

MDH-DSL: Reduction-Aware Data Parallelism via Multi-Dimensional Homomorphisms



Richard Schulze (r.schulze@uni-muenster.de), Sergei Gorlatch

Introduction

We present MDH-DSL:

- expresses *data-parallel computations* in a *reduction-aware* manner
- allows *user-defined* reduction operators
- grounded in the MDH formalism [1]

[1] Rasch, "(De/Re)-Composition of Data-Parallel Computations via Multi-Dimensional Homomorphisms", TOPLAS'24

Limitations of Existing DSLs

Halide program for MatVec (C++):

```
1 Func matvec(Func M, Func v, int K) {
2   Func w("w");
3   Var i;
4   RDom k(0, K);
5   w(i) = 0.0f;
6   w(i) += M(i, k) * v(k);
7   return w; }
```

⇒ Limited to fixed set of *built-in operators*

TVM program for MatVec (Python):

```
1 def matvec(I,K):
2   M = te.placeholder( (I,K), dtype='float32' )
3   v = te.placeholder( (K,) , dtype='float32' )
4   k = te.reduce_axis ( (0,K), name = 'k' )
5   w = te.compute ( (I,),
6     lambda i: te.sum( M[i,k] * v[k] , axis=k ) )
7   return [ M,v,w ]
```

⇒ Limited support for *nested reductions*

Lift program for MatVec (Scala):

```
1 def matvec =
2   nFun(K => nFun(I =>
3     fun(M: [[float] K] I => fun(v: [float] K =>
4       M :>> map(fun(row =>
5         zip(v, row) :>> map(*) :>> reduce(+, 0)
6       )))) )
```

⇒ Struggles with *nested reductions*

Linalg program for MatVec (MLIR):

```
1 func @matvec(%M: memref<128x64xf32>,
2   %v: memref<64xf32>,
3   %w: memref<128xf32>) {
4   %zero = arith.constant 0.0 : f32
5   linalg.fill ins(%zero : f32)
6     outs(%w : memref<128xf32>)
7   linalg.generic {
8     indexing_maps = [
9       affine_map<(i,k) -> (i,k)>,
10      affine_map<(i,k) -> (k)>,
11      affine_map<(i,k) -> (i)>
12    ],
13    iterator_types = ["parallel", "reduction"]
14  } ins(%M, %v : memref<128x64xf32>, memref<64xf32>)
15    outs(%w : memref<128xf32>) {
16      ^bb0(%m: f32, %vk: f32, %acc: f32):
17        %prod = arith.mulf %m, %vk : f32
18        %sum = arith.addf %acc, %prod : f32
19        linalg.yield %sum : f32
20    }
21  return }
```

⇒ Lacks explicit *semantic information*

MDH-DSL

MDH-DSL program for MatVec (Python):

```
1 def matvec( T:BasicType, I:int, K:int ):
2   @mdh()
3   def matvec__T_I_K():
4     return (
5       out_view[T]( w = [lambda i,k: (i) ] ),
6
7       md_hom[I,K]( f_mul, (cc, pw(add)) ),
8
9       inp_view[T,T]( M = [lambda i,k: (i,k) ] ,
10        v = [lambda i,k: (k) ] )
11     )
12   return matvec__T_I_K
```

Annotations: dimension sizes, multiply elements in M and v, concatenate in dimension i, sum in dimension k, buffers, scalar types, iteration space indices, data accesses

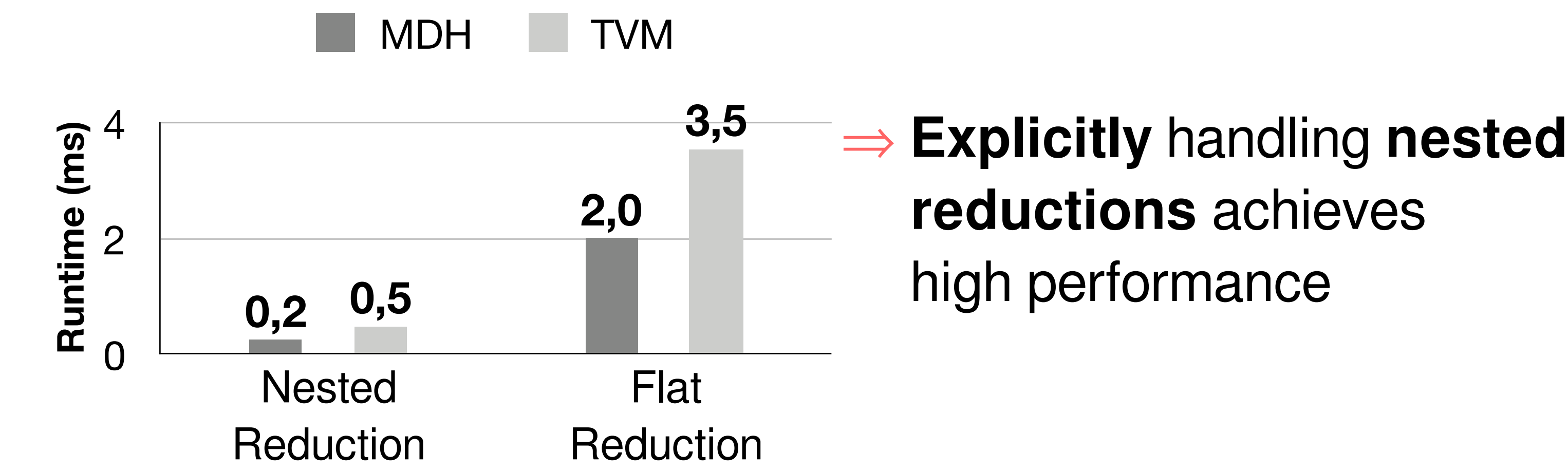
Comparison of DSL features regarding reductions:

DSL	Operators Explicit	Nested Reductions	Custom Operators
Halide	yes	no	no
TVM	yes	no	limited
Lift	yes	limited	limited
Linalg	no	no	yes
MDH-DSL	yes	yes	yes

Evaluation

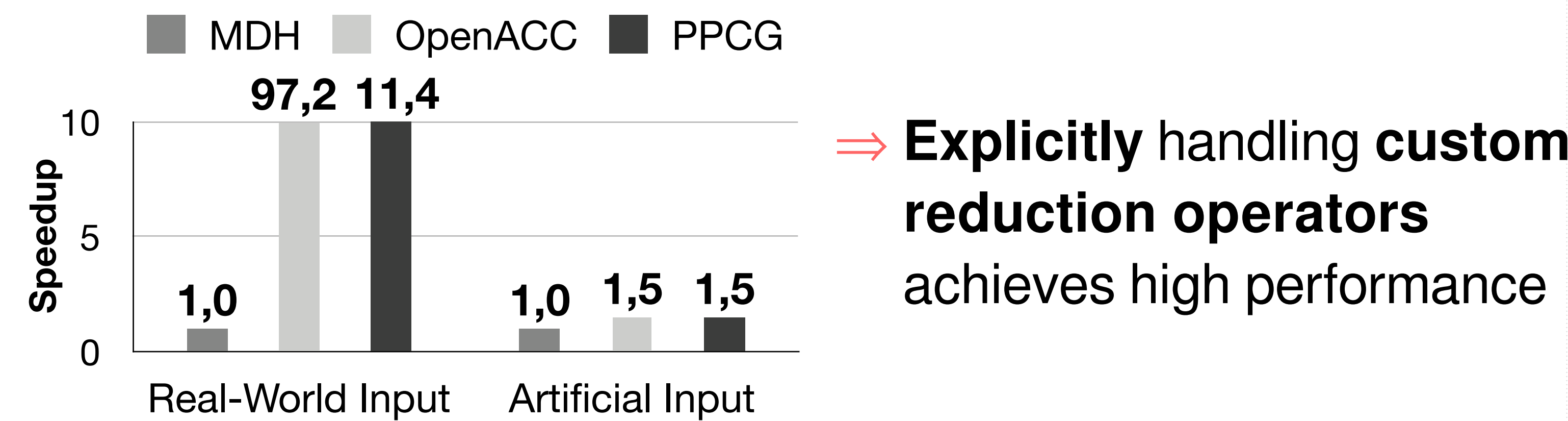
Performance advantage for nested reductions

Case Study: Deep Learning (MCC) on NVIDIA A100 GPU



Performance advantage for custom reductions

Case Study: Data Mining (PRL) on NVIDIA A100 GPU



172.5x faster than TVM for Dot on NVIDIA GPU

1.6x faster than TVM for MCC on NVIDIA GPU

5.1x faster than TVM for Dot on Intel CPU

MDH-DSL enables **better performance** than well-performing approaches on real-world data.

2.4x faster than cuDNN for MCC on NVIDIA GPU

1.1x faster than cuBLAS for Dot on NVIDIA GPU

9.4x faster than PPCG for MCC on NVIDIA GPU

3.9x faster than oneDNN for MCC on Intel CPU

6.1x faster than EKR for PRL on Intel CPU

5.4x faster than Pluto for Dot on Intel CPU

Student Research Competition

CGO 2026

Sydney, Australia

