Benchmarks and Evaluations



DE PARIS







PALLAS

HPC Trace Analysis at scale

Catherine Guelque Francois Trahay Valentin Honoré

ENSIIE - Benagil INRIA research team

Pallas - valentin.honore@ensiie.fr

1/22

INTRODUCTION & CONTEXT







Pallas 0000C Benchmarks and Evaluations

Conclusior 00000



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





Benchmarks and Evaluations

Conclusion 00000



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 !

Benchmarks and Evaluations



Traces

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

Traces & tracing tools

• • • •

Tracing tools

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

V			Tra	te View - Cr/U	sers/Vempir/Large	/wif.otf * - Vemp	ir 👘			
1000 177 IN 1	a 🗛 🔳	0. 🐽	S 🔥	情報	h & 15	1 E		0.4883		
					Timelos	/		and consider a		212
84	1.70s 84	.75s 8	4.80s	84.85s	84.90s	84.95s	85,00s	85.05s 85	.10s 85.15s	
Process 0	YSU	(UMULUS	DRIVER	00 0		90	o oo	0000	×
Process 1	CUMULU	JS_DRIVER	ognpi	_Wait	•		0 0	0 00	e de la compo	2
Process 2	CUMULUS_	DRIVER	00	MPI_W	ait		000	0 000	0000	
Process 3		9				OMPI_Wa	sit 💽 😐 🗠	6 0	9999	
Process 4	YSU		CUMULU	S_DRIVER				0 00	9999	
Process 5	CUMULU	S_DRIVER	9	MPI_Wa	it 💡 👘		0 0	0 00	0.00	
Process 6	CUMULUS	DRIVER	00	MPI_W	ait		0 0 0	p (000)	0.00	
Process 7		0				OMPI_Wa	sit 🔹 🔍	() ()	1999	
Process 8	YSU		CUMULUS	_DRIVER				0 00	P999	
Process 9	CUMULUS	DRIVER		MPI_Wa	it 🥐 🔛		00	0 00	0.00	
Process 10	CUMULU	S_DRIVER	0	MPI_W	ait		0 0 0	p 🚥	0.00	
Process 11	CUMULUS_	DRIVER	0.0			OMPI_Wa	sit 🔹 🔍		999	
Process 12	YSU	CUMULU	IS_DRIVE	R				0 000	1999	
Process 13	CUMULUS_	DRIVER	MPI_Wa	it•MPI_Wa	it 💡 🔛		000	0 900	0000	
 Process 14 		6) (P	MPI_W	ait		P 🔍 🤅	o 🐽	0000	
Process 15	Descrete LI					Q MPI_Wa	sit 💽 o	9.0	9.89	
	100055 1.	2								
e l										•

Figure 1: An OTF2 Trace visualised with Vampir.

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

Types of traces

Palla 000

Benchmarks and Evaluations 000000000 Conclusion



Sequential (OTF2 format)

Array of events in chronological order

- Straightforward to read & write
- $\blacksquare \ {\sf Redundancy} \to {\sf heavy \ traces}$

Structural (Pilgrim format)

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

- Better compression
- More information
- Easier analysis





Benchmarks and Evaluations

Conclusion

Exascale tracing : What do we need ?

Innío

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









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





Ínría_

EZTrace

Intercepted MPI function:

Example: EZTrace

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

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



Pallas ○○○●○

Benchmarks and Evaluations



OTF2 to Pallas

Structure detection

- 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



Pallas ○○○●○ Benchmarks and Evaluations 000000000 Conclusion



OTF2 to Pallas

Structure detection

- 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



Pallas ○○○●○ Benchmarks and Evaluations 000000000 Conclusion



OTF2 to Pallas

Structure detection

- 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

Structure detection

- 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

Structure detection

- 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

Benchmarks and Evaluations



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 for different Kernels.

Benchmarks and Evaluations

Conclusion 00000

Trace size (no compression here)





Size of traces for different kernels.

Compression

Conclusio 00000



Size of traces for different kernels with different compressions.



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

Benchmarks and Evaluations

Conclusior 00000

Analysis speed: Communication Matrix



Time to plot a communication matrix from different trace formats.

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



Benchmarks and Evaluations

Conclusior 00000

Analysis speed: Communication Matrix



Time to plot a communication matrix from different trace formats.

- $\blacksquare \ \textbf{Pallas} \approx \textbf{Pilgrim}$
- Analysis speed uncorrelated with actual trace size.



Benchmarks and Evaluations

Conclusio 00000

Memory usage: Communication Matrix



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

- $\blacksquare \ \textbf{Pallas} \approx \textbf{Pilgrim}$
- Analysis speed uncorrelated with actual trace size.



Pallas 0000C Benchmarks and Evaluations ○○○○○○○●

Conclusior 00000

Memory consumption: Contention detection



Memory consumption to detect contention from different traces.



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



CONCLUSION







Conclusion



Pallas:

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

Benchmarks and Evaluations

Conclusion 00●0<u>0</u>

Future developments



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

	Trace : main.pr	illas / Thread nº[0]
Timestamp		
0.00000000	Loop L0 20×53	Sequence S1
0.00000000	TSequence 53	dwin : 608 davg : 1232 dmax: 11087
0.00000000	Sequence 52	
0.00000000	-Enter 1 (function_0)	50[0][0] - L0[0][0] - 53[0][0] - 52[0][0] - 51[0]
90397589.675947294	TSequence S1	
90397589.675947294	⇒Enter 2 (function_1)	
180795179.351894587	Scheave 2 (function_1)	
180795179.351894587	Leave 1 (function_0)	
129826076.647994652	Sequence 53	
129826076.647994652	Sequence S2	
129826076.647994652	→Enter 1 (function_0)	
220223666.323941946	TSequence 51	
220223666.323941946	-Enter 2 (function_1)	
310621255.999889255	-Leave 2 (function_1)	
310621255.999889255	Leave 1 (function_0)	
4194799832.674826145	TSequence S3	
4194799832.674826145	Sequence 52	
4194799832.674826145	-Enter 1 (function_0)	
4285197422.350773335	Sequence S1	
4285197422.350773335	→Enter 2 (function_1)	
4375595012.026721001	Leave 2 (function_1)	
4375595012.026721001		
9428513129.559278488	TSequence 53	
9428513129.559278488	Sequence S2	
9428513129.559278488	←Enter 1 (function_0)	
9518910719.235225677	TSequence S1	
9518910719.235225677	-Enter 2 (function_1)	
9609308308.911176959	Generation_1)	
9609308308.911178959	Leave 1 (function_0)	
12617536732.081556320	Sequence S3	
12617536732.081556320	Sequence S2	
12617536732.081556328	-Enter 1 (function_0)	
12707934321.757503510	TSequence S1	
12707934321.757503510	-Enter 2 (function_1)	
12798331911.433458699	Leave 2 (function_1)	
12798331911.433458699		
967756016.184134968		

On-going work

Pallas 00000

Benchmarks and Evaluations

Conclusion 000●0

Ínría_

- \blacksquare 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)



Benchmarks and Evaluations

Conclusion

Thank you for your attention!

Ínnía_

Theorem (A little theorem for Yves)

Because there cannot be a presentation without a theorem!

Appendix

Timestamp compression & encoding

Ínzia

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

Ínría_

000000

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

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

Ínría_

000000

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

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

Ínría_

000000

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

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

Ínría_

000000

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

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

Ínría_

000000

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

- 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

Ínnía -

000000



Using ncurses

Timestamp 0.000000000

0.000000000 0.000000000

6.0800800800

90397589.675947294

180795179.351894587 180795179.351894587 129826076.647994652

129826076.647994652 129826076.647994652 220223666.323941946

220223666.323941946 310621255.999889255 310621255.999889255 4194799832.674826145 4194799832.674826145 4194799832.674826145 4285197422.350773335 4285197422.350773335 4375595012.026721001 4375595012.026721001 9428513129.559278488 9428513129.559278488 9518910719.235225677 9609308308.911170959 9609308308.911170959 12617536732.081556320 12617536732.081556320

12617536732.081556320 12707934321.757503510 12707934321.757503510 12798331911.433450699 12798331911.433450699 967756016.184134960 Frsequ

En

TSe

C,

-Sequ

TSe

Se

Trace : main.pa	llas / Thread nº[0]
0 20*53 ence 53	Sequence S1 dmin : 608 davg : 1232 dmax: 11087
<pre>uence 52 uence 52 uence 51 uence 52 uence 51 uence 5</pre>	

(nnía-

StarPU Analysis : Cholesky

ínnía_



Number of events per event type