RÉPUBLIQUE FRANÇAISE
Liberté Égalité Fraternité

PROGRAMME DE RECHERCHE
NUMÉRIQUE POUR L'EXASCALE

Apollo - Automatic speculative POLyhedral Loop Optimizer

Exa-Soft Annual Meeting

Erwan Auer

September 24th 2025

Inria CAMUS, University of Strasbourg - ICPS team

Outline

1. Apollo
2. Polyhedral model
3. Runtime polyhedral optimization
4. The path ahead
5. Try it for yourself

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

FRANCE 2030

PROGRAMME DE RECHERCHE
NUMÉRIQUE POUR L'EXASCALE

cea

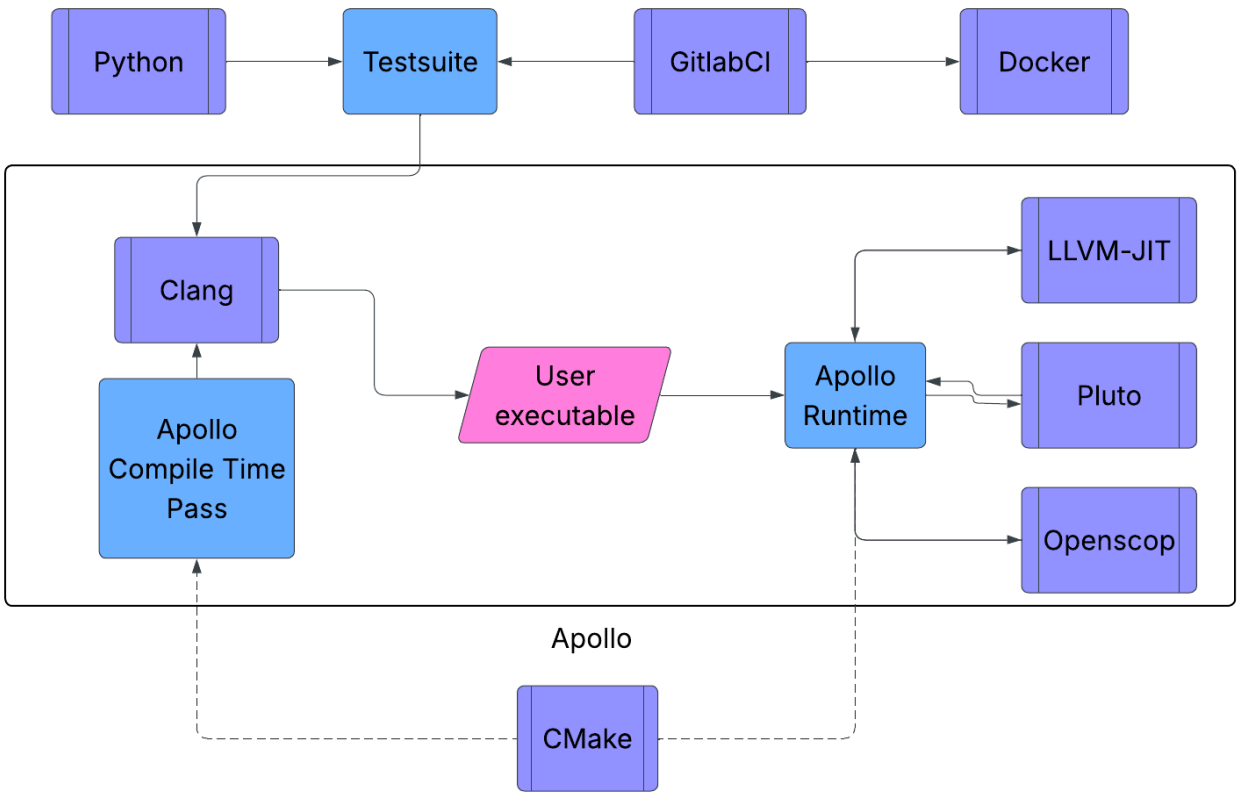
cnrs

Inria

Apollo

- Compiler built on Clang and LLVM
- Seek to apply the polyhedral model at runtime
- Aimed at scientific code

Toolchain



Apollo Toolchain

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

FRANCE 2030
PROGRAMME DE RECHERCHE
NUMÉRIQUE POUR L'EXASCALE

cea cnrs Inria

Polyhedral model

Theory

- In an ideal loop, all statements should run in parallel
- In an ideal loop, memory accesses should have a good locality
-
-
-

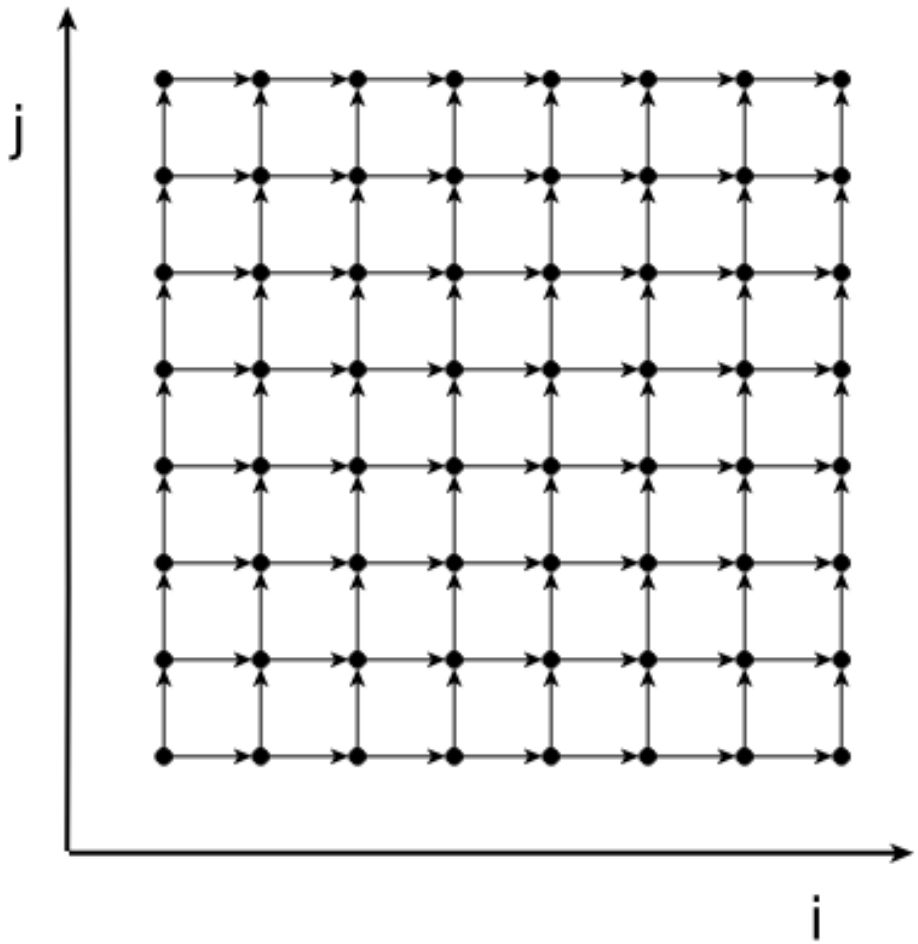
Theory

- In an ideal loop, all statements should run in parallel
- In an ideal loop, memory accesses should have a good locality
- Loop nest can be represented as an iteration domain
- This iteration domain is a discrete polyhedron, which can be transformed
- The underlying loop nest can thus be transformed and optimized

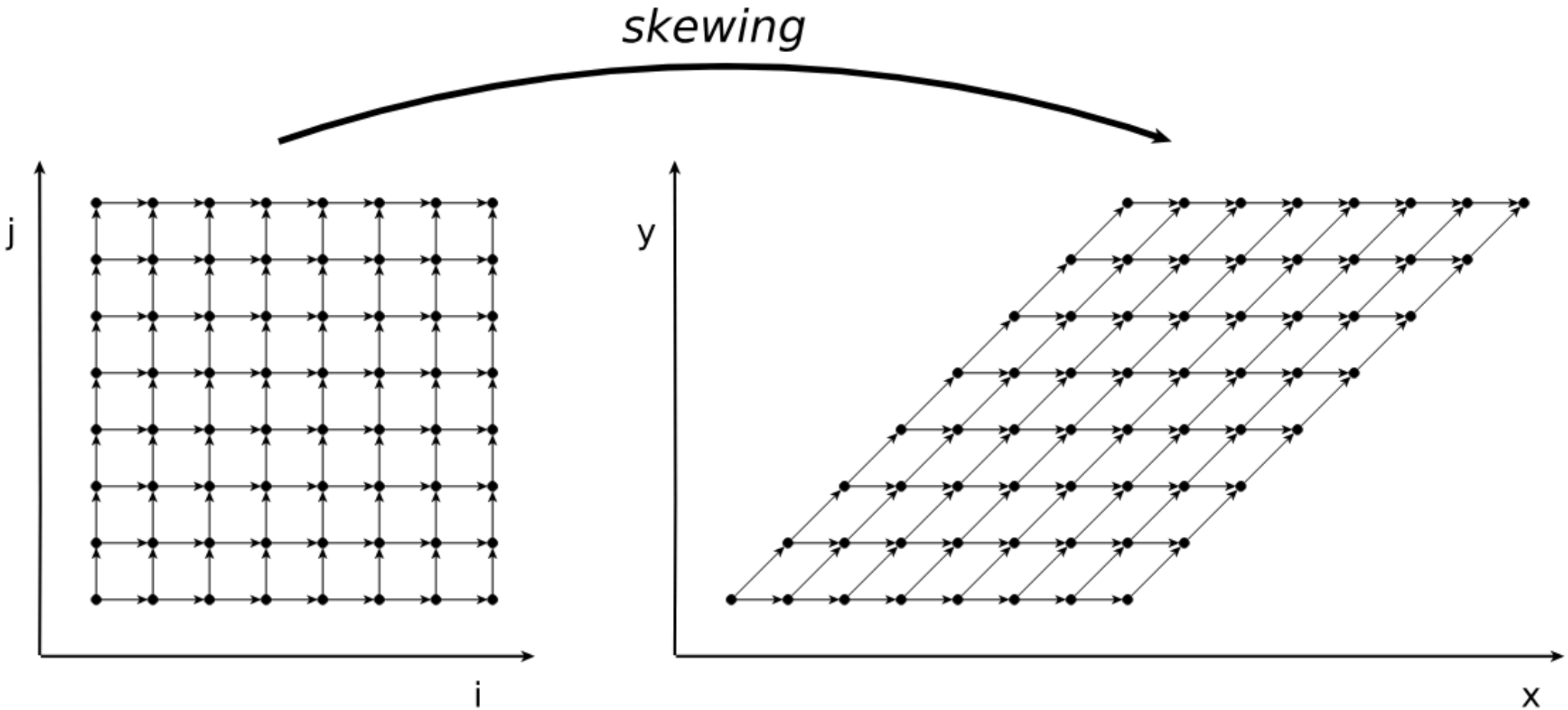
Parallelize statements

```c
1  for (i = 1; i < N; i++) // i axis
2    for (j = 1; j < M; j++) // j axis
3      a[i][j] += a[i-1][j] * a[i][j-1]; // (i,j) point
```

Problem



Applying the model



Parallelized statements

```
1  for (x = 2; x <= N+M-2; x++)                                    c
2    #pragma omp parallel for
3    for (y = max(1, x-N+1); y <= min(x-1, M-1); y++)
4      a[x-y][y] += a[x-y-1][y] * a[x-y][y-1];
```

Solution

Other polyhedral optimizations

- Loop tiling
- Loop unrolling
- Loop fusion
- ...

Applying the model

```c
#pragma scop
  for (i = 0; i < M; i++)
    for (j = 0; j < N; j++)
      for (k = 0; k < K; k++)
        C[i][j] = beta * C[i][j] + alpha * A[i][k] * B[k][j];
#pragma endscop
```

c

pluto/examples/matmul/matmul.c

Applying the model

```
1   #def S1(...) C[i][j] = beta*C[i][j]+alpha*A[i][k]*B[k][j];
2   [...]
3   #pragma omp parallel for private(lbv,ubv,t2,t3,t4,t5,t6)
4     for (t1=lbp;t1<=ubp;t1++) {
5       [...]
6       for (t5=32*t3;t5<=(min(K-1,32*t3+31))-7;t5+=8) {
7         [...]
8           S1(t1,t2,t3,(t4+7),t6,(t5+7));
9   [...]
10  for (;t4<=min(M-1,32*t1+31);t4++) {
11      [...]
12        S1(t1,t2,t3,t4,t6,t5);
13  [...]
```

c

Limitations

- Limits on the iteration domain
- Indexes must be known and affine from one another
- Memory accesses must be affine from indexes
- Memory mapping must be known at compile time
-
-
-
-

Limitations

- Limits on the iteration domain
- Indexes must be known and affine from one another
- Memory accesses must be affine from indexes
- Memory mapping must be known at compile time
- A[x] = B[y - 1] + C[x] => OK
- A[X] += B[C[y - 1]] => KO, indirect access
- (A + x) += (B + y) => KO, A alias B ?
- while(x) => KO, unknown bounds

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

FRANCE 2030
PROGRAMME DE RECHERCHE
NUMÉRIQUE POUR L'EXASCALE

cea cnrs Inria

Runtime polyhedral optimization

Reasoning

- Indexes or accesses can hide their affineness
- Pointer can be unraveled at runtime
- Memory analysis at runtime allows for speculation

Implementation

Compile time

- Preanalyse the code
- Instrument the loop for runtime analysis
- Split the loop into rearrangeable code pieces
- Create an entry point for the JIT

Implementation

Compile time

- Preanalyse the code
- Instrument the loop for runtime analysis
- Split the loop into rearrangeable code pieces
- Create an entry point for the JIT

Run time

- Profile samples at runtime to create speculation pattern
- Analyse the reorganized loop using working polyhedral tool
- Run optimized loop while ensuring speculation error correction

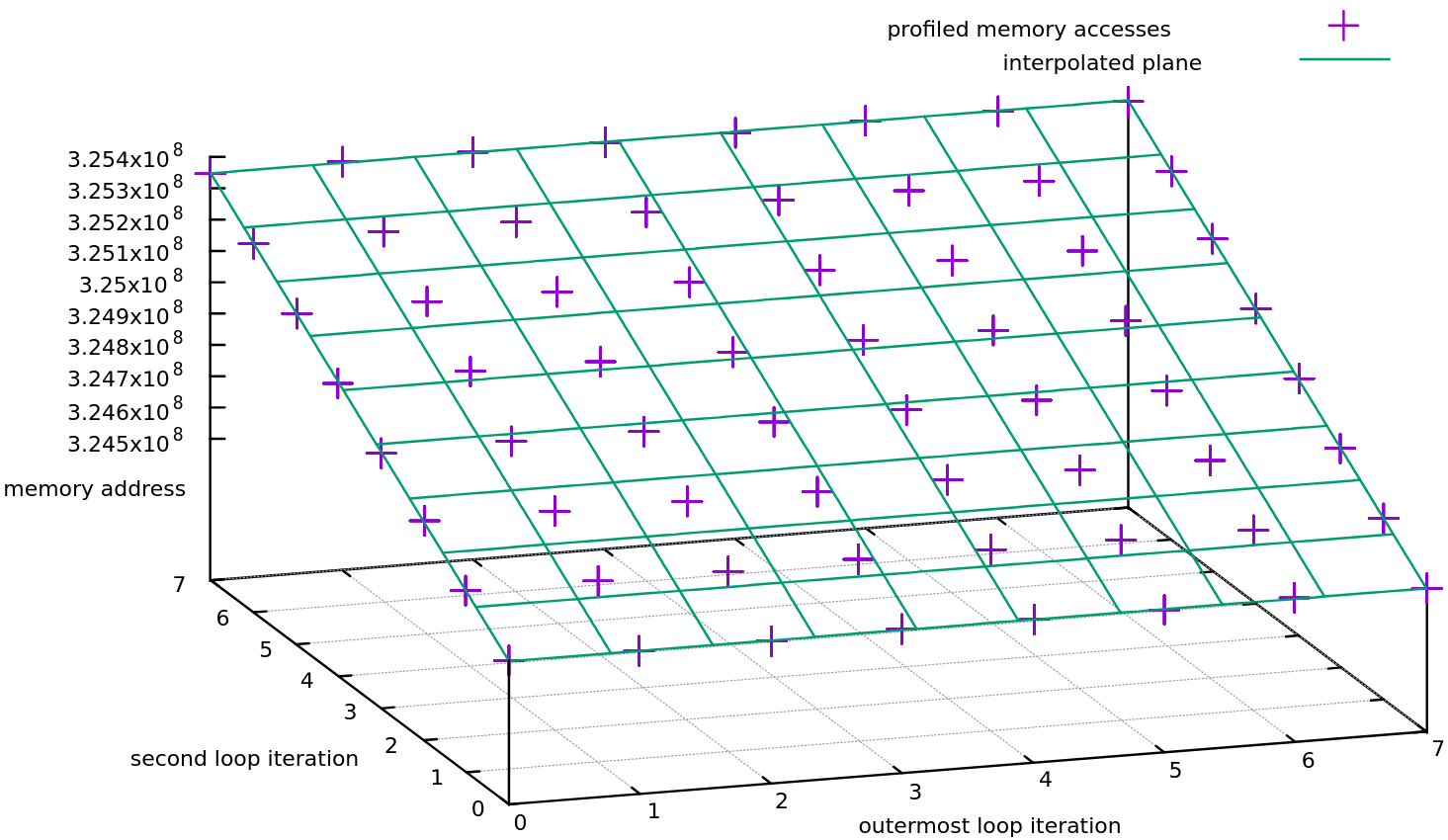
Sparse matrix multiplication

```c
for(row = 1; row <= left->Size; row++) {
  pElem = left->FirstInRow[row];
  while(pElem) {
    for(col = 1; col <= cols; col++) {
      result[row][col] += pElem->Real * right[pElem->Col][col];
    }
    pElem = pElem->NextInRow;
  }
}
```

c

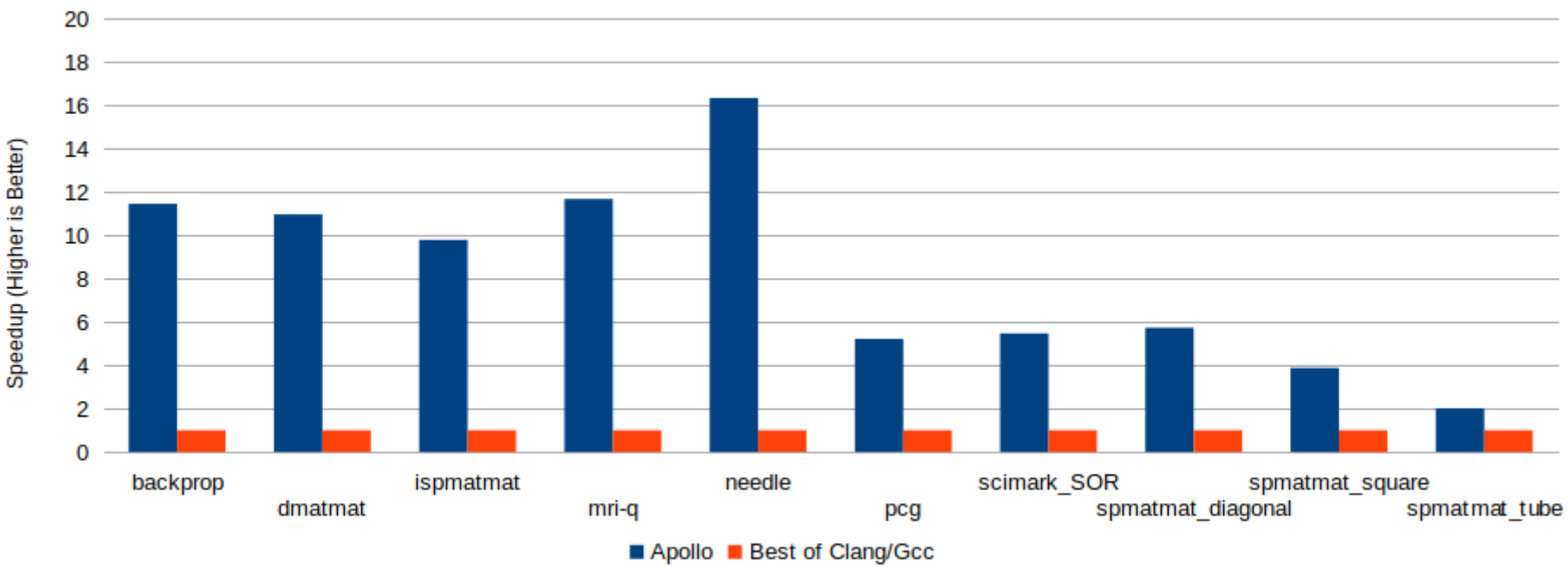
Example

Sparse matrix multiplication



Runtime analysis

Some numbers on polybenchs



AMD Opteron 6172, 2X12-core

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

FRANCE 2030
PROGRAMME DE RECHERCHE
NUMÉRIQUE POUR L'EXASCALE

cea cnrs Inria

The path ahead

Multiversioning

- Optimization parameters can greatly ehance performance
- Tile size, loop unrolling factor, ...
- Finding the right parameters is a tricky part
- Parameters trials combined with multiversioning could be the key

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

FRANCE 2030
PROGRAMME DE RECHERCHE
NUMÉRIQUE POUR L'EXASCALE

cea cnrs Inria

Try it for yourself

With Docker

```sh
$ docker run --pull=always -ti registry.gitlab.inria.fr/pclauss/apollo/debian10:latest sh
$ apolloc -O3 -o matmul /root/apollo-build/src/examples/matmul.c
$ ./matmul
```

With Spack

```sh
$ spack config add modules:prefix_inspections:lib64:[LD_LIBRARY_PATH]
$ spack config add modules:prefix_inspections:lib:[LD_LIBRARY_PATH]
$ spack env activate --create myenv
$ spack add apollo
$ spack install apollo@develop
$ apolloc -O3 -o matmul matmul.c
$ ./matmul
```

All links on our website

- https://webpages.gitlabpages.inria.fr/apollo/download

Thank you for your time.

Questions ?