

Liberté Égalité Fraternité





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







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
 - $\circ \ 10k+ \ \mathsf{CPU} \ \mathsf{Nodes}$
 - 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
 - o ...

Tracing tools

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

0	4.70s 84.75s	84.80s	84.855 84.9	Timeline 205 84.955	85.00s 85.0	05s 85.10s	85,156
Process 0	YSU				60 P	0 0	1999
Process 1	CUMULUS_DRI			2	000	00	- aŭ o
Process 2	CUMULUS_DRIVER		MPI_Walt		<u> </u>		-409
Process 3		0 00		OSPI_W		• •	9999
Process 4	YSU				99 P		- 9999
Process 5	CUMULUS_DRIV		MPI_Wait		000		(1 1)
Process 6	CUMULUS_DRIVE		MPI_Walt		0 00		- 1
Process 7		0 0 0		OSPI_W		• •	1999
Process 8	YSU		IS_DRIVER		<u>99 P</u> .	<u> </u>	- 9999
Process 9	CUMULUS_DRIVE		Sours Tange	<u> </u>			- (1)
Process 10	CUMULUS_DRIV		MPI_Walt		o o o	900	0.00
Process 11	CUMULUS_DRIVER			OSPI_W	ait 💿	••	- 1999
Process 12		ULUS_DRIV				400	1999
Process 13	CUMULUS_DRIVER		ait MPI_Wait		0000	400	ogijo
Process 14		0 0	MPI_Wait		P 🖸 🌼	- <u> </u>	1000
Process 15	Process 15			o opt_w	ait 👓		4 892
	FIGUESS 13						

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

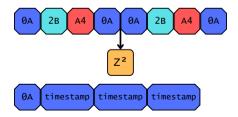
- Straightforward to read & write
- Redundancy \rightarrow heavy traces

Structural

HPC apps are predictable \rightarrow include the structure of the program

- Better compression
- More information
- Easier analysis







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





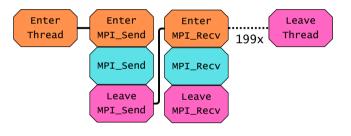
Example: EZTrace

EZTrace

Intercepted MPI function:

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

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



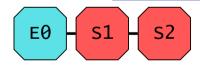


OTF2 to Pallas

- 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

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





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

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



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

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



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

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

$$E0 + L0 = 2 * S3 + S1 + S2$$

 $51 + S2$



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

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



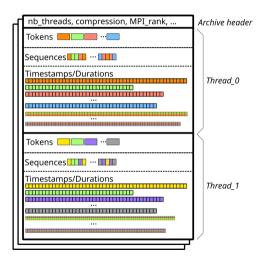
Trace format

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

- 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 !
- $\bullet \ \mathsf{OTF2} < \mathsf{Pallas} < \mathsf{Pilgrim}$
- Low overhead for Pallas
- Pilgrim struggles with event variety

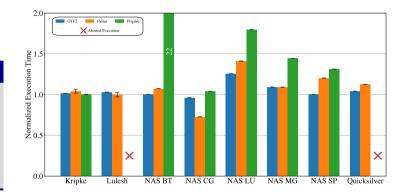


Figure 2: Execution time of the different tracing scenario, normalized by the vanilla run of the application, for the different applications over 4096 MPI processes.

Trace size & compression

Key points

- Lower is better* !
- Pilgrim < Pallas < OTF2
- OTF2 \approx 10 \cdot Pilgrim
- Pilgrim collects less information than EZTrace
- Pigrim compresses all the timestamps together.

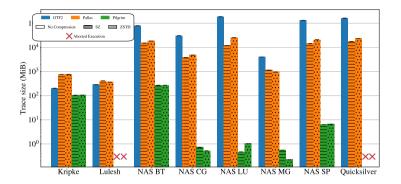
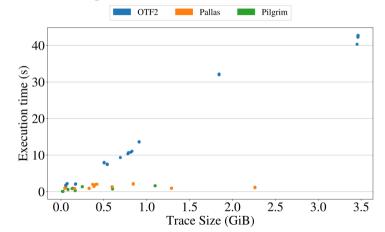


Figure 3: Comparison of trace size for different trace formats, when tracing the different applications over 4096 MPI processes.



Analysis speed: Communication Matrix

Time to plot a communication matrix vs trace size.



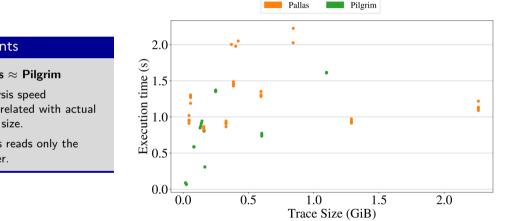
Key points

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



Analysis speed: Communication Matrix

Time to plot a communication matrix vs trace size.



Key points

- Pallas \approx Pilgrim
- Analysis speed uncorrelated with actual trace size.
- Pallas reads only the header.





Conclusion

Conclusion

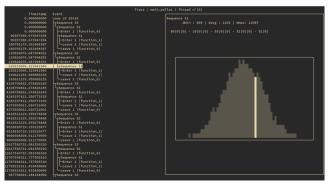
Pallas:

- ✓ Low Overhead
 - 🖙 That scales well
- ✓ Structure detection
- \checkmark Efficient timestamp storage with compression / encoding
- \times Efficient compression
- ✓ Basic scalable & performant analysis
- \checkmark 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 \rightarrow "Vertical" scalability
- Testing more efficient compression techniques
- More complex and scalable analysis
- Automatic event filtering





Appendix



Timestamp compression & encoding

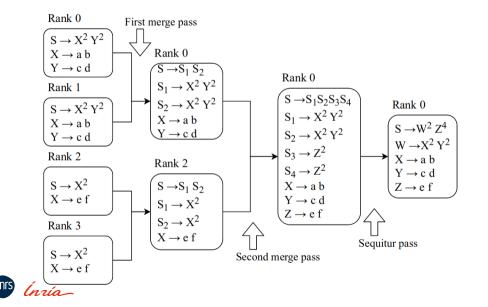
Durations are similar \rightarrow easily compressible Different storage options:

- 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



(Pilgrim) Inter-trace compression



Using NCURSES

Timestamp	Event
0.00000000	Loop L0 20*S3
0.00000000	Sequence S3
0.00000000	Sequence S2
0.00000000	Enter 1 (function 0)
90397589.675947294	-Sequence S1
90397589.675947294	Enter 2 (function 1)
180795179.351894587	Leave 2 (function_1)
180795179.351894587	Leave 1 (function_0)
129826076.647994652	
129826076.647994652	Sequence S2
129826076.647994652	Enter 1 (function_0)
220223666.323941946	TSequence S1
220223666.323941946	Enter 2 (function_1)
310621255.999889255	Leave 2 (function_1)
310621255.999889255	Leave 1 (function_0)
194799832.674826145	TSequence S3
194799832.674826145	Sequence S2
194799832.674826145	Enter 1 (function_0)
285197422.350773335	Sequence S1
285197422.350773335	Enter 2 (function_1)
375595012.026721001	Leave 2 (function_1)
375595012.026721001	Leave 1 (function_0)
428513129.559278488	TSequence S3
428513129.559278488	Sequence S2
428513129.559278488	Enter 1 (function_0)
518910719.235225677	TSequence S1
518910719.235225677	Enter 2 (function_1)
609308308.911170959	Leave 2 (function_1)
609308308.911170959	Leave 1 (function_0)
617536732.081556320	Sequence S3
617536732.081556320	Sequence S2
617536732.081556320	Enter 1 (function_0)
707934321.757503510	Sequence S1
707934321.757503510	Enter 2 (function_1)
798331911.433450699	Leave 2 (function_1)
798331911.433450699	Leave 1 (function_0)
967756016.184134960	Sequence S3

Trace : main	.pallas ,	/ Thread	l n°[0]
--------------	-----------	----------	---------

quence Sl dmin : 608 | davg : 1232 | dmax: 110

S0[0][0] - L0[0][0] - S3[0][0] - S2[0][0] - S1[0]

