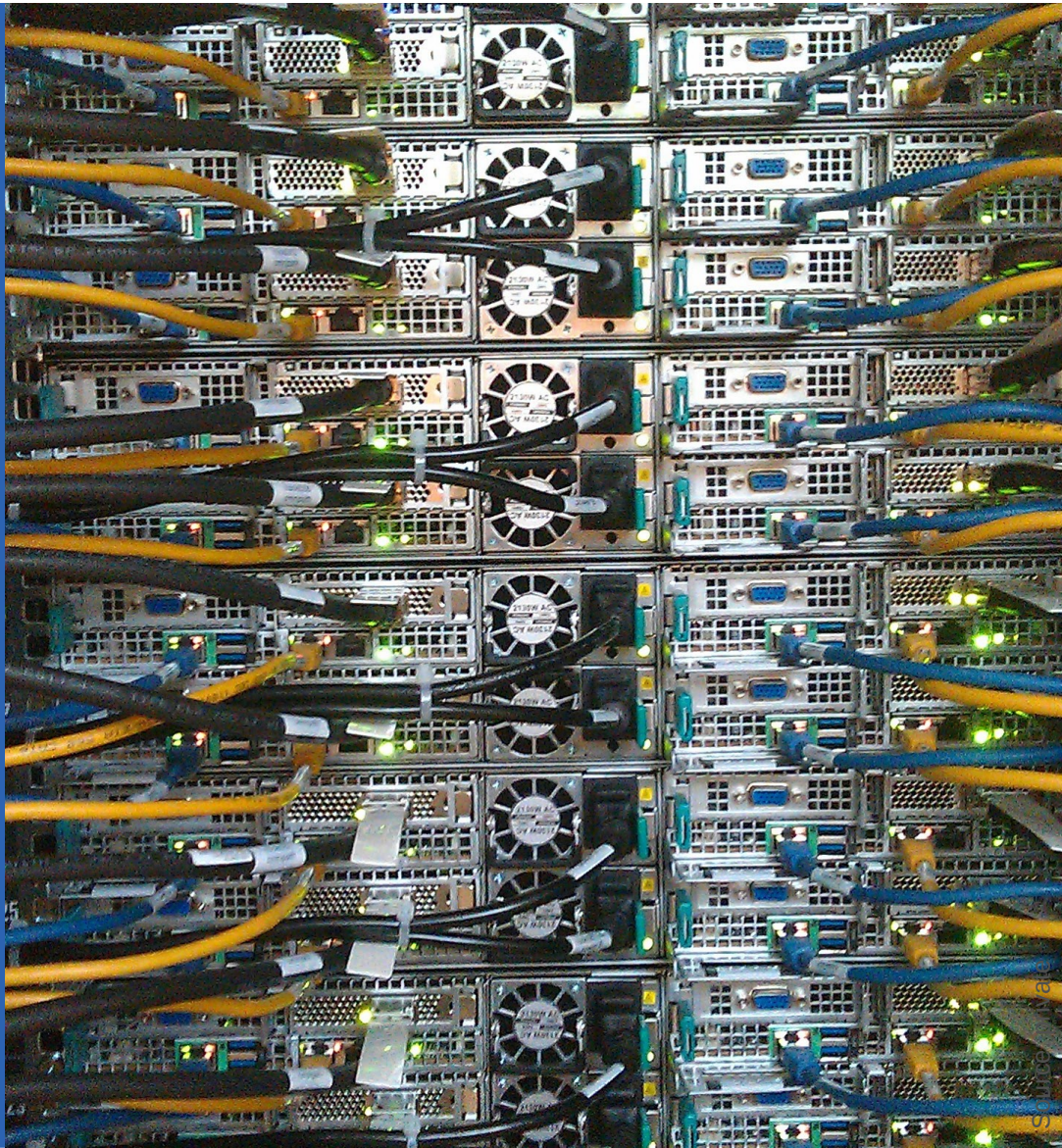
BAW
Federal Waterways Engineering and Research Institute

Jacek A. Jankowski

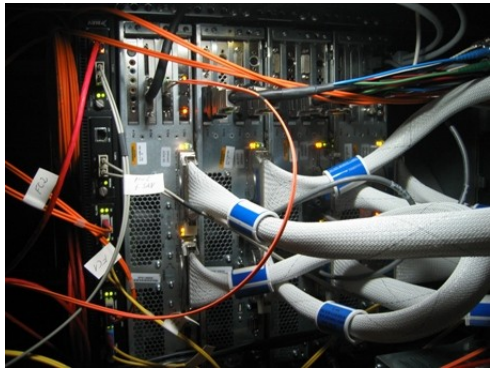
Parallel IO for parallel UnTRIM

16th UnTRIM Workshop

Trento, 13-15 May 2019



Content



1. A repetitive historical retrospective
2. The domain decomposition and message passing
3. IO: Input and Output in partitioned domains
4. Solutions for native User Interface files [2010]
5. Desire paths – will the users be happy with our designs?
6. Specific solutions for netCDF4 / HDF5 [2018]
7. Decisions to be made

Main efforts in the past

UnTRIM parallelisation with message passing, 2nd workshop, Sirmione, 23-25 May **2005**

UnTRIM parallelisation with message passing, 3rd workshop, Trento, 15-17 May **2006**

Parallel streamline tracking for UnTRIM, 4th workshop, Trento, 7-9 May **2007**

Verification and validation of the parallel UnTRIM, 5th workshop, Trento, 19-21 May **2008**




One paper, one report

INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS
Int. J. Numer. Meth. Fluids 2009; **59**:1157–1179
Published online 5 August 2008 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/fld.1859

Parallel implementation of a non-hydrostatic model for free surface flows with semi-Lagrangian advection treatment

Jacek A. Jankowski*,†

Department of Hydraulic Engineering in Inland Areas, Federal Waterways Engineering and Research Institute (BAW), Kussmaulstr. 17, Karlsruhe 76187, Germany

SUMMARY

The parallel implementation of an unstructured-grid, three-dimensional, semi-implicit finite difference and finite volume model for the free surface Navier–Stokes equations (UnTRIM) is presented and discussed. The new developments are aimed to make the code available for high-performance computing in order to address larger, complex problems in environmental free surface flows. The parallelization is based on the mesh partitioning method and message passing and has been achieved without negatively affecting any of the advantageous properties of the serial code, such as its robustness, accuracy and efficiency. The key issue is a new, autonomous parallel streamline backtracking algorithm, which allows using semi-Lagrangian methods in decomposed meshes without compromising the scalability of the code. The implementation has been carefully verified not only with simple, abstract test cases illustrating the application domain of the code but also with advanced, high-resolution models presently applied for research and engineering projects. The scheme performance and accuracy aspects are researched and discussed. Copyright © 2008 John Wiley & Sons, Ltd.

Received 28 January 2008; Revised 7 April 2008; Accepted 4 May 2008

KEY WORDS: parallel; free surface; non-hydrostatic; semi-Lagrangian; unstructured grid; UnTRIM

1. INTRODUCTION

High-performance computing for modelling of environmental flows is often regarded as a highly specialized domain associated with large supercomputer facilities and requires specific technical know-how. The appearance of widely available multi-core processors changes this situation, making parallel computing readily affordable and awaking a broader interest in the high-performance computing issues. In order to profit from computational resources delivered by modern parallel computers of any kind or size, an additional, specific effort is required from the code developers. Unfortunately, presently only in exceptional cases hydrodynamic codes are designed as parallel

*Correspondence to: Jacek A. Jankowski, Bundesanstalt für Wasserbau, Postfach 21 02 53, Karlsruhe 76152, Germany.
†E-mail: jacek.jankowski@baw.de

Int.J.Num.Meth. in Fluids 2009; 59:1157–1179

BAW Bundesanstalt für Wasserbau
Kompetenz für die Wasserstraßen

Technical Report

Mathematical Model UnTRIM

MPI Version Manual

– Version February 2019 (1.2) –

[BAW Technical Report]

Applications & Developments 2007-2011

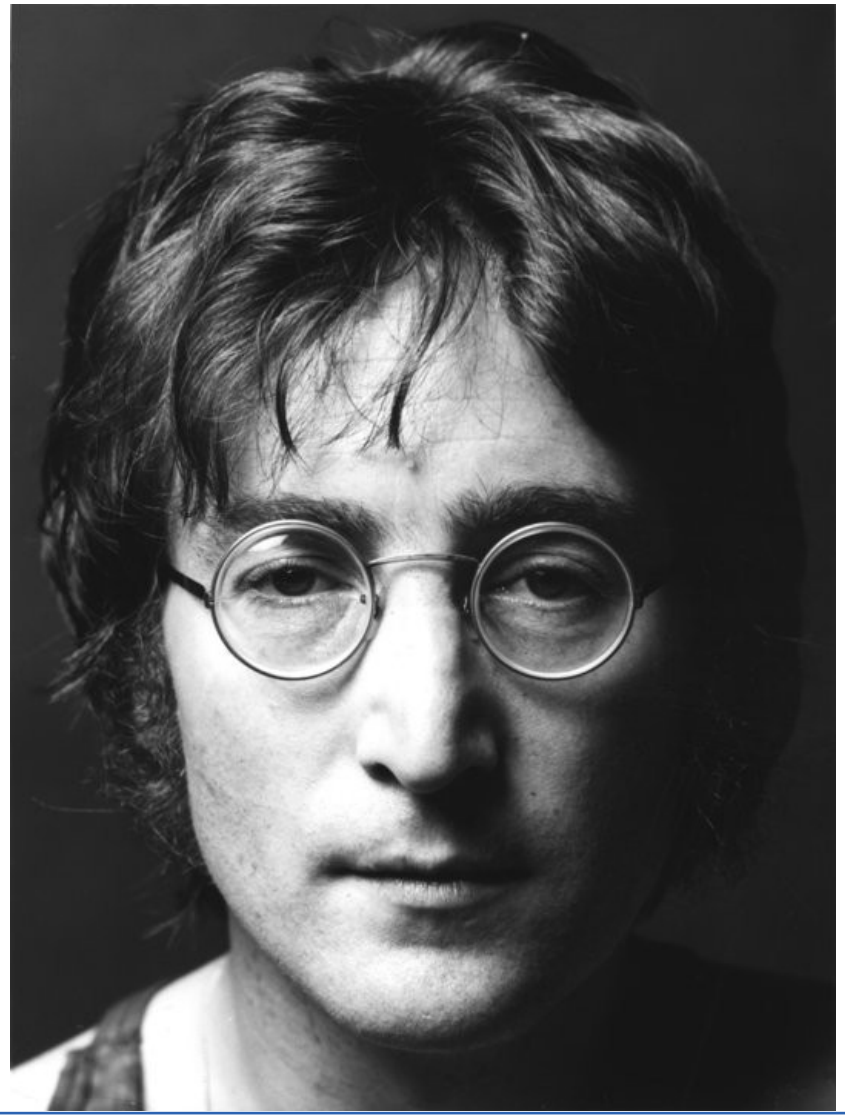
- The parallelized kernel with User Library specifically adapted for river engineering started to be applied since ca. 2007

- Quite a number of practical applications, just to name:
  - The Elbe River – stretches by Coswig, Lenzen, Hitzacker, Geesthacht…
  - The Danube River – Mühlhamer Loop, 80km-stretch, the Isar confluence…
  - The Saale River

- Additional developments in this period concerned
  - the scope of physical processes included (roughness, turbulence…)
  - pre- and post-processing, mesh treatment, parallel I/O (not applied)

John Lennon knew it well

Life is what happens
when you are busy
making other plans.



[Source: Facebook]

Applications & Developments 2011-today

- Developments:
  - working with GPUs, MICs, vectorisation, MPI and parallel I/O in UnTRIM
  - *(Un)TRIM-like* 2D-schemes with GPUs 2010-11
  - MPI-parallelised UnTRIM2 (subgrids) spring 2013 (and a study 2016)

- Politics 2013/14:
  - Decision to limit the methodological scope by inland waterways *to Telemac only*

- Initiatives to profit from the past developments:
  - UnTRIM2 or UnTRIM3 to be MPI-parallelized „soon“ (embedded in PROGHOME)

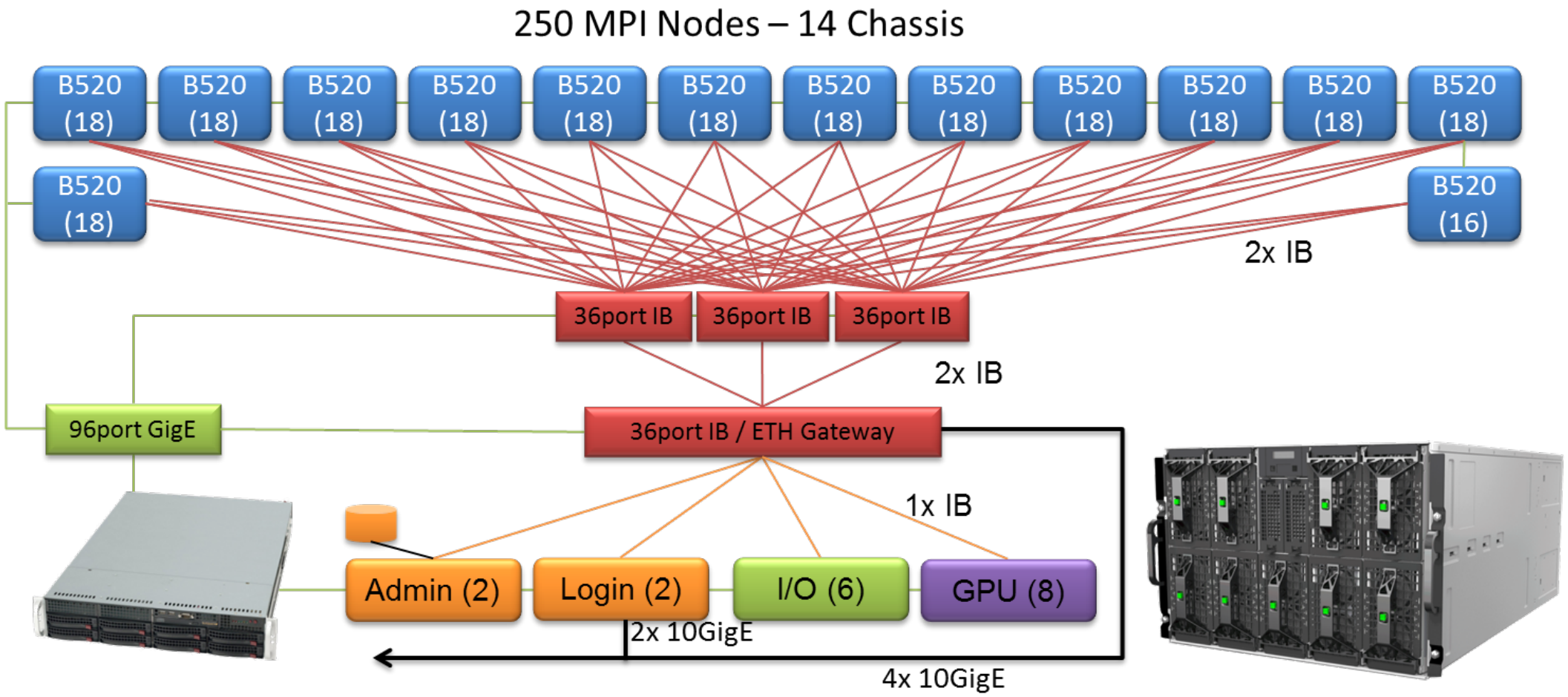
Parallel IO for UnTRIM: three principal aims

- Reach possibly efficient IO using *high performance computing* capabilites
- Apply *professionally acclaimed* data formats
- Convince users and other developers *to endorse* these solutions

Supercomputers availability: a privilege or a duty?

- Easy come, easy go…?
- Mission: use the hardware to promote the most promising scientific solutions
- Temptation: use the hardware power to speed up outdated but *well-validated* schemes

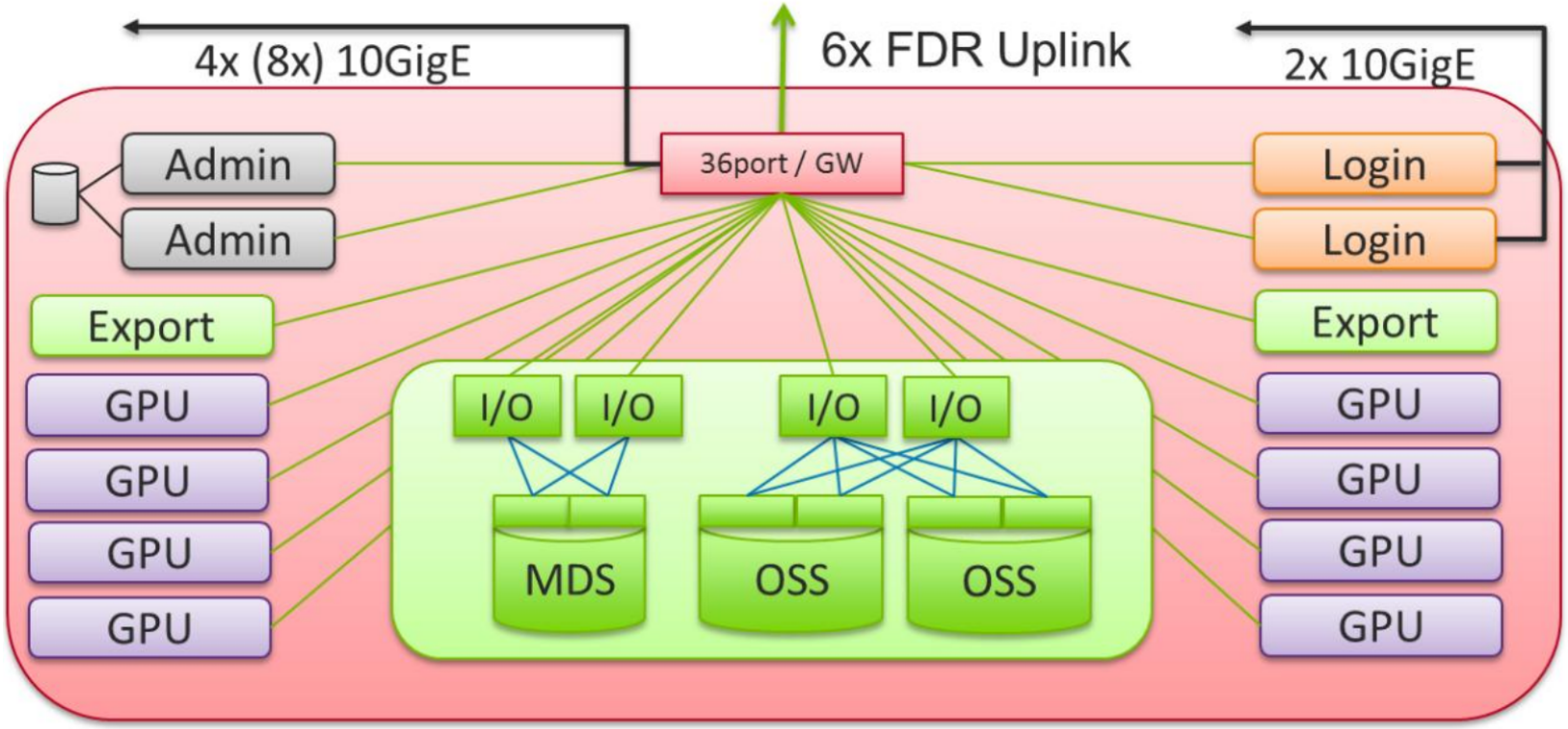
A parallel file system in the cluster configuration



Source: BAW/Bull *Betriebshandbuch* for the server hera

Periphery of a compute cluster

- *Lustre* parallel FS
- Requests from clients sent to **Meta Data Server** MDS
- MDS informs where to work on **Object Storage Servers** OSS
- Clients read/write form OSSs directly



Source: Bull's introductory lecture for hera cluster users

Structure of a file stored in a parallel file system

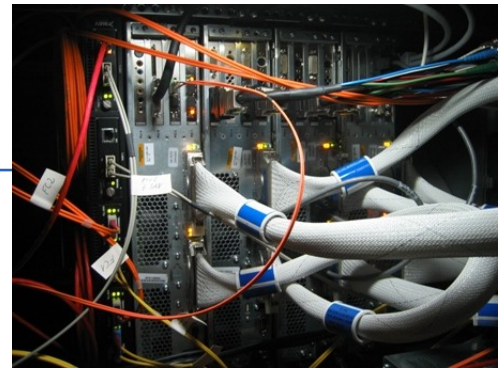
```
jaj@hera2:/lustre/w4/jaj/netcdf_performance_onefile > ls -lah file.nc
-rw------- 1 jaj w4 105G 10. Mai 11:01 file.nc

jaj@hera2:/lustre/w4/jaj/netcdf_performance_onefile > lfs getstripe file.nc

file.nc
lmm_stripe_count:   20
lmm_stripe_size:    1048576
lmm_pattern:        1
lmm_layout_gen:     0
lmm_stripe_offset:  19
        obdidx           objid          objid          group
            19       258222734      0xf642a8e              0
             0       258767839      0xf6c7bdf              0
            10       258232121      0xf644f39              0
             1       258789885      0xf6cd1fd              0
            11       258209578      0xf63f72a              0
             2       258769733      0xf6c8345              0
            12       258210165      0xf63f975              0
             3       258757519      0xf6c538f              0
            13       258217110      0xf641496              0
             4       258769218      0xf6c8142              0
            14       258207507      0xf63ef13              0
             5       258784108      0xf6cbb6c              0
            15       258228457      0xf6440e9              0
             6       258769122      0xf6c80e2              0
            16       258227340      0xf643c8c              0
             7       258779668      0xf6caa14              0
            17       258217147      0xf6414bb              0
             8       258132841      0xf62cb69              0
            18       258219105      0xf641c61              0
             9       258124474      0xf62aaba              0
```

file.nc, 105GB

MDS provides meta-data about the file

Data stored in 20 objects, addresses on OSSs

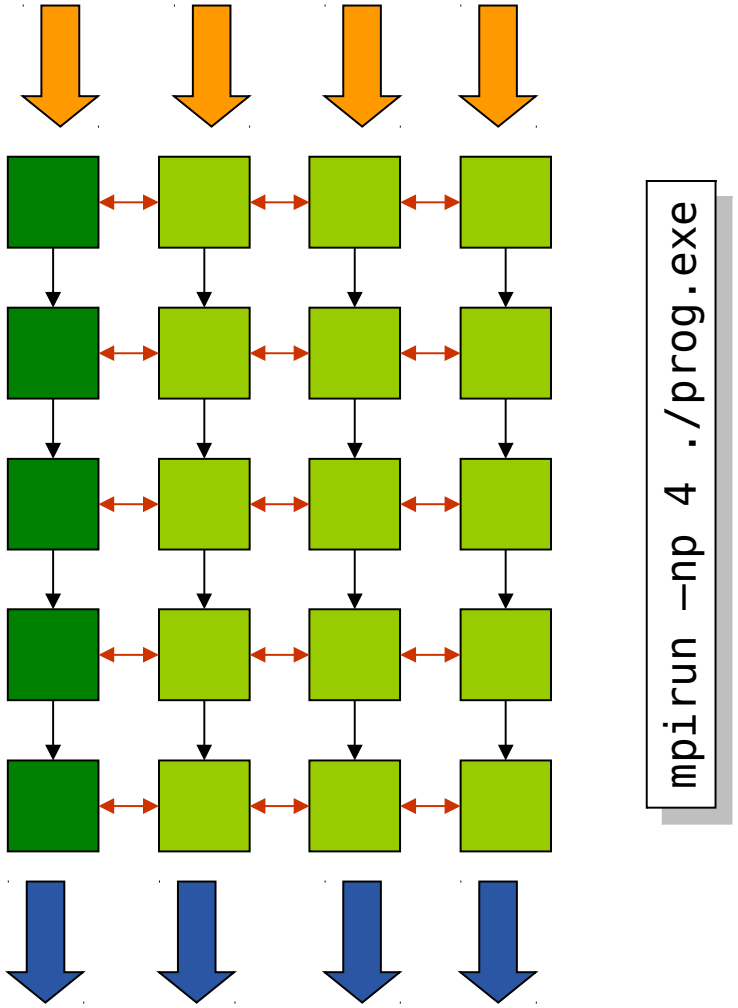
Custom made for a parallel machine



- **Domain decomposition method** – partitioning input data and merging output
- **Message-passing** – efficiency, synchronous or asynchronous communication
- **Standard swapping communication** – for ~90% of the effort
- **Tailored „hand-made“ solutions for** –
  - explicit diffusion operator – assembling contributions
  - Lagrangian advection scheme – streamline backtracking over partitions, global communication
- **Verification, validation** – evidence of serial results conformity and speedup
- Some **node-level optimizations**

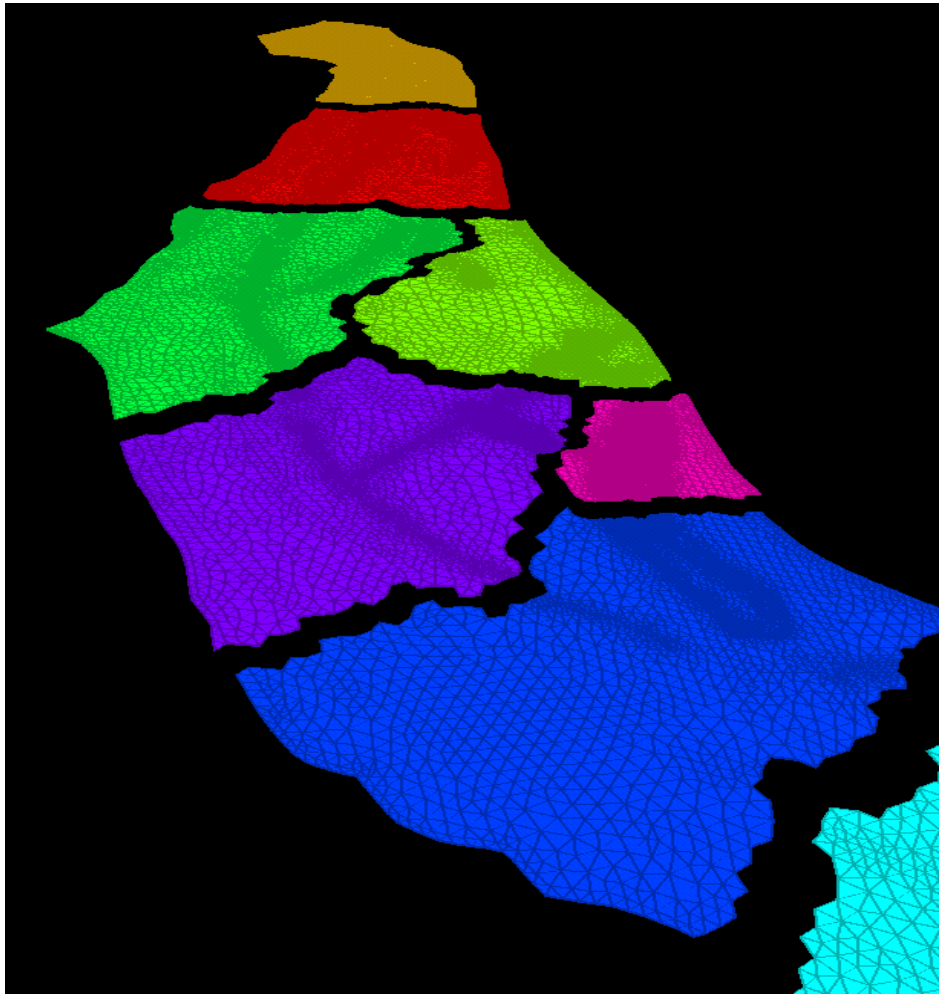
Message-passing parallelism

- Each rank executes a program copy *with its own data*
- Communication limits the scalability of the code:
  - preparing data for sending
  - communication *itself*
  - integrating the received data
- Each rank can perform its very own input and output
- Other solutions influence scalability (until MPI-IO arrived)



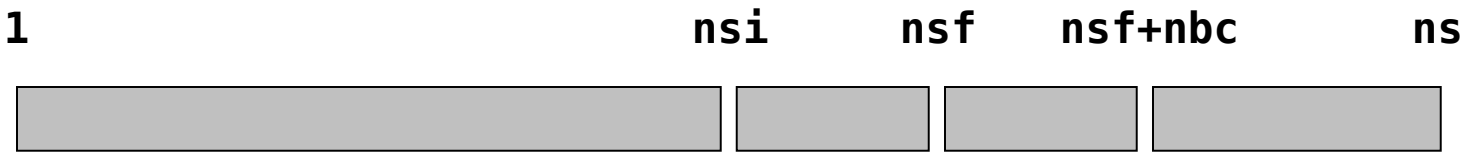
Domain decomposition method

- Parallel implementation with domain decomposition and **overlapping** mesh partitions
- Each mesh partition must be a correctly prepared mesh for each separate rank
- Partitioning results in awkward relationships between global and local sorting of mesh objects

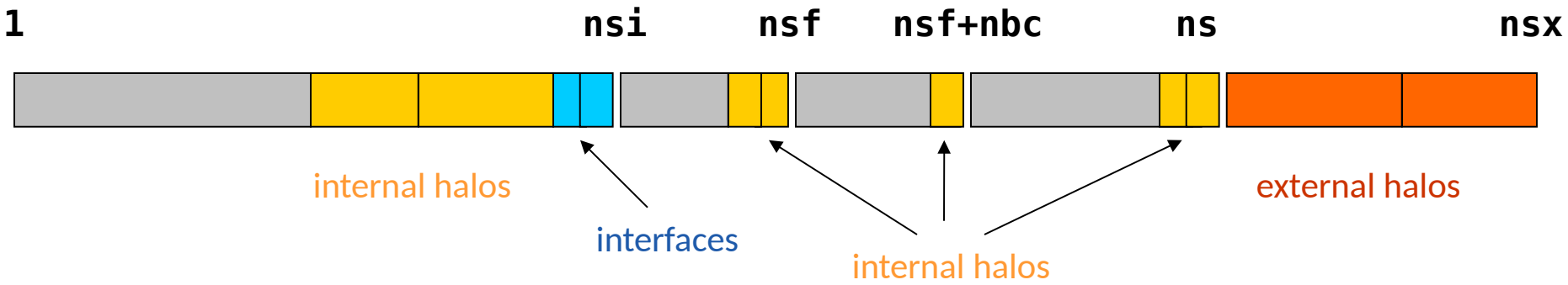


Extended fields: edges – ranges define the role played

```
ALLOCATE (u(1:ns))
```

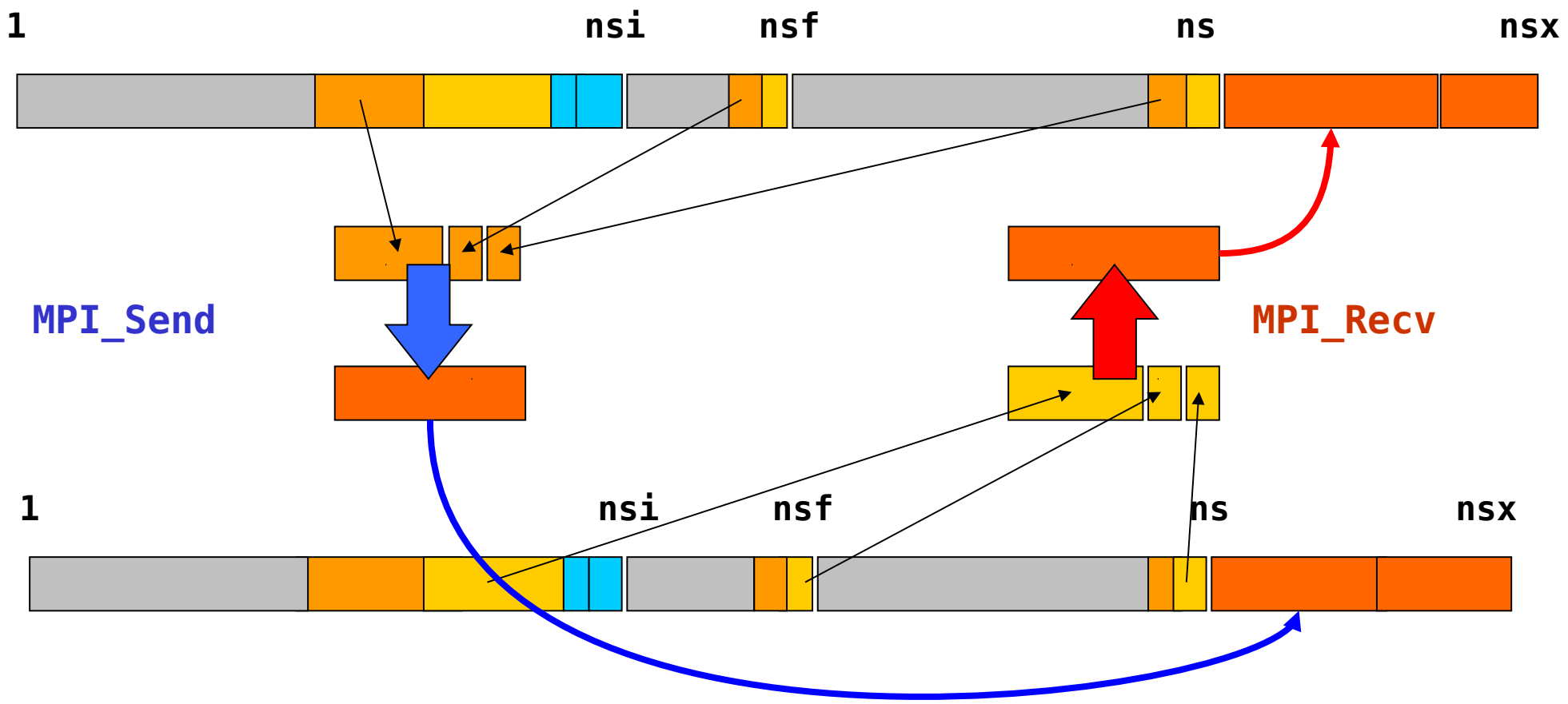



```
ALLOCATE (u(1:nsx))
```

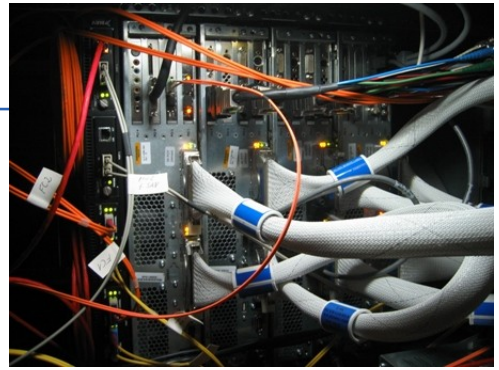


For 3D: `ALLOCATE (u(1:n3sx))`

MPI_SendRecv with buffers

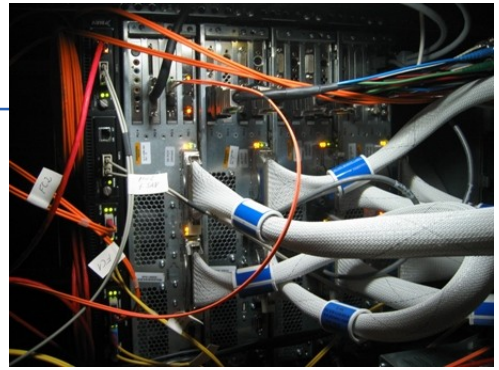


Why MPI-IO?



- Operational systems per default assume only one process can access a file at a time
- Traditional method of treating input/output in message passing programs:
  - Each rank works on his very own set of files
- Modern file systems “like” a small amount of files, but arbitrarily large
- MPI 2 allows ordered IO-operations of a set of ranks accessing together a single and usually very large file – MPI-IO
  - different implementations, e.g. ROMIO

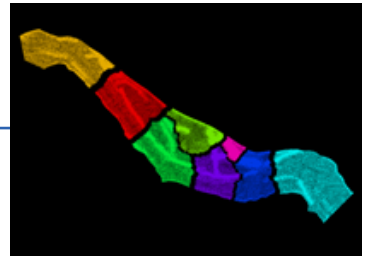
Why MPI -IO?



- The historical domain decomposition method trouble: large amount of small files – or worse, in subdirectories – counter-productive for HPC FS, awkward for users – stemming from pioneer clusters with separate disks

- MPI I/O: basic technology for parallel IO
- Uses cons data description – similar to message passing
- Optimizations for large consecutive data transfers
- “Hints” for applying specific hardware features
- Stages of merging and restart partitioning reduced

The threefold trouble

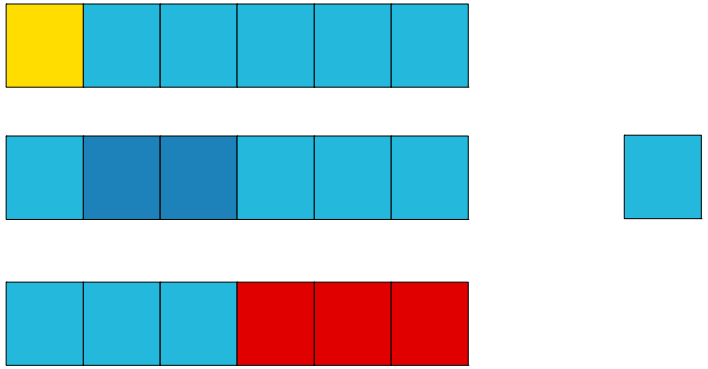


- Mesh partitions overlap with inner and outer halos
  - One has to set a hierarchy of sub-domains
  - The ever existing trouble with interfaces
- Partitioning brings awkward relationships between global and local sorting of mesh objects
  - Can be treated by the resorting of the original global mesh in the sequence of partitions, BUT:
- UnTRIM sorts specific BC edges, cells into defined index ranges
  - This must be done in the mesh partitions as well…

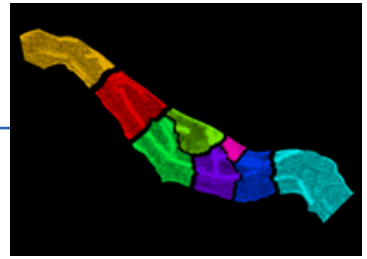
MPI-IO for UnTRIM – 2010 workshop

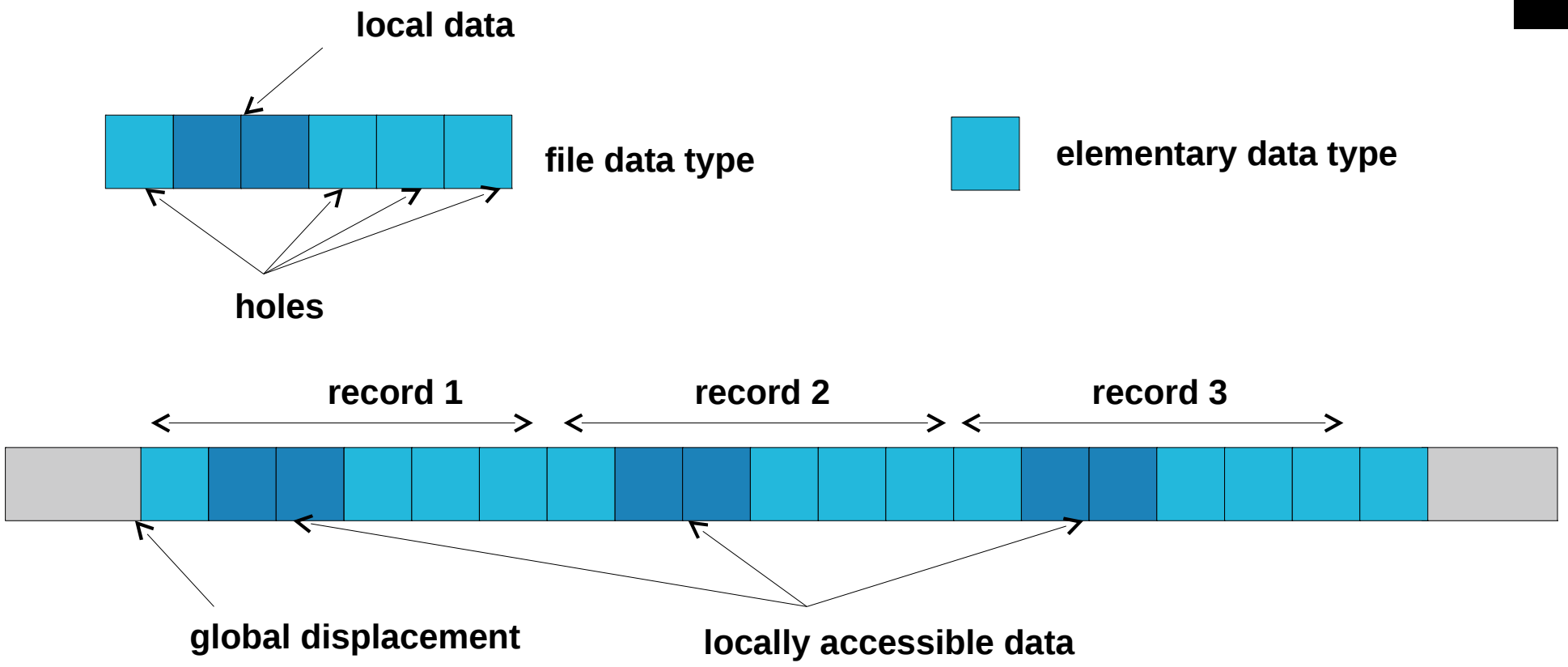
Improving and optimising UnTRIM MPI Library, 7th workshop, Trento, 3-5 May **2010**

- **Parallel I/O with MPI**
- (Improving communication itself)

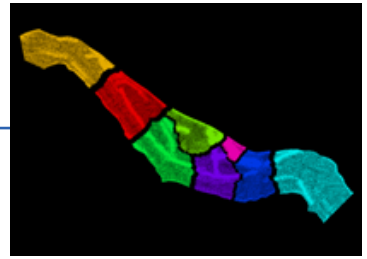


Single rank file view – writing record by record

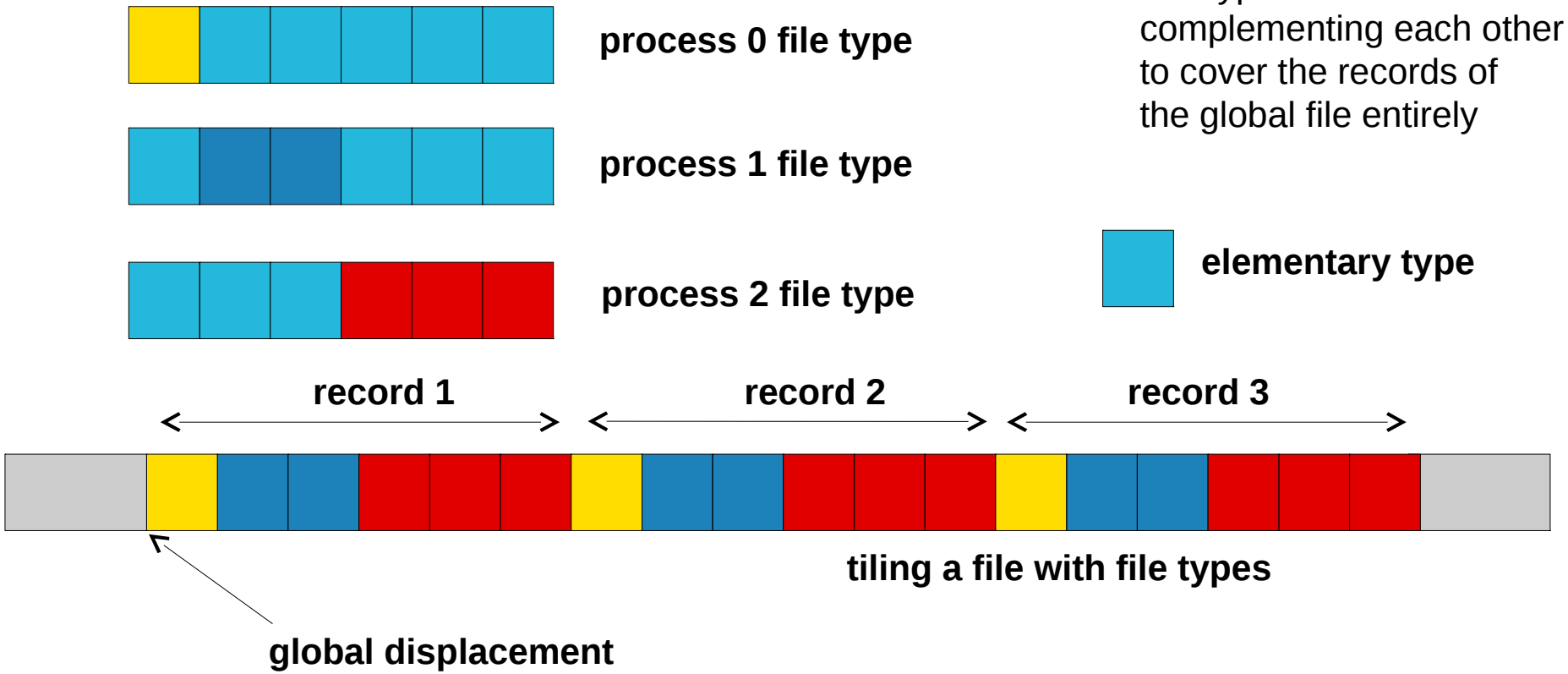




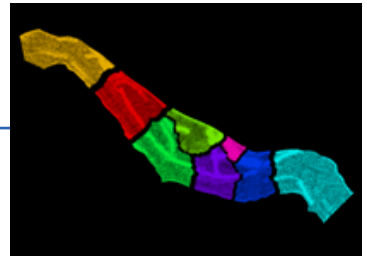
Complementary file types



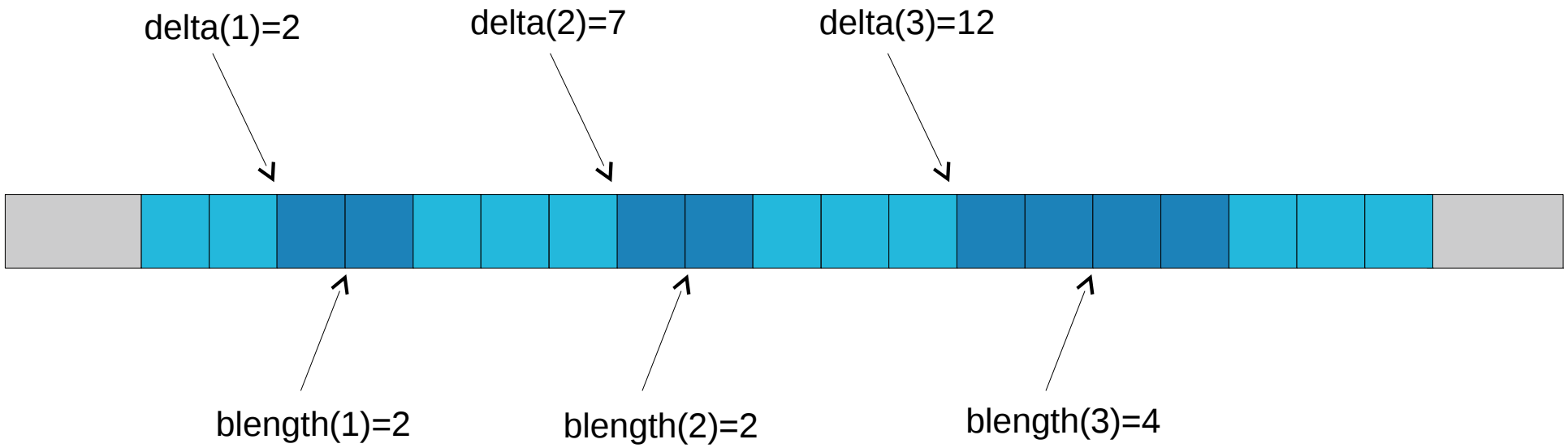
File types of three ranks complementing each other to cover the records of the global file entirely



Committing a file view

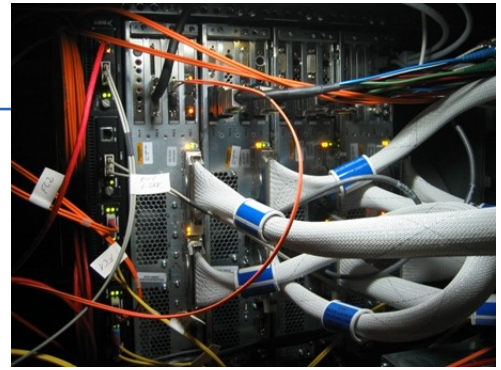


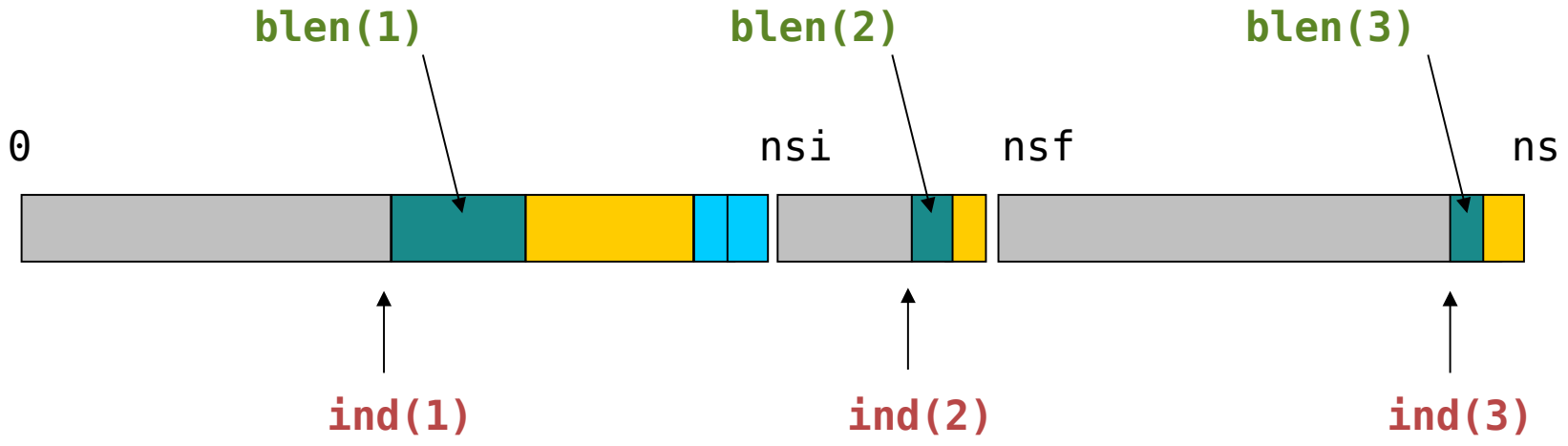
A non-decreasing displacement-blocklength description of the length len=3




```
CALL MPI_Type_Indexed &
   & (len,blength,delta,MPI_REAL8,view,ier)
CALL MPI_Type_Commit(view,ier)
```

MPI_Type_Indexed

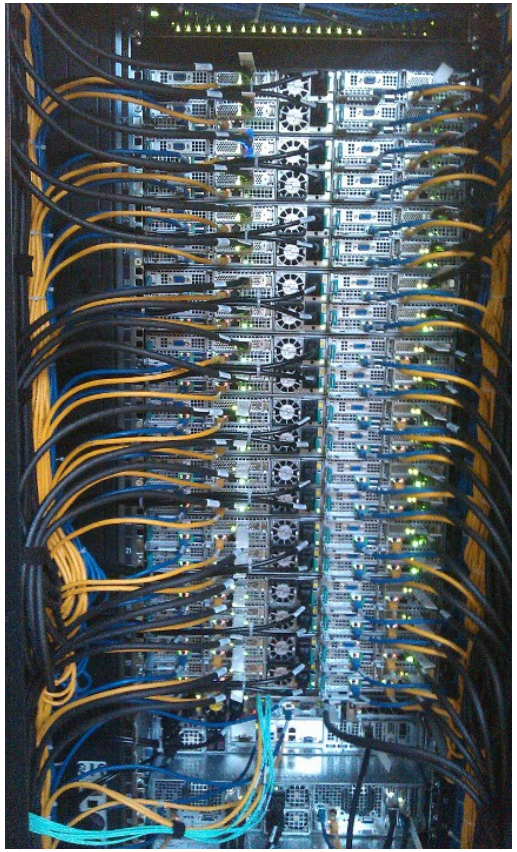





```
CALL MPI_Type_Indexed
(3,blen,ind,MPI_INTEGER,newtype,ier)
CALL MPI_Commit (newtype, ier)
CALL MPI_Send (field,1,newtype,...)
CALL MPI_Free (newtype,ier)
```

Writing a MPI-I/O file

```
CALL MPI_File_Open &
       & (MPI_COMM_WORLD,TRIM(mpiio_restart_file), &
       & IOR(MPI_MODE_CREATE,MPI_MODE_WRONLY), &
       & MPI_INFO_NULL, fhrst, ier)

CALL MPI_File_Set_View &
       & (fhrst, idisp, MPI_REAL8, view, 'native', &
       & MPI_INFO_NULL, ier)

CALL MPI_File_Write_All &
       & (fhrst, iobuffer, lenbuf, MPI_REAL8, &
       & MPI_STATUS_IGNORE, ier)
```



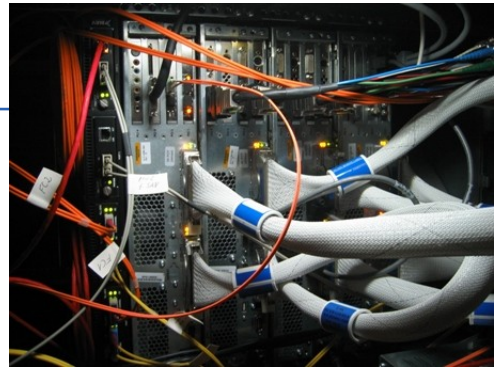
Developments 2010

- *The non-decreasing displacement-length descriptions for local file views can become prohibitively long by arbitrary unstructured mesh partitioning*

- *Countermeasures:*
  - **Resorting the original mesh in the sequence of partitions**
  - Very short descriptions (big interior + BCs + interfaces)
  - 2x-8x gain on even a non-parallel file system, the larger files the better

- *The developments have never been applied for daily work*
  - Thinking of applying parallel data formats **HDF5 or parNetCDF** (2010)

A few remarks on I/O: designers say…



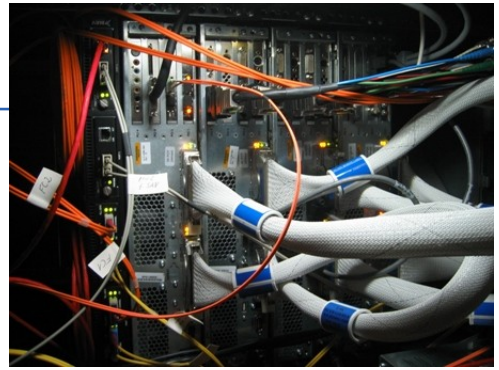
- *Choice of a widely accepted platform-independent format most adequate for pedantic inclusion of meta-data, data exchange, archiving (**netCDF**)*

- *Choice of the technology allowing efficient read & write on all computer platforms – adequate for parallel computers with flexible configuration choices (**HDF5**)*

- *For fine-tuned optimization one can:*
  - consider MPI I/O *and* HDF5 *and* netCDF requirements
  - formulate "hints" for specific hardware requirements

- *Take care of the whole ecosystem around the data and work-flows*

A few remarks on IO: critical users say…



- Chaos in data formats for numerical engines and graphics
- Commercial software – tying consumers on purpose
- So called *"simple and pragmatic solutions"* make things even worse
- Establishing bad practices – while the amount of data grows and grows

- National institutions and hardware vendors try to propose and introduce standards and lasting solutions by example

- …but it is extremely hard to convince anyone to anything…

Desire paths

- The contradiction between the design and user experience
- The least resistance at work
- Rule of the thumb: the design is really bad when the desire paths are broader as the designed ones

Source: Natalia Klishina factionmedia.com

Desire paths

- Obvious idiocy by designers and users
- Completely useless design
- Illogical user behaviour

Source: Natalia Klishina factionmedia.com

Desire paths

- Enforce your will!
- Enforce your will!
- Enforce your will!

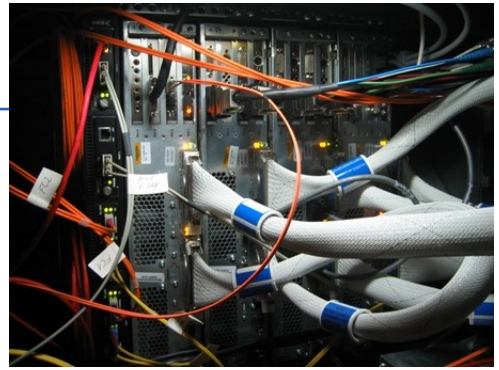
Source: Wikipedia

Desire paths

- Pave the cow paths
- A natural compromise

Source: Ohio State University

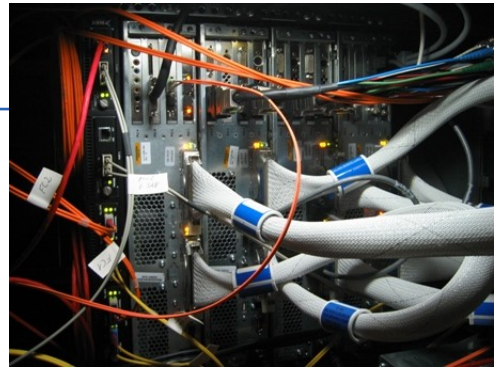
Aspects to consider for IO



- Intuitively simple usage – users and programmers
- Reading/writing with clear advantages when parallel file systems are available (*Lustre*, *PanFS*, *GPFS*, *BeeGFS*…)
- Covering the most of the typical geophysical modelling
- Search-able and archive-able (catalog, meta-data)
- If necessary, lossless storage without data conversions possible

- Must contain **meta-data** and **conventions** making it clear not only what is stored but also *how to deal with the stored data*

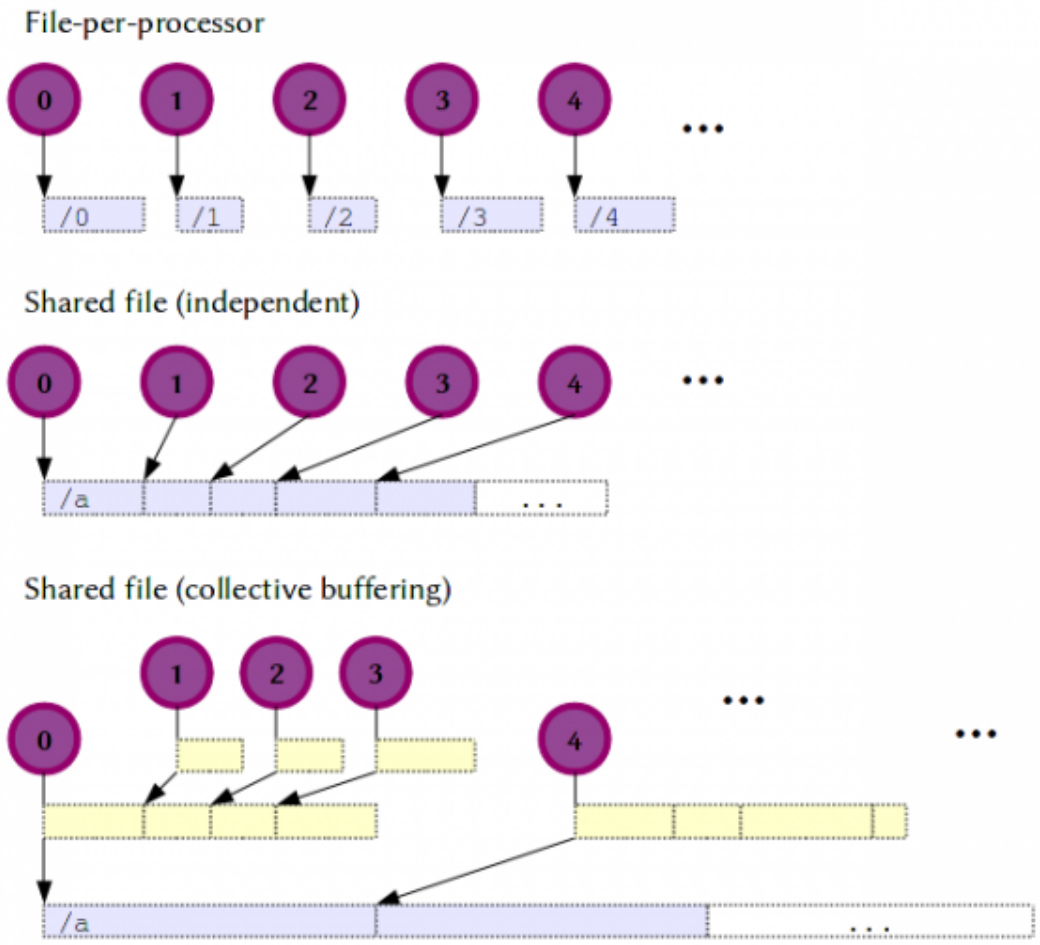
Professional intuition says…



- *Radical – save the data and programs that write and read them:*
  - this is equivalent to establishing conventions
- *Platform-independent – especially for the future:*
  - standard data types and accuracy
  - no dependencies on the platforms and programming languages
- *Hardware-aware programming methods:*
  - help to optimize the hardware to the software, e.g. parallel file systems
- *Data visualization and analysis without data conversions*

HDF5: *Hierarchic Data Format*

- Data model, library, data format for storing and managing scientific data – binary independent cross platforms
- HPC: designed for reading/writing HUGE amounts of data
- Data access: collective, independent
- Parallel IO based on MPI-IO
- Supports hardware “hints”, “chunking”
- Basic meta-data description
- Site: https://www.hdfgroup.org/



http://www.nersc.gov/users/training/online-tutorials/introduction-to-scientific-i-o

netCDF: *network common data form*

- *netCDF* https://www.unidata.ucar.edu/software/netcdf/
  - Mainly for geophysical data – Unidata 1989-2008 (“mature”)
  - Large versatility (model results as well as measurements, archiving)
  - Conventions: published agreements how the data is to be stored in order to foster interoperability
  - Commitment to backward compatibility
  - https://www.unidata.ucar.edu/software/netcdf/software.html

- *Data containers: ASCII or binary formats as “legacy”*
  - HPC-oriented own development parNetCDF given up in favour of HDF5
  - netCDF4 uses HDF5 as binary data containers with redundant metadata

Developments 2018

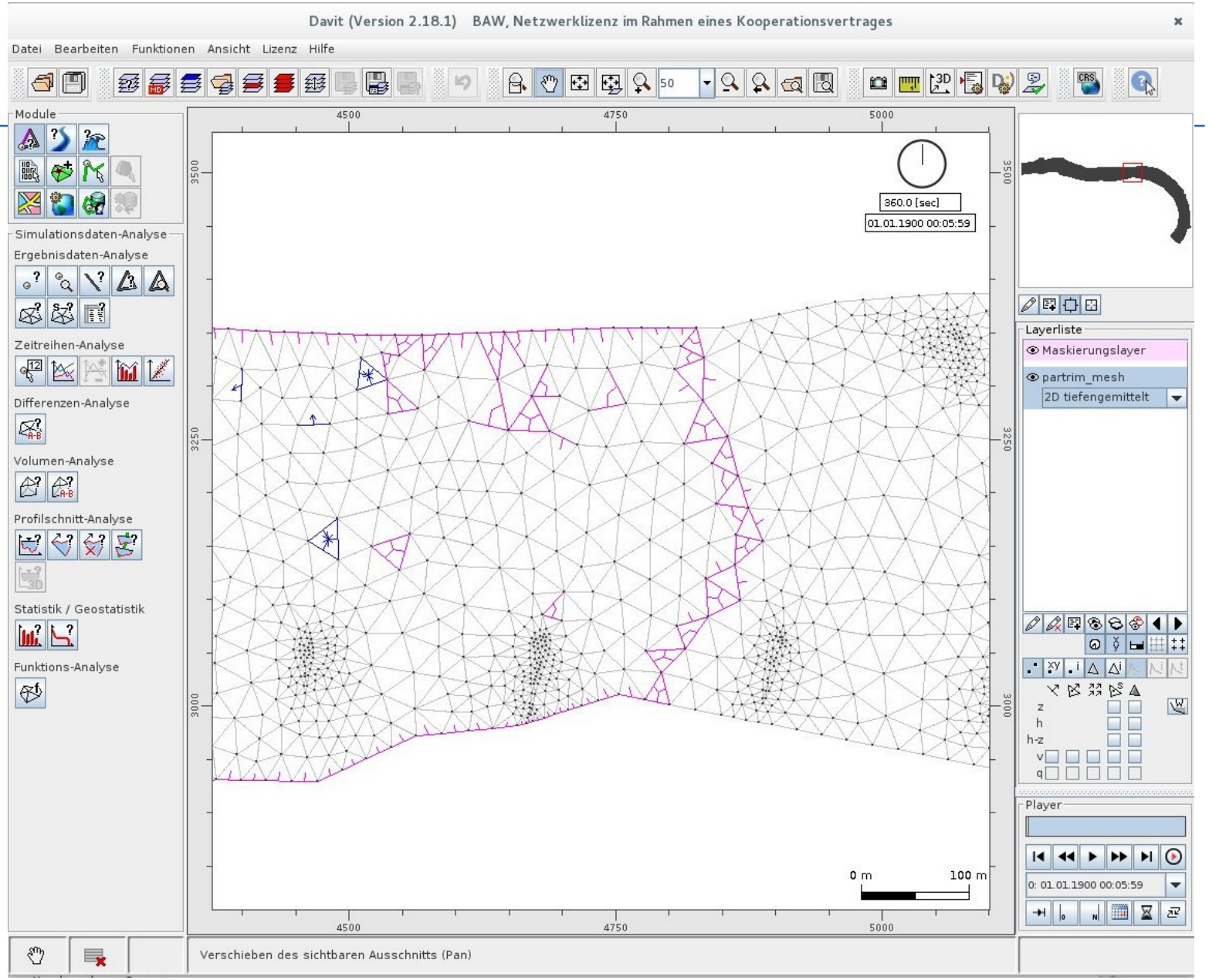
- ***HDF5 and therefore netCDF4/HDF5 require variable values to be a completely consecutive data field pro rank***
- *This can be reached by re-sorting the global (original) mesh, but:*
  - the sorting of BC object into specific index ranges is gone
  - overlapping between partitions must be treated strictly
- *Awkwardness:*
  - The resulting re-sorted mesh on which the results are written *is not* a correct UnTRIM mesh…
  - We have to mark boundaries, BC types differently…
  - *We have one mesh for computations and …*
    …a second one the for output of variables…

Developments 2018

- ***HDF5 and therefore netCDF4/HDF5 require variable values to be a completely consecutive data field pro rank***
- *This can be reached by re-sorting the global (original) mesh, but:*
  - the sorting of BC object into specific index ranges is gone
  - overlapping between partitions must be treated strictly
- *Awkwardness:*
  - The resulting re-sorted mesh on which the results are written *is not* a correct UnTRIM mesh…
  - We have to mark boundaries, BC types differently…
  - *We have one mesh for computations and …*
    …a second one the for output of variables…

Davit fooled

- Post-processors “knowing something” about the UnTRIM mesh structure can deliver strange results…
- Adaptations to an arbitrary mesh sorting are needed



Workflow 2018 [work in progress!]

- *Partitioner delivers:*
  - the partitioned *computational mesh* files as usual
  - a netCDF4/HDF5 file containing sorted *output mesh*
  - translate files – computational to output mesh
- *Parallel run:*
  - restart files written as usual for the computational mesh
  - a netCDF4/HDF5 file is appended time step by time step with variables defined on the output mesh – collective output from all tasks
- *Post-processing*
  - a single netCDF4 file, non-conform mesh [3D at work…]

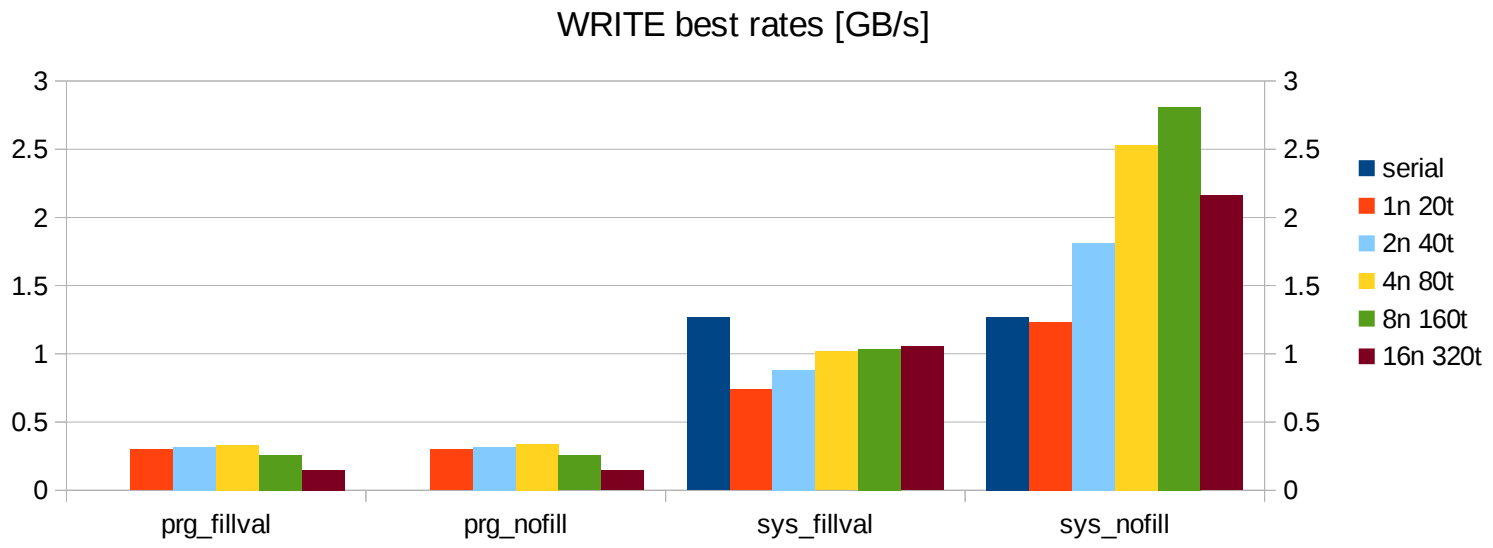
Sketch – re-opening the netCDF output file

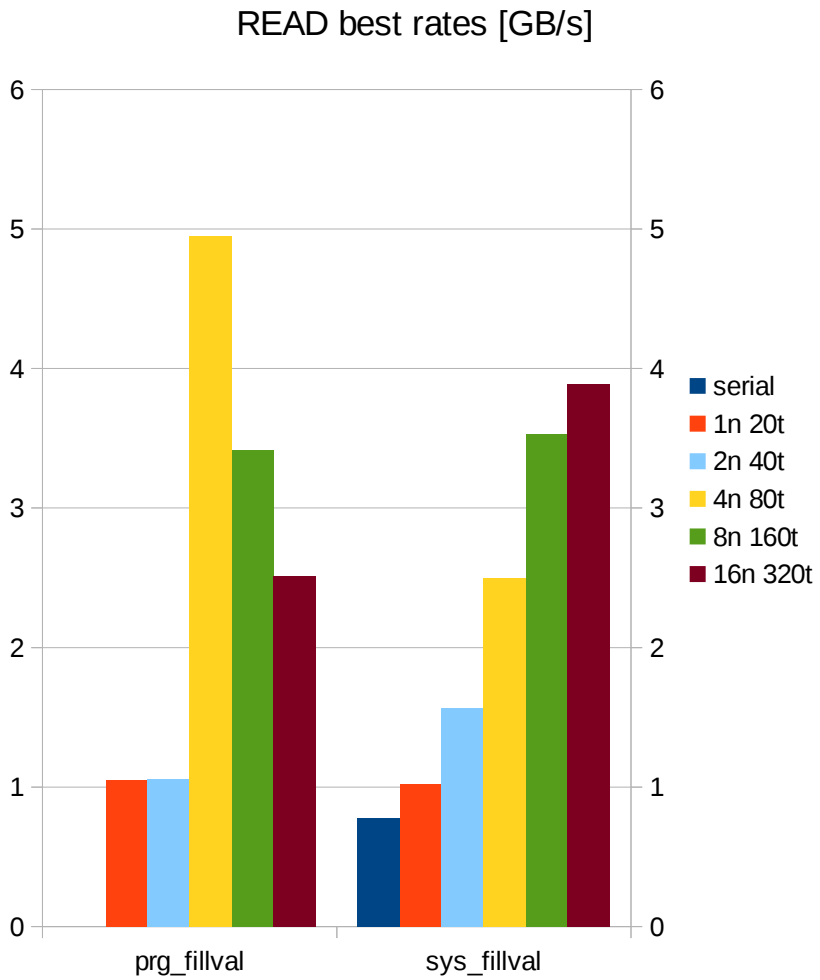
```
CALL nftry(nf90_open(TRIM(get_netcdf_output_file()), &
         & IOR(NF90_WRITE,NF90_MPIIO), ncid, &
         & comm = MPI_COMM_WORLD, info = MPI_INFO_NULL))

CALL nftry(nf90_var_par_access(ncid, Mesh2_edge_Variable_2d_id(n),
     & NF90_COLLECTIVE))

work(:,:,:) = get_face_vector_velocity()

! ugly!
Mesh2_edge_Variable_2d(1:edge_len) = work(1,is_lo(1:edge_len),1)

CALL nftry( nf90_put_var (ncid, Mesh2_edge_Variable_2d_id(n), &
                              & Mesh2_edge_Variable_2d(:), &
                              & Mesh2_edge_Variable_2d_start, &
                              & Mesh2_edge_Variable_2d_count) )
```

BeeGFS with GL`s netCDF4/HDF5 test (automatix)

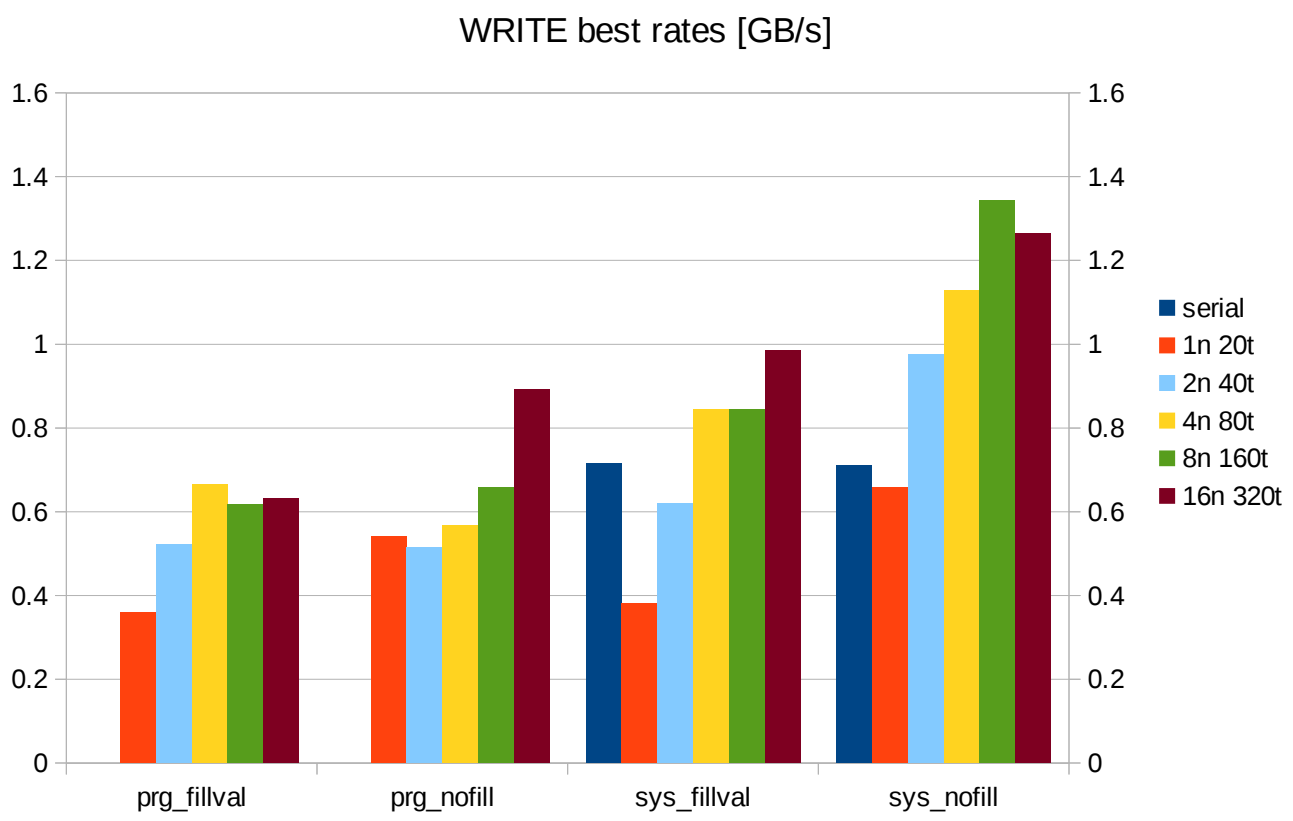
Günther Lang's test program mimicking UnTRIM output writing/reading in various netCDF settings (chunking, fill):

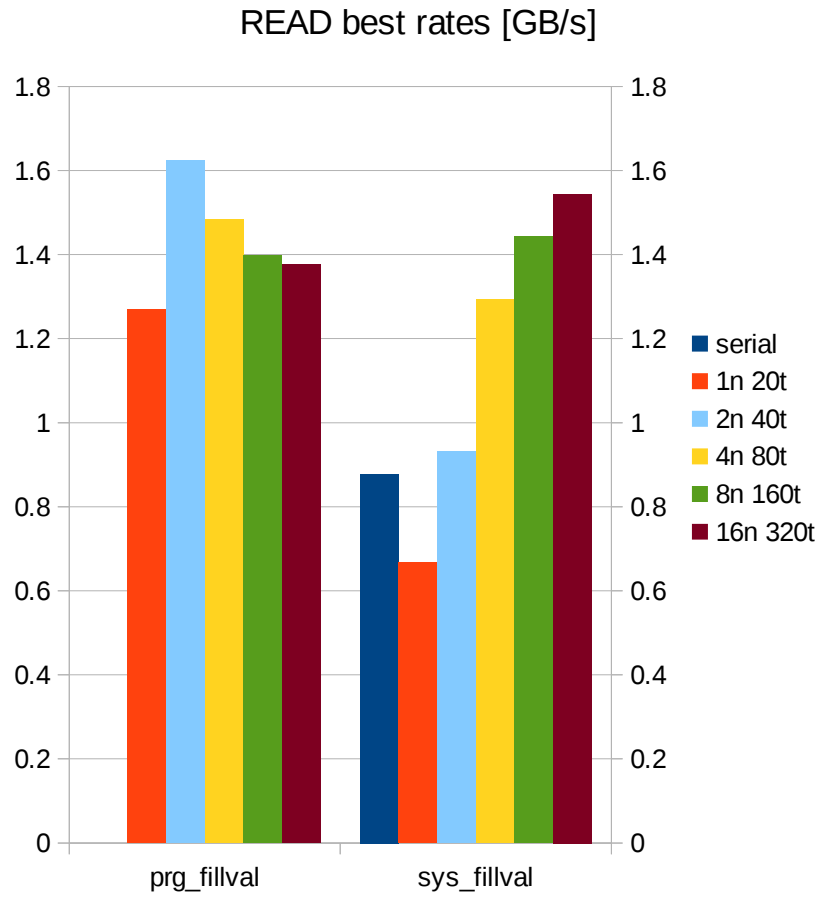
- 1000 **records** (time steps) of
- **water level** (2D) and
- **velocity** (3D) defined on
- a mesh of 640 000 polygons and 20 levels
- **ca. 107GB** of data



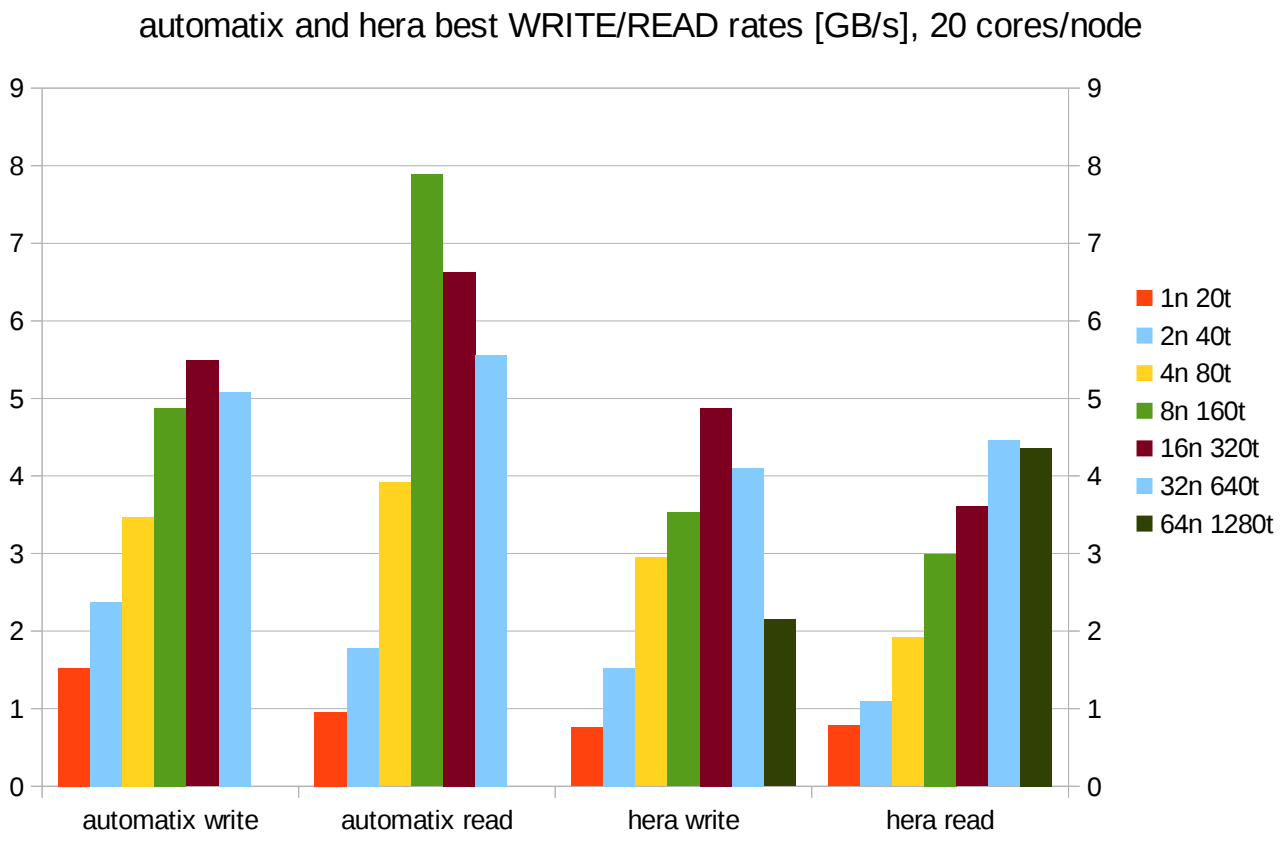


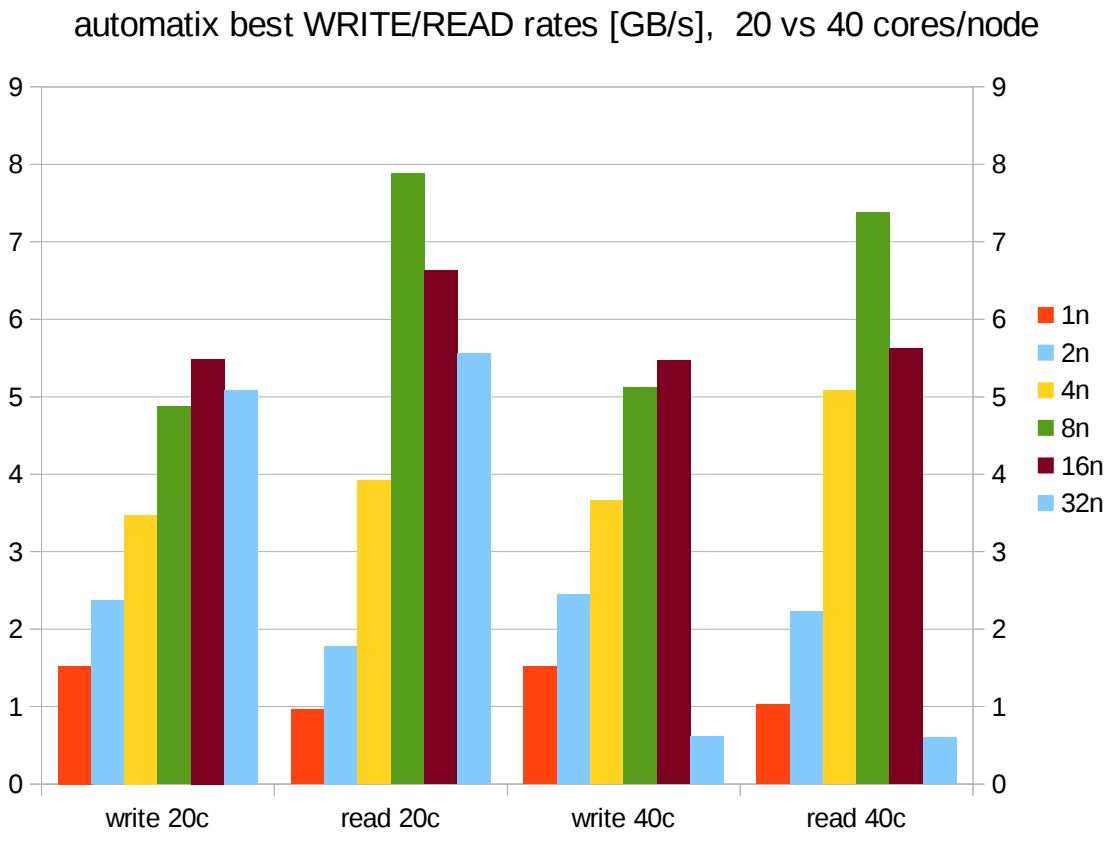
Lustre FS with GL`s netCDF4/HDF5 test (hera)





Native MPI-IO on both machines and both file systems





MPI lightweight stats netCDF test: 20 tasks/node, 1 node

#	[time]	[calls]	<%mpi>	<%wall>
# MPI_File_read_at_all	2835.47	60000	49.52	48.92
# MPI_File_write_at_all	2821.22	60000	49.27	48.68
# MPI_File_set_view	32.0408	240000	0.56	0.55
# MPI_Init	15.0917	20	0.26	0.26
# MPI_Allreduce	5.25941	360000	0.09	0.09
# MPI_File_write_at	3.69075	687	0.06	0.06
# MPI_File_read_at	3.26539	13640	0.06	0.06
# MPI_Type_commit	2.48151	800000	0.04	0.04
# MPI_Type_free	1.46513	3.44e+06	0.03	0.03
# MPI_File_open	1.38786	60	0.02	0.02
# MPI_File_sync	0.939526	40	0.02	0.02
# MPI_Finalize	0.92819	20	0.02	0.02
# MPI_Barrier	0.639764	320	0.01	0.01
# MPI_File_set_size	0.529375	40	0.01	0.01
# MPI_Type_vector	0.417033	1.28e+06	0.01	0.01
# MPI_Type_create_hindexed	0.28945	680000	0.01	0.00
# MPI_Type_contiguous	0.257054	640000	0.00	0.00
# MPI_Type_create_resized	0.220269	680000	0.00	0.00
# MPI_Type_create_struct	0.135908	160000	0.00	0.00
# MPI_Comm_dup	0.127668	120	0.00	0.00
# MPI_Type_get_extent	0.0894065	1.28e+06	0.00	0.00
# MPI_Bcast	0.0469584	220	0.00	0.00
# MPI_File_close	0.0450406	60	0.00	0.00
# MPI_Get_elements	0.0275884	134327	0.00	0.00
# MPI_Comm_free	0.0163414	120	0.00	0.00
# MPI_Type_size	0.0154474	134327	0.00	0.00
# MPI_File_read	0.00260162	20	0.00	0.00
# MPI_Comm_set_attr	0.0023284	20	0.00	0.00
# MPI_File_get_size	0.00206518	2	0.00	0.00
# MPI_Comm_free_keyval	5.93662e-05	20	0.00	0.00
# MPI_Comm_create_keyval	4.19617e-05	20	0.00	0.00
# MPI_Comm_size	1.07288e-05	60	0.00	0.00
# MPI_Comm_rank	1.00136e-05	60	0.00	0.00
# MPI_TOTAL	5726.1	9.9642e+06	100.00	98.80

WRITE and READ balanced

A large number of specialized MPI routines for dealing with data structures

Dealing with MPI-IO parameters and “hints” for the file system

MPI lightweight stats netCDF test: 20 tasks/node, 16 nodes

#	[time]	[calls]	<%mpi>	<%wall>
# MPI_File_write_at_all	17659.5	960000	44.65	44.41
# MPI_Barrier	8294.88	5120	20.97	20.86
# MPI_File_read_at_all	7911.37	960000	20.00	19.89
# MPI_File_set_view	2535.39	3.84e+06	6.41	6.38
# MPI_File_write_at	1107.35	687	2.80	2.78
# MPI_Init	1097.5	320	2.78	2.76
# MPI_File_read_at	278.945	218240	0.71	0.70
# MPI_File_open	239.537	960	0.61	0.60
# MPI_Allreduce	231.684	5.76e+06	0.59	0.58
# MPI_Comm_dup	65.711	1920	0.17	0.17
# MPI_Type_commit	39.0033	1.28e+07	0.10	0.10
# MPI_File_set_size	35.4273	640	0.09	0.09
# MPI_Type_free	16.4591	5.504e+07	0.04	0.04
# MPI_File_sync	13.6052	640	0.03	0.03
# MPI_Type_vector	5.04694	2.048e+07	0.01	0.01
# MPI_Type_create_hindexed	3.87808	1.088e+07	0.01	0.01
# MPI_Type_contiguous	2.92011	1.024e+07	0.01	0.01
# MPI_Type_create_resized	2.57493	1.088e+07	0.01	0.01
# MPI_Type_create_struct	2.00874	2.56e+06	0.01	0.01
# MPI_Type_get_extent	1.43242	2.048e+07	0.00	0.00
# MPI_Finalize	1.33385	320	0.00	0.00
# MPI_Bcast	0.935276	3520	0.00	0.00
# MPI_File_read	0.576205	320	0.00	0.00
# MPI_Get_elements	0.427633	2.13893e+06	0.00	0.00
# MPI_Type_size	0.262584	2.13893e+06	0.00	0.00
# MPI_Comm_free	0.159705	1920	0.00	0.00
# MPI_File_close	0.118994	960	0.00	0.00
# MPI_Comm_set_attr	0.00435066	320	0.00	0.00
# MPI_File_get_size	0.00179887	2	0.00	0.00
# MPI_Comm_create_keyval	0.000505447	320	0.00	0.00
# MPI_Comm_free_keyval	0.000430346	320	0.00	0.00
# MPI_Comm_size	0.00018549	960	0.00	0.00
# MPI_Comm_rank	0.000174284	960	0.00	0.00
# MPI_TOTAL	39548.1	1.59396e+08	100.00	99.45

WRITE and READ out of balance

MPI_Barrier takes a lot of time: processes waiting for each other!

It happens seemingly by writing files!

MPI lightweight stats MPI-IO test 20 tasks/node: 1n vs 16n

```
#                          [time]        [calls]       <%mpi>        <%wall>
# MPI_File_write_all       2797.7        20000         49.96         48.60
# MPI_File_read_all        2766.03       20000         49.39         48.05
# MPI_File_set_view        25.3986       40000         0.45          0.44
# MPI_Init                 9.9248        20            0.18          0.17
# MPI_File_open            1.04715       40            0.02          0.02
# MPI_File_close           0.029809      40            0.00          0.00
# MPI_Barrier              0.0235887     20            0.00          0.00
# MPI_Type_commit          0.018961      40            0.00          0.00
# MPI_Type_indexed         0.00389314    40            0.00          0.00
# MPI_Finalize             0.000149488   20            0.00          0.00
# MPI_File_get_info        2.71797e-05   2             0.00          0.00
# MPI_Get_processor_name   1.64509e-05   20            0.00          0.00
# MPI_Type_free            1.00136e-05   40            0.00          0.00
# MPI_Comm_rank            6.91414e-06   20            0.00          0.00
# MPI_Comm_size            5.00679e-06   20            0.00          0.00
# MPI_TOTAL                5600.17       80322         100.00        97.28
#                          [time]        [calls]       <%mpi>        <%wall>
# MPI_File_read_all        10125.3       320000        53.28         53.17
# MPI_File_write_all       7254.58       320000        38.17         38.10
# MPI_Init                 781.957       320           4.11          4.11
# MPI_File_set_view        535.468       640000        2.82          2.81
# MPI_File_open            300.459       640           1.58          1.58
# MPI_Barrier              4.51386       320           0.02          0.02
# MPI_File_close           1.2539        640           0.01          0.01
# MPI_File_delete          0.320671      34            0.00          0.00
# MPI_Type_commit          0.0308437     640           0.00          0.00
# MPI_Type_indexed         0.0175378     640           0.00          0.00
# MPI_Finalize             0.00455642    320           0.00          0.00
# MPI_Get_processor_name   0.000209808   320           0.00          0.00
# MPI_Comm_rank            7.62939e-05   320           0.00          0.00
# MPI_Comm_size            7.43866e-05   320           0.00          0.00
# MPI_Type_free            5.48363e-05   640           0.00          0.00
# MPI_File_get_info        2.40803e-05   2             0.00          0.00
# MPI_TOTAL                19004         1.28516e+06   100.00        99.80
```

WRITE and READ balanced almost independently of the number of nodes, tasks

A small number of specialized MPI routines for dealing with data structures

Writing less efficient with growing number of nodes, tasks

Understandable behaviour

Is it really our desire path?

- *Transfer rates:*
  - MPI-IO version writing the same data in the 'native' way is up to a few times better in transfer rates than netCDF4 with HDF5
  - this is not tragic – but requires some attention [**work in progress!**]

- *Post-processing:*
  - users would have to accept that the output mesh is very different than the one they have generated for UnTRIM

- *Decisions required, because…*
  - Maybe, e.g.,it is better to use the native MPI methods for speed on a given machine and then convert the files to something more *"professional"*?

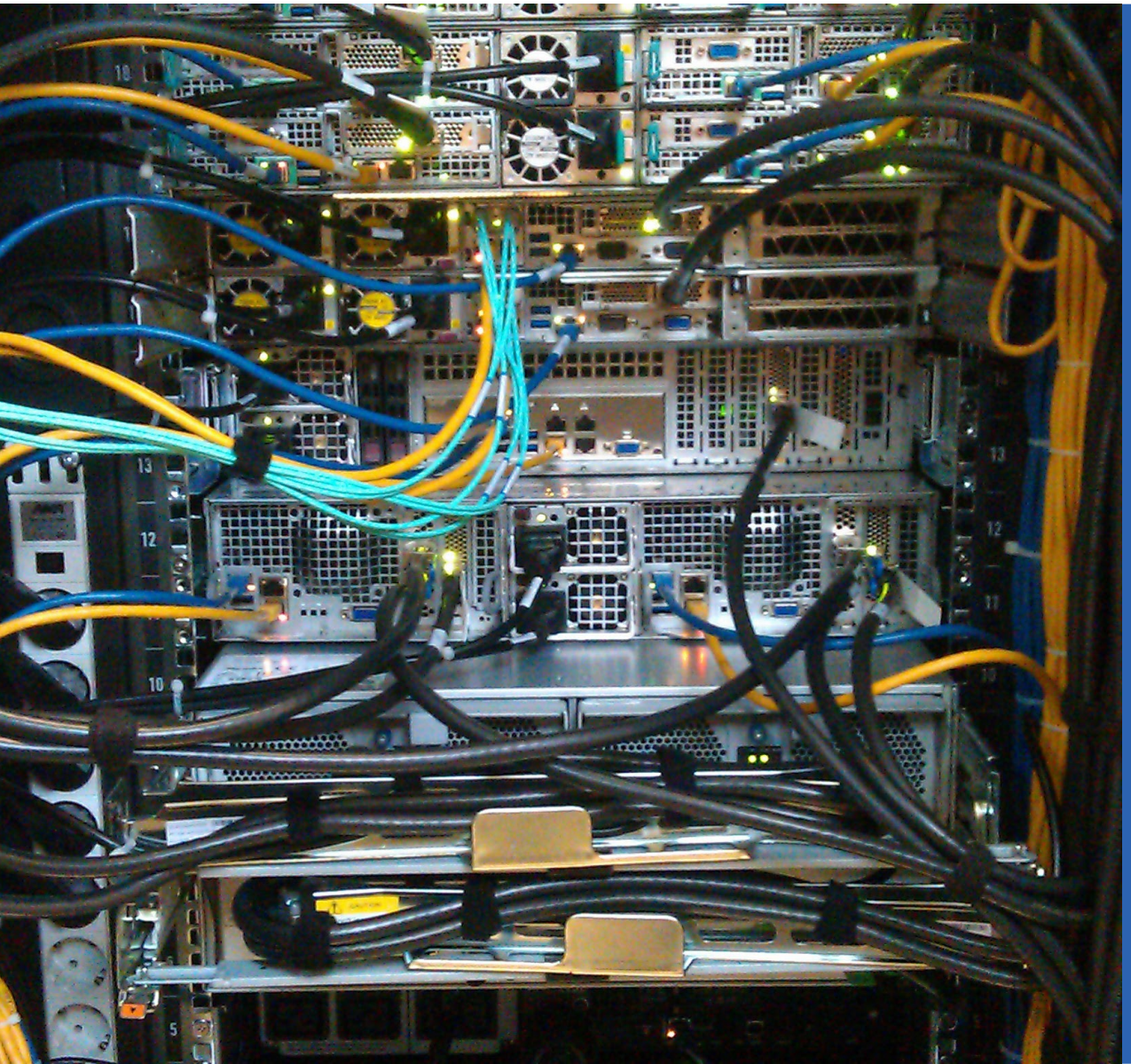
Repeating the outlook from 2018

- The usage of parallelism in algorithms and data is the key to success
  (there is no solution for the lack of resolution)

- Coarse-grained parallelism is the main method of obtaining speedup
  (engineers mostly accept they obtain their results overnight)

- UnTRIM is going to stay with us for some more time
  (let us parallelise it again)

- Success “ten years after” is possible!
  (no philosophical part follows)



Parallel IO for parallel UnTRIM

Bundesanstalt für Wasserbau
76187 Karlsruhe, Germany

www.baw.de