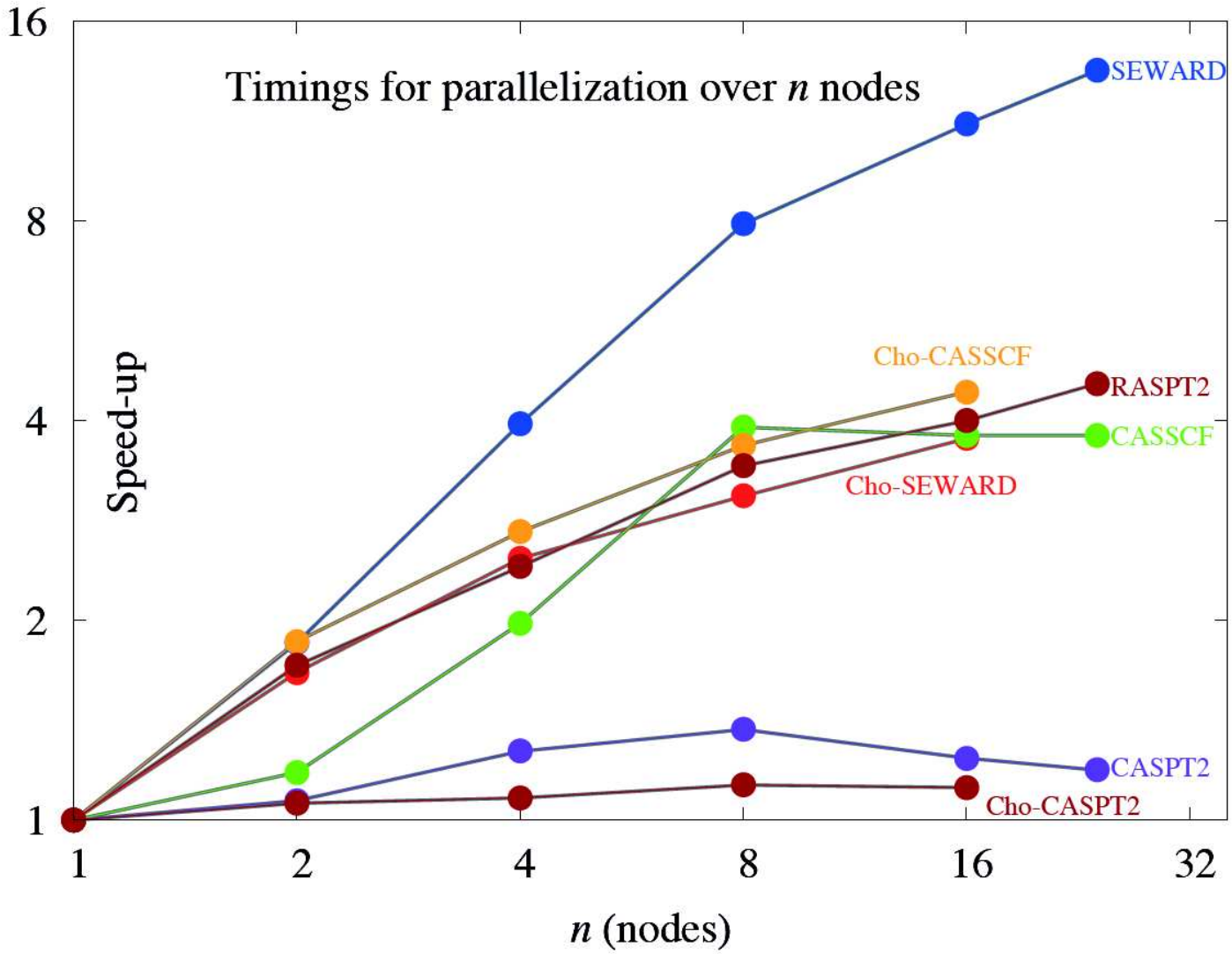
- more (!) RAM
- fast CPU/GPU
- large cache and bus speed
- fast HDD, or better Solid State Disk
- network and intercommunication

Cluster/SMP/Multicore

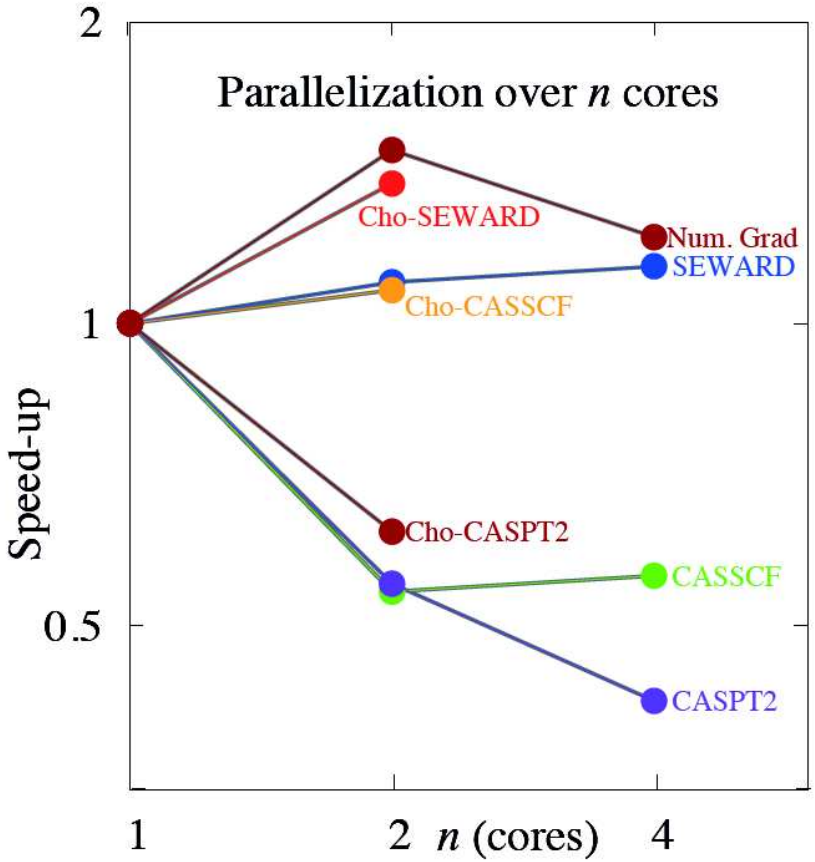
	RAM access	Disk	Hardware cost	Coding cost
Cluster	distributed	distributed	cheap	high
SMP	shared	external	expensive	low
Multicore	shared	shared	average	average

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

across nodes..



across cores..



Compilers (for Linux)

Name	free?	stable?	performance	parallel
g77	Yes	Yes	average	Yes
gfortran	Yes	Yes	good	Yes
g95	Yes	Yes	average	not tested
Intel	No	yes	excellent	Yes
SunStudio	Yes	yes	good	not tested
NAG	No	yes	good	some problems
Portland	No	yes	good	some problems

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
- do not overload MOLCASMEM !
- How to allocate memory for a parellel 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 API

Molcas has functions to access data from binary files, e.g. RunFile, and to perform 'standard' operations.
Documented functions: *molcas help src*

An example

```
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
```

Molcas Dynamic memory

```
      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 confinient use of dynamic memory

```
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 Print_COORD(Work(ipCoo),iAtom)
 Call Free_Work(ipCoo)
end
subroutine Print_COORD(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