TELECOM SudParis | INSTITUT POLYTECHNIQUE DE PARIS | Inria | Institut Mines-Télécom | FRANCE 2030

PALLAS

HPC Trace Analysis at scale

Catherine Guelque Francois Trahay **Valentin Honoré**

ENSIIE - Benagil INRIA research team

Introduction & Context

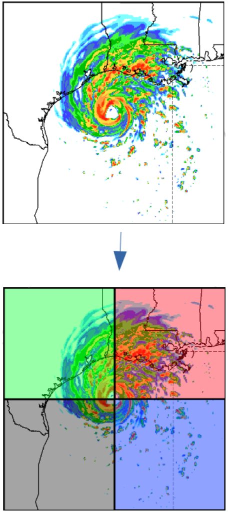
Context

High Performance Computing

- Focus on developing **parallel processing algorithms and software** to divide programs in **small independent parts**.
- Various paradigms: **MPI**, **CUDA**, **StarPU**

PEPR NumPEx

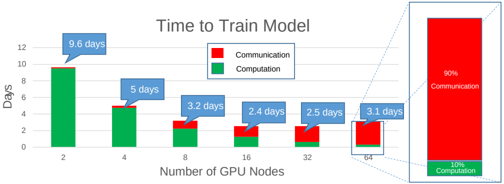
- Creating the software stack for **exascale** computers
- Jules Vernes (2025): **Heterogenous architectures**
  - 10k+ CPU Nodes
  - 10k+ GPUs

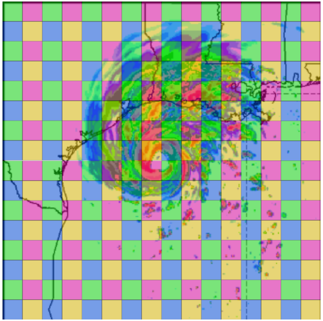


Context

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

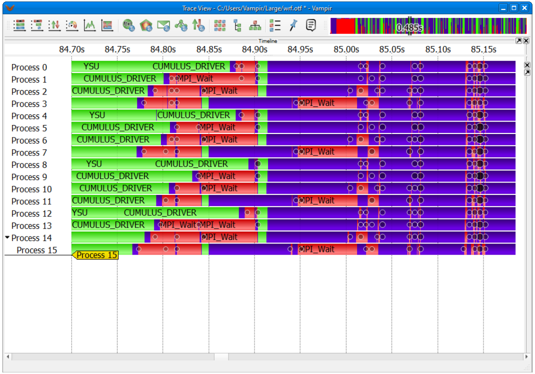


Figure 1: An OTF2 Trace visualised with Vampir.

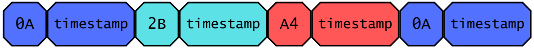
Issue: traces quickly become huge (hard to store and analyze/visualize)

Types of traces

Sequential (OTF2 format)

Array of events in chronological order

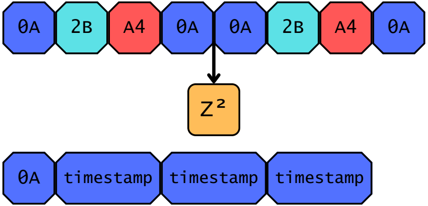
- Straightforward to read & write
- Redundancy → heavy traces



Structural (Pilgrim format)

HPC apps are predictable → include the structure of the program

- Better compression
- More information
- Easier analysis



Exascale tracing : What do we need ?

We need a new, more **scalable** trace format, with the best of all worlds:

- low overhead (unobtrusive)
- structure detection
- scalable analysis
- efficient compression
- generic (MPI, CUDA, OpenMP, StarPU etc)

i.e a **generic 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

EZTrace

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

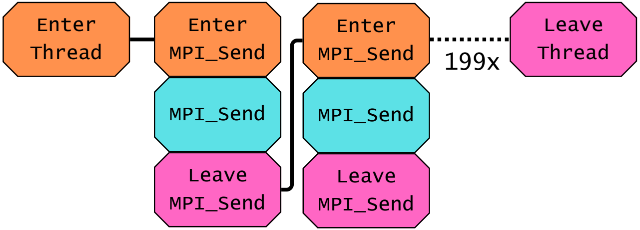
Example: EZTrace

EZTrace

Intercepted MPI function:

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

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



Structure detection

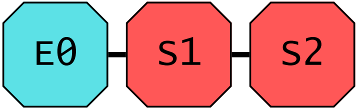
OTF2 to Pallas

- 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

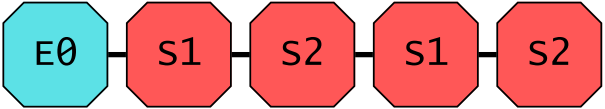
OTF2 to Pallas

- 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

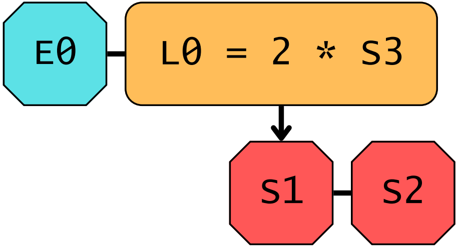
OTF2 to Pallas

- 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

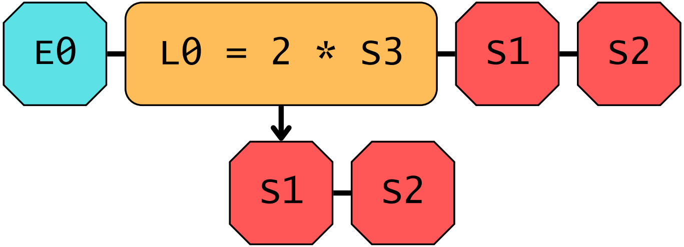
OTF2 to Pallas

- 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

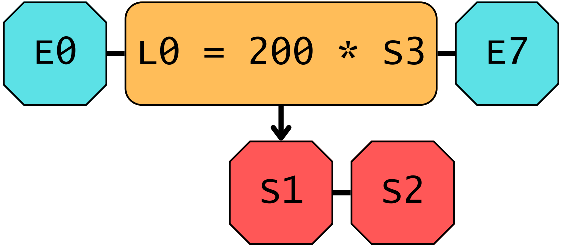
OTF2 to Pallas

- 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



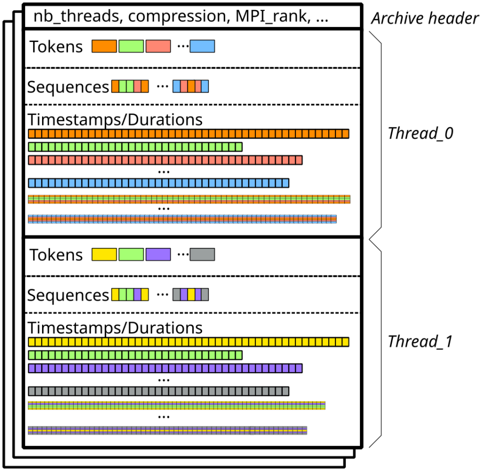
Trace format

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



Benchmarks and Evaluations

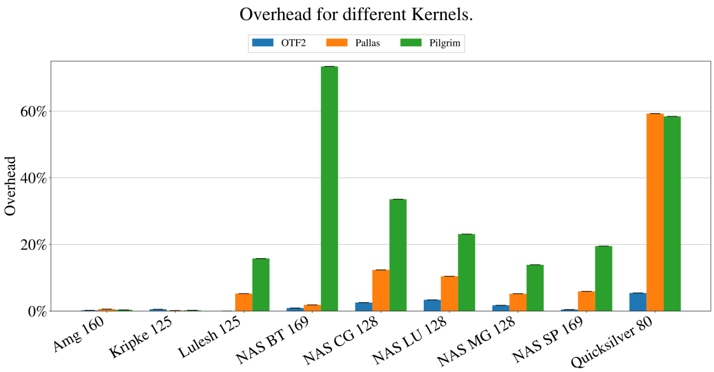
Experimental parameters (focus on MPI so far)

- NAS Parallel Benchmarks, AMG, MiniFE, Lulesh & Quicksilver
- Every experiment was run on **Jean-Zay**
- 5 iterations per benchmark
- Tested with
  - OTF2 using EZTrace
  - Pallas using EZTrace and OTF2 API
  - Pilgrim (trace format & event interception)
- Comparison with vanilla run

Overhead

Key points

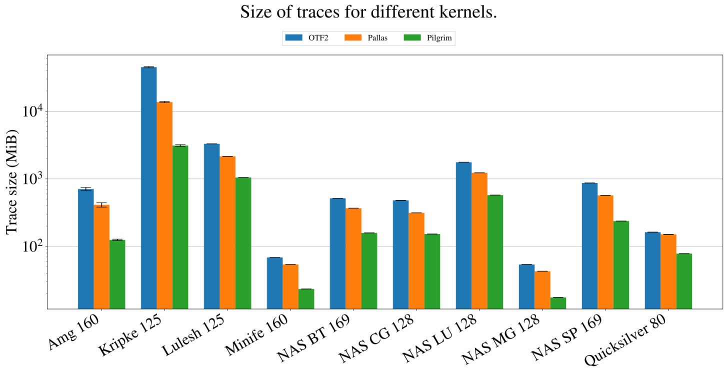
- **Lower is better !**
- OTF2 < Pallas < Pilgrim
- Low overhead for Pallas
- 60% OH for **Quicksilver** → no pattern



Trace size (no compression here)

Key points

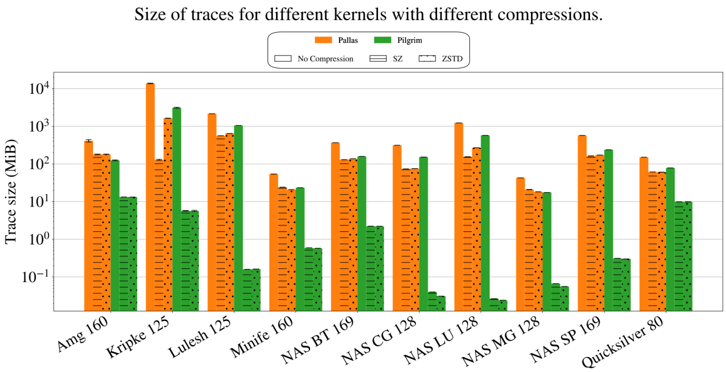
- **Lower is better !**
- OTF2 > Pallas > Pilgrim
- Pallas ≈ 10 · Pilgrim
- Possibility: Pilgrim collects less information than EZTrace (Strings, thread names, etc.)



Compression

Key points

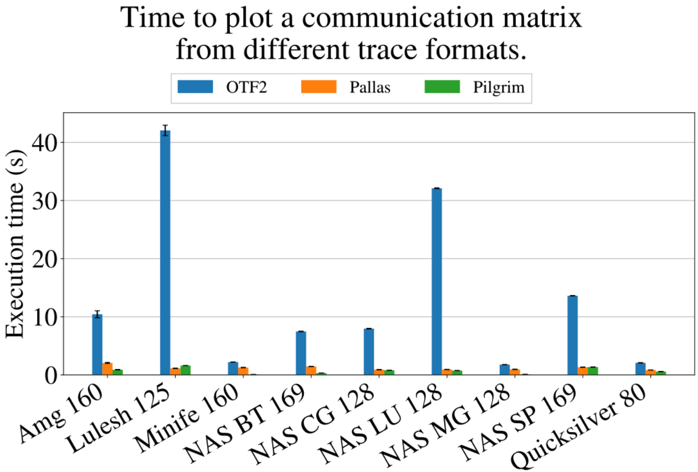
- **Lower is better !**
- Pilgrim ⋘ Pallas
- Pilgrim **compresses all the timestamps together**.



Analysis speed: Communication Matrix

Key points

- **Lower is better !**
- Pilgrim/Pallas ⋘ OTF2
- Not pictured: Kripke OTF2 analysis was 450s

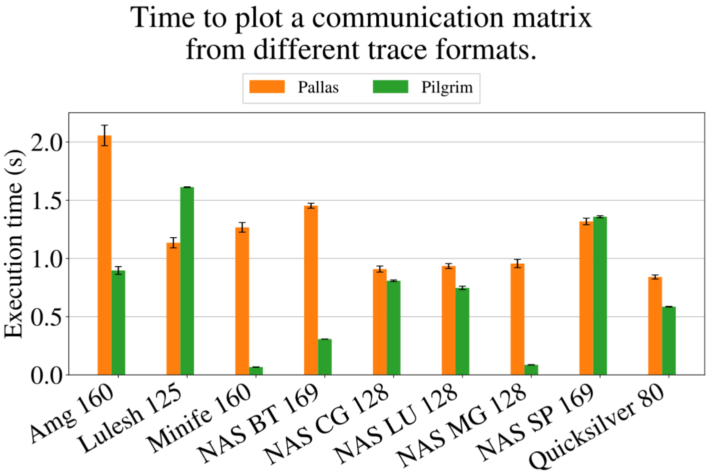


Analysis speed: Communication Matrix

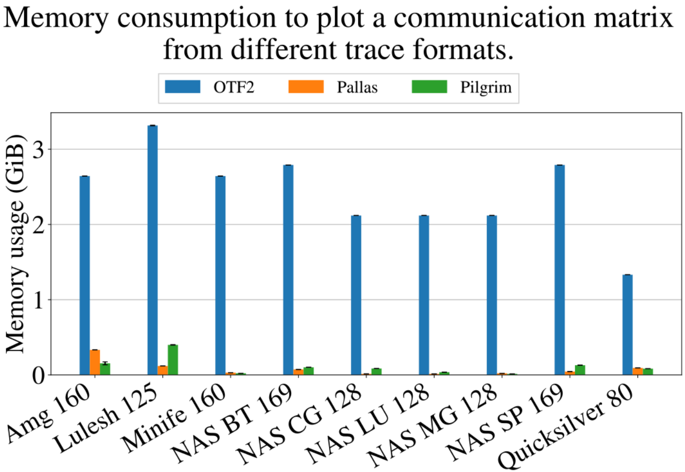
Inria

Key points

- **Pallas ≈ Pilgrim**
- Analysis speed uncorrelated with actual trace size.



Memory usage: Communication Matrix



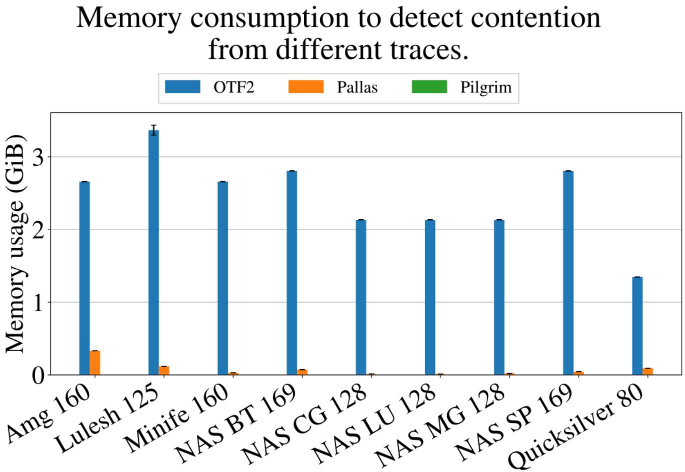
Key points

- **Pallas ≈ Pilgrim**
- Analysis speed uncorrelated with actual trace size.

Memory consumption: Contention detection

Key points

- **Lower is better !**
- Pallas $\ll$ OTF2
- No data for Pilgrim yet (should consume more memory).



Conclusion

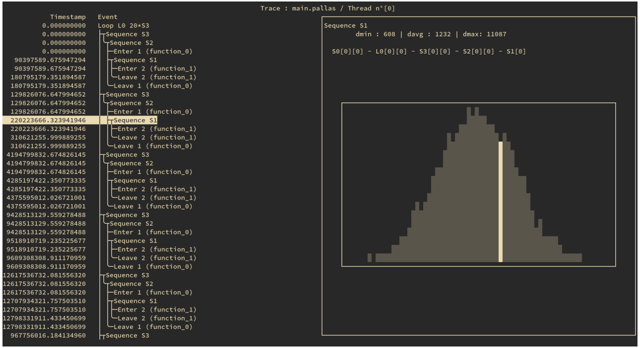
Conclusion

Pallas:

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

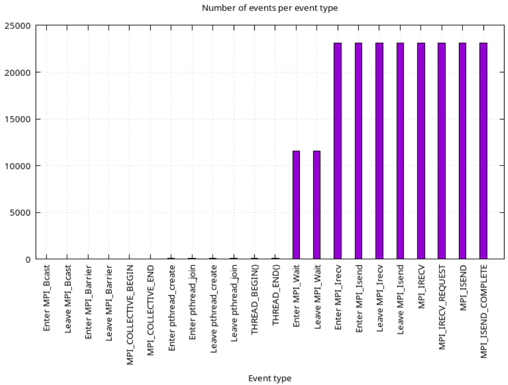
Future developments

- Tracing non-MPI kernels
- Inter-trace compression → "Vertical" scalability
- Testing more efficient compression techniques
- More complex and scalable analysis (`pallas_tui`, scheduling issues etc)
- Online analysis



On-going work

- Scalability up to thousand(s) threads (nothing crashed up to ~ 2000 :))
- On-the-fly buffer flushing on disks
- Tracing non-MPI kernels (focus on StarPU in the PEPR)



Thank you for your attention!

Theorem (A little theorem for Yves)

Because there cannot be a presentation without a theorem!

Appendix

Timestamp compression & encoding

Durations are similar → 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

Trace validity

Benchmark	OTF2	Pallas	Pilgrim
NAS LU	2 446 988	2 446 988	2 446 988
NAS CG	671 280	671 280	671 312
NAS FT	800	800	832
NAS MG	138 480	138 480	136 880
NAS SP	308 448	308 448	430 288
LULESH	5 831 832	5 831 832	6 596 229
MiniFE	94 768	94 768	98 320
AMG	15 805 354	887 466	3 671 584

Table 1: Number of MPI calls recorded by EZTrace/OTF2, EZTrace/Pallas and Pilgrim for AMG

- Pilgrim records `MPI_Comm_rank`, `MPI_Comm_size` and `MPI_Get_count`
- AMG has a busy-waiting loop using `MPI_IProbe`

Trace validity

Benchmark	OTF2	Pallas	Pilgrim
NAS LU	2 446 988	2 446 988	2 446 988
NAS CG	671 280	671 280	671 312
NAS FT	800	800	832
NAS MG	138 480	138 480	136 880
NAS SP	308 448	308 448	430 288
LULESH	5 831 832	5 831 832	6 596 229
MiniFE	94 768	94 768	98 320
AMG	15 805 354	887 466	3 671 584

Table 1: Number of MPI calls recorded by EZTrace/OTF2, EZTrace/Pallas and Pilgrim for AMG

- Pilgrim records `MPI_Comm_rank`, `MPI_Comm_size` and `MPI_Get_count`
- AMG has a busy-waiting loop using `MPI_IProbe`

Trace validity

Benchmark	OTF2	Pallas	Pilgrim
NAS LU	2 446 988	2 446 988	2 446 988
NAS CG	671 280	671 280	671 312
NAS FT	800	800	832
NAS MG	138 480	138 480	136 880
NAS SP	308 448	308 448	430 288
LULESH	5 831 832	5 831 832	6 596 229
MiniFE	94 768	94 768	98 320
AMG	15 805 354	887 466	3 671 584

Table 1: Number of MPI calls recorded by EZTrace/OTF2, EZTrace/Pallas and Pilgrim for AMG

- Pilgrim records `MPI_Comm_rank`, `MPI_Comm_size` and `MPI_Get_count`
- AMG has a busy-waiting loop using `MPI_IProbe`

Trace validity

Benchmark	OTF2	Pallas	Pilgrim
NAS LU	2 446 988	2 446 988	2 446 988
NAS CG	671 280	671 280	671 312
NAS FT	800	800	832
NAS MG	138 480	138 480	136 880
NAS SP	308 448	308 448	430 288
LULESH	5 831 832	5 831 832	6 596 229
MiniFE	94 768	94 768	98 320
AMG	15 805 354	887 466	3 671 584

Table 1: Number of MPI calls recorded by EZTrace/OTF2, EZTrace/Pallas and Pilgrim for AMG

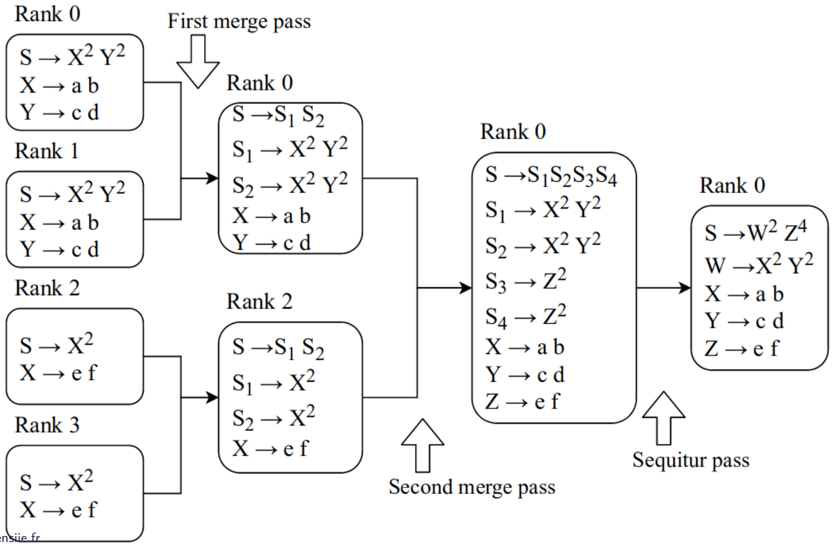
- Pilgrim records `MPI_Comm_rank`, `MPI_Comm_size` and `MPI_Get_count`
- AMG has a busy-waiting loop using `MPI_IProbe`

Benchmark	OTF2	Pallas	Pilgrim
NAS LU	2 446 988	2 446 988	2 446 988
NAS CG	671 280	671 280	671 312
NAS FT	800	800	832
NAS MG	138 480	138 480	136 880
NAS SP	308 448	308 448	430 288
LULESH	5 831 832	5 831 832	6 596 229
MiniFE	94 768	94 768	98 320
AMG	15 805 354	887 466	3 671 584

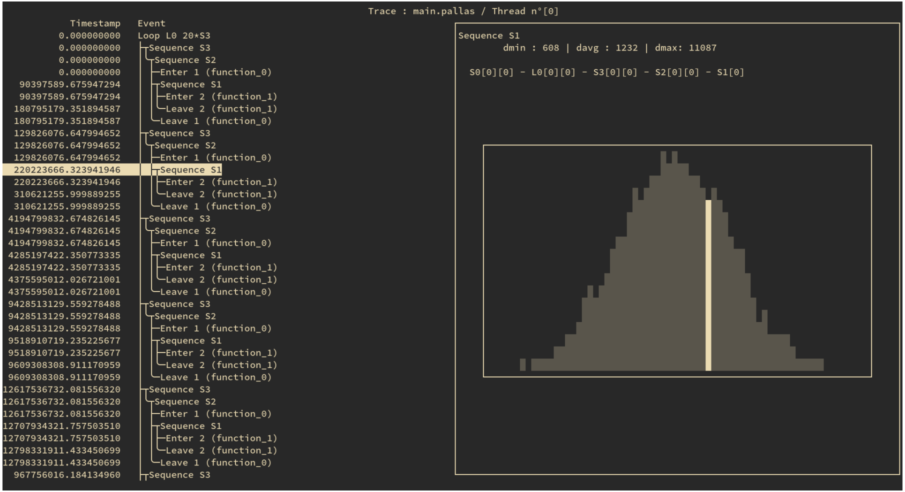
Table 1: Number of MPI calls recorded by EZTrace/OTF2, EZTrace/Pallas and Pilgrim for AMG

- Pilgrim records `MPI_Comm_rank`, `MPI_Comm_size` and `MPI_Get_count`
- AMG has a busy-waiting loop using `MPI_IProbe`

(Pilgrim) Inter-trace compression



Using ncurses



StarPU Analysis : Cholesky