RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

FRANCE 2030
PROGRAMME DE RECHERCHE
NUMÉRIQUE POUR L'EXASCALE

Pallas

HPC Trace Analysis at scale

Catherine Guelque Francois Trahay Valentin Honoré

Télécom SudParis - Benagil INRIA research team

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

FRANCE 2030

PROGRAMME DE RECHERCHE
NUMÉRIQUE POUR L'EXASCALE

Introduction & Context

Context

My PhD

Working at Francois Trahay and Valentin Honoré
And also people from INRIA Bordeaux !

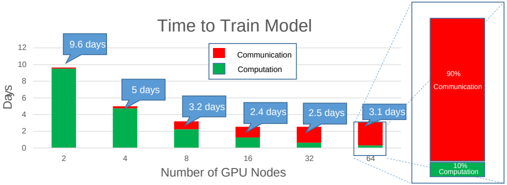
PEPR NumPEx (that's us !)

- Creating the software stack for **exascale** computers
- Alice Recoque (2025): **Heterogenous architectures**
  - 10k+ CPU Nodes
  - 10k+ GPUs
- Various paradigms: **MPI**, **CUDA**, **StarPU**

Context

Scalability issues

- Load-balancing
- Concurrent access to resources
- Interactions between threads
- Non-negligeable communication times



To scale/debug/optimize these apps, **we need performance analysis tools !**

Traces & tracing tools

Traces

- Timeline of an execution
- Stores events with data
  - Timestamps
  - Arguments
  - Callstack
  - ...

Tracing tools

Intercept known function calls (MPI, OMP, CUDA) and log them to create a trace

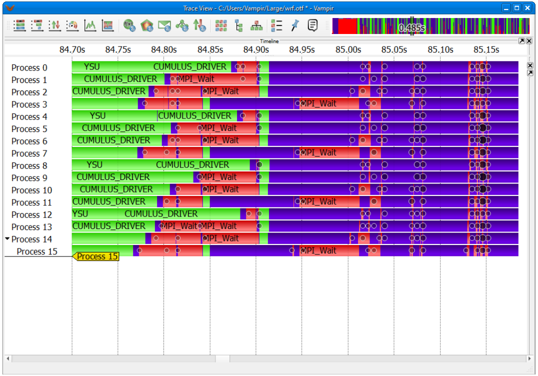


Figure 1: An OTF2 Trace visualised with Vampir.

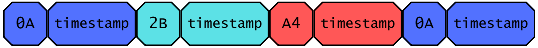
Issue: traces quickly become huge (hard to store and analyse)

Types of traces

Sequential

Array of events in chronological order

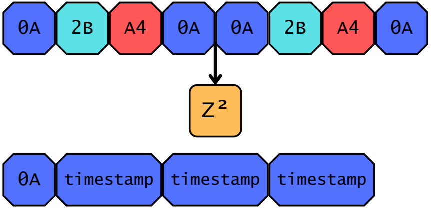
- Straightforward to read & write
- Redundancy → heavy traces



Structural

HPC apps are predictable → include the structure of the program

- Better compression
- More information
- Easier analysis



What do we need ?

We need a new, more **scalable** trace format, with:

- low overhead (unobtrusive)
- structure detection
- scalable analysis
- efficient compression

i.e an **analysis-focused highly compressible trace format**

Pallas

Pallas

Trace format

- **Structural**, **generic** trace format
- Automatic sequence detection
- Provides reading/writing API via C/C++ library
- Provides an OTF2 writing API (compatible with many tools)

EZTrace

- Intercepts MPI/OMP/CUDA calls
- Builds OTF2 traces via OTF2 library
- With our API, creates Pallas traces

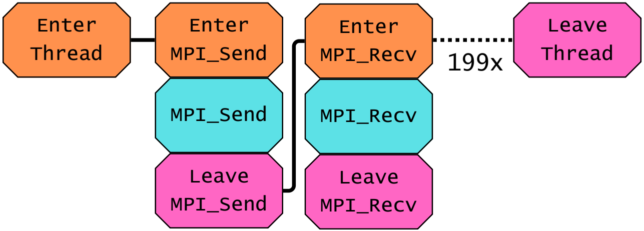
Example: EZTrace

EZTrace

Intercepted MPI function:

- Enter and Leave events = scope
- Punctual event = message sent

```
int main() {
    DO_FOR(200) {
        MPI_Send(...);
        MPI_Recv(...);
    }
}
```



Structure detection

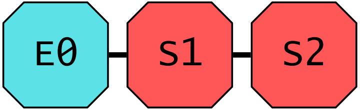
OTF2 to Pallas

- Events are stored as **generic tokens**
- Enter/Leave events are converted to Sequences (makes shorter arrays)
- Sequences and Loops are also generic tokens.

Structure detection

Add a token → Loop detection algorithm
Repetition is detected:

- Check already existing Sequences with hashing function
- Replace repeating Tokens with new Loops token



Structure detection

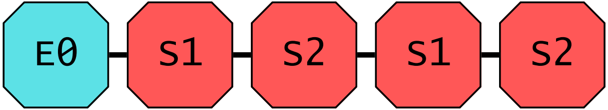
OTF2 to Pallas

- Events are stored as **generic tokens**
- Enter/Leave events are converted to Sequences (makes shorter arrays)
- Sequences and Loops are also generic tokens.

Structure detection

Add a token → Loop detection algorithm
Repetition is detected:

- Check already existing Sequences with hashing function
- Replace repeating Tokens with new Loops token



Structure detection

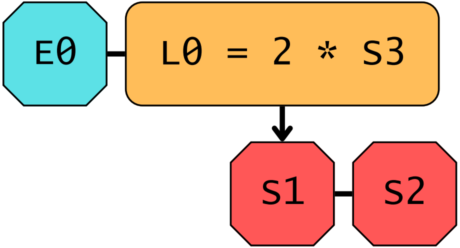
OTF2 to Pallas

- Events are stored as **generic tokens**
- Enter/Leave events are converted to Sequences (makes shorter arrays)
- Sequences and Loops are also generic tokens.

Structure detection

Add a token → Loop detection algorithm
Repetition is detected:

- Check already existing Sequences with hashing function
- Replace repeating Tokens with new Loops token



Structure detection

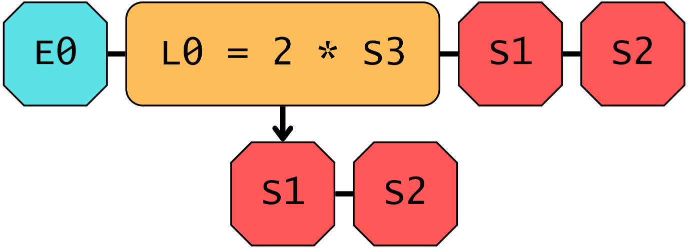
OTF2 to Pallas

- Events are stored as **generic tokens**
- Enter/Leave events are converted to Sequences (makes shorter arrays)
- Sequences and Loops are also generic tokens.

Structure detection

Add a token → Loop detection algorithm
Repetition is detected:

- Check already existing Sequences with hashing function
- Replace repeating Tokens with new Loops token



Structure detection

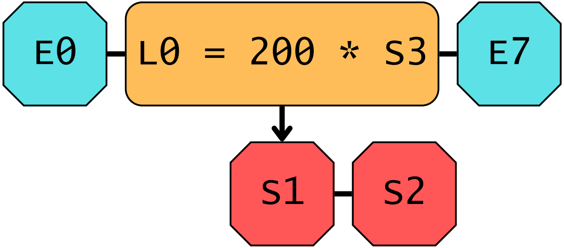
OTF2 to Pallas

- Events are stored as **generic tokens**
- Enter/Leave events are converted to Sequences (makes shorter arrays)
- Sequences and Loops are also generic tokens.

Structure detection

Add a token → Loop detection algorithm
Repetition is detected:

- Check already existing Sequences with hashing function
- Replace repeating Tokens with new Loops token



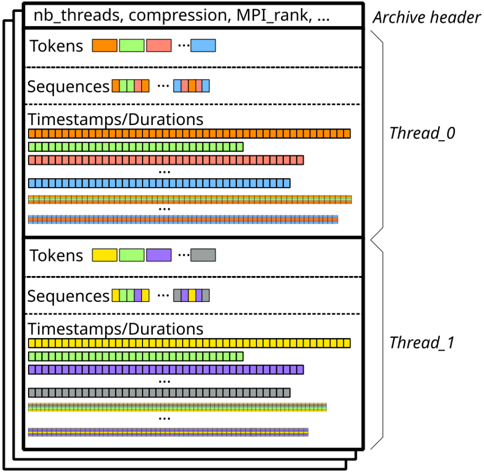
Trace format

Parallel Write/Read

- One folder per process
- No concurrent writing
- Easy parallel reading

Smart data storage & retrieval

- Structure, statistics & metadata are independent of data
  - On-demand accessibility
- Durations are grouped by tokens
  - Decent compression



RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

FRANCE 2030

PROGRAMME DE RECHERCHE
NUMÉRIQUE POUR L'EXASCALE

Benchmarks and Evaluations

Experimental parameters

- NAS Parallel Benchmarks, AMG, MiniFE, Lulesh & Quicksilver
- Every experiment was run on **Jean-Zay**
- Tested with
  - OTF2 using EZTrace
  - Pallas using EZTrace and OTF2 API
  - Pilgrim (trace format & event interception)
- Almost all experiences on 4096 MPI processes.

Overhead

Key points

- **Lower is better !**
- OTF2 < Pallas < Pilgrim
- Low overhead for Pallas
- Pilgrim struggles with event variety

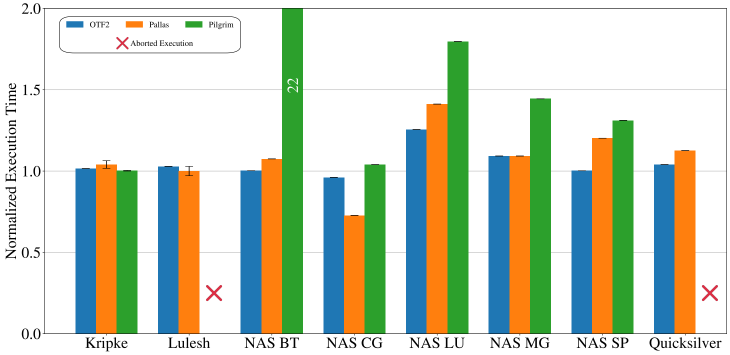


Figure 2: Execution time of the different tracing scenario, normalized by the vanilla run of the application, for the different applications over 4096 MPI processes.

Trace size & compression

Key points

- **Lower is better∗ !**
- Pilgrim < Pallas < OTF2
- OTF2 ≈ 10 · Pilgrim
- Pilgrim collects less information than EZTrace
- Pigrim **compresses all the timestamps together**.

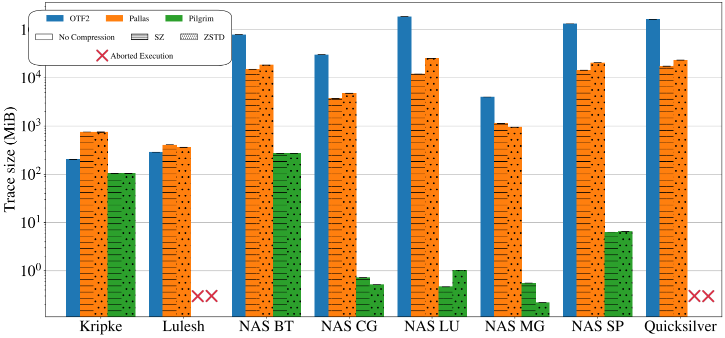
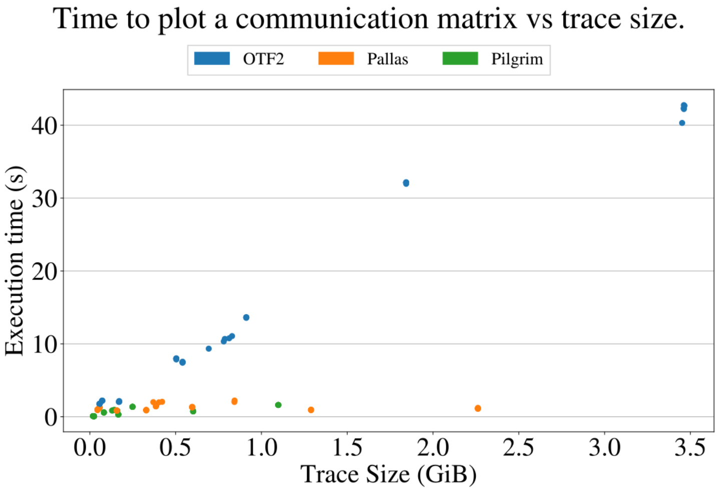


Figure 3: Comparison of trace size for different trace formats, when tracing the different applications over 4096 MPI processes.

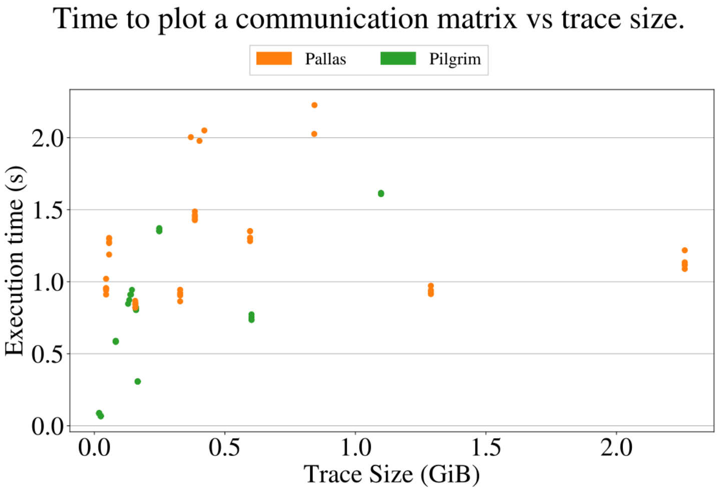
Analysis speed: Communication Matrix

Key points

- Pilgrim/Pallas $\lll$ OTF2
- Pilgrim/Pallas $\rightarrow$ scalable
- Not pictured: Kripke OTF2 analysis was 450s



Analysis speed: Communication Matrix



Key points

- **Pallas ≈ Pilgrim**
- Analysis speed uncorrelated with actual trace size.
- Pallas reads only the header.

Conclusion

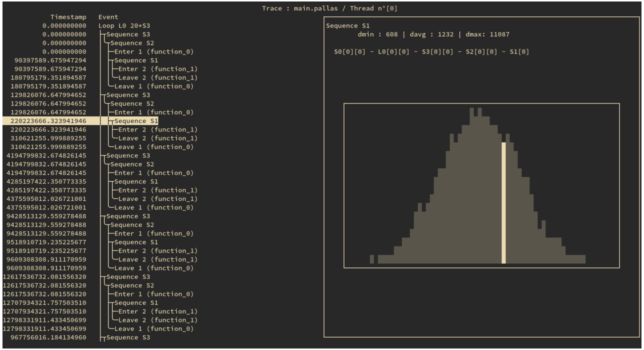
Conclusion

Pallas:

- ✓ Low Overhead
  - ☞ That scales well
- ✓ Structure detection
- ✓ Efficient timestamp storage with compression / encoding
- ✗ Efficient compression
- ✓ Basic scalable & performant analysis
- ✓ On demand-trace loading and exploration

Future developments

- Evaluating Pallas tracing on non-MPI kernels
- Evaluating Pallas at larger scales (currently testing 4k threads)
- Inter-trace compression → "Vertical" scalability
- Testing more efficient compression techniques
- More complex and scalable analysis
- Automatic event filtering



Appendix

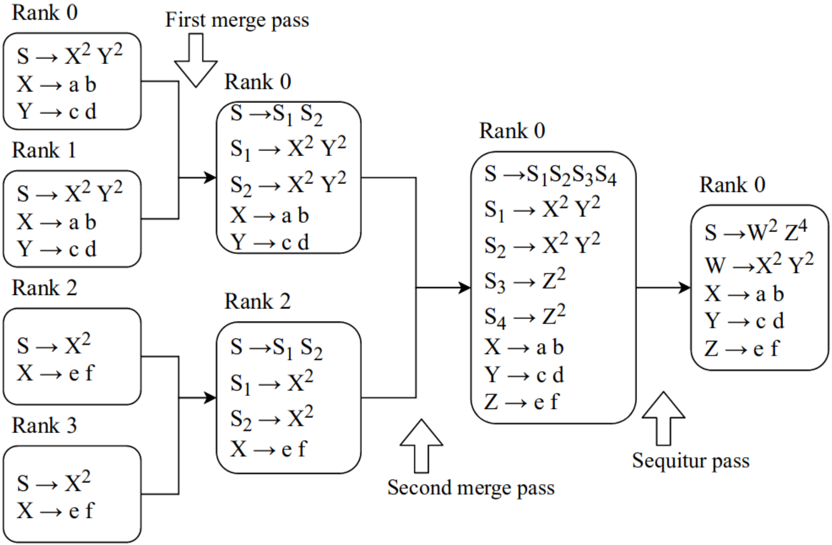
Timestamp compression & encoding

Durations are similar $\rightarrow$ easily compressible
Different storage options:

- No timestamps (Structure only)
- Encoding:
  - Removed leading 0s
  - Replace leading 0s (as presented before)
- Compression:
  - ZSTD
  - SZ
  - ZFP
  - Bin-based (similar to QSDG)
  - Histogram-based (same thing but Gaussian distribution)

} Lossy compression (SZ, ZFP, Bin-based, Histogram-based)

(Pilgrim) Inter-trace compression



Using NCURSES