RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

FRANCE 2030
PROGRAMME DE RECHERCHE
NUMÉRIQUE POUR L'EXASCALE

Polyhedral Model for Kokkos Code Optimization

ExaSoft General Assembly 2024

Ugo BATTISTON
PhD Student under the supervision of Philippe Clauss and Marc Pérache

November 7, 2024

INRIA

Table of Contents

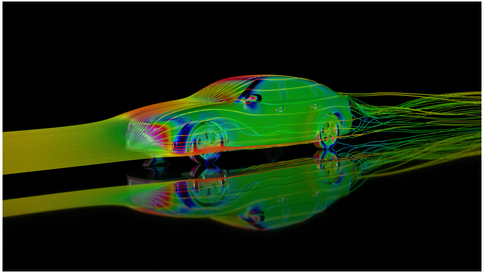
Context

Thèse

Thesis Topic: (starting 01/10/2023)
Disambiguation of C++ Complexity for Advanced Program Optimization and Parallelization.

- **Programme de recherche exploratoire (PEPR) Numérique pour l'exascale (NumPEx)**
  - **Work Package 1**
    Efficient and composable programming models.
  - **Work Package 2**
    Just-in-Time code optimization with continuous feedback loop.

Simulation code



- Complex code with multiple computational kernels and numerous nested loops
- Supercomputer with millions of multi-architecture cores

Changing Simulation Programming Language

With the evolution of programming paradigms and architectures, the **CEA** needs to modernize the language used to write simulation code.

- Transition from Fortran to C++.
- Need to replace old Fortran technologies.
- Need to provide developers with tools to easily program simulation codes in C++.

Changing Simulation Programming Language

With the evolution of programming paradigms and architectures, the **CEA** needs to modernize the language used to write simulation code.

- Transition from Fortran to C++.
- Need to replace old Fortran technologies.
- Need to provide developers with tools to easily program simulation codes in C++.

A choice has been made to use the **Kokkos** library to provide a high-level abstraction of parallelism.

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

FRANCE 2030

PROGRAMME DE RECHERCHE
NUMÉRIQUE POUR L'EXASCALE

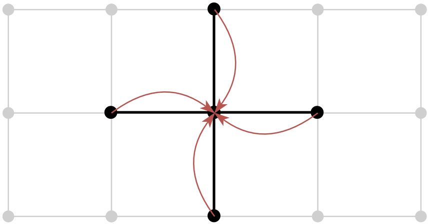
cea cnrs inria

Kokkos

What is Kokkos ?

- Modern C++ template library
- Developed by the United States Department of Energy
- Abstraction of parallelism
- Performance portability across different architectures (CPU, GPU)

Kokkos: Stencil example C++ code



```
void stencil(float *A, float *B, long m, long n)
{
  for (int i = 1; i < m - 1; i++)
    for (int j = 1; j < n - 1; j++)
      B[i * n + j] = A[i * n + j] + A[i * n + (j - 1)] + A[i * n + (j + 1)] +
                     A[(i - 1) * n + j] + A[(i + 1) * n + j];
}
```

Listing 1: Native C++ code of stencil

Kokkos: Stencil example code

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

  Kokkos::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);
    });
}
```

Listing 2: Kokkos code of stencil

Kokkos optimizations

- **Performance optimization**:
  - Manual structure of code optimizations (linear array, tilling).
  - Preprocessor macros can be used to enable optimizations.
- **Performance portability**:
  - Kokkos provides a high-level abstraction of parallelism.
  - Code written with Kokkos can be executed on different architectures.

Objectives

The main objective is:

- Understand the structure of Kokkos codes.
- Propose a transparent solution to the user to optimize the code.
- Evaluate the performance of the proposed solution.

Objectives

The main objective is:

- Understand the structure of Kokkos codes.
- Propose a transparent solution to the user to optimize the code.
- Evaluate the performance of the proposed solution.

Kokkos remarks:

- GPU like writing kernels (users do not have to write loops).
- Natural loops with constant lower/upper bounds and a regular iteration space.

Objectives

The main objective is:

- Understand the structure of Kokkos codes.
- Propose a transparent solution to the user to optimize the code.
- Evaluate the performance of the proposed solution.

Kokkos remarks:

- GPU like writing kernels (users do not have to write loops).
- Natural loops with constant lower/upper bounds and a regular iteration space.

To do this, we will use the **Polyhedral Model** to optimize the code.

Polyhedral Model

What is the Polyhedral Model?

- A mathematical representation of loop nests in a program using polyhedra within a multidimensional space.
- Enables linear transformations of loops to optimize memory access, reduce dependencies, reorganize instructions, cache locality, vectorization, loop tilling and identify parallelism opportunities.

Constraints for Applying the Polyhedral Model

- **Static Control Parts (SCoPs)**:
  - Code must be composed of regions where control flow is statically predictable.

Constraints for Applying the Polyhedral Model

- **Static Control Parts (SCoPs)**:
  - Code must be composed of regions where control flow is statically predictable.
- **Affine Loop Bounds**

Constraints for Applying the Polyhedral Model

- **Static Control Parts (SCoPs)**:
  - Code must be composed of regions where control flow is statically predictable.
- **Affine Loop Bounds**
- **Affine Array Accesses**

Constraints for Applying the Polyhedral Model

- **Static Control Parts (SCoPs)**:
  - Code must be composed of regions where control flow is statically predictable.
- **Affine Loop Bounds**
- **Affine Array Accesses**
- **No Side Effects**

Constraints for Applying the Polyhedral Model

- **Static Control Parts (SCoPs):**
  - Code must be composed of regions where control flow is statically predictable.
- **Affine Loop Bounds**
- **Affine Array Accesses**
- **No Side Effects**
- **Single Entry and Exit Points**

Constraints for Applying the Polyhedral Model

- **Static Control Parts (SCoPs)**:
  - Code must be composed of regions where control flow is statically predictable.
- **Affine Loop Bounds**
- **Affine Array Accesses**
- **No Side Effects**
- **Single Entry and Exit Points**
- **No Irregular Control Flow**
  - Avoid irregular control flow constructs like `goto` statements or early exits.

Constraints for Applying the Polyhedral Model

- **Static Control Parts (SCoPs)**:
  - Code must be composed of regions where control flow is statically predictable.
- **Affine Loop Bounds**
- **Affine Array Accesses**
- **No Side Effects**
- **Single Entry and Exit Points**
- **No Irregular Control Flow**
  - Avoid irregular control flow constructs like `goto` statements or early exits.
- **Data Dependence Analysis**:
  - Code should allow for clear determination of read and write accesses to memory locations.

Polyhedral Model Example

Visual representation of a polyhedral transformation:

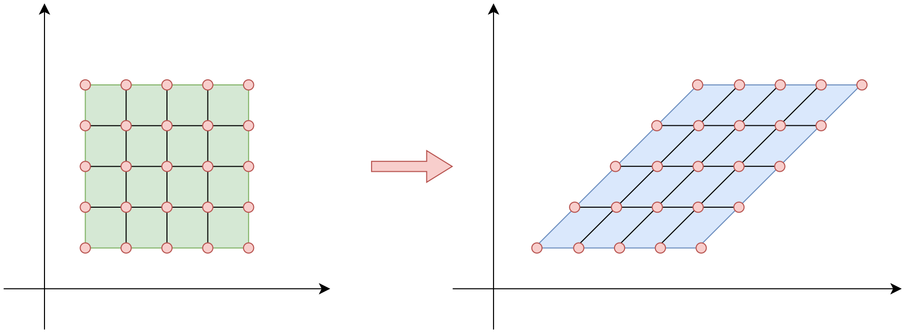


Figure 1: Polyhedral representation of a skewing transformation

Polyhedral Model Example

```c
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];
```

Listing 3: Stencil before transformation

Polyhedral Model Example

```c
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];
```

Listing 5: Stencil before transformation

```c
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];
}
```

Listing 6: Stencil after transformation

Several Polyhedral Tools

- **Polly**:
  - An LLVM project that provides a polyhedral optimizer for LLVM.
  - Transforms loop nests in LLVM intermediate representation (IR) to optimize performance.

Several Polyhedral Tools

- **Polly**:
  - An LLVM project that provides a polyhedral optimizer for LLVM.
  - Transforms loop nests in LLVM intermediate representation (IR) to optimize performance.
- **Pluto**:
  - A polyhedral optimizer for C programs.
  - Transforms loop nests in C code to optimize performance.

Several Polyhedral Tools

- **Polly**:
  - An LLVM project that provides a polyhedral optimizer for LLVM.
  - Transforms loop nests in LLVM intermediate representation (IR) to optimize performance.
- **Pluto**:
  - A polyhedral optimizer for C programs.
  - Transforms loop nests in C code to optimize performance.
- **Apollo**:
  - A runtime polyhedral optimizer.
  - Transforms loop nests based on runtime behavior.

Constraints of Kokkos Codes

Application of the Polyhedral Model on Kokkos Codes:

- Regular "perfect" loops for the application of the polyhedral model.
- Code already parallelized.
- C++ too complex for the source code to be analyzed.
- Complexity of the analysis due to the complexity of the generated code.
- Targets different architectures.

Constraints of Kokkos Codes

Application of the Polyhedral Model on Kokkos Codes:

- Regular "perfect" loops for the application of the polyhedral model.
- Code already parallelized.
- C++ too complex for the source code to be analyzed.
- Complexity of the analysis due to the complexity of the generated code.
- Targets different architectures.

Currently no tools are able to apply the polyhedral model to Kokkos codes.

Why polly ?

- Analyzes and optimizes loop nests directly in LLVM intermediate representation.
- Disambiguation of the C++ complexity.
- Simple to use, directly integrated in LLVM.
- Provides a set of optimization options to transform loop nests.

Current work

Polly modifications

- Manual search of SCoPs
- Interprocedural analysis to extend and inline functions in order to maximize the size of SCoPs.

Kokkos modifications

Polly's modifications is not enough we need to modify Kokkos:

- Modification of the user API to allow the addition of an option to give the user the choice to use instrumentation for Polly.
- Rewriting of the Serial backend to have clean loops (remove tiles).

Kokkos modifications

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

  Kokkos::parallel_for<Kokkos::usePolyOpt>(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);
    });
}
```

Listing 7: Kokkos code of stencil with modification

Current work

- Currently no results.
- Only applicable on Serial Kokkos backend.
- The SCoPs search still depends on code generation.

Questions?