U.S. DEPARTMENT OF ENERGY

ECP EXASCALE COMPUTING PROJECT

Argonne NATIONAL LABORATORY

# Autotuning Search Space for Loop Transformations

## 6th Workshop on the LLVM Compiler Infrastructure in HPC

Michael Kruse, Xingfu Wu, Hal Finkel

Argonne National Laboratory

2020-11-12

# Outline

1. Preface: User-Directed Loop Transformations
2. Vector-Space Autotuning
3. Tree-Space Autotuning
4. Conclusion

# Outline

1 **Preface: User-Directed Loop Transformations**

- Compiler Optimization Hints
- Extended User-Directive Loop Transformation Directives

2 **Vector-Space Autotuning**

3 **Tree-Space Autotuning**

4 **Conclusion**

# Partial Unrolling

```
#pragma unroll 4
for (int i = 0; i < n; i += 1)
  Stmt(i);
```

```
if (n > 0) {
  for (int i = 0; i+3 < n; i += 4) {
    Stmt(i);
    Stmt(i + 1);
    Stmt(i + 2);
    Stmt(i + 3);
  }
  switch (n % 4) {
  case 3:
    Stmt(n - 3);
  case 2:
    Stmt(n - 2);
  case 1:
    Stmt(n - 1);
  }
}
```

# Compiler-Supported Pragmas

### Clang

```
#pragma unroll
#pragma clang loop unroll(enable)
#pragma unroll_and_jam
#pragma clang loop distribute(enable)
#pragma clang loop vectorize(enable)
#pragma clang loop interleave(enable)
#pragma clang loop pipeline(disable)
```

### gcc

```
#pragma GCC unroll
#pragma GCC ivdep
```

### msvc

```
#pragma loop(hint_parallel(0))
#pragma loop(no_vector)
#pragma loop(ivdep)
```

### Cray

```
#pragma _CRI unroll
#pragma _CRI fusion
#pragma _CRI nofission
#pragma _CRI blockingsize
#pragma _CRI interchange
#pragma _CRI collapse
```

### OpenMP

```
#pragma omp simd
#pragma omp for
#pragma omp target
```

### PGI

```
#pragma concur
#pragma vector
#pragma ivdep
#pragma nodepchk
```

### xlc

```
#pragma unrollandfuse
#pragma stream_unroll
#pragma block_loop
#pragma loopid
```

### SGI/Open64

```
#pragma fuse
#pragma fission
#pragma blocking size
#pragma altcode
#pragma noinvarif
#pragma mem prefetch
#pragma interchange
#pragma ivdep
```

### OpenACC

```
#pragma acc kernels
```

### icc

```
#pragma parallel
#pragma offload
#pragma unroll_and_jam
#pragma nofusion
#pragma distribute_point
#pragma simd
#pragma vector
#pragma swp
#pragma ivdep
#pragma loop_count(n)
```

### Oracle Developer Studio

```
#pragma pipeloop
#pragma nomemorydepend
```

### HP

```
#pragma UNROLL_FACTOR
#pragma IF_CONVERT
#pragma IVDEP
#pragma NODEPCHK
```

# Loop Transformation Zoo

## Tiling

```
#pragma clang transform tile sizes(4,4)
for (int i = 0; i < m; i += 1)
  for (int j = 0; j < n; j += 1)
    Body(i,j);
```

⇩

```
for (int i1 = 0; i1 < m; i1 += 4)
  for (int j1 = 0; j1 < n; j1 += 4)
    for (int i2 = i1; i2 < min(i1+4,m); i2 += 1)
      for (int j2 = i1; j2 < min(j1+4,n); j2 += 1)
        Body(i2,j2);
```

## Unrolling

```
#pragma clang transform unroll partial(4)
for (int i = 0; i < n; i += 1)
  Body(i);
```

⇩

```
for (int i = 0; i < n; i1 += 4) {
  Body(i);
  Body(i+1);
  Body(i+2);
  Body(i+3);
}
```

## Fusion

```
#pragma clang transform fuse
for (int i = 0; i < n; i+=1)
  BodyA(i);
for (int i = 0; i < n; i+=1)
  BodyB(i);
```

⇩

```
for (int i = 0; i < n; i+=1) {
  BodyA(i);
  BodyB(i);
}
```

## Space-Filling Curves

```
#pragma clang transform spacefill curve(hilbert)
for (int i = 0; i < m; i += 1)
  for (int j = 0; j < n; j += 1)
    Body(i,j);
```

⇩

```
for (int idx = 0; idx < m*n; idx += 1) {
  tie(i,j) = hilbert2d_from_index(idx,m,n);
  Body(i,j);
}
```

## Interchange

```
#pragma clang transform interchange
for (int i = 0; i < m; i+=1)
  for (int j = 0; j < n; j+=1)
    Body(i,j);
```

⇩

```
for (int j = 0; j < n; j+=1)
  for (int i = 0; i < m; i+=1)
    Body(i,j);
```

## Reversal

```
#pragma clang transform reverse
for (int i = 0; i < n; i+=1)
  Body(i);
```

⇩

```
for (int i = n-1; i >= 0; i-=1)
  Body(i);
```

# Composition of Transformations

```
#pragma unroll 2
#pragma reverse
for (int i = 0; i < 128; i+=1)
  Stmt(i);
```

```
#pragma unroll 2
for (int i = 127; i >= 0; i-=1)
  Stmt(i);
```

```
for (int i = 127; i >= 0; i-=1) {
  Stmt(i);
  Stmt(i-1);
}
```

```
Stmt(127); Stmt(126); Stmt(125); Stmt(124); ...
```

```
#pragma reverse
#pragma unroll 2
for (int i = 0; i < 128; i+=1)
  Stmt(i);
```

```
#pragma reverse
for (int i = 0; i < 128; i+=2) {
  Stmt(i);
  Stmt(i+1);
}
```

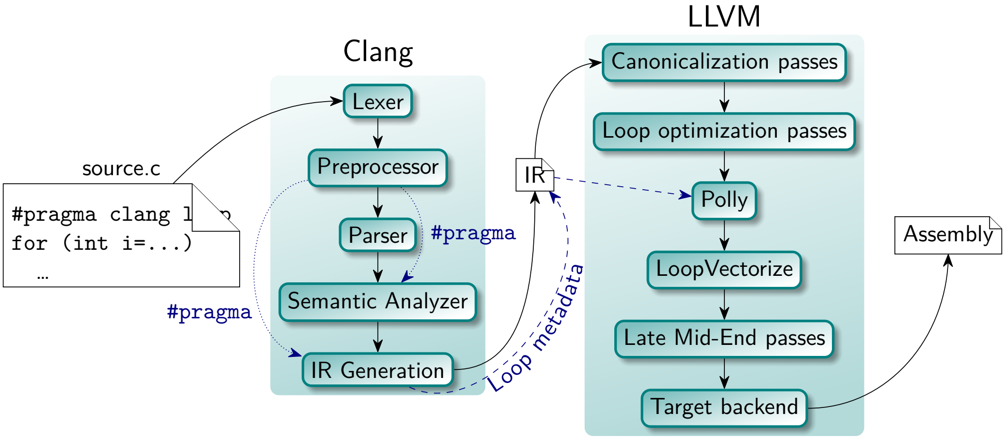
```
for (int i = 126; i >= 0; i-=2) {
  Stmt(i);
  Stmt(i+1);
}
```

```
Stmt(126); Stmt(127); Stmt(124); Stmt(125); ...
```

# Advanced Loop Directives Prototype

- *#pragma clang loop id(...)*  
  Name a loop to disambiguate transformation order
- *#pragma clang loop reverse*  
  Loop Reversal
- *#pragma clang loop tile sizes(...)*  
  Tiling
- *#pragma clang loop interchange permutation(...)*  
  Loop interchange/permutation
- *#pragma clang loop unrolling factor(...)*  
  Loop Unrolling
- *#pragma clang loop unrollingandjam factor(...)*  
  Loop Unroll-and-Jam
- *#pragma clang loop parallelize_thread*  
  Thread-Parallelization
- *#pragma clang loop pack array(....)*  
  Array Packing

# Prototype Implementation



# Implementations

- *#pragma clang loop(id) <transformation> <clauses...>*
  (Prototype implementation, LLVM-in-HPC'18)
  `https://github.com/sollve/llvm-project/tree/pragma-clang-loop`
- *#pragma clang transform <transformation> on(id) <clauses...>*
  (Composable transformations in Clang)
  `https://reviews.llvm.org/D69088`
- *#pragma clang omp <construct> <clauses...>*
  (OpenMP 5.1 implementation for Clang)
  `https://reviews.llvm.org/D76342`

# PolyBench/C dgemm

```
void matmul(int M, int N, int K, double alpha, double beta,
            double C[const restrict static M][N],
            double A[const restrict static M][K],
            double B[const restrict static K][N]) {

  for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
      C[i][j] *= beta;
      for (int k = 0; k < K; k++)
        C[i][j] += alpha * A[i][k] * B[k][j];
    }

}
```

# PolyBench/C dgemm

```
void matmul(int M, int N, int K, double alpha, double beta,
            double C[const restrict static M][N],
            double A[const restrict static M][K],
            double B[const restrict static K][N]) {

  for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++)
      C[i][j] *= beta;

  for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++)
      for (int k = 0; k < K; k++)
        C[i][j] += alpha * A[i][k] * B[k][j];

}
```

# PolyBench/C dgemm

## Using Inner Loop Vectorization

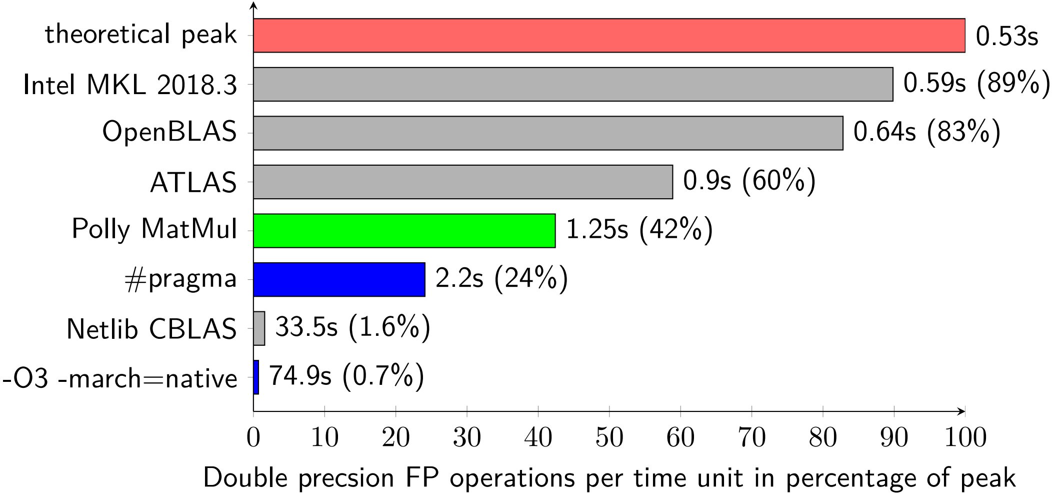
```
#pragma clang loop(j2) pack array(A)
#pragma clang loop(i1) pack array(B)
#pragma clang loop(i1,j1,k1,i2,j2) interchange permutation(j1,k1,i1,j2,i2)
#pragma clang loop(i,j,k) tile sizes(96,2048,256) \
                          floor_ids(i1,j1,k1) tile_ids(i2,j2,k2)

#pragma clang loop id(i)
for (int i = 0; i < M; i += 1)
  #pragma clang loop id(j)
  for (int j = 0; j < N; j += 1)
    #pragma clang loop id(k)
    for (int k = 0; k < K; k += 1)
      C[i][j] += alpha * A[i][k] * B[k][j];
```

# Polybench dgemm

## After Transformation (Sanitized)

```
double Packed_B[256][2048];
double Packed_A[96][256];
if (runtime check) {
  if (M >= 1)
    for (int c0 = 0; c0 <= floord(N - 1, 2048); c0 += 1)    // Loop j1
      for (int c1 = 0; c1 <= floord(K - 1, 256); c1 += 1) { // Loop k1

        // Copy-in: B -> Packed_B
        for (int c4 = 0; c4 <= min(2047, N - 2048 * c0 - 1); c4 += 1)
          for (int c5 = 0; c5 <= min(255, K - 256 * c1 - 1); c5 += 1)
            Packed_B[c4][c5] = B[256 * c1 + c5][2048 * c0 + c4];

        for (int c2 = 0; c2 <= floord(M - 1, 96); c2 += 1) { // Loop i1

          // Copy-in: A -> Packed_A
          for (int c6 = 0; c6 <= min(95, M - 96 * c2 - 1); c6 += 1)
            for (int c7 = 0; c7 <= min(255, K - 256 * c1 - 1); c7 += 1)
              Packed_A[c6][c7] = A[96 * c2 + c6][256 * c1 + c7];

          for (int c3 = 0; c3 <= min(2047, N - 2048 * c0 - 1); c3 += 1)   // Loop j2
            for (int c4 = 0; c4 <= min(95, M - 96 * c2 - 1); c4 += 1)     // Loop i2
              for (int c5 = 0; c5 <= min(255, K - 256 * c1 - 1); c5 += 1) // Loop k2
                C[96 * c2 + c4][2048 * c0 + c3] += Packed_A[c4][c5] * Packed_B[c3][c5];
        }
     }
} else {
  /* original code */
}
```

# Polybench dgemm



# Motivation

- User-directives requires the user to know what to do …
- … why not find the transformations automatically?

# Outline

1 Preface: User-Directed Loop Transformations

2 Vector-Space Autotuning

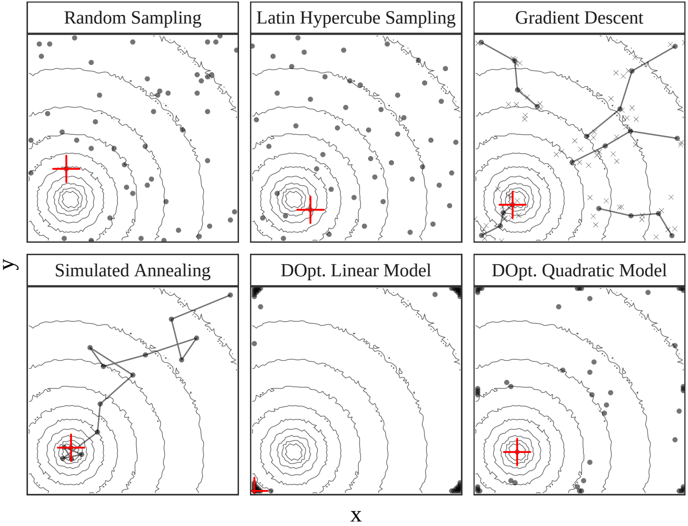
- Loop Autotuning
- Conclusion

3 Tree-Space Autotuning

4 Conclusion

# Vector Space Combinatorial Optimization

## Predictive Modeling / Empirical



P. Bruel et. al.
*Autotuning under Tight Budget Constraints:*
*A Transparent Design of Experiments Approach*
(CCGRID'19)

# Components

- **ytopt**: Autotuning framework, supports multiple machine learning strategies
  `https://github.com/ytopt-team/ytopt`
- **Plopper**: Source rewrite engine
  `https://github.com/ytopt-team/autotune`
- **pragma-clang-loop**: Loop transformation implementation
  `https://github.com/sollve/llvm-project/tree/pragma-clang-loop`

# Plopper

## syr2k

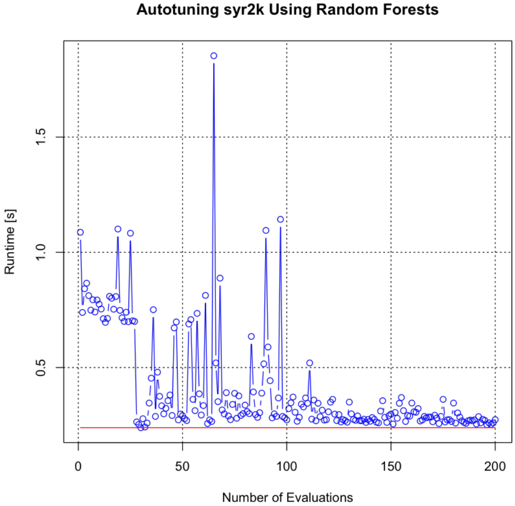
```
#P0
#P1
#P2
#pragma clang loop(i,j,k) tile sizes(#P3,#P4,#P5) \
        floor_ids(i1,j1,k1) tile_ids(i2,j2,k2)

#pragma clang loop id(i)
for (i = 0; i < _PB_N; i++)
    #pragma clang loop id(j)
    for (j = 0; j < _PB_M; j++)
     #pragma clang loop id(k)
        for (k = 0; k <= i; k++)
          C[i][k] += A[k][j]*alpha*B[i][j] + B[k][j]*alpha*A[i][j];
```

# ytopt Search Space Definition

```
P0=CSH.CategoricalHyperparameter(name='P0', default_value=' ',
    choices=["#pragma clang loop(j2) pack array(A) allocate(malloc)", " "])
P1=CSH.CategoricalHyperparameter(name='P1', default_value=' ',
    choices=["#pragma clang loop(i1) pack array(B) allocate(malloc)", " "])
P2=CSH.CategoricalHyperparameter(name='P2', default_value=' ',
    choices=["#pragma clang loop(i1,j1,k1,i2,j2) interchange permutation(j1,k1,i1,j2,i2)", " "])
P3=CSH.OrdinalHyperparameter(name='P3', default_value='96',
    sequence=['4','8','16','20','32','50','64','80','96','100','128'])
P4=CSH.OrdinalHyperparameter(name='P4', default_value='2048',
     sequence=['4','8','16','20','32','50','64','80','100','128','2048'])
P5=CSH.OrdinalHyperparameter(name='P5',  default_value='256',
    sequence=['4','8','16','20','32','50','64','80','100','128','256'])
cs.add_hyperparameters([P0, P1, P2, p3, P4, P5])
cond1 = CS.InCondition(P1, P0, ['#pragma clang loop(j2) pack array(A) allocate(malloc)'])
```

# Syr2k Tuning

## Random Forests



# Syr2k Tuning Result

```
#pragma clang loop(j2) pack array(A) allocate(malloc)
#pragma clang loop(i1) pack array(B) allocate(malloc)
#pragma clang loop(i1,j1,k1,i2,j2) interchange permutation(j1,k1,i1,j2,i2)
#pragma clang loop tile sizes(128,128,100) \
        floor_ids(i1,j1,k1) tile_ids(i2,j2,k2)
for (int i = 0; i < _PB_N; i++)
    for (int j = 0; j < _PB_M; j++)
        for (int k = 0; k <= i; k++)
            C[i][k] += A[k][j]*alpha*B[i][j] + B[k][j]*alpha*A[i][j];
```

# Vector-Space Autotuning

## Strengths

- Machine learning over tuples of numbers well established
- Exploit contiguous relationships

## Weaknesses

- User has to define loop transformations
- Search space is necessarily incomplete
  (Number of transformations is potentially infinite)
- Depending on previous choices, some parameters are unused
- Some parameters cause abrupt changes
  (e.g. no predictable relationship between loop permutations)

# Related Work

## CHiLL (TVM, ...)

- Transformations in hand-written *recipes*
- Tune placeholders in recipe (e.g. tile sizes)

## LIFT (SPIRAL,...)

- Based on rewriting engine
- Using ATF (Auto-Tuning Framework)
- Parameter constraints to prune search space

## LeTSeE

- Encode schedule function coefficients
- Correctness constrains encoded as polytope boundaries

## PATUS (CLTuner,...)

- Specific to categories of kernels (stencils)
- Assume trivially parallel kernels
- Tune fixed set of parameters (tile size, workgroup size, ...)

## HalideTuner

- Using OpenTuner
- Encode transformation sequence in constant-size vector
- Verifier prunes configuration that do no encode a valid transformation sequence

# Outline

1 Preface: User-Directed Loop Transformations

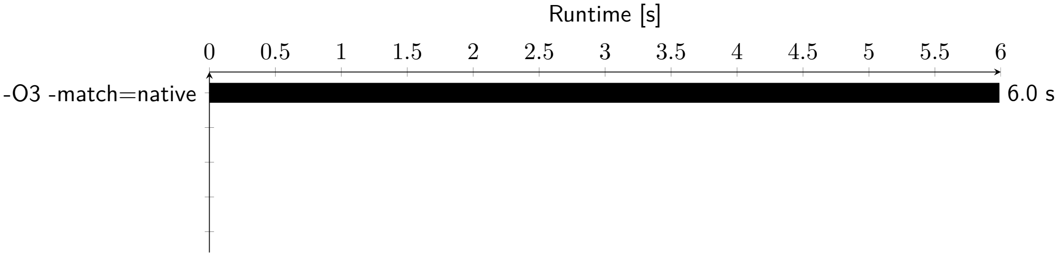
2 Vector-Space Autotuning

3 Tree-Space Autotuning
- Motivation
- Implementation
- Evaluation
- Related Work

4 Conclusion

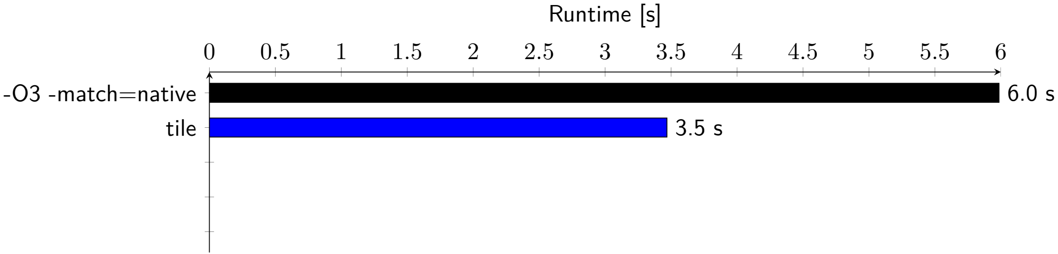
# Successive Optimization

```
for (int i = 0; i < M; i += 1)
  for (int j = 0; j < N; j += 1)
    for (int k = 0; k < K; k += 1)
      C[i][j] += A[i][k] * B[k][j];
```



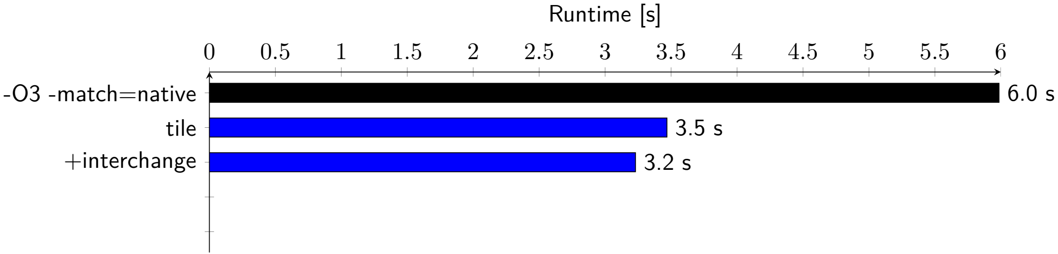
# Successive Optimization

```
#pragma clang loop tile sizes(96,2048,256) floor_ids(i1,j1,k1) tile_ids(i2,j2,k2)
for (int i = 0; i < M; i += 1)
  for (int j = 0; j < N; j += 1)
    for (int k = 0; k < K; k += 1)
      C[i][j] += A[i][k] * B[k][j];
```



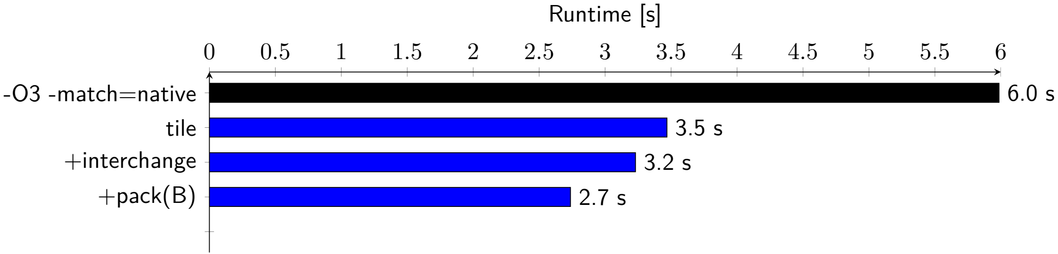
# Successive Optimization

```
#pragma clang loop(i1,j1,k1,i2,j2) interchange permutation(j1,k1,i1,j2,i2)
#pragma clang loop tile sizes(96,2048,256) floor_ids(i1,j1,k1) tile_ids(i2,j2,k2)
for (int i = 0; i < M; i += 1)
  for (int j = 0; j < N; j += 1)
    for (int k = 0; k < K; k += 1)
      C[i][j] += A[i][k] * B[k][j];
```



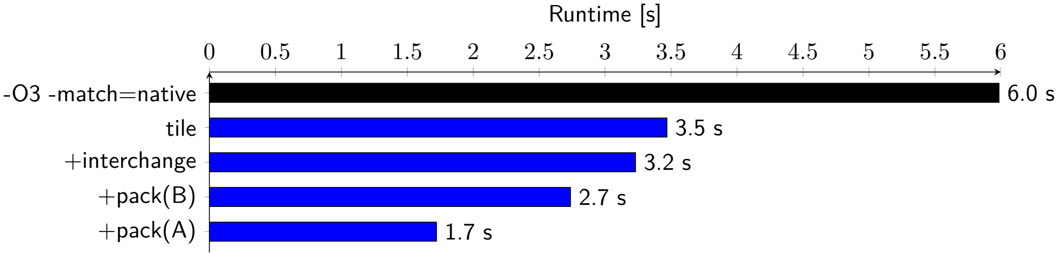
# Successive Optimization

```
#pragma clang loop(i1) pack array(B)
#pragma clang loop(i1,j1,k1,i2,j2) interchange permutation(j1,k1,i1,j2,i2)
#pragma clang loop tile sizes(96,2048,256) floor_ids(i1,j1,k1) tile_ids(i2,j2,k2)
for (int i = 0; i < M; i += 1)
  for (int j = 0; j < N; j += 1)
    for (int k = 0; k < K; k += 1)
      C[i][j] += A[i][k] * B[k][j];
```

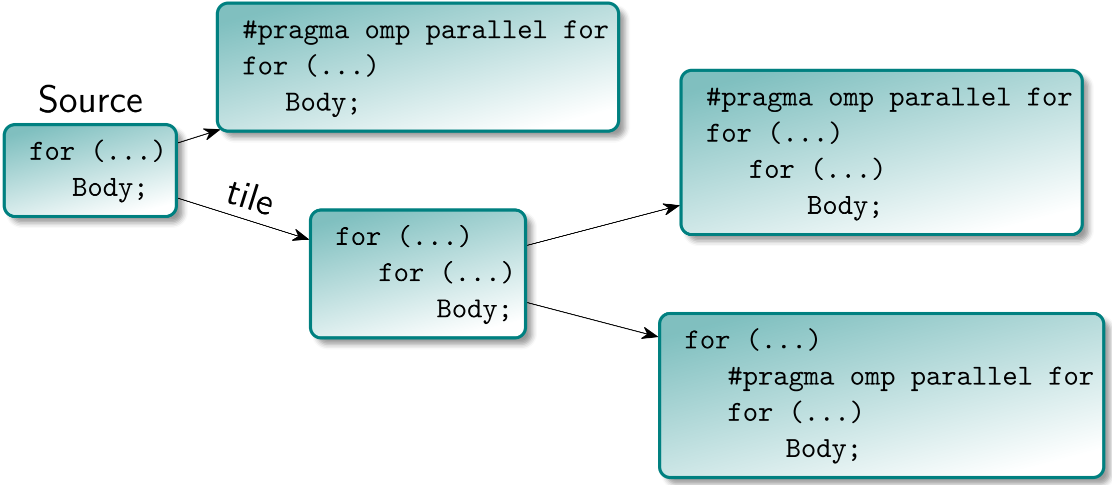


# Successive Optimization

```
#pragma clang loop(j2) pack array(A)
#pragma clang loop(i1) pack array(B)
#pragma clang loop(i1,j1,k1,i2,j2) interchange permutation(j1,k1,i1,j2,i2)
#pragma clang loop tile sizes(96,2048,256) floor_ids(i1,j1,k1) tile_ids(i2,j2,k2)
for (int i = 0; i < M; i += 1)
  for (int j = 0; j < N; j += 1)
    for (int k = 0; k < K; k += 1)
      C[i][j] += A[i][k] * B[k][j];
```



# Transformation Composition Possibilities



Source

```
for (...)
  Body;
```

tile

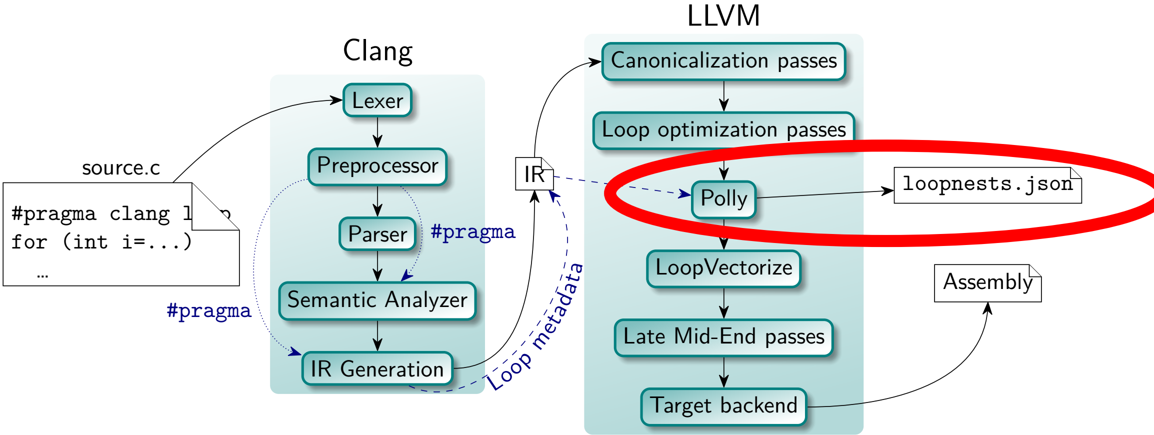
```
#pragma omp parallel for
for (...)
  Body;
```

```
for (...)
  for (...)
    Body;
```

```
#pragma omp parallel for
for (...)
  for (...)
    Body;
```

```
for (...)
  #pragma omp parallel for
  for (...)
    Body;
```

# Prototype Implementation

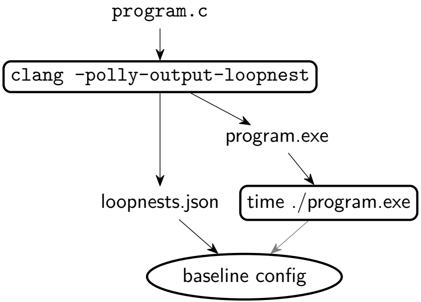


# Implementation: Start

1. Compile
   - Output: Loop nest structure in `loopnests.json`
2. Run
   - Measure baseline wall time
3. Generate search tree ("*mctree*")

## Proof-of-Concept Transformations

- Tiling
  - Tile sizes: 4, 16, 64, 256, 1024
- Interchange
- Thread parallelization (`schedule(static)`)

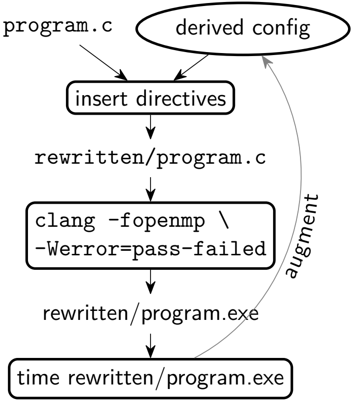


# Implementation: Iterate

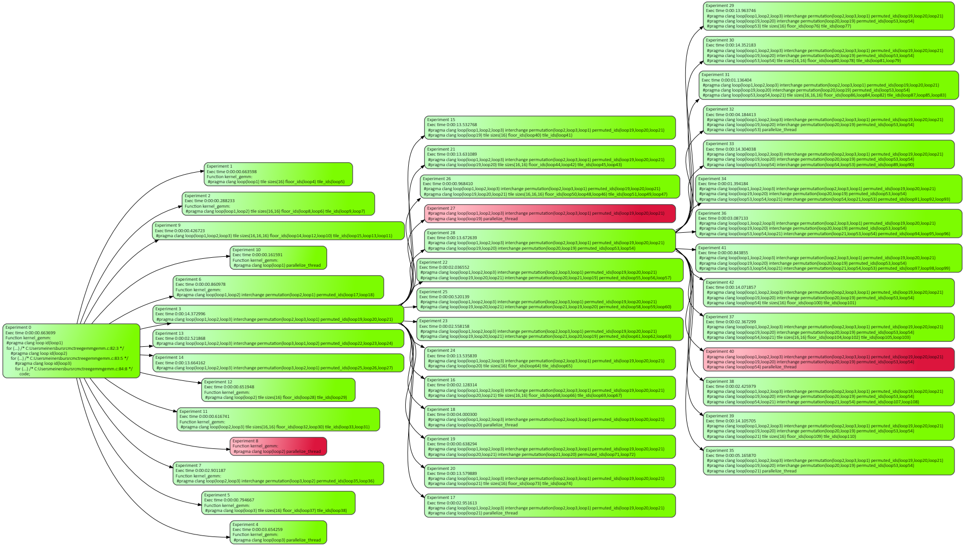
1. Select configuration to run
   - According search strategy
2. Insert transformation directives
3. Compile
4. Run
   - Measure baseline wall time
5. Rinse, Repeat

## Proof-of-Concept Search Strategy

- Select child of fastest-yet



# Tree Search Space Autotuning



```
Experiment 0
Exec time 0:00:00.663699
Function kernel_gemm:
 #pragma clang loop id(loop1)
 for (...) /* C:Usersmeinersbursrcmctreegemmgemm.c:82:3 */
   #pragma clang loop id(loop2)
   for (...) /* C:Usersmeinersbursrcmctreegemmgemm.c:83:5 */
     #pragma clang loop id(loop3)
     for (...) /* C:Usersmeinersbursrcmctreegemmgemm.c:84:8 */
       code;
```

```
Experiment 1
Exec time 0:00:00.663598
Function kernel_gemm:
 #pragma clang loop(loop1) tile sizes(16) floor_ids(loop4) tile_ids(loop5)

Experiment 2
Exec time 0:00:00.288233
Function kernel_gemm:
 #pragma clang loop(loop1,loop2) tile sizes(16,16) floor_ids(loop8,loop6) tile_ids(loop9,loop7)

Experiment 9
Exec time 0:00:00.426723
 #pragma clang loop(loop1,loop2,loop3) tile sizes(16,16,16) floor_ids(loop14,loop12,loop10) tile_ids(loop15,loop13,loop11)

Experiment 10
Exec time 0:00:00.161591
Function kernel_gemm:
 #pragma clang loop(loop1) parallelize_thread

Experiment 6
Exec time 0:00:00.860978
Function kernel_gemm:
 #pragma clang loop(loop1,loop2) interchange permutation(loop2,loop1) permuted_ids(loop17,loop18)

Experiment 3
Exec time 0:00:14.372996
 #pragma clang loop(loop1,loop2,loop3) interchange permutation(loop2,loop3,loop1) permuted_ids(loop19,loop20,loop21)

Experiment 13
Exec time 0:00:02.521868
 #pragma clang loop(loop1,loop2,loop3) interchange permutation(loop3,loop1,loop2) permuted_ids(loop22,loop23,loop24)

Experiment 14
Exec time 0:00:13.664162
 #pragma clang loop(loop1,loop2,loop3) interchange permutation(loop3,loop2,loop1) permuted_ids(loop25,loop26,loop27)

Experiment 12
Exec time 0:00:00.651948
Function kernel_gemm:
 #pragma clang loop(loop2) tile sizes(16) floor_ids(loop28) tile_ids(loop29)

Experiment 11
Exec time 0:00:00.616741
Function kernel_gemm:
 #pragma clang loop(loop2,loop3) tile sizes(16,16) floor_ids(loop32,loop30) tile_ids(loop33,loop31)

Experiment 8
Function kernel_gemm:
 #pragma clang loop(loop2) parallelize_thread

Experiment 7
Exec time 0:00:02.901187
Function kernel_gemm:
 #pragma clang loop(loop2,loop3) interchange permutation(loop3,loop2) permuted_ids(loop35,loop36)

Experiment 5
Exec time 0:00:00.794667
Function kernel_gemm:
 #pragma clang loop(loop3) tile sizes(16) floor_ids(loop37) tile_ids(loop38)

Experiment 4
Exec time 0:00:03.654259
Function kernel_gemm:
 #pragma clang loop(loop3) parallelize_thread
```

```
Experiment 15
Exec time 0:00:13.532768
 #pragma clang loop(loop1,loop2,loop3) interchange permutation(loop2,loop3,loop1) permuted_ids(loop19,loop20,loop21)
 #pragma clang loop(loop19) tile sizes(16) floor_ids(loop40) tile_ids(loop41)

Experiment 21
Exec time 0:00:13.631089
 #pragma clang loop(loop1,loop2,loop3) interchange permutation(loop2,loop3,loop1) permuted_ids(loop19,loop20,loop21)
 #pragma clang loop(loop19,loop20) tile sizes(16,16) floor_ids(loop44,loop42) tile_ids(loop45,loop43)

Experiment 26
Exec time 0:00:00.968410
 #pragma clang loop(loop1,loop2,loop3) interchange permutation(loop2,loop3,loop1) permuted_ids(loop19,loop20,loop21)
 #pragma clang loop(loop19,loop20,loop21) tile sizes(16,16,16) floor_ids(loop50,loop48,loop46) tile_ids(loop51,loop49,loop47)

Experiment 27
 #pragma clang loop(loop1,loop2,loop3) interchange permutation(loop2,loop3,loop1) permuted_ids(loop19,loop20,loop21)
 #pragma clang loop(loop19) parallelize_thread

Experiment 28
Exec time 0:00:13.672639
 #pragma clang loop(loop1,loop2,loop3) interchange permutation(loop2,loop3,loop1) permuted_ids(loop19,loop20,loop21)
 #pragma clang loop(loop19,loop20) interchange permutation(loop20,loop19) permuted_ids(loop53,loop54)

Experiment 22
Exec time 0:00:02.036552
 #pragma clang loop(loop1,loop2,loop3) interchange permutation(loop2,loop3,loop1) permuted_ids(loop19,loop20,loop21)
 #pragma clang loop(loop19,loop20,loop21) interchange permutation(loop20,loop21,loop19) permuted_ids(loop55,loop56,loop57)

Experiment 25
Exec time 0:00:00.520139
 #pragma clang loop(loop1,loop2,loop3) interchange permutation(loop2,loop3,loop1) permuted_ids(loop19,loop20,loop21)
 #pragma clang loop(loop19,loop20,loop21) interchange permutation(loop21,loop19,loop20) permuted_ids(loop58,loop59,loop60)

Experiment 23
Exec time 0:00:02.558158
 #pragma clang loop(loop1,loop2,loop3) interchange permutation(loop2,loop3,loop1) permuted_ids(loop19,loop20,loop21)
 #pragma clang loop(loop19,loop20,loop21) interchange permutation(loop21,loop20,loop19) permuted_ids(loop61,loop62,loop63)

Experiment 24
Exec time 0:00:13.535839
 #pragma clang loop(loop1,loop2,loop3) interchange permutation(loop2,loop3,loop1) permuted_ids(loop19,loop20,loop21)
 #pragma clang loop(loop20) tile sizes(16) floor_ids(loop64) tile_ids(loop65)

Experiment 16
Exec time 0:00:02.128314
 #pragma clang loop(loop1,loop2,loop3) interchange permutation(loop2,loop3,loop1) permuted_ids(loop19,loop20,loop21)
 #pragma clang loop(loop20,loop21) tile sizes(16,16) floor_ids(loop68,loop66) tile_ids(loop69,loop67)

Experiment 18
Exec time 0:00:04.000300
 #pragma clang loop(loop1,loop2,loop3) interchange permutation(loop2,loop3,loop1) permuted_ids(loop19,loop20,loop21)
 #pragma clang loop(loop20) parallelize_thread

Experiment 19
Exec time 0:00:00.638294
 #pragma clang loop(loop1,loop2,loop3) interchange permutation(loop2,loop3,loop1) permuted_ids(loop19,loop20,loop21)
 #pragma clang loop(loop20,loop21) interchange permutation(loop21,loop20) permuted_ids(loop71,loop72)

Experiment 20
Exec time 0:00:13.579889
 #pragma clang loop(loop1,loop2,loop3) interchange permutation(loop2,loop3,loop1) permuted_ids(loop19,loop20,loop21)
 #pragma clang loop(loop21) tile sizes(16) floor_ids(loop73) tile_ids(loop74)

Experiment 17
Exec time 0:00:02.951613
 #pragma clang loop(loop1,loop2,loop3) interchange permutation(loop2,loop3,loop1) permuted_ids(loop19,loop20,loop21)
 #pragma clang loop(loop21) parallelize_thread
```

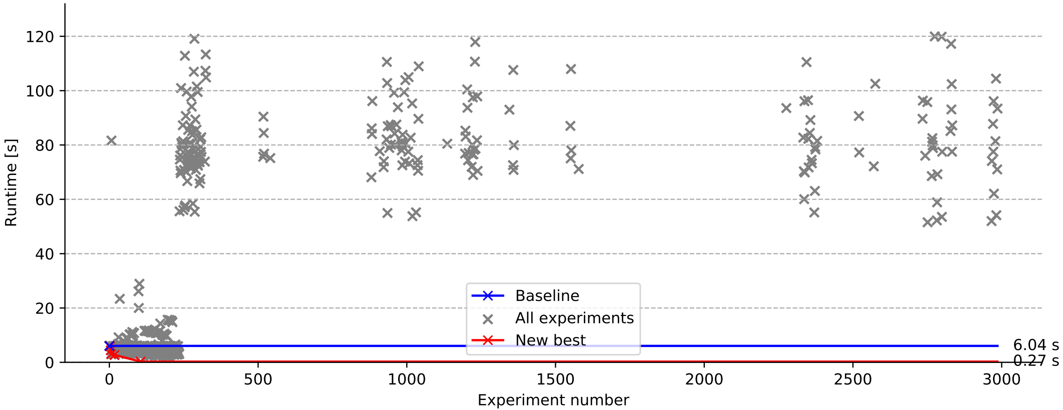
```
Experiment 29
Exec time 0:00:13.963746
 #pragma clang loop(loop1,loop2,loop3) interchange permutation(loop2,loop3,loop1) permuted_ids(loop19,loop20,loop21)
 #pragma clang loop(loop19,loop20) interchange permutation(loop20,loop19) permuted_ids(loop53,loop54)
 #pragma clang loop(loop53) tile sizes(16) floor_ids(loop76) tile_ids(loop77)

Experiment 30
Exec time 0:00:14.352183
 #pragma clang loop(loop1,loop2,loop3) interchange permutation(loop2,loop3,loop1) permuted_ids(loop19,loop20,loop21)
 #pragma clang loop(loop19,loop20) interchange permutation(loop20,loop19) permuted_ids(loop53,loop54)
 #pragma clang loop(loop53,loop54) tile sizes(16,16) floor_ids(loop80,loop78) tile_ids(loop81,loop79)

Experiment 31
Exec time 0:00:01.136404
 #pragma clang loop(loop1,loop2,loop3) interchange permutation(loop2,loop3,loop1) permuted_ids(loop19,loop20,loop21)
 #pragma clang loop(loop19,loop20) interchange permutation(loop20,loop19) permuted_ids(loop53,loop54)
 #pragma clang loop(loop53,loop54,loop21) tile sizes(16,16,16) floor_ids(loop86,loop84,loop82) tile_ids(loop87,loop85,loop83)

Experiment 32
Exec time 0:00:04.184413
 #pragma clang loop(loop1,loop2,loop3) interchange permutation(loop2,loop3,loop1) permuted_ids(loop19,loop20,loop21)
 #pragma clang loop(loop19,loop20) interchange permutation(loop20,loop19) permuted_ids(loop53,loop54)
 #pragma clang loop(loop53) parallelize_thread

Experiment 33
Exec time 0:00:14.304038
 #pragma clang loop(loop1,loop2,loop3) interchange permutation(loop2,loop3,loop1) permuted_ids(loop19,loop20,loop21)
 #pragma clang loop(loop19,loop20) interchange permutation(loop20,loop19) permuted_ids(loop53,loop54)
 #pragma clang loop(loop53,loop54) interchange permutation(loop54,loop53) permuted_ids(loop89,loop90)

Experiment 34
Exec time 0:00:01.394184
 #pragma clang loop(loop1,loop2,loop3) interchange permutation(loop2,loop3,loop1) permuted_ids(loop19,loop20,loop21)
 #pragma clang loop(loop19,loop20) interchange permutation(loop20,loop19) permuted_ids(loop53,loop54)
 #pragma clang loop(loop53,loop54,loop21) interchange permutation(loop54,loop21,loop53) permuted_ids(loop91,loop92,loop93)

Experiment 36
Exec time 0:00:03.087133
 #pragma clang loop(loop1,loop2,loop3) interchange permutation(loop2,loop3,loop1) permuted_ids(loop19,loop20,loop21)
 #pragma clang loop(loop19,loop20) interchange permutation(loop20,loop19) permuted_ids(loop53,loop54)
 #pragma clang loop(loop53,loop54,loop21) interchange permutation(loop21,loop53,loop54) permuted_ids(loop94,loop95,loop96)

Experiment 41
Exec time 0:00:00.843855
 #pragma clang loop(loop1,loop2,loop3) interchange permutation(loop2,loop3,loop1) permuted_ids(loop19,loop20,loop21)
 #pragma clang loop(loop19,loop20) interchange permutation(loop20,loop19) permuted_ids(loop53,loop54)
 #pragma clang loop(loop53,loop54,loop21) interchange permutation(loop21,loop54,loop53) permuted_ids(loop97,loop98,loop99)

Experiment 42
Exec time 0:00:14.071857
 #pragma clang loop(loop1,loop2,loop3) interchange permutation(loop2,loop3,loop1) permuted_ids(loop19,loop20,loop21)
 #pragma clang loop(loop19,loop20) interchange permutation(loop20,loop19) permuted_ids(loop53,loop54)
 #pragma clang loop(loop54) tile sizes(16) floor_ids(loop100) tile_ids(loop101)

Experiment 37
Exec time 0:00:02.367299
 #pragma clang loop(loop1,loop2,loop3) interchange permutation(loop2,loop3,loop1) permuted_ids(loop19,loop20,loop21)
 #pragma clang loop(loop19,loop20) interchange permutation(loop20,loop19) permuted_ids(loop53,loop54)
 #pragma clang loop(loop54,loop21) tile sizes(16,16) floor_ids(loop104,loop102) tile_ids(loop105,loop103)

Experiment 40
 #pragma clang loop(loop1,loop2,loop3) interchange permutation(loop2,loop3,loop1) permuted_ids(loop19,loop20,loop21)
 #pragma clang loop(loop19,loop20) interchange permutation(loop20,loop19) permuted_ids(loop53,loop54)
 #pragma clang loop(loop54) parallelize_thread

Experiment 38
Exec time 0:00:02.425979
 #pragma clang loop(loop1,loop2,loop3) interchange permutation(loop2,loop3,loop1) permuted_ids(loop19,loop20,loop21)
 #pragma clang loop(loop19,loop20) interchange permutation(loop20,loop19) permuted_ids(loop53,loop54)
 #pragma clang loop(loop54,loop21) interchange permutation(loop21,loop54) permuted_ids(loop107,loop108)

Experiment 39
Exec time 0:00:14.105705
 #pragma clang loop(loop1,loop2,loop3) interchange permutation(loop2,loop3,loop1) permuted_ids(loop19,loop20,loop21)
 #pragma clang loop(loop19,loop20) interchange permutation(loop20,loop19) permuted_ids(loop53,loop54)
 #pragma clang loop(loop21) tile sizes(16) floor_ids(loop109) tile_ids(loop110)

Experiment 35
Exec time 0:00:05.165870
 #pragma clang loop(loop1,loop2,loop3) interchange permutation(loop2,loop3,loop1) permuted_ids(loop19,loop20,loop21)
 #pragma clang loop(loop19,loop20) interchange permutation(loop20,loop19) permuted_ids(loop53,loop54)
 #pragma clang loop(loop21) parallelize_thread
```

# Tree Search Space Autotuning

- PolyBench/C 4.2 kernels: dgemm, syr2k, covariance
  - `EXTRALARGE_DATASET` (largest predefined problem size)
  - Manually distributed
  - syr2k and covariance are non-rectangular
- 2x Intel Xeon Platinum 8180M
  - 28 cores each, 112 logical threads
  - 376 GiB RAM, 38.5 MiB L3, 1 MiB L2, 32 KiB L1d
- 6h tuning time
  - Program execution time limit: 120 secs.

# dgemm Tuning

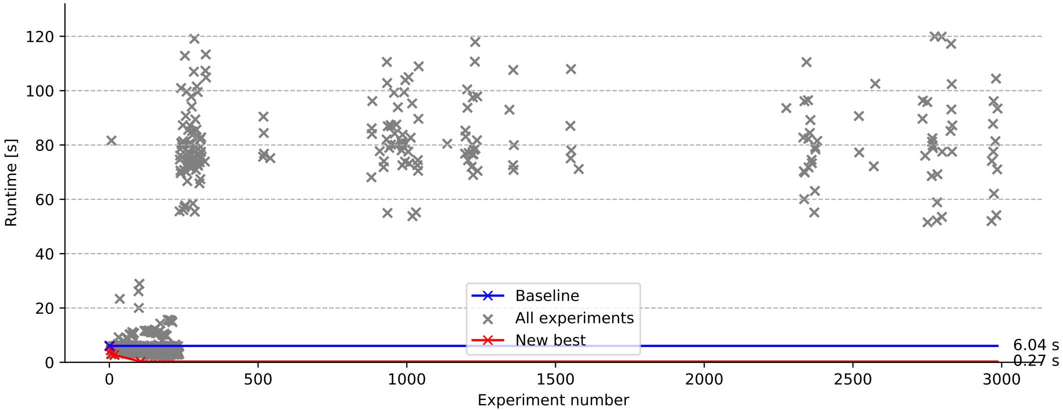
## Interchange/Tile/Parallel



Best configuration:

# dgemm Tuning

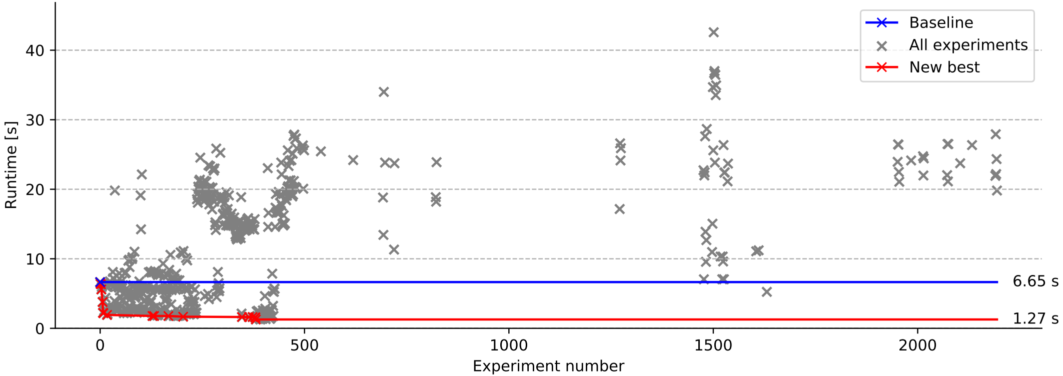
## Interchange/Tile/Parallel



Best configuration: `#pragma clang loop(i) parallelize_thread`

# dgemm Tuning

## Interchange/Tile

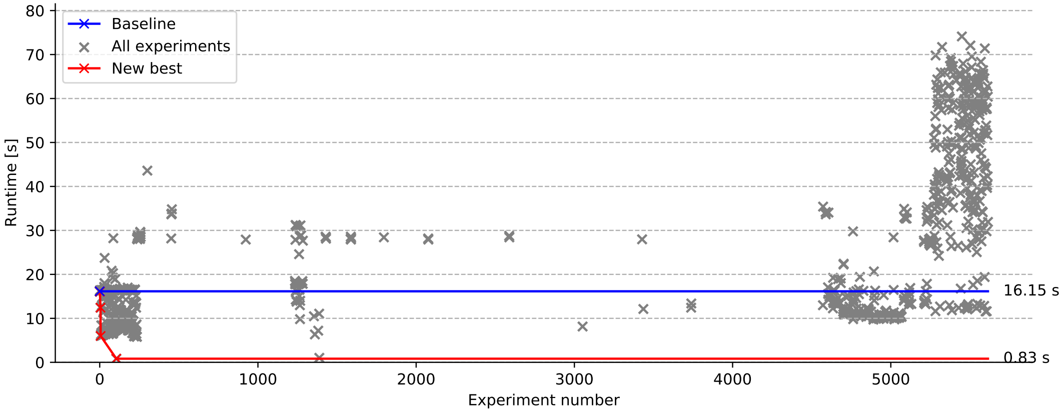


Best configuration:

```
#pragma clang loop(j,k,i) tile sizes(1024,64,16)
#pragma clang loop(i,j,k) interchange permutation(j,k,i)
```

# syr2k Tuning

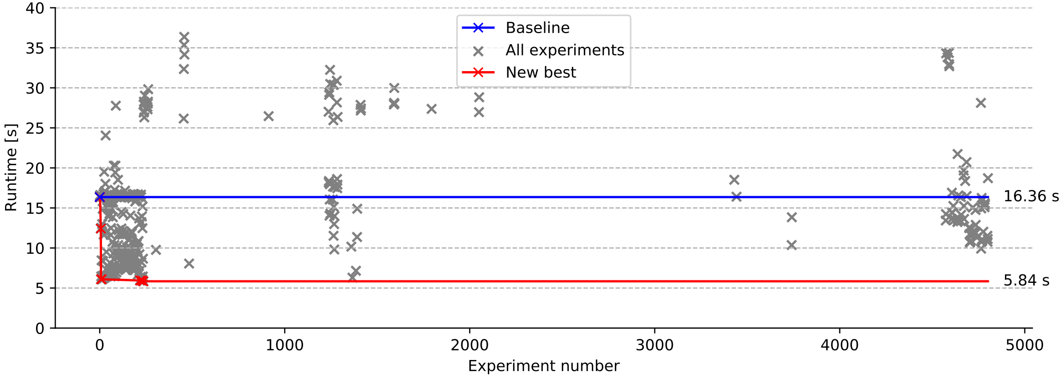
## Interchange/Tile/Parallel



Best configuration: `#pragma clang loop(i) parallelize_thread`

# syr2k Tuning

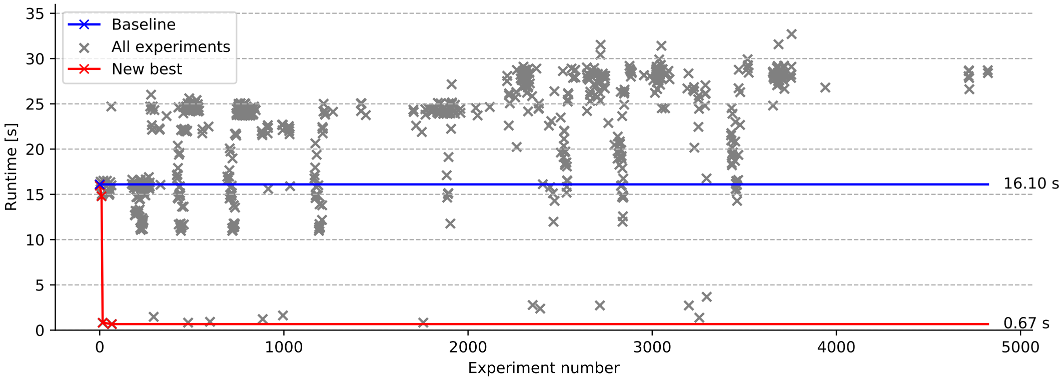
## Interchange/Tile



Best configuration: *#pragma clang loop(i,j,k) tile sizes(64,256,4)*

# covariance Tuning Result

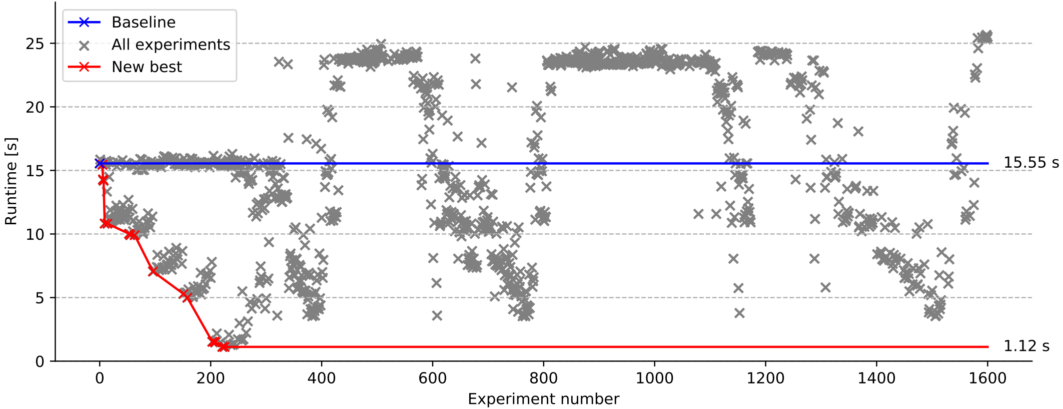
## Interchange/Tile/Parallel



Best configuration: *#pragma clang loop(i) parallelize_thread*

# covariance Tuning Result

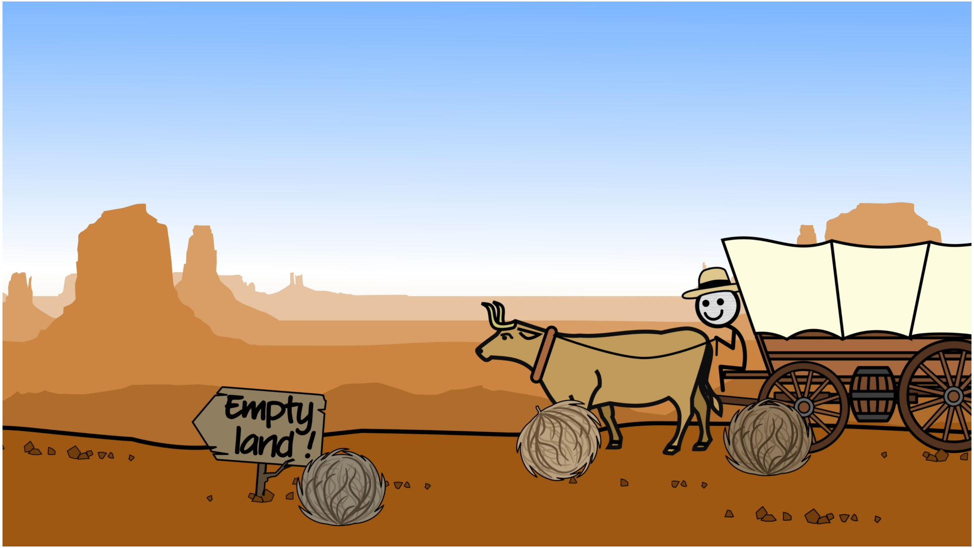
## Interchange/Tile



Best configuration:

```
#pragma clang loop(j,k,i) tile sizes(1024,64,16)
#pragma clang loop(i,j,k) interchange permutation(j,k,i)
```

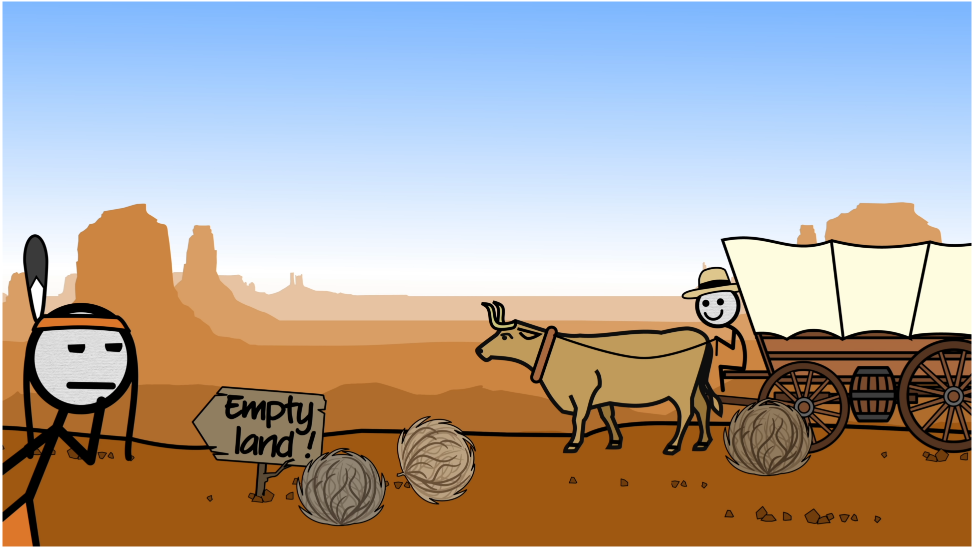
# Similar Work



CGP Grey: The Trouble With Tumbleweed

youtu.be/hsWr_JWTZss

# Similar Work



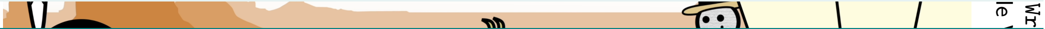
CGP Grey: The Trouble With Tumbleweed
youtu.be/hsWr_JWTZss

# Similar Work

## Halide (ACM TOG, July 2019)

`https://dl.acm.org/doi/abs/10.1145/3306346.3322967`

- Use transformation tree space (organized in stages)
- Beam Search
- Magnitudes faster than HalideTune
- Twice as fast as Halide's latest more restricted default autotuner.



## ProTuner (arXiv, May 2020)

`https://arxiv.org/abs/2005.13685`

- Use same transformation tree space
- Monte-Carlo tree search
- Up to 3.6 times faster executable that with Beam Search

# Outline

1 Preface: User-Directed Loop Transformations

2 Vector-Space Autotuning

3 Tree-Space Autotuning

4 Conclusion
- Future Work
- Summary

# Future Work / Summary

## Summary

- Two search space shapes
  - Vector space
  - Tree space
- Implement loop transformation config space generator (*mctree*)
- Implement proof-of-concept autotuner
  - Greedy search space exploitation
- Apply to PolyBench kernels
  - gemm, syr2k, covariance

## Future Work

- Implement Monte-Carlo tree search
  - Do not get stuck in local optima
- Prune search space
  - Interchange twice is redundant
  - Multiple paths to same configuration (⇒ DAG)
- Combine approaches
  - MCTS for transformation composition
  - ytopt for transformation parameters
- Measure/tune loop nests individually
  - Avoid combinatorial explosion

## Acknowledgments

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration, in particular its subproject PROTEAS-TUNE.

This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

This material was based in part upon funding from the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357.

That's all Folks!