RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

cea cnrs Inria

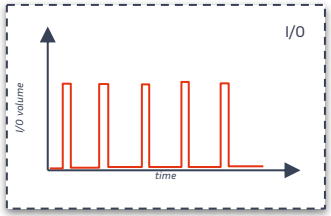
FRANCE 2030
PROGRAMME DE RECHERCHE
NUMÉRIQUE POUR L'EXASCALE

ExaDoST - WP1
Application Motifs

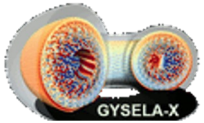
WP Leaders:
Francieli Boito (Université de Bordeaux) &
François Tessier (Inria Rennes)

Goal of this session

- Preparation of the next deliverable: Analysis of relevant application motifs and their covering by the project illustrators.
- Identification and description of application motifs that will drive the research in our WP…
- **…and** that are representative of our illustrators and beyond.
- Method: brainstorming with both the application community and the computer scientists.



Example: Checkpoint of 100 GB of data every 10 iterations on the node-local storage. Data is temporarily persistent and moved at regular intervals to long-term storage (PFS)

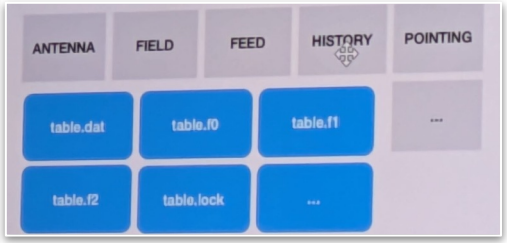


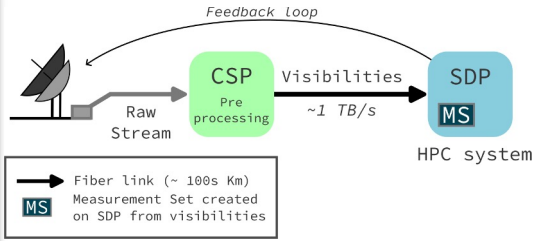
Example: Gysela exhibits this type of pattern

Applicative Working Groups - Notes

Session 1 - SKA

- In radio astronomy, files are stored in a specific format called **Measurement Set.** A MS is a directory containing several files that are distributed across the processes
  - Contain the raw data (which sensor, when, which frequency, etc…) and lots of metadata
  - Structured data split in different files of highly varying size (from small to very large) and used as a relational database (files point to each other)
  - Known to be **inefficient…**
  - It is a standard. Discussions are on-going between SKAO and NRAO to produce a new version of the format (move from v2 which is latest stable to v4).
  - **Typical size: 1 MS from 10s MB to 100s GB**
    - Ex: 1 observation, 8 hours, 1ch/SB, dt=1s -> 100GB
  - A MS can be augmented with new data (result of pre-processing)
    - within the initial MS but raw data integrity need to be ensured during the processing OR in a new MS
- The current parallelization of a MS is by file. A process manages one or several files
  - A **temporal/spatial parallelization** could take place but data is distributed across files
    - Potential solutions: files accessed by multiple processes OR new data layout a save it in new files
- Reading and writing data from/to a MS is **exponentially expensive when scaling up**
- **Casacore library not parallel-IO ready, nor thread-safe**
- The SDP receives streaming visibilities from the correlator (CSP which calculates them at the (current) maximum rate of 1TB/s)
  - SDP tags visibilities affected by radio-frequency interferences (RFI)
  - SDP preprocesses visibilities (exposure stacking, baseline averaging) prior to imaging (but other tasks require preserving all input data)
  - data is continuously streamed from the CSP to the SDP so the SDP needs to be able to ingest it continuously
  - MS are created when visibilities need to be stored on the buffer
  - observations can last up to 8h



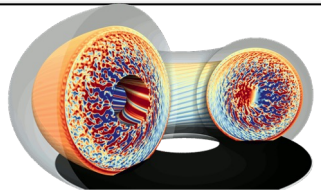


Motif inspired by SKA

An application generates a large amount of data and writes it into Measurement Set (MS). A MS file is directory where a collection of files store diverse structured data and metadata. An MS can range in size from a few dozen MB to several hundreds of GB. For data analysis and processing, the MS is read multiple times. Data is distributed to processes according to several dimensions (temporal, frequency (channels), spatial (facets), resolution (baselines)). Following the data analysis/processing phase, new data can be added to a MS. Until all processing is complete, the integrity of the raw data in a MS must be guaranteed. We identified multiple issues with the MS format:

- The casa-core library used to access a MS is not designed for parallel I/O
- Not using MS files does not seem to be an option. The MS format is widely used in the community.
- A MS is directory containing small to large files (can be split at creation time? During a pre-processing phase? Depends on the resources available for the processing phase)
- It would be best to use libraries that can adapt to hardware, instead of having to adapt the applications (because it can change, and we don't fully know how it will be yet)
- The I/O time needs to be predicted in order to manage a storage tier whose space is very limited (at which timestamp you can get rid of a MS). Data placement is also very important.
- shared namespace is important (like a PFS)
- the final products of imaging are stored as a single or multiple FITS files (several or single channel per file). Mid image cubes are estimated to be ~15 TB

Session 2 (Gysela) notes

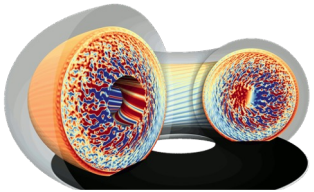


- original: checkpoint file is file-per-process
  - 22784 mpi ranks -> same number of files, total of 16.2 TB for the largest scale they tested
  - done with PDI
  - they moved to parallel HDF5 to do shared-file
- there is also some output that is saved, they want to compress it smartly with AI
- contacts: Dorian Midou and Kevin Obrejan
- not one process per node (as in Julien's description), but a few MPi ranks per node, and they all write. There will typically be one rank per gpu, or one rank per NUMA domain. In total the distribution function uses ¼ of te node RAM, if there are multiple processes they will each write a portion of it.
- ¼ memory in total, may be gpu
- 5d array of double (they call it the "distribution function"
- each process gets an equal size, but it may not be the case in the future
- this happens every N timesteps
- keep two sets of files
- synchronous, don'twant to dedicate compute resources fr I/O, maybe for gpu. they tried with fti and a house-made solution, it was not worthy it. even using a local ssd was not worthy it
- minimum is one checkpointing at the end, the rate depends on the time between failures. it is the start of a new simulation to save time, starting already from a certain steady state
- it can be any format
- it'sn ot possible to restart with a different scale, but they would like to
- ghost cells: for now it's like 2 every 32 points for each dimension, but it will be different in future versions (not clear how it will be)
- there are always at the limit of memory, no memory copies are tolerated
- the application works with two versions of the 5d array, updating one at atime. So that means that if there was a way of doing asynchronous I/O, that would be possible without memory copy, because the array that is being written is not going to be modified for a sub-timestep. But that is quite a small time scale, a few seconds maybe
- the 5d data is just for checkpointing, it's never ever used for anything else (they make 3d reduced data they write and it's very fast, and they use that for analysis)

Motif inspired by Gysela

The application periodically writes a checkpoint file, that is used for fault tolerance reasons only. The very last checkpoint file is also used to restart a next simulation, so it is mandatory. The application runs with a few processes per node (one per GPU or per NUMA domain, for example), and the amount of data written corresponds to ¼ of the node memory (each process in the node will write a part of it). The data is not contiguous in memory, but contiguous writes could be done with the cost of writing more data (including “ghost cells”). Processes may have the same amount of data or not.

- the application uses all the node RAM, so solutions that require memory copies are not acceptable
- to dedicate compute resources for I/O could not be possible right now, but maybe in the future when using GPU (CPU resources could be used to advance I/O)
- they would like to be able to restart with a different number of processes
- predictability of checkpointing time at the end is an issue
- hundreds of TB to a few PB of checkpointing data at exascale



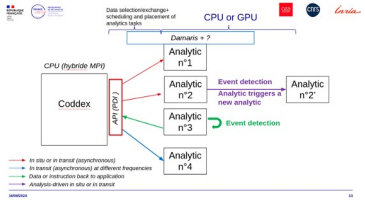
Notes during the session

New functionality in slurm where we can send a signal to a job saying they should stop

wp2 proposes that checkpointing happens as a plugin for pdi/damaris, so it can use the same core that is already dedicated for analytics. That wll prevent a situation where there are a thousando dedicated cores to a thousand different functionality

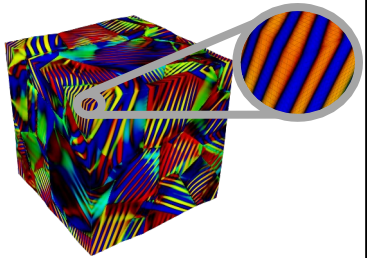
in this sense it would be better to take fti (that is no longer supported) as a part of the pdi project, rather than using veloc

Vision:



Applicative Working Groups - Notes

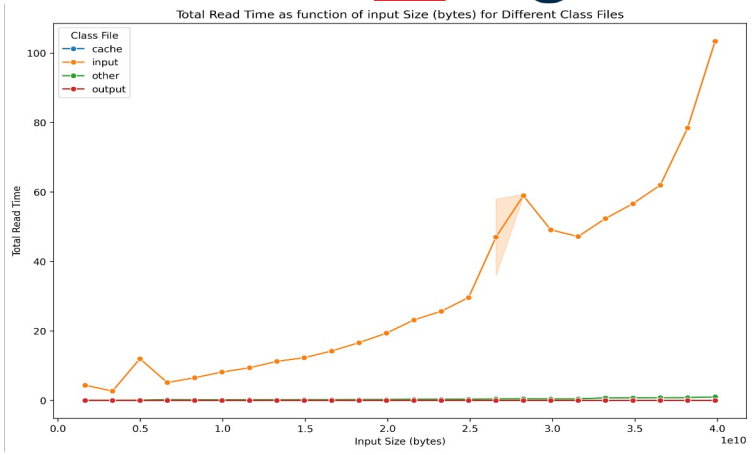
Session 3 - Coddex & Other

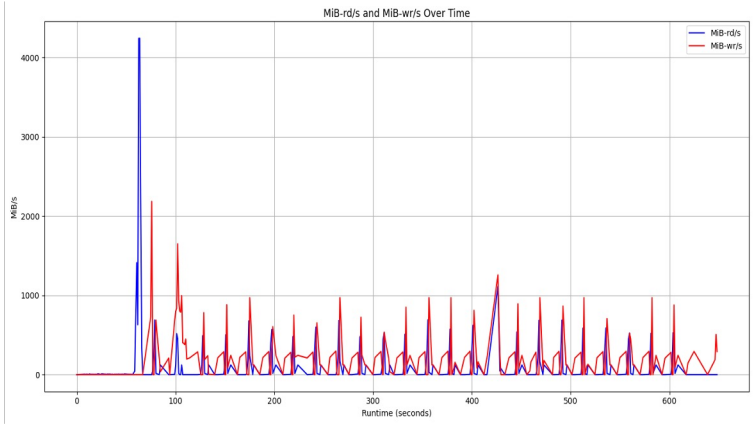


- Coddex is a FEM simulation (Smoothed Element Galerkin solver) - multi-physics simulation
- **50*10^6 elements .. may go up to 50-200 million**
- C++ code with Python interface (MPI + OpenMP) - 110k LOC, 40 regression tests + 11 other tests
  - has a end-to-end miniapp (input files, some demonstration of elasticity, and output files)
- MPI Parallelism
  - only MPI parallelism is in production
  - voronoi for domain decomposition that is balanced across nodes
- there is multiple high frequency visualisation pipelines
  - (legacy?, fastest) with postscript (“2D” reduced data projection of cube, multiple grid cells per visual representation) - (data volumes?)
  - vtk (which require full volume - (data volumes?)- 2x 10`6 elements (4096 cores, 64 MPI rank)
  - **simulated X-ray diffraction (4k cores, 64 mpi proces, 10 hours, 2 million elements) - Paraview - read IO is a bottleneck - currently post-mortem in situ candidate**
  - raw slow for large simulation, writes to single file(?) - 3.5 million (4k cores, 64 MPI ranks,, 10 hours) - read IO bottleneck - long to process, no grid? maybe does FFT
- Plan to move different analysis to in situ to bypass storage and reduce checkpoint frequency that need to be written - desire elasticity for additional analysis that may be triggered based on some condition during simulation runtime

SKA - DDFacet

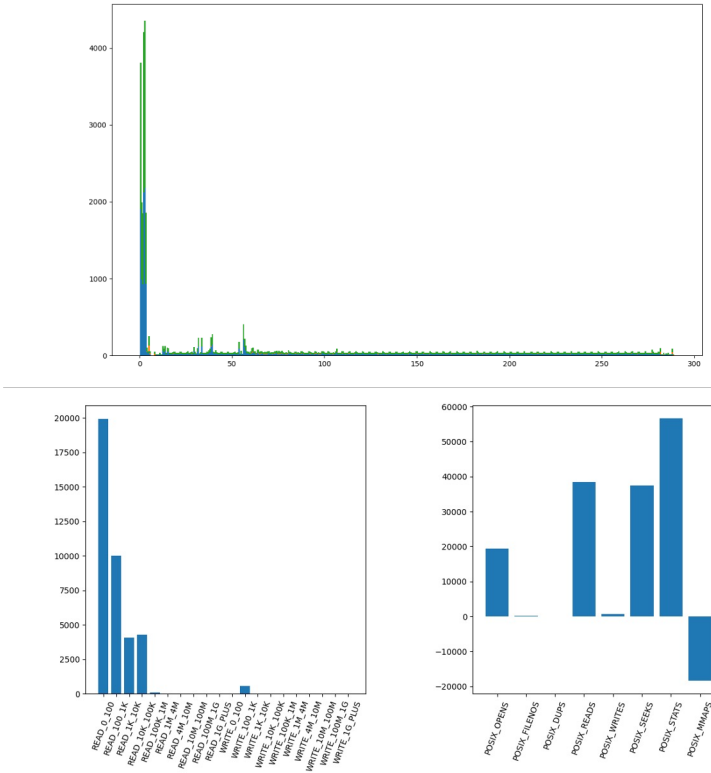
- Experiments with a single-node version during internship (TADaaM+LAB)
- Read time is ~15% of execution time
  - scalability may be an issue
- The application **writes and reads in a “cache” folder** (~50GB when input is ~40GB)
  - output is negligible
  - the “cache” is reused by next steps of the pipeline
- Reads and writes throughout the execution
- **Darshan is NOT able** to properly profile the write operations





SKA - QuartiCal

- Fast radio interferometric calibration routines exploiting complex optimisation
- Ugo Thay's internship: **I/O monitoring of QuartiCal**
- Combination of Darshan and strace, experiments run on PlaFRIM (cluster at Inria Bordeaux)
- Findings:
  - QC is **read and metadata-intensive**
    - Lots of small files accesses
    - Lots of seek/stat operations
  - **Locking issues** slowing down the I/O performance



Motifs

1. Application reads a large amount of data, with high bandwidth, in the beginning of the execution
    a. Examples: DDFacet
2. Application writes data that it reads back during the same execution
    a. Examples: DDFacet
3. Application writes data that other applications will want to read soon (different jobs?)
    a. Examples: DDFacet

Challenges

Provisioning of I/O resources for applications that have single bursts of high I/O activity

- provide good performance while avoiding interference
- stripe count tuning for read operations

Data placement for data that is written and then read back

- across jobs?

Conclusion

- T

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

cea cnrs Inria