

Connecting Kokkos with the Polyhedral Model

ExaSoft General Assembly 2025

Ugo BATTISTON (PhD Student)

Supervisors: Philippe Clauss and Marc Pérache
Workpackage 1 - 2

September 23, 2025

INRIA CAMUS



PROGRAMME
DE RECHERCHE
NUMÉRIQUE
POUR L'EXASCALE

Table of Contents

Kokkos

Polyhedral Model

Polly: Automatic Polyhedral Tool

kokkos extention making the polyedral model available

Results

Next Steps

Future Work

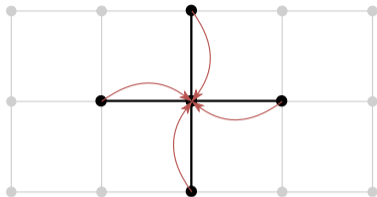
Kokkos



kokkos

- Modern C++ template library.
- Abstraction of parallelism.
- Promising performance portability across architectures (CPU, GPU).

Kokkos: Stencil example code



```
1 void stencil(View<float**> &A, View<float**> &B, long m, long n)
  {
3   const auto policy = MDRangePolicy<Serial, Rank<2>>({1, 1}, {m - 1, n - 1});

5   parallel_for(policy, KOKKOS_LAMBDA(long i, long j) {
      B(i, j) = A(i, j) + A(i, j - 1) + A(i, j + 1) + A(i - 1, j) + A(i + 1, j);
7   });
  }
```

Improve Kokkos performance with the polyhedral model

Why is the `parallel_for` structure a good candidate for the polyhedral model?

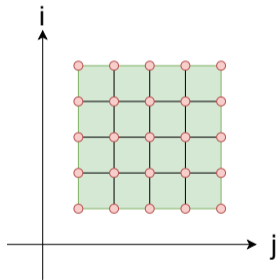
- Regular loop structures
- Well-defined loop bounds
- Affine memory access patterns

Polyhedral Model

What is the Polyhedral Model?

- A mathematical model representing loop iterations as integer points in a polyhedron.
- Geometric operations on these polyhedra are translated into program transformations.
- Enables loop transformations to optimize memory access, reduce dependencies, reorganize instructions, improve cache locality, enable vectorization, apply loop tiling, and expose parallelism opportunities.

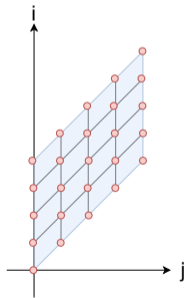
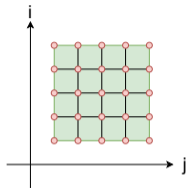
```
1 for (i : 1 -> n)  
  for (j : 1 -> n)  
3   Statement(i, j);
```



Polyhedral Model Example

```
for (int i = 1; i < n - 1; i++)  
  for (int j = 1; j < m - 1; j++)  
    A[i][j] = A[i][j] + A[i+1][j] + A[i][j+1];
```

```
for (int i = 0; i <= n + m - 2; i++) {  
  int lbj = max(0, i - n + 1);  
  int ubj = min(i, m - 1);  
  #pragma omp parallel for  
  for (int j = lbj; j <= ubj; j++)  
    A[i-j][j] = A[i-j][j] + A[i-j][j+1] + A[i-j+1][j];  
}
```



Polly: Automatic Polyhedral Tool



Polly

- An LLVM subproject.
- Usable directly from Clang.
- Directly analyzes the LLVM Intermediate Representation (IR), rather than the source code.
- Operating on the LLVM IR abstracts away the complexity of C++.

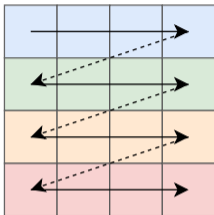
kokkos extention making the polyedral
model available

Modifications to Kokkos

- Isolation of loops.
- Removal of automatic tiling.
- Removal of micro-optimizations in the `Kokkos::View` structure.
- Added annotations with useful information for the polyhedral model (array dimensions, array pointers, and delinearized iterators).

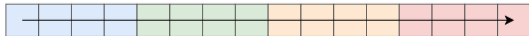
Polly modifications: Delinearization

User vision of Kokkos::View accesses



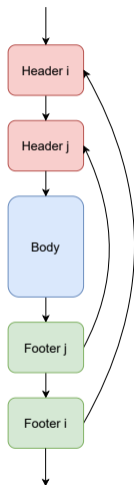
```
1 Kokkos::View<float **> A("A", N, M);  
2 for(size_t i = 0; i < N; ++i)  
   for(size_t j = 0; j < M; ++j)  
4   A(i, j) = ....
```

Compiler vision of Kokkos::View accesses



```
1 Kokkos::View<float **> A("A", N, M);  
2 for(size_t i = 0; i < N; ++i)  
3   for(size_t j = 0; j < M; ++j)  
   A.data[i * M + j] = ....
```

Polly modifications: Generated IR restructuring



Some loads have been moved outside the loop

- Array pointers
- Array dimensions

With Kokkos information structure, we know that these values do not change during the loop execution.

New options to kokkos : usePolyOpt

Added an option to enable or disable polyhedral analysis

```
1 void stencil(View<float **> &A, View<float **> &B, long m, long n)
  {
3   auto policy = MDRangePolicy<Serial, Rank<2>>({1, 1}, {m - 1, n - 1});

5   parallel_for<usePolyOpt>(policy, KOKKOS_LAMBDA(long i, long j) {
       B(i, j) = A(i, j) + A(i, j - 1) + A(i, j + 1) +
7         A(i - 1, j) + A(i + 1, j);
   });
9 }
```

Results

Benchmark suite: Polybench

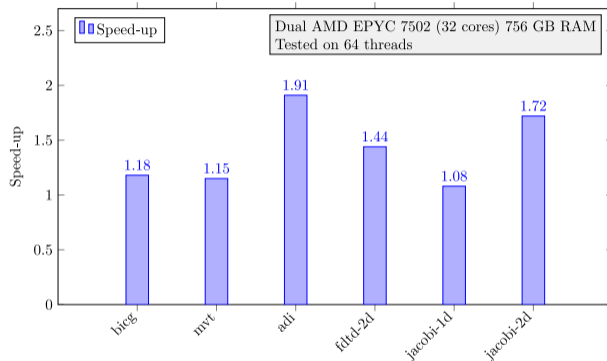


Figure 1: Speedup of Kokkos + Polly (usePolyOpt) compared to Kokkos

Next Steps

Allow viewing of multiple nested loops

- Some algorithms require multiple calls to `parallel_for`.
- A limited view of a single loop nest may miss certain optimization opportunities.
- To improve polyhedral optimizations, the compiler needs a global view of the loop nests.

Implementation

```
void function(View<"A", float **> &A, View<"B", float **> &B, View<"C", float **> &
    C, View<"D", float **> &D, View<"tmp", float **> &tmp, size_t ni, size_t nj,
    size_t nl, size_t nk, double alpha, double beta) {
2   const auto policy_2D_1 = MDRangePolicy<Rank<2>>({0, 0}, {ni, nj});
3   const auto policy_2D_2 = MDRangePolicy<Rank<2>>({0, 0}, {ni, nl});
4
5   /* D := alpha*A*B*C + beta*D */
6   Kokkos::parallel_for<usePolyOpt, "p1.l1 == p2.l1">(
7       policy_2D_1, KOKKOS_LAMBDA(size_t i, size_t j) {
8           tmp(i, j) = 0.0;
9           for (size_t k = 0; k < nk; ++k)
10              tmp(i, j) += alpha * A(i, k) * B(k, j);
11        },
12        policy_2D_2, KOKKOS_LAMBDA(size_t i, size_t j) {
13            D(i, j) *= beta;
14            for (size_t k = 0; k < nj; ++k)
15                D(i, j) += tmp(i, k) * C(k, j);
16        }
17    );
18 }
```

Adding the Pluto Scheduler

Adding the schedule produced by **Pluto** to the Kokkos polyhedral toolchain:

- Generate a polyhedral representation suitable for Pluto.
- Apply polyhedral transformations using Pluto and its heuristics.
- Re-inject the transformations back into Polly.

Future Work

More Benchmarks and Tests

Additional benchmarks are needed to validate the implementation:

- More benchmarks from the Polybench suite and others.
- Validate the kernel fusion implementation.
- Test and validate Pluto's usage.

Current limitations in Kokkos GPU code generation:

- Generation of a GPU schedule using PPCG (Polyhedral Parallel Code Generation).
- Re-injection of the schedule into Polly and generation of the LLVM GPU IR.

Thank you!