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

Richard Schulze, Sergei Gorlatch

Universität Münster

Introduction

We present `MDH-DSL`:

- expresses *data-parallel computations* in a *reduction-aware manner*
- allows *user-defined* reduction operators
- is 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++):

```
Func matvec(Func M, Func v, int K) {
  Func w("w");
  Var i;
  RDom k(0, K);
  w(i) = 0.0f;
  w(i) += M(i, k) * v(k);
  return w; }
```

$\Rightarrow$ Limited to fixed set of *built-in operators*

Limitations of Existing DSLs

TVM program for MatVec (Python):

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

$\Rightarrow$ Limited support for *nested reductions*

Limitations of Existing DSLs

Lift program for MatVec (Scala):

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

⟹ Struggles with *nested reductions*

Limitations of Existing DSLs

Linalg program for MatVec (MLIR):

```
func @matvec(%M: memref<128x64xf32>,
             %v: memref<64xf32>,
             %w: memref<128xf32>) {
```

⋮

```
    iterator_types = ["parallel", "reduction"]
  } ins(%M, %v : memref<128x64xf32>,memref<64xf32>)
    outs(%w : memref<128xf32>) {
    ^bb0(%m: f32, %vk: f32, %acc: f32):
      %prod = arith.mulf %m, %vk : f32
      %sum  = arith.addf %acc, %prod : f32
      linalg.yield %sum : f32
    }
  return }
```

$\Rightarrow$ Lacks explicit *semantic information*

Addressing the Limitations with MDH-DSL

MDH-DSL program for MatVec (Python):

```
def matvec( T:BasicType, I:int, K:int ):
  @mdh()
  def matvec__T_I_K():
    return (
      out_view[T]( w = [lambda i,k: (i) ] ),

      md_hom[I,K]( f_mul, (cc, pw(add)) ),

      inp_view[T,T]( M = [lambda i,k: (i,k) ] ,
                     v = [lambda i,k: (k)   ] )
    )
  return matvec__T_I_K
```

dimension sizes

multiply elements in M and v

concatenate in dimension i

sum in dimension k

buffers

scalar types

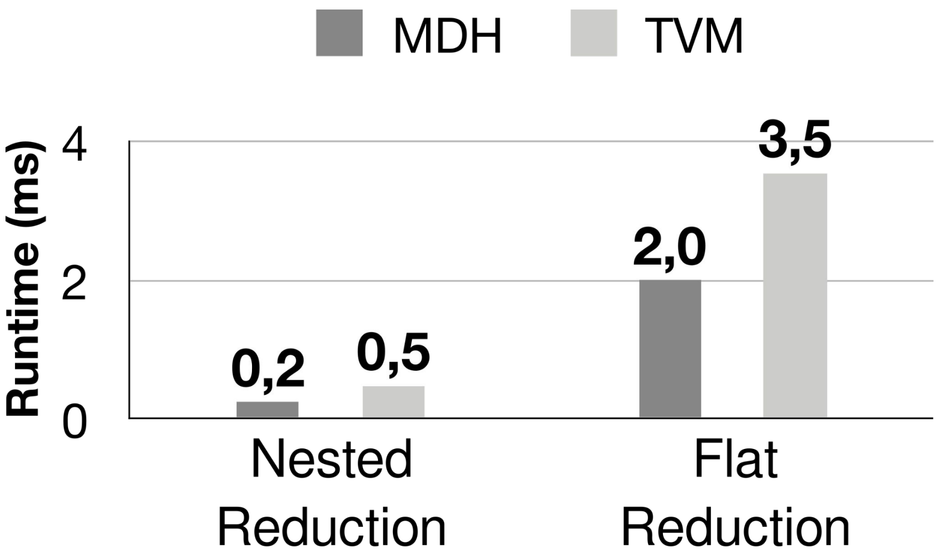
iteration space indices

data accesses

- ⇒ Reductions are first-class citizens
- ⇒ Reductions can be user-defined
- ⇒ Nested Reductions can explicitly be expressed

Evaluation: Performance Advantage for Nested Reductions

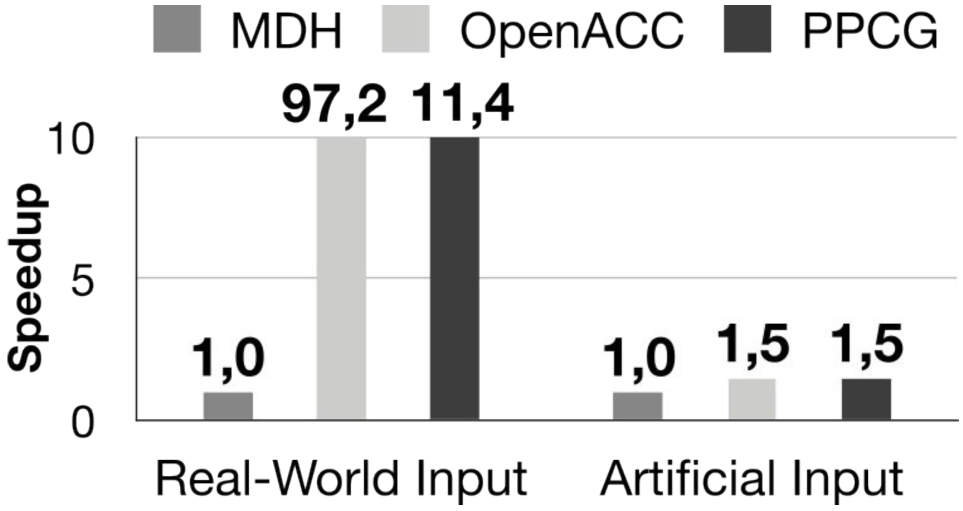
Case Study: Deep Learning (MCC) on NVIDIA A100 GPU



⇒ **Explicitly** handling **nested reductions** achieves high performance

Evaluation: Performance Advantage for Nested Reductions

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



⇒ **Explicitly** handling **custom reduction operators** achieves high performance

Evaluation

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**

6.1x faster than **EKR** for **PRL** on **Intel CPU**

3.9x faster than **oneDNN** for **MCC** on **Intel CPU**

9.4x faster than **PPCG** for **MCC** on **NVIDIA GPU**

5.4x faster than **Pluto** for **Dot** on **Intel CPU**

Questions?

https://richardschulze.net
r.schulze@uni-muenster.de

Nested Reduction vs. Flat Reduction