WESTFÄLISCHE
WILHELMS-UNIVERSITÄT
MÜNSTER

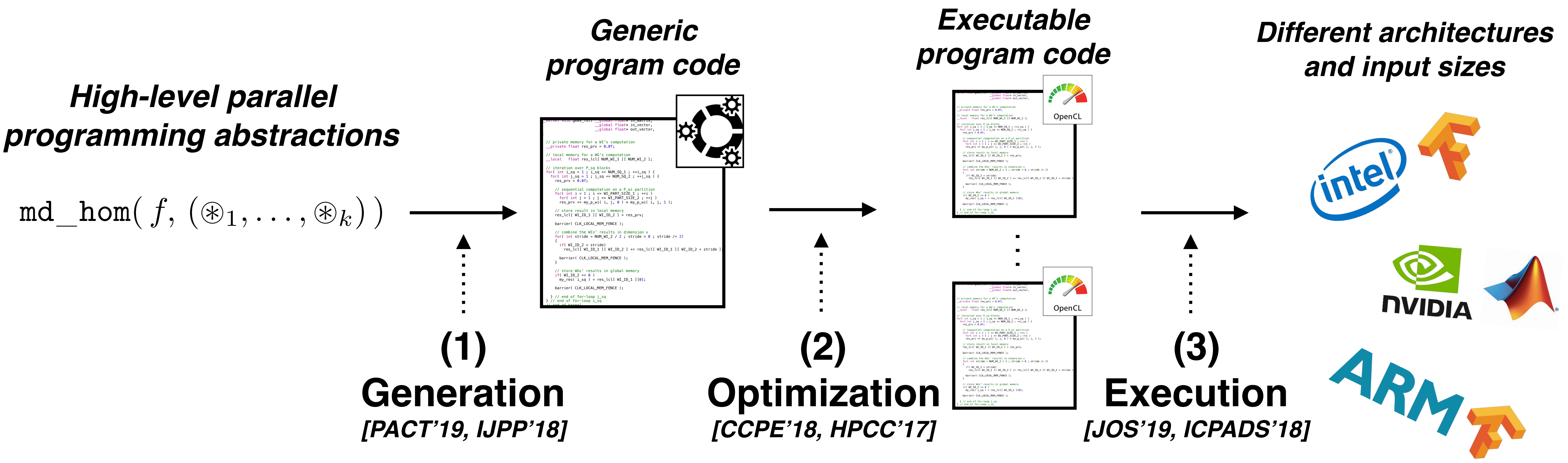
`md_poly`: A Performance-Portable Polyhedral Compiler based on Multi-Dimensional Homomorphisms

Ari Rasch, Richard Schulze, Sergei Gorlatch

University of Münster, Germany

Our Background

We are the developers of the MDH code generation approach:



High-level parallel programming abstractions

$\texttt{md\_hom}(\, f,\ (\circledast_1,\ldots,\circledast_k)\,)$

Generic program code

Executable program code

Different architectures and input sizes

(1) Generation *[PACT'19, IJPP'18]*

(2) Optimization *[CCPE'18, HPCC'17]*

(3) Execution *[JOS'19, ICPADS'18]*

- **Multi-Dimensional Homomorphisms (MDHs)** are a **formally defined** class of functions that **cover important data-parallel computations**, e.g.: linear algebra routines (BLAS), stencils computations, …
- We enable **conveniently** implementing MDHs by providing a **high-level DSL** for them.
- We provide a **DSL compiler** that **automatically generates OpenCL code** — the standard for uniformly programming different parallel architectures (e.g., CPU and GPU).
- Our OpenCL code is **fully automatically optimizable** (auto-tunable) — for each combination of a **target architecture**, and **input size** — by being generated as targeted to **OpenCL's abstract device models** and as **parametrized in these models' performance-critical parameters.**

Experimental Results

Stencils

CPU	Gaussian (2D)		Jacobi (3D)	
	RW	PC	RW	PC
Lift [2]	4.90	5.96	1.94	2.49
MKL-DNN	6.99	14.31	N/A	N/A

GPU	Gaussian (2D)		Jacobi (3D)	
	RW	PC	RW	PC
Lift [2]	2.33	1.09	1.14	1.02
cuDNN	3.78	19.11	N/A	N/A

[2] Hagedorn et. al, "High Performance Stencil Code Generation with LIFT.", **CGO'18 (Best Paper Award)**.

Data Mining

CPU	Probabilistic Record Linkage					
	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
EKR [5]	1.87	2.06	4.98	13.86	28.34	39.36

[5] Forchhammer et al. "Duplicate Detection on GPUs.", **HFSL'13**.

Our MDH approach achieves often better performance than well-performing competitors [1]

[1] Rasch, Schulze, Gorlatch. "Generating Portable High-Performance Code via Multi-Dimensional Homomorphisms.", PACT'19

Linear Algebra

CPU	GEMM		GEMV	
	RW	PC	RW	PC
Lift [1]	fails	3.04	1.51	1.99
MKL	4.22	0.74	1.05	0.87

GPU	GEMM		GEMV	
	RW	PC	RW	PC
Lift [1]	4.33	1.17	3.52	2.98
cuBLAS	2.91	0.83	1.03	1.00

[1] Steuwer et. al, "Lift: A Functional Data-Parallel IR for High-Performance GPU Code Generation", **CGO'17**.

Tensor Contractions

GPU	Tensor Contractions								
	RW 1	RW 2	RW 3	RW 4	RW 5	RW 6	RW 7	RW 8	RW 9
COGENT [3]	1.26	1.16	2.12	1.24	1.18	1.36	1.48	1.44	1.85
F-TC [4]	1.19	2.00	1.43	2.89	1.35	1.54	1.25	2.02	1.49

[3] Kim et. al. "A Code Generator for High-Performance Tensor Contractions on GPUs.", **CGO'19**.

[4] Vasilache et al. "The Next 700 Accelerated Layers: From Mathematical Expressions of Network Computation Graphs to Accelerated GPU Kernels, Automatically.", ***TACO,* 2019**.

Observation

Comparison: ***MDH Approach*** vs. ***Polyhedral Approaches*** **(e.g. PPCG)**

- ***Polyhedral approaches*** often provide better *productivity* → automatically parallelize sequential program code (rather than relying on a DSL).

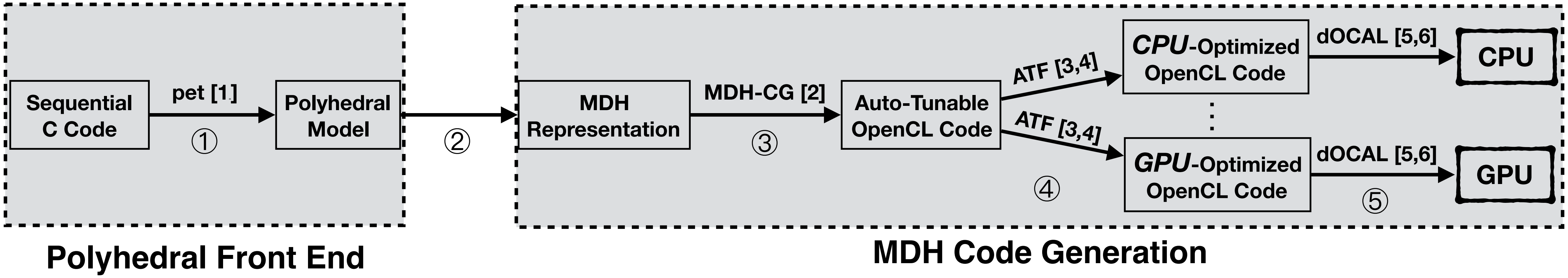
- ***The MDH approach*** achieves often higher *performance* than polyhedral compilers; its generated code is *portable* over different architectures (e.g., GPU and CPU).

Goal of this work:

Combining the advantages of both approaches

Idea

Using a polyhedral front end for the MDH code generator:

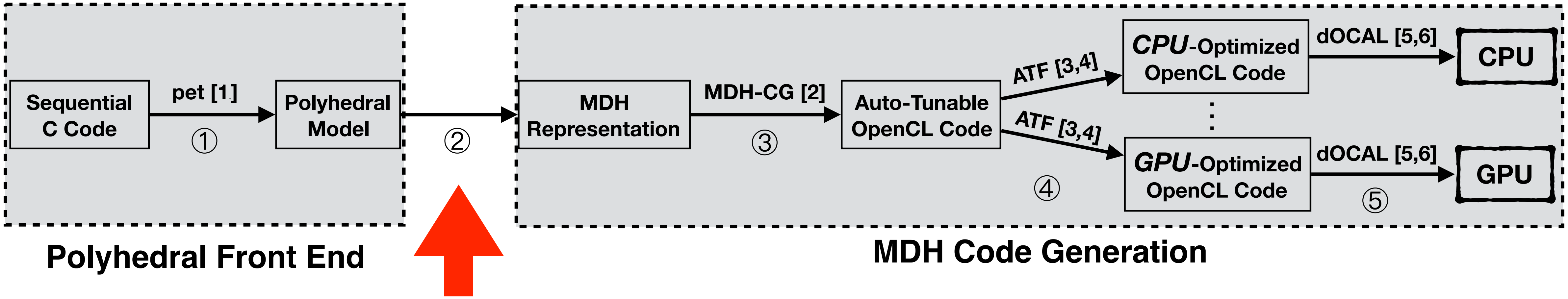


1. Transforming sequential C program to polyhedral model via PET.
2. Transforming polyhedral model to MDH representation.
3. Generating auto-tunable OpenCL code from MDH representation.
4. Auto-tuning OpenCL code for particular device and problem size.
5. Executing auto-tuned OpenCL code.

[1] Verdoolaege, Grosser, "Polyhedral Extraction Tool.", IMPACT’12
[2] Rasch, Schulze, Gorlatch, "Generating Portable High-Performance Code via Multi-Dimensional Homomorphisms.", PACT’19
[3] Rasch, Haidl, Gorlatch, "ATF: A Generic Auto-Tuning Framework.", HPCC’17
[4] Rasch, Gorlatch, "ATF: A Generic, Directive-Based Auto-Tuning Framework.", CCPE’19
[5] Rasch, Wrodarczyk, Schulze, Gorlatch, ”OCAL: An Abstraction for Host-Code Programming with OpenCL and CUDA.”, *ICPADS’18*
[6] Rasch, Bigge, Wrodarczyk, Schulze, Gorlatch. "dOCAL: high-level distributed programming with OpenCL and CUDA.", JOS’19

Idea

Using a polyhedral front end for the MDH code generator:



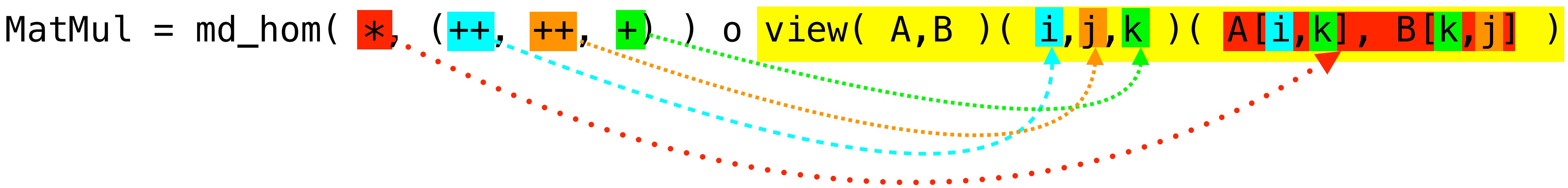
Sequential C Code
pet [1]
①
Polyhedral Model
Polyhedral Front End
②
MDH Representation
MDH-CG [2]
③
Auto-Tunable OpenCL Code
ATF [3,4]
CPU-Optimized OpenCL Code
ATF [3,4]
④
GPU-Optimized OpenCL Code
dOCAL [5,6]
CPU
dOCAL [5,6]
⑤
GPU
MDH Code Generation

1. Transforming sequential C program to polyhedral model via PET.
2. **Transforming polyhedral model to MDH representation.**
3. Generating auto-tunable OpenCL code from MDH representation.
4. Auto-tuning OpenCL code for particular device and problem size.
5. Executing auto-tuned OpenCL code.

[1] Verdoolaege, Grosser, "Polyhedral Extraction Tool.", IMPACT’12
[2] Rasch, Schulze, Gorlatch, "Generating Portable High-Performance Code via Multi-Dimensional Homomorphisms.", PACT’19
[3] Rasch, Haidl, Gorlatch, "ATF: A Generic Auto-Tuning Framework.", HPCC’17
[4] Rasch, Gorlatch, "ATF: A Generic, Directive-Based Auto-Tuning Framework.", CCPE’19
[5] Rasch, Wrodarczyk, Schulze, Gorlatch, ”OCAL: An Abstraction for Host-Code Programming with OpenCL and CUDA.”, *ICPADS’18*
[6] Rasch, Bigge, Wrodarczyk, Schulze, Gorlatch. "dOCAL: high-level distributed programming with OpenCL and CUDA.", JOS’19

The MDH DSL

Example: *Matrix Multiplication*



```
MatMul = md_hom( *, (++, ++, +) ) o view( A,B )( i,j,k )( A[i,k], B[k,j] )
```

```
for( int i = 0; i < M ; ++i )
  for( int j = 0; i < N ; ++j )
    for( int k = 0; i < K ; ++k )
      C[i][j] += A[i][k] * B[k][j];
```

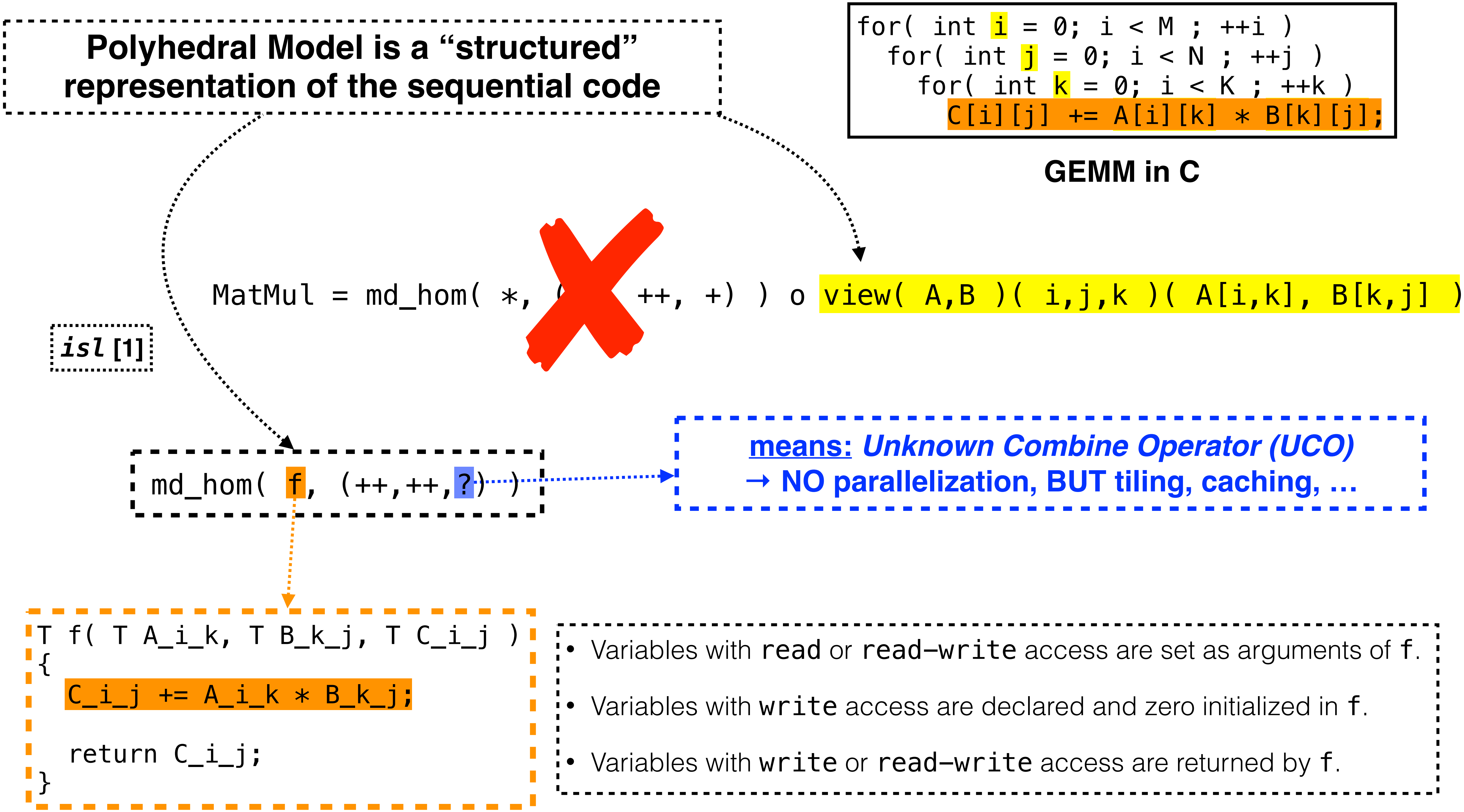
GEMM in C

What's happening?

1. Prepare the domain-specific input uniformly for `md_hom`; for this, our DSL provides pattern `view`.
   - here: fuse matrices A and B to 3-dimensional array of pairs consisting of the elements in A and B to multiply: `i,j,k ↦ (A[i,k],B[k,j])`.
2. Apply multiplication (denoted as `*`) to each pair.
3. Combine results in dimension `k` by addition (`+`).
4. Combine results in dimensions `i` and `j` by concatenation (`++`).

Transformation

Polyhedral Model → MDH Representation:



Polyhedral Model is a "structured" representation of the sequential code

```
for( int i = 0; i < M ; ++i )
  for( int j = 0; i < N ; ++j )
    for( int k = 0; i < K ; ++k )
      C[i][j] += A[i][k] * B[k][j];
```

GEMM in C

```
MatMul = md_hom( *, (      ++, +) ) o view( A,B )( i,j,k )( A[i,k], B[k,j] )
```

***isl* [1]**

```
md_hom( f, (++,++,?) )
```

<u>means:</u> *Unknown Combine Operator (UCO)*
→ NO parallelization, BUT tiling, caching, …

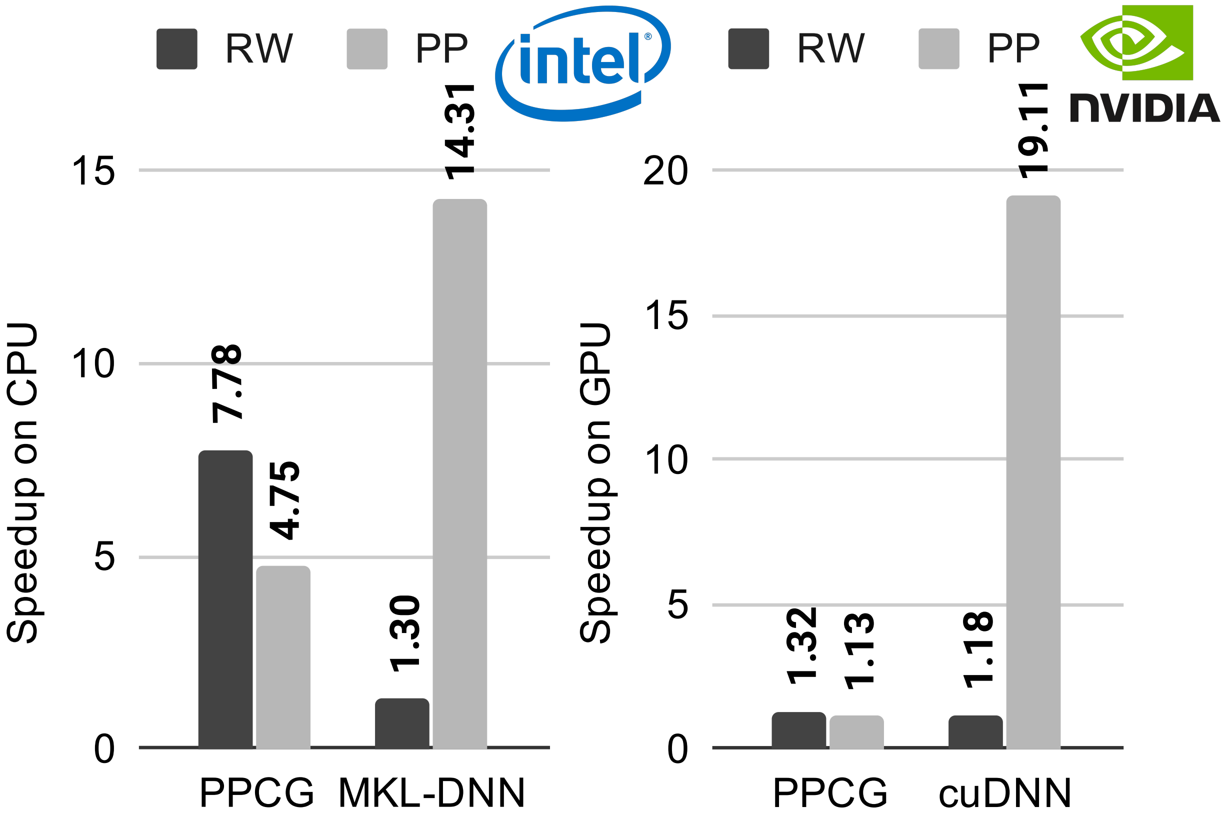
```
T f( T A_i_k, T B_k_j, T C_i_j )
{
  C_i_j += A_i_k * B_k_j;

  return C_i_j;
}
```

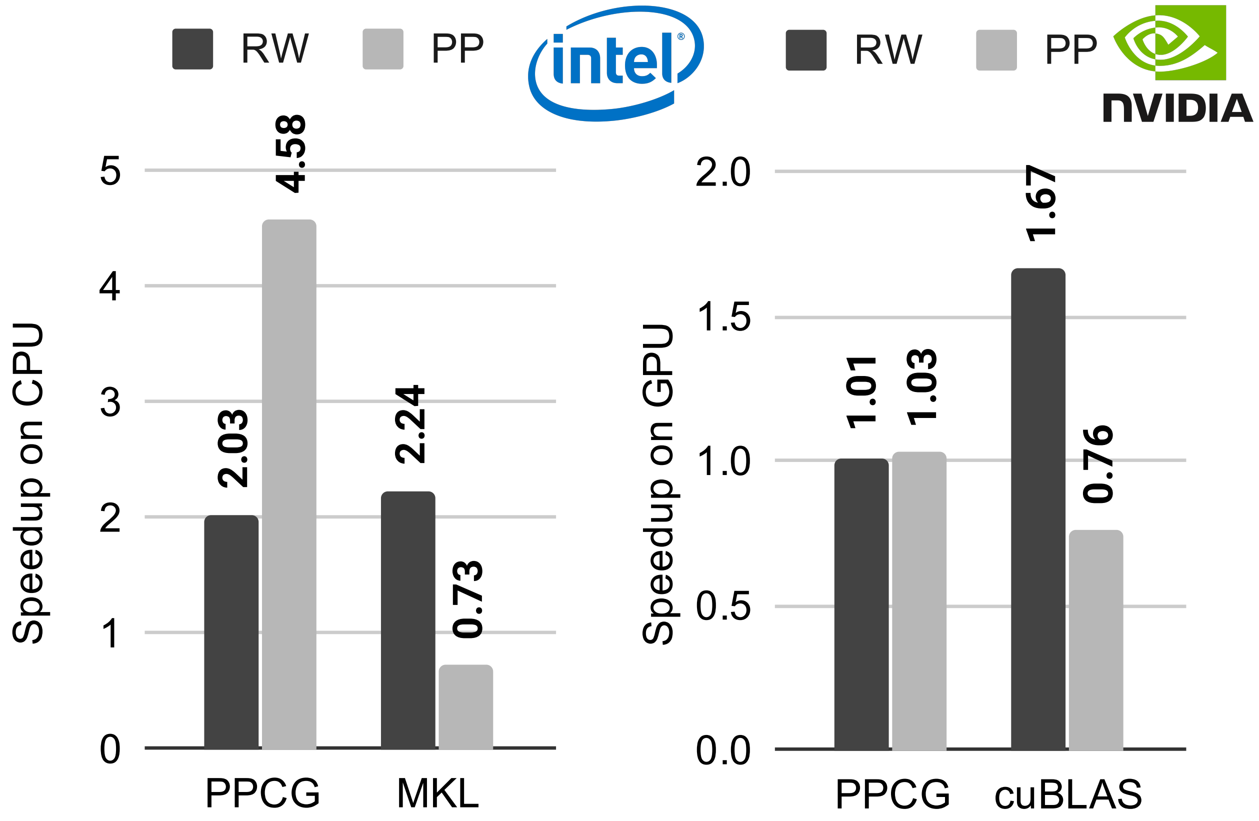
- Variables with `read` or `read-write` access are set as arguments of `f`.
- Variables with `write` access are declared and zero initialized in `f`.
- Variables with `write` or `read-write` access are returned by `f`.

[1] Verdoolaege, "isl: An Integer Set Library for the Polyhedral Model", *ICMS'10*

Experimental Results



Gaussian Convolution



Matrix Multiplication

Hardware

- CPU: Intel Xeon E5
- GPU: NVIDIA V100

Gaussian Convolution

- RW: 1×512×7×7×512
- PP: 1x1x4096x4096x1

Matrix Multiplication

- RW: M,N,K = 10,500,64
- PP: M,N,K = 1024

- **Compared to PPCG:**
  - Competitive performance on GPU: `1.01x – 1.32x`
  - Better performance on CPU: `2.03x – 7.78x`
- **Compared to Intel MKL/MKL-DNN & NVIDIA cuBLAS/cuDNN:**
  - Competitive and sometimes better performance: `0.73x – 2.24x (19.11x)`

Conclusion

We present `md_poly`:

- `md_poly` is based on both the polyhedral model and the MDH code generation approach;
- `md_poly` combines productivity (as in polyhedral compilers) and portable high performance (as in the MDH approach);
- `md_poly` achieves sometimes better performance than hand-optimized approaches.

Future Work: Analyze and Evaluate `md_poly` for all applications in PolyBench.

We are looking for a polyhedral expert as collaboration partner!

Reviewer Questions

Q: *Unclear whether all polyhedral programs can be converted to MDH?*

```
__kernel void foo( __global int* a )
{
  *a = 42;
}
```

“programs without loops (e.g., "a = 42;”)"

Reviewer Questions

Q: *Unclear whether all polyhedral programs can be converted to MDH?*

```
for( int i = 1; i < K ; ++i )
{
  A[ N-i ] = A[ i ];
}
```

N ≥ 2K → *parallelizable*
else → *NOT parallelizable*

isl ?

“programs with parametric dependence distance (e.g., A[N-i] = A[i])"

Reviewer Questions

Q: *Unclear whether all polyhedral programs can be converted to MDH?*

Sequential

```
for( int t = 1; t < N ; ++t )
{
  if (t % 2 == 0)
  {
    // ...
  }
}
```

Parallel

```
__kernel void foo( ... )
{
  int t = get_global_id(0);

  if (t % 2 == 0)
  {
    // ...
  }

}
```

OpenCL

isl ?

“if-conditionals using modulo arithmetic
(e.g., if (t % 2 == 0) where t is a surrounding loop iterator)”

Reviewer Questions

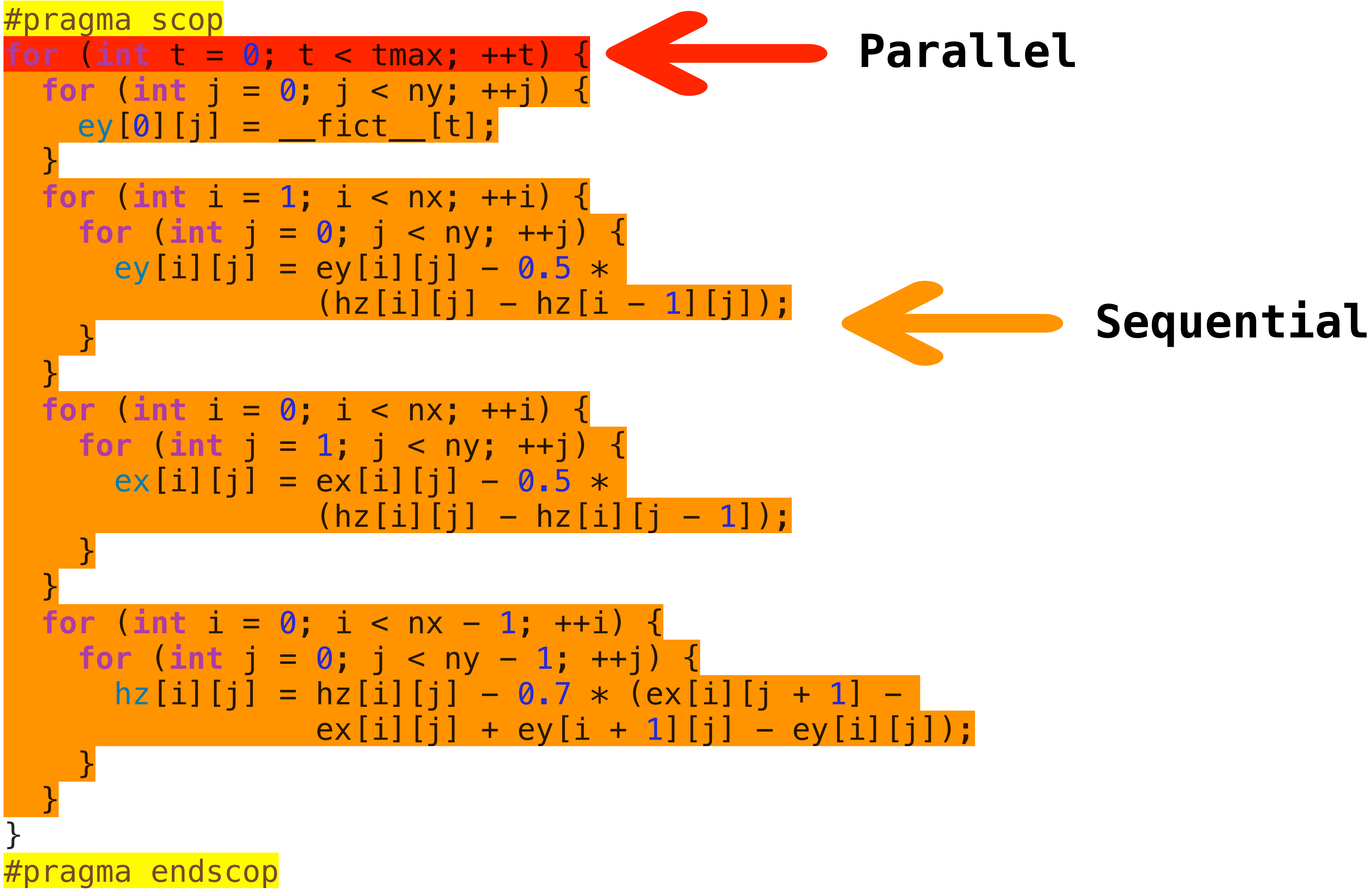
Q: *Unclear whether all polyhedral programs can be converted to MDH?*

```
#pragma scop
for (int t = 0; t < tmax; ++t) {
  for (int j = 0; j < ny; ++j) {
    ey[0][j] = __fict__[t];
  }
  for (int i = 1; i < nx; ++i) {
    for (int j = 0; j < ny; ++j) {
      ey[i][j] = ey[i][j] - 0.5 *
                 (hz[i][j] - hz[i - 1][j]);
    }
  }
  for (int i = 0; i < nx; ++i) {
    for (int j = 1; j < ny; ++j) {
      ex[i][j] = ex[i][j] - 0.5 *
                 (hz[i][j] - hz[i][j - 1]);
    }
  }
  for (int i = 0; i < nx - 1; ++i) {
    for (int j = 0; j < ny - 1; ++j) {
      hz[i][j] = hz[i][j] - 0.7 * (ex[i][j + 1] -
                 ex[i][j] + ey[i + 1][j] - ey[i][j]);
    }
  }
}
#pragma endscop
```

“imperfectly nested loops (e.g., FDTD-2D in polybench)”

Reviewer Questions

Q: *Unclear whether all polyhedral programs can be converted to MDH?*



```
#pragma scop
for (int t = 0; t < tmax; ++t) {
  for (int j = 0; j < ny; ++j) {
    ey[0][j] = __fict__[t];
  }
  for (int i = 1; i < nx; ++i) {
    for (int j = 0; j < ny; ++j) {
      ey[i][j] = ey[i][j] - 0.5 *
                 (hz[i][j] - hz[i - 1][j]);
    }
  }
  for (int i = 0; i < nx; ++i) {
    for (int j = 1; j < ny; ++j) {
      ex[i][j] = ex[i][j] - 0.5 *
                 (hz[i][j] - hz[i][j - 1]);
    }
  }
  for (int i = 0; i < nx - 1; ++i) {
    for (int j = 0; j < ny - 1; ++j) {
      hz[i][j] = hz[i][j] - 0.7 * (ex[i][j + 1] -
                 ex[i][j] + ey[i + 1][j] - ey[i][j]);
    }
  }
}
#pragma endscop
```

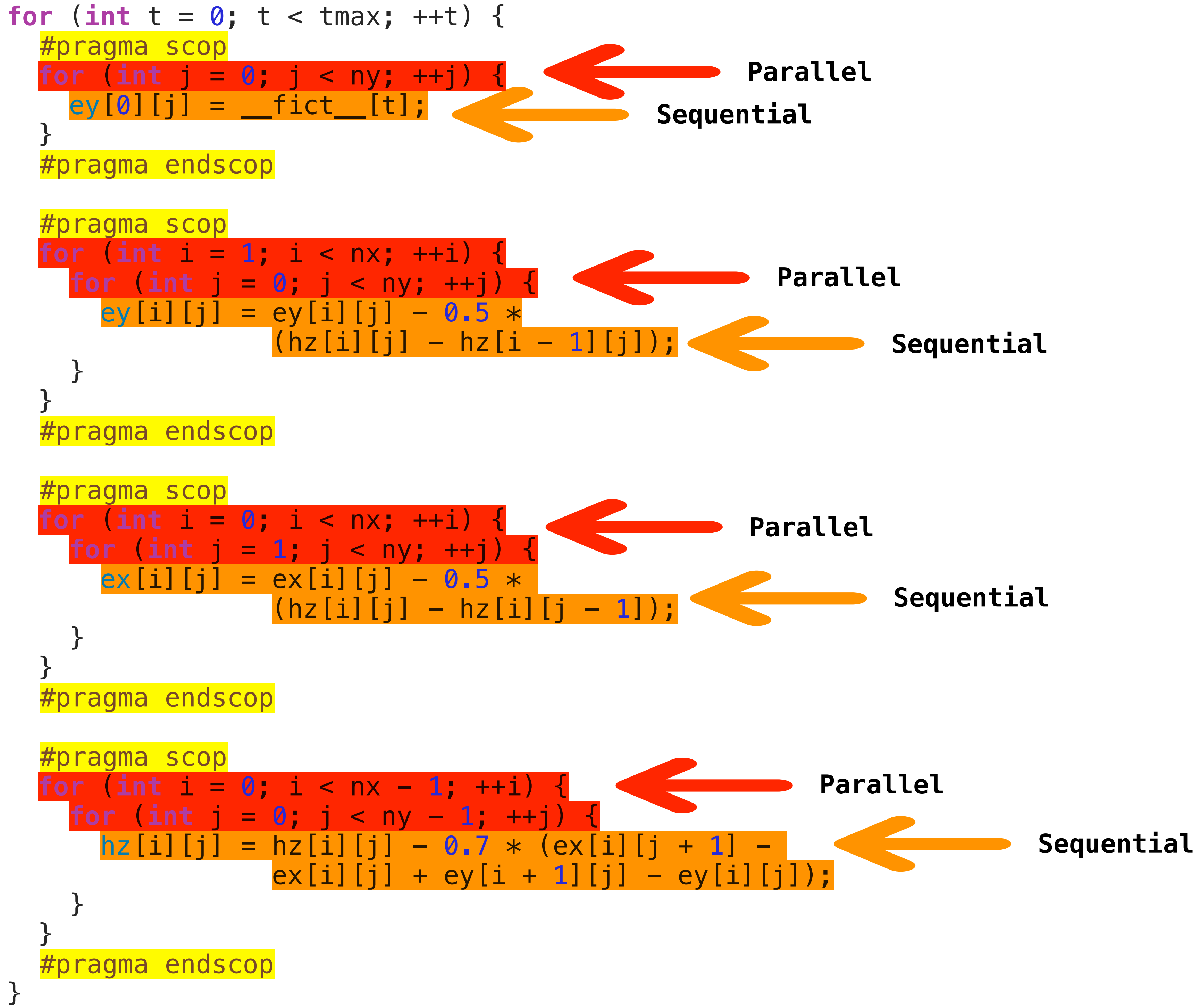
Parallel

Sequential

“imperfectly nested loops (e.g., FDTD-2D in polybench)”

Reviewer Questions

Q: *Unclear whether all polyhedral programs can be converted to MDH?*



```
for (int t = 0; t < tmax; ++t) {
  #pragma scop
  for (int j = 0; j < ny; ++j) {
    ey[0][j] = __fict__[t];
  }
  #pragma endscop

  #pragma scop
  for (int i = 1; i < nx; ++i) {
    for (int j = 0; j < ny; ++j) {
      ey[i][j] = ey[i][j] − 0.5 ∗
                 (hz[i][j] − hz[i − 1][j]);
    }
  }
  #pragma endscop

  #pragma scop
  for (int i = 0; i < nx; ++i) {
    for (int j = 1; j < ny; ++j) {
      ex[i][j] = ex[i][j] − 0.5 ∗
                 (hz[i][j] − hz[i][j − 1]);
    }
  }
  #pragma endscop

  #pragma scop
  for (int i = 0; i < nx − 1; ++i) {
    for (int j = 0; j < ny − 1; ++j) {
      hz[i][j] = hz[i][j] − 0.7 ∗ (ex[i][j + 1] −
                 ex[i][j] + ey[i + 1][j] − ey[i][j]);
    }
  }
  #pragma endscop
}
```

“imperfectly nested loops (e.g., FDTD-2D in polybench)”

Reviewer Questions

Q: *Your claim that combine operators other than concatenation cannot be extracted looks way too strong.*

$\texttt{PRL = md\_hom(weight, (++, }\otimes_{\texttt{max}}\texttt{)) o view(…)}$

```
for (int i = 0; i < NUM_NEW_RECORDS; ++i) {
    match_id[i] = 0;
    match_weight[i] = 0;
    id_measure[i] = 0;
    for (int j = 0; j < NUM_EXISTING_RECORDS; ++j)
{
        // calculate weight
        double weight = calc_weight(...);
        // calculate identity measure
        int id_measure = calc_id_measure(...);
        // store result
        if ((weight >= 15.0 || id_measure == 14) &&
(weight > *match_weight_res)) {
            match_id[i] = i_id[j];
            match_weight[i] = weight;
            id_measure[i] = id_measure;
        }
    }
}
```

Automatically extractable?

Rasch, Schulze, Gorus, Hiller, Bartholomäus, Gorlatch. "High-Performance Probabilistic Record Linkage via Multi-Dimensional Homomorphisms.", SAC'19