HYPERF: End-to-End Autotuning Framework for High-Performance Computing

HPDC 2025

Juseong Park[*,2], Yongwon Shin[*,2], Junghyun Lee[1], Junseo Lee[1], Juyeon Kim[3], Oh-Kyoung Kwon[4], Hyojin Sung[1]

[1]Seoul National University (SNU)

[2]Pohang University of Science and Technology (POSTECH)

[3]Ewha Womans University

[4]Korea Institute of Science and Technology Information (KISTI)

Republic of Korea

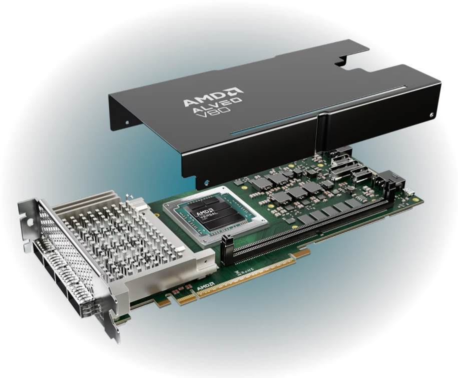
[*]Equal contribution

Modern High-Performance Computing Systems

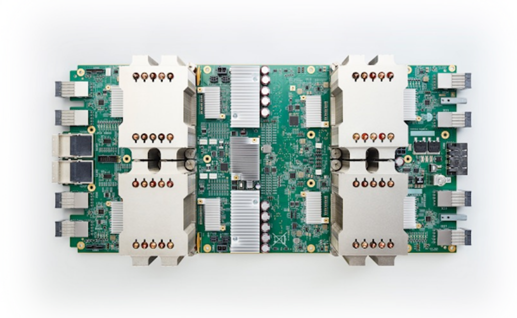
- Modern HPC platforms utilize heterogeneous and parallel hardware for high throughput
  - Growing demands from diverse application domains
    - Scientific simulations, big data processing, ML/DL workloads
  - Increasing computational requirements driven by large-scale data

CPU[1]

GPU[2]



FPGA[3]

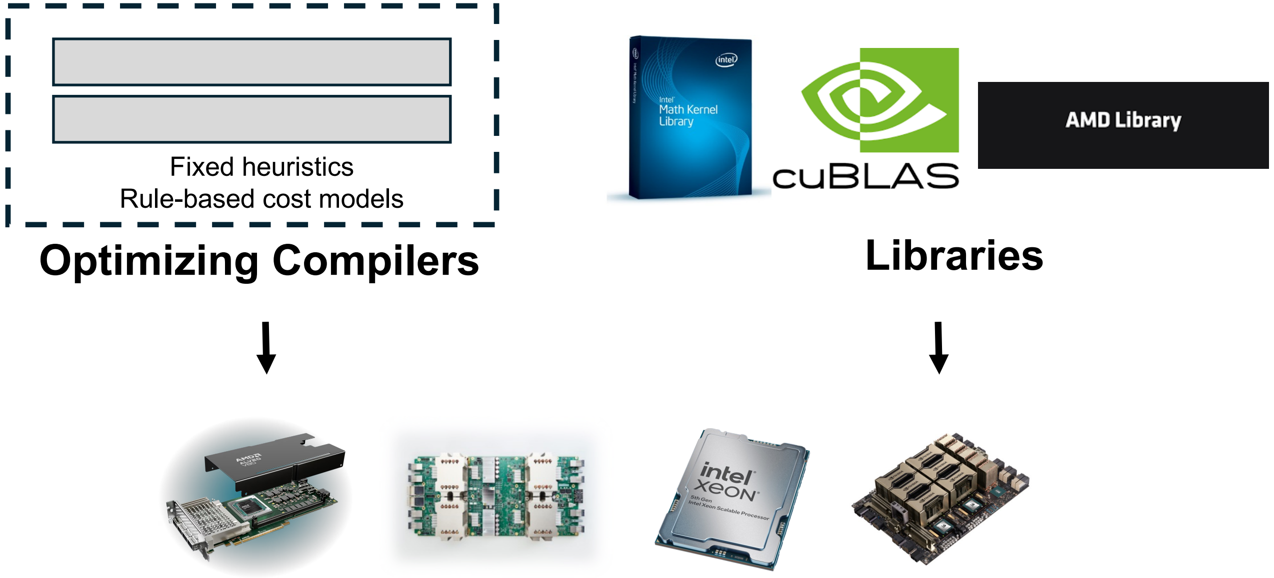


NPU[4]

[1] https://no.mouser.com/new/intel/intel-5th-gen-xeon-processors
[2] https://developer.nvidia.com/blog/introducing-hgx-a100-most-powerful-accelerated-server-platform-for-ai-hpc/
[3] https://www.amd.com/ko/products/accelerators/alveo/v80.html
[4] https://korea.googleblog.com/2017/05/google-cloud-offer-tpus-machine-learning.html

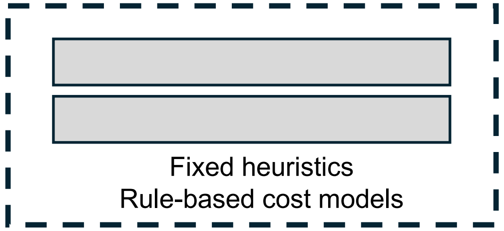
Software Stacks for HPC

- Optimizing the software stack is crucial for harnessing HW parallelism
  - Parallel programming models, compiler/runtime, libraries



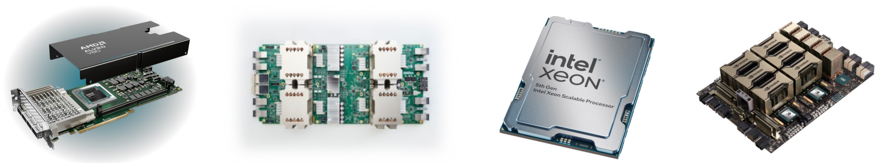
Software Stacks for HPC

- Optimizing the software stack is crucial for harnessing HW parallelism
  - Parallel programming models, compiler/runtime, libraries
- Complex and diverse hardware poses optimization challenges
  - **Repeated engineering effort** is required for hand-tuned libraries
  - Fixed heuristic-based optimizations often lead to **suboptimal performance**



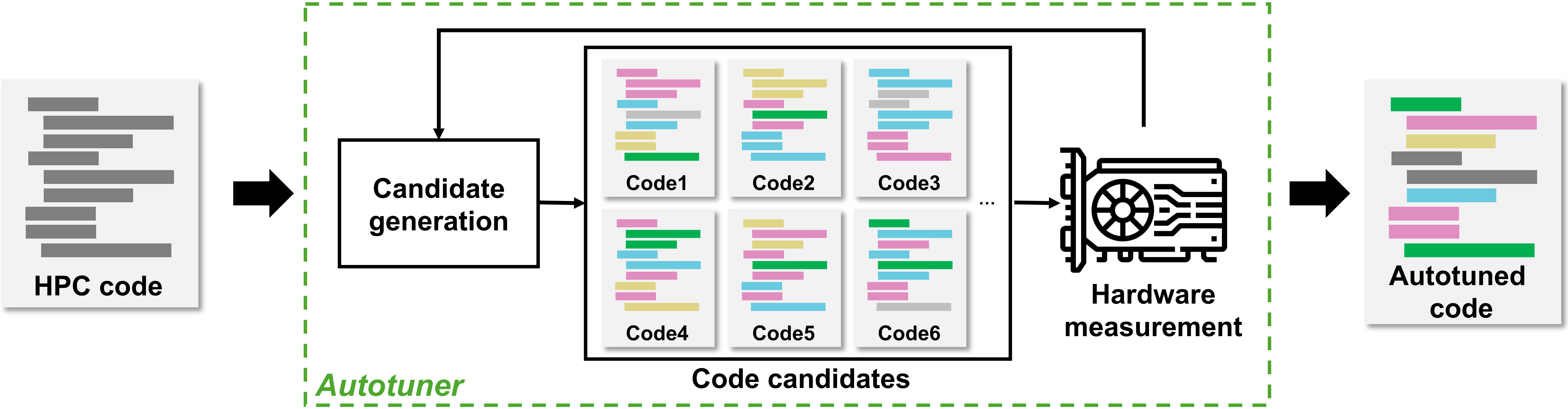
Optimizing Compilers

Libraries



Autotuning Approaches

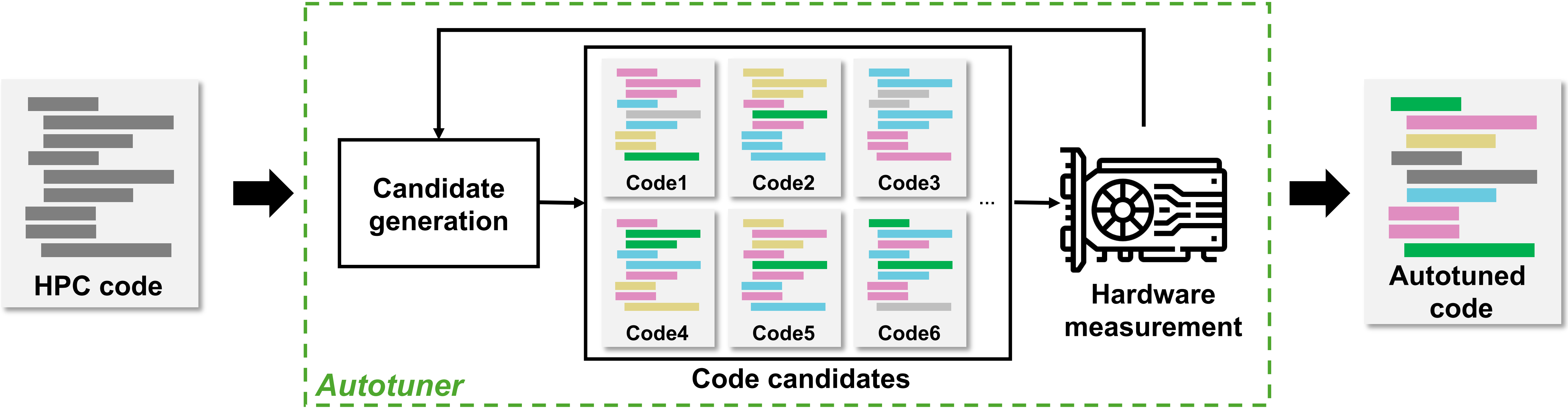
- Search-based, iterative optimization
  - Generate candidates for a given code
  - Search for high-performing versions through hardware evaluations



Autotuning Approaches

- Search-based, iterative optimization
  - Generate candidates for a given code
  - Search for high-performing versions through hardware evaluations

→ Adaptive and flexible, not requiring re-implementation



Autotuning for HPC

- **Pragma-based** approaches[1-3] annotate low-level code to guide optimization decisions

```
#pragma problem
for (i = 0; i < M; i++)
 for (j = 0; j < N; j++)
  y[i] += A[i][j] * x[j];
```

GEMV
low-level code

[1] Wu, Xingfu, et al. "Autotuning polybench benchmarks with llvm clang/polly loop optimization pragmas using bayesian optimization."
[2] Ansel, Jason, et al. "Opentuner: An extensible framework for program autotuning."
[3] Wu, Xingfu, et al. "ytopt: Autotuning scientific applications for energy efficiency at large scales."

Autotuning for HPC

- **Pragma-based** approaches[1-3] annotate low-level code to guide optimization decisions
- Autotuner uses user annotations to apply different compiler and optimization parameters and generate candidates

Configuring directives without modifying low-level code

```
#pragma problem
for (i = 0; i < M; i++)
 for (j = 0; j < N; j++)
  y[i] += A[i][j] * x[j];
```

GEMV low-level code

```
#pragma omp
tile parallel for
for (i = 0; i < M; i++)
 for (j = 0; j < N; j++)
  y[i] += A[i][j] * x[j];
```

[1] Wu, Xingfu, et al. "Autotuning polybench benchmarks with llvm clang/polly loop optimization pragmas using bayesian optimization."
[2] Ansel, Jason, et al. "Opentuner: An extensible framework for program autotuning."
[3] Wu, Xingfu, et al. "ytopt: Autotuning scientific applications for energy efficiency at large scales."

Autotuning for HPC

- **Pragma-based** approaches[1-3] annotate low-level code to guide optimization decisions
- Autotuner uses user annotations to apply different compiler and optimization parameters and generate candidates

Configuring directives without modifying low-level code

```
#pragma problem
for (i = 0; i < M; i++)
 for (j = 0; j < N; j++)
  y[i] += A[i][j] * x[j];
```

GEMV low-level code

```
#pragma omp
tile parallel for
for (i = 0; i < M; i++)
 for (j = 0; j < N; j++)
  y[i] += A[i][j] * x[j];
```

```
#pragma omp simd
...
```

```
#pragma omp simd tile
...
```

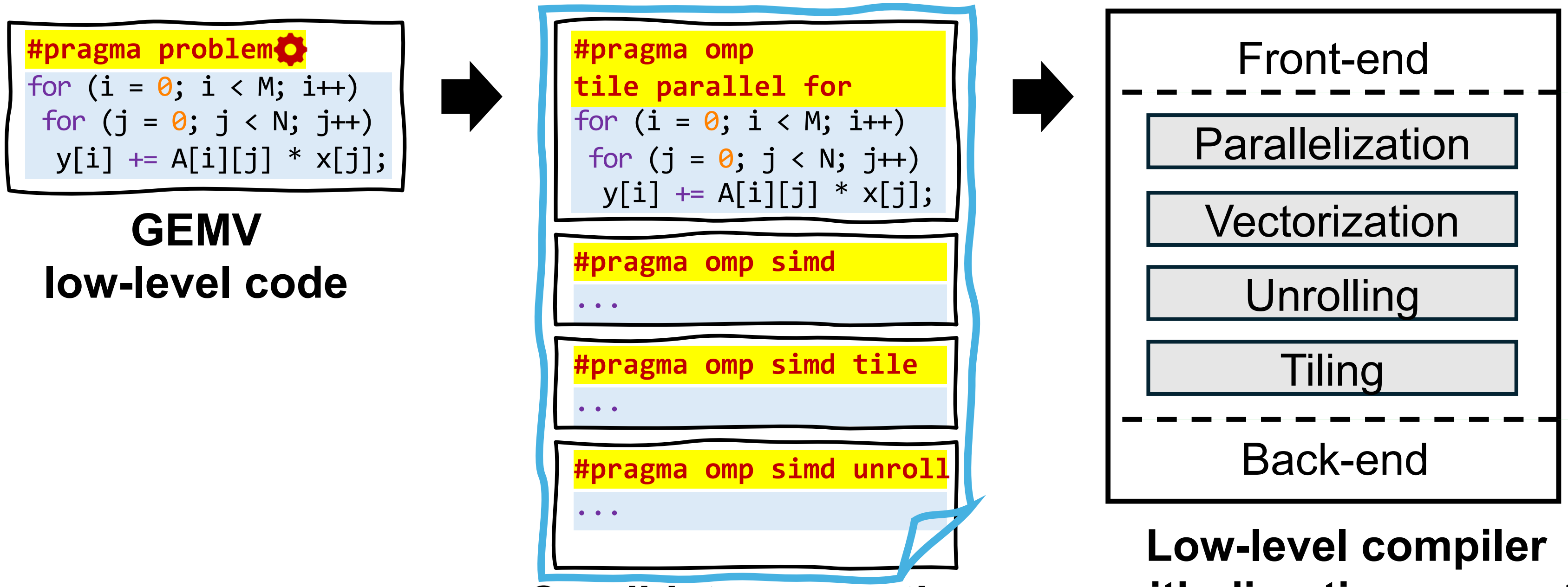
```
#pragma omp simd unroll
...
```

Candidate generation

[1] Wu, Xingfu, et al. "Autotuning polybench benchmarks with llvm clang/polly loop optimization pragmas using bayesian optimization."
[2] Ansel, Jason, et al. "Opentuner: An extensible framework for program autotuning."
[3] Wu, Xingfu, et al. "ytopt: Autotuning scientific applications for energy efficiency at large scales."

Autotuning for HPC

- Pragma-based approaches[1-3] annotate low-level code to guide optimization decisions
- Autotuner uses user annotations to apply different compiler and optimization parameters and generate candidates
  - How codes are transformed is determined by low-level compiler implementations

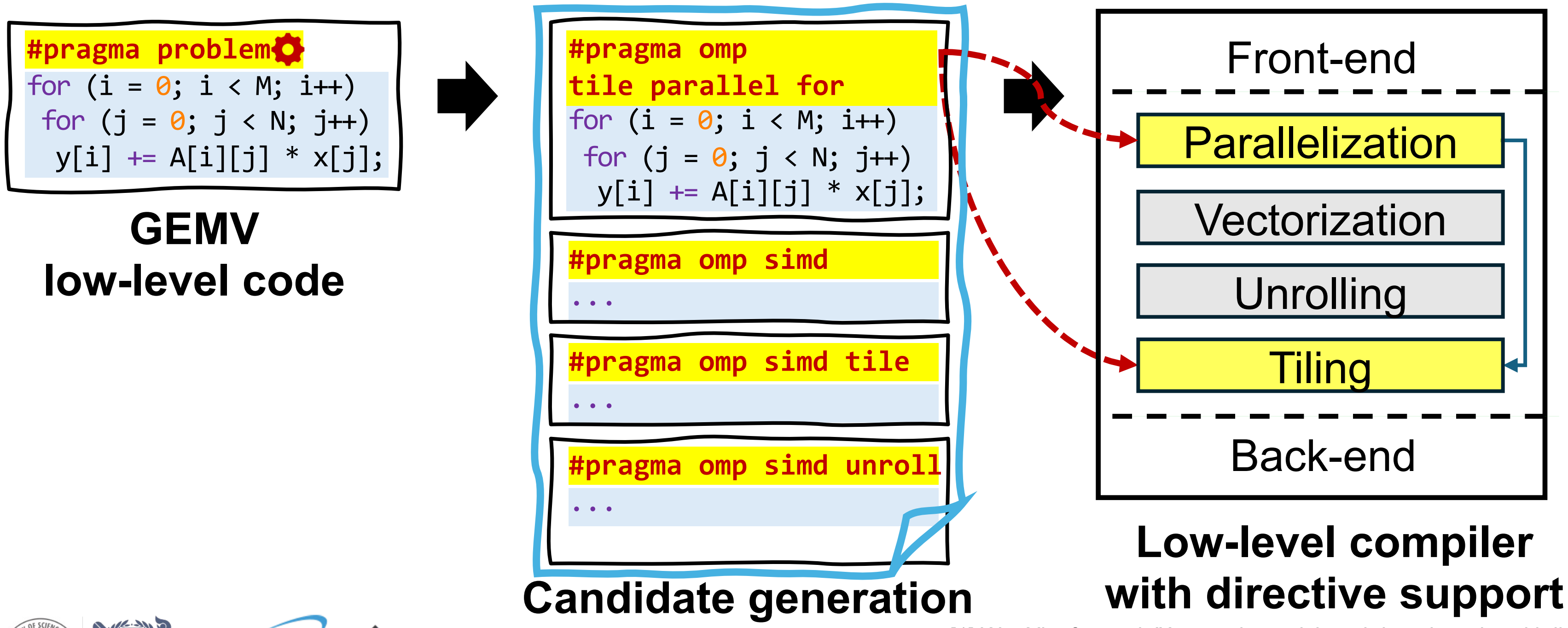


GEMV low-level code | **Candidate generation** | **Low-level compiler with directive support**

[1] Wu, Xingfu, et al. "Autotuning polybench benchmarks with llvm clang/polly loop optimization pragmas using bayesian optimization."
[2] Ansel, Jason, et al. "Opentuner: An extensible framework for program autotuning."
[3] Wu, Xingfu, et al. "ytopt: Autotuning scientific applications for energy efficiency at large scales."

Autotuning for HPC

- **Pragma-based** approaches[1-3] annotate low-level code to guide optimization decisions
- Autotuner uses user annotations to apply different compiler and optimization parameters and generate candidates
  - How codes are transformed is determined by low-level compiler implementations

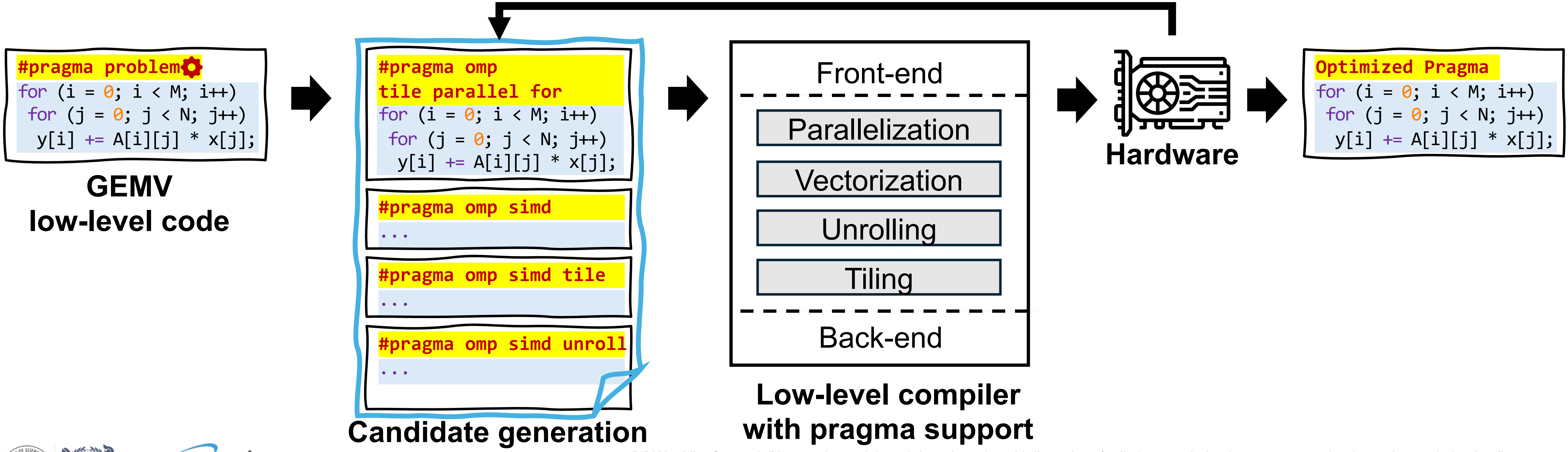


Pragmas only guide optimization passes

[1] Wu, Xingfu, et al. "Autotuning polybench benchmarks with llvm clang/polly loop optimization pragmas using bayesian optimization."
[2] Ansel, Jason, et al. "Opentuner: An extensible framework for program autotuning."
[3] Wu, Xingfu, et al. "ytopt: Autotuning scientific applications for energy efficiency at large scales."

Autotuning for HPC

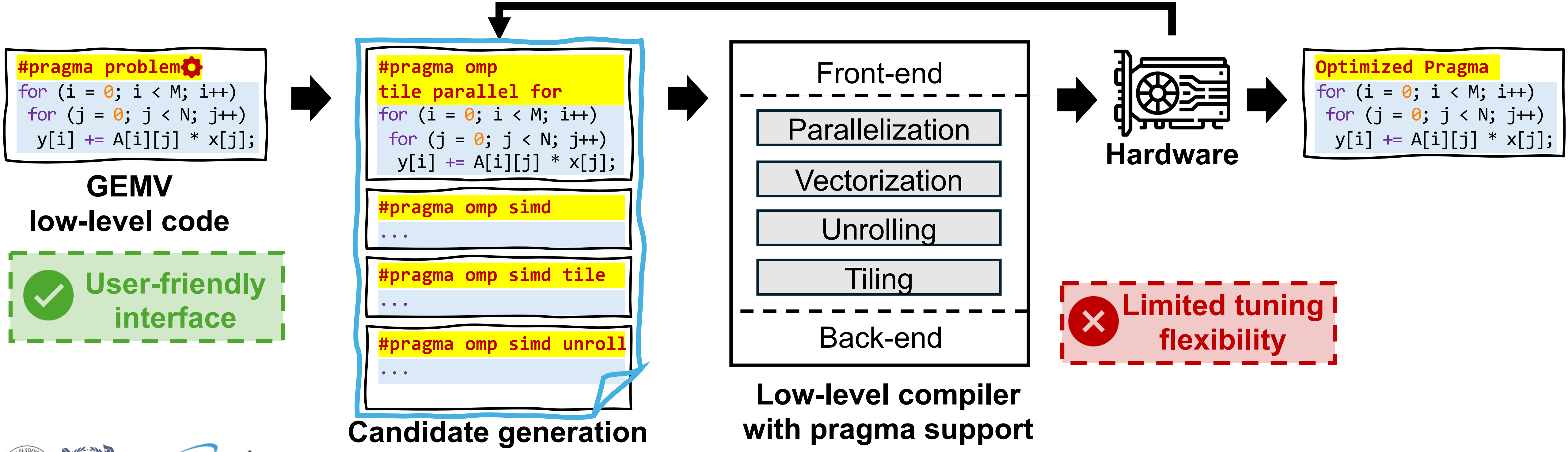
- Pragma-based approaches[1-3] annotate low-level code to guide optimization decisions
- Autotuner uses user annotations to apply different compiler and optimization parameters and generate candidates
  - How codes are transformed is determined by low-level compiler implementations



[1] Wu, Xingfu, et al. "Autotuning polybench benchmarks with llvm clang/polly loop optimization pragmas using bayesian optimization."
[2] Ansel, Jason, et al. "Opentuner: An extensible framework for program autotuning."
[3] Wu, Xingfu, et al. "ytopt: Autotuning scientific applications for energy efficiency at large scales."

Autotuning for HPC

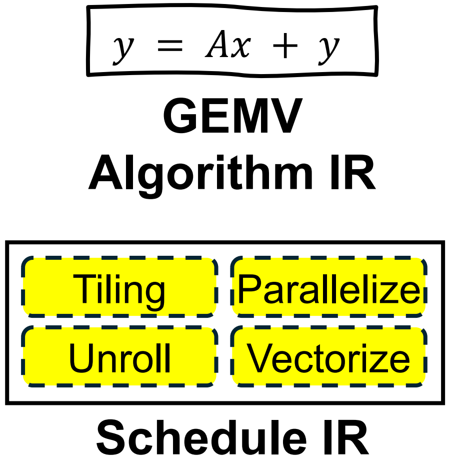
- Enables autotuning of existing low-level code
- Autotuning scope and flexibility are inherently limited by the original code structure and compiler capabilities



[1] Wu, Xingfu, et al. "Autotuning polybench benchmarks with llvm clang/polly loop optimization pragmas using bayesian optimization."
[2] Ansel, Jason, et al. "Opentuner: An extensible framework for program autotuning."
[3] Wu, Xingfu, et al. "ytopt: Autotuning scientific applications for energy efficiency at large scales."

Autotuning for HPC

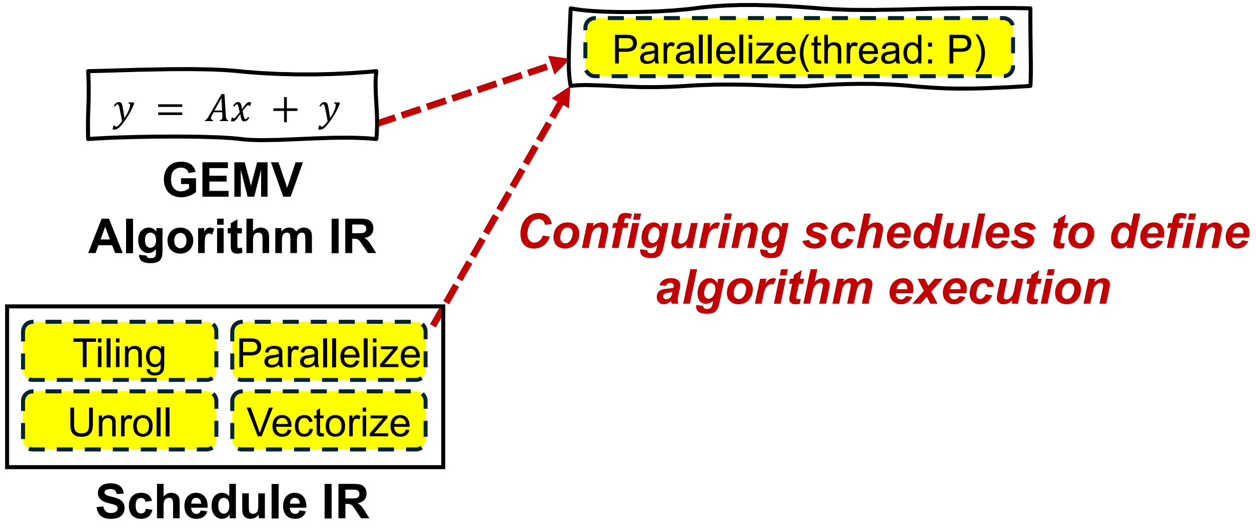
- **Schedule-based** approaches[1-2] use domain-specific IRs to specify high-level algorithms and their implementations



[1] Shao, Junru, et al. "Tensor program optimization with probabilistic programs." Advances in Neural Information Processing Systems
[2] Zheng, Lianmin, et al. "Ansor: Generating {High-Performance} tensor programs for deep learning." OSDI 20

Autotuning for HPC

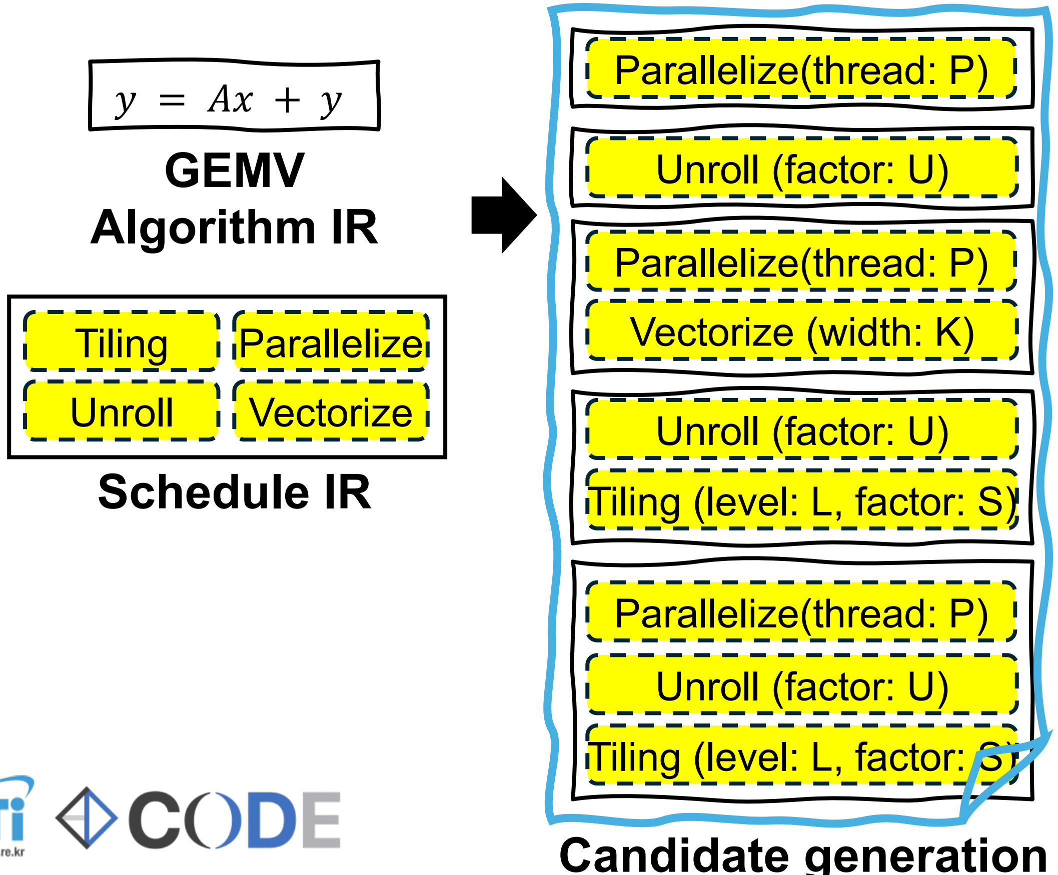
- Schedule-based approaches[1-2] use domain-specific IRs to specify high-level algorithms and their implementations
  - Autotuner generates candidates by composing "schedules" and assigning randomized parameter values



[1] Shao, Junru, et al. "Tensor program optimization with probabilistic programs." Advances in Neural Information Processing Systems
[2] Zheng, Lianmin, et al. "Ansor: Generating {High-Performance} tensor programs for deep learning." OSDI 20

Autotuning for HPC

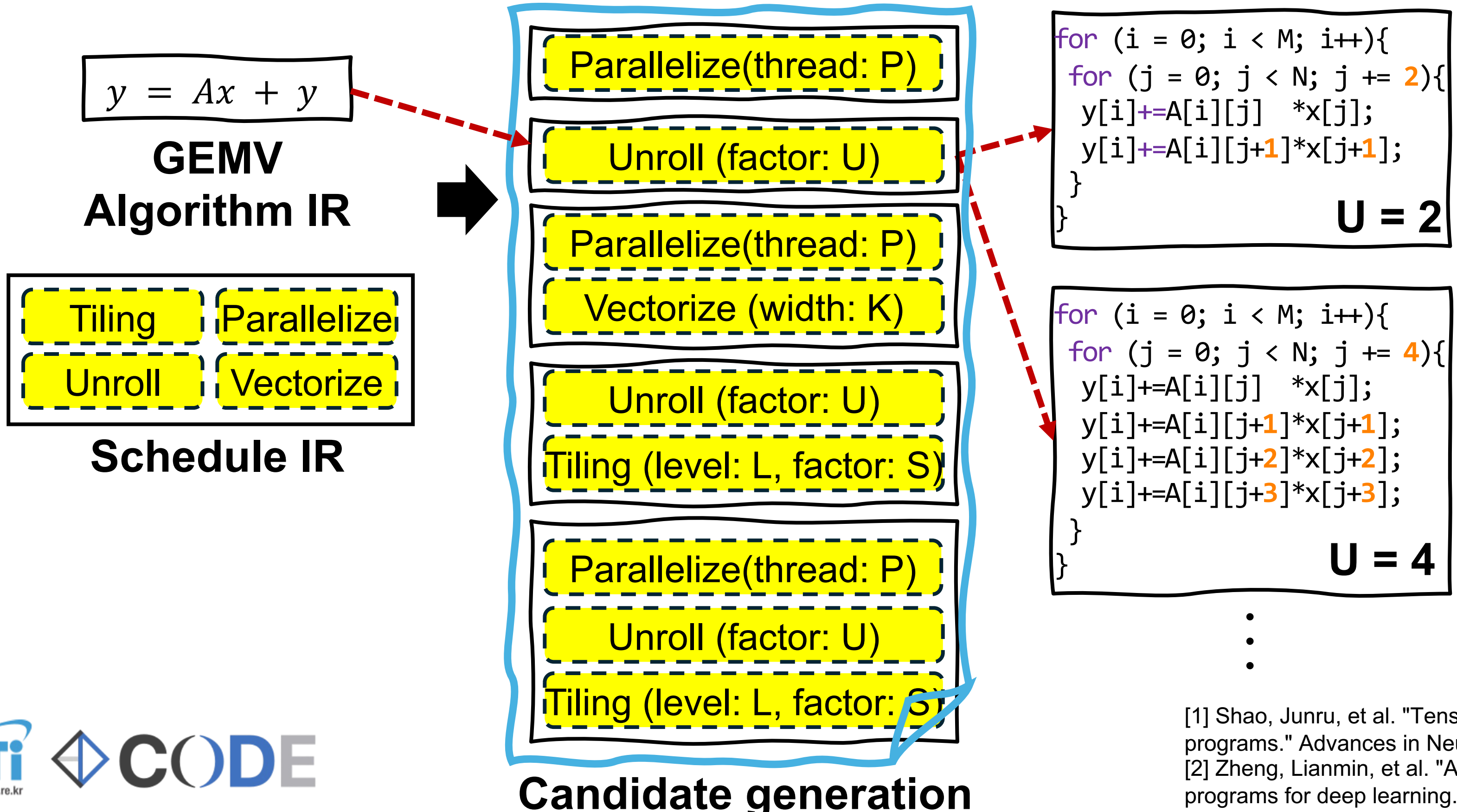
- Schedule-based approaches[1-2] use domain-specific IRs to specify high-level algorithms and their implementations
  - Autotuner generates candidates by composing "schedules" and assigning randomized parameter values



[1] Shao, Junru, et al. "Tensor program optimization with probabilistic programs." Advances in Neural Information Processing Systems
[2] Zheng, Lianmin, et al. "Ansor: Generating {High-Performance} tensor programs for deep learning." OSDI 20

Autotuning for HPC

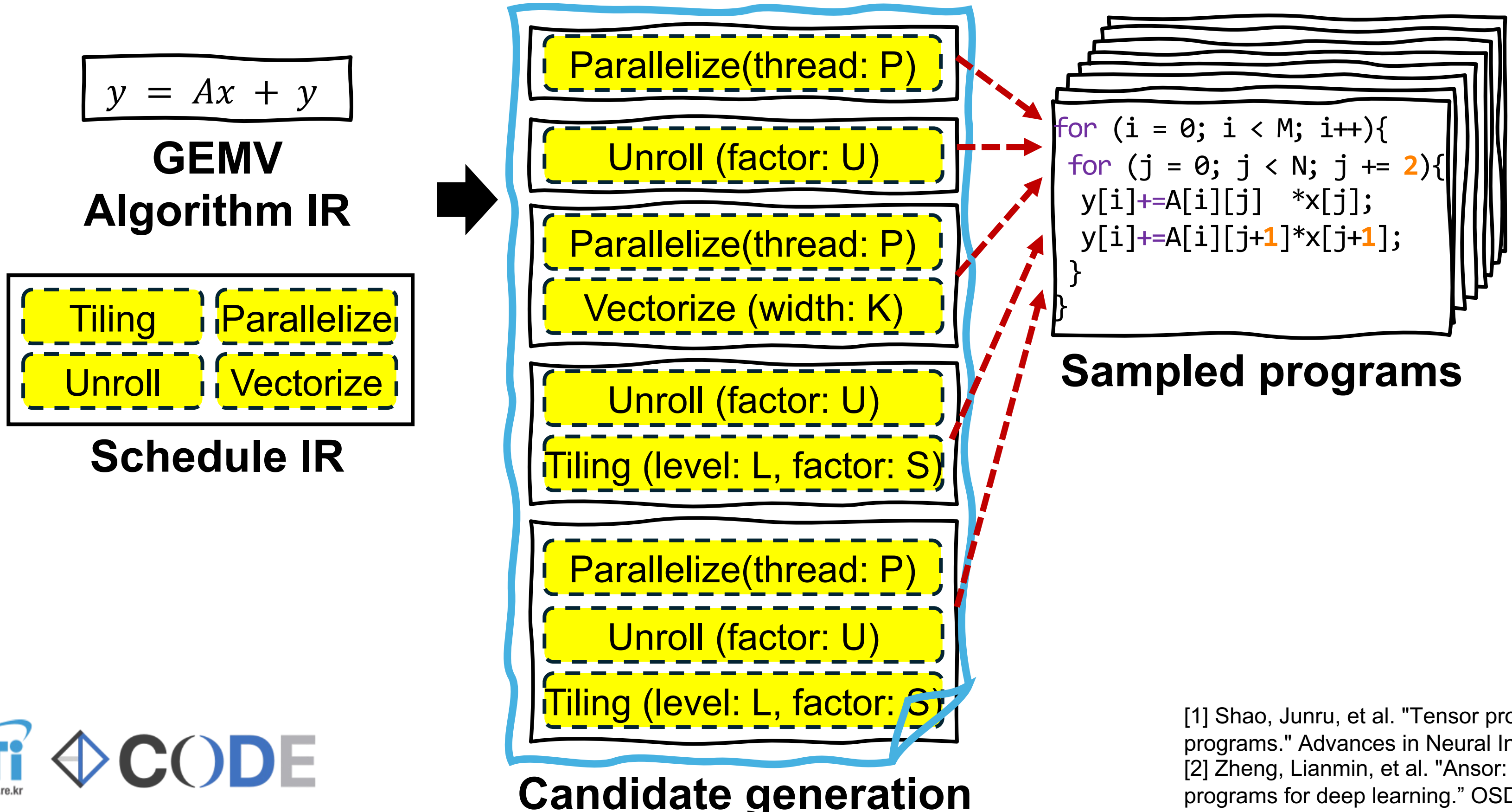
- **Schedule-based** approaches[1-2] use domain-specific IRs to specify high-level algorithms and their implementations
  - Autotuner generates candidates by composing "schedules" and assigning randomized parameter values



[1] Shao, Junru, et al. "Tensor program optimization with probabilistic programs." Advances in Neural Information Processing Systems
[2] Zheng, Lianmin, et al. "Ansor: Generating {High-Performance} tensor programs for deep learning." OSDI 20

Autotuning for HPC

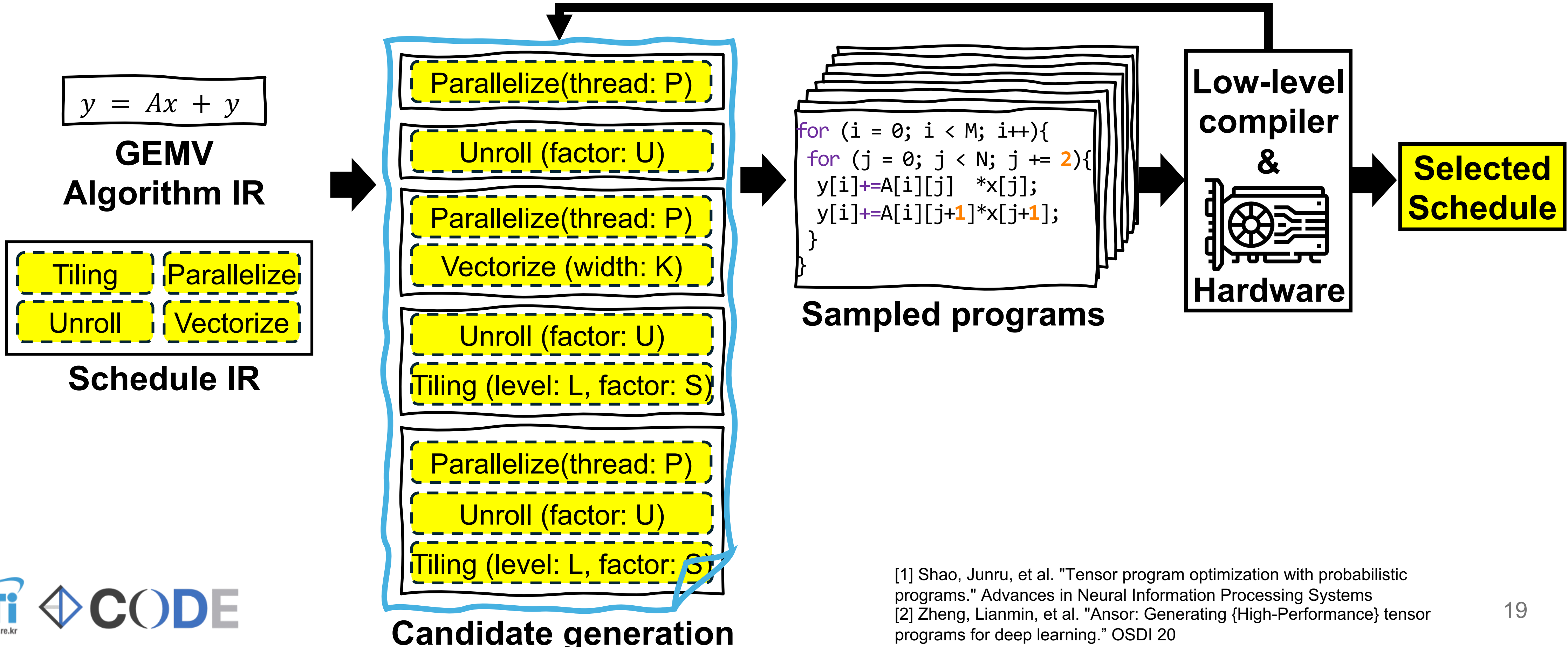
- **Schedule-based** approaches[1-2] use domain-specific IRs to specify high-level algorithms and their implementations



[1] Shao, Junru, et al. "Tensor program optimization with probabilistic programs." Advances in Neural Information Processing Systems
[2] Zheng, Lianmin, et al. "Ansor: Generating {High-Performance} tensor programs for deep learning." OSDI 20

Autotuning for HPC

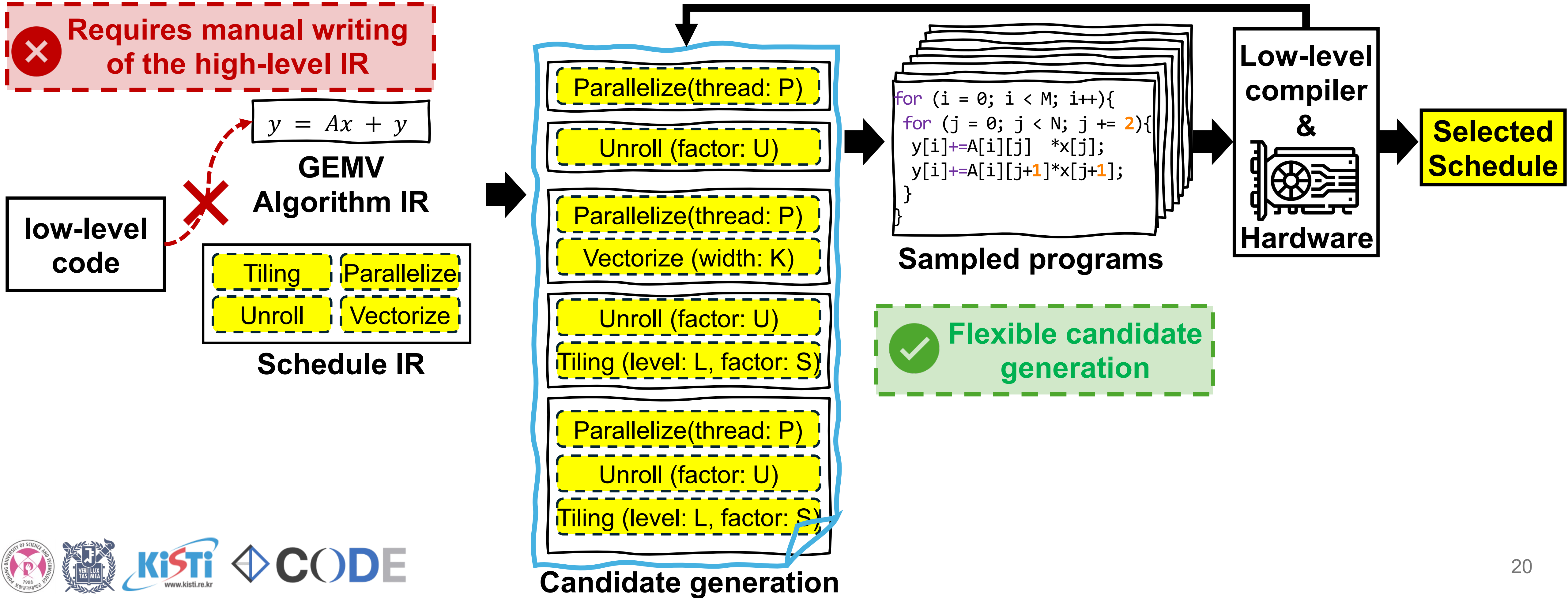
- Schedule-based approaches[1-2] use domain-specific IRs to specify high-level algorithms and their implementations



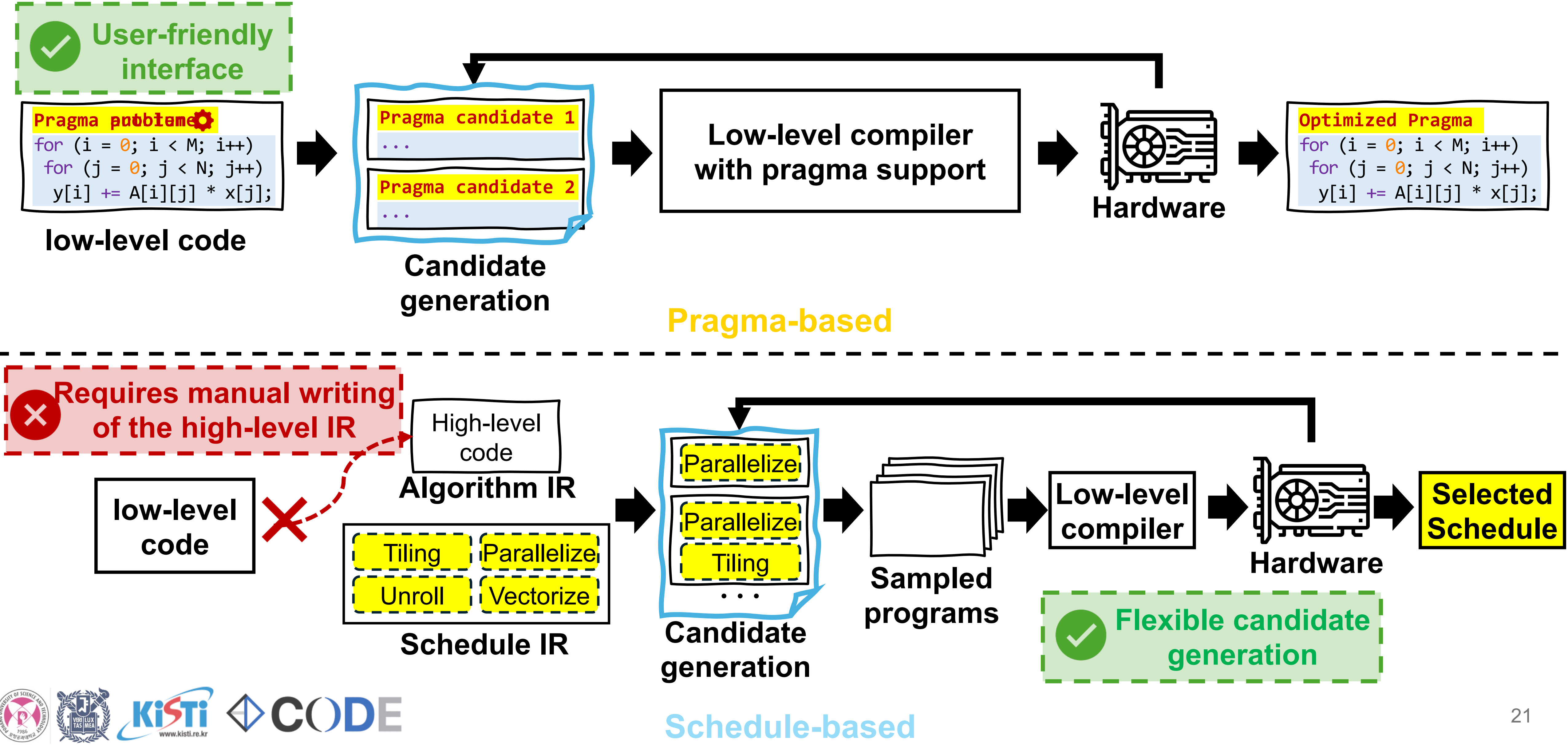
[1] Shao, Junru, et al. "Tensor program optimization with probabilistic programs." Advances in Neural Information Processing Systems
[2] Zheng, Lianmin, et al. "Ansor: Generating {High-Performance} tensor programs for deep learning." OSDI 20

Autotuning for HPC

- **Enables flexible candidate generation with a rich search space**
- **Requires definitions of high-level algorithm and schedule IRs**



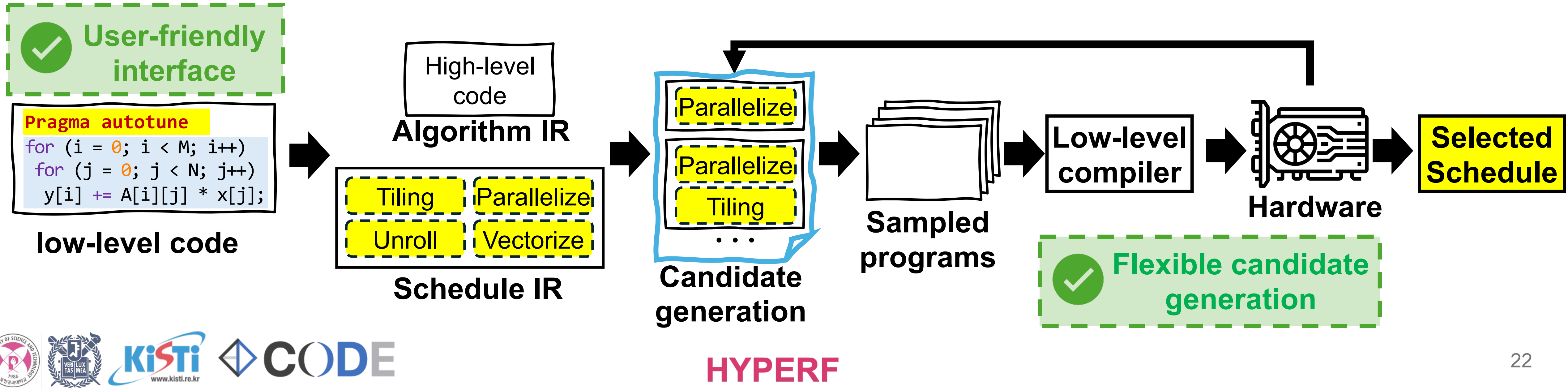
Our Proposal: HYPERF



Our Proposal: HYPERF

Can we combine the best of both pragma-based and schedule-based approaches to build an autotuning solution for HPC?

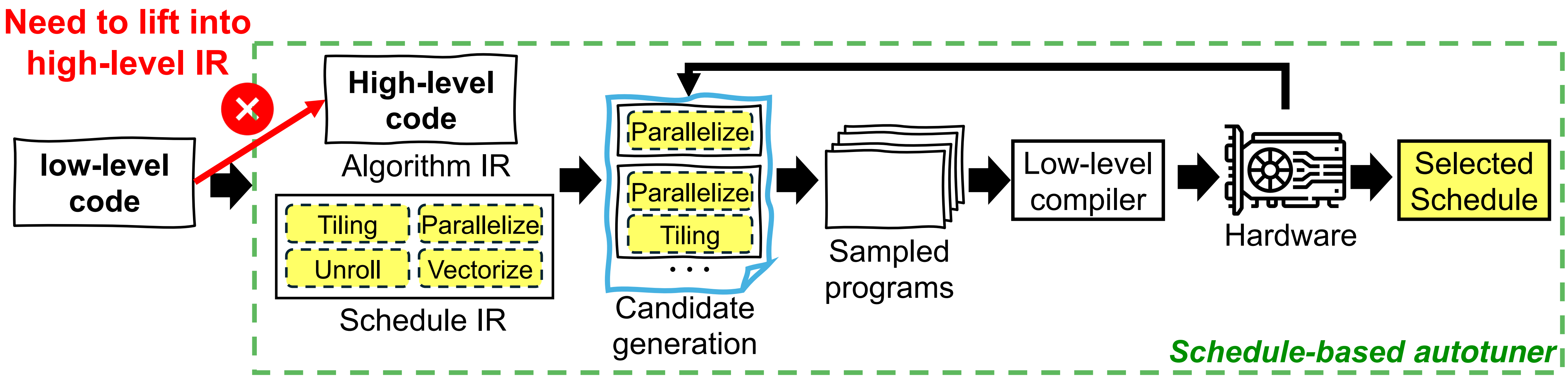
→ HYPERF: End-to-End Autotuning Framework for HPC



Key Challenges

1. Bridging the abstraction gap between HPC loops and algorithm IR

- Translating low-level programs into algorithm IRs, enabling schedule-based autotuning



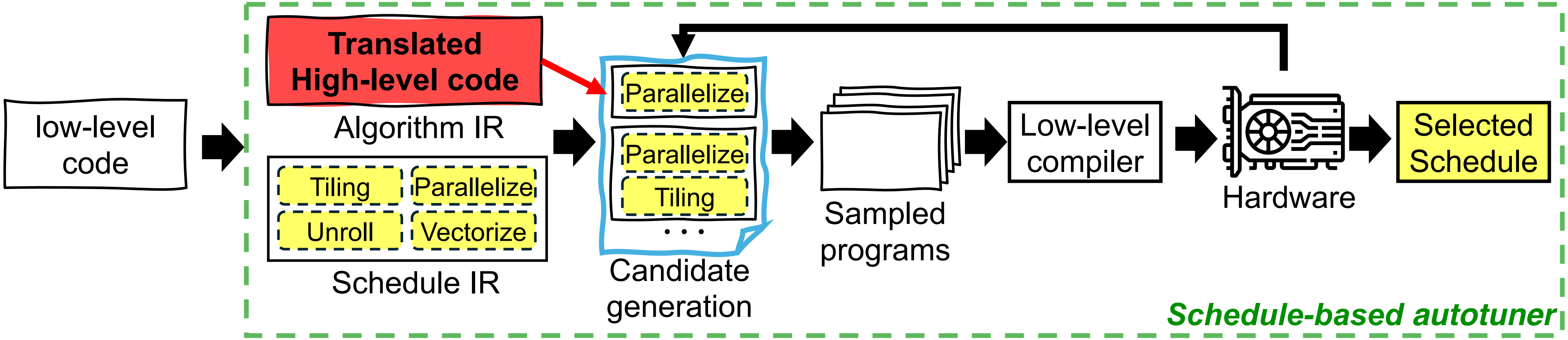
Key Challenges

1. Bridging the abstraction gap between HPC loops and algorithm IR

2. Handling structural differences between HPC and DL loop

- Schedule-based autotuner cannot directly handle complex and arbitrary HPC loop structures

Need to restructure complex HPC loops!

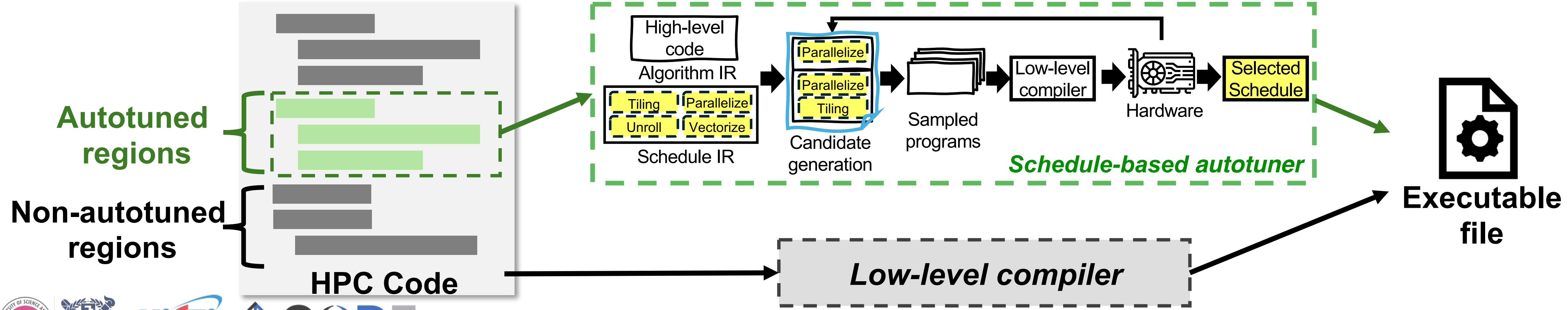


Key Challenges

1. Bridging the abstraction gap between HPC loops and algorithm IR

2. Handling structural differences between HPC and DL loops

3. Integrating compilation flows for a smooth user experience
   - Compile and optimize both autotuned and non-autotuned regions



Our Proposal: HYPERF

1. Bridging the abstraction gap between HPC loops and algorithm IR

→ OpenMP C/C++-to-TIR translator that recovers high-level semantics to enable Schedule-based autotuning

2. Handling structural differences between HPC and DL loop

→ TVM-HPC applies TIR canonicalization and expands the autotuning scope to support arbitrary HPC loop structures

3. Integrating compilation flows for a smooth user experience

→ Autotuning driver replaces autotuned loops with outlined calls and compiles the remaining code

Our Proposal: HYPERF

We propose **HYPERF**, an end-to-end HPC autotuning framework that combines **user-friendly pragma-based interfaces** with **schedule-based autotuning** to achieve flexible and efficient optimization

Our Proposal: HYPERF

HYPERF achieves up to **103.5×** speedup over baseline OpenMP, with an average of **6.1×** over prior HPC autotuners and **4.2×** over polyhedral compilers

Outline

Introduction & Motivation

Background

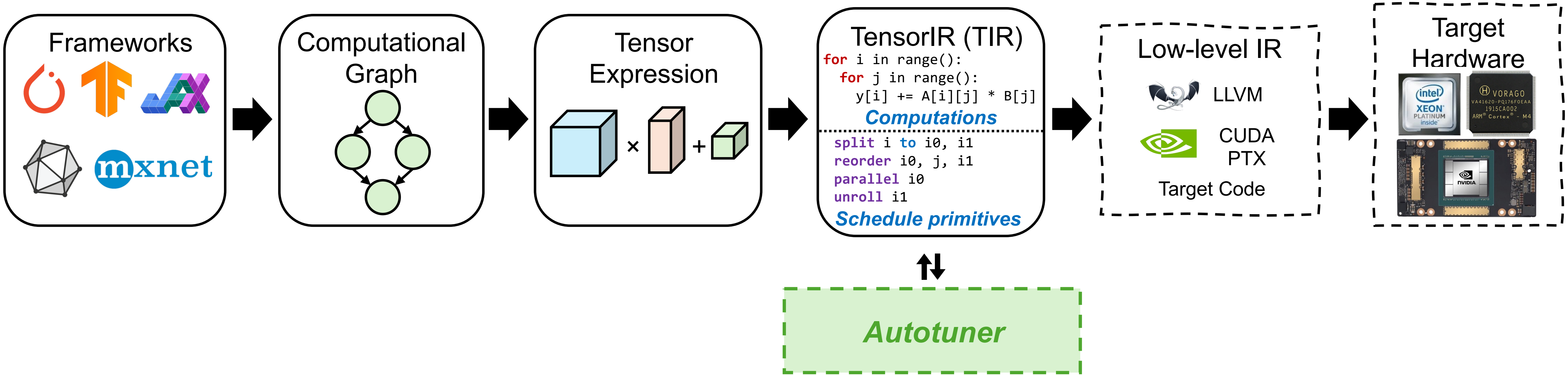
HYPERF

- Overview
- OpenMP C/C++ Autotuning Driver
- TVM-HPC

Evaluation Results

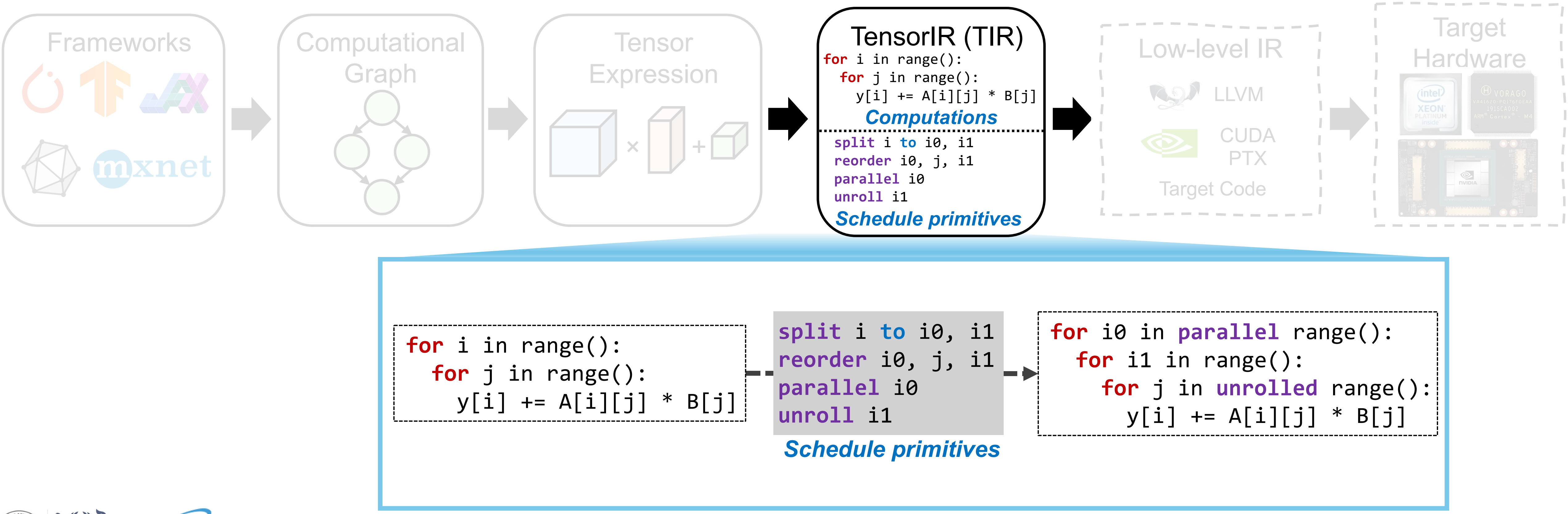
Apache TVM Compiler

- A DL compiler that uses internal IR layers and an autotuner to optimize models for deployment on diverse hardware



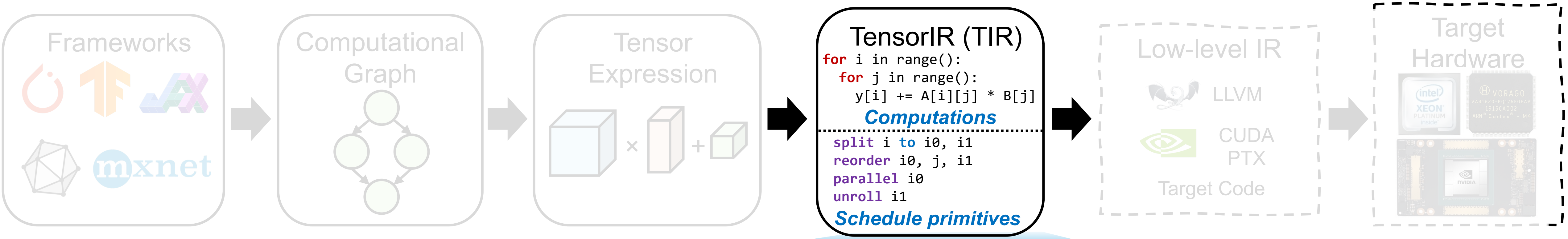
Apache TVM Compiler

- **TensorIR (TIR)** separates the algorithm from the schedule, enabling flexible tensor operations



Apache TVM Compiler

- **TIR blocks** define computations and data within loops, separating the loop structure from computation
- Block includes key info: axis, data access, and initialization

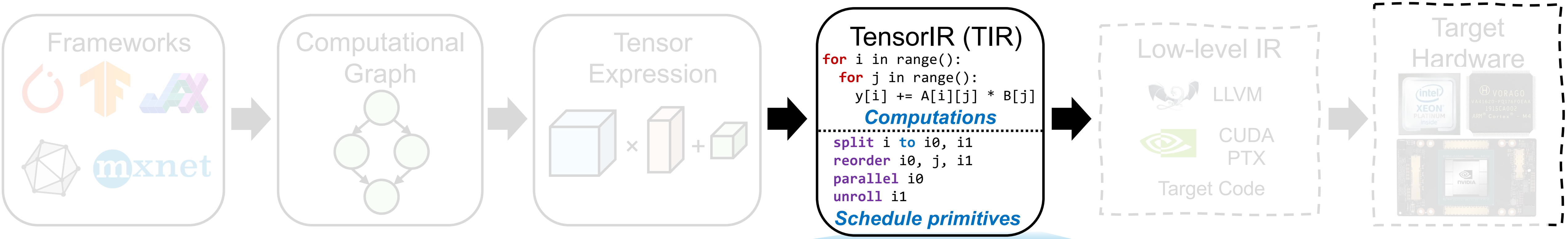



```
for i0 in parallel range():
  for i1 in range():
    for j in unrolled range():
      with tir.block("block"):
        i = tir.axis.spatial(i)
        j = tir.axis.reduce(j)
        tir.reads(A, B)
        tir.writes(y)
        with tir.init():
        y[i] += A[i][j] * B[j]
```

- Block axis
- Block access info
- Block initialization

Apache TVM Compiler

- **TIR blocks** define computations and data within loops, separating the loop structure from computation
- Block includes key info: axis, data access, and initialization

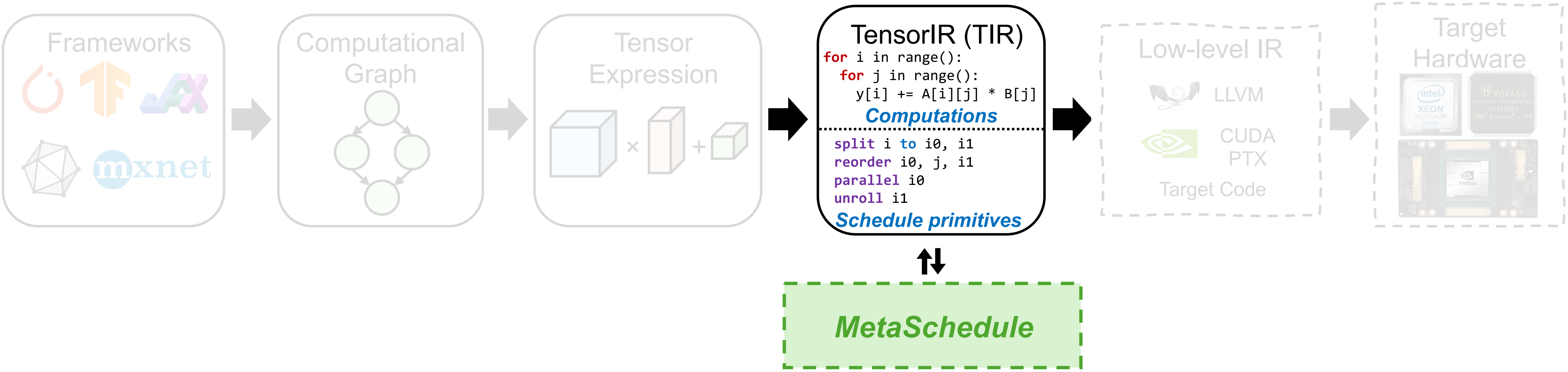



```
for i0 in parallel range():
  for i1 in range():
    for j in unrolled range():
      with tir.block("block"):
        i = tir.axis.spatial(i)
        j = tir.axis.reduce(j)
        tir.reads(A, B)
        tir.writes(y)
        with tir.init():
        y[i] += A[i][j] * B[j]
```

i → spatial (output index)
j → reduction (accumulate)

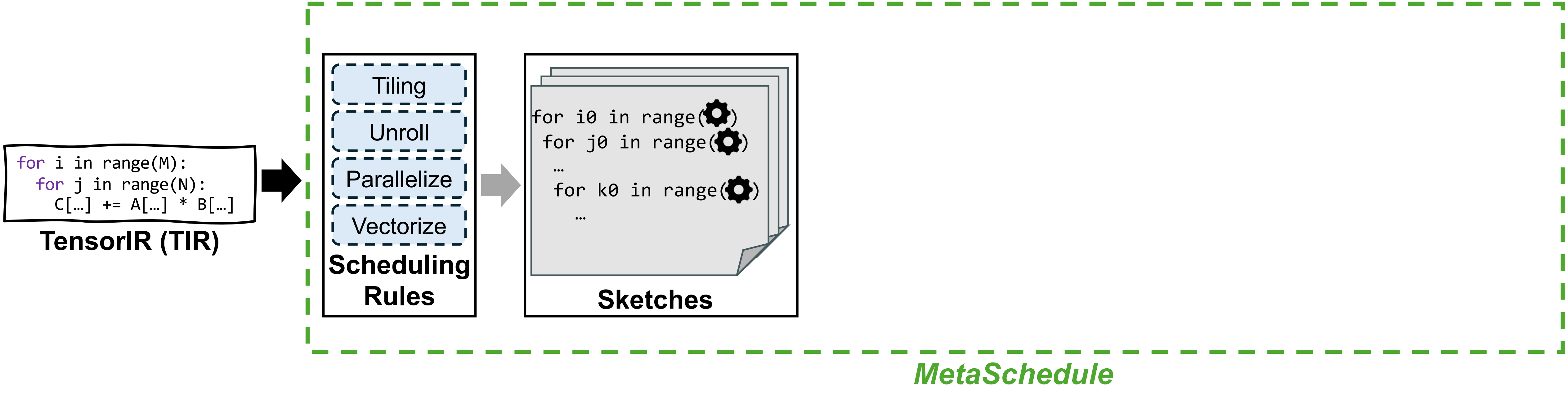
Apache TVM Compiler

- **MetaSchedule** autotunes schedule primitives to find the best-performing versions



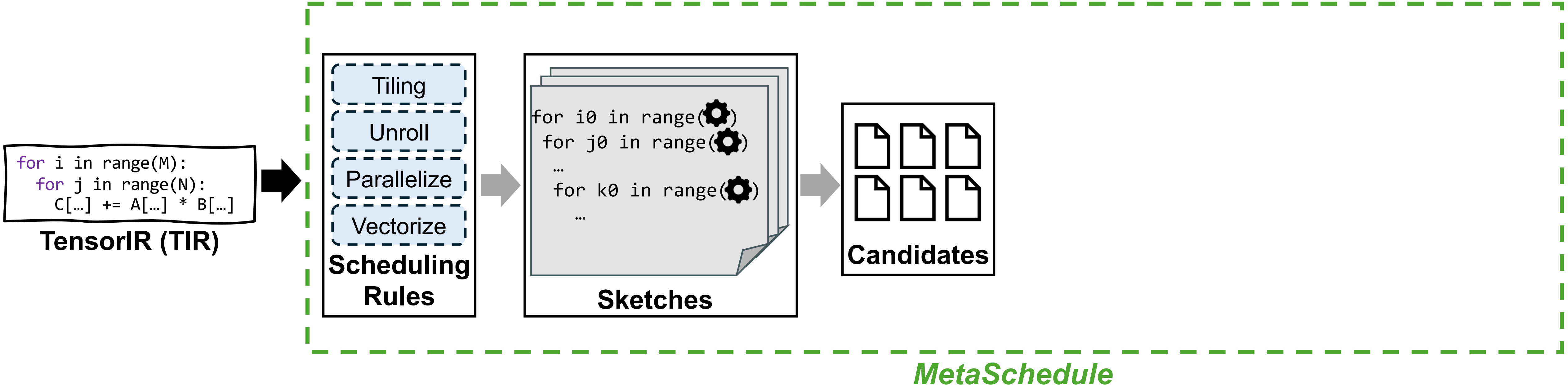
Apache TVM Compiler

- **MetaSchedule** autotunes schedule primitives to find the best-performing versions
  - Generates multiple sketches based on predefined scheduling rules



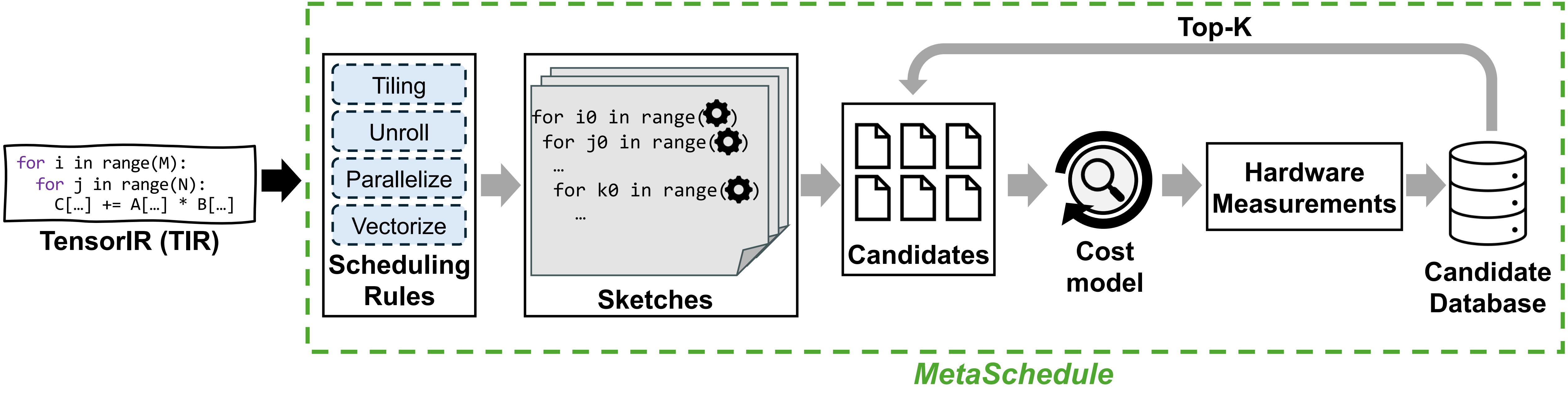
Apache TVM Compiler

- **MetaSchedule** autotunes schedule primitives to find the best-performing versions
  - Produces various candidates by adjusting parameters within these sketches



Apache TVM Compiler

- **MetaSchedule** autotunes schedule primitives to find the best-performing versions
  - Selects candidates with a cost model, runs them, and finds the best schedule



Outline

Introduction & Motivation

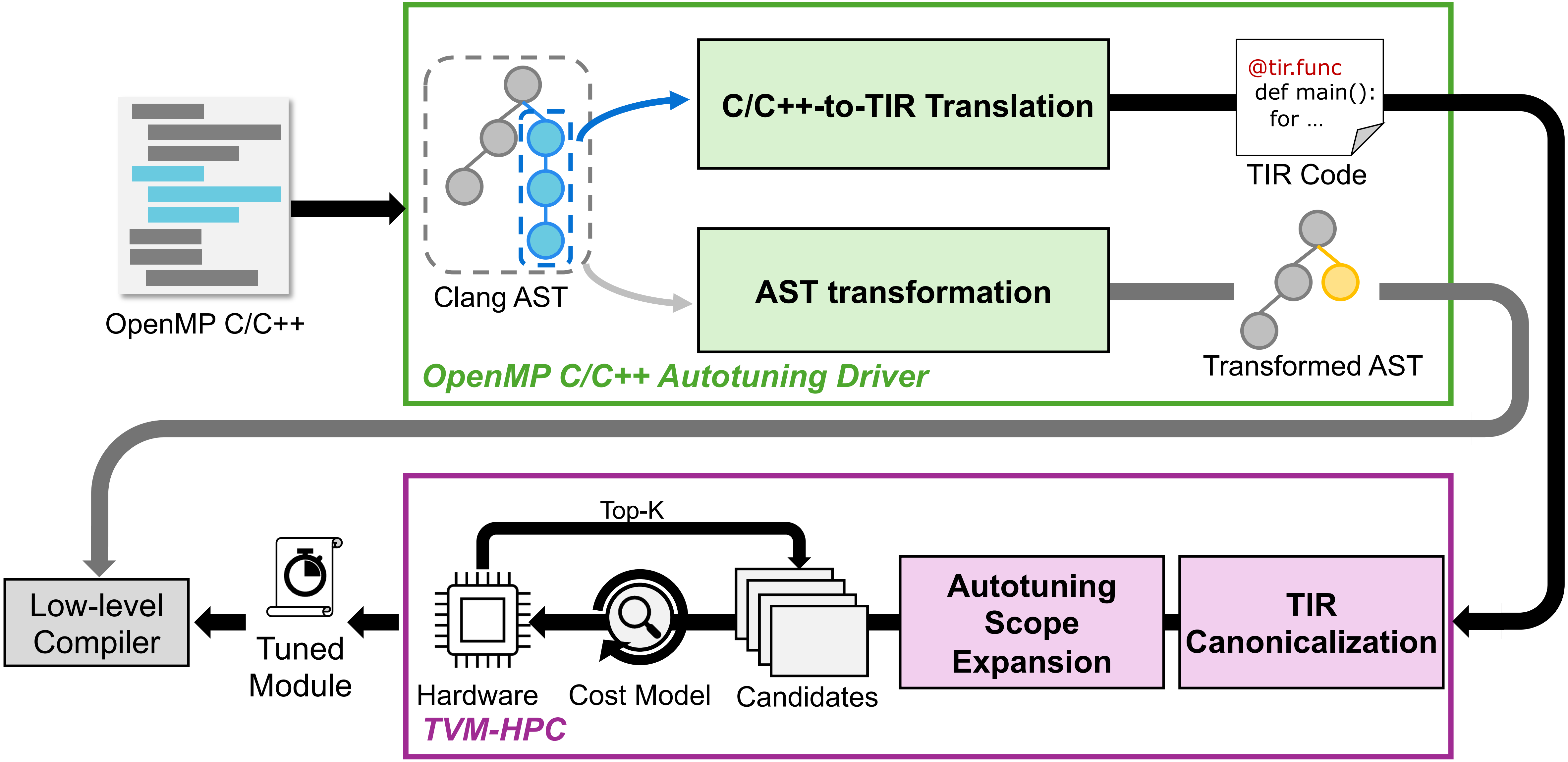
Background

HYPERF

- **Overview**
- OpenMP C/C++ Autotuning Driver
- TVM-HPC

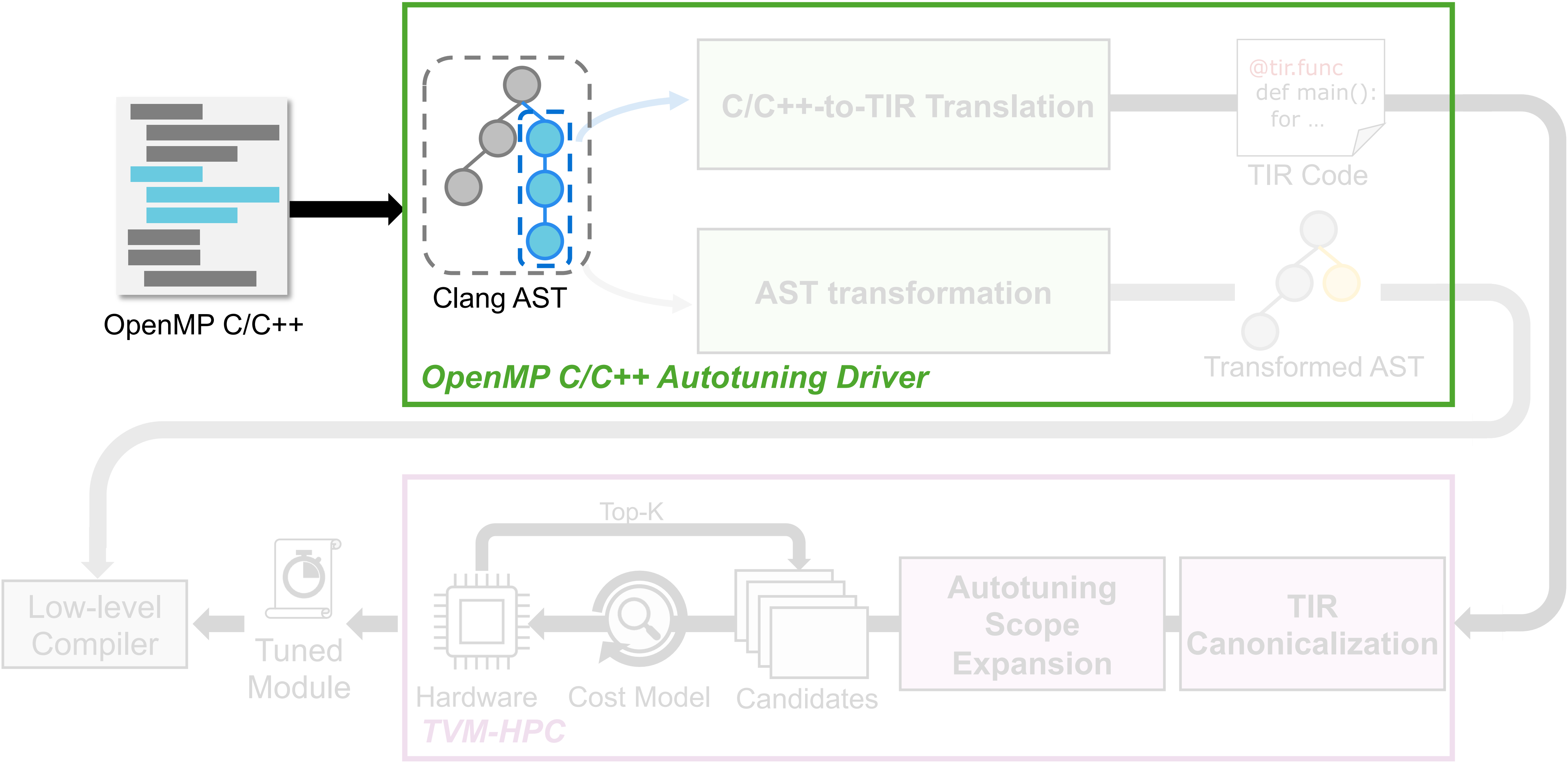
Evaluation Results

Overview

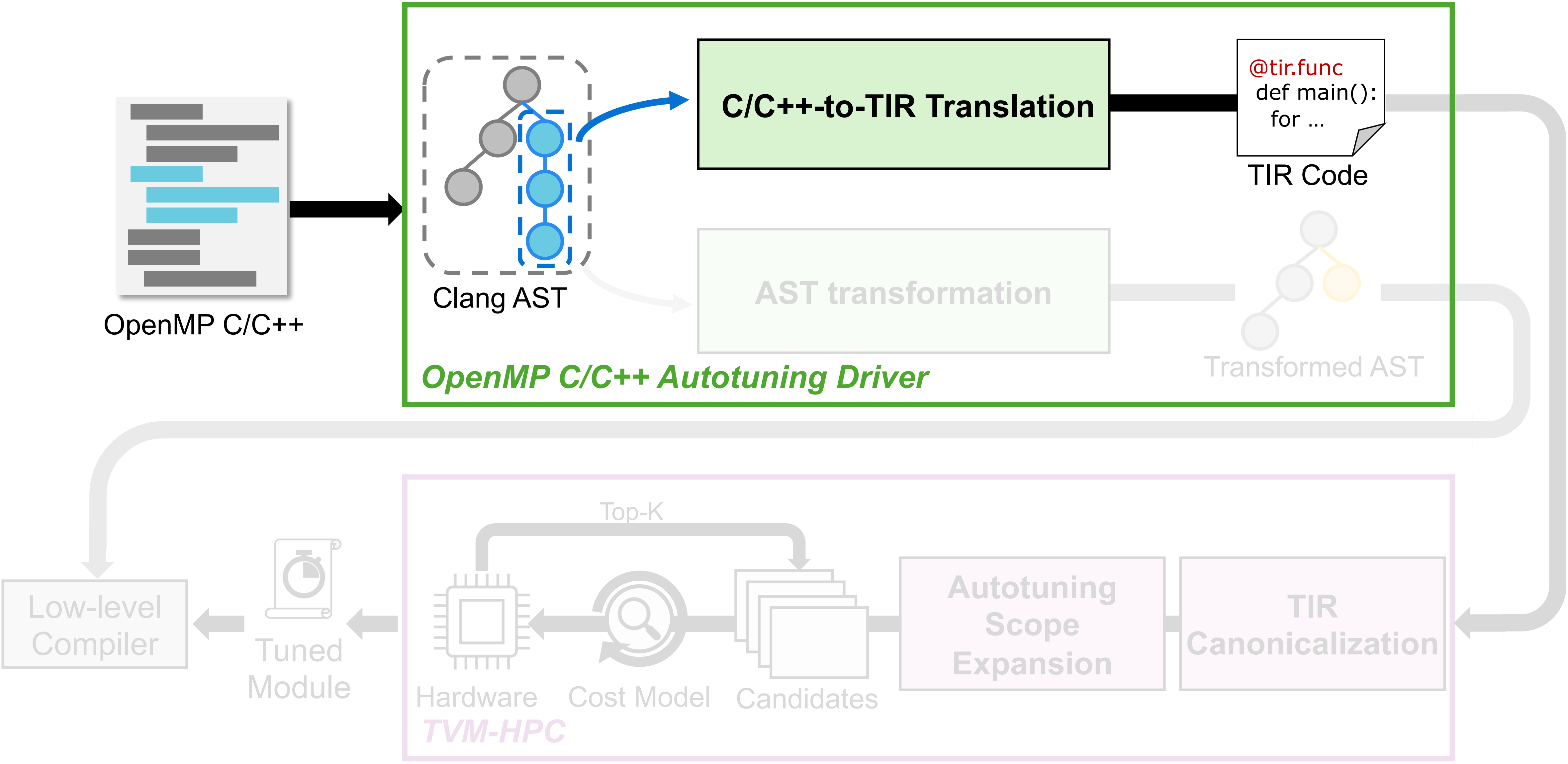


OpenMP C/C++
Clang AST
C/C++-to-TIR Translation
@tir.func
def main():
for ...
TIR Code
AST transformation
Transformed AST
OpenMP C/C++ Autotuning Driver
Top-K
Low-level Compiler
Tuned Module
Hardware
Cost Model
Candidates
Autotuning Scope Expansion
TIR Canonicalization
TVM-HPC

Overview

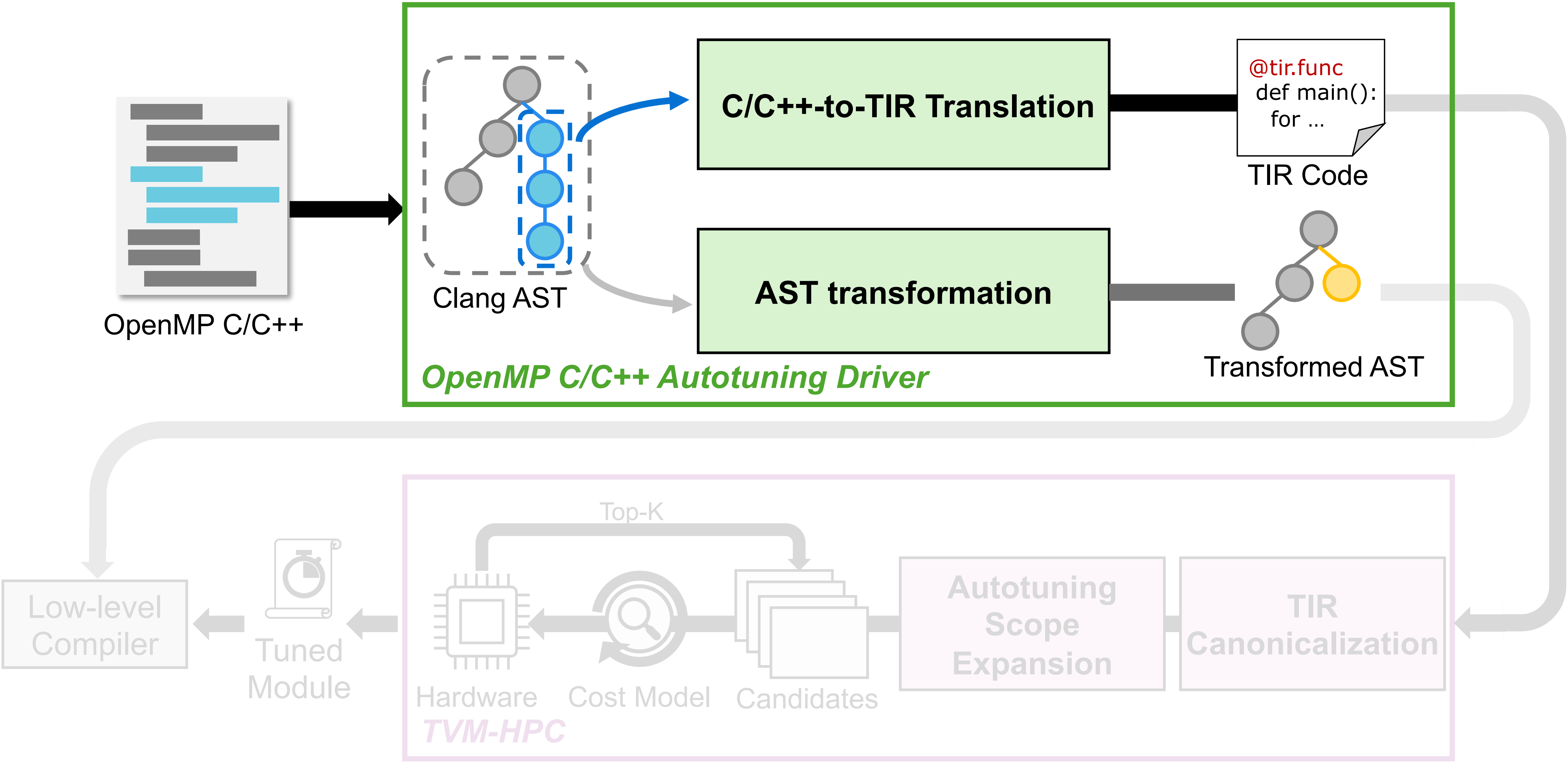

OpenMP C/C++
Clang AST
C/C++-to-TIR Translation
@tir.func
def main():
for ...
TIR Code
AST transformation
Transformed AST
OpenMP C/C++ Autotuning Driver
Top-K
Low-level Compiler
Tuned Module
Hardware
Cost Model
Candidates
Autotuning Scope Expansion
TIR Canonicalization
TVM-HPC

Overview



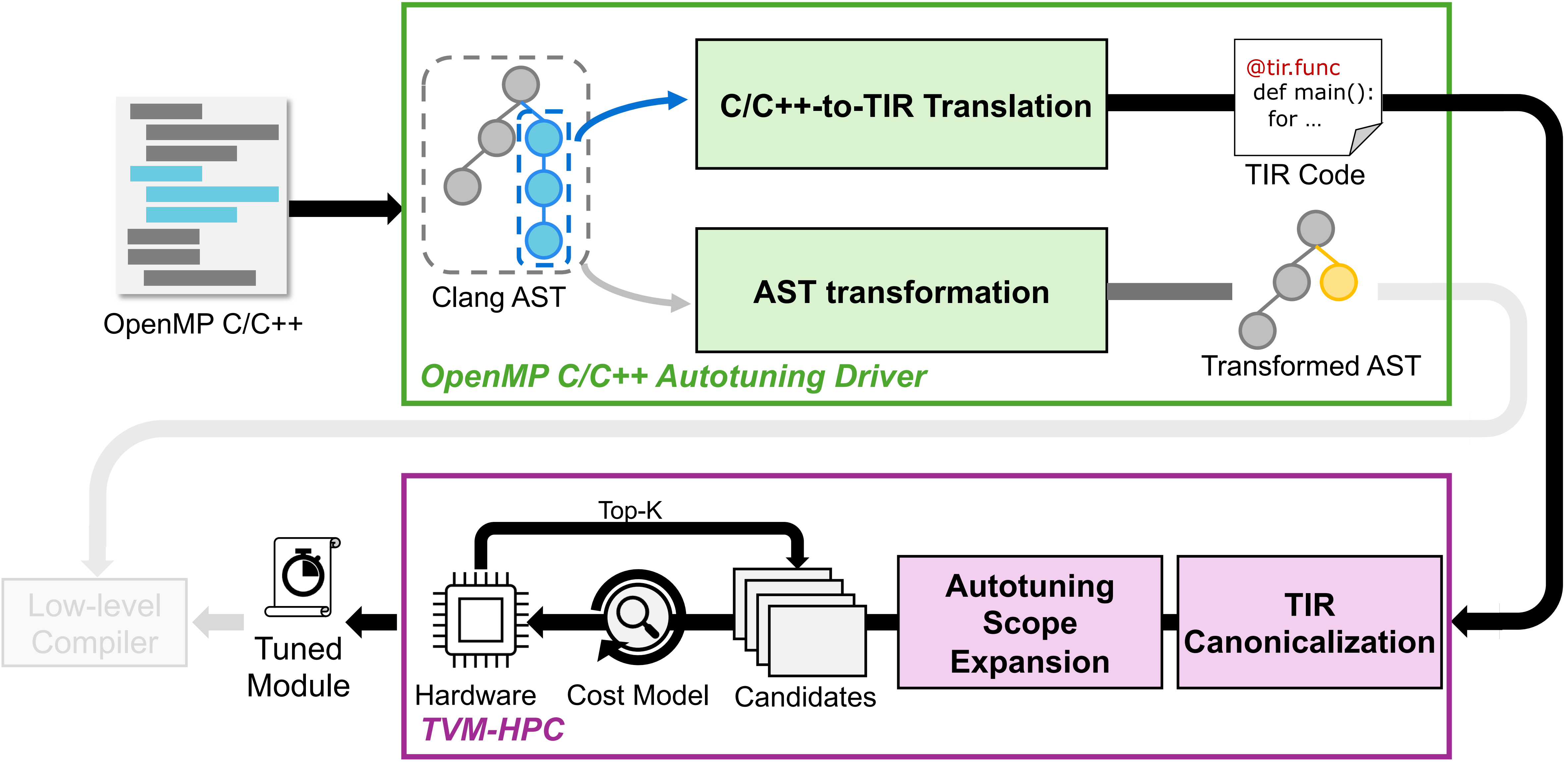
OpenMP C/C++
Clang AST
C/C++-to-TIR Translation
@tir.func
def main():
for ...
TIR Code
AST transformation
Transformed AST
OpenMP C/C++ Autotuning Driver
Top-K
Low-level Compiler
Tuned Module
Hardware
Cost Model
Candidates
Autotuning Scope Expansion
TIR Canonicalization
TVM-HPC

Overview



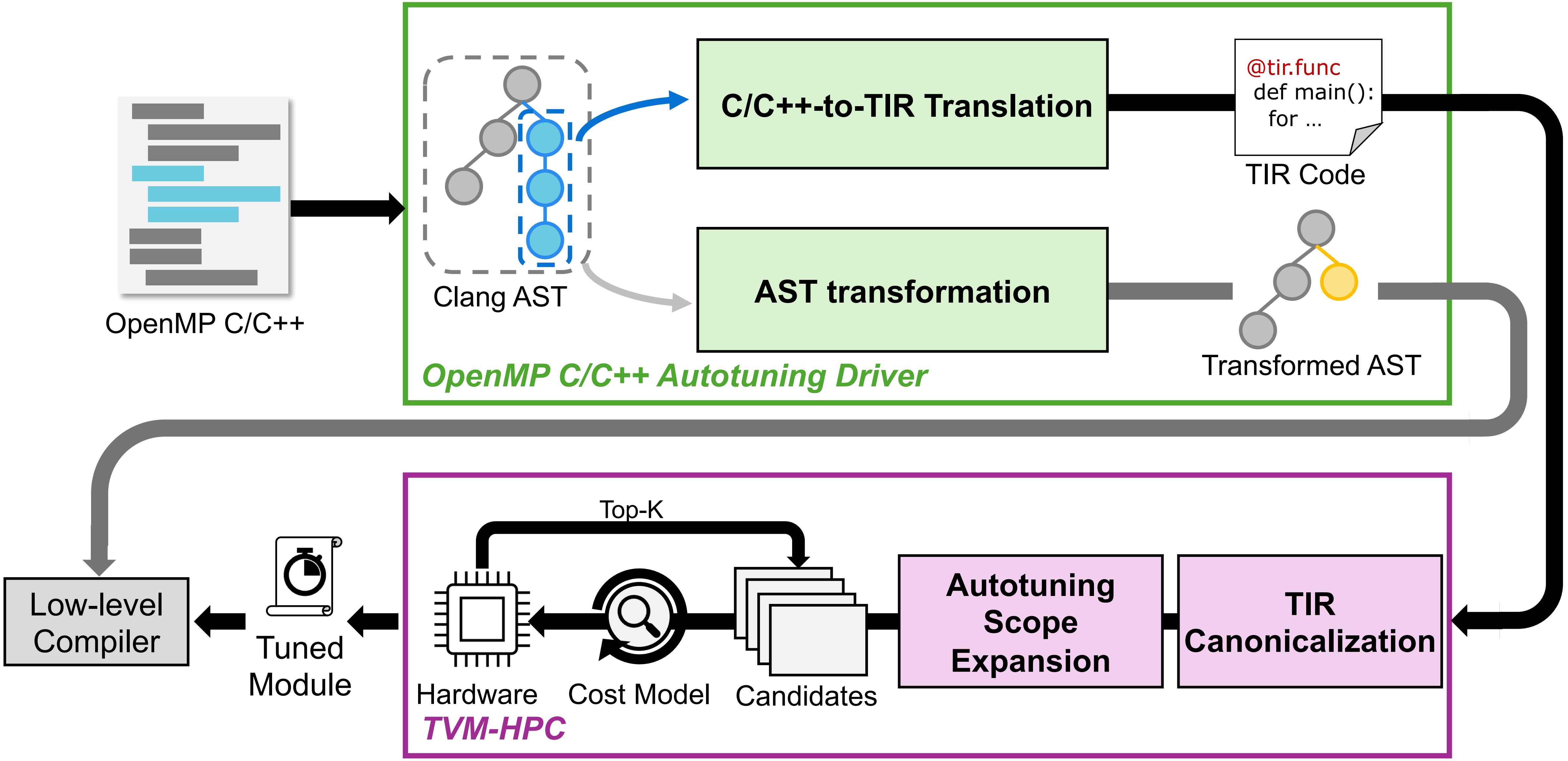
OpenMP C/C++
Clang AST
C/C++-to-TIR Translation
@tir.func
def main():
for ...
TIR Code
AST transformation
Transformed AST
OpenMP C/C++ Autotuning Driver
Top-K
Low-level Compiler
Tuned Module
Hardware
Cost Model
Candidates
Autotuning Scope Expansion
TIR Canonicalization
TVM-HPC

Overview



OpenMP C/C++
Clang AST
C/C++-to-TIR Translation
@tir.func
def main():
for ...
TIR Code
AST transformation
Transformed AST
OpenMP C/C++ Autotuning Driver
Top-K
Low-level
Compiler
Tuned
Module
Hardware
Cost Model
Candidates
Autotuning
Scope
Expansion
TIR
Canonicalization
TVM-HPC

Overview



OpenMP C/C++
Clang AST
C/C++-to-TIR Translation
@tir.func
def main():
for ...
TIR Code
AST transformation
Transformed AST
OpenMP C/C++ Autotuning Driver
Top-K
Autotuning Scope Expansion
TIR Canonicalization
Hardware
Cost Model
Candidates
TVM-HPC
Tuned Module
Low-level Compiler

Outline

Introduction & Motivation

Background

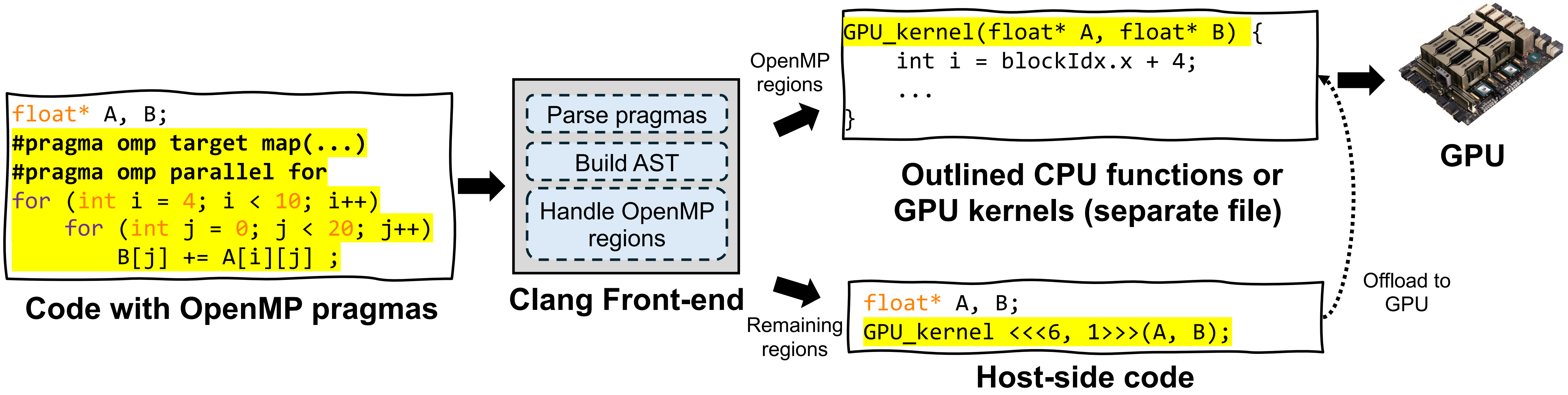
HYPERF

- Overview
- **OpenMP C/C++ Autotuning Driver**
- TVM-HPC

Evaluation Results

OpenMP C/C++ Autotuning Driver

- In an OpenMP compilation, pragmas are handled in the front-end and transformed into outlined functions or GPU kernels for offloading



OpenMP C/C++ Autotuning Driver

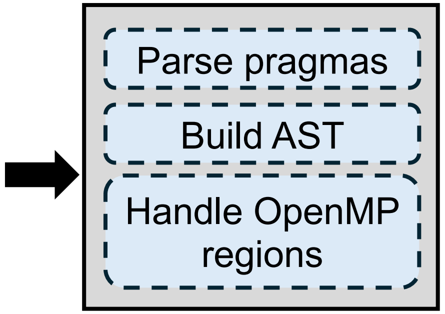
HYPERF extends the **OpenMP programming model with an autotune directive** and **follows a similar compilation flow**

Mark autotuning regions using an OpenMP pragma interface

```
float* A, B;
#pragma omp autotune for
for (int i = 4; i < 10; i++)
    for (int j = 0; j < 20; j++)
        B[j] += A[i][j] ;
```

Code with OpenMP pragmas

OpenMP for Offloading-based Autotuning



Clang Front-end

OpenMP C/C++ Autotuning Driver

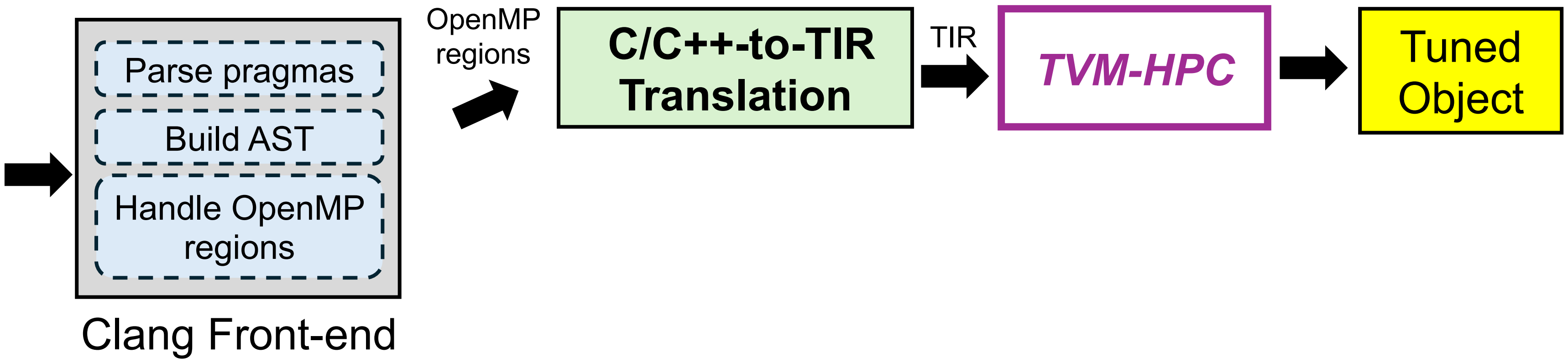
HYPERF extends the **OpenMP programming model with an autotune directive** and **follows a similar compilation flow**

Translate identified OpenMP regions to TIR and offload to TVM-HPC for tuning

```
float* A, B;
#pragma omp autotune for
for (int i = 4; i < 10; i++)
    for (int j = 0; j < 20; j++)
        B[j] += A[i][j] ;
```

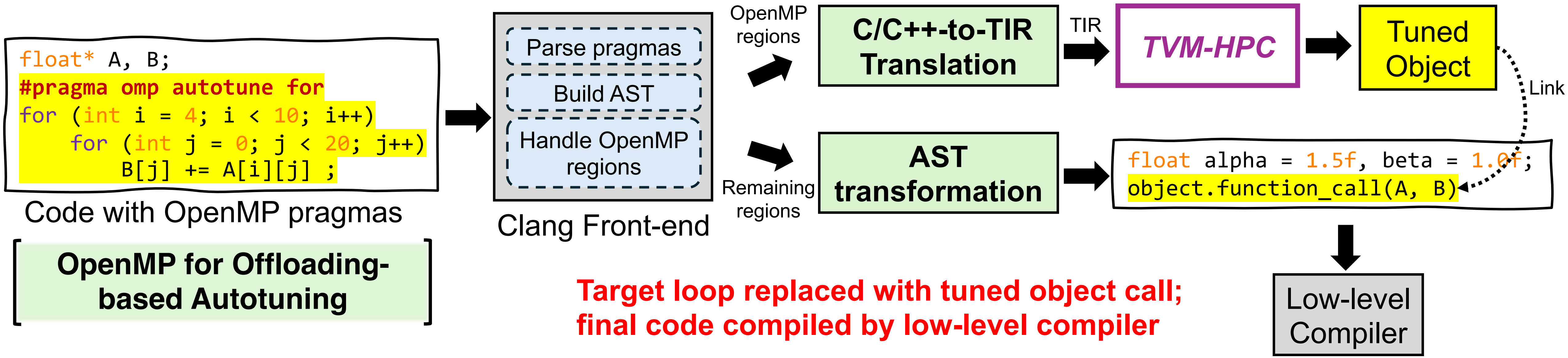
Code with OpenMP pragmas

OpenMP for Offloading-based Autotuning



OpenMP C/C++ Autotuning Driver

HYPERF extends the **OpenMP programming model with an autotune directive** and **follows a similar compilation flow**



Target loop replaced with tuned object call; final code compiled by low-level compiler

OpenMP for Offloading-based Autotuning

- The **‘autotune for’** directive specifies an autotuning target

Directive and Clause	Syntax	Note
Autotune for Directive	`#pragma omp autotune for [clause[ ],..]`	• Specifies loop autotuning via OpenMP pragmas

OpenMP for Offloading-based Autotuning

- `map clause:` Specifies shared data pointers and array metadata

Directive and Clause	Syntax	Note
Autotune for Directive	`#pragma omp autotune for [clause[ ],..]`	• Specifies loop autotuning via OpenMP pragmas
|--- **Map Clause**	`map(map-type: locator-list)`	• `locator-list:` List of arrays with bounds and lengths • `map-type:` Data movement direction

OpenMP for Offloading-based Autotuning

- **`reduction clause:`** Specifies reduction variables and sets block axis properties

Directive and Clause	Syntax	Note
Autotune for Directive	`#pragma omp autotune for [clause[ ],..]`	• Specifies loop autotuning via OpenMP pragmas
|--- **Map Clause**	`map(map-type: locator-list)`	• `locator-list:` List of arrays with bounds and lengths • `map-type:` Data movement direction
|--- **Reduction Clause**	`reduction(op: list)`	• `op:` Specifies the reduction operator • `list:` Variables to be reduced across threads

OpenMP for Offloading-based Autotuning

- **`private clause:`** Specifies a private variable
- **`struct_info clause:`** Provides information on struct members

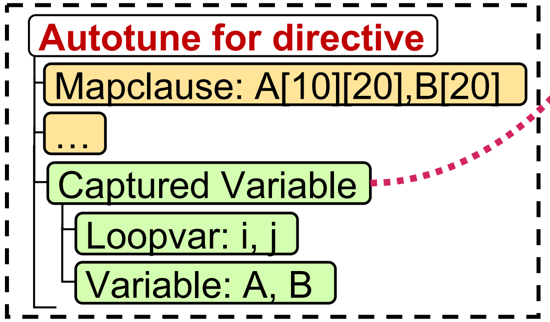
Directive and Clause	Syntax	Note
Autotune for Directive	`#pragma omp autotune for [clause[ ],..]`	• Specifies loop autotuning via OpenMP pragmas
|--- **Map Clause**	`map(map-type: locator-list)`	• `locator-list`: List of arrays with bounds and lengths • `map-type`: Data movement direction
|--- **Reduction Clause**	`reduction(op: list)`	• `op`: Specifies the reduction operator • `list`: Variables to be reduced across threads
|--- **Private Clause**	`private(list)`	• `list`: Thread-private variable
|--- **Struct Info Clause**	`struct_info(struct-list)`	• `struct-list`: Structure elements included in the OpenMP region

OpenMP C/C++-to-TIR Translation

1. Variable analysis: Identify and analyze variables via AST and clauses to construct a symbol table of TIR variables

OpenMP C/C++ Code

```
#pragma omp autotune for
map(tofrom: A[0:10][0:20]…)
reduction(+: B[0:20])
for (int i = 4; i < 10; i++)
  for (int j = 0; j < 20; j++)
    B[j] += A[i][j] ;
```



Clang AST

Variable: "i", type: int, shape: []
Variable: "j", type: int, shape: []
Variable: "A" — type: array(float), shape: []
Variable: "B" — type: array(float), shape: []

Variables used in the loop body

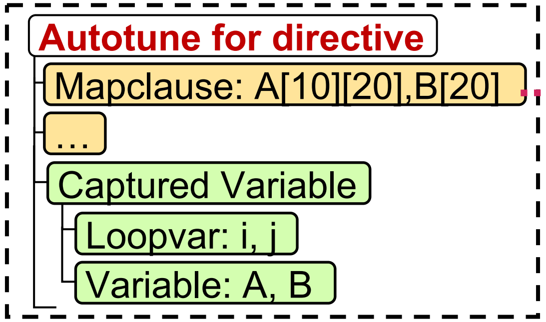
Transformed TIR

OpenMP C/C++-to-TIR Translation

1. Variable analysis: Identify and analyze variables via AST and clauses to construct a symbol table of TIR variables

OpenMP C/C++ Code

```
#pragma omp autotune for
map(tofrom: A[0:10][0:20]…)
reduction(+: B[0:20])
for (int i = 4; i < 10; i++)
  for (int j = 0; j < 20; j++)
    B[j] += A[i][j] ;
```



Clang AST

Variable: "i", type: int, shape: []
Variable: "j", type: int, shape: []
Variable: "A" — type: array(float), shape: [10, 20]
Variable: "B" — type: array(float), shape: [20]

Variables used in the loop body

Transformed TIR

OpenMP C/C++-to-TIR Translation

1. Variable analysis: Identify and analyze variables via AST and clauses to construct a symbol table of TIR variables

OpenMP C/C++ Code

```
#pragma omp autotune for
map(tofrom: A[0:10][0:20]…)
reduction(+: B[0:20])
for (int i = 4; i < 10; i++)
  for (int j = 0; j < 20; j++)
    B[j] += A[i][j] ;
```

- Autotune for directive
  - Mapclause: A[10][20],B[20]
  - …
  - Captured Variable
    - Loopvar: i, j
    - Variable: A, B

Clang AST

Variable: "i", type: int, shape: []
Variable: "j", type: int, shape: []
Variable: "A" — type: array(float), shape: [10, 20]
Variable: "B" — type: array(float), shape: [20]

Variables used in the loop body

```
tir::Var(i,int)
tir::Var(j,int)
tir.buffer([10][20], float, A)
tir.buffer([20], float, B)
```

TIR primitive representation

Transformed TIR

OpenMP C/C++-to-TIR Translation

1. Variable analysis: Identify and analyze variables via AST and clauses to construct a symbol table of TIR variables

OpenMP C/C++ Code

```
#pragma omp autotune for
map(tofrom: A[0:10][0:20]…)
reduction(+: B[0:20])
for (int i = 4; i < 10; i++)
  for (int j = 0; j < 20; j++)
    B[j] += A[i][j] ;
```

⬇

- Autotune for directive
  - Mapclause: A[10][20],B[20]
  - …
  - Captured Variable
    - Loopvar: i, j
    - Variable: A, B

Clang AST

Variable: "i", type: int, shape: []
Variable: "j", type: int, shape: []
Variable: "A" — type: array(float), shape: [10, 20]
Variable: "B" — type: array(float), shape: [20]

Variables used in the loop body

```
tir::Var(i,int)
tir::Var(j,int)
tir.buffer([10][20], float, A)
tir.buffer([20], float, B)
```

TIR primitive representation

i	tir::Var(i,int)
j	tir::Var(j,int)
A	tir.buffer([10][20], float, A)
B	tir.buffer([20], float, B)

Symbol table

Transformed TIR

OpenMP C/C++-to-TIR Translation

2. Loop structure analysis and generation: Analyze AST loop nodes to generate valid TIR loop headers (variable, start, extent)

OpenMP C/C++ Code

```
#pragma omp autotune for
map(tofrom: A[0:10][0:20]…)
reduction(+: B[0:20])
for (int i = 4; i < 10; i++)
  for (int j = 0; j < 20; j++)
    B[j] += A[i][j] ;
```

- Autotune for directive
  - Captured body
    - Forstmt
      - var=i, start=4, end=10
      - Body: Forstmt
        - var=j, start=0, end=20

Clang AST

Loop variable: "i" , start: 4, extent: 10
Loop variable: "j" , start: 0, extent: 20

```
for i in range(4, 10):
  for j in range(20):
```

TIR Loop structure

Transformed TIR

i	tir::Var(i,int)
j	tir::Var(j,int)
A	tir.buffer([10][20], float, A)
B	tir.buffer([20], float, B)

Symbol table

OpenMP C/C++-to-TIR Translation

2. Loop structure analysis and generation: Analyze AST loop nodes to generate valid TIR loop headers (variable, start, extent)

OpenMP C/C++ Code

```
#pragma omp autotune for
map(tofrom: A[0:10][0:20]…)
reduction(+: B[0:20])
for (int i = 4; i < 10; i++)
  for (int j = 0; j < 20; j++)
    B[j] += A[i][j] ;
```

⬇

- Autotune for directive
  - Captured body
    - Forstmt
      - var=i, start=4, end=10
      - Body: Forstmt
        - var=j, start=0, end=20

Clang AST

```
for i in range(4, 10):
   for j in range(20):
```

TIR Loop structure

10-4

TIR programs require start values to be zero!

```
for i_off in range(6):
  for j in range(20):
    tir.LetStmt(i=i_off+4)
```

Transformed TIR

i	tir::Var(i,int)
j	tir::Var(j,int)
A	tir.buffer([10][20], float, A)
B	tir.buffer([20], float, B)

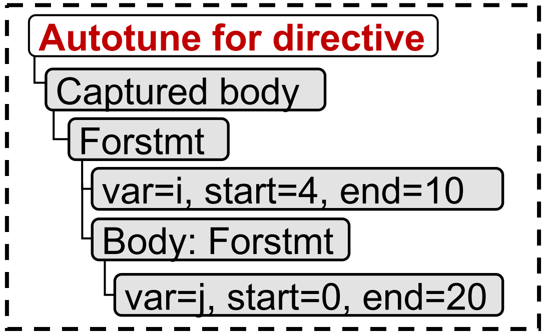
Symbol table

OpenMP C/C++-to-TIR Translation

2. Loop structure analysis and generation: Analyze AST loop nodes to generate valid TIR loop headers (variable, start, extent)

OpenMP C/C++ Code

```
#pragma omp autotune for
map(tofrom: A[0:10][0:20]…)
reduction(+: B[0:20])
for (int i = 4; i < 10; i++)
  for (int j = 0; j < 20; j++)
    B[j] += A[i][j] ;
```



Clang AST

```
for i_off in range(6):
  for j in range(20):
    tir.LetStmt(i=i_off+4)
```

TIR Loop structure

Transformed TIR

```
for i_off in range(6):
  for j in range(20):
    tir.LetStmt(i=i_off+4)
```

i	tir::Var(i,int)
j	tir::Var(j,int)
A	tir.buffer([10][20], float, A)
B	tir.buffer([20], float, B)

Symbol table

OpenMP C/C++-to-TIR Translation

3. Loop body generation: Convert AST nodes to TIR operations based on translation rules

OpenMP C/C++ Code

```
#pragma omp autotune for
map(tofrom: A[0:10][0:20]…)
reduction(+: B[0:20])
for (int i = 4; i < 10; i++)
  for (int j = 0; j < 20; j++)
    B[j] += A[i][j] ;
```

- Autotune for directive
  - Captured body
    - Forstmt
      - Body: Forstmt
        - Body: BinaryOp(+=)
          - LHS: ArraySubscriptExpr (B, [j])
          - RHS: ArraySubscriptExpr (A, [i , j])

Clang AST

Clang AST	TVM TIR
ArraySubscriptExpr(var, index)	tir::Load(var, index)

variable: "A", index: [i, j]
variable: "B", index: [j]

```
rhs=tir.load(A[i][j])
lhs=tir.load(B[j])
```

Transformed TIR

```
for i_off in range(6):
  for j in range(20):
    tir.LetStmt(i=i_off+4)
```

i	tir::Var(i,int)
j	tir::Var(j,int)
A	tir.buffer([10][20], float, A)
B	tir.buffer([20], float, B)

Symbol table

OpenMP C/C++-to-TIR Translation

3. Loop body generation: Convert AST nodes to TIR operations based on translation rules

OpenMP C/C++ Code

```
#pragma omp autotune for
map(tofrom: A[0:10][0:20]…)
reduction(+: B[0:20])
for (int i = 4; i < 10; i++)
  for (int j = 0; j < 20; j++)
    B[j] += A[i][j] ;
```

- Autotune for directive
  - Captured body
    - Forstmt
      - Body: Forstmt
        - Body: BinaryOp(+=)
          - LHS: ArraySubscriptExpr (B, [j])
          - RHS: ArraySubscriptExpr (A, [i , j])

Clang AST

Clang AST	TVM TIR
ArraySubscriptExpr(var, index)	tir::Load(var, index)
BinaryOp(lhs, +, rhs)	tir::<add>(lhs, rhs)
BinaryOp(lhs, =, rhs)	tir::Store(lhs, rhs)

B[j] = LHS + RHS

`tir.add(lhs, rhs)`

```
rhs=tir.load(A[i][j])
lhs=tir.load(B[j])
```

Transformed TIR

```
for i_off in range(6):
  for j in range(20):
    tir.LetStmt(i=i_off+4)
```

i	tir::Var(i,int)
j	tir::Var(j,int)
A	tir.buffer([10][20], float, A)
B	tir.buffer([20], float, B)

Symbol table

OpenMP C/C++-to-TIR Translation

3. Loop body generation: Convert AST nodes to TIR operations based on translation rules

OpenMP C/C++ Code

```
#pragma omp autotune for
map(tofrom: A[0:10][0:20]…)
reduction(+: B[0:20])
for (int i = 4; i < 10; i++)
  for (int j = 0; j < 20; j++)
    B[j] += A[i][j] ;
```

- Autotune for directive
  - Captured body
    - Forstmt
      - Body: Forstmt
        - Body: BinaryOp(+=)
          - LHS: ArraySubscriptExpr (B, [j])
          - RHS: ArraySubscriptExpr (A, [i , j])

Clang AST

Clang AST	TVM TIR
ArraySubscriptExpr(var, index)	tir::Load(var, index)
BinaryOp(lhs, +, rhs)	tir::<add>(lhs, rhs)
BinaryOp(lhs, =, rhs)	tir::Store(lhs, rhs)

B[j] = LHS + RHS

`tir.store(B[j],tir.add(lhs, rhs))`

```
rhs=tir.load(A[i][j])
lhs=tir.load(B[j])
tir.store(B[j],tir.add(lhs, rhs))
```

Transformed TIR

```
for i_off in range(6):
  for j in range(20):
    tir.LetStmt(i=i_off+4)
```

i	tir::Var(i,int)
j	tir::Var(j,int)
A	tir.buffer([10][20], float, A)
B	tir.buffer([20], float, B)

Symbol table

OpenMP C/C++-to-TIR Translation

3. Loop body generation: Convert AST nodes to TIR operations based on translation rules

OpenMP C/C++ Code

```
#pragma omp autotune for
map(tofrom: A[0:10][0:20]…)
reduction(+: B[0:20])
for (int i = 4; i < 10; i++)
  for (int j = 0; j < 20; j++)
    B[j] += A[i][j] ;
```

- Autotune for directive
  - Captured body
    - Forstmt
      - Body: Forstmt
        - Body: BinaryOp(+=)
          - LHS: ArraySubscriptExpr (B, [j])
          - RHS: ArraySubscriptExpr (A, [i , j])

Clang AST

```
rhs=tir.load(A[i][j])
lhs=tir.load(B[j])
tir.store(B[j],tir.add(lhs, rhs))
```

TIR primitive representation

```
B[j]+=A[i][j]
```

Script-style TIR

Transformed TIR

```
for i_off in range(6):
  for j in range(20):
    tir.LetStmt(i=i_off+4)
    B[j]+=A[i][j]
```

i	tir::Var(i,int)
j	tir::Var(j,int)
A	tir.buffer([10][20], float, A)
B	tir.buffer([20], float, B)

Symbol table

OpenMP C/C++-to-TIR Translation

4. TIR block generation: Define block axes (spatial/reduce) and data access info using reduction clauses and AST analysis

OpenMP C/C++ Code

```
#pragma omp autotune for
map(tofrom: A[0:10][0:20]…)
reduction(+: B[0:20])
for (int i = 4; i < 10; i++)
  for (int j = 0; j < 20; j++)
    B[j] += A[i][j] ;
```

Autotune for directive
- Reduction clause: (+, B)
- Forstmt
  - Body: Forstmt
    - Body: BinaryOp(+=)
      - LHS: ArraySubscriptExpr (B, [j])
      - RHS: ArraySubscriptExpr (A, [i , j])

Clang AST

Reduction var = B

Transformed TIR

```
for i_off in range(6):
  for j in range(20):
    tir.LetStmt(i=i_off+4)
     B[j]+=A[i][j]
```

i	tir::Var(i,int)
j	tir::Var(j,int)
A	tir.buffer([10][20], float, A)
B	tir.buffer([20], float, B)

Symbol table

OpenMP C/C++-to-TIR Translation

4. TIR block generation: Define block axes (spatial/reduce) and data access info using reduction clauses and AST analysis

OpenMP C/C++ Code

```
#pragma omp autotune for
map(tofrom: A[0:10][0:20]…)
reduction(+: B[0:20])
for (int i = 4; i < 10; i++)
  for (int j = 0; j < 20; j++)
    B[j] += A[i][j] ;
```

Autotune for directive
Reduction clause: (+, B)
Forstmt
Body: Forstmt
Body: BinaryOp(+=)
LHS: ArraySubscriptExpr (B, [j])
RHS: ArraySubscriptExpr (A, [i , j])

Clang AST

Reduction var = B

LHS loop axis: j
RHS loop axes: i,j

Transformed TIR

```
for i_off in range(6):
  for j in range(20):
    tir.LetStmt(i=i_off+4)
     B[j]+=A[i][j]
```

i	tir::Var(i,int)
j	tir::Var(j,int)
A	tir.buffer([10][20], float, A)
B	tir.buffer([20], float, B)

Symbol table

OpenMP C/C++-to-TIR Translation

4. TIR block generation: Define block axes (spatial/reduce) and data access info using reduction clauses and AST analysis

OpenMP C/C++ Code

```
#pragma omp autotune for
map(tofrom: A[0:10][0:20]…)
reduction(+: B[0:20])
for (int i = 4; i < 10; i++)
  for (int j = 0; j < 20; j++)
    B[j] += A[i][j] ;
```

Clang AST

- Autotune for directive
  - Reduction clause: (+, B)
  - Forstmt
    - Body: Forstmt
      - Body: BinaryOp(+=)
        - LHS: ArraySubscriptExpr (B, [j])
        - RHS: ArraySubscriptExpr (A, [i , j])

Reduction var = B

LHS loop axis: j
RHS loop axes: i,j

Reduction axis = i
Spatial axis = j

Operation accumulates to buffer B on the 'j' axis, 'i' is identified as the reduction axis

Transformed TIR

```
for i_off in range(6):
  for j in range(20):
    tir.LetStmt(i=i_off+4)
    B[j]+=A[i][j]
```

i	tir::Var(i,int)
j	tir::Var(j,int)
A	tir.buffer([10][20], float, A)
B	tir.buffer([20], float, B)

Symbol table

OpenMP C/C++-to-TIR Translation

4. TIR block generation: Define block axes (spatial/reduce) and data access info using reduction clauses and AST analysis

OpenMP C/C++ Code

```
#pragma omp autotune for
map(tofrom: A[0:10][0:20]…)
reduction(+: B[0:20])
for (int i = 4; i < 10; i++)
  for (int j = 0; j < 20; j++)
    B[j] += A[i][j] ;
```

↓

- Autotune for directive
  - Reduction clause: (+, B)
  - Forstmt
    - Body: Forstmt
      - Body: BinaryOp(+=)
        - LHS: ArraySubscriptExpr (B, [j])
        - RHS: ArraySubscriptExpr (A, [i , j])

Clang AST

Reduction axis = i
Spatial axis = j

```
with T.block()
    i=T.axis.reduce(i)
    j=T.axis.spatial(j)
```

TIR block

Transformed TIR

```
for i_off in range(6):
  for j in range(20):
    tir.LetStmt(i=i_off+4)
     B[j]+=A[i][j]
```

i	tir::Var(i,int)
j	tir::Var(j,int)
A	tir.buffer([10][20], float, A)
B	tir.buffer([20], float, B)

Symbol table

OpenMP C/C++-to-TIR Translation

4. TIR block generation: Define block axes (spatial/reduce) and data access info using reduction clauses and AST analysis

OpenMP C/C++ Code

```
#pragma omp autotune for
map(tofrom: A[0:10][0:20]…)
reduction(+: B[0:20])
for (int i = 4; i < 10; i++)
  for (int j = 0; j < 20; j++)
    B[j] += A[i][j] ;
```

- Autotune for directive
  - Reduction clause: (+, B)
  - Forstmt
    - Body: Forstmt
      - Body: BinaryOp(+=)
        - LHS: ArraySubscriptExpr (B, [j])
        - RHS: ArraySubscriptExpr (A, [i , j])

Clang AST

Read buffer: ***A, B***
Write buffer: ***B***

```
with T.block()
    i=T.axis.reduce(i)
    j=T.axis.spatial(j)
```

TIR block

Transformed TIR

```
for i_off in range(6):
  for j in range(20):
    tir.LetStmt(i=i_off+4)
     B[j]+=A[i][j]
```

i	tir::Var(i,int)
j	tir::Var(j,int)
A	tir.buffer([10][20], float, A)
B	tir.buffer([20], float, B)

Symbol table

OpenMP C/C++-to-TIR Translation

4. TIR block generation: Define block axes (spatial/reduce) and data access info using reduction clauses and AST analysis

OpenMP C/C++ Code

```
#pragma omp autotune for
map(tofrom: A[0:10][0:20]…)
reduction(+: B[0:20])
for (int i = 4; i < 10; i++)
  for (int j = 0; j < 20; j++)
    B[j] += A[i][j] ;
```

Autotune for directive
- Reduction clause: (+, B)
- Forstmt
  - Body: Forstmt
    - Body: BinaryOp(+=)
      - LHS: ArraySubscriptExpr (B, [j])
      - RHS: ArraySubscriptExpr (A, [i , j])

Clang AST

Read buffer: ***A, B***
Write buffer: ***B***

```
with T.block()
  i=T.axis.reduce(i)
  j=T.axis.spatial(j)
  tir.reads(A, B)
  tir.writes(B)
```

TIR block

Transformed TIR

```
for i_off in range(6):
  for j in range(20):
    tir.LetStmt(i=i_off+4)
     B[j]+=A[i][j]
```

i	tir::Var(i,int)
j	tir::Var(j,int)
A	tir.buffer([10][20], float, A)
B	tir.buffer([20], float, B)

Symbol table

OpenMP C/C++-to-TIR Translation

4. TIR block generation: Define block axes (spatial/reduce) and data access info using reduction clauses and AST analysis

OpenMP C/C++ Code

```
#pragma omp autotune for
map(tofrom: A[0:10][0:20]…)
reduction(+: B[0:20])
for (int i = 4; i < 10; i++)
  for (int j = 0; j < 20; j++)
    B[j] += A[i][j] ;
```

- Autotune for directive
  - Reduction clause: (+, B)
  - Forstmt
    - Body: Forstmt
      - Body: BinaryOp(+=)
        - LHS: ArraySubscriptExpr (B, [j])
        - RHS: ArraySubscriptExpr (A, [i , j])

Clang AST

```
with T.block()
  i=T.axis.reduce(i)
  j=T.axis.spatial(j)
  tir.reads(A, B)
  tir.writes(B)
```

TIR block

Transformed TIR

```
for i_off in range(6):
  for j in range(20):
    tir.LetStmt(i=i_off+4)
    with T.block()
      i=T.axis.reduce(i)
      j=T.axis.spatial(j)
      tir.reads(A, B)
      tir.writes(B)
      B[j]+=A[i][j]
```

i	tir::Var(i,int)
j	tir::Var(j,int)
A	tir.buffer([10][20], float, A)
B	tir.buffer([20], float, B)

Symbol table

OpenMP C/C++-to-TIR Translation

5. TIR function generation: Build the final TIR function using the generated body and input variables

OpenMP C/C++ Code

```
#pragma omp autotune for
map(tofrom: A[0:10][0:20]…)
reduction(+: B[0:20])
for (int i = 4; i < 10; i++)
  for (int j = 0; j < 20; j++)
    B[j] += A[i][j] ;
```

Input buffers: A(10, 20), B(20))

```
def main0(A(10, 20), B(20))
```

TIR function signature

Transformed TIR

```
for i_off in range(6):
  for j in range(20):
    tir.LetStmt(i=i_off+4)
    with T.block()
      i=T.axis.reduce(i)
      j=T.axis.spatial(j)
      tir.reads(A, B)
      tir.writes(B)
      B[j]+=A[i][j]
```

i	tir::Var(i,int)
j	tir::Var(j,int)
A	tir.buffer([10][20], float, A)
B	tir.buffer([20], float, B)

Symbol table

OpenMP C/C++-to-TIR Translation

5. TIR function generation: Build the final TIR function using the generated body and input variables

OpenMP C/C++ Code

```
#pragma omp autotune for
map(tofrom: A[0:10][0:20]…)
reduction(+: B[0:20])
for (int i = 4; i < 10; i++)
  for (int j = 0; j < 20; j++)
    B[j] += A[i][j] ;
```

```
def main0(A(10, 20), B(20))
```

TIR function signature

Transformed TIR

```
def main0(A(10, 20), B(20))
  for i_off in range(6):
    for j in range(20):
      tir.LetStmt(i=i_off+4)
      with T.block()
        i=T.axis.reduce(i)
        j=T.axis.spatial(j)
        tir.reads(A, B)
        tir.writes(B)
        B[j]+=A[i][j]
```

Outline

Introduction & Motivation

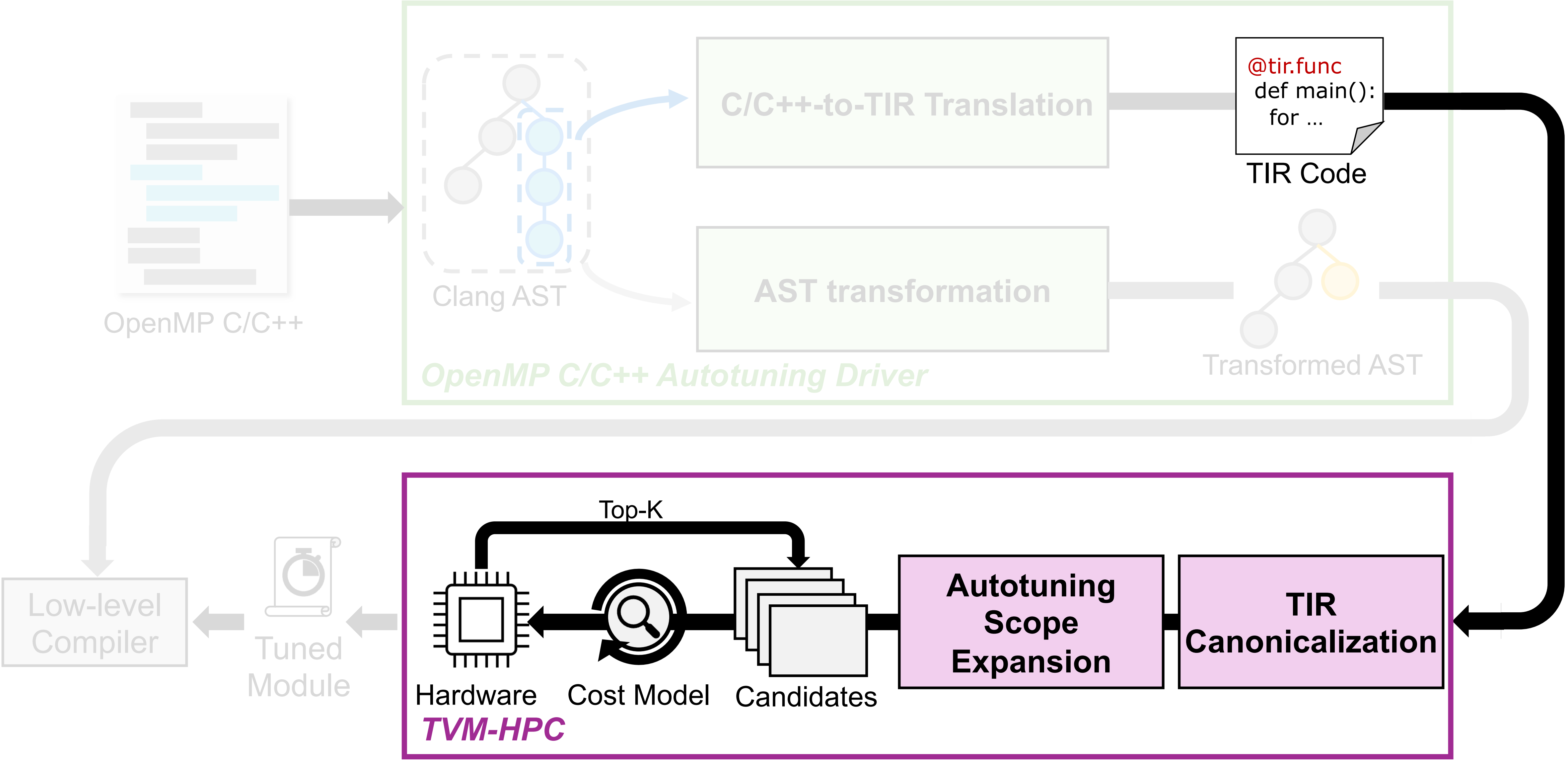
Background

HYPERF

- Overview
- OpenMP C/C++ Autotuning Driver
- **TVM-HPC**

Evaluation Results

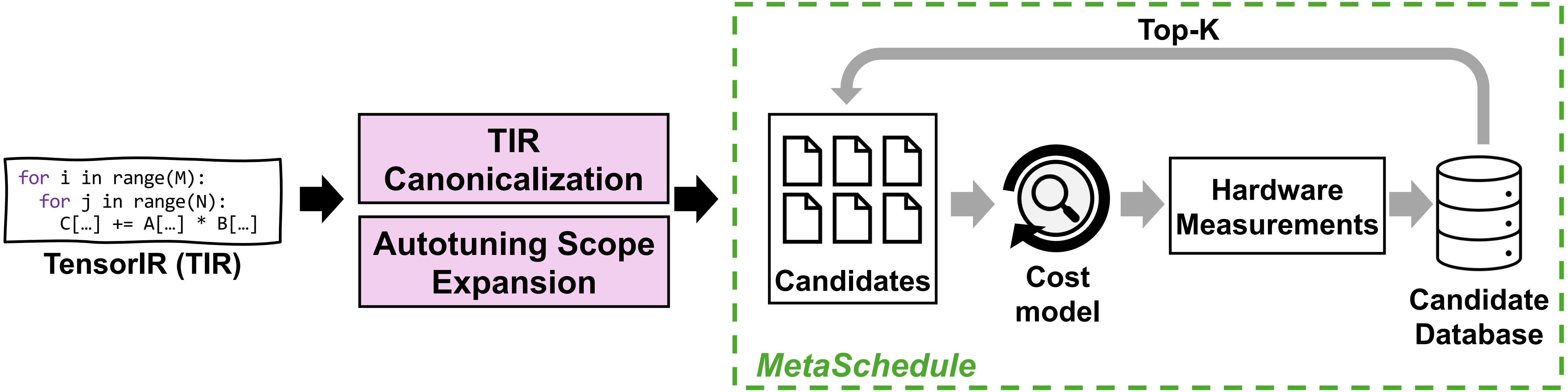
Overview



OpenMP C/C++
Clang AST
C/C++-to-TIR Translation
@tir.func
def main():
for ...
TIR Code
AST transformation
Transformed AST
OpenMP C/C++ Autotuning Driver
Low-level Compiler
Tuned Module
Top-K
Hardware
Cost Model
Candidates
Autotuning Scope Expansion
TIR Canonicalization
TVM-HPC

TVM-HPC

- TVM-HPC builds on TVM autotuning but extends it in two ways:
  - TIR canonicalization
  - Autotuning scope expansion



TIR Canonicalization

- TVM only triggers autotuning when the **TIR meets legality and profitability constraints**

→ TIR canonicalization passes ensure the TIR generated by the autotuning driver is **compatible with MetaSchedule autotuning**

TIR Canonicalization

Identify autotuning constraints and canonicalize the TIR

- **Only static loop extents are supported**

```
for i in range(100):
  for j in range(i):
    B[j]+=A[i][j]
```

TVM MetaSchedule

Rejected: j-loop extent is dynamic

TIR Canonicalization

Identify autotuning constraints and canonicalize the TIR

- **Only static loop extents are supported**

→ **Consolidate dynamic loop bounds into static extents**

```
for i in range(100):
  for j in range(i):
    B[j]+=A[i][j]
```

→

```
for i in range(100):
  for j in range(99):
    B[j]+=A[i][j]
```

Propagate outer bounds to give inner loops static max extents

TIR Canonicalization

Identify autotuning constraints and canonicalize the TIR

- **Only static loop extents are supported**

  → **Consolidate dynamic loop bounds into static extents**

```
for i in range(100):
  for j in range(i):
    B[j]+=A[i][j]
```

→

```
for i in range(100):
  for j in range(99):
    B[j]+=A[i][j]
```

→

```
for i in range(100):
  for j in range(99):
    tir.where(j < i)
    B[j]+=A[i][j]
```

Add `tir.where` to prevent inner loops from exceeding original extents

TIR Canonicalization

Identify autotuning constraints and canonicalize the TIR

- **Reduction blocks must be separate from other TIR blocks**

```
for i in range(100):
  for j in range(100):
    with T.block(“reduce”):
      vi=T.axis.reduce(i)
      Vj=T.axis.spatial(j)
      sum += A[vi][vj]
  with T.block(“spatial”):
    vi=T.axis.spatial(i)
    y[vi] = y[vi] * sum
```

TVM MetaSchedule

Rejected: Reduction and spatial blocks must be separate

TIR Canonicalization

Identify autotuning constraints and canonicalize the TIR

- **Reduction blocks must be separate from other TIR blocks**

→ Separate spatial and reduction loops

```
for i in range(100):
  for j in range(100):
    with T.block(“reduce”):
      vi=T.axis.reduce(i)
      Vj=T.axis.spatial(j)
      sum += A[vi][vj]
  with T.block(“spatial”):
    vi=T.axis.spatial(i)
    y[vi] = y[vi] * sum
```

→

```
for i in range(100):
  for j in range(100):
    with T.block(“reduce”):
      vi=T.axis.reduce(i)
      Vj=T.axis.spatial(j)
      sum += A[vi][vj]
--------------------------------
for i in range(100):
  with T.block(“spatial”):
    vi=T.axis.spatial(i)
    y[vi] = y[vi] * sum
```

Each block is split into a separate loop for independent execution and optimization

TIR Canonicalization

Identify autotuning constraints and canonicalize the TIR

- **Reduction blocks must be separate from other TIR blocks**

  → Separate spatial and reduction loops

```
for i in range(100):
  for j in range(100):
    with T.block(“reduce”):
      vi=T.axis.reduce(i)
      Vj=T.axis.spatial(j)
      sum += A[vi][vj]
  with T.block(“spatial”):
    vi=T.axis.spatial(i)
    y[vi] = y[vi] * sum
```

```
for i in range(100):
  for j in range(100):
    with T.block(“reduce”):
      vi=T.axis.reduce(i)
      Vj=T.axis.spatial(j)
      sum += A[vi][vj]
-------------------------------
for i in range(100):
  with T.block(“spatial”):
    vi=T.axis.spatial(i)
    y[vi] = y[vi] * sum
```

```
for i in range(100):
  for j in range(100):
    with T.block(“reduce”):
      vi=T.axis.reduce(i)
      Vj=T.axis.spatial(j)
      sum_expand[vi] += A[vi][vj]
-------------------------------
for i in range(100):
  with T.block(“spatial”):
    vi=T.axis.spatial(i)
    y[vi] = y[vi] * sum_expand[vi]
```

Expand reduction buffers to keep values after loop fission

TIR Canonicalization

Identify autotuning constraints and canonicalize the TIR

- **Buffer indices must be defined using loop variables**

```
for i in range(99):
  tmp = i + 1
  C[tmp] = A[tmp] * B[tmp]
```

TVM MetaSchedule

Rejected: Buffer indices not loop-dependent

TIR Canonicalization

Identify autotuning constraints and canonicalize the TIR

- **Buffer indices must be defined using loop variables**

  → Replace temporary variables in buffer indices

```
for i in range(99):
  tmp = i + 1
  C[tmp] = A[tmp] * B[tmp]
```

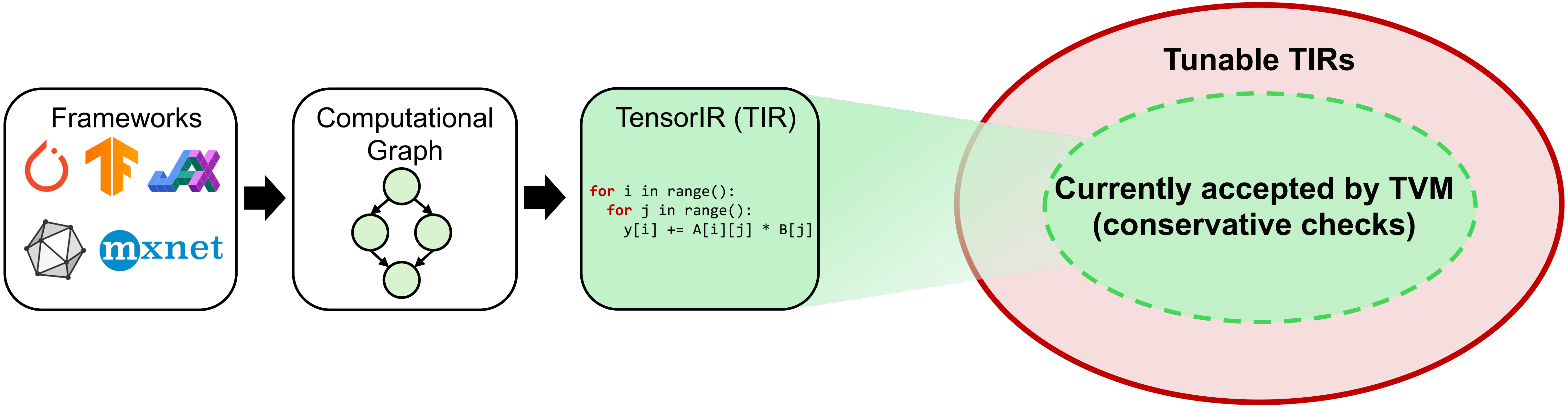
➡

```
for i in range(99):
  C[i + 1] = A[i + 1] * B[i + 1]
```

Rewrite temp vars to make buffer indices depend only on loop vars

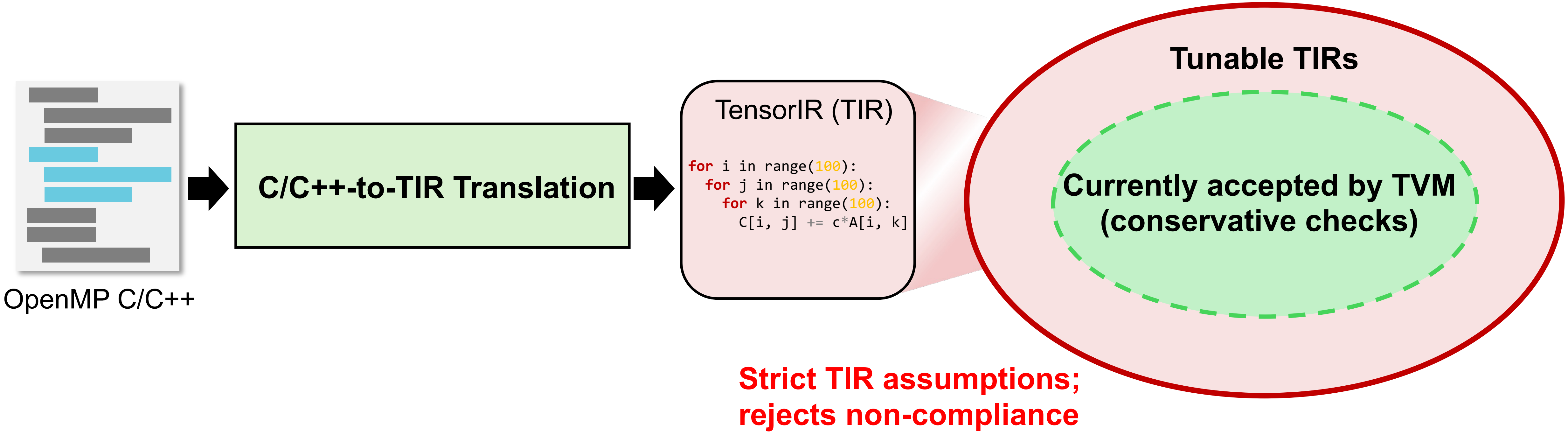
Autotuning Scope Expansion

- Original TVM autotuner assumes TIRs from TVM front-end
  - Adding **extra safety checks** before generating schedules to ensure correct optimization and stability



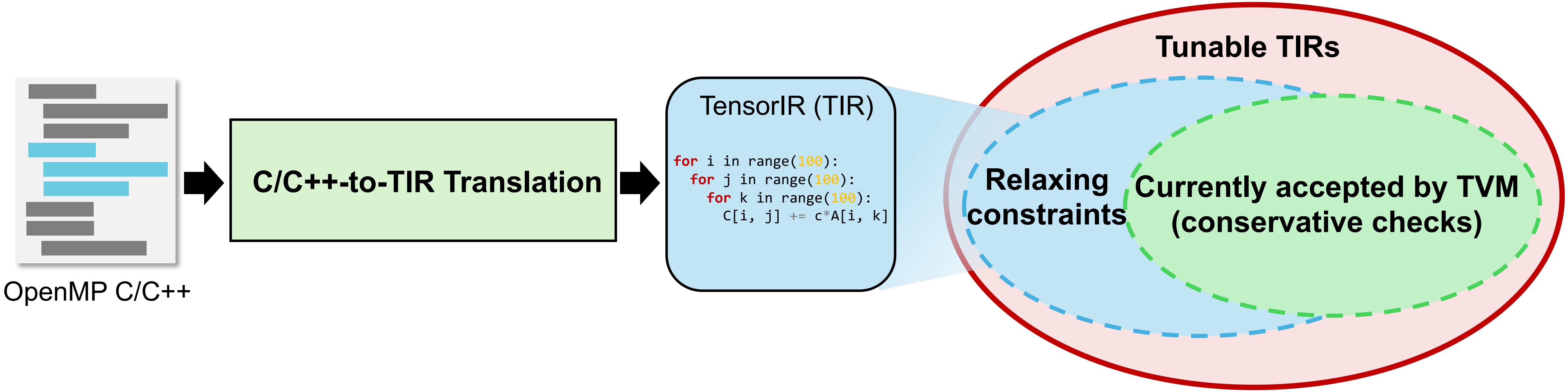
Autotuning Scope Expansion

- These checks may reject valid TIRs from HYPERF's front-end



Autotuning Scope Expansion

- These checks may reject valid TIRs from HYPERF's front-end
- **Relaxing constraints** enables more autotuning opportunities
  - Relax block dependency constraint
  - Relax constant buffer access range requirement (refer to the paper)



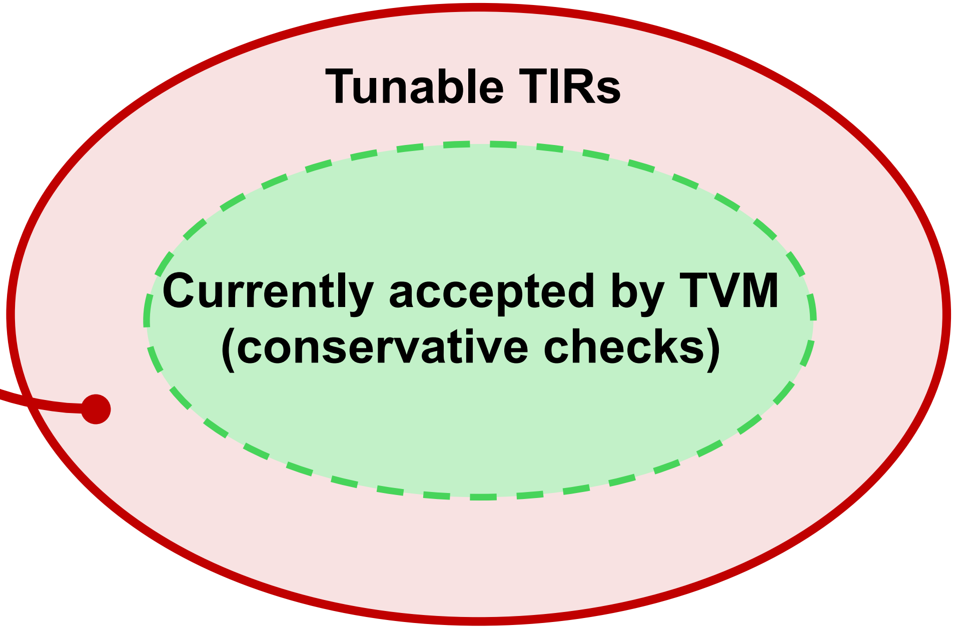
Autotuning Scope Expansion

- **Relax Block Dependency Constraint**
  - TVM front-end (e.g., DL graphs) ensures strict dependency constraints
  - Translated TIR may introduce WAR dependencies by reusing read buffers

Execution flow

```
for i in range(100):
  for j in range(100):
    with T.block("block 1"):
    sum[i, j] += A[i, j] * B[j]
for i in range(100):
  for j in range(100):
    with T.block("block 2"):
    A[i, j] = sum[i, j]
```

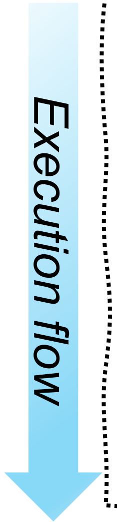
WAR dependency



Directly rejected due to WAR dependency, even though it is safe

Autotuning Scope Expansion

- **Relax Block Dependency Constraint**
  - TVM front-end (e.g., DL graphs) ensures strict dependency constraints
  - Translated TIR may introduce WAR dependencies by reusing read buffers



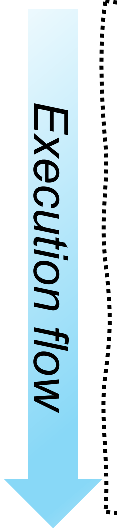

```
for i in range(100):
  for j in range(100):
    with T.block("block 1"):
    sum[i, j] += A[i, j] * B[j]
for i in range(100):
  for j in range(100):
    with T.block("block 2"):
    A[i, j] = sum[i, j]
```

Block 2 runs after Block 1 completes

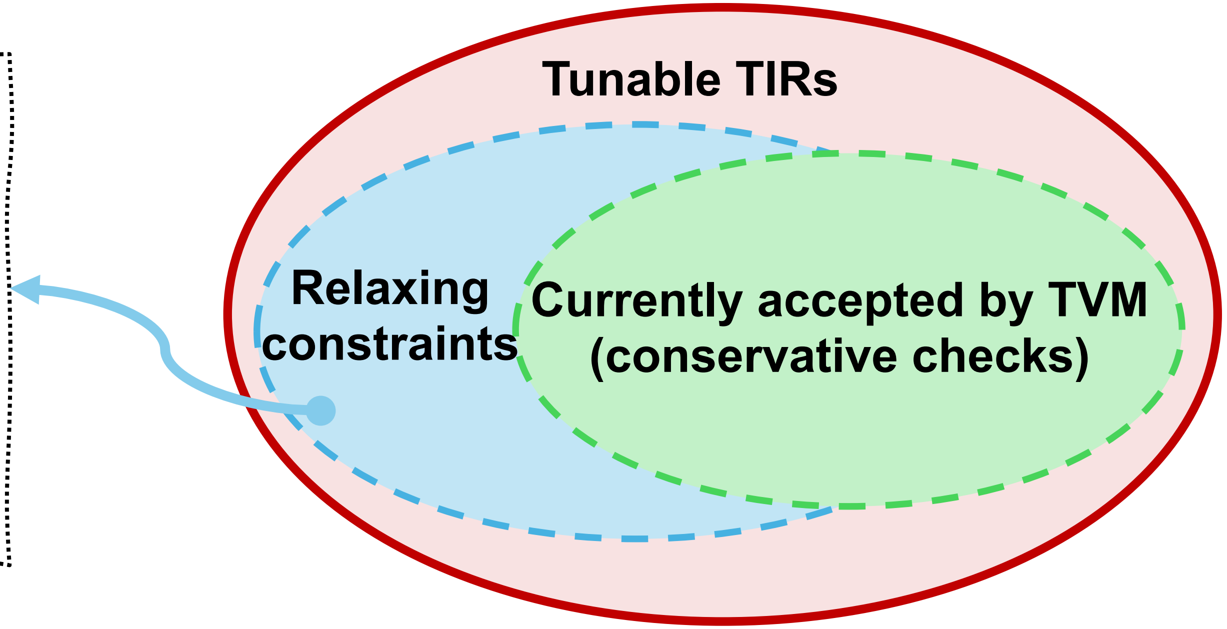
Autotuning Scope Expansion

- **Relax Block Dependency Constraint**
  - TVM front-end (e.g., DL graphs) ensures strict dependency constraints
  - Translated TIR may introduce WAR dependencies by reusing read buffers

→**Relax block dependency constraint** to enable tuning even with WAR dependencies




```
for i in range(100):
  for j in range(100):
    with T.block("block 1"):
    sum[i, j] += A[i, j] * B[j]
for i in range(100):
  for j in range(100):
    with T.block("block 2"):
    A[i, j] = sum[i, j]
```



Outline

Introduction & Motivation

Background

HYPERF

- Overview
- OpenMP C/C++ Autotuning Driver
- TVM-HPC

Evaluation Results

Methodology

Evaluated Benchmarks

- PolyBench: *gemm, 2mm, 3mm, syrk, syr2k, gesummv, mvt, atax, gemver, bicg, convolution-2d, convolution-3d, durbin*
  - *Input sizes: small, standard, and large, as defined by PolyBench*
- Rodinia: *hotspot, particlefilter, srad*

Experiment Environment

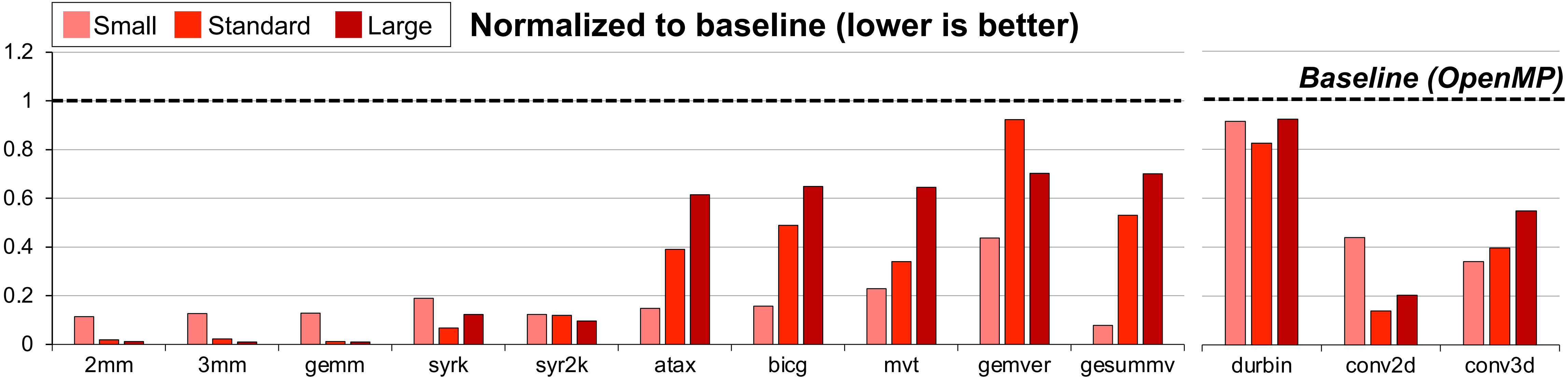
- Dual-socket Intel Xeon Gold 6426Y CPU with 512GB of DDR5-4800 memory

Experiment Configurations

Configuration	Optimization Method
Baseline	OpenMP versions
Pluto/Polly	Polyhedral compilers
ytopt/OpenTuner	Prior HPC autotuners (pragma-based)
HYPERF	Our proposed autotuning solution

PolyBench Results

HYPERF vs. Baseline

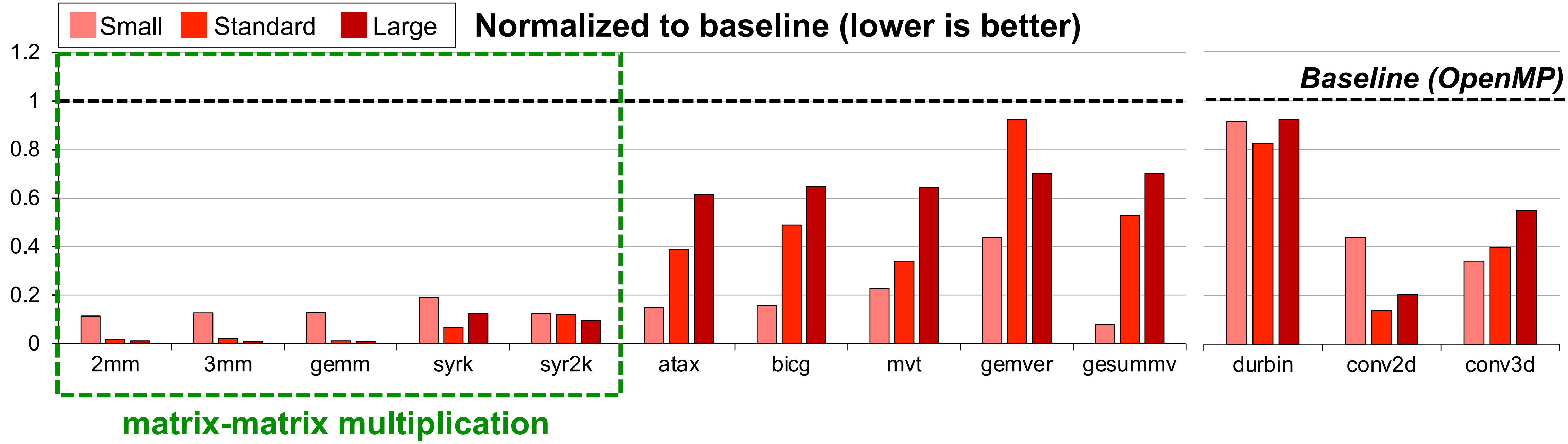


- **HYPERF** achieves speedups of up to **103.5× (5.5× on average)** over **baseline**

→ Driver effectively translates pragma C/C++ to TIR; TVM-HPC boosts performance via autotuning

PolyBench Results

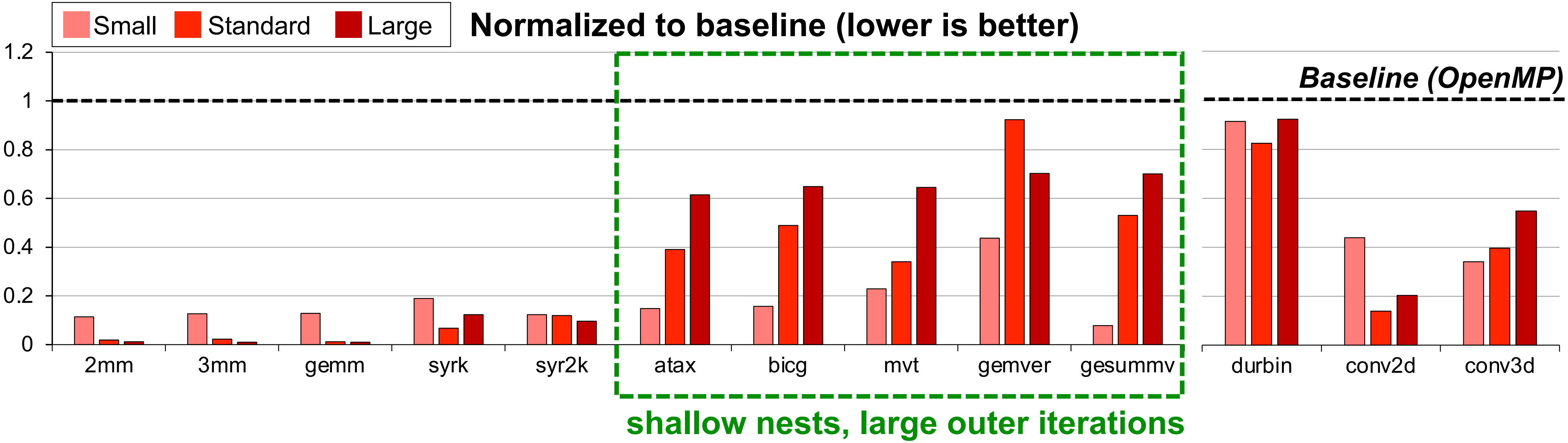
HYPERF vs. Baseline



- TVM autotuner **focuses on tensor loops**, like matrix-matrix multiplication, and tunes them using **tiling and unrolling**
- Larger inputs improve **data reuse** and **cache efficiency**, boosting performance

PolyBench Results

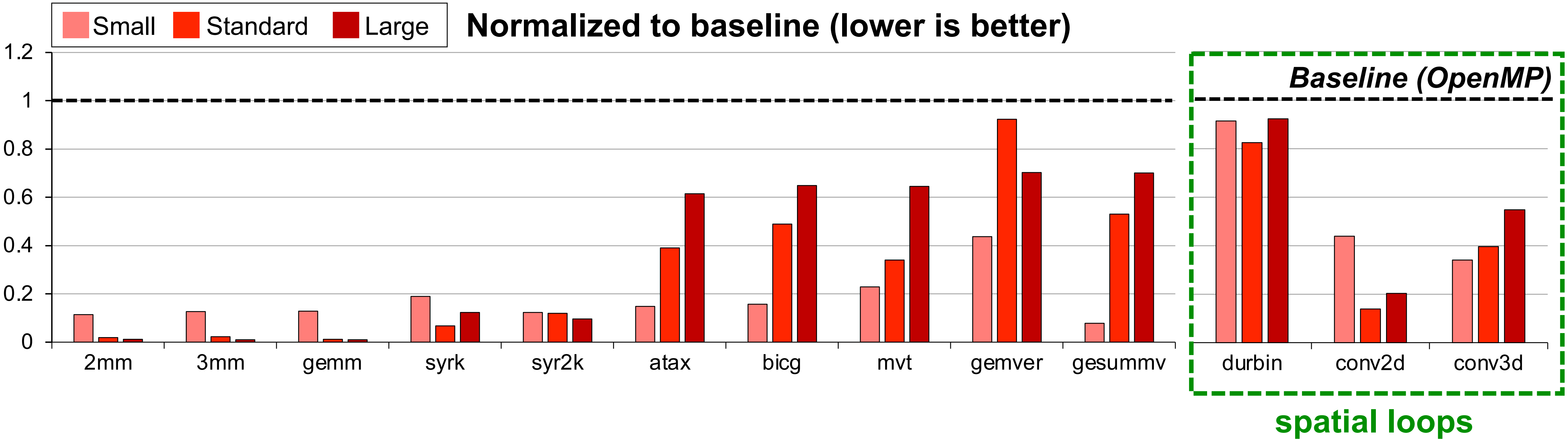
HYPERF vs. Baseline



- Performance gains mainly come from **parallelization and vectorization**
- These benchmarks involve matrix-vector computations with **data reuse only in vector accesses**

PolyBench Results

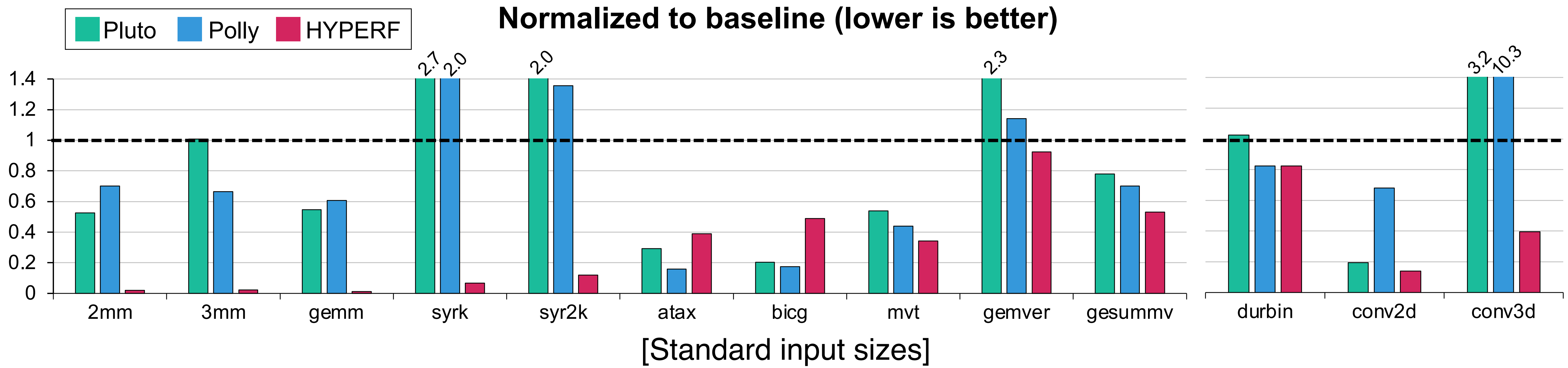
HYPERF vs. Baseline



- Since baseline already parallelizes and vectorizes well, extra gains were less visible
- For conv-2d, **HYPERF** uses **wider vector instructions** (e.g., ZMM), achieving up to **7.2× speedup** over baseline

PolyBench Results

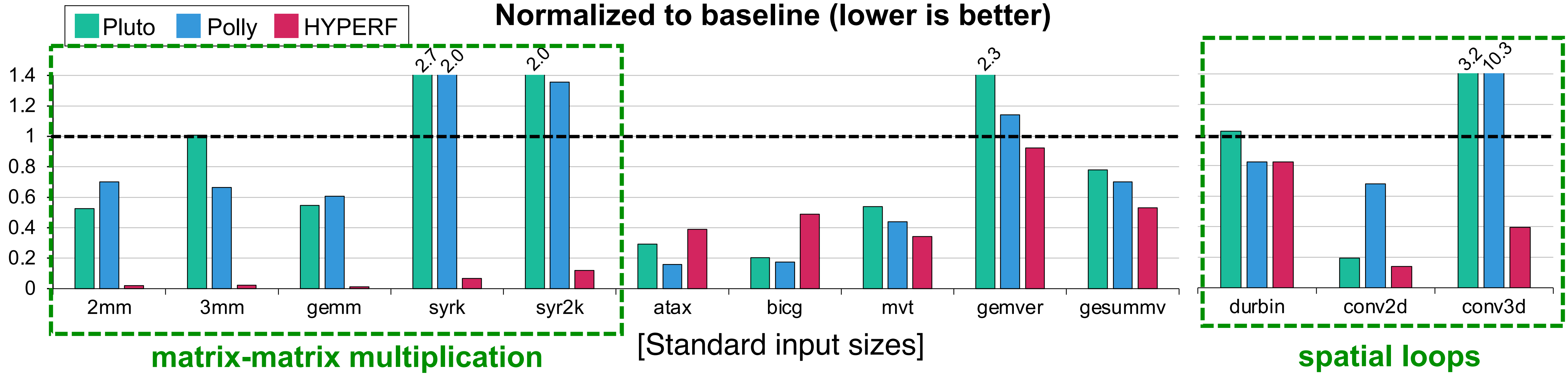
HYPERF vs. Pluto/Polly



- **HYPERF** outperforms **Pluto** and **Polly** by **4×** and **4.3×** on average (up to 49.4× and 54.8×), respectively

PolyBench Results

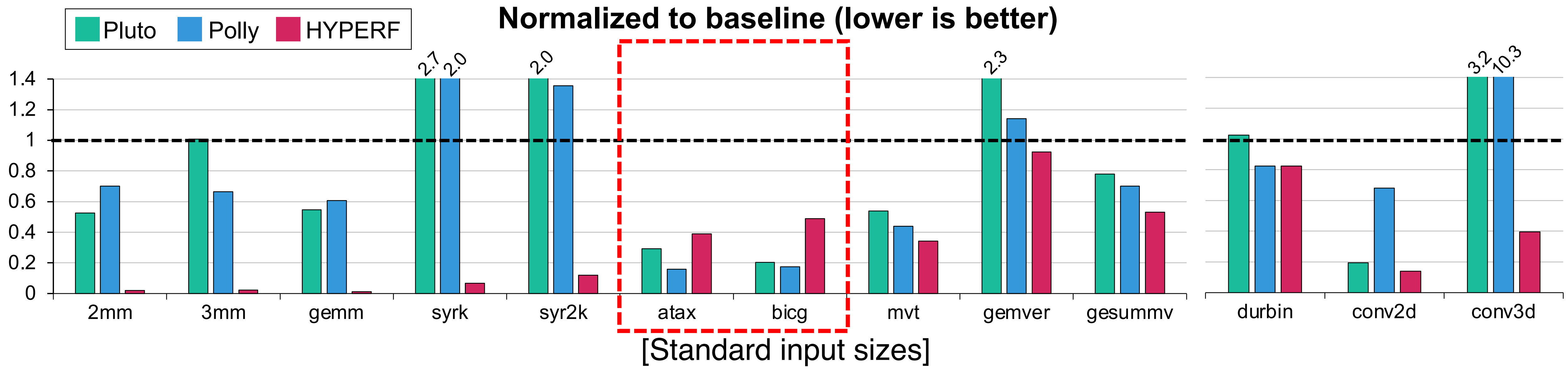
HYPERF vs. Pluto/Polly



- While Pluto and Polly also optimize loops, **HYPERF** is faster thanks to **empirical tuning, multi-level tiling, and loop collapsing**
- Pluto and Polly utilize only a **fixed tile size**, which limits reuse and parallelism

PolyBench Results

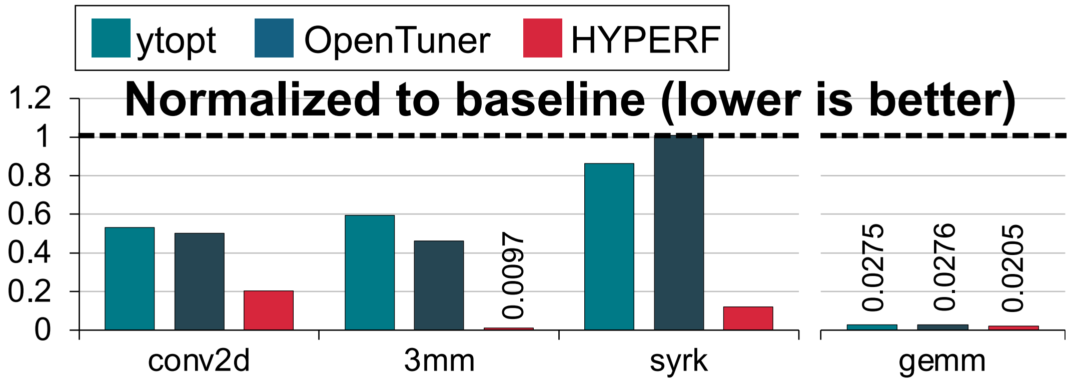
HYPERF vs. Pluto/Polly



- Pluto and Polly outperform HYPERF on certain benchmarks (e.g., atax, bicg) by applying **loop interchange to improve spatial locality**
- **HYPERF** (built on TVM) currently does not support loop interchange

PolyBench Results

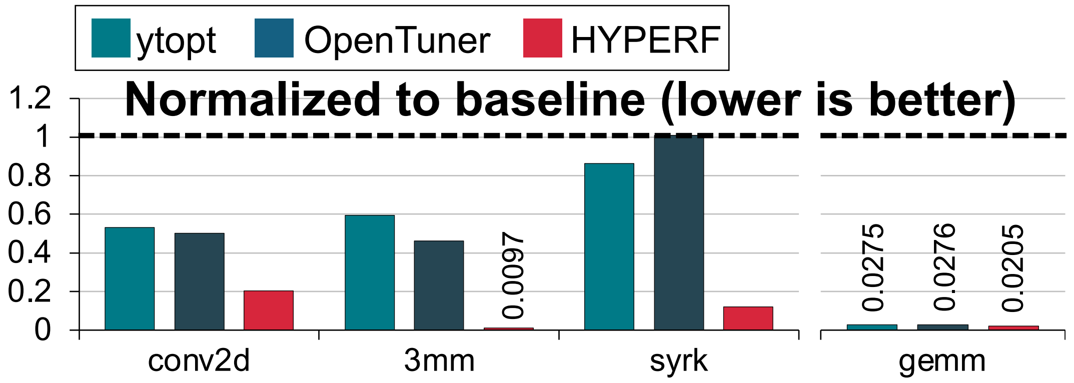
HYPERF vs. ytopt/OpenTuner

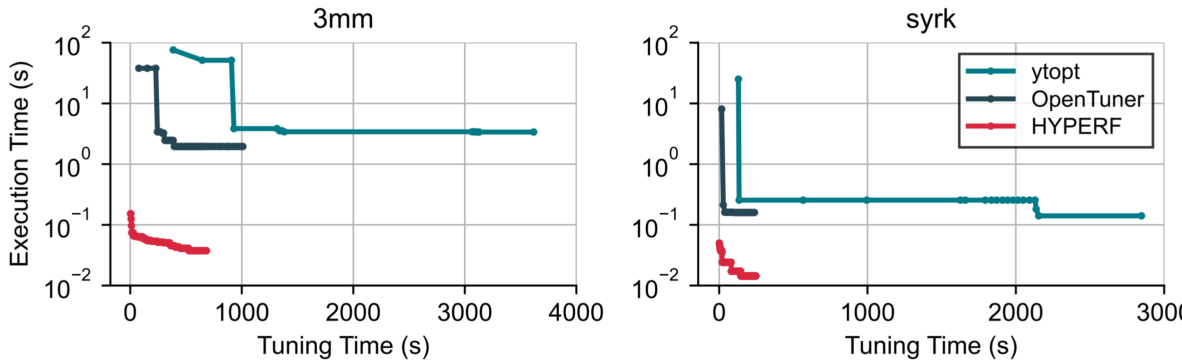


- **HYPERF** outperforms ytopt and OpenTuner by **6.2×** and **6×** on average, respectively

PolyBench Results

HYPERF vs. ytopt/OpenTuner





- **HYPERF** outperforms ytopt and OpenTuner by **6.2×** and **6×** on average, respectively
- **HYPERF** reduces autotuning time, converging **7.8× faster** than ytopt and **1.2× faster** than OpenTuner on average
  - Explores a wide search space with a learned cost model

PolyBench Results

Code snippets of the 3mm kernel

```
#pragma omp parallel for private(j, k)
for (i = 0; i < 2000; i++)
  for (j = 0; j < 2000; j++){
    C[i][j] = 0;
    for (k = 0; k < 2000; ++k)
      C[i][j] += A[i][k] * B[k][j];
  }
```

Baseline(OpenMP)

```
#pragma omp parallel for schedule(static,4)
private(j,k) num_threads(32)
for (i = 0; i < 2000; i++)
  for (j = 0; j < 2000; j++){
    C[i][j] = 0;
    for (k = 0; k < 2000; ++k)
      C[i][j] += A[i][k] * B[k][j];
  }
```

ytopt-optimized version

```
with T.block("root"):
  T.block_attr("parallel": 1024, "unroll": 512, "vectorize": 64)
  for i_0 in range(1):
   for j_0 in range(10):
    for i_1 in range(1000):
     for j_1 in range(2):
      for k_0 in range(400):
       for i_2 in range(1):
        for j_2 in range(100):
         for k_1 in range(5):
          for i_3 in range(2):
           for j_3 in range(1):
            with T.block("reduction_block: C"):
             i = T.axis.spatial(2000, i_0 % 10 * 200 + i_1 * 20 + i_2)
             j = T.axis.spatial(2000, j_0 // 10 * 80 + j_1 * 16 + j_2)
             k = T.axis.reduce(2000, k_0 * 50 + k_1)
             C[i, j] = C[i, j] + A[i, k] * B[k, j]
```

tiling, thread parallelization, vectorization, and loop unrolling

HYPERF-optimized version

- **HYPERF** generates **flexible and powerful candidates** using high-level TVM IR

Summary

HYPERF

- Bridges the abstraction gap by translating **OpenMP-style C/C++ to TIR**, integrating existing HPC codes into advanced autotuning
- Proposes **TIR canonicalization** and **autotuning scope expansion**, enabling highly flexible candidate generation and powerful optimization for complex HPC loops
- Proposes an **autotuning driver** that cleanly integrates outlined loops, ensuring seamless compilation

- Provides an **end-to-end HPC autotuning framework** combining familiar OpenMP-style programmability with robust, efficient schedule-based optimization
- **Achieves superior performance**, outperforming existing HPC autotuners and polyhedral compilers

Code available at: https://github.com/SNU-CODElab/HYPERF

HYPERF: End-to-End Autotuning Framework for High-Performance Computing

HPDC 2025

Juseong Park[*,2], Yongwon Shin[*,2], Junghyun Lee[1], Junseo Lee[1], Juyeon Kim[3], Oh-Kyoung Kwon[4], Hyojin Sung[1]

[1]Seoul National University (SNU)

[2]Pohang University of Science and Technology (POSTECH)

[3]Ewha Womans University

[4]Korea Institute of Science and Technology Information (KISTI)

Republic of Korea

[*]Equal contribution