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:

- 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

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

Stencils

<table>
<thead>
<tr>
<th>CPU</th>
<th>Gaussian (2D)</th>
<th>Jacobi (3D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lift [2]</td>
<td>4.90</td>
<td>5.96</td>
</tr>
<tr>
<td>MKL-DNN</td>
<td>6.99</td>
<td>14.31</td>
</tr>
<tr>
<td>GPU</td>
<td>Gaussian (2D)</td>
<td>Jacobi (3D)</td>
</tr>
<tr>
<td>Lift [2]</td>
<td>2.33</td>
<td>1.09</td>
</tr>
<tr>
<td>cuDNN</td>
<td>3.78</td>
<td>19.11</td>
</tr>
</tbody>
</table>


Data Mining

<table>
<thead>
<tr>
<th>CPU</th>
<th>(2^{15})</th>
<th>(2^{16})</th>
<th>(2^{17})</th>
<th>(2^{18})</th>
<th>(2^{19})</th>
<th>(2^{20})</th>
</tr>
</thead>
</table>


Tensor Contractions

<table>
<thead>
<tr>
<th>GPU</th>
<th>Numbers</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>COGENT [3]</td>
<td>1.26</td>
<td>1.16</td>
<td>2.12</td>
<td>1.24</td>
<td>1.18</td>
<td>1.36</td>
</tr>
<tr>
<td>F-TC [4]</td>
<td>1.19</td>
<td>2.00</td>
<td>1.43</td>
<td>2.89</td>
<td>1.35</td>
<td>1.54</td>
</tr>
</tbody>
</table>


Linear Algebra

<table>
<thead>
<tr>
<th>CPU</th>
<th>GEMM</th>
<th>GEMV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lift [1]</td>
<td>fails</td>
<td>3.04</td>
</tr>
<tr>
<td>MKL</td>
<td>4.22</td>
<td>0.74</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GPU</th>
<th>GEMM</th>
<th>GEMV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lift [1]</td>
<td>4.33</td>
<td>1.17</td>
</tr>
<tr>
<td>cuBLAS</td>
<td>2.91</td>
<td>0.83</td>
</tr>
</tbody>
</table>

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.

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
The MDH DSL

Example: Matrix Multiplication

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

What’s happening?

1. Prepare the domain-specific input uniformly for \texttt{md\_hom}; for this, our DSL provides pattern \texttt{view}.
   - \texttt{here:} fuse matrices \texttt{A} and \texttt{B} to 3-dimensional array of pairs consisting of the elements in \texttt{A} and \texttt{B} to multiply: \( i, j, k \mapsto (A[i,k], B[k,j]) \).

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

\[
\text{MatMul} = \text{md\_hom}( \ast, (+++, +) ) \circ \text{view}(A,B)(i,j,k)(A[i,k], B[k][j])
\]

MDH Representation:

**T f(T A_i_k, T B_k_j, T C_i_j) {**

\[C_{i,j} += A_{i,k} \ast B_{k,j};\]

\} return C_{i,j};**

\]

GEMM in C

\[
\begin{align*}
\text{for( int } i = 0; i < M; ++i ) \\
&\quad \text{for( int } j = 0; i < N; ++j ) \\
&\quad\quad \text{for( int } k = 0; i < K; ++k ) \\
&\quad\quad\quad C[i][j] += A[i][k] \ast B[k][j];
\end{align*}
\]

**means: Unknown Combine Operator (UCO) → NO parallelization, BUT tiling, caching, ...**

**isl[1]**

Experimental Results

• **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 \texttt{md\_poly}:

- \texttt{md\_poly} is based on both the polyhedral model and the MDH code generation approach;

- \texttt{md\_poly} combines productivity (as in polyhedral compilers) and portable high performance (as in the MDH approach);

- \texttt{md\_poly} achieves sometimes better performance than hand-optimized approaches.

\textbf{Future Work:} Analyze and Evaluate \texttt{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 ];
}
```

\( N \geq 2K \rightarrow \text{parallelizable} \)

else \( \rightarrow \text{NOT parallelizable} \)

“programs with parametric dependence distance (e.g., A[N-i] = A[i])"
Q: Unclear whether all polyhedral programs can be converted to MDH?

“if-conditionals using modulo arithmetic (e.g., if \((t \mod 2 == 0)\) 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)”
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?

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

```c
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;
        }
    }
}
```