Parallel IO on HLRN-IV systems

Klaus Ketelsen

13. April 2021

Inhaltsverzeichnis

1 Introduction

The following report describes the findings that were gained when implementing the parallel IO in the PALM model (https://palm.muk.uni-hannover.de). PALM uses NetCDF for model output and native MPI-IO for restart handling.

Parallel IO on Lustre file systems is a very complex topic and this report will not be able to handle all aspects. But maybe the experiences can help to improve the IO performance in other areas.

2 Compiler and environment

Most of the time measurements below were done with the default Compiler and MPI modules, which currently are

intel/19.0.5(default) (Berlin) or intel/18.0.5(default) (Göttingen)

impi/2018.5(default) (Berlin) or impi/2018.5(default) (Göttingen)

It should be noted that with impi/2018.5 the Lustre drivers are not loaded by default. The following environment setup increases the IO performance:

```
export I_MPI_EXTRA_FILESYSTEM=on
export I_MPI_EXTRA_FILESYSTEM_LIST=lustre
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$I_MPI_ROOT/lib64
```

With impi/2019.x the following Environment should be set:

```
# export I_MPI_EXTRA_FILESYSTEM=on
# export I_MPI_EXTRA_FILESYSTEM_FORCE=lustre
```

Table 1 shows the transfer speed for writing a file with different setups. The job ran on 16 nodes and the stripe-count was set to 16.

For unknown reason, the performance values in table 1 could only be reached, when the compiler and impi modules are loaded in the compile script and not in .bashrc.

table 1: Io-Performance depending on compiler and runtime setup

Compiler	MPI	Environmet set	Transfer speed
intel/18.0.6	impi/2018.5	no	6499 MB/s
intel/18.0.6	impi/2018.5	yes	9938 MB/s
intel/19.0.5	impi/2019.9	no	6323 MB/s
intel/19.0.5	impi/2019.9	yes	9756 MB/s

3 Striping and stripe size

The following measurements were done with a standalone test program based on PALM IO-routines. The parallel IO is done with native MPI-IO, no NetCDF or hdf5 has been used.

On Lustre file systems, striping of the parallel IO streams has significant influence of the IO Performance.

The following variables change the setup of Lustre striping for single files or directories with their containing files.

1. stripe-count

   The stripe count indicates, how many streams are used in parallel for MPI-IO.

   The stipe count can be set by:

      lfs setstripe --stripe-count *count-value file-name* (or *directory name)*

2. stripe-size

   The stripe size indicates the size of striped IO block.

   The stipe size can be set by:

      lfs setstripe --stripe-size *size-value file-name* (or *directory name)*

Table 2 and figure 1 show the influence of stripe count and strips size on the transfer rate. Measurement were done with the test program described above.

transfer direction:	write
global size of 3D array	1536*1536*512

The transfer rates in the following table are in MB/s *table 2: Influence of striping to IO-Performance*

stripe count	size 0.5 m	size 1m	size 2m	size 4m	size 8m
1	807	810	864	868	867
2	1590	1465	1524	1498	1592
4	2736	2809	2868	2864	2971
8	4732	5103	5322	5150	5523
16	8803	9262	9437	9860	10459
24	12053	12693	13233	13667	13970

The following graphic shows the transfer rate depending on the stripe-count for stripe-size of 0.5 MB and 8 MB

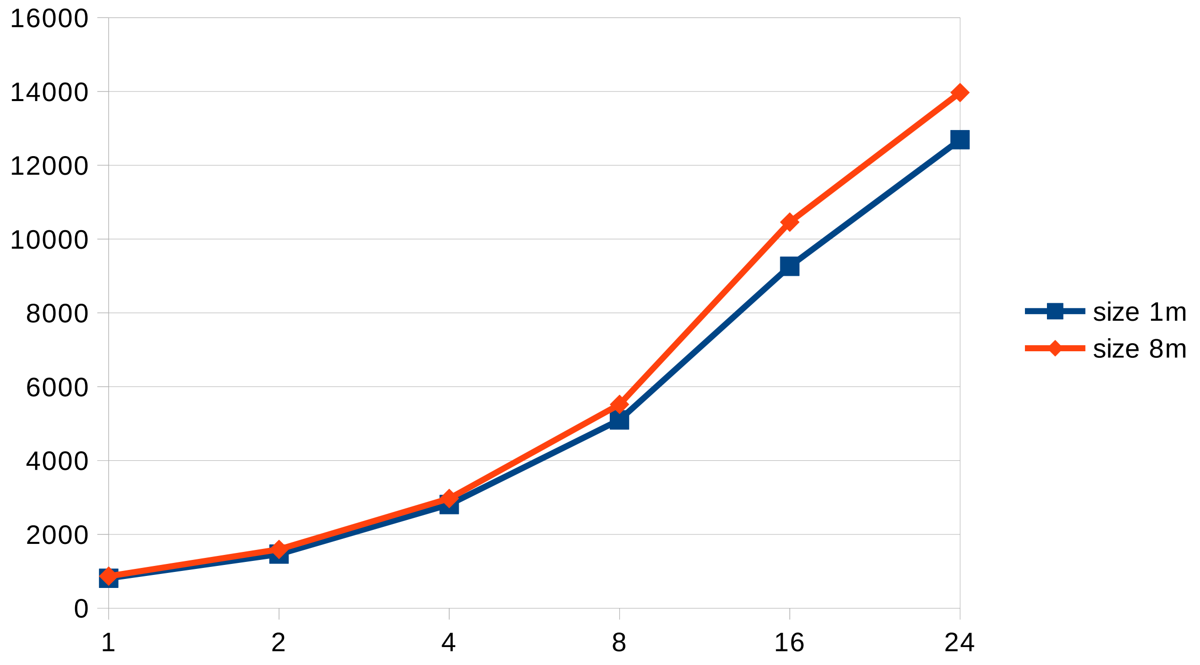

figure 1: IO Performace vs stripe-count

Table and graphic show significant speedup increasing the stripe-count, whereas the effect of changing the stripe-size is minor.

4 Influence of data layout for MPi-IO

MPI-IO is very sensitive of the data layout during parallel read or write.

Performance hints

1. Write or read big chunks
2. Locally, try to get largest chunks in the first dimension of the output array.
3. If possible use *MPI_File_write_all* (collective IO) instead of *MPI_File_write (non collective IO)*
4. No gaps and no overlay areas in the global IO array
5. Setup IO arrays with *MPI_Type_create_subarray*

Table 3 compares the IO performance of runs with collective and non collective IO routines.

*table 3: Collective or non-collective IO: Array Dimension (1536*1536*512)*

nr. nodes	Tasks/node	Nr. procs	Direction	Collective	time sec	IO-Rate MB/sec
16	96	1536	write	NO	**98**	2067
16	96	1536	write	YES	**44**	4614
16	96	1536	read	NO	**110**	1841
16	96	1536	read	YES	**54**	3714

Besides 2-D and 3-D arrays, there is a special datatype surface data in PALM. Surface elements are stored in 1-D indexed arrays, i.e. there is a grid-point based index array, which points to the elements of the respective grid-point in that 1-D element array. This compares with unstructured mesh in other models.

Surface data can have a very irregular structure, e.g. walls of buildings.

To handle surface data IO conform with the performance hints above, the following procedure has been implemented.

1. Transform the local index space into global index space
2. Compute the total number of surface elements over all MPI processes.
3. Divide the global 1-d array of surface elements in equal size chunks.
4. Distribute these chunks across the MPI processes
5. Move the surface elements to their designed output processes using MPI RMA communication

6. Write the elements to disk using *MPI_File_write_all*

Figure 2 shows how the surface elements are transferred from the PALM context to the IO Context and vice versa. On the left side, surface elements are only located on processes 2 and 3 (e.g.building walls). On the right side, the same surface elements are equally distributed on all MPI processes.

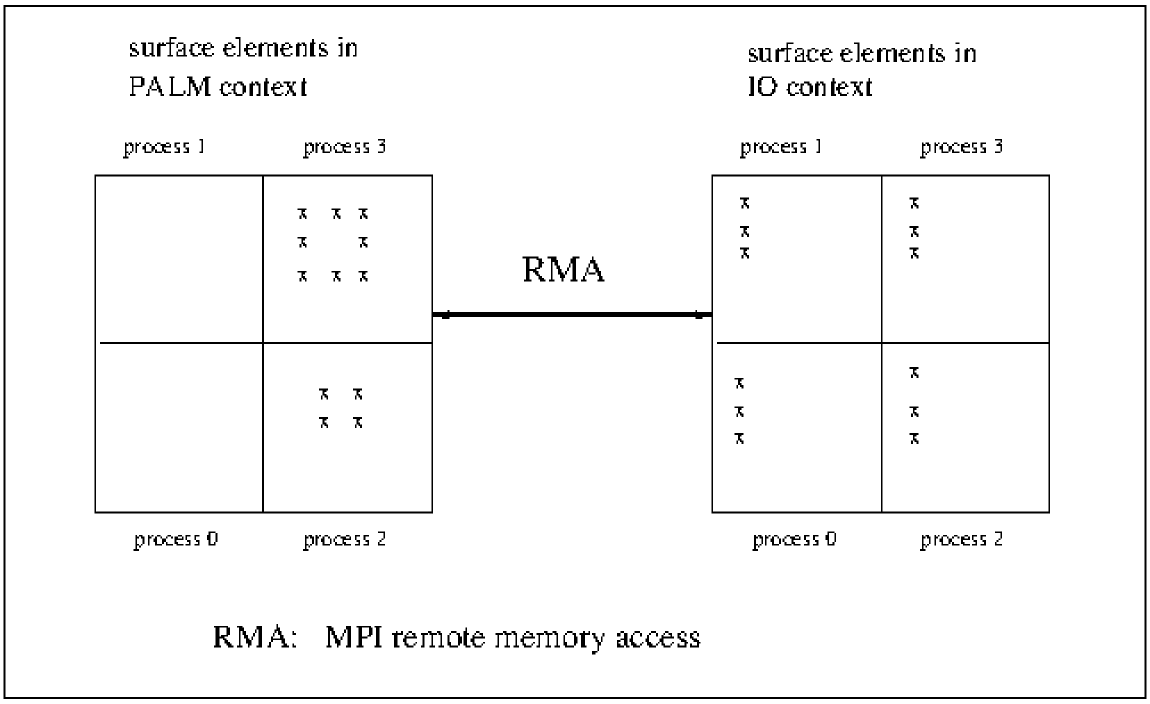
surface elements in PALM context
surface elements in IO context
process 1
process 3
process 0
process 2
RMA
RMA: MPI remote memory access

figure 2: reorder surface elemnts for IO

Due to the irregular data structure, all attempts to write the data only to the local processes had been unsuccessful. With the method above, good overall IO performance was achieved. Another advantage of this IO scheme is that the restart data does not depend on the processor grid, i.e. the restart file can be read in a different processor layout than when writing.

5 Reduced number of IO nodes

Timing measurements show, that especially on the HLRN-IV system in Berlin the IO-rate decreases when all processes of a node are involved in IO. To speedup IO, a procedure has been developed where IO is done only on reduced number of processes on the nodes. The IO processes access the output data via MPI shared memory from the other processes.

Basic Idea:

1. On a node, create one or more groups of MPI-processes
2. Process 0 of the respective group is designed as IO processes
3. Create a communicator for all IO processes (comm_io)
4. Only IO processes open file with *MPI_File_open* using the IO communicator.
5. All processes share the input/output buffer of their respective IO process.
6. actions while writing one record:
   - all processes fill output buffer shared with their respective IO-process
   - Synchronize IO group
   - The IO processes write using MPI_File_write_all on the IO communicator
   - Synchronize IO group
7. actions while reading:
   - Synchronize IO group
   - The IO processes read using MPI_File_read_all on the IO communicator
   - Synchronize IO group
   - all processes can access data from input buffer on their respective IO-process
8. IO processes close file

Figure 3 shows the process layout at an example of 2 nodes. The data array is shown as example on the first IO group on every node, it exists of course an all IO groups.

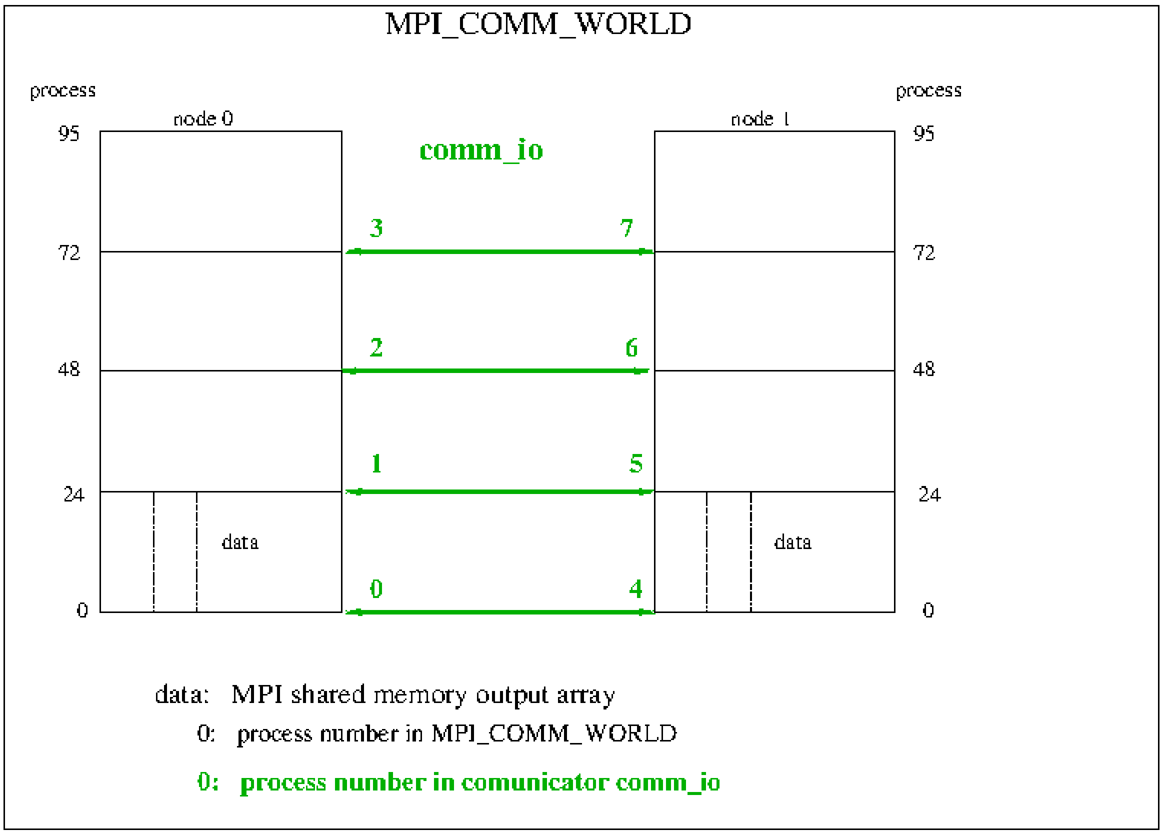
MPI_COMM_WORLD
process
node 0
95
72
48
24
0
comm_io
3
7
2
6
1
5
0
4
node 1
process
95
72
48
24
0
data
data
data: MPI shared memory output array
0: process number in MPI_COMM_WORLD
0: process number in comunicator comm_io

figure 3: process distibution for reduced IO processes

For more details of the implementation of IO on reduced number of processes please contact the author.

klaus-ketelsen@t-online.de

Table 4 and 5 show the IO times for writing and reading restart file for a PALM production run. Especially for reading there is significant speedup. In addition to regular 3-D arrays, many indexed surface arrays are written in this actual PALM setup.

writing

*table 4: IO-rates: Array Dimension (2940x2940*232), File size ca. 390 GB*

nr. nodes	Tasks/node	Nr. procs	striping_ factor	reduced IO nodes	time sec	IO-Rate MB/sec
59	96	5612	8	NO	**189**	2014
59	96	5612	8	YES	**71.2**	5362

reading

*table 5: IO-rates: Array Dimension (2940x2940*232), File size ca. 390 GB*

nr. nodes	Tasks/node	Nr. procs	striping_ factor	reduced IO nodes	time sec	IO-Rate MB/sec
59	96	5612	8	NO	**562**	670
59	96	5612	8	YES	**60**	6363

Two effects may influence the behavior of this method:

1. There are less MPI processes using the limited IO ports simultaneously.
2. Using fewer IO processes automatically increases the chuck size on the IO processes.

At this point of time it is not clear, which of the two effects has larger influence of the better performance using reduced IO processes.

6 NetCDF

Many models, especially in environmental research use NetCDF for storing results. The latest flavor NETCDF4 internally uses hdf5

The knowledge from the previous chapters can also be applied to NetCDF-IO.

6.1 Unlimited dimension

Most models do the output of their results in NetCDF format. Typically, the output is done within the simulation time loop and it is common and convenient to declare the time dimension as UNLIMITED in NetCDF context.

Unfortunately, the performance drops significantly, using unlimited time dimension compared with the equivalent IO with fixed dimension (see table 6).

*table 6: Compare limited and unlimited time dimension: Array Dimension (1536*1536*512)*

nr. nodes	Tasks/node	Nr. procs	striping_ factor	Unlimited dimension	time sec	IO-Rate MB/sec
16	96	1536	8	YES	**161**	848
16	96	1536	8	NO	**41**	3332

In PALM, the following workaround is implemented. The number of output steps is calculated in advance at the beginning of a model run. Knowing the number of output step allows to declare fixed time dimension.

6.2 Compile and link

It is strongly recommended to use the same modules (intel Compiler and impi) for compiling the model and the hdf5/NetCDF libraries. Otherwise, the results may be unpredictable. Even when compiling went OK, the program may abort at runtime.

For example, the PALM model aborts when loading modules intel/19.0.5 and impi/2019.5 and using hdf5/NetCDF generated with impi/2018.

7 Conclusion

This report shows that there are many methods that can affect the performance of the Lustre filesystem. The findings are based on experiences with parallel IO of the PALM model. Some can be used without changing the source code and it is recommended to check to what extent they can be applied when using another model on the HLRN-IV systems.