Диалект MLIR для Общего Тайлинга Циклов

А.В. Левченко

2@lto.ru

Суперкомпьютерный центр СПбПУ

Диалект MLIR Common Loop Tiling (CLT)

Progressive lowering

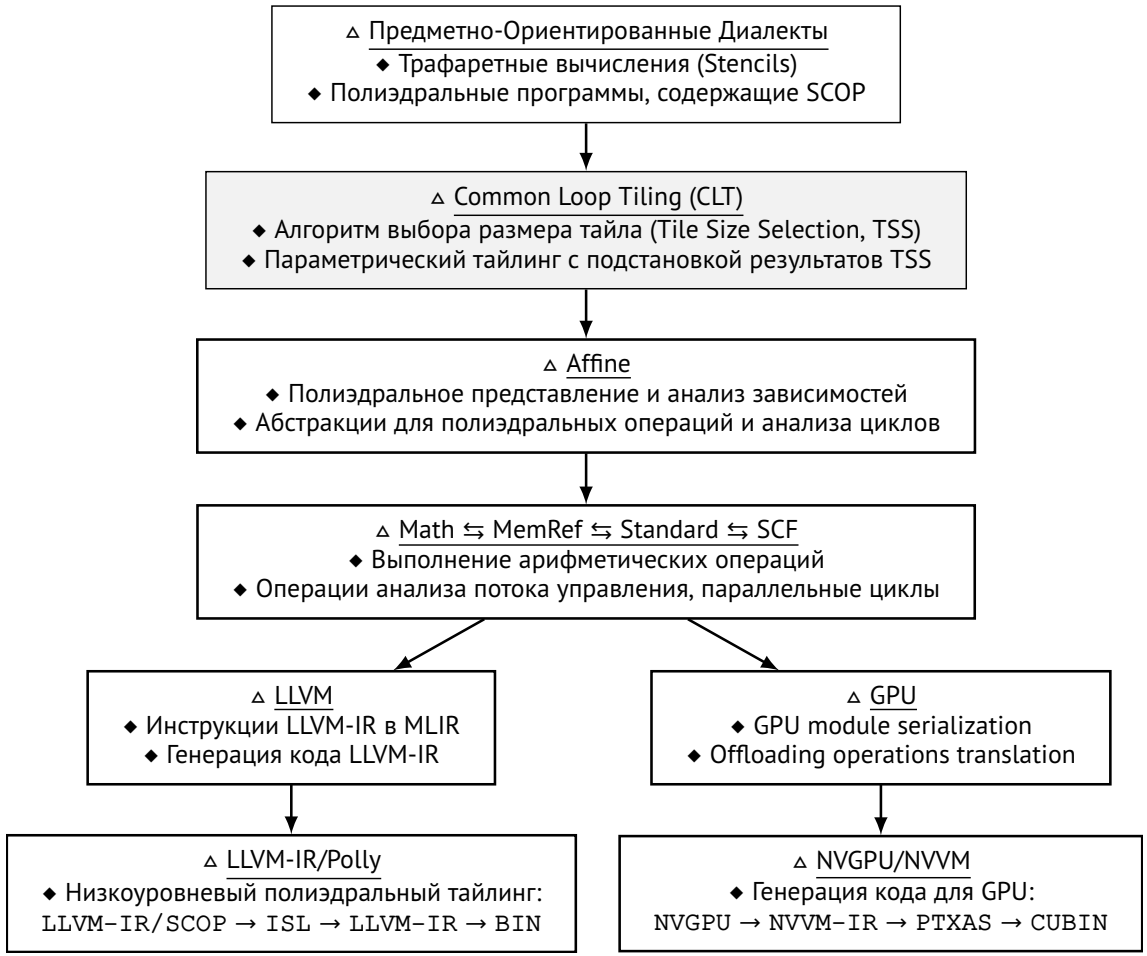


Figure 1 — Схема взаимодействия CLT с некоторыми диалектами инфраструктуры MLIR

Диалект MLIR Common Loop Tiling (CLT)

Полиэдральный тайлинг с использованием конверсии диалектов Affine → GPU

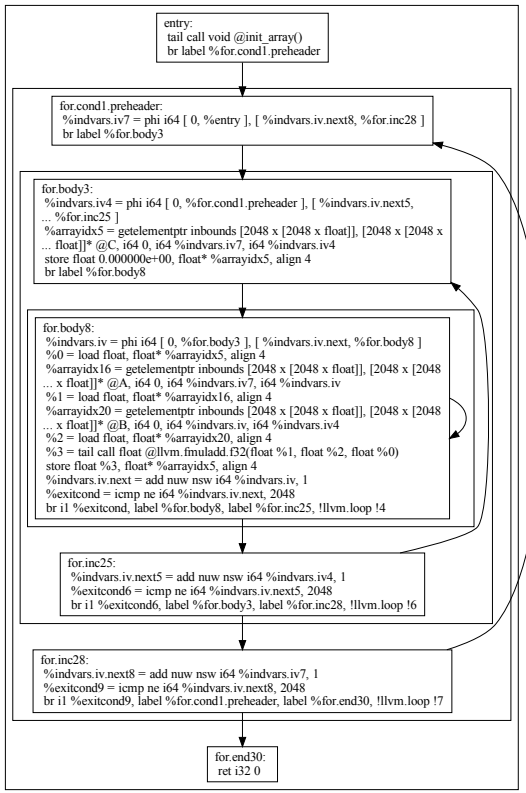
```
// Циклы CLT/Affine
func.func @clt_loop(%arg0: memref<?x?x?x?xf32>,
  % arg1: memref<?x?x?x?xf32>) {
clt.launch {dev = "gpu"} {
clt.for %arg2 = 0 to 512 {
  clt.for %arg3 = 0 to 256 {
    clt.for %arg4 = 0 to 128 {
      clt.for %arg5 = 0 to 64 {
        % 0 = memref.load %arg0[%arg2, %arg3, %arg4,
          % arg5] : memref<?x?x?x?xf32>
        memref.store %0, %arg1[%arg2, %arg3, %arg4, %arg5]
          : memref<?x?x?x?xf32>
[...]
  clt.terminator
// Тайлинг ( размер тайла = 256, размер кеша = 1024 )
func.func @loop_tiling(
% arg0: memref<196608x1xf32>,
% arg1: memref<196608x1xf32>)
-> memref<196608x1xf32> {
clt.launch {dev = "gpu", tss = 256, cache = 1024} {
  clt.for %arg2 = 0 to 196608 step 1024 {
    clt.for %arg3 = 0 to 1 step 256 {
      clt.for %arg4 = #map(%arg2) to #map1(%arg2) step 256 {
        clt.for %arg5 = #map(%arg3) to #map2(%arg3) step 256 {
          clt.if #set(%arg4) {
            clt.for %arg6 = #map(%arg4) to #map2(%arg4) {
              clt.for %arg7 = #map(%arg5) to #map3(%arg5) {
                % 0 = clt.load %arg0[%arg6, %arg7]
                  : memref<196608x1xf32>
                clt.store %0, %arg1[%arg6, %arg7]
                  : memref<196608x1xf32>
[...]
}
```

```
// Циклы GPU/Affine
module {
  func.func @clt_loop(%arg0: memref<?x?x?x?xf32>,
    % arg1: memref<?x?x?x?xf32>) {
    % c0 = arith.constant 0 : index
    % c512 = arith.constant 512 : index
    % 0 = arith.subi %c512, %c0 : index
    % c1 = arith.constant 1 : index
    % c0_0 = arith.constant 0 : index
    % c256 = arith.constant 256 : index
    % 1 = arith.subi %c256, %c0_0 : index
    % c1_1 = arith.constant 1 : index
    % c1_2 = arith.constant 1 : index
    // Операция запуска ядра GPU
    gpu.launch blocks(%arg2, %arg3, %arg4)
    in (%arg8 = %0, %arg9 = %c1_2, %arg10 = %c1_2)
    threads(%arg5, %arg6, %arg7)
    in (%arg11 = %1, %arg12 = %c1_2, %arg13 = %c1_2) {
    % 2 = arith.addi %c0, %arg2 : index
    % 3 = arith.addi %c0_0, %arg5 : index
    affine.for %arg14 = 0 to 128 {
      affine.for %arg15 = 0 to 64 {
        % 4 = memref.load %arg0[%2, %3, %arg14, %arg15]
          : memref<?x?x?x?xf32>
        memref.store %4, %arg1[%2, %3, %arg14, %arg15]
          : memref<?x?x?x?xf32>
      }
    }
    gpu.terminator
    }
    return
  }
};
```

Figure 2 — Циклы с параметрическим тайлингом в MLIR CLT/Affine (листинг 1) и пример конверсии в GPU (листинг 2)

Низкоуровневый полиэдральный тайлинг

LLVM/Polly + Integer Set Library




```
1   // Полиэдральное представление
2   Statements {
3     Stmt_for_body3
4       Domain := { Stmt_for_body3[i0, i1] : 0 <= i0 <= 2047 and 0 <= i1 <= 2047 };
5       Schedule := { Stmt_for_body3[i0, i1] -> [0, i0, i1, 0, 0, 0, 0, 0] };
6       MustWriteAccess :=  [Reduction Type: NONE] [Scalar: 0]
7         { Stmt_for_body3[i0, i1] -> MemRef_C[i0, i1] };
8     Stmt_for_body8
9     Domain := { Stmt_for_body8[i0, i1, i2]
10      : 0 <= i0 <= 2047 and 0 <= i1 <= 2047 and 0 <= i2 <= 2047 };
11      Schedule := { Stmt_for_body8[i0, i1, i2] -> [1, o1, o2, o3, i0, i2, o6, i1]
12        : (o1)mod 128 = 0 and (o2) mod 128 = 0 and (o3) mod 128 = 0 and (o6)
13        mod 4 = 0 and -63 + i0 <= o1 <= i0 and -63 + i1 <= o2 <= i1
14        and -63 + i2 <= o3 <= i2 and -3 + i1 <= o6 <= i};
15      ReadAccess := [Reduction Type: NONE] [Scalar: 0]
16        { Stmt_for_body8[i0, i1, i2] -> MemRef_C[i0, i1] };
17      ReadAccess := [Reduction Type: NONE] [Scalar: 0]
18        { Stmt_for_body8[i0, i1, i2] -> MemRef_A[i0, i2] };
19      ReadAccess := [Reduction Type: NONE] [Scalar: 0]
20        { Stmt_for_body8[i0, i1, i2] -> MemRef_B[i2, i1] };
21      MustWriteAccess := [Reduction Type: NONE] [Scalar: 0]
22        { Stmt_for_body8[i0, i1, i2] -> MemRef_C[i0, i1] };}
23  // Генерация тайлового кода
24  for (int c1 = 0; c1 <= 2047; c1 += 1)
25    for (int c2 = 0; c2 <= 2047; c2 += 1)
26      Stmt_for_body3(c1, c2);
27    for (int c1 = 0; c1 <= 2047; c1 += 128)
28      for (int c2 = 0; c2 <= 2047; c2 += 128)
29        for (int c3 = 0; c3 <= 2047; c3 += 128)
30          for (int c4 = c1; c4 <= c1 + 63; c4 += 1)
31            for (int c5 = c3; c5 <= c3 + 63; c5 += 1)
32              for (int c6 = c2; c6 <= c2 + 63; c6 += 4)
33                for (int c7 = c6; c7 <= c6 + 3; c7 += 1)
34                  Stmt_for_body8(c4, c7, c5);
```

Figure 3 — Представление циклов в SCOP в LLVM-IR (листинг 1) и в полиэдральном описании с генерацией тайлового кода (листинг 2)

Экспериментальные результаты трансляции SCOP

Оценка эффективности операций

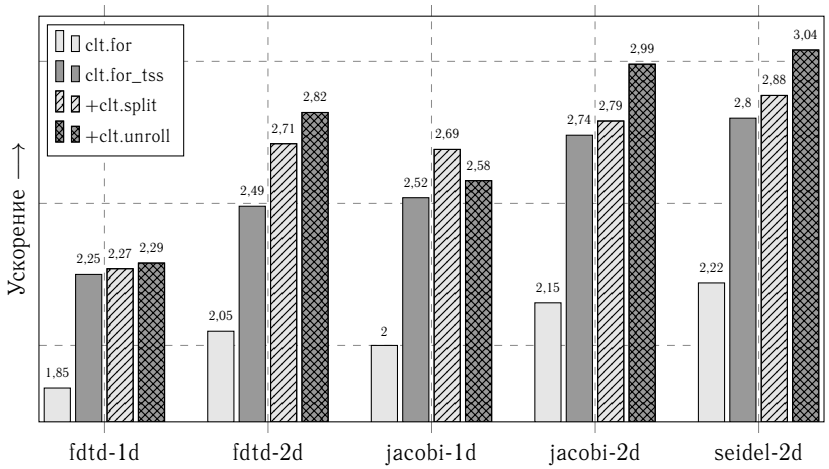


Figure 4 — CLT/Affine → GPU → NVGPU → NVVM-IR

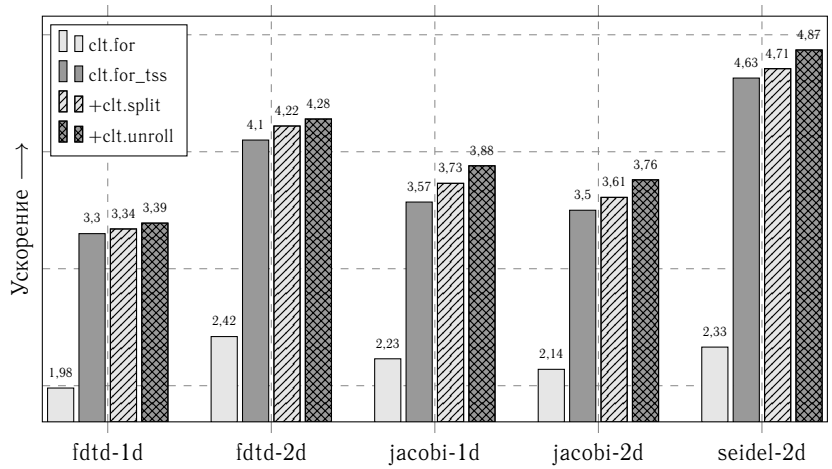


Figure 5 — CLT/Affine → LLVM → LLVM-IR → Polly + ISL

Figure 6 — Ускорение тайлового кода Stencils относительно версии NoOpt

- ftp://ftp.lto.ru/clt.pdf
- Gysi, Müller, Zinenko, Herhut, Davis, Wicky, Fuhrer, Hoefler, Grosser. Domain-Specific Multi-Level IR Rewriting for GPU: The Open Earth Compiler for GPU-accelerated Climate Simulation (2021)
- Moses, Chelini, Zhao, Zinenko. Polygeist: Raising C to Polyhedral MLIR (2021)