Dos and don’ts when using parallelization in Molcas

A multitasking field guide

Valera Veryazov & Steven Vancoillie

User’s perspective

Valera Veryazov

Is parallel execution faster?

1. Amdahl's law
   - If 25% of the code runs 10 times faster…
   - If 90% of walltime is spent in code that can be perfectly parallelized: max 10x speed-up on about 32-64 processes
2. Overheads related to initialization and data transfer
3. Sharing resources
   - I/O (shared by all processes on a node)
   - Memory (shared by processes on a socket)
4. In many scientific projects the task parallelism is already used.

Only real stories from users..

1. Help, Test 000 runs slower in parallel !!
2. Test runs 3 min 20 sec instead of 3 min 10 sec
3. RASSI does not run faster in parallel
4. “mpirun -n 2 molcas input” does not work
5. Parallel run on a fancy cluster is too slow
6. I requested more Memory, and it runs slower
7. parnell takes too long time
8. No convergence in RASSCF if run in parallel
9. No performance improvement due to GPU
   (to be explained!)

Parallel dictionary

Software

Processes

MOLCAS_NPROCS

Threads

OMP_NUM_THREADS

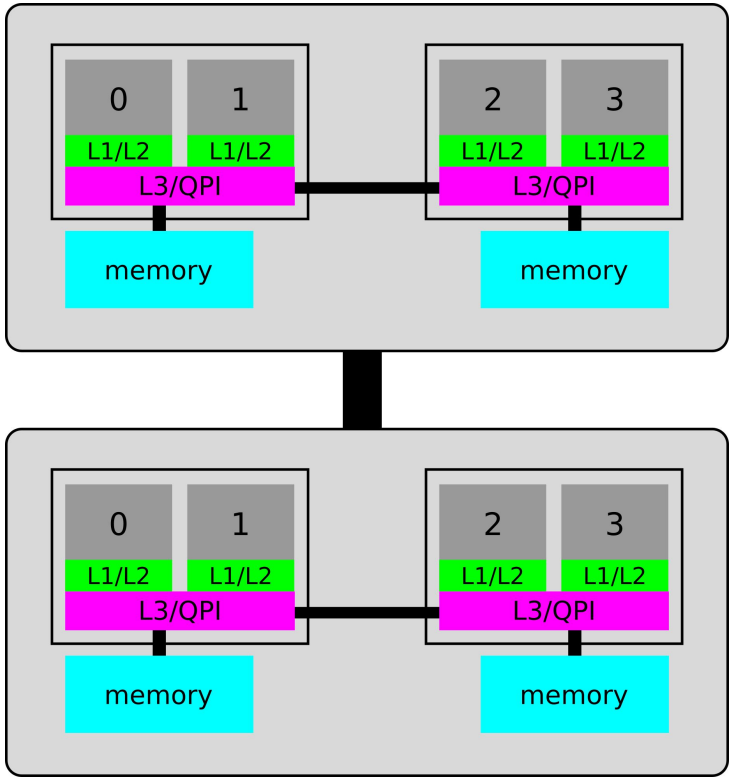
OS

Hardware

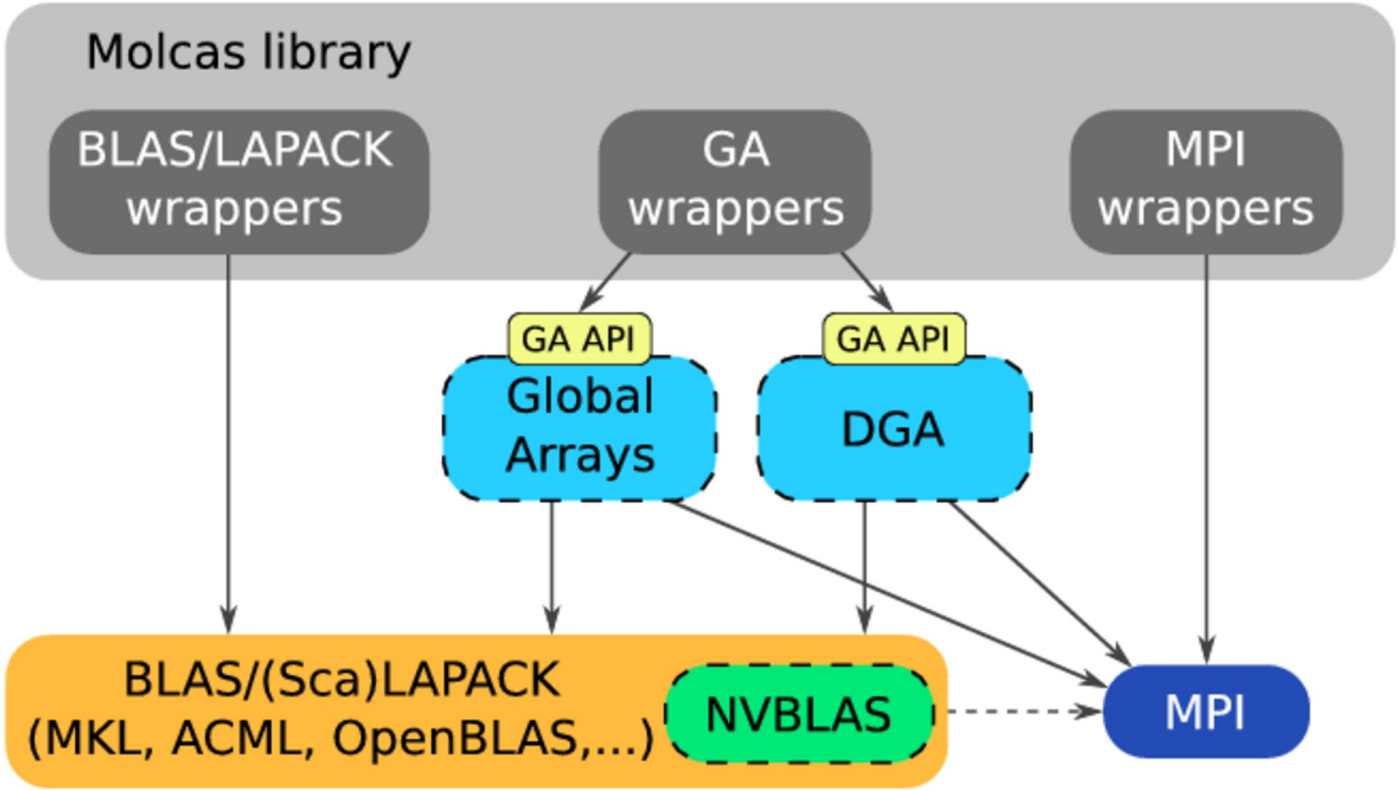
Nodes

Sockets

Cores



Molcas software stack



Choices in configuration

- MPI?
  - OpenMPI
  - MPICH
  - Intel MPI
- OpenMP?
  - Use threaded BLAS/LAPACK library? (-OMP | -DOPENMP=ON)
- GA?
  - Internal GA implementation fully based on MPI? (| -DGA=OFF)
  - External GA implementation? (-garoot | -DGA=ON, export GAROOT=...)
- NVBLAS?

Running in parallel

Don’t try ‘mpirun -n 2 molcas input’ or ‘mpirun -n 2 bin/molcas.exe input’

Everything is determined by the file “**molcas.rte**”. Check the **RUNBINARY** variable inside the file to customize how you launch programs.
Default: **RUNBINARY=’mpirun -n $MOLCAS_NPROCS $program’**

The number of processes is controlled by **MOLCAS_NPROCS**, while the number of threads is controlled by the variable **OMP_NUM_THREADS**.

The number of processes/threads is printed in the header!

Nesting MPI processes

MPI_INIT can be called only once.

So, it is not possible to run a parallel code from a parallel code.

Ex 1. numerical gradients: even though it runs different geometries in parallel, the single-point energy calculations are run in serial.

Taskfarm is still under construction

Ex 2. Running of multithreaded BLAS from parallel code: if you set MOLCAS_NPROCS=4 (for CPU with 4 cores), BLAS library is running in serial.

Choosing #processes x #threads

A computer with 4 cores and 8 Gb RAM

MOLCAS_NPROCS can be set as 1, 2, 3, 4.

MKL always run in serial, if it is called from parallel environment.

Other BLAS, like OpenBLAS, might make a wrong decision based on OMP_NUM_THREADS.

So, running with MOLCAS_NPROCS=1 (but using parallel BLAS) might be the best alternative. Also, think about MOLCAS_MEM.

Memory and running in parallel

`MOLCAS_MEM` is per process!!

So, for a computer with 4Gb RAM and 6 cores usage MOLCAS_NPROCS=6 and MOLCAS_MEM=3Gb means that Molcas will use up to 6*3=18Gb of RAM.

The calculation will run, but astonishingly slow!

The rule of thumb: never use more than 75% of RAM.

Parnell and parallel infrastructure

If Molcas run in serial - all WorkDirs are accessible for the driver (so, files can be transferred to/from WorkDir by UNIX commands)

In parallel execution WorkDirs *could* be inaccessible. The file transfer is made by parnell program. This should not take up too much time.

Parallel WorkDirs have names tmp_XX/

tmp_XX/ directories can be accessed by all processes (shared disk) or not.

All files are locally accessed in tmp_XX/ (unless ‘lustre’ option is used).

Output of slaves is diverged to a local stdout file. It might be a good idea to check tmp_XX/stdout files

Can a parallel run give different numbers?

Yes, it can!

1. Different order of arithmetic operations
2. Numerical noise in iterative procedures (especially RASSCF)
3. Different algorithms

Our verification is tested only for MOLCAS_NPROCS=2

DGA vs. GA

1. License issue
2. DGA is designed for small and medium size parallelization
3. DGA does not buffer communications (works poor for badly written codes)
4. DGA is small and transparent

EMIL commands

Purpose: user should not think about ‘parallel environment’

Currently relative to “$WorkDir”, but might change… USE ABSOLUTE PATH

```
>> COPY $CurrDir/startorb $WorkDir/
>> CLONE $WorkDir/file1 $WorkDir/file2
>> SAVE $WorkDir/$Project.RasOrb $CurrDir/
>> COLLECT $WorkDir/stdout $CurrDir/


>> SHELL echo “hello world”
>> EXEC /usr/bin/echo hello
```

Only real stories..(explained)

1. Help, Test 000 runs slower in parallel !!
2. Test runs 3 min 20 sec instead of 3 min 10 sec
3. RASSI does not run faster in parallel
4. “mpirun -n 2 molcas input” does not work
5. Parallel run on a fancy cluster is too slow
6. I requested more Memory, and it runs slower
7. parnell takes too long time
8. No convergence in RASSCF if run in parallel
9. No performance improvement due to GPU

Help, Test 000 runs slower in parallel !!

For testing purpose use tests running hours (for a single executable), not seconds. Running test000 you a) initiate MPI, b) running parnell (to transfer files between nodes) - after each executable.

Test runs 3 min 20 sec instead of 3 min 10 sec

1. There is no point to benchmark small tests
2. Note that the scaling might be different in real runs.
3. Analyze which program (executable) runs slower

RASSI does not run faster in parallel

Yes, not all programs in Molcas are parallel. Running RASSI you do it in a replica mode. Consult the manual for details.

Note that in so-called supermodules, e.g. numerical gradients each code runs in serial.

“mpirun -n 2 molcas input” does not work

It is the best way to print Molcas logo twice :)

Check molcas.rte file to see the actual mpirun command.

Parallel run on a fancy cluster is too slow

“A fancy” cluster can have strange MPI library which is not supported in the best way by GA/DGA.

“A fancy” cluster might have a network storage.. Check the physical location of your WorkDir.

I requested more Memory, and it runs slower

> Yes, MOLCAS_MEM is set for each process, so in some moment the physical memory is exhausted, and the calculation is performed via swapping..

parnell takes too long time

Parnell is a simple tool to execute commands remotely or transfer files between nodes. It should not take any time. And if it does it means that there is a problem, e.g. in communication between nodes. Or a problem with MPI library.

No convergence in RASSCF if run in parallel

Not all algorithms in Molcas are numerically stable.
Several advices can be given:

- Try to change BLAS library (especially if you use MKL)
- Stop the calculation earlier
- Try to clean the orbitals by symmetry

No performance improvement due to GPU

GPU (currently) used in Molcas only for some BLAS calls. So, the size of the matrices should be large enough to gain from using GPUs.

Developer’s perspective

Steven Vancoillie

Planning parallelization of your module

- Could I benefit from parallelism?
  - Identify typical usages of the code (kinds of small/large jobs?)
  - Profile the code to get a view of bottlenecks
    - detailed view: valgrind (run with MOLCAS_DEBUGGER=...)
    - coarser view: gperftools (compile with -DGPERFTOOLS=ON)
- Which parallel scheme to use?
  - BLAS/LAPACK: no action needed, link to threaded versions
  - CPU-bound:
    - MPI-based simple parallel loop scheme (see later)
    - OpenMP for low-level speed-ups
  - Memory-bound: distribute array(s) in memory (GA)

Processes versus threads

Process: executing instance of a program

Thread: thing that can be scheduled for execution

Different threads can exist within one process: they share process resources.

A process consists of at least 1 thread.

Threads can run simultaneously on different cores.

mpirun launches a bunch of processes, they have 1 or more threads.

Writing/Reading

- ‘**`write(6,`** ‘ for slaves writes output to **`tmp_XX/stdout`** file
- **`$WorkDir`** is different (and always translated to **`$WorkDir/tmp_XX`**)
- **`$CurrDir`** is the same
- Writing to the same file is dangerous
- Write to **`$WorkDir`**, and collect/distribute files after (use EMIL commands)
- For the reading of the same information for all slaves - developer might consider ‘lustre’ option

Parallel modes

There is a module “**numerical_gradients**” that uses MPI parallelism to distribute serial calculations. Unfortunately this interferes with the way parallelism is implemented.

The way Molcas deals with this is to distinguish between ***a “true” work-sharing parallel mode***, and ***a “fake” non-work-sharing mode***. When numerical gradients calls another module, it first switches to the latter fake mode, calls the module, then comes back and enters the former true mode.

There are some differences between parallel modes that affect the operation of your parallelisation. The function IS_REAL_PAR() can tell you if you are in real or fake parallel mode. The function KING() will always tell you if a process is the absolute master process, regardless of the mode!

Unless your code needs to be aware of the mode you are in, **ALWAYS** use MyRank and nProcs, as these are guaranteed to be 0 and 1 inside fake parallel.

Parallelization: global variables

The user-side variables are made available through:

```
#include “para_info.fh”
   -> Integer MyRank (serial/NG: always 0)
   -> Integer nProcs (serial/NG: always 1)
   -> Logical King (NG: only .True. for master!)
   -> Logical Is_Real_Par (NG: .False.)
```

Serial code: excluding other processes

If you want to run something only on the master process that does ***no communication***(!), one should not use the left solution, it's a typical legacy anti-pattern that should be avoided.

```
#include "para_info.fh"
#ifdef _MOLCAS_MPP_
      If (Is_Real_Par()) Then
        If (KING()) Then
         ...
        End If
      Else
        ...
      End If
#else
      ...
#endif
```

```
#include "para_info.fh"
      If (MyRank.EQ.0) Then
         ...
      End If
```

Parallel code: splitting a loop

A simple parallel code will typically make decisions based on the **process rank**!

```
INTEGER :: I
REAL*8 :: A(N)

DO I=1,N
  A(I)=expensive_function(I)
END DO
```

```
#include “para_info.fh”
      INTEGER :: I
      REAL*8 :: A(N)
      CALL DZERO(A,N)
      DO I=1+MYRANK,N,NPROCS
        A(I)=expensive_function(I)
      END DO
      CALL GADSUM(A,N)
```

Parallel code: task list

Tasks with (wildly) varying cost

```
#include “para_info.fh”

      INTEGER :: I
      REAL*8 :: A(N)
      CALL DZERO(A,N)

      DO I=1+MYRANK,N,NPROCS
        A(I)=expensive_function(I**4)
      END DO

      CALL GADSUM(A,N)
```

```
#include “para_info.fh”
      LOGICAL, EXTERNAL :: RSV_TSK
      INTEGER :: TASKLIST, I
      REAL*8 :: A(N)
      CALL DZERO(A,N)
      CALL Init_Tsk(TASKLIST,N)
111   If (.NOT.Rsv_Tsk(TASKLIST,I)) GOTO 999
      A(I)=expensive_function(I**4)
      GOTO 111
999   CALL Free_Tsk(TASKLIST)
      CALL GADSUM(A,N)
```

Parallel code: distributed data

If you want to run something in parallel that calls GA_ (or MPI) directly, you need to use the _MOLCAS_MPP_ preprocessor macro to prevent linking to these routines for a serial installation. However, you also need to take into account numerical_gradient code!!

```
#include “para_info.fh”
#ifdef _MOLCAS_MPP_
      Call GA_Subroutine()
#else
      …
#endif
```

```
#include “para_info.fh”
#ifdef _MOLCAS_MPP_
      If (Is_Real_Par()) Then
        Call GA_Subroutine()
      Else
        …
      End If
#else
      …
#endif
```

Avoiding (unwanted) parallelism

When a code is NOT parallelized, it can be advantageous to run it on a single process only. The reason is usual that memory and I/O contention will be less and the program will be faster than if it had to run in a replicate fashion.

```
Call start('module')
Call module()
Call finish('module')
```

```
#include "para_info.fh"
      Call start('module')
      IF(MYRANK.EQ.0) Call module()
      Call finish('module')
```

Two potential problems to be aware of:

1. your module subroutine calls some communication routine (e.g. through the Molcas library). Maybe there is a need for a clear identification of parallelized Molcas library interfaces!!
2. output data is stored in file(s) that need to be available to other modules (e.g. rasscf JobIph needed by each caspt2 process).

Things to be read from PG (programmer's guide)….

```
>>export MOLCAS_NAP=90

How to attach PID to GDB?

How to profile the code?

How to use callgrind/kcachegrind
```