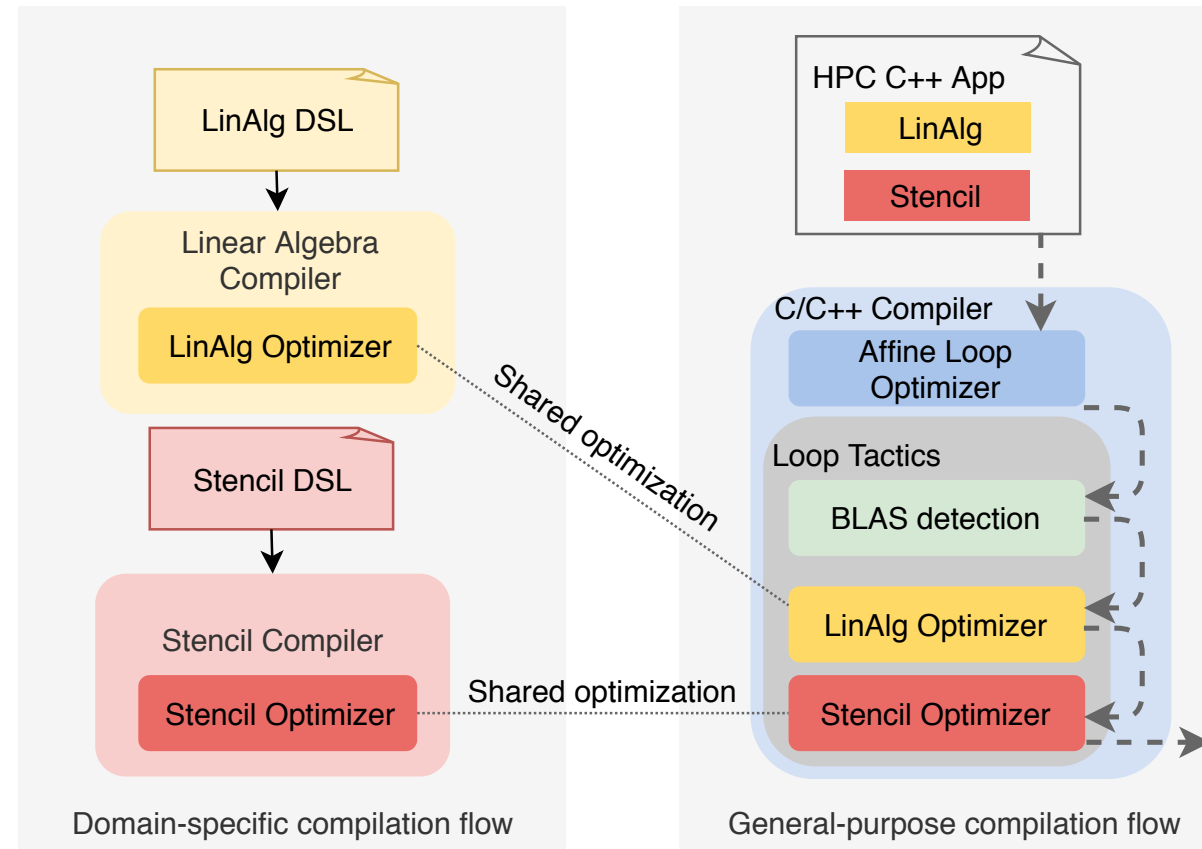
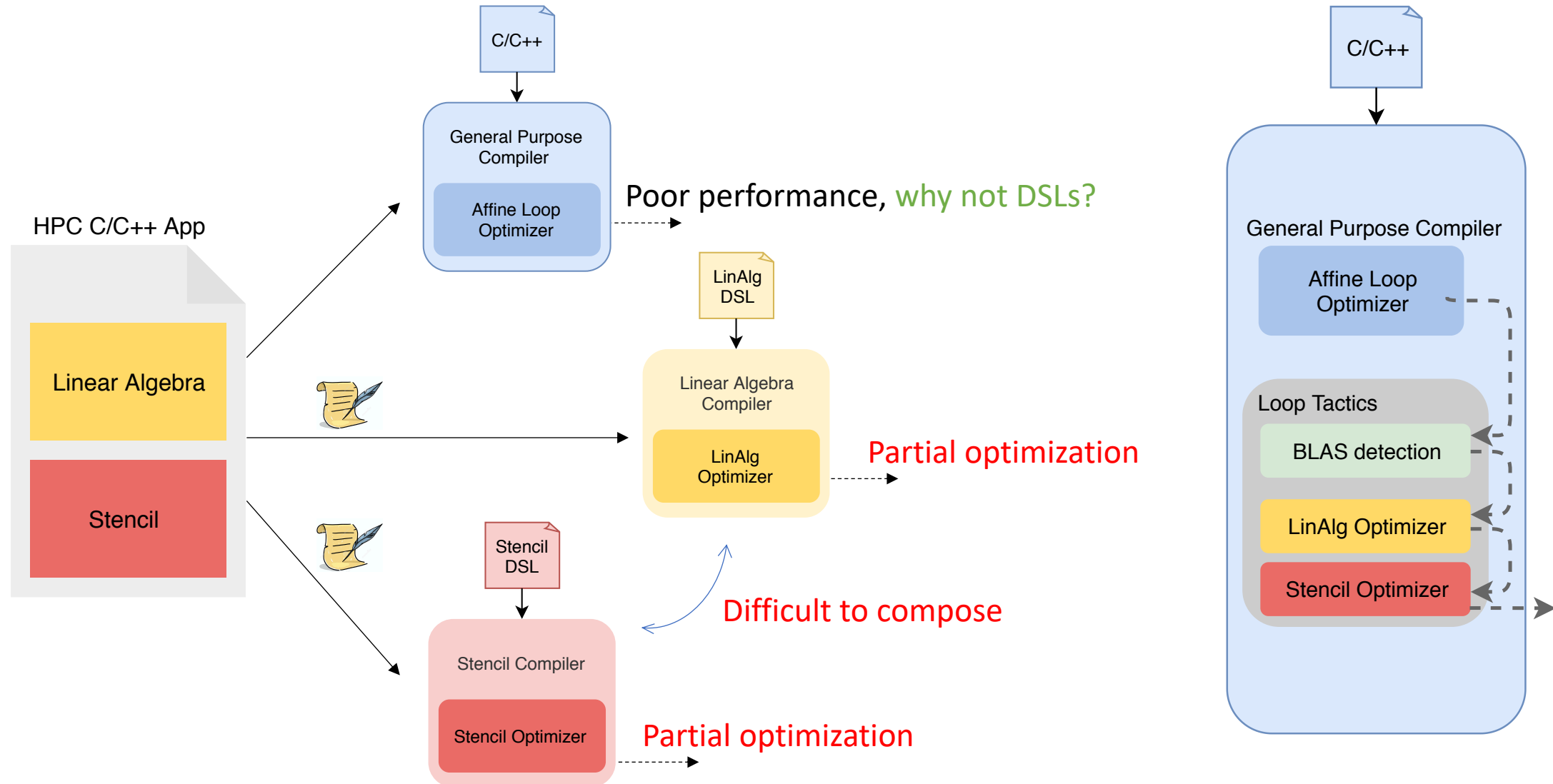


Declarative Loop Tactics for Domain-Specific Optimizations

Lorenzo Chelini, Oleksandr Zinenko, Tobias Grosser and Henk Corporaal

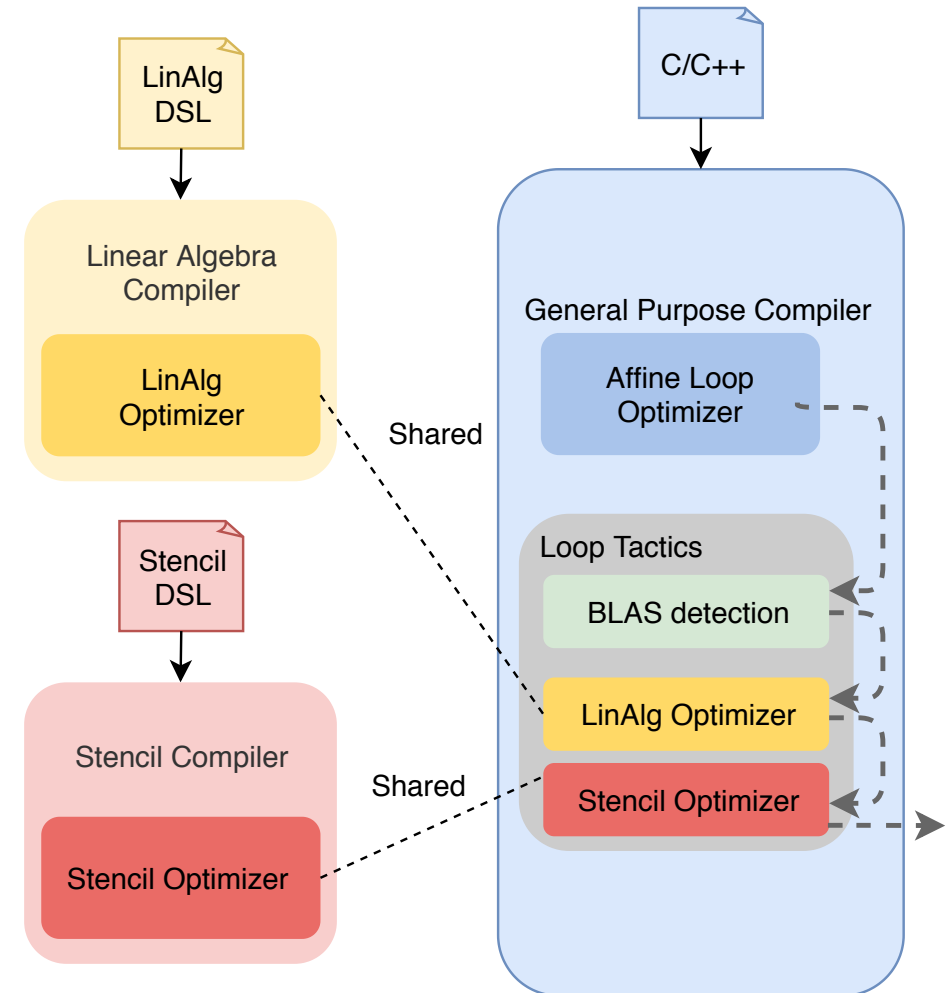


Optimizing multi-domain applications

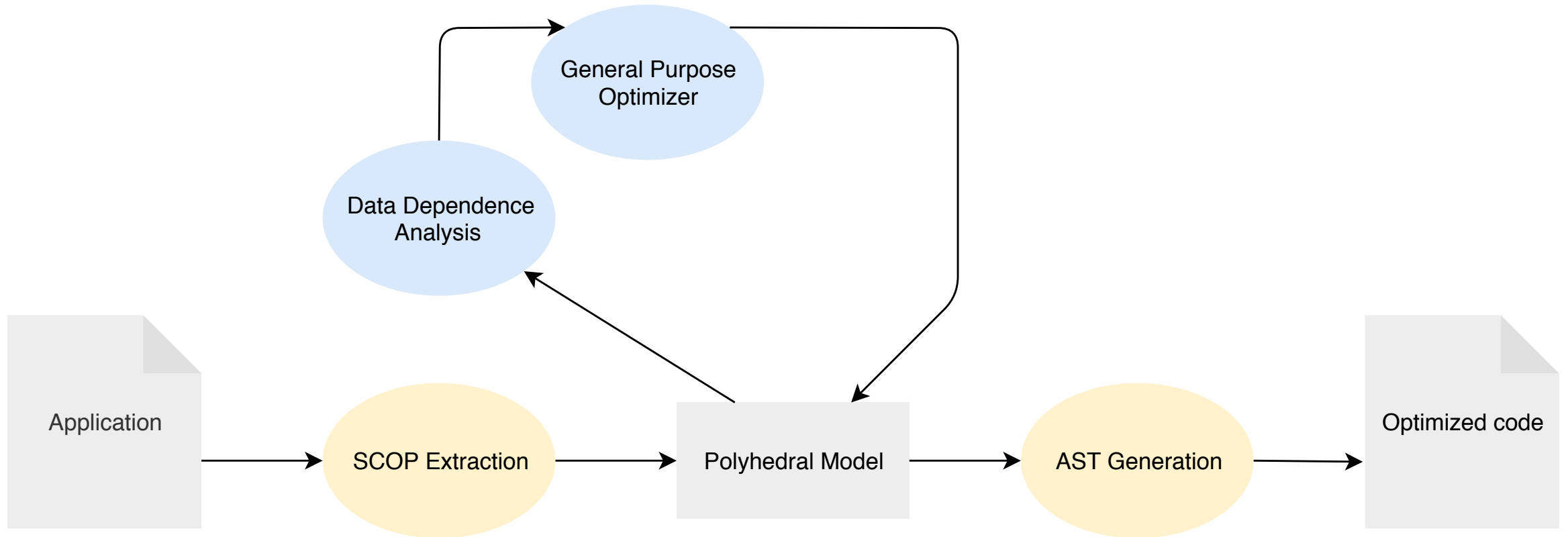


Loop tactics – why are they needed?

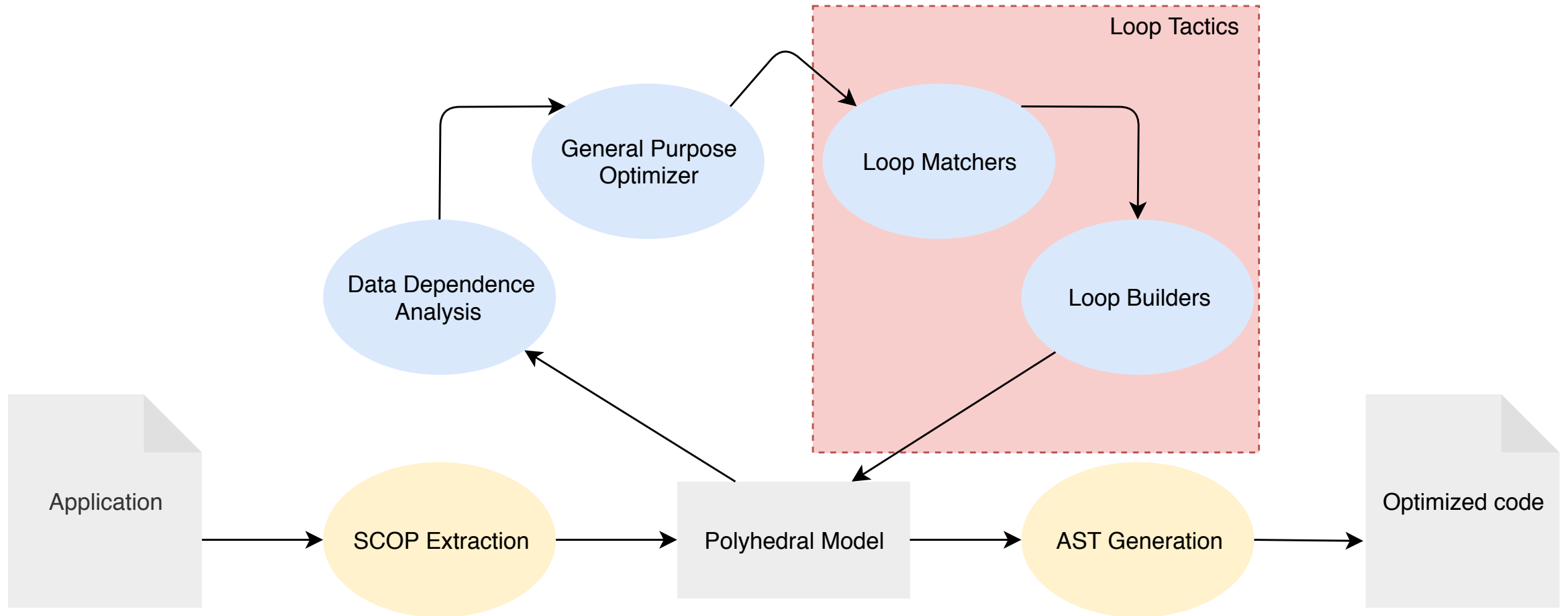
- Share domain-specific optimizations
- Optimize multi-domain applications
- Reduce the need for DSLs
- Transparent use of high-performance routines



Loop tactics in the compilation flow



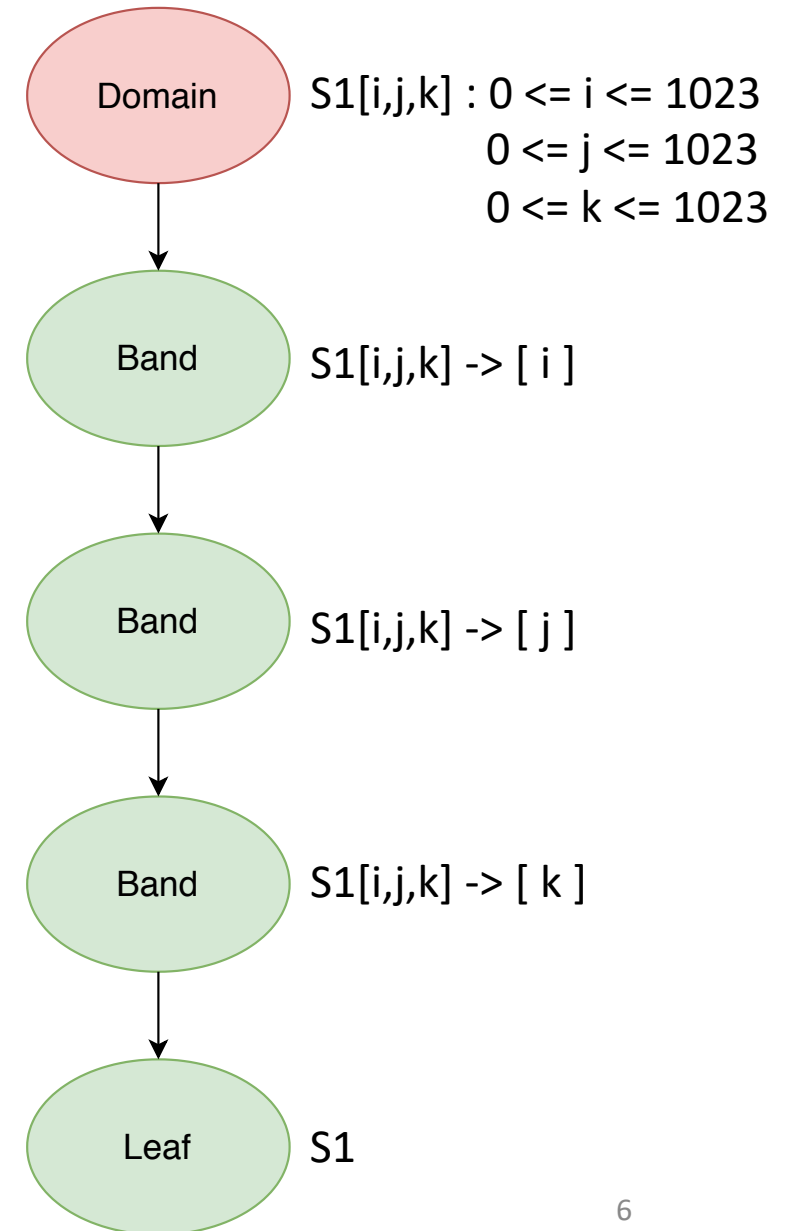
Loop tactics in the compilation flow



Tree-like internal representation

```
int C[1024][1024];
int A[1024][1024];
int B[1024][1024];

void matmul()
{
    for (int i = 0; i < 1024; i++)
        for (int j = 0; j < 1024; j++)
            for (int k = 0; k < 1024; k++)
S1:        C[i][j] += A[i][k] * B[k][j];
}
```



A 3-step approach to optimization

```
int C[1024][1024];
int A[1024][1024];
int B[1024][1024];

void matmul()
{
    for (int i = 0; i < 1024; i++)
        for (int j = 0; j < 1024; j++)
            for (int k = 0; k < 1024; k++)
S1:                C[i][j] += A[i][k] * B[k][j];
}
```

1. Structure

- 3 nested loops

2. Access Patterns

- #Reads ≥ 3
- Reads to different arrays
- #Writes ≥ 1
- Satisfy placeholder $[i,j] \rightarrow [i,k][k,j]$

3. Transformations – GotoBLAS

1. Interchange
2. Tile loop
3. Tile register
4. Data-layout

1. Structural matchers

```
int C[1024][1024];  
int A[1024][1024];  
int B[1024][1024];
```

```
void matmul()  
{
```

```
    for (int i = 0; i < 1024; i++)  
        for (int j = 0; j < 1024; j++)  
            for (int k = 0; k < 1024; k++)  
S1:         C[i][j] += A[i][k] * B[k][j];  
}
```

```
/* Captured node */  
isl::schedule_node stmt;
```

```
/* Structural matcher */  
auto matcher = [&]() {  
    using namespace matchers;  
    return
```

```
        band(has3Dimension, isPermutable  
              leaf(stmt));  
    }();
```

Captured node for subsequent opt.

2. Access matchers

```
int C[1024][1024];
int A[1024][1024];
int B[1024][1024];

void matmul()
{
    for (int i = 0; i < 1024; i++)
        for (int j = 0; j < 1024; j++)
            for (int k = 0; k < 1024; k++)
S1:         C[i][j] += A[i][k] * B[k][j];
}
```

```
/* Extract accesses from "stmt"*/
```

```
auto _i = placeholder(ctx);
auto _j = placeholder(ctx);
auto _k = placeholder(ctx);
auto _A = arrayPlaceholder();
auto _B = arrayPlaceholder();
auto _C = arrayPlaceholder();
```

```
auto psRead = allOf(access(_A, _i, _k),
                    access(_B, _k, _j),
                    access(_C, _i, _j));
```

```
auto psWrite = allOf(access(_C, _i, _j));
```

```
return match(reads, psRead).size() == 1
        && match(writes, psWrite).size() == 1
```

3. Transformation builders

```
int C[1024][1024];
int A[1024][1024];
int B[1024][1024];

void matmul()
{
    for (int i = 0; i < 1024; i++)
        for (int j = 0; j < 1024; j++)
            for (int k = 0; k < 1024; k++)
S1:        C[i][j] += A[i][k] * B[k][j];
}
```

```
/* Captured node */
isl::schedule_node node;

/* Tile sizes */
auto sizes = {32, 32, 32};

auto builder =
    band([&]() {return tileSchedule(node, sizes); },
        band([&]() {return pointSchedule(node, sizes);
    }));
```

Code generation

```
/* Tiled GEMM */
for(int ii=0; ii<1024; ii+=32)
    for(int jj=0; jj<1024; jj+=32)
        for(int kk=0; kk<1024; kk+=32)
            for(int i=ii; i<ii+32; ++i)
                for(int j=jj; j<jj+32; ++j)
                    for(int k=kk; k<kk+32; ++k)
                        /* matmul */
```

Tile loops {

Point loops {

Complex builder: The “true” GEMM optimization

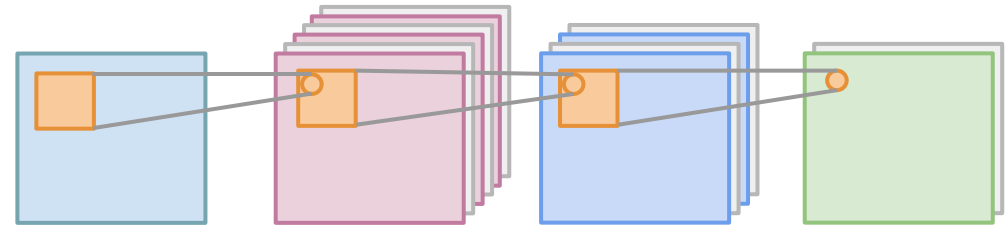
```
// Inter iteration alias-free
// 1st level tiling - Tiles
{
for (int c3 = 0; c3 <= 255; c3 += 1)
    // Data layout - packing transformation
    for (int c4 = 0; c4 <= 255; c4 += 1)
        CopyStmt_0(0, c3, c4);
        for (int c2 = 0; c2 <= 3; c2 += 1) {
            for (int c3 = 64 * c2; c3 <= 64 * c2 + 63; c3 += 1)
                // Data layout - packing transformation
                for (int c5 = 0; c5 <= 255; c5 += 1)
                    CopyStmt_1(c3, 0, c5);
                    // 1st level tiling - Points
                    // Register tiling - Tiles
                    for (int c3 = 0; c3 <= 63; c3 += 1)
                        for (int c4 = 0; c4 <= 15; c4 += 1)
                            ...
                        }
                    }
            }
        }
}
```

```
for (int c5 = 0; c5 <= 255; c5 += 1) {
    // Loop Vectorizer Disabled
    // Register tiling - Points
    {
        Stmt12(64 * c2 + 4 * c4, 4 * c3, c5);
        Stmt12(64 * c2 + 4 * c4, 4 * c3 + 1, c5);
        Stmt12(64 * c2 + 4 * c4, 4 * c3 + 2, c5);
        Stmt12(64 * c2 + 4 * c4, 4 * c3 + 3, c5);
        Stmt12(64 * c2 + 4 * c4 + 1, 4 * c3, c5);
        Stmt12(64 * c2 + 4 * c4 + 1, 4 * c3 + 1, c5);
        Stmt12(64 * c2 + 4 * c4 + 1, 4 * c3 + 2, c5);
        Stmt12(64 * c2 + 4 * c4 + 1, 4 * c3 + 3, c5);
        Stmt12(64 * c2 + 4 * c4 + 2, 4 * c3, c5);
        Stmt12(64 * c2 + 4 * c4 + 2, 4 * c3 + 1, c5);
        Stmt12(64 * c2 + 4 * c4 + 2, 4 * c3 + 2, c5);
        Stmt12(64 * c2 + 4 * c4 + 2, 4 * c3 + 3, c5);
        Stmt12(64 * c2 + 4 * c4 + 3, 4 * c3, c5);
        Stmt12(64 * c2 + 4 * c4 + 3, 4 * c3 + 1, c5);
        Stmt12(64 * c2 + 4 * c4 + 3, 4 * c3 + 2, c5);
        Stmt12(64 * c2 + 4 * c4 + 3, 4 * c3 + 3, c5);
    }
}
```

```
else { /* original code */ }
```

Application case studies

- Hand-tuned GEMM
- Short-SIMD stencil vectorization
- Transparent library invocation



$$\begin{bmatrix} A & C \\ B & D \end{bmatrix} \times \begin{bmatrix} E \\ F \end{bmatrix} = \begin{bmatrix} A * E + C * F \\ B * E + D * F \end{bmatrix}$$

cuBLAS



Application case studies

- Hand-tuned GEMM
- Short-SIMD stencil vectorization
- Transparent library invocation

How pattern matching is done today? Custom pattern matching (i.e., Polly)

```
bool ScheduleTreeOptimizer::isMatrMultPattern(isl::schedule_node Node,
                                              const Dependencies *D,
                                              MatMulInfoTy &MMI) {
    auto PartialSchedule = isl::manage(
        isl_schedule_node_band_get_partial_schedule(Node));
    Node = Node.child(0);
    auto LeafType = isl_schedule_node_get_type(Node);
    Node = Node.parent();
    if (LeafType != isl_schedule_node_leaf ||
        isl_schedule_node_band_n_member(Node.get()) != 0 ||
        Node.get_schedule_depth() != 0 ||
        isl_union_map_n_map(PartialSchedule.get()) != 1)
        return false;
    auto NewPartialSchedule = isl::map::from_union_map(PartialSchedule);
    if (containsMatrMult(NewPartialSchedule, D, MMI))
        return true;
    return false;
}

static bool isMatMulOperandAcc(isl::set Domain,
                              int &FirstPos,
                              int &SecondPos) {
    // More Code

    int FirstDims[] = {0, 0, 1, 1, 2, 2};
    int SecondDims[] = {1, 2, 2, 0, 0, 1};
    for (int i = 0; i < 6; i += 1) {
        auto PossibleMatMul =
            Universe.equate(isl::dim::in, FirstDims[i], isl::dim::out, 0)
                .equate(isl::dim::in, SecondDims[i], isl::dim::out, 1);

        AccMap = AccMap.intersect_domain(Domain);
        PossibleMatMul = PossibleMatMul.intersect_domain(Domain);

        // More Code
    }
}
```

Complex code !!

Optimization is still missing !!

```
static bool containsMatrMult(isl::map PartialSchedule, const Dependencies *D,
                             MatMulInfoTy &MMI) {
    auto InputDimsId = PartialSchedule.get_tuple_id(isl::dim::in);
    auto *Stmt = static_cast<ScopStmt *>(InputDimsId.get_user());
    if (Stmt->size() <= 1)
        return false;
    auto Accesses = getAccessesInOrder(*Stmt);
    for (auto *MemA = Accesses.end() - 1; MemA != Accesses.begin(); MemA--) {
        auto *MemAccessPtr = *MemA;
        if (MemAccessPtr->isLatestArrayKind())
            continue;
        if (MemAccessPtr->isWrite())
            continue;
        if (MemAccessPtr->getLatestAccessRelation() !=
            MemAccessPtr->getLatestAccessRelation())
            continue;
        if (!MemAccessPtr->isL0OperandAcc(Stmt->getDomain(), AccMap, MMI.i, MMI.j))
            continue;
        if (MemAccessPtr->isL0OperandAcc(Stmt->getDomain(), AccMap, MMI.i, MMI.j))
            return true;
        break;
    }
    if (!containsOnlyMatMulDep(PartialSchedule, D, MMI.k))
        return false;
    if (MMI.WriteToC || !containsOnlyMatrMultAcc(PartialSchedule, MMI))
        return false;
    if (MMI.B || !MMI.ReadFromC)
        return false;
    auto Dep = isl::union_map::from_union_map(PartialSchedule, const Dependencies *D,
                                              int &Pos) {
        isl::union_map Dep = D->getDependencies(Dependencies::TYPE_RAW);
        isl::union_map Red = D->getDependencies(Dependencies::TYPE_RED);
        if (Red)
            Dep = Dep.unite(Red);
        auto DomainSpace = Schedule.get_space().domain();
        auto Space = DomainSpace.map_from_domain_and_range(DomainSpace);
        auto Deltas = Dep.extract_map(Space).deltas();
        int DeltasDimNum = Deltas.dim(isl::dim::set);
        for (int i = 0; i < DeltasDimNum; i++) {
            auto Val = Deltas.plain_get_val_if_fixed(isl::dim::set, i);
            Pos = Pos < 0 && Val.is_one() ? i : Pos;
            if (Val.is_nan() || !(Val.is_zero() || (i == Pos && Val.is_one())))
                return false;
        }
        if (DeltasDimNum == 0 || Pos < 0)
            return false;
        return true;
    }
}
```

Custom pattern matching - observations

- Too much code
- Does not scale
- What about new patterns?
- *Can we do custom pattern matching in a more generic way?*

Pattern matching with loop tactics

```
for(int i = 0; i < 1024; ++i)
  for(int j = 0; j < 1024; ++j)
    for(int k = 0; k < 1024; ++k)
      C[i][j] += A[i][k] * B[k][j]
```

Structural matchers

```
/* Captured node */
isl::schedule_node stmt;

/* Structural matcher */
auto matcher = [&]() {
  using namespace matchers;
  return
    band(has3Dimension, isPermutable
          leaf(stmt));
}();
```

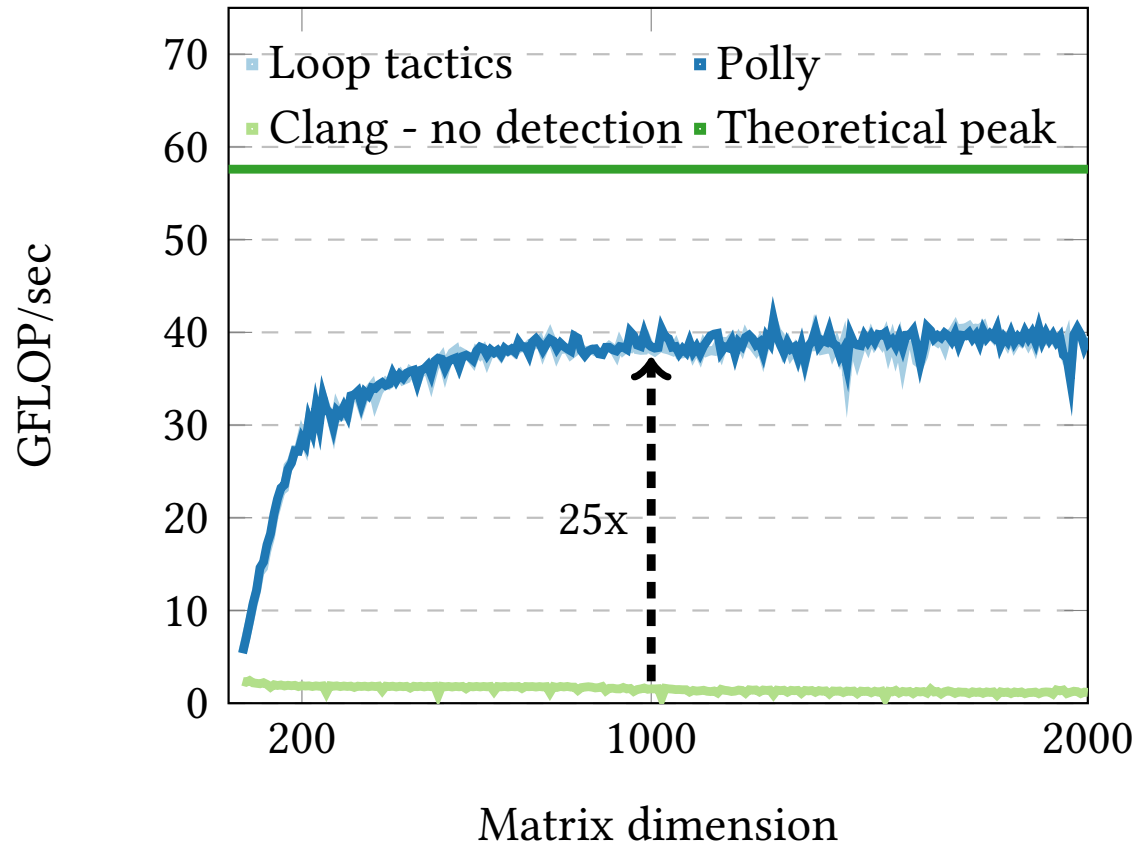
Access matchers

```
auto psRead =
  allOf(access(_C, _i, _j), access(_A, _i, _k), access(_B, _k, _j));
auto readMatches = match(reads, psRead);
auto psWrite = allOf(access(_C, _i, _j));
auto writeMatches = match(writes, psWrite);
/* Additional checks */
```

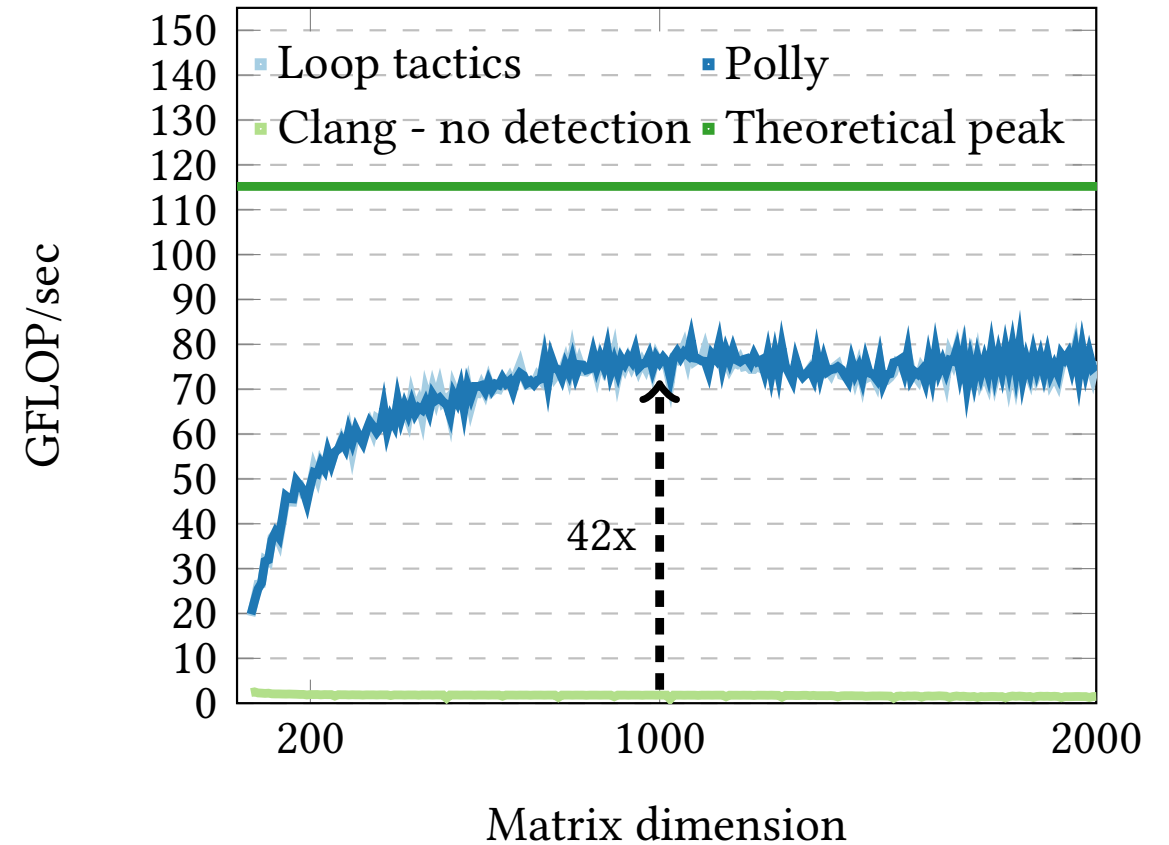
Transform !!

How good are loop tactics: GEMM

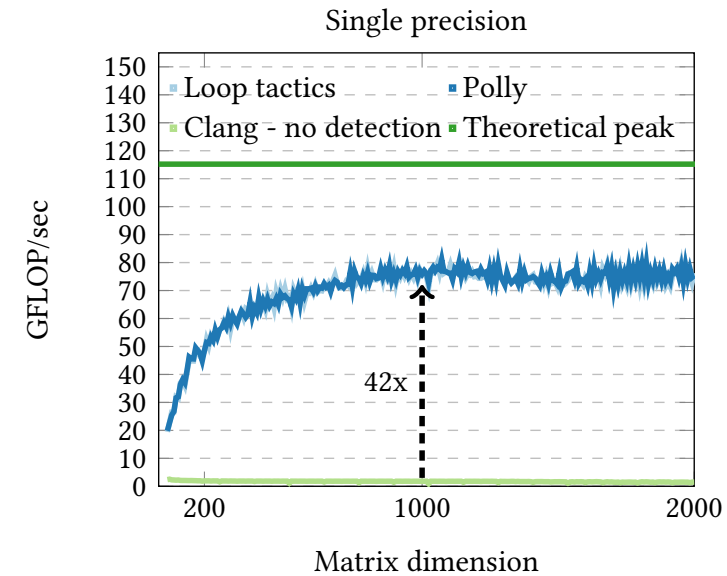
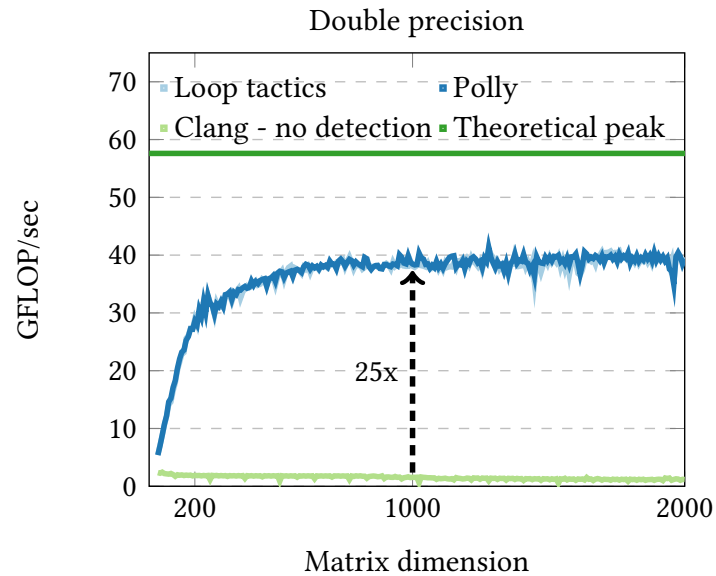
Double precision



Single precision



How good are loop tactics: GEMM



Loop Tactics - 234 LOC



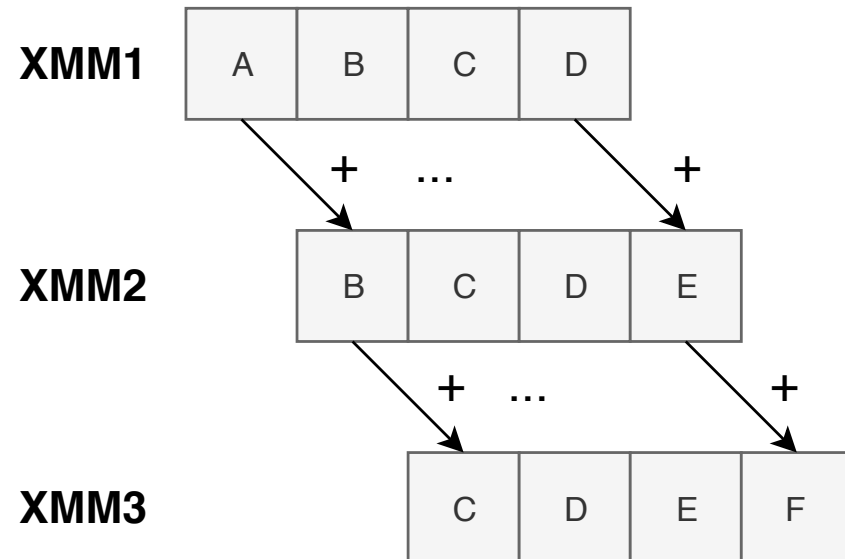
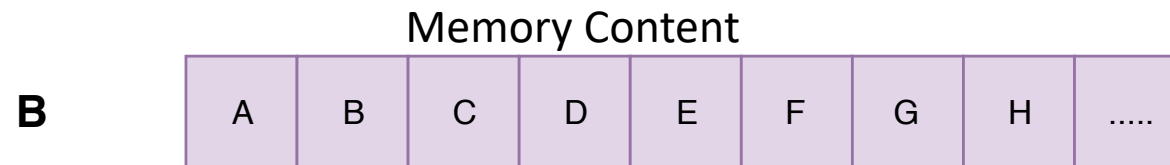
Polly's transformation - 410 LOC



Detection Interchange
Macro-kernel Micro-kernel

Problem with stencil vectorization

```
for(int i = 0; i < T; ++i)
  for(int j = 0; j < W - 1; ++j)
    A[i][j] = B[i][j-1] + B[i][j] + B[i][j+1];
```



```
movaps B(..), %xmm1
movaps 4+B(..), %xmm2
movaps 8+B(..), %xmm3
;; Register state here
addps %xmm1, %xmm2
addps %xmm2, %xmm3
```

Solution - DLT for stencils

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Original layout

A	B	C	D	E	F
G	H	I	J	K	L
M	N	O	P	Q	R
S	T	U	V	W	X

Dimension Lifted

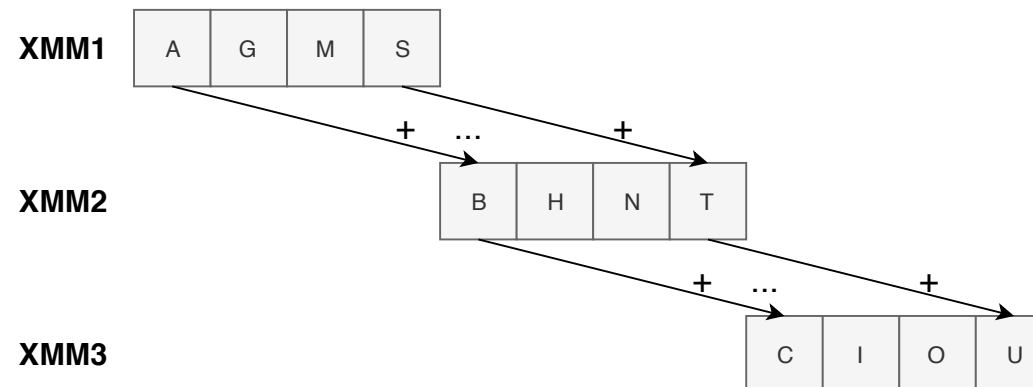
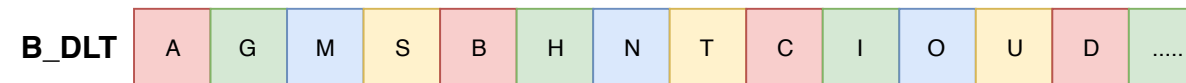
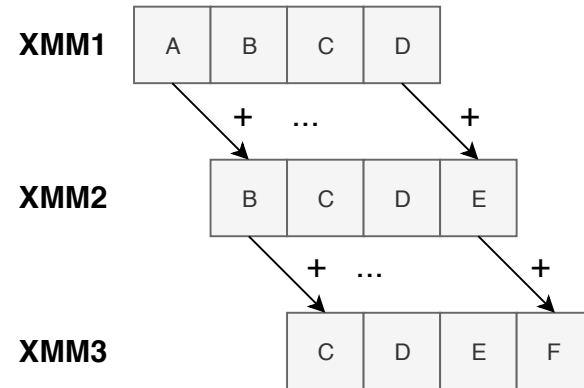
A	G	M	S
B	H	N	T
C	I	O	U
D	J	P	V
E	K	Q	W
F	L	R	X

Transposed

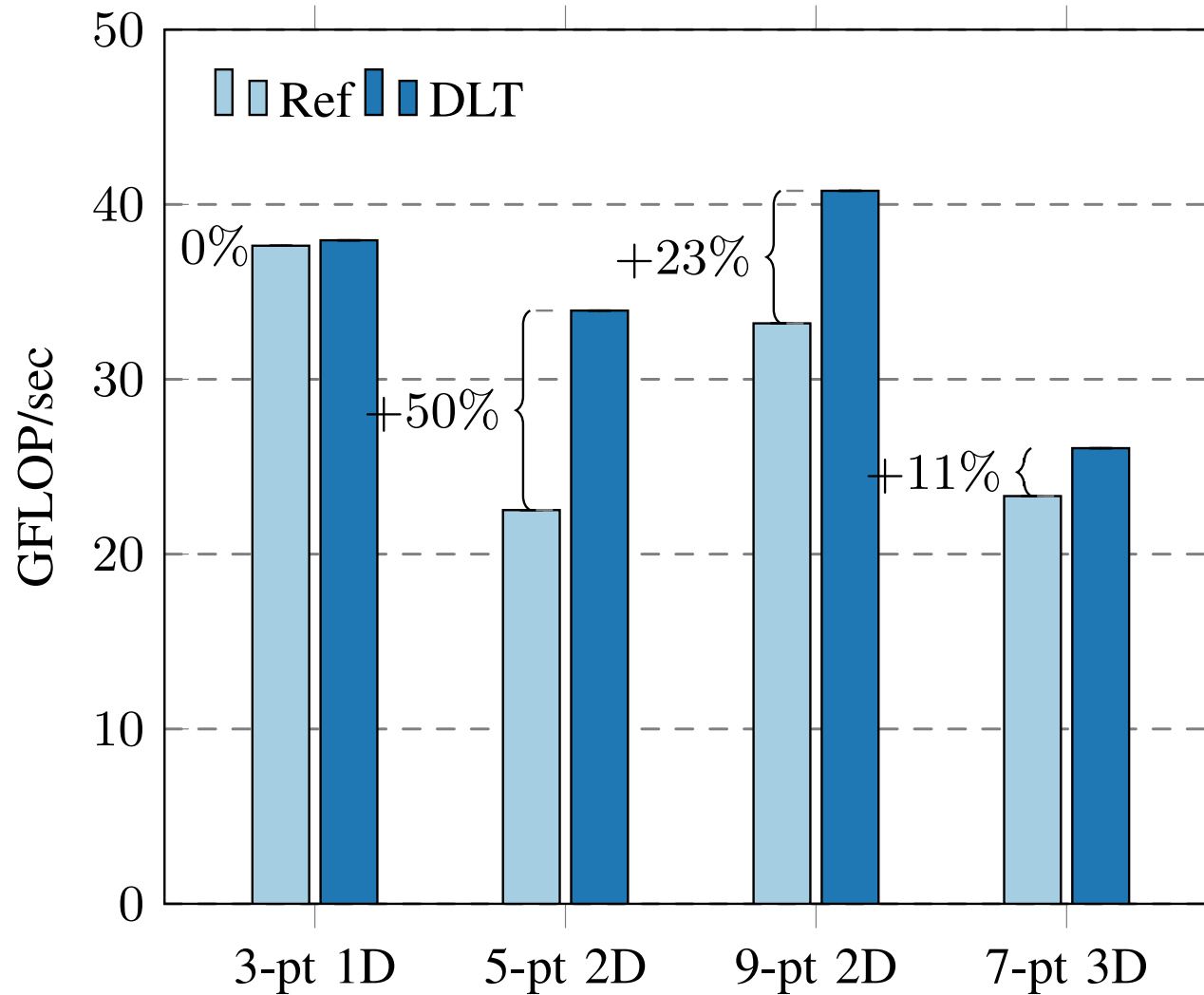
A	G	M	S	B	H	N	T	C	I	O	U	D	J	P	V	E	K	Q	W	F	L	R	X
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Transformed layout

Solution - DLT for stencils



Stencils with loop tactics



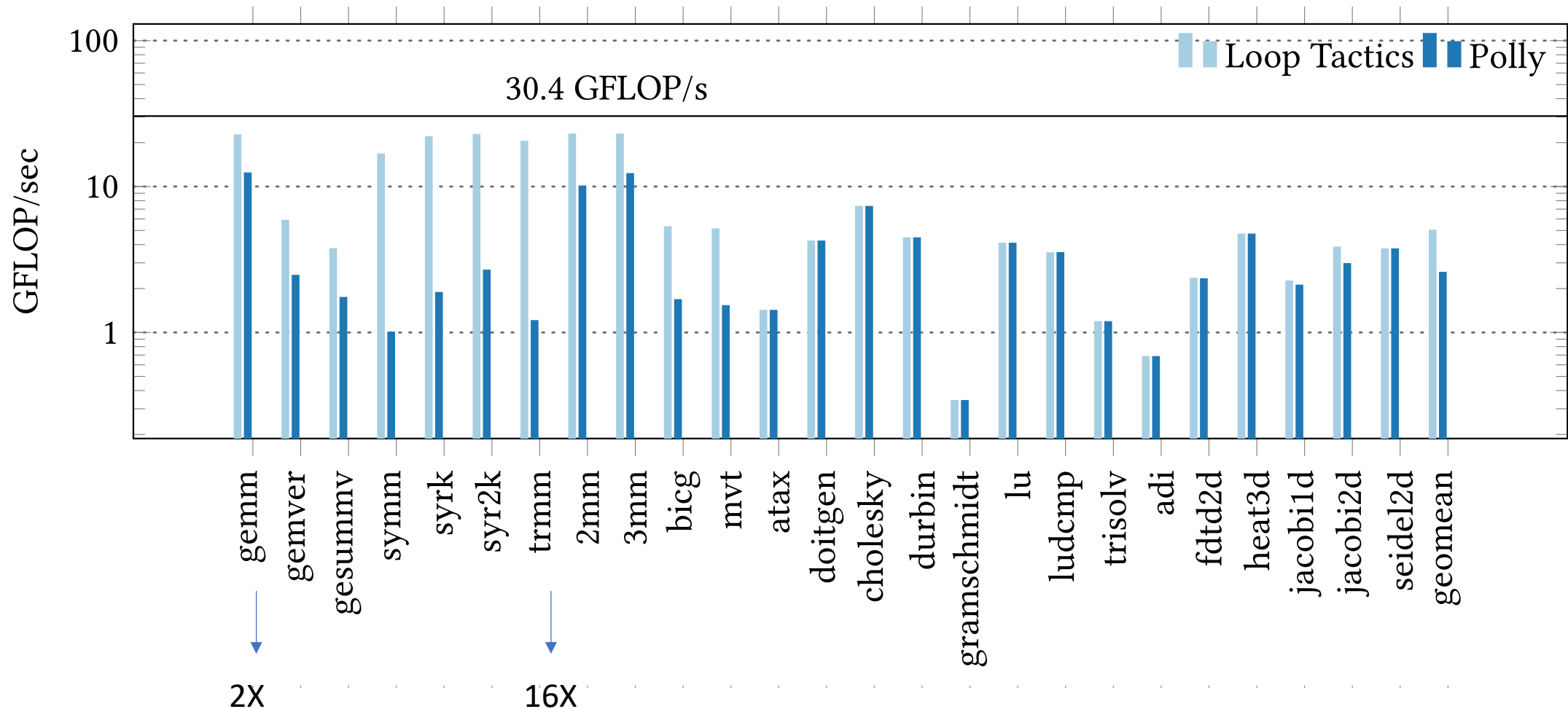
Stencils with loop tactics – output code

```
for(int i = 0; i < 24; ++i)
    B[i] = (A[i-1] + A[i] + A[i+1])
```

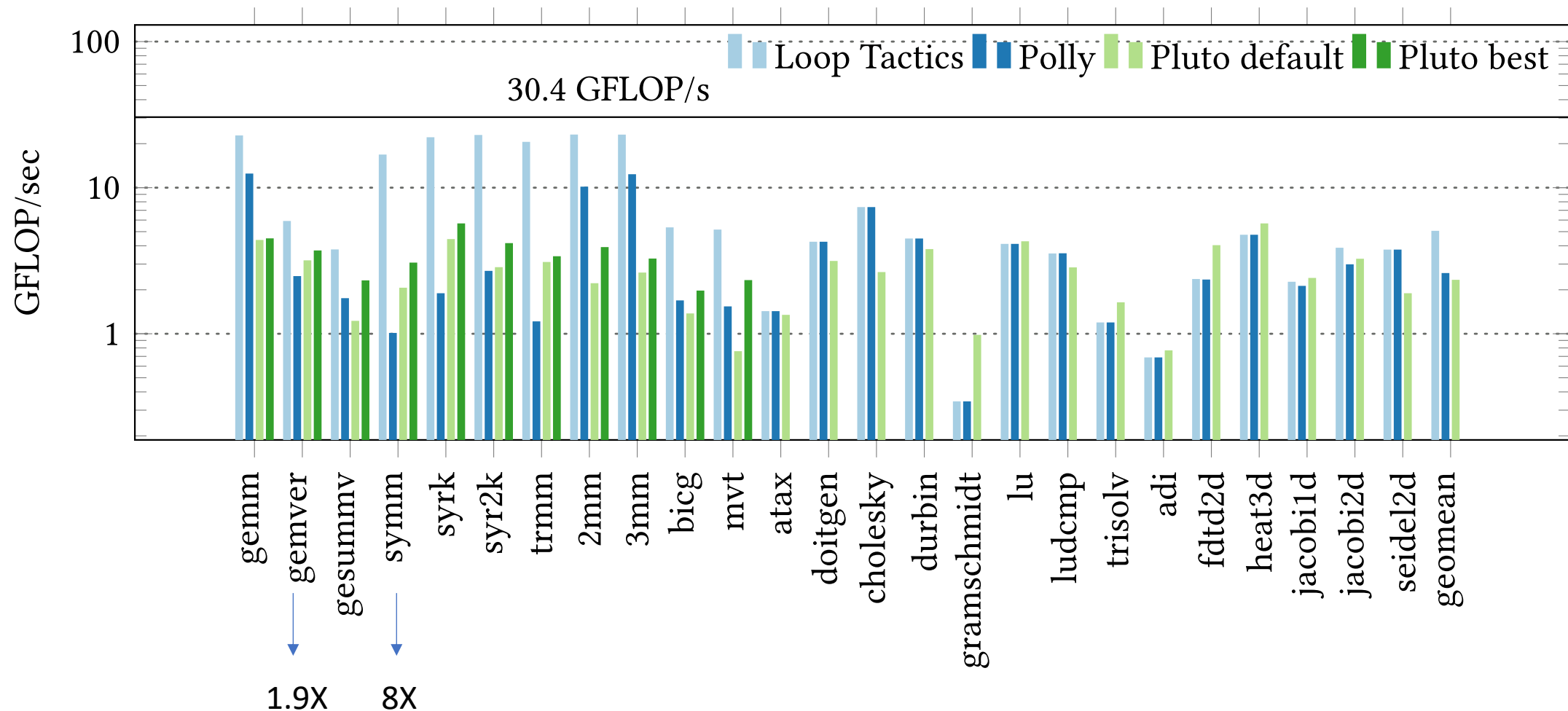
```
/* Original computation */
vmovups  A(,%rdx,4)}, %ymm7
vmovups  32+A(,%rdx,4)}, %ymm11
vaddps   4+A(,%rdx,4)}, %ymm7, %ymm8
vaddps   36+A(,%rdx,4)}, %ymm11, %ymm12
vaddps   8+A(,%rdx,4)}, %ymm8, %ymm9
vaddps   40+A(,%rdx,4)}, %ymm12, %ymm13
vmulps   %ymm9, %ymm5, %ymm10
vmulps   %ymm13, %ymm5, %ymm14
vmovups   %ymm10, 4+B(,%rdx,4)
vmovups   %ymm14, 36+B(,%rdx,4)
/* More code */
```

```
/* Computation after data-layout transformation */
vmovups   32+A_DLT(,%rdx,4)}, %ymm7
vmovups   64+A_DLT(,%rdx,4)}, %ymm11
vmovups   96+A_DLT(,%rdx,4)}, %ymm15
vaddps   A_DLT(,%rdx,4)}, %ymm7, %ymm4
vaddps   %ymm11, %ymm7, %ymm8
vaddps   %ymm11, %ymm4, %ymm5
vaddps   %ymm15, %ymm8, %ymm9
vmulps   %ymm5, %ymm3, %ymm6
vmulps   %ymm9, %ymm3, %ymm10
vmovups   %ymm6, 4+B_DLT(,%rdx,4)
vmovups   %ymm10, 36+B_DLT(,%rdx,4)
/* More code */
```

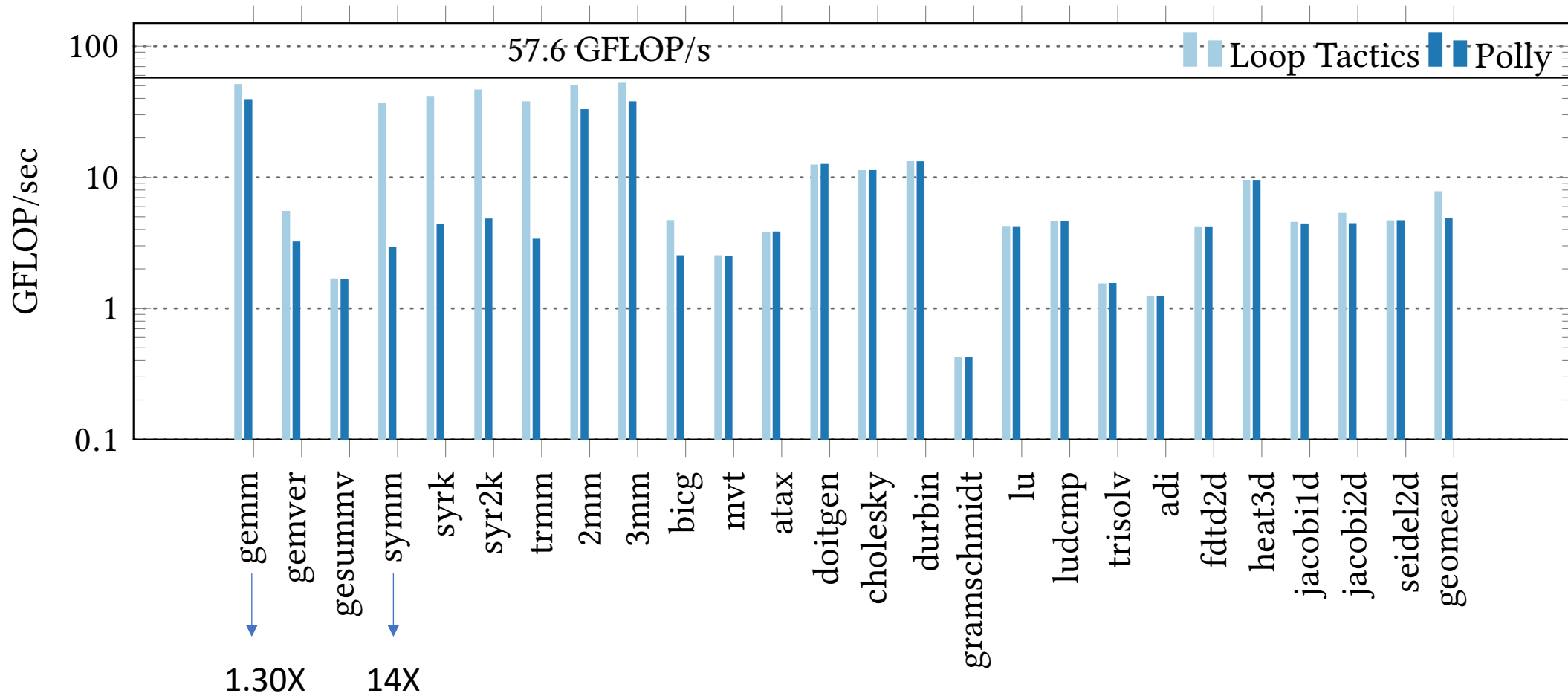
Transparent BLAS optimization with loop tactics on Power9 – comparison with Polly



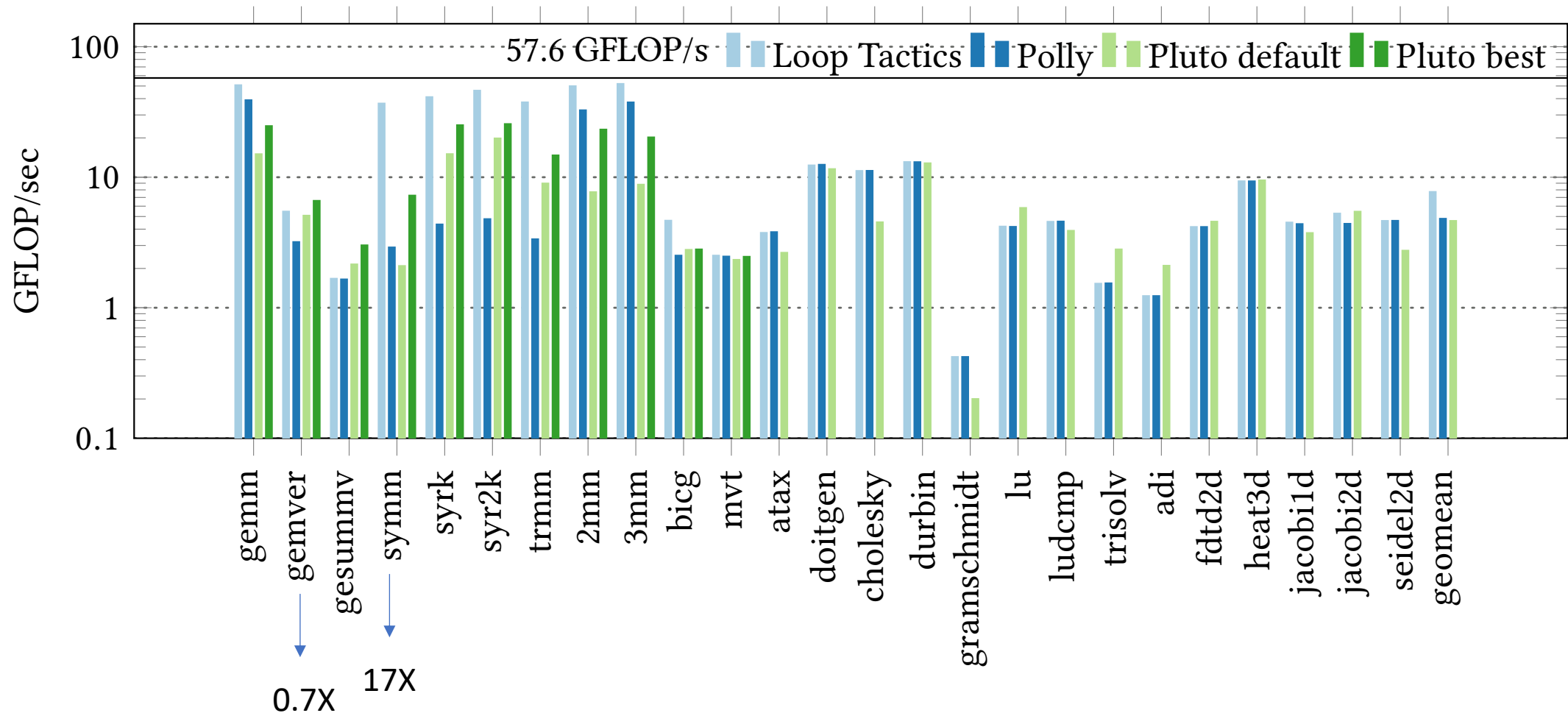
Transparent BLAS optimization with loop tactics on Power9 – also comparison with Pluto



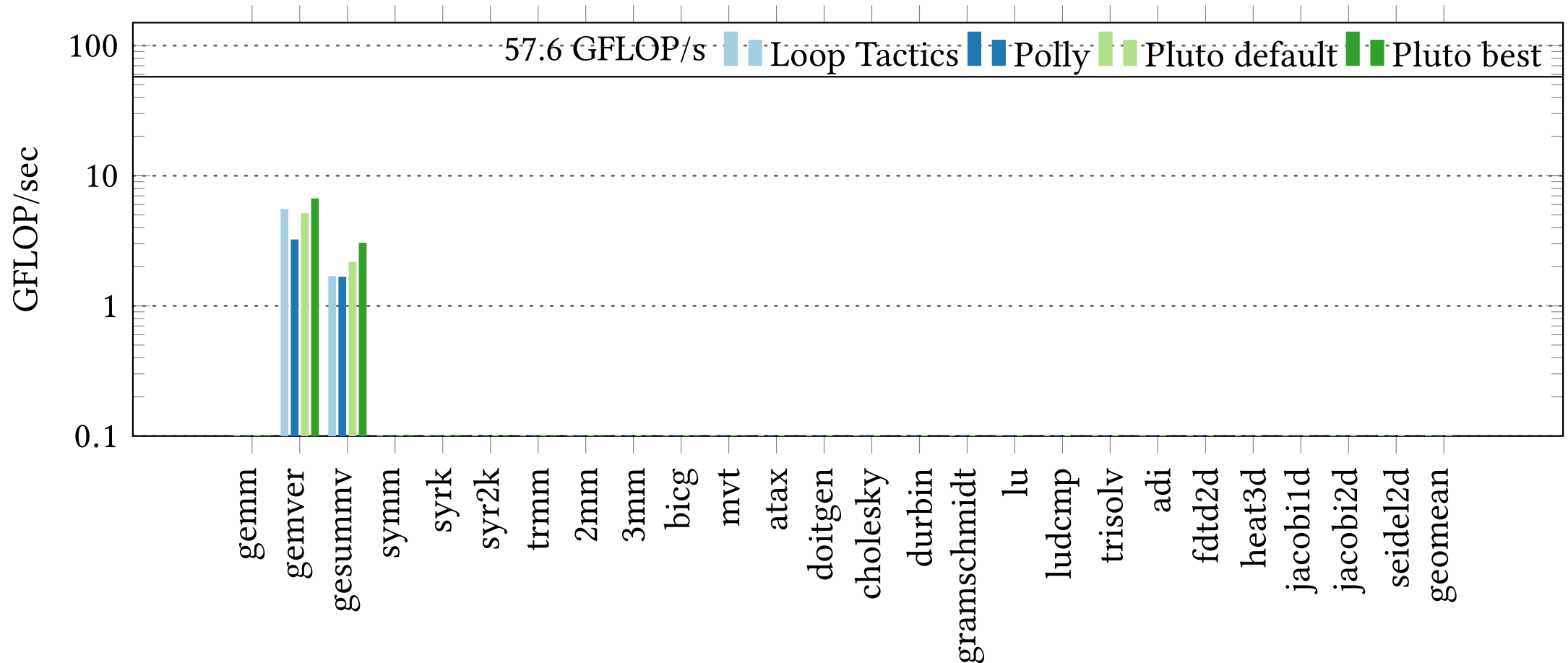
Transparent BLAS optimization with loop tactics on Intel i7 – comparison with Polly



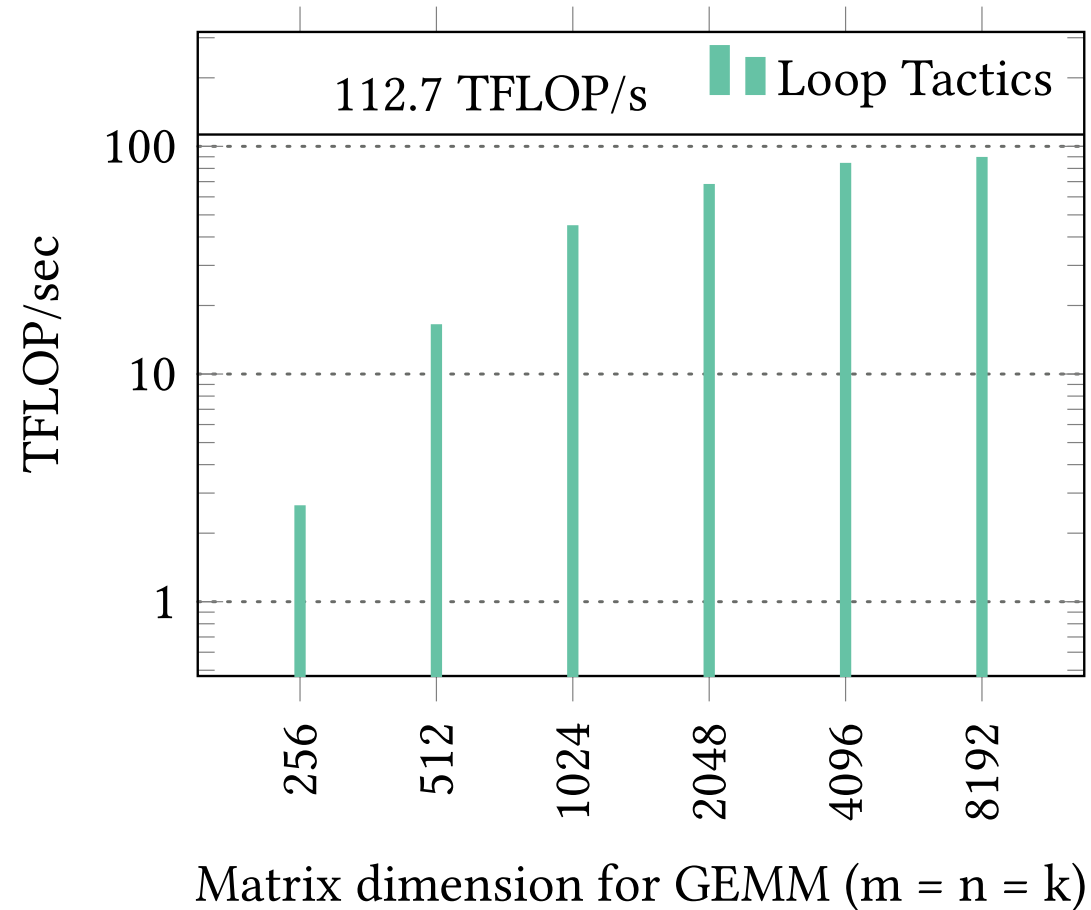
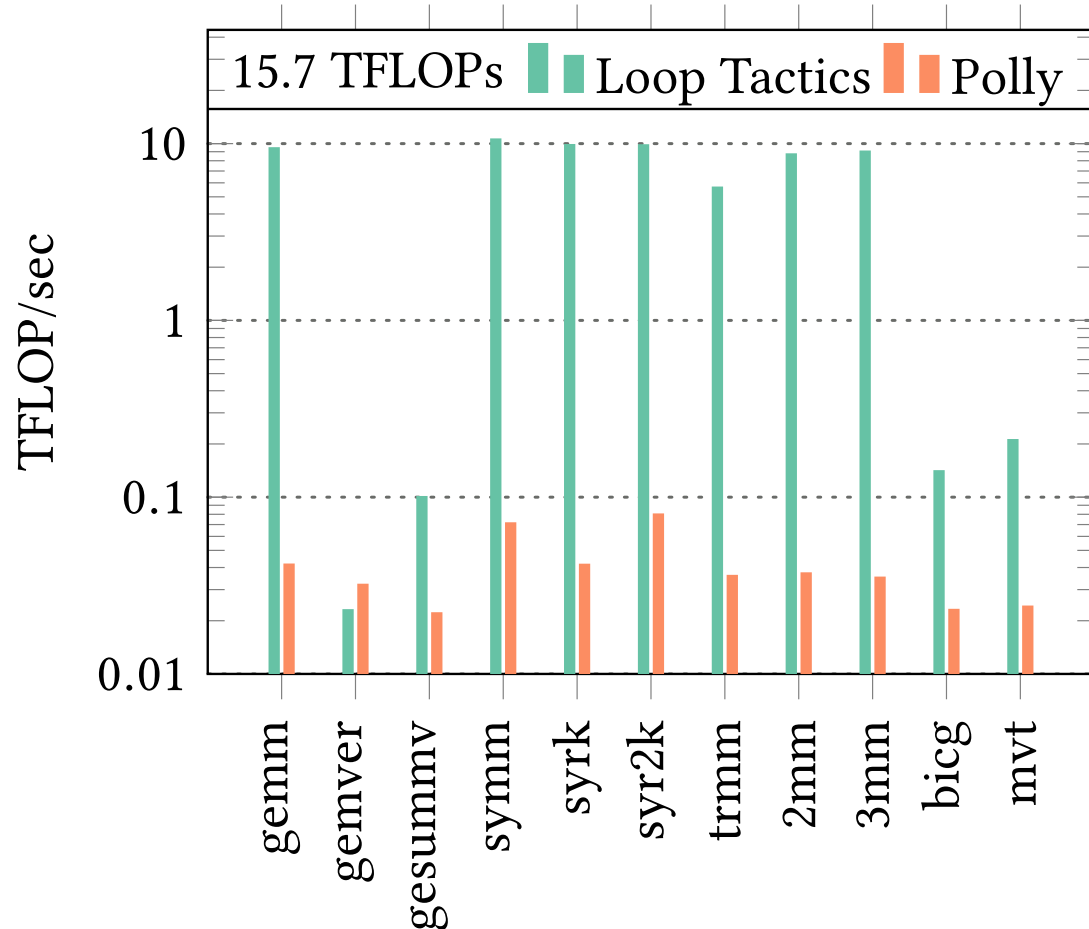
Transparent BLAS optimization with loop tactics on Intel i7 – also comparison with Pluto



Transparent BLAS optimization with loop tactics on Intel i7 – also comparison with Pluto



Transparent BLAS optimization with loop tactics on GPU Volta V100

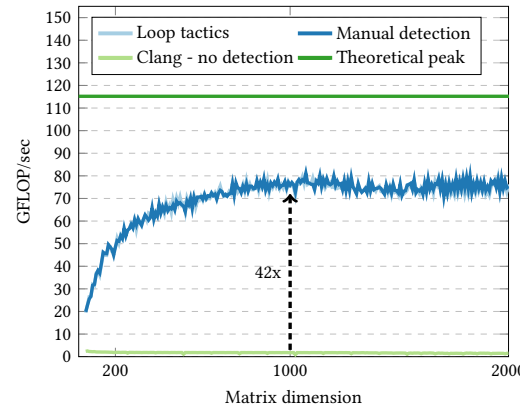
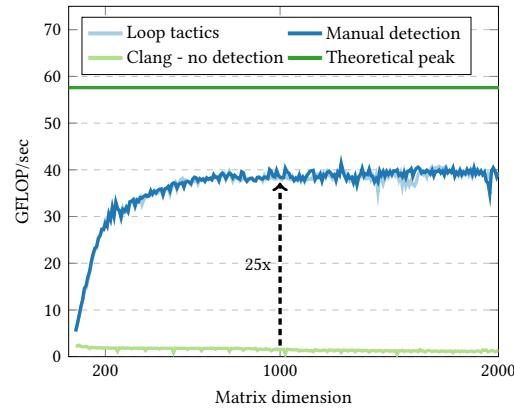


Transparent BLAS optimization - matchers composability

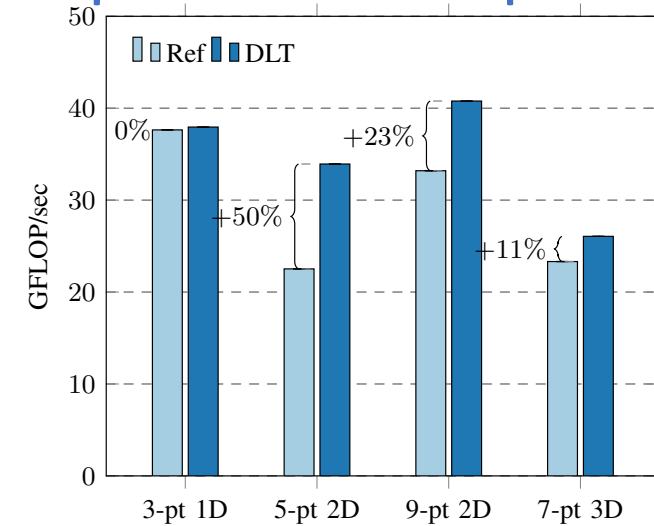
```
auto isGemm/*isSyrk*/Like = [&](schedule_node node) {  
    auto gemm/*syrk*/Core =  
        band(and_(is3D, isPermutable, hasGemm/*hasSyrk*/Pattern), leaf());  
    auto gemm/*syrk*/Init =  
        band(and_(is2D, hasDescendant(gemm/*syrk*/Core), has2DInitPattern),  
            anyTree());  
    return isMatching(node, gemm/*syrk*/Core) || isMatching(node, gemm/*syrk*/Init);  
};
```

```
auto hasGemmPattern = [&](schedule_node node)  
{  
    auto _i = placeholder();  
    auto _j = placeholder();  
    auto _k = placeholder();  
    auto _A = arrayPlaceholder();  
    auto _B = arrayPlaceholder();  
    auto _C = arrayPlaceholder();  
    auto reads = /* get read accesses */;  
    auto writes = /* get write accesses */;  
    auto mRead =  
        allOf(access(_C, _i, _j),  
              access(_A, _i, _k),  
              // for syrk use (_A, _j, _k)  
              access(_B, _k, _j));  
    auto mWrite = allOf(access(_C, _i, _j));  
    return match(reads, mRead).size() == 1 &&  
        match(writes, mWrite).size() == 1;  
};
```

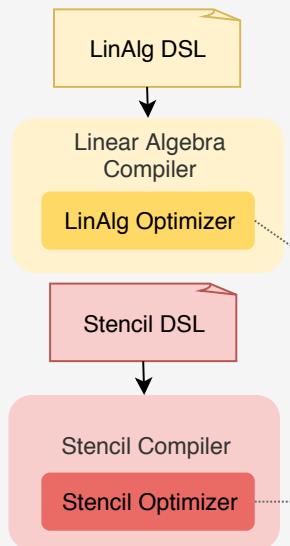
“Loop tactics allows to express domain-specific optimization *directly* in general-purpose compilers”



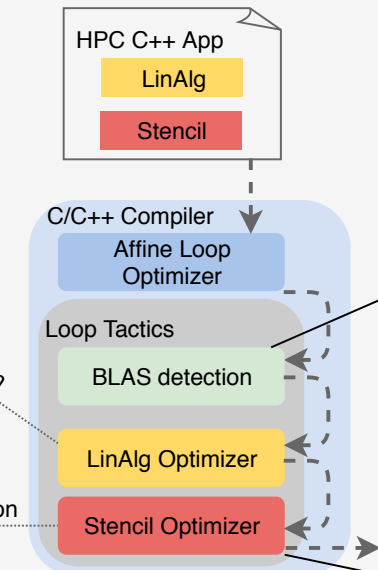
Linear algebra domain



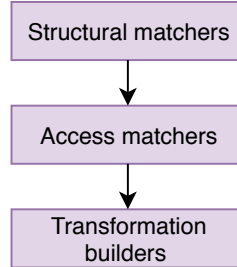
Stencil domain



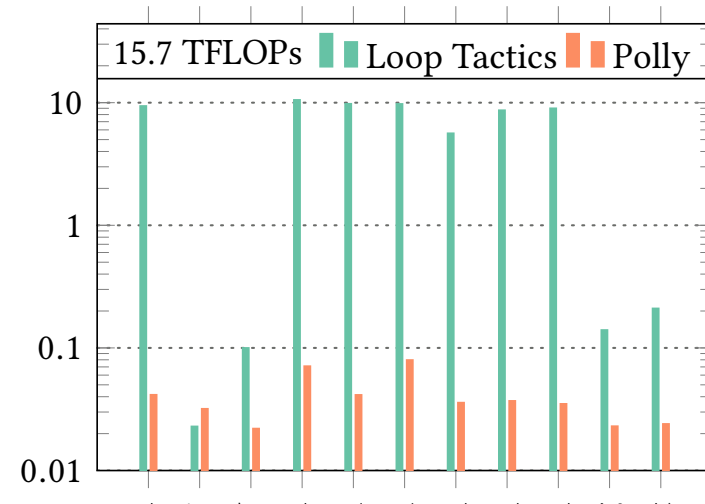
Domain-specific compilation flow



General-purpose compilation flow



TFLOP/sec



Transparent BLAS Opt.