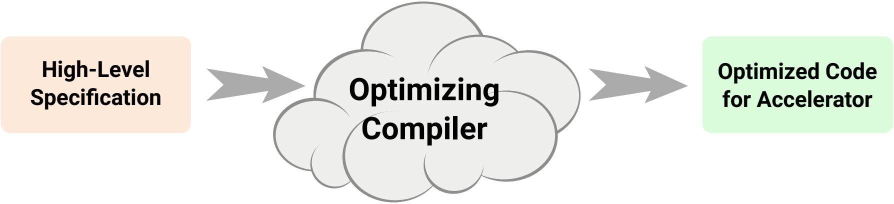
TC-CIM: Empowering Tensor Comprehensions for Computing In Memory

Andi Drebes[1] Lorenzo Chelini[2,3] Oleksandr Zinenko[4]
Albert Cohen[4] Henk Corporaal[2] Tobias Grosser[5]
Kanishkan Vadivel[2] Nicolas Vasilache[4]

[1]Inria and École Normale Supérieure [2]TU Eindhoven [3]IBM Research Zurich
[4]Google [5]ETH Zurich

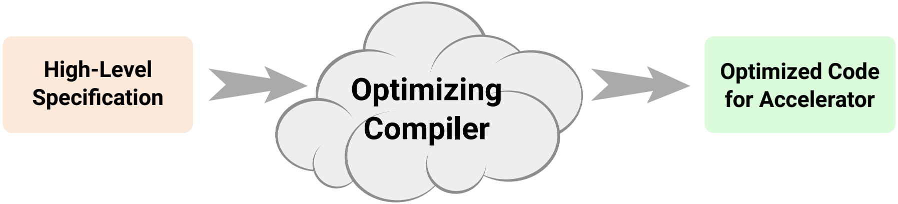
01/22/2020

Detecting Operations for Accelerators



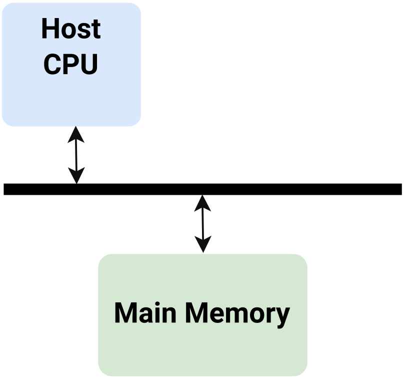
- Goal: Reliably detect operations for efficient offloading

Detecting Operations for Accelerators

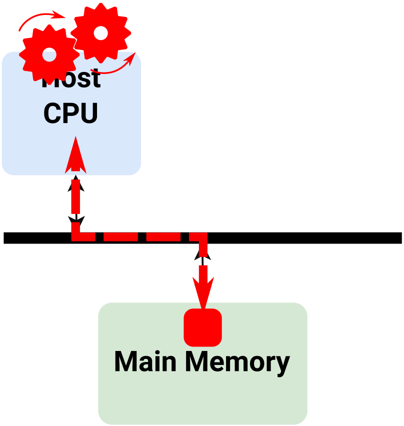


- Goal: Reliably detect operations for efficient offloading
- At which stage?
- On which representation?
- Create reusable infrastructure

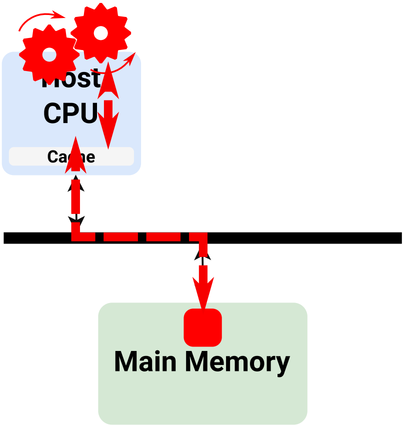
Von-Neumann Bottleneck



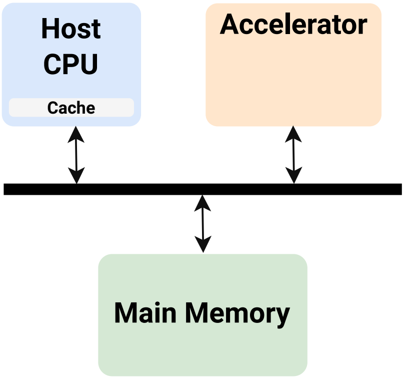
Von-Neumann Bottleneck



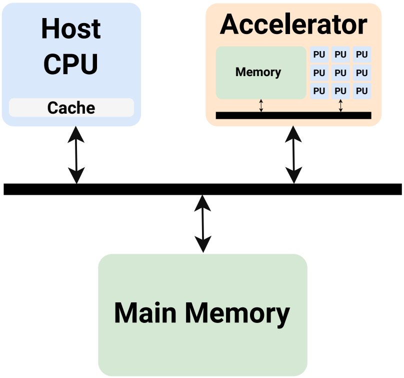
Von-Neumann Bottleneck



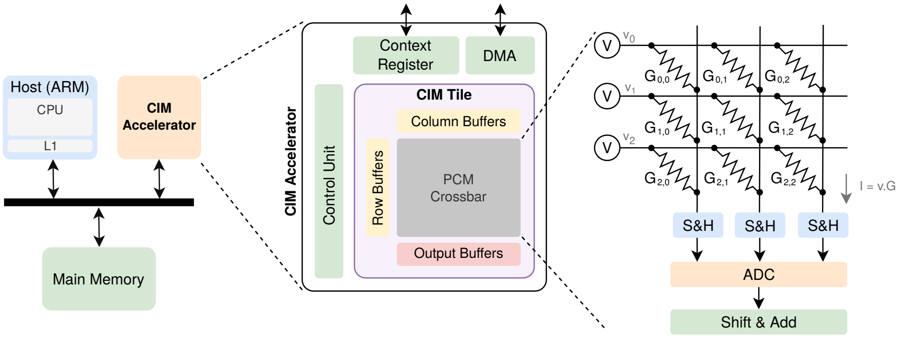
Von-Neumann Bottleneck



Von-Neumann Bottleneck

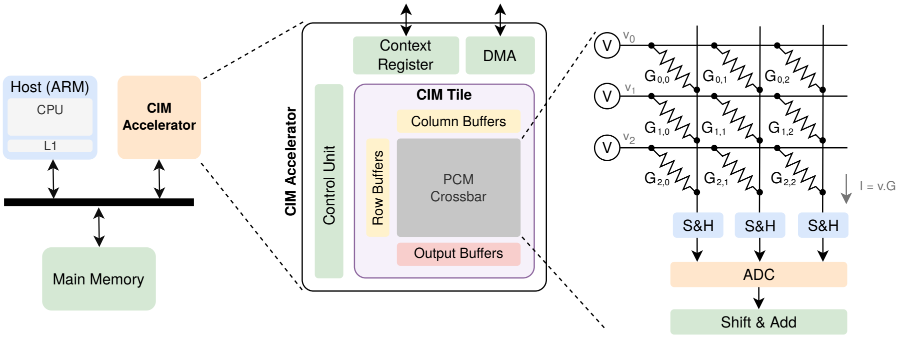


Compute In Memory (CIM)



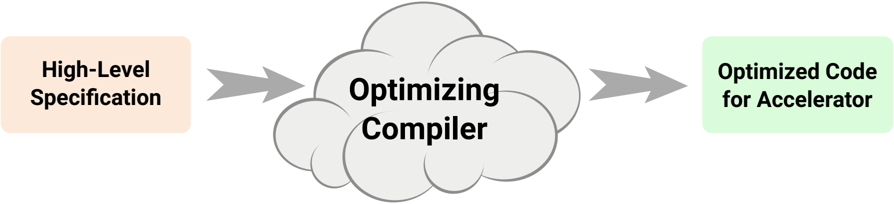
- Interweave Computation and Storage
- Example: Memristor-based Architecture from MNEMOSENE project (`https://www.mnemosene.eu`)

Compute In Memory (CIM)



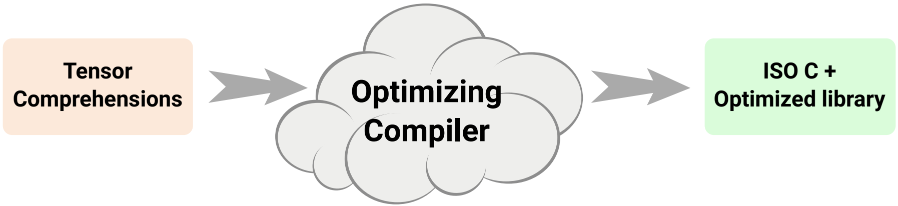
- Interweave Computation and Storage
- Example: Memristor-based Architecture from MNEMOSENE project (`https://www.mnemosene.eu`)
- High energy efficiency and throughput with fixed functions (e.g., matrix multiplication)

Detecting Accelerated Operations for CIM



- Goal: Reliably detect operations for efficient offloading
- At which stage?
- On which representation?
- Create reusable infrastructure

Detecting Accelerated Operations for CIM



- Goal: Reliably detect operations for efficient offloading
- At which stage?
- On which representation?
- Create reusable infrastructure

Tensor Comprehensions

Math-like notation

- Expresses operations on tensors
- Only information needed to define operation unambiguously
- Compiler infers shapes and iteration domains

Tensor Comprehensions

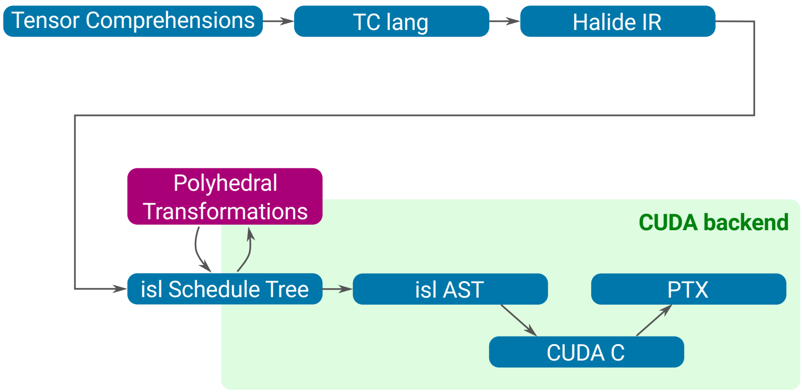
Math-like notation

- Expresses operations on tensors
- Only information needed to define operation unambiguously
- Compiler infers shapes and iteration domains

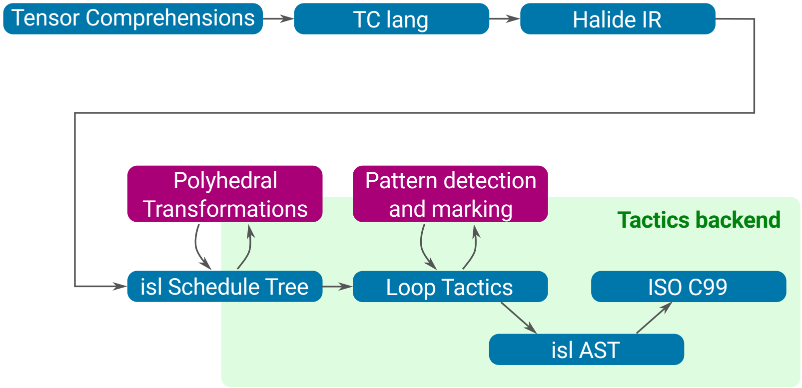
Example:

```
def mv(float(M,K) A, float(K) x) -> (C)
{
  C(i) +=! A(i,k) * x(k)
}
```

Tensor Comprehensions: Compilation



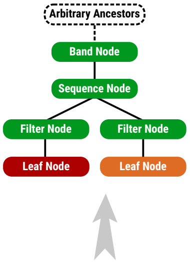
Integration of Loop Tactics



Matching Example: Matrix Multiplications

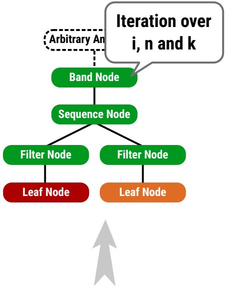
```
def kernel(
     float(M,N) A,
     float(N,K) B) -> (C)
{
   ...
   C(i,k) +=!
     A(i, n) * B(n, k)
   ...
}
```

Matching Example: Matrix Multiplications



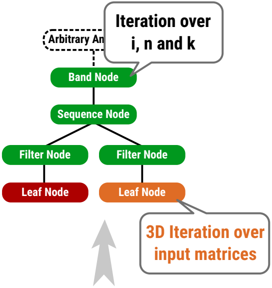

```
def kernel(
     float(M,N) A,
     float(N,K) B) -> (C)
{
   ...
   C(i,k) +=!
     A(i, n) * B(n, k)
   ...
}
```

Matching Example: Matrix Multiplications



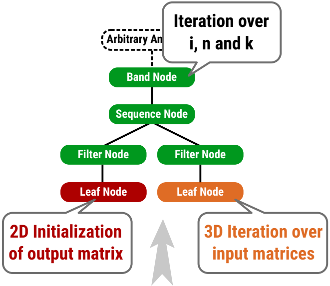

```
def kernel(
     float(M,N) A,
     float(N,K) B) -> (C)
{
   ...
   C(i,k) +=!
     A(i, n) * B(n, k)
   ...
}
```

Matching Example: Matrix Multiplications



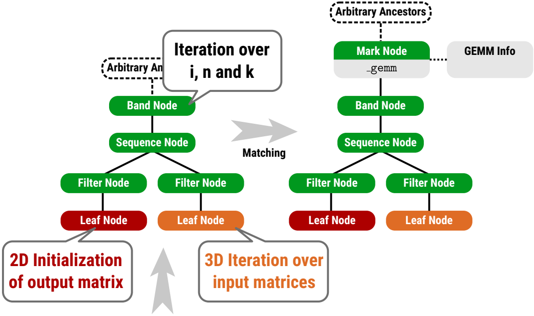

```
def kernel(
     float(M,N) A,
     float(N,K) B) -> (C)
{
   ...
   C(i,k) +=!
     A(i, n) * B(n, k)
   ...
}
```

Matching Example: Matrix Multiplications



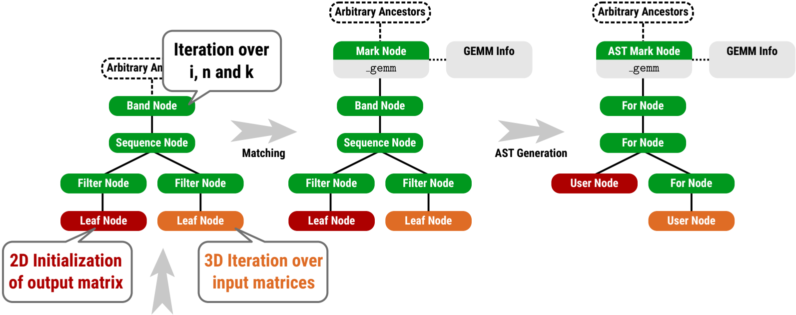

```
def kernel(
     float(M,N) A,
     float(N,K) B) -> (C)
{
   ...
   C(i,k) +=!
     A(i, n) * B(n, k)
   ...
}
```

Matching Example: Matrix Multiplications



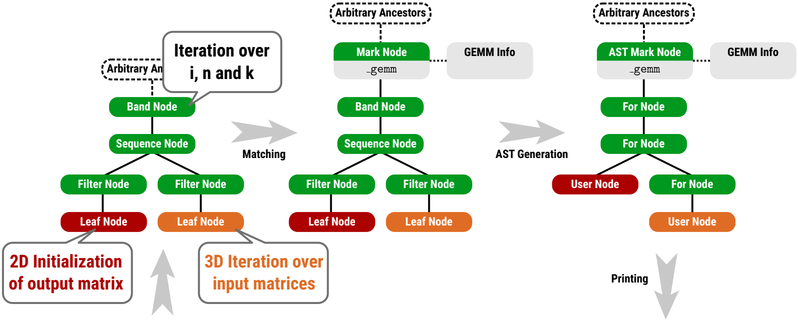

```
def kernel(
     float(M,N) A,
     float(N,K) B) -> (C)
{
   ...
   C(i,k) +=!
     A(i, n) * B(n, k)
   ...
}
```

Matching Example: Matrix Multiplications




```
def kernel(
     float(M,N) A,
     float(N,K) B) -> (C)
{
   ...
   C(i,k) +=!
     A(i, n) * B(n, k)
   ...
}
```

Matching Example: Matrix Multiplications



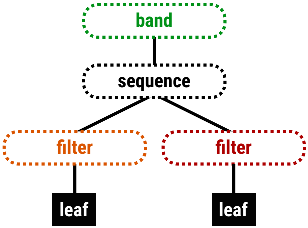

```
def kernel(
      float(M,N) A,
      float(N,K) B) -> (C)
{
   ...
   C(i,k) +=!
     A(i, n) * B(n, k)
   ...
}
```

```
void kernel(int K, int M, int N,
            float* C,
            float* A,
            float* B)
{
   ...
   cimblas_gemm(K, M, N,
                C, A, B,
                ...);
   ...
}
```

Loop Tactics: Tree Matchers

Tree Matcher defines pattern for subtree and captures nodes

```
schedule_node body;
schedule_node initBody;
schedule_node schedule;

auto matcher =
    band(schedule,
        sequence(
            filter(initBody,
                hasGemmInitPattern,
                leaf()),

            filter(body,
                hasGemmPattern,
                leaf())));
```



Loop Tactics Access Relation Matchers

Access Relation Matcher: Matches tensor accesses

```
auto hasGemmPattern = [&](schedule_node node) {
  auto _i = placeholder();
  auto _j = placeholder();
  auto _k = placeholder();
  auto _A = arrayPlaceholder();
  auto _B = arrayPlaceholder();
  auto _C = arrayPlaceholder();

  auto reads = /* get read accesses */;
  auto writes = /* get write accesses */;
  auto mRead = allOf(
      access(_C, _i, _j),
      access(_A, _i, _k),
      access(_B, _k, _j));

  auto mWrite = allOf(access(_C, _i, _j));
  return match(reads, mRead).size() == 1 &&
         match(writes, mWrite).size() == 1;
};
```

Loop Tactics Access Relation Matchers

Access Relation Matcher: Matches tensor accesses

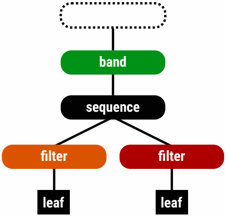
```
auto hasGemmPattern = [&](schedule_node node) {
  auto _i = placeholder();
  auto _j = placeholder();
  auto _k = placeholder();
  auto _A = arrayPlaceholder();
  auto _B = arrayPlaceholder();
  auto _C = arrayPlaceholder();

  auto reads = /* get read accesses */;
  auto writes = /* get write accesses */;
  auto mRead = allOf(
      access(_C, _i, _j),
      access(_A, _i, _k),
      access(_B, _k, _j));

  auto mWrite = allOf(access(_C, _i, _j));
  return match(reads, mRead).size() == 1 &&
         match(writes, mWrite).size() == 1;
};
```

Additionally match leaf expressions

Loop Tactics: Tree Builders

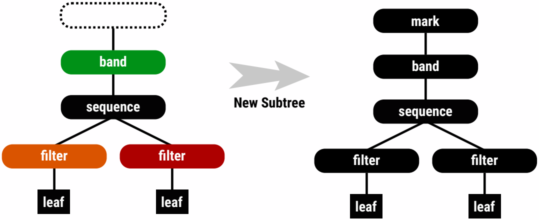
Tree Builder generates Subtree after Transformation

```
auto builder =
  mark ([&]() { return marker() },
    band([&]() { return schedule.getSchedule() },
      sequence(
        filter([&]() { return initBody.getFilter() }),
        filter([&]() { return body.getFilter() }))));
```



Tree Builder generates Subtree after Transformation

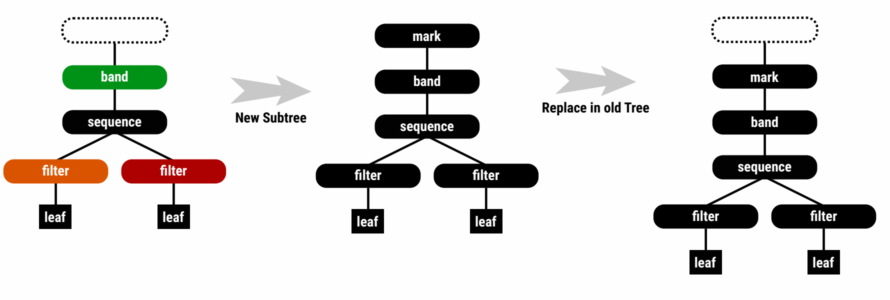
```
auto builder =
  mark ([&]() { return marker() },
    band([&]() { return schedule.getSchedule() },
      sequence(
        filter([&]() { return initBody.getFilter() }),
        filter([&]() { return body.getFilter() }))));
```



Tree Builder generates Subtree after Transformation

```
auto builder =
  mark ([&]() { return marker() },
    band([&]() { return schedule.getSchedule() },
      sequence(
        filter([&]() { return initBody.getFilter() }),
        filter([&]() { return body.getFilter() }))));
```

Loop Tactics: Tree Builders



Tree Builder generates Subtree after Transformation

```
auto builder =
  mark ([&]() { return marker() },
    band([&]() { return schedule.getSchedule() },
      sequence(
        filter([&]() { return initBody.getFilter() }),
        filter([&]() { return body.getFilter() }))));
```

Experimental Methodology

Implemented Matchers

- Matrix-matrix multiplications
- Matrix-vector multiplications

Experimental Methodology

Implemented Matchers

- Matrix-matrix multiplications
- Matrix-vector multiplications

Benchmarks

- Benchmarks: mm, mv, batchMM, 3mm, 4cmm, mlp3

Experimental Methodology

Implemented Matchers

- Matrix-matrix multiplications
- Matrix-vector multiplications

Benchmarks

- Benchmarks: mm, mv, batchMM, 3mm, 4cmm, mlp3

Static Impact

- Percentage of detected patterns in the code
- Test robustness against prior Transposition / Tiling

Experimental Methodology

Implemented Matchers

- Matrix-matrix multiplications
- Matrix-vector multiplications

Benchmarks

- Benchmarks: mm, mv, batchMM, 3mm, 4cmm, mlp3

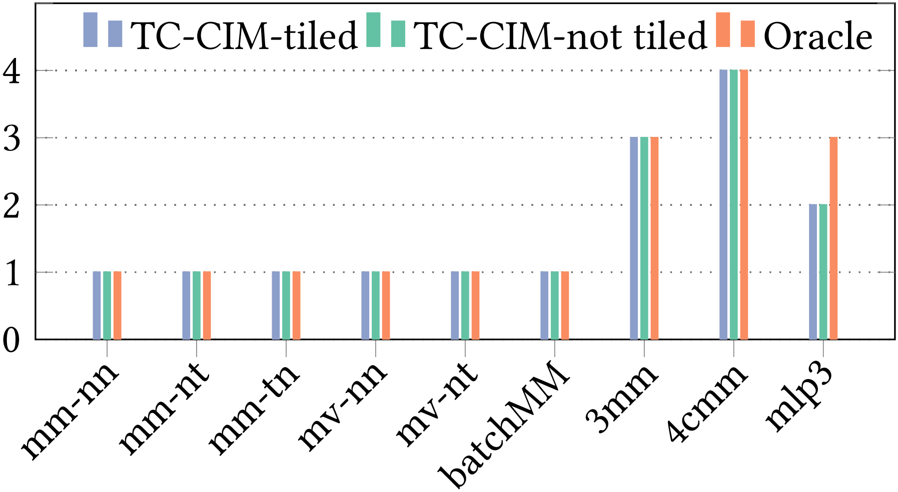
Static Impact

- Percentage of detected patterns in the code
- Test robustness against prior Transposition / Tiling

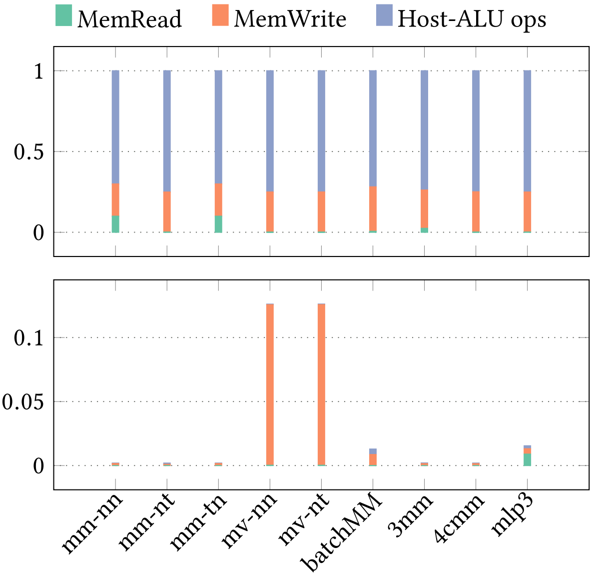
Dynamic Impact

- Dynamic instruction count unoptimized vs. optimized version

Detected Patterns in the Code



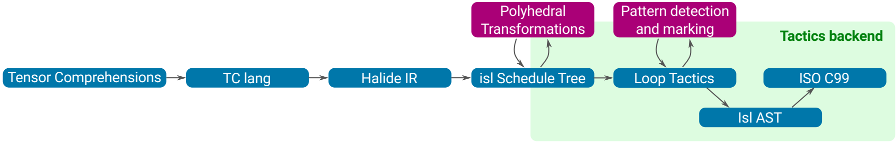
Breakdown of Dynamic Host CPU Instructions



No offloading (host CPU only)

With offloading Normalized to #instructions without offloading

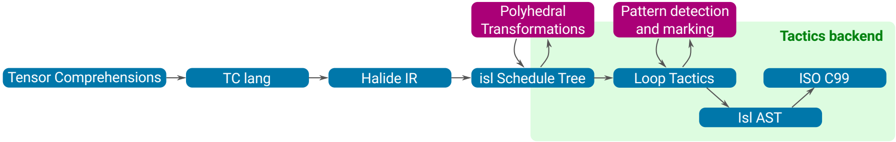
Positioning in the Pipeline



Matching after Affine Scheduling without Rescheduling:

- Leverages enabling transformations (e.g., fusion, tiling)
- Initial schedule as canonical form (e.g., permutability, band coalescing)
- No feedback for transformations (e.g., no architecture-specific tiling, fusion decisions, etc.)
- Complexity of matchers rises with prior transformations

Positioning in the Pipeline



Matching after Affine Scheduling without Rescheduling:

- Leverages enabling transformations (e.g., fusion, tiling)
- Initial schedule as canonical form (e.g., permutability, band coalescing)
- No feedback for transformations (e.g., no architecture-specific tiling, fusion decisions, etc.)
- Complexity of matchers rises with prior transformations

Matching earlier (at higher level of abstraction)

- More high-level information for matchers
- Simpler matchers & builders
- Less / no benefits from affine transformations

Summary and Future Work

Summary

- TC-CIM: Compilation flow for (CIM) accelerators
- Integration of Loop Tactics into Tensor Comprehensions
- Reliable detection + significant dynamic impact

Summary and Future Work

Summary

- TC-CIM: Compilation flow for (CIM) accelerators
- Integration of Loop Tactics into Tensor Comprehensions
- Reliable detection + significant dynamic impact

Future Work

- Explore positioning in the pipeline
- More complex matchers: fusion / minimizing data transfers
- Matching in MLIR (e.g., raise from lower-level dialects to high-level dialects)