Tiling in the VMAD framework

Author
Martinez Caamaño, Juan Manuel
jmartinezcaamao@gmail.com

INRIA - Camus team
Université de Strasbourg

What is **VMAD** ?

Loop instrumentation + Runtime optimization + Polytope model

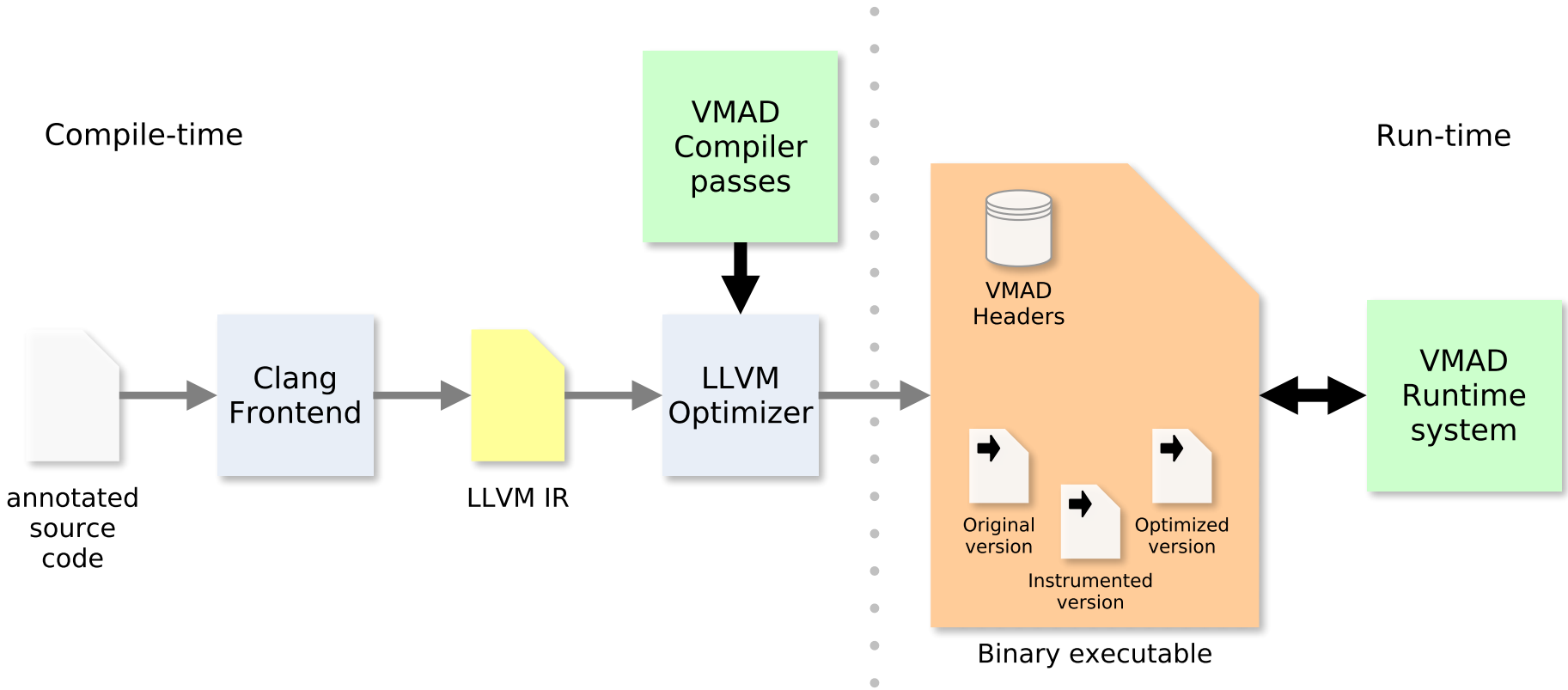
What is **VMAD** ?

Loop instrumentation + Runtime optimization + Polytope model

Predecessor of the **APOLLO** framework.

VMAD framework

Framework overview



$$
\begin{aligned}
&\mathit{example}(\mathbf{float} * *A, \mathbf{float} * *B)\{ \\
&\quad \mathbf{for}(i = 1; i \leq N; ++i) \\
&\qquad \mathbf{for}(j = 1; j \leq N; ++j) \\
&\qquad\quad A[i][j] = B[i-1][j] + B[i][j-1] \\
&\}
\end{aligned}
$$

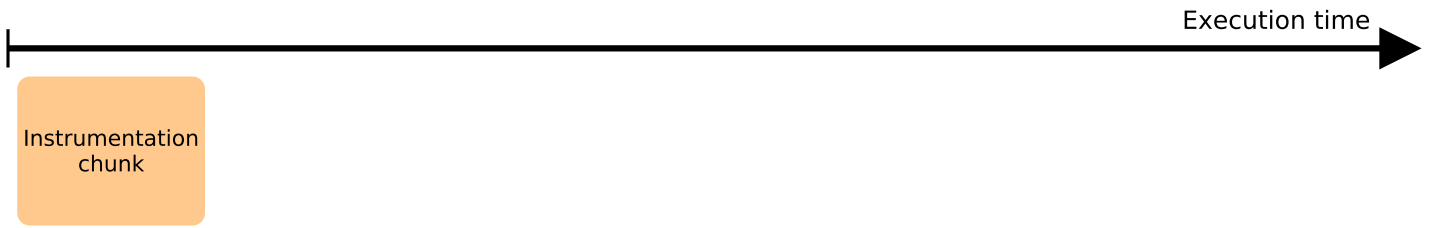
Can be invoked as $\mathit{example}(M_1, M_2)$ or $\mathit{example}(M_1, M_1)$.

Loop execution by **chunks**.

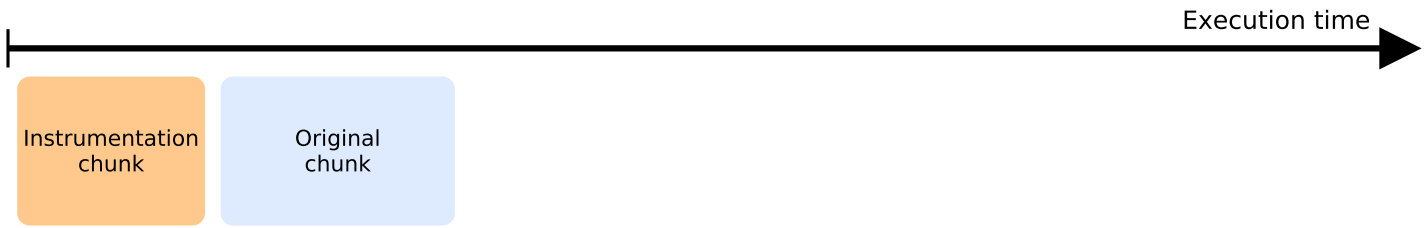
Loop execution by **chunks**.

A chunk is a set of contiguous iterations of the outermost loop.

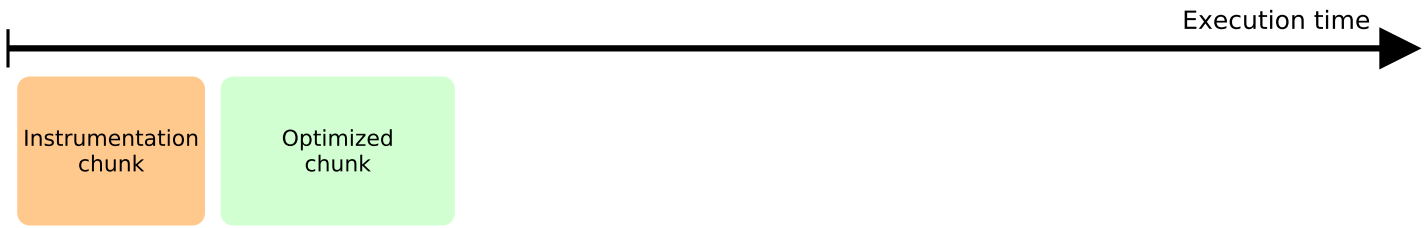
Loop execution by **chunks**.
Collect information by instrumenting some iterations.



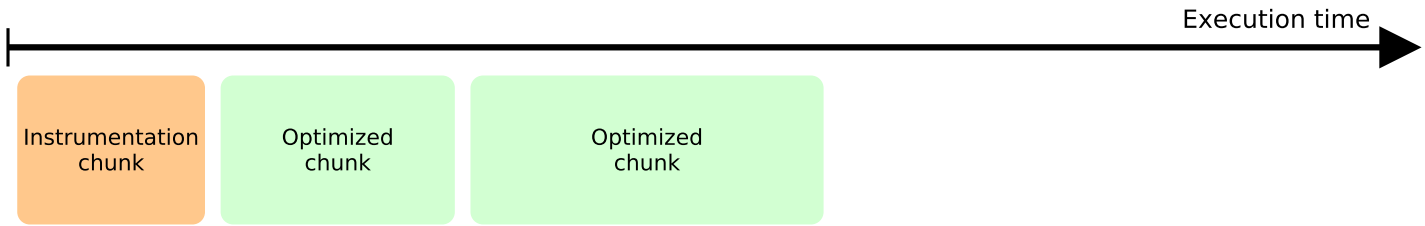
Loop execution by **chunks**.
Keep the original behavior.



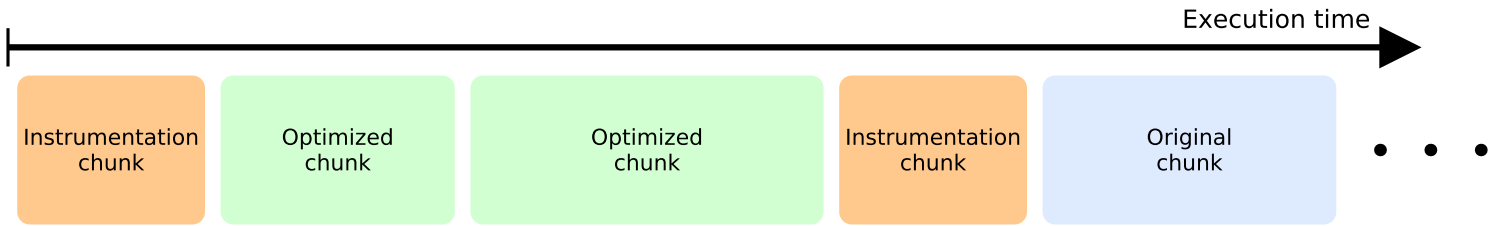
Loop execution by **chunks**.
Apply polyhedral optimizations.



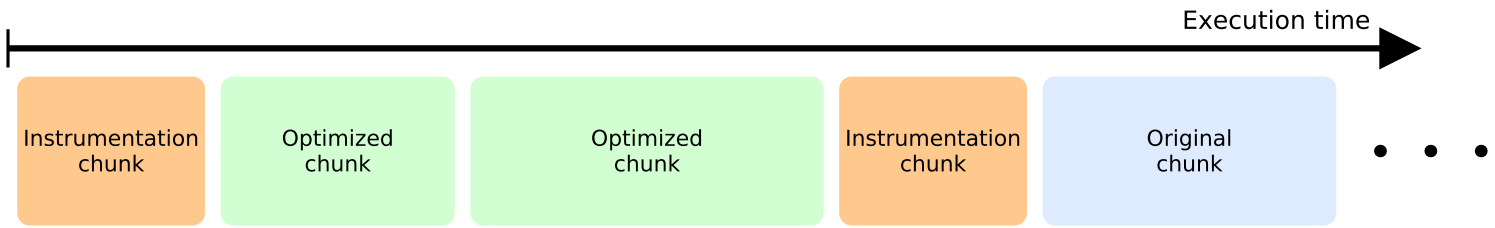
Loop execution by **chunks**.



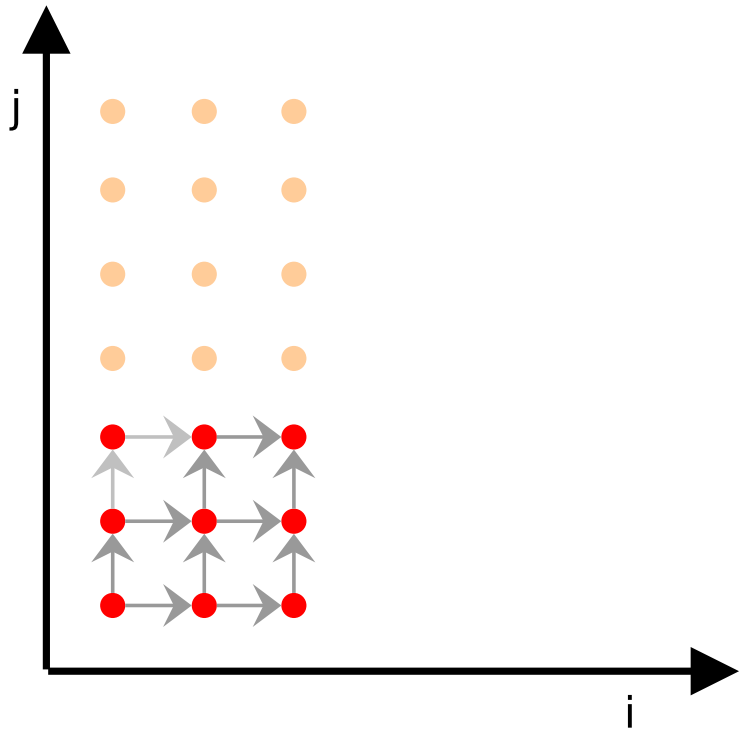
Loop execution by **chunks**.



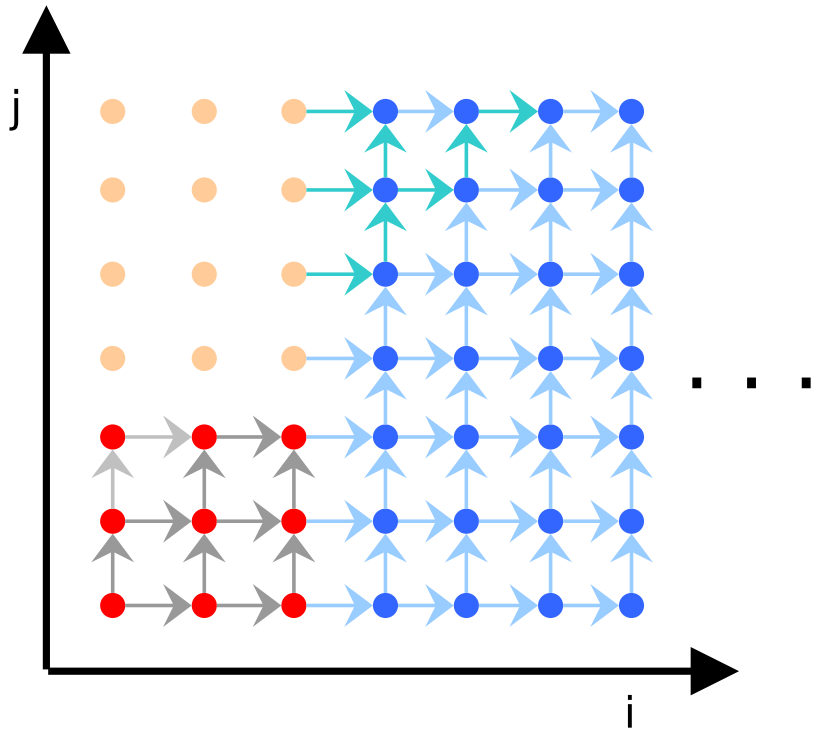
Loop execution by **chunks**.
Adapt the execution depending on the loop behavior.



Predict the behavior from a **few** instrumented iterations.



Predict the behavior from a **few** instrumented iterations.



Predictions may go wrong.
We must **detect** mispredictions.

Predictions may go wrong.
We must **detect** mispredictions.

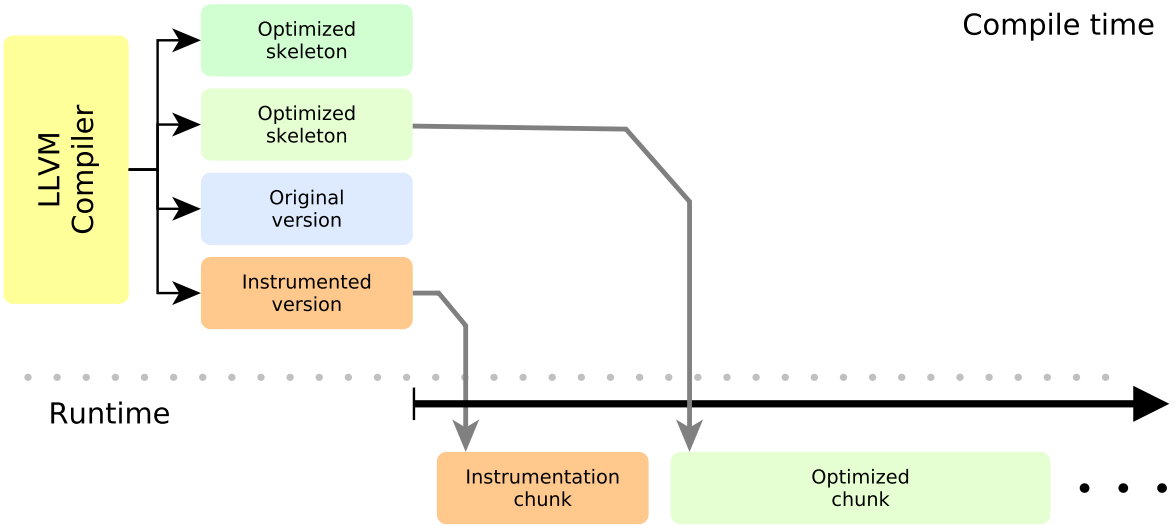
$$\text{pred} = a_1 * i + a_2 * j + a_3$$
if &A[i][j] $\neq$ pred **then** SignalMisspredict ion()
store float val, float* &A[i][j]

Inserted to verify predicted scalars, memory accesses and loop bounds.

How can we **adjust the loop** to execute efficiently at different stages ?

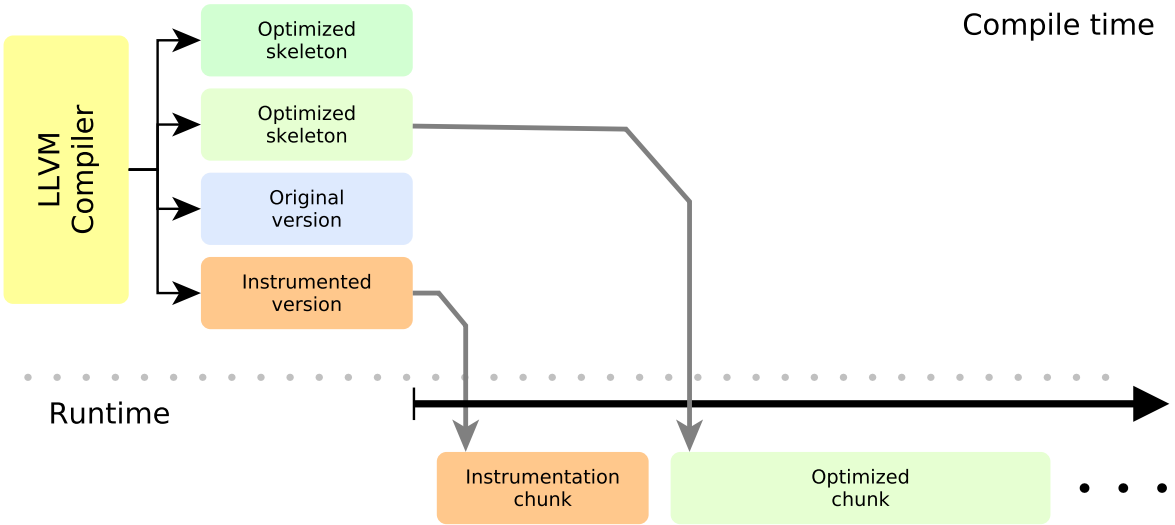
How can we **adjust the loop** to execute efficiently at different stages ?

- We generate several specialized versions from the original nest.



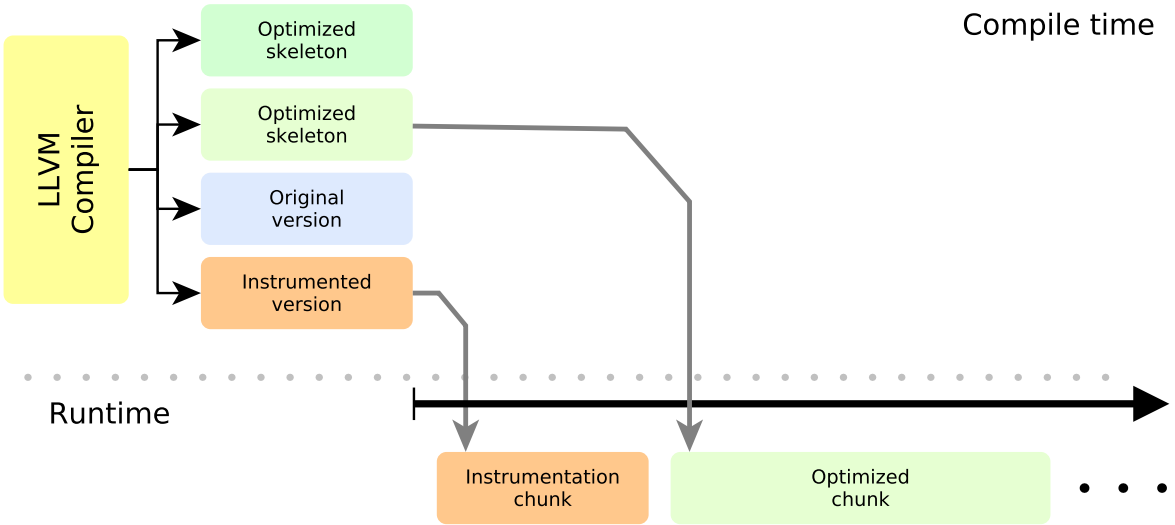
How can we **adjust the loop** to execute efficiently at different stages ?

- We generate several specialized versions from the original nest.
  - One with *instrumentation code* embedded.



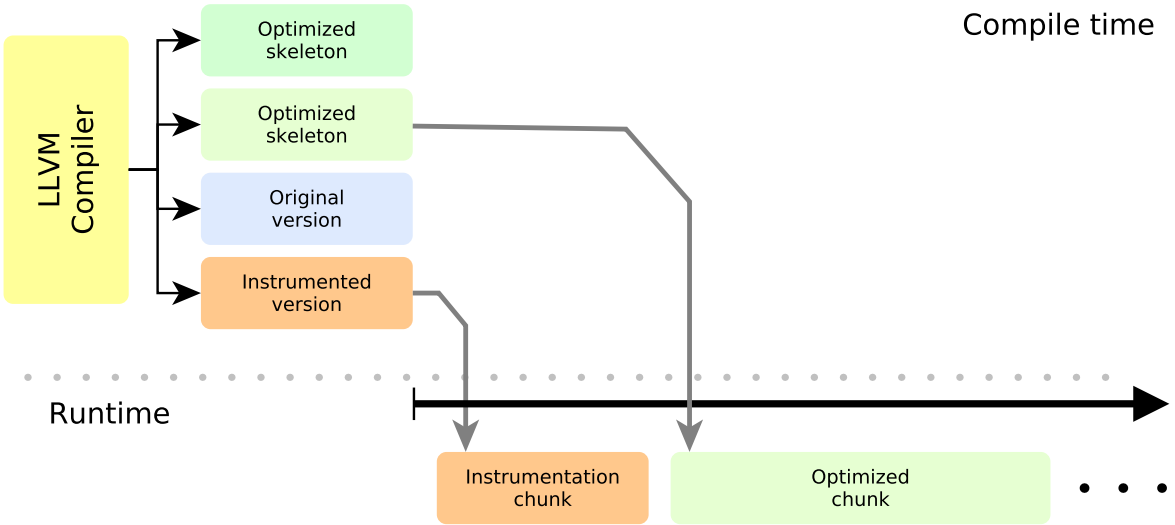
How can we **adjust the loop** to execute efficiently at different stages ?

- We generate several specialized versions from the original nest.
  - One with *instrumentation code* embedded.
  - One matching the original nest behavior.



How can we **adjust the loop** to execute efficiently at different stages ?

- We generate several specialized versions from the original nest.
  - One with *instrumentation code* embedded.
  - One matching the original nest behavior.
  - Several **code skeletons**.



A **code skeleton** is a parametrized nest to match a kind of transformations.

$$
\begin{aligned}
&\textbf{for}(x = Lower_x(); x < Upper_x(); ++x)\{ \\
&\quad \textbf{forall}(y = Lower_y(x); y < Upper_y(x); ++y)\{ \\
&\qquad \begin{pmatrix} i \\ j \end{pmatrix} = T^{-1} \begin{pmatrix} x \\ y \end{pmatrix} \\
&\qquad \ldots //Body \\
&\quad \} \\
&\}
\end{aligned}
$$

A **code skeleton** is a parametrized nest to match a kind of transformations.

for$(x = chunk\_no + 1; x \leq chunk\_no + opt\_chunk\_size + N - 1; ++x)\{$

forall$(y = 1; y < min(x, N); ++y)\{$

$$\begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} 1 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

$\ldots //Body$

$\}$

$\}$

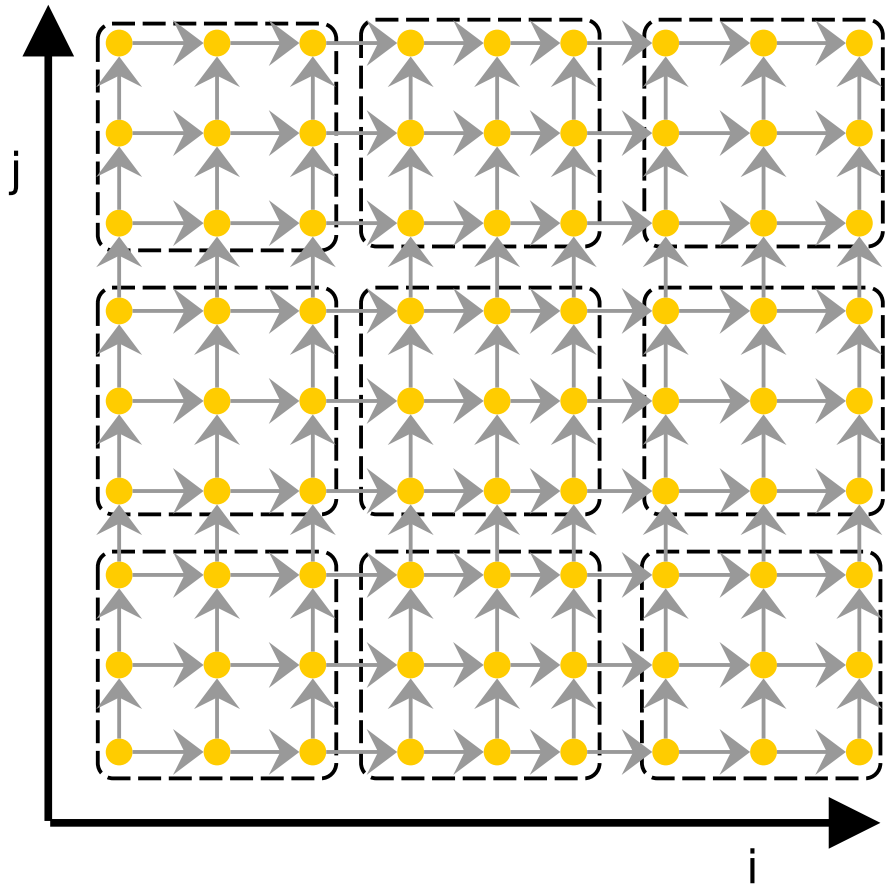
Consequences of this approach:

- The prediction verification imposes an **overhead**.
- The number of transformations is **limited** by the available skeletons.

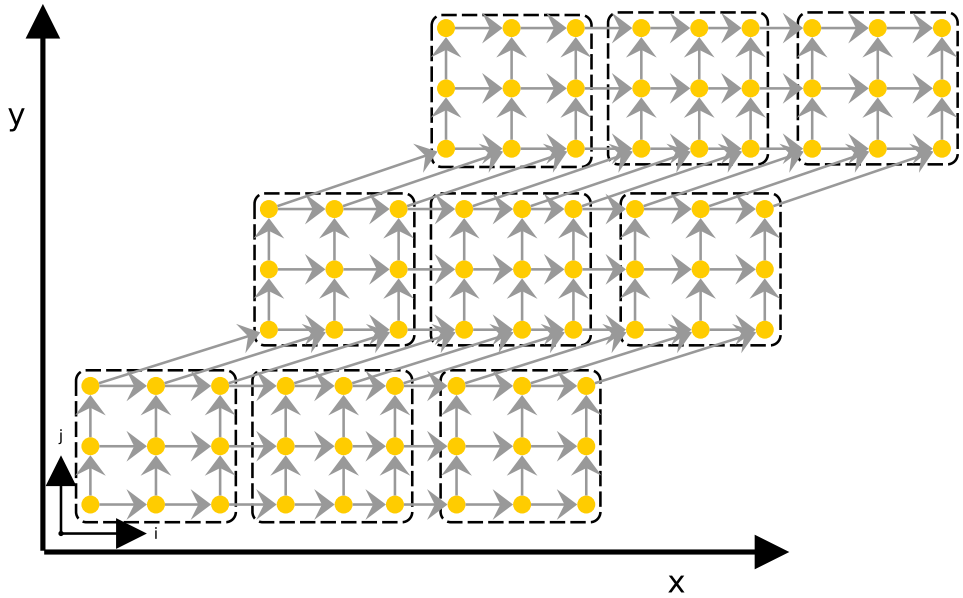
Contributions

Contribution 1

New code skeleton enabling the *tiling* transformation.



New code skeleton enabling the *tiling* transformation.



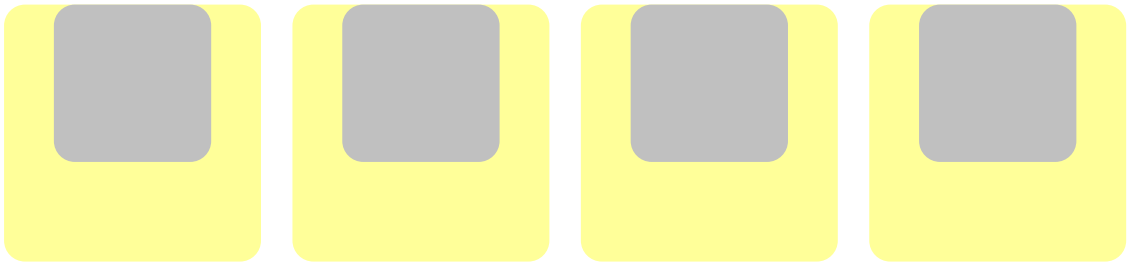
Contribution 1

Tiling transformation

- **Alters** the nest structure, it needs $2\times$ the number of loops.
- Extension of the dependence analysis and transformation selection.
- Development of a mechanism for adjusting the tile sizes.

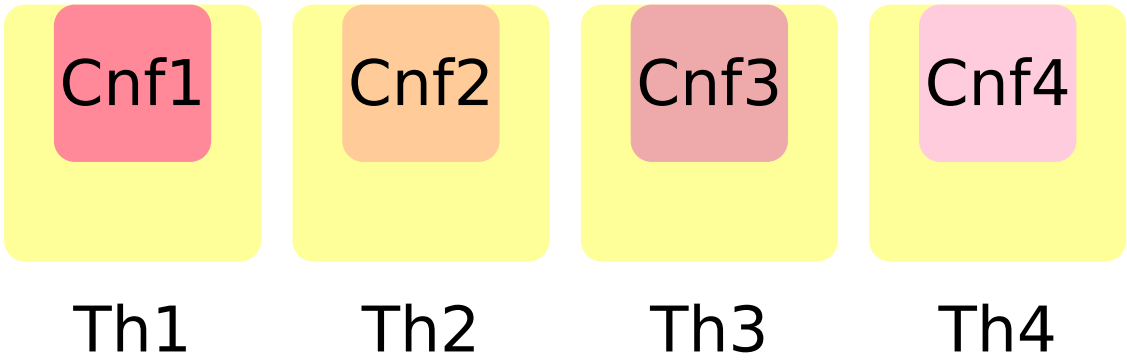
Tile size adjustment algorithm:

- Before executing the first chunk, assign random values for the tile sizes.



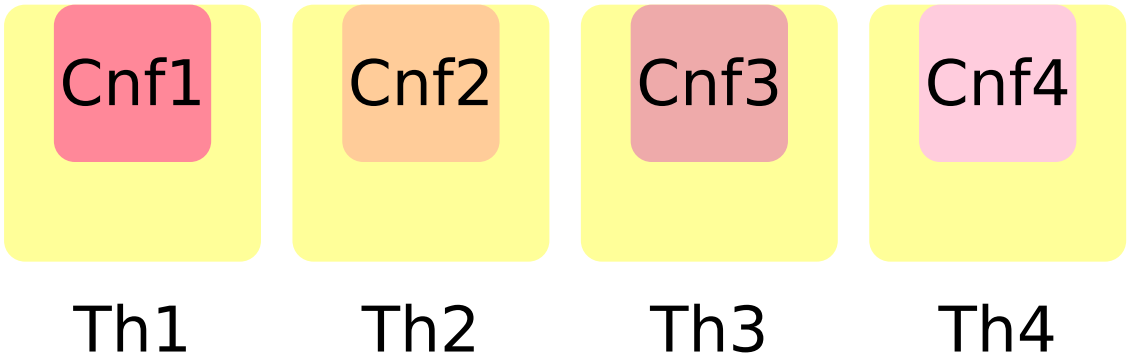
Tile size adjustment algorithm:

- Before executing the first chunk, assign random values for the tile sizes.



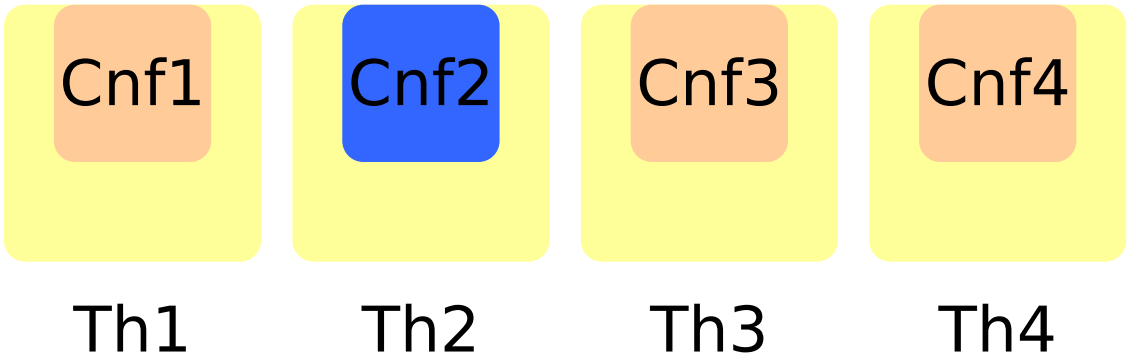
Tile size adjustment algorithm:

- Launch the chunk and **collect information** about the execution.
  - Execution time and number of tiles executed.
- After the chunk execution finishes:
  - Obtain a *score* for each tile size configuration and choose the winner.



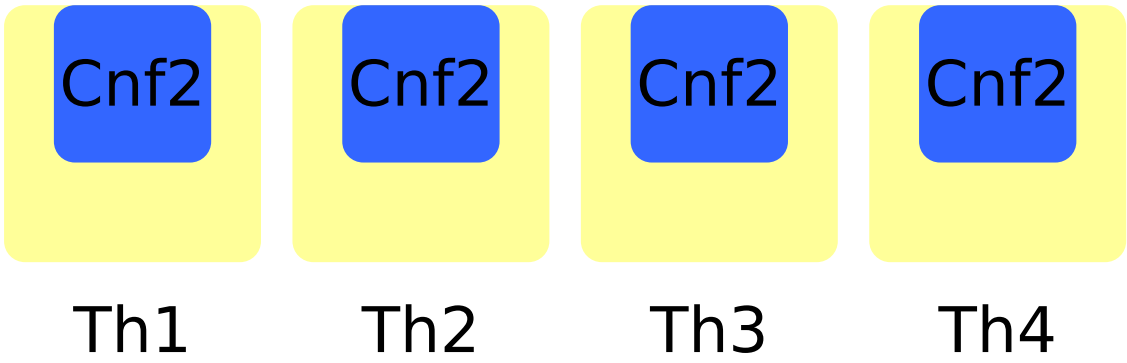
Tile size adjustment algorithm:

- Launch the chunk and **collect information** about the execution.
  - Execution time and number of tiles executed.
- After the chunk execution finishes:
  - Obtain a *score* for each tile size configuration and choose the winner.



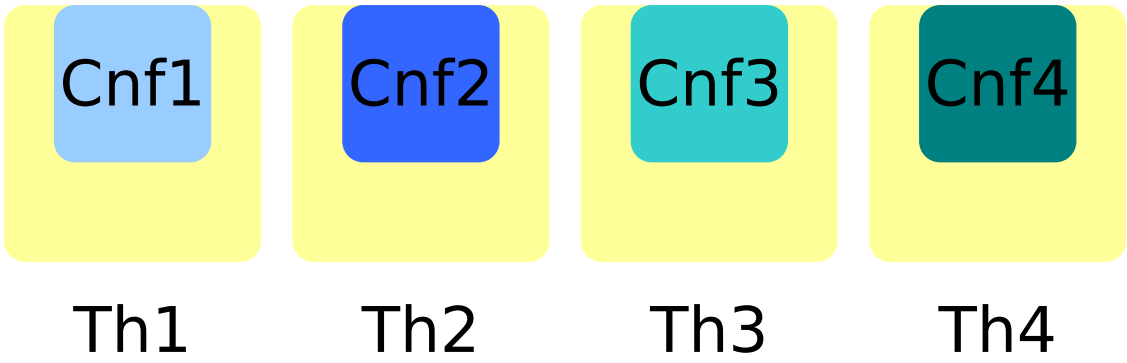
Tile size adjustment algorithm:

- Between the execution of chunks:
  - Choose the best configuration based on the *score*, replicate it on each thread.

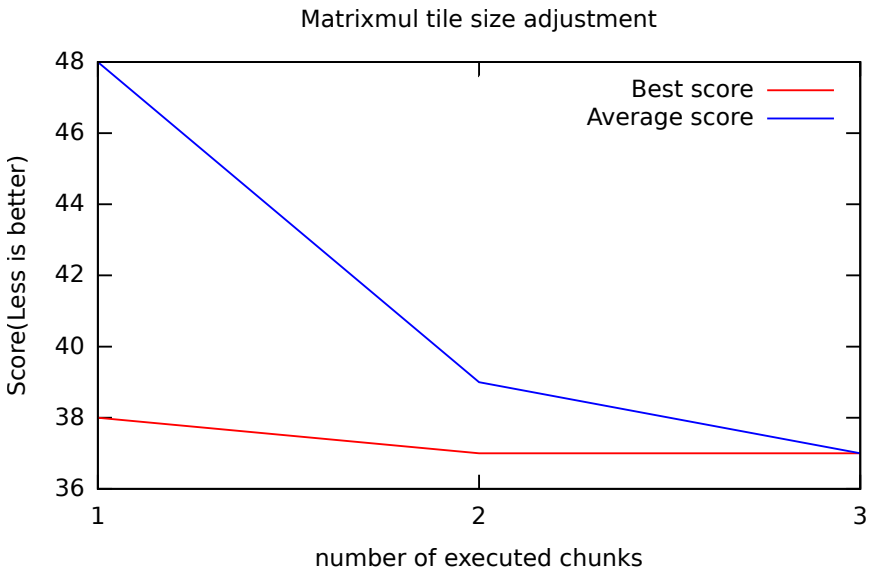


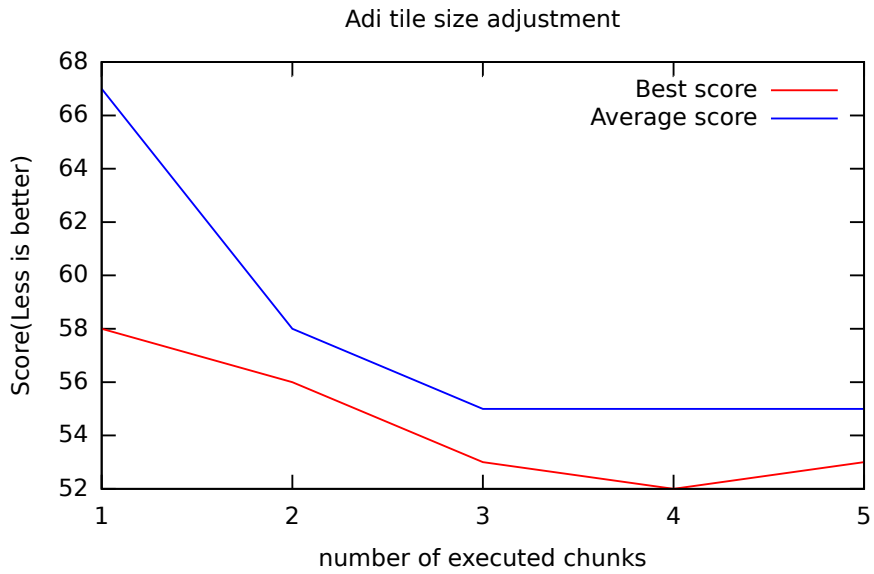
Tile size adjustment algorithm:

- Between the execution of chunks:
  - Slightly adjust the tile size configurations.



Tile size adjustment for one loop execution.

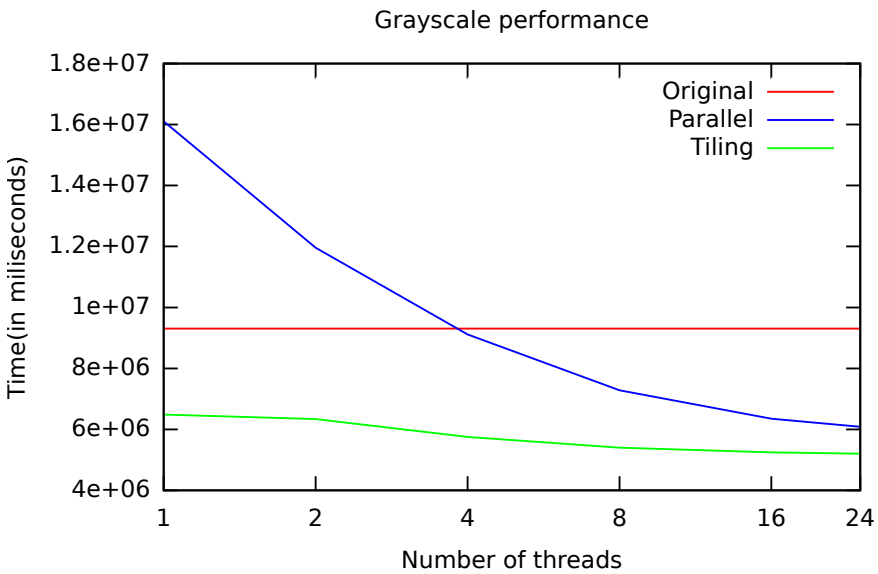


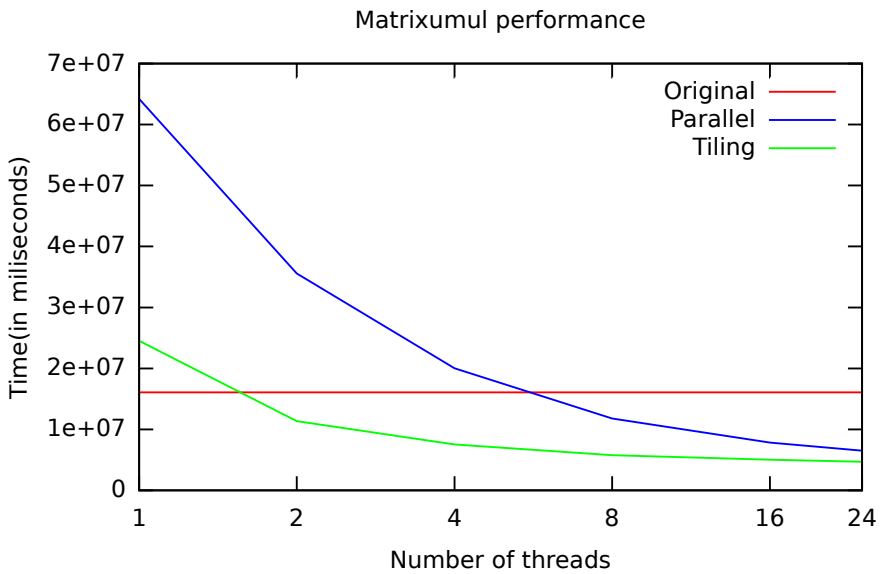


Contribution 1

Tiling transformation

- Improves **data locality**.
- Reduced amount of verification code.





Contribution 2

Contribution 2

Speculative induction variable recognition

Verification code checks if the memory accesses satisfy the predicted linear functions.

$$\text{pred} = a_1 * i + a_2 * j + a_3$$
if &A[i][j] $\neq$ pred **then** SignalMisspredication()
store float val, float* &A[i][j]

This verification introduces time **overhead**.

Contribution 2

Speculative induction variable recognition

Can we **avoid** some of this verification code ?

$$
\begin{aligned}
&\textbf{for}(\ldots)\{\\
&\quad *(ptr) = \ldots\\
&\quad *(ptr+1) = \ldots\\
&\quad *(ptr+2) = \ldots\\
&\quad *(ptr+3) = \ldots\\
&\}
\end{aligned}
$$

Contribution 2

Speculative induction variable recognition

Can we **avoid** some of this verification code ?

We introduced an analysis to identify *memory accesses* which don't need verification.

- Avoid verification if a memory address is a **linear transformation** from a linear value (predicted scalar or another memory address).
- Catches common cases such as loop unrolls and several accesses to a structure.

Conclusions

Conclusions

VMAD purposed a promising approach to automatic loop parallelization.
In this talk I presented a new extension to this framework:

- A new code skeleton, enabling the tiling transformation, that improves data locality and has reduced verification code.
- An analysis to identify memory accesses where verification code is not necessary.

Perspectives

APOLLO is an evolution of this framework.

¿?