Pallas: HPC Trace analysis at scale

Catherine Guelque[*], **Philippe Swartvagher**[▲], **Valentin Honoré**[*], **François Trahay**[*]
[*] Télécom SudParis, Institut Polytechnique de Paris, Inria Benagil
[▲] Bordeaux INP, Inria Topal

Context: performance analysis at exascale

Investigating performance problems

- Many sources of performance problems at scale
  - Collective communication, load imbalance, network contention, NUMA effects, ...
- Finding a bottleneck requires investigation
  - Run the application once, analyze its execution trace
- Applications mix programming models (MPI+OpenMP, MPI+CUDA), and libraries (StarPU, netcdf, Pytorch, ...)
- Need for **generic, modulable tracing tools**

Analyzing huge performance data

Finding the needle in the haystack

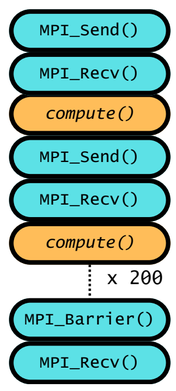
- Traces are **records of execution** made for **post-mortem analysis**
  - Store detailed information.
  - Can take huge amounts of storing space
  - Require a lot of processing before being useful
- Gets worse with execution time and number of threads used.
- Traces of HPC applications have recurring patterns, which means they have a lot of **redundant data.**

Proposal: Pallas trace format

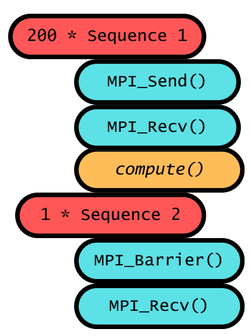
Detecting sequences of events for better data analysis

- Intercepted functions are **grouped by call signature** as Tokens.
- Their **durations** are stored for analysis.
- **Repetition** of similar functions are detected using memcmp and **stored as loops**, allowing for better view of the structure of the trace.

Example of a sequential trace



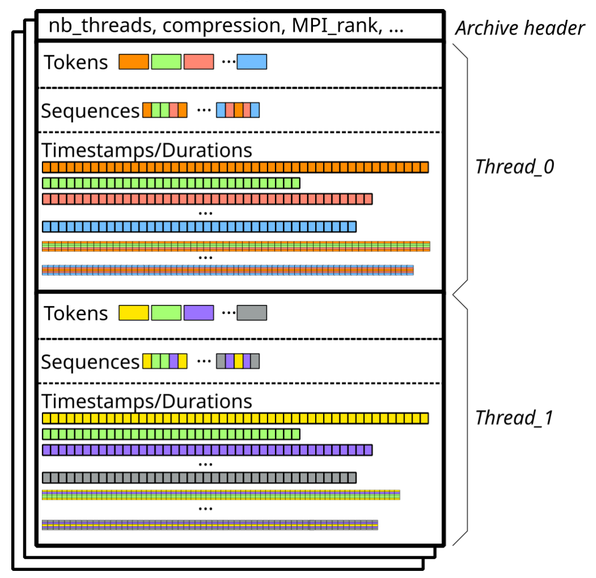
Example of a Pallas trace



Analysis-optimized trace storage

Separating data from metadata

- One folder per MPI rank
- Header file with general information (lightweight)
- Data file with durations:
  - Efficient compression
  - Easily removable
  - On-demand loading



Evaluations

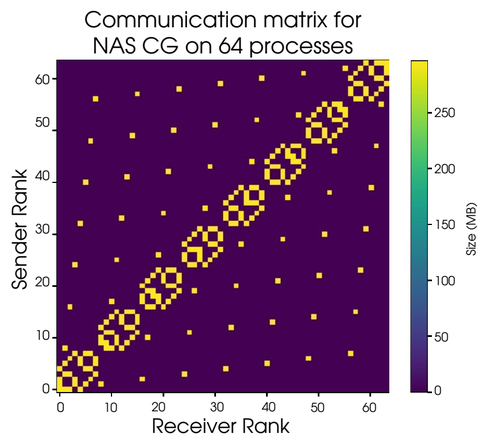
Comparison with **Pilgrim** (pattern recognition and compression) and **OTF2** (sequential format with timestamp encoding).

- Negligible overhead for most applications (1-10%) except for **irregular ones** (57% for Quicksilver)
- Trace size lies between OTF2 and Pilgrim
- Pilgrim intercepts less function and compresses all timestamps at once → Smaller traces, better compression
- Compression can reduce Pallas trace size by a factor of 10

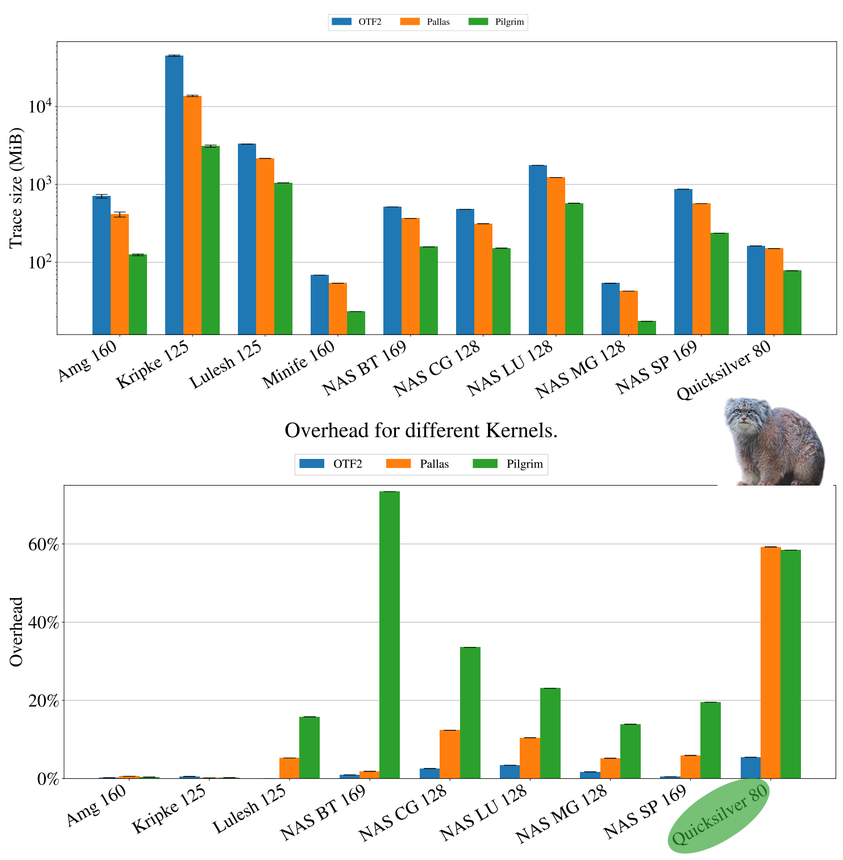
Trace analysis

We have implemented two analysis tools for OTF2, Pallas and Pilgrim: one to **plot a communication matrix**, the other to **detect contention**.

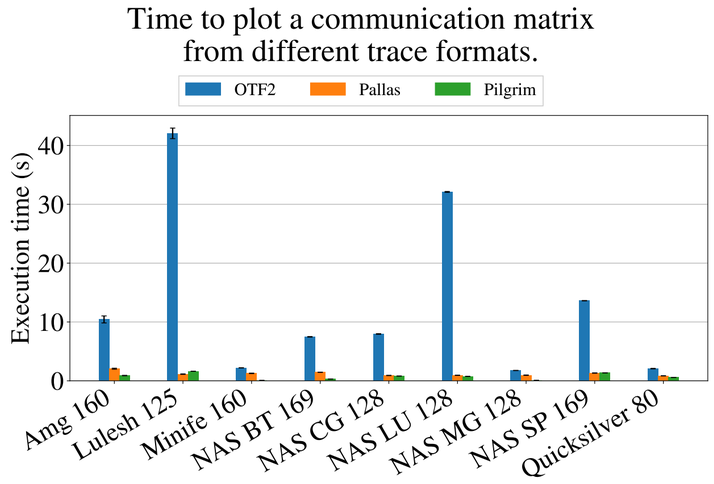
- Both Pilgrim and Pallas run the tools near-instantaneously.
- Pallas uses a lot less memory than Pilgrim.



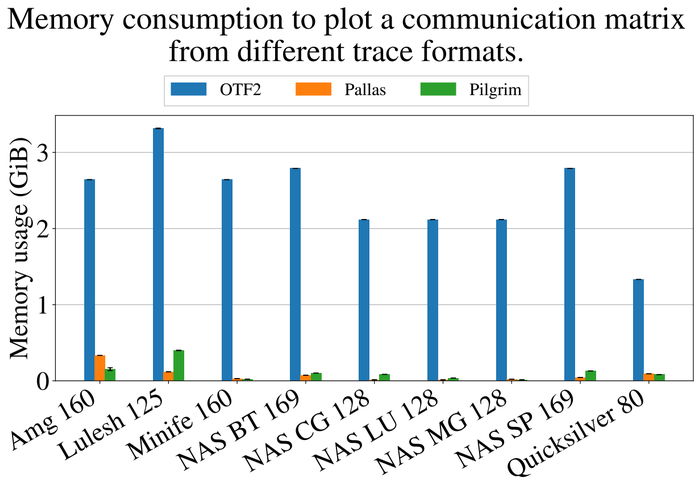
Size of traces for different kernels.



Time to plot a communication matrix from different trace formats.



Memory consumption to plot a communication matrix from different trace formats.



Overhead for different Kernels.

Conclusion

- Low overhead (<5%) for regular applications
- Reasonable trace size compared to alternatives
- Quick and memory-light analyses

Ongoing work

- Provide trace visualization tools
- Pattern detection for irregular applications
- Develop complex trace analysis methods
- Tracing non-MPI kernels (CUDA, StarPU etc)
- Performance analysis at scale (TBs of data)
- Provide better compression techniques

Contact catherine.guelque@ip-paris.fr