RÉPUBLIQUE FRANÇAISE
Liberté Égalité Fraternité

PROGRAMME DE RECHERCHE
NUMÉRIQUE POUR L'EXASCALE

Automatic Multi-Versioning of Computation Kernels

Exa-Soft Annual Meeting

Raphaël Colin

September 25th 2025

Inria CAMUS, University of Strasbourg - ICPS team
PhD advisors: Philippe Clauss & Thierry Gautier

Work Package 2

Outline

1. Context
2. Multi-versioning
3. Implementation in Apollo
4. Evaluation
5. Future work

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

FRANCE 2030

PROGRAMME DE RECHERCHE
NUMÉRIQUE POUR L'EXASCALE

cea

cnrs

Inria

Context

Motivation

Compute kernels :

- Called many times
- Critical for performance

Optimization decisions

- Not always suited for the current execution context (software and hardware)

Motivation

Compute kernels :

- Called many times
- Critical for performance

Optimization decisions

- Not always suited for the current execution context (software and hardware)

Multi-Versioning

- **generate** multiple versions of a given compute kernel.
- **select** an efficient version for the current **execution context**.

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

FRANCE 2030
PROGRAMME DE RECHERCHE
NUMÉRIQUE POUR L'EXASCALE

cea cnrs Inria

Multi-versioning

Context and versions

Execution Context

- Kernel parameters (function arguments)
- Available hardware resources

Version generation

- Optimization parameters: loop tiling (tile size), loop unrolling (unroll factor), etc.

Tools

- Clang / LLVM
- Polyhedral optimizations (Bondhugula et al. 2008; Feautrier and Lengauer 2011)
- Apollo (Sukumaran Rajam 2015; Martinez Caamaño 2016; Martinez Caamaño et al. 2017)

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

FRANCE 2030
PROGRAMME DE RECHERCHE
NUMÉRIQUE POUR L'EXASCALE

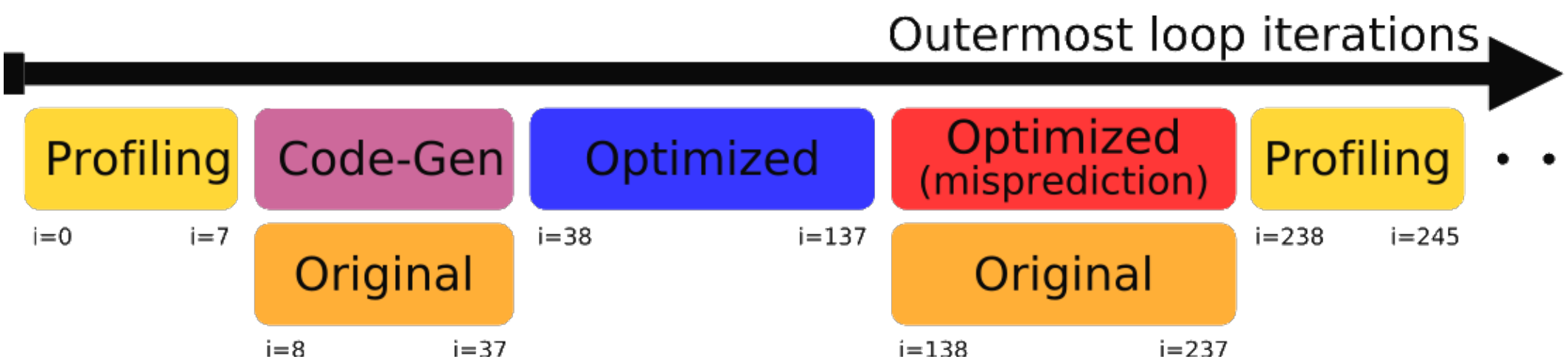
cea cnrs Inria

Implementation in Apollo

Apollo

(Sukumaran Rajam 2015; Martinez Caamaño et al. 2017)

- Analysis of memory accesses in a loop nest
- Speculation
- Rollback if needed
- Polyhedral transformations (Pluto (Bondhugula et al. 2008))



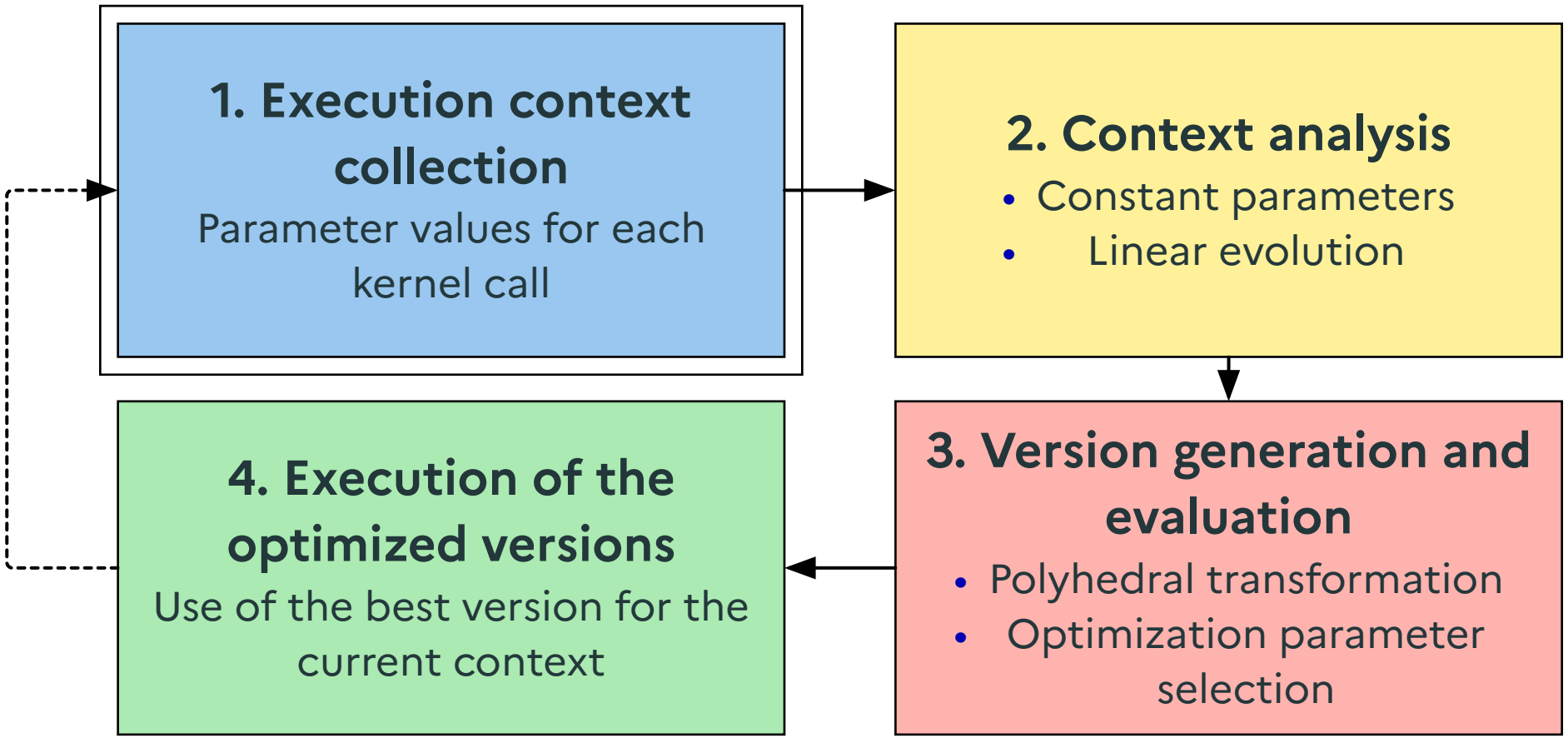
Loop nest execution with Apollo.

Multi-versioning in Apollo

Limitations of current multi-versioning in Apollo (Lazcano et al. 2020) :

- Context analysis is limited
- Optimization parameters are chosen before runtime
- Depends on the execution flow of the original program

Multi-versioning in Apollo



Multi-versioning system design.

Compilation

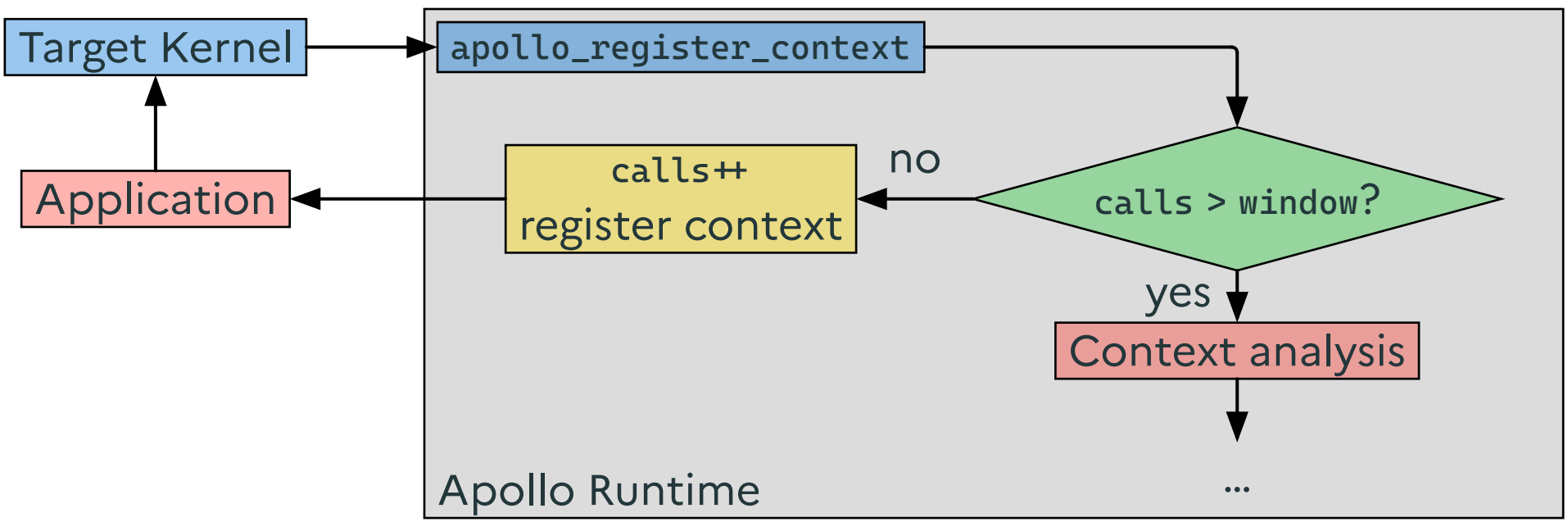
```c
#pragma apollo kernel buf(A, n)
double kernel(double *A, size_t n) {
  // ...
}
```

`pragma apollo kernel` usage example.

- Wrappers are generated to interact with the runtime.
- Declaration of the buffers used in the kernel and their size in order to allocate the right amount of memory.

Execution context collection

- `apollo_register_context` is the main entry point of the runtime
- `calls` = number of registered calls to the kernel
- `window` = length of the observation phase (in number of calls)



Execution contexts collection

Representation

Kernel calls are modeled as parameter vectors:

$$\texttt{kernel}(p_1, p_2, \ldots, p_n) \rightarrow (p_1, p_2, \ldots p_n)$$

Parameters are assumed to be unsigned 64 bits integers.

Representation

Kernel calls are modeled as parameter vectors:

$$\texttt{kernel}(p_1, p_2, ..., p_n) \rightarrow (p_1, p_2, ...p_n)$$

Parameters are assumed to be unsigned 64 bits integers.

Analysis

Given a set of vectors (size = `window`), analyze the aligned vectors.
→ linear evolution for each parameter.

TODO

handling pointers, floating point parameters, hardware parameters, etc.

Optimization and parameter space search

Tiling (tile size)

Apollo

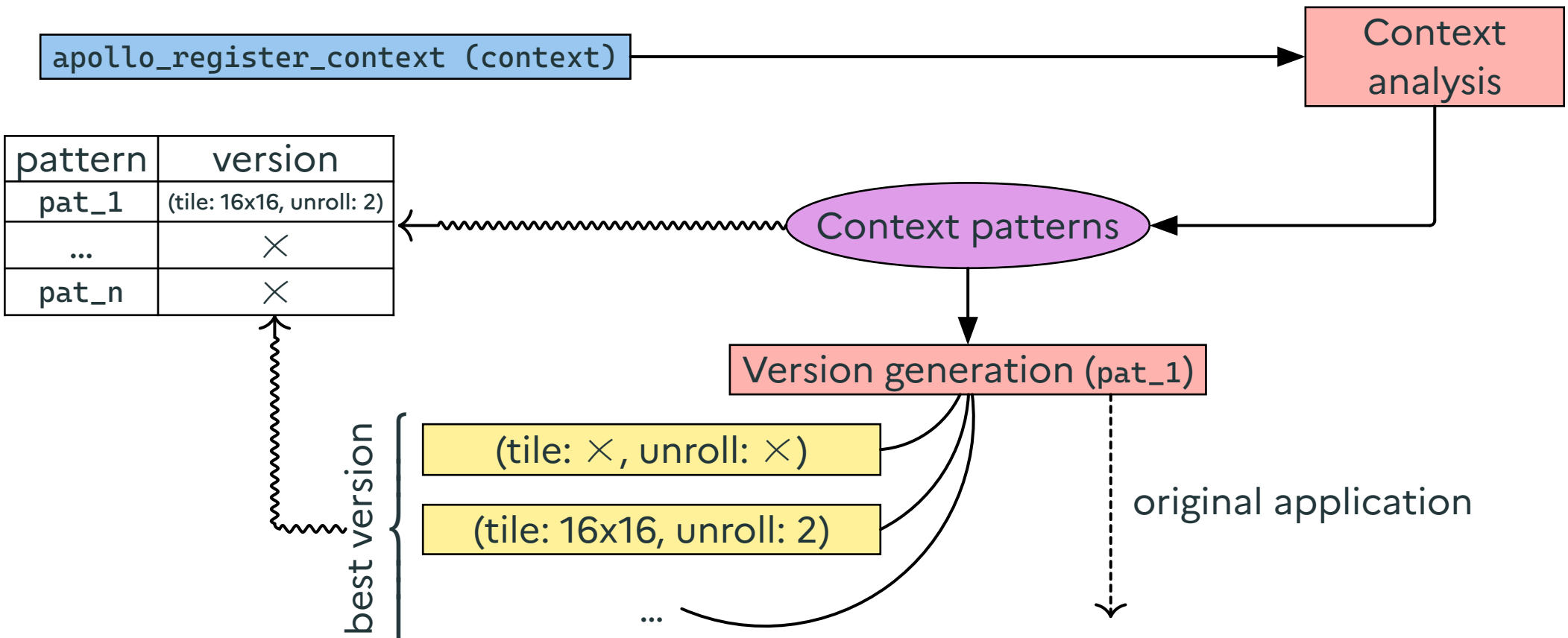
Modification of Apollo's optimization parameters (passed to Pluto).

Why tile sizes?

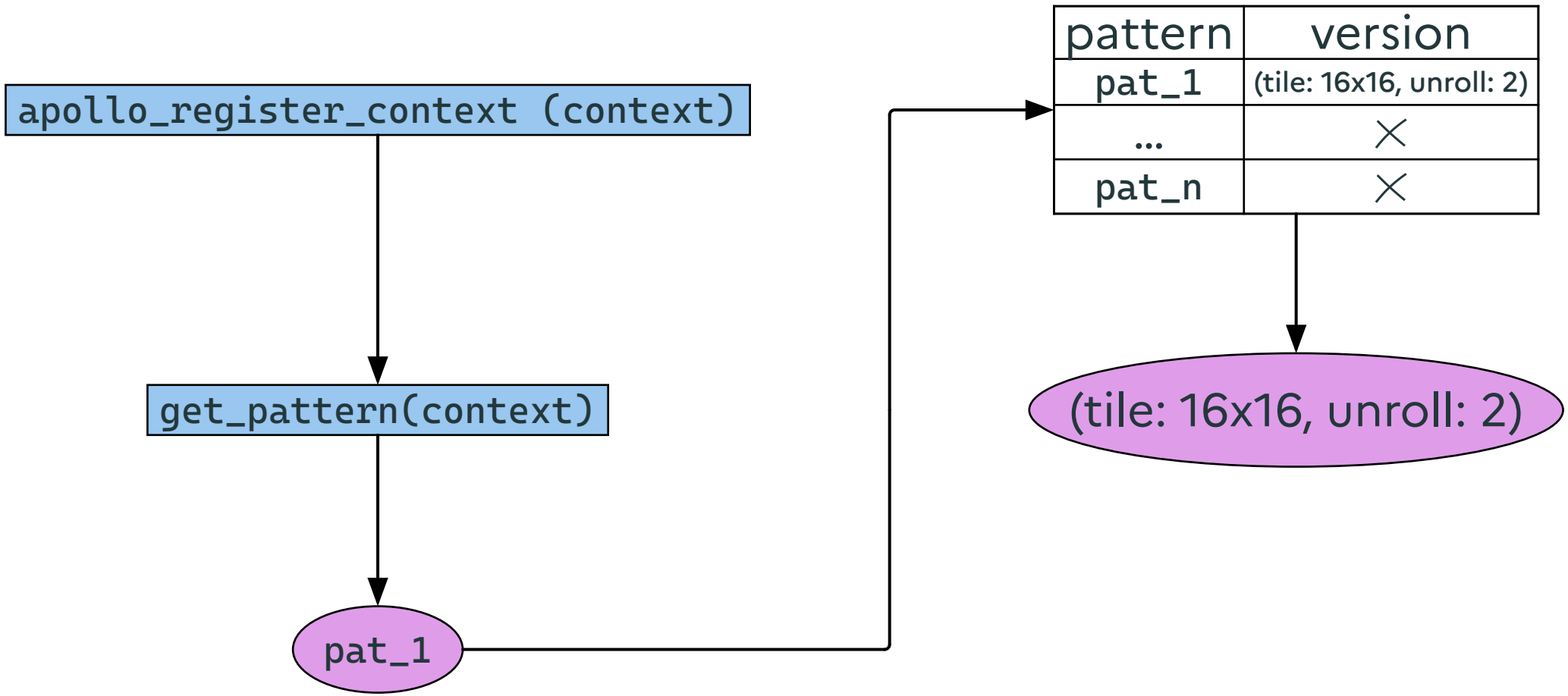
T0	T1	T2	time
2	2	2	1.923s
2	2	4	1.836s
...	...	...	...
32	32	32	0.186s
...	...	...	...
64	64	32	0.153s
...	...	...	...
128	2	512	2.486s
128	4	2	1.518s
...	...	...	...
256	4	32	3.738s

Performance with different tile sizes on 3mm kernel from Polybench on AMD EPYC 7502 32-Core Processor, run in parallel on 32 threads.

Version generation



Version selection



Tile size selection

Challenge

How can we choose the tile sizes to test?

- pre-determined sizes → doesn't leverage the runtime context
- select using heuristics → doesn't leverage the real execution time and available resources

Tile size selection

Challenge

How can we choose the tile sizes to test?

- pre-determined sizes → doesn't leverage the runtime context
- select using heuristics → doesn't leverage the real execution time and available resources

Idea

Define a reduced optimization space with an analytical method, then evaluate the performance of tile sizes within this bounded space. (Shirako et al. 2012)

Optimization selection

Idea

Using a mix of analytical methods to bound the search space and runtime evaluation for true measures seems interesting.

- Bounds → reduced optimization space to explore
- Runtime execution and evaluation → results close to real execution time

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

FRANCE 2030

PROGRAMME DE RECHERCHE
NUMÉRIQUE POUR L'EXASCALE

Evaluation

Benchmarks

Polybench

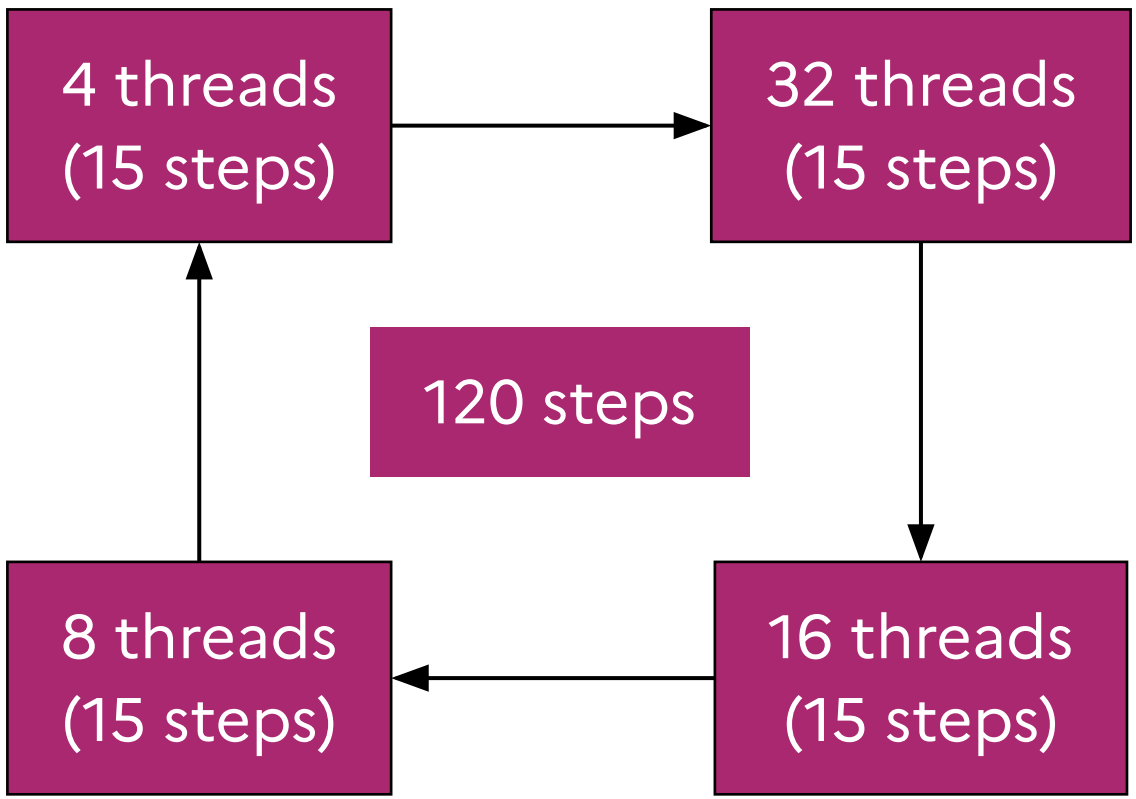
- Made of widely used compute kernels
- Each benchmark is just a single kernel called once
- Needs to be modified to introduce multiple execution contexts

Polybench

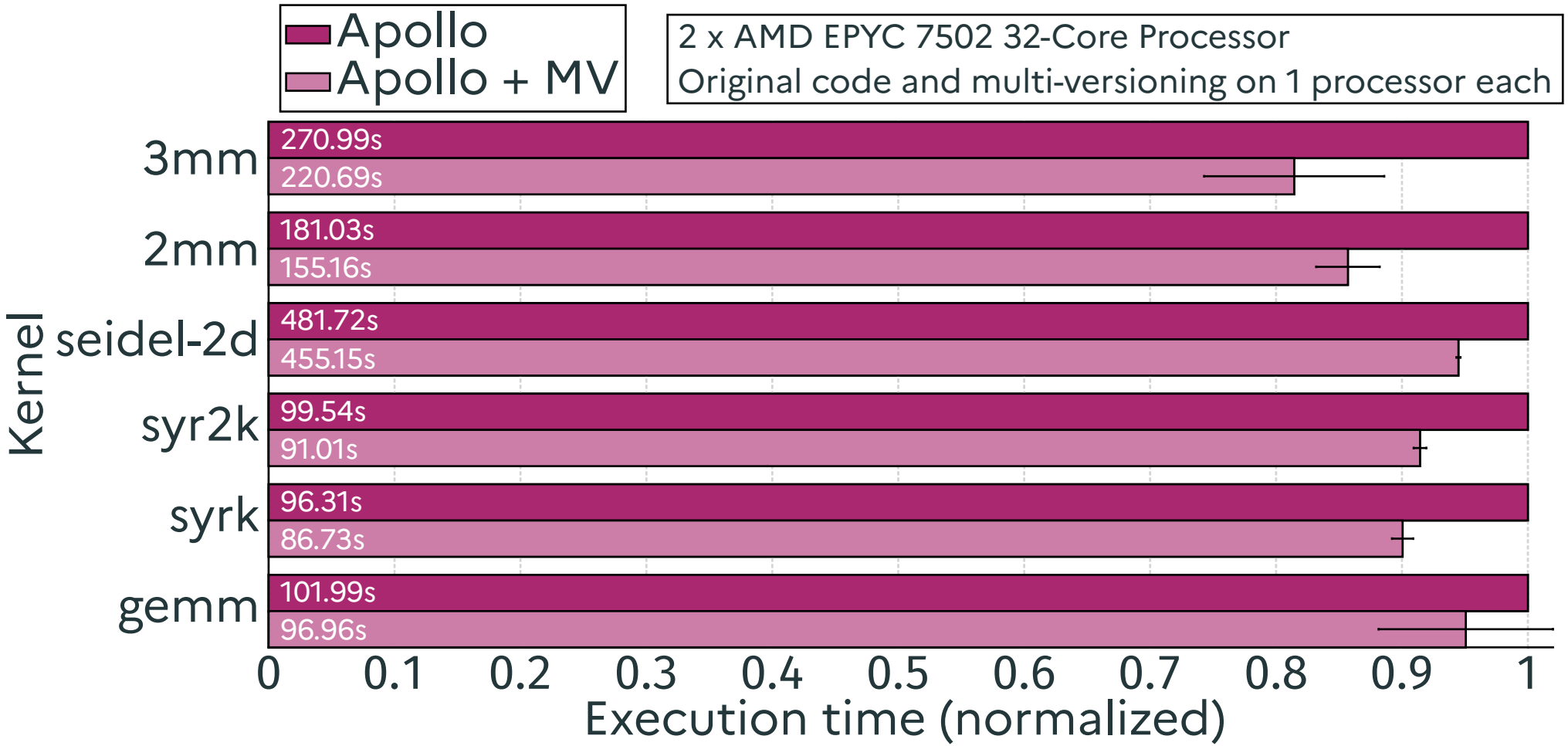
Creation of a script to introduce an evolution of the execution context in polybenchs:

- Add `#pragma apollo dcop` and `#pragma apollo kernel`
- Call the kernel in a loop, changing the size of the data or the number of available threads every `STEP` iterations.

Anatomy of an execution



Some results



Execution times for modified Polybenchs with varying number of threads (32, 16, 8, 4)

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

FRANCE 2030
PROGRAMME DE RECHERCHE
NUMÉRIQUE POUR L'EXASCALE

cea cnrs Inria

Future work

Short term

- Use algebraic tiling for better load balance
- Handle other optimization parameters than tiling
- Save best found versions with their context on disk for later use

Long term

- Multi-versioning on GPU
  - Polyhedral optimizations on GPU? With PPCG (Polyhedral Parallel Code Generation for CUDA) (Verdoolaege et al. 2013)

Thank you for your time.

Questions ?

Bibliography

Bondhugula U, Hartono A, Ramanujam J, Sadayappan P. 2008. A practical automatic polyhedral parallelizer and locality optimizer. In: Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation. Tucson AZ USA: ACM. pp. 101–113. [accessed 2025 Feb 21]. https://dl.acm.org/doi/10.1145/1375581.1375595.

Feautrier P, Lengauer C. 2011. Polyhedron Model. Padua D, editor. Encyclopedia of Parallel Computing.:1581–1592. doi:10.1007/978-0-387-09766-4_502. [accessed 2025 Feb 21]. https://doi.org/10.1007/978-0-387-09766-4_502.

Lazcano R, Madroñal D, Juarez E, Clauss P. 2020. Runtime multi-versioning and specialization inside a memoized speculative loop optimizer. In: Proceedings of the 29th International Conference on Compiler Construction. San Diego CA USA: ACM. pp. 96–107. [accessed 2025 Feb 19]. https://dl.acm.org/doi/10.1145/3377555.3377886.

Martinez Caamaño JM. 2016. Fast and Flexible Compilation Techniques for Effective Speculative Polyhedral Parallelization. [accessed 2025 Jun 12]. https://inria.hal.science/tel-01377758.

Martinez Caamaño JM, Sukumaran-Rajam A, Baloian A, Selva M, Clauss P. 2017. APOLLO: Automatic speculative POLyhedral Loop Optimizer. p. 8. [accessed 2025 Mar 13]. https://inria.hal.science/hal-01533692.

Shirako J, Sharma K, Fauzia N, Pouchet L-N, Ramanujam J, Sadayappan P, Sarkar V. 2012. Analytical Bounds for Optimal Tile Size Selection. In: O'Boyle M, editor. Compiler Construction. Berlin, Heidelberg: Springer. pp. 101–121.

Sukumaran Rajam A. 2015. Beyond the realm of the polyhedral model : combining speculative program [illegible] lyhedral compilation. [accessed 2025 Jun 12]. https://theses.hal.science/tel-01342007.

Verdoolaege S, Juega JC, Cohen A, Gómez JI, Tenllado C, Catthoor F. 2013. Polyhedral Parallel Code Generation for CUDA. ACM Transactions on Architecture and Code Optimization. 9(4):art.54:1. doi:10.1145/2400682.2400713. [accessed 2025 Jun 6]. https://inria.hal.science/hal-00786677.