1 Installation Guide for Columbus Version 7.1 with OpenMolcas Support (colmol-2.1.0)

1.1 Overview

Columbus and Molcas cooperate by exchanging well-defined, transferable quantities, specifically AO integrals, AO density matrices and MO coefficients in the native Molcas format by using library functions from the Molcas library. Additionally, the RunFile is processed, which is used by Molcas to store additional information often required to be passed in between different Molcas modules. Molcas provides some mechanism to incorporate external programs and making them accessible through the standard Molcas input file.

Since it is necessary to link a portion of the Molcas library, both Columbus and Molcas must be compiled and linked in a compatible way, that is to say (i) the **same** compilers and (ii) **compatible** compiler options. Hence, both codes cooperate on a binary level and do not rely on some file conversion utilities.

Please note, that the Molcas library is frequently restructured, so that files produced with a previous version may or may not be compatible with the current version of Molcas (cf. page 4).

Additionally, there are now two molcas driver utilities: `molcas.exe` in Molcas-8.2 developer version and `pymolcas` in the OpenMolcas version, which may not behave exactly the same way.

1.2 New Features

Primarily, rewritten versions of the SCF, MCSCF, AO-MO transformation and CI-gradient code are introduced. With exception of the CI gradient code, all codes are now parallel and make extensive use of shared memory and much more efficient intermediate data storage.

The same driver (`columbus.exe`) now also supports an old modified version of the dalton code (affecting primarily the integral and density matrix handling and storage format in the old HERMIT and ABACUS code portions). This allows to save a lot of disk space and disk band width while at the same time avoids I/O intense sort steps.

In combination with SEWARD/ALASKA corresponding modifications make little sense, since SEWARD/ALASKA have close ties to almost every module in molcas. To get rid of I/O and many unfortunate sorting steps, it is likely to be more profitable and to some extent easier to make use of the Cholesky Decomposition infrastructure. In a fairly obvious way this simplifies the MCSCF and AO-MO transformation code, however, with the given code structure there is no benefit for the CI code.

1.3 Limitations

Starting from colmol version 2.0, Columbus 7.1 is distributed in the form of a Docker image containing a fully operational version of Columbus in combination with OpenMolcas. This collection of codes contains the (adjusted) `OPENMOLCAS` binaries (including sources snapshot, installation instruction, git hash id of the OpenMolcas snap shot), and a subset of the suitably COLUMBUS binaries along with `columbus.exe`, a wrapper program that is accessed by the MOLCAS drivers `molcas.exe`, `pymolcas`.

The Docker image allows to run the same binaries across multiple Linux distributions and versions without running into technical runtime problems, most notably missing or incompatible libraries, kernels, environments etc.

Although the easiest way is to run this COLUMBUS-MOLCAS-build via Docker (which requires root access rights in order to start the image, hence, computer centres may not be delighted), it is equally straightforward to extract the binaries and run the binaries in the native environment provided you sort out any technical issues. In order to simplify many issues in this respect, The binaries are tied to shared library versions of MPICH and OpenBlas, which allows to replace these shared libraries by versions adjusted to your hardware and environment (for some hints see below). Since the Docker image is based upon Suse 42.3, it is likely to be almost without any effort to migrate the binaries from the Docker image to a Suse 42.3 based installation running them native.

1.4 Downloads

The specific version bundled with OpenMolcas is available from the Molcas website (www.molcas.org/Columbus) This version limited to parallel operation on a single node and serves as a good starting point to get accustomed to the code. With a modern system and 32 cores /128 GB per node, CSF spaces of a few hundred millions are accessible; here the code is primarily compute-bound. There is no registration required albeit productive use resulting in published scientific papers should cite Columbus appropriately (cf. below).

Upon registration suitable Docker images are available to painlessly rebuild the Columbus-7.1/Molcas/OpenMolcas interface with more recent versions of Molcas/OpenMolcas running across multiple nodes - however, they may have to be build specifically for a particular hardware setup (e.g. different MPI and compiler version etc.).

Finally, upon registration for Columbus and dalton, the available Docker images also include pre-configured binaries, that have full Columbus 7.1 functionality using either dalton, argos or molcas for integrals.

Note, that currently there is **no** full source code distribution of Columbus 7.1 not even with registration. For any version, except for the one deposited on the Molcas server, please contact me directly.

1.5 Binary Distributions

1.5.1 Installation of Docker

Practically all wide-spread distributions (OpenSuse, Ubuntu, Centos, ...) provide pre-configured Docker binaries that can be installed with the available distribution specific methods (e.g. yast, zypper, apt ...). In case of OpenSuse 42.3 the packages docker and containerd have to be installed and the docker and container daemon processes must be started.

1.5.2 Running the downloaded Docker image

After downloading `suse42.3:colmol_2.1.tar.gz` load it into the image repository (as root).

```
gunzip suse42.3:colmol_2.1.tar.gz
docker import suse42.3:colmol_2.1.tar suse42.3:colmol_2.1
```

The command `docker images` should now list this image (also):

```
REPOSITORY TAG IMAGE ID CREATED SIZE
suse42.3 colmol_2.1 2343e7e6c829 2 days ago 1.12GB
```

The practical use of this docker images poses a couple of restrictions:

- set the maximum memory to about 80% of the physically available memory e.g. 16GB (-m 16000000000)
- set the maximum shared memory to about 80% of the physically available shared memory (visible via `du -ks /dev/shm`), e.g. 12 GB (–shm-size 12000000000); the size of the shared memory on the host may be adjusted at runtime via `mount -o,remount,size=14000000000 /dev/shm` to have 14 GB shared memory.
- calculations are carried out from within the docker container but on a local fast hard disk or SSD; assuming that /SSD is a scratch directory mounted on a SSD, make this directory accessible from inside the docker container e.g. with the path /scratch (`-v /SSD:/scratch`)
- give the container some host name for easy identification (`-h colmol`)

There is a plethora of option with `docker run` - inspection may be helpful.

Execute the docker image via

```
docker run -v /SSD:/scratch -h colmol -m 16000000000 --shm-size 12000000000 -ti suse42.3:col
/bin/bash
```

You are now inside the docker container (as local root, i.e. a root that is distinct from root on the host!). The entire Columbus/Molcas related software stack is found in `/software`. `/software/bin` included in the default `$PATH`.

Switch to /scratch (i.e. to the local directory residing on your SSD) execute `runtests_colmol.sh -mintest.x` which should result in the message:

```
 {\tt
executing simple mpi test program (mpitest.x)
Process #   0
    hostname:colmol
    workdir:/scratch/scratch
Process #   1
    hostname:colmol
    workdir:/scratch/scratch
Process #   2
    hostname:colmol
    workdir:/scratch/scratch
Process #   3
    hostname:colmol
    workdir:/scratch/scratch
executing seriel test case 0
* Molecule: CH2
* Basis: cc-pvtz
* Symmetry: C2v
* Program Flow: Seward-Scf-Rasscf-columbus(mrci)  Single Point
*********** ALL TESTS PASSED (11) *************
executing parallel test case 0p
* Molecule: CH2
* Basis: cc-pvtz
* Symmetry: C2v
* Program Flow: Seward-Scf-Rasscf-columbus(mrci)  Single Point
*********** ALL TESTS PASSED (11) *************
}
```

1.6 Execution (inside the docker container)

Copy your input files (on the host) to `/SSD` (referring to the previous setup). Then (inside the docker containter) goto `/scratch` and run

```
runcolmol <project_name>
```

1.7 Requirements/Incompatabilities

Host dependencies boil down to the version of the docker runtime environment. The provided images should work on any system with docker xxx installed and a 3.x or 4.x kernel - it might also work on a LTS kernel 2.x (untested). Dependenicies on glibc, perl, bash and other stuff closely related to OS are absent.

Compatibility of pre-compiled Columbus binaries with a more recent Molcas/OpenMolcas version cannot be guaranteed. The docker images are occasionally updated and Columbus-related fixes are applied to the Molcas/OpenMolcas sources.

A serious cause of incompatability is usually the RUNFILE, since the basic size of the data items stored there is subject to change (e.g. March 2018: size of unique basis function names 10 bytes, Sept. 2018: same quantity, 14 bytes)

With a development image that contains the Columbus binaries in a different configuration, you may try to load and install the more recent OpenMolcas version into the container, i.e. you compile Molcas according to some prescription within the container. Similar applies to incorporating additional modules not initially available in the docker image.

1.8 Executation outside the docker container

Execute the distributed image within a docker container and copy (within the suggested setup) the entire /software directory to /scratch (`cp -a /software /scratch/`). Run `ldd /software/Columbus71/` lists the shared library dependencies of the code.

On the host, create a directory /software and copy the contents of /SSD/software to /software. Run `ldd /software/Columbus71/builds/emt64_ifort18_mpich_openblas/pciudg.x` Use this to adjust the LD_LIBRARY_PATH environment variable such that the same files as within the docker image are accessed. If the code crashes immediately after startup with segmentation violation or weird kernel/system library messages, linux-vdso.so.1 and/or /lib64/ld-linux-x86-64.so.2 and/or kernel version are incompatible. In this case you can run it only from inside a docker container.

1.9 Verification procedure

The `TESTS` subdirectory contains for each test case the input file (`*.input`), a reference output file (`*.output`) , an input file for the verification script (`*.config`) which contains the pair of file names to be tested for consistency along with a variety of tests in terms of a regex expressions. The output of each verification run can be found in `*.verifyls`.

1.9.1 Semantics of the configuration file

The configuration file is parsed line by line. Comment lines start with `#`, lines specifying the pair of files to be compared start by `%` and the pair of files including relative or absolute paths are separated by `:`. The following lines are considered as one test specification per line refering to the last specified pair of files. Hence, tests following another specification line for the pair of files refer to the latter.

Each test specification consists of 3 or 5 columns separated by `!`.

The first column specifies the regex pattern of the line(s) to be compared.

The second column specifies the column of the data in these lines to be tested (assuming separation of columns by white space).

The third column indicates the comparison threshold.

The pattern of the fourth and fifth column constrain the search for the pattern of column 1 to a range of lines starting with the first appearance of the pattern in column four and ending with the first subsequent appearance of the pattern in column five.

1.10 Trouble Shooting

COLUMBUS 7.1 differs considerable from its predecessor both in terms of technical implementation as well as in terms of algorithmical details. Hence, although all test cases pass the test criteria, only a subset of the possible usage combinations are covered.

Interoperability issues may arise if the COLUMBUS version is combined with your own MOLCAS version. Incompatibility arises here from (i) different (incompatible) file structure of your own MOLCAS version typically giving rise to message complaining about non-existing or unreadable files or (ii) possible inconsistencies when mixing binaries produced by different compilers.

1.11 Revision Log

r1.0.0 initial version open for testing

r1.0.1
- corrected -mpitest.x option
- corrected call of parallel Columbus codes in columbus.exe
- added support to read LUMORB format 2.0

r1.0.2
- added support for changes in the initialization of integral routines for 8.1 newer than March 2015
- added support for generalized slapaf bookkeeping handling single state optimizations; corresponding minimalistic changes to 8.1 sources commited on 15/12/08.
- depending on the molcas version string the codes switches between the usage of `tran.x` and `tran.x_81`.

r1.0.4
- Several bugfixes and optimizations in the Columbus part.
- Switched from mpich1.2.7 to mpich2-1.4.1.
  Note, that mpich1.2.7 is script based while mpich2-1.4.1 is partially based on binaries loading shared libraries. This might produce failures with older operating system distributions.
- The fix from feb 9,2016 prevents a failure due to the incorporated BLAS libraries on a system with avx2 registers and has missing links added to the mpich2/bin subdirectory.

r1.0.5
- Several bugfixes and optimizations in the Columbus part, especially fixing the `daio: maxrec exceeded`, due to an incorrect file size estimate (thanks to N. Bogdanov, to point out the problem and provision of a test case).

  - This version changes for some additional files to compression, thereby reducing the disk and memory space requirements. Per default the parallel CI code keeps everything in memory while the seriel CI code uses disk space, instead. Although, it is not yet moved in to the interface, there is the possibility to drastically reduce the memory consumption for the subspace vectors forcing them to disk (ciudgin: fileloc=1,0,0 to be added manually, currently) - for the parallel code, consider using a SSD then. Vice versa, the seriel code can be forced to run fully in memory (ciudgin: fileloc=1,1,1 to be added manually).
  - Early termination of the CI code due to incorrect evaluation of integral distribution size fixed.

r1.1.0 Major changes in the low-level operation in order to facilitate multi- and many-core operation:

  - Global Array Toolkit completely replaced by a small library directly utilizing shared memory functionality
  - global shared counter replaced by a faster scheme based on semaphores which does not suffer from saturation effects
  - new integral storage format for more efficient storage of integrals, densities and ci vectors reducing disk usage for extended systems by up to 90%
  - new parallelization strategy that largely decouples load balancing from the associated communication volume leading to a reduction of the network load by more than one order of magnitude

r1.3.0 Minor changes at program and interface level adding new features
Built on top of Molcas v8.0.16-08-07, Intel Compiler+MKL 14.0.2.144

  - all internally used locks (in /dev/shm) are now properly deleted
  - support added for single point excited state calculations with the MR-AQCC, MR-ACPF, MR-AQCC-v and LRT-MRAQCC methods
  - support to extract multiple electronic states of possibly different symmetry within a single COLUMBUS section; this also supports multiple electronic states to be computed with state-specific methods (MR-AQCC, MR-ACPF, MR-AQCC-v)
  - support to specify the reference state for the LRT-MRAQCC method and to subsequently use this method for multiple electronic states of possibly different symmetry
  - support for excited state structure optimizations for state-specific methods as well as LRT-MRAQCC

Examples test30p through test37p illustrate some of these features.

r2.1.0 Major changes at program and interface level
Built on top of OpenMolcas Mar 11 09:36:12 2018, Intel Compiler 18.0.2.199, OpenBlas 0.3.0

  - faster and parallel MCSCF code added (shared memory intense)

- faster and parallel AO-MO integral transformation code added (shared memory intense)
- faster and parallel SCF code added (shared memory intense)
- faster CIGRD code added (shared memory intense)
- minor fix to alaska applied (fixes incorrect pickup of gradients)
- MCSCF, SCF and MRCI code now directly produce density matrices, expectation values, symmetric and antisymmetric transition densities (if applicable) and corresponding expectation values
- direct support of frozen-core and frozen-virtual orbitals added to the mcscf code.
- in combination with dalton, large molecules (e.g. 500 basis functions, no symmetry, using frozen core orbitals) are tractable within acceptable turn-around times. Cf. to the /software/Columbus/builds/BENCHMARKS subdir.
- BLAS and MPI support through shared libraries

Jülich, September 11, 2018