Locus: A System and a Language for Program Optimization

Thiago Teixeira[*], Corinne Ancourt[+], David Padua[*], William Gropp[*]

[*]Department of Computer Science, University of Illinois at Urbana-Champaign, USA
[+]MINES ParisTech, PSL University, France

CGO - Washington, DC - Feb 2019

Introduction

Introduction

- Very complex machines
- Gap between performance of hand-tuned and compiler-generated code has grown substantially

Introduction

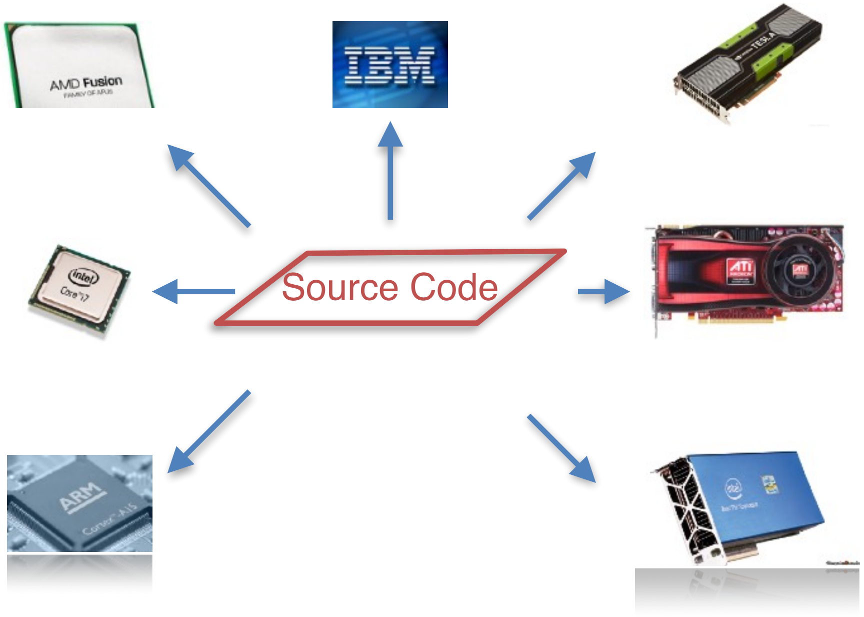
- Very complex machines
- Gap between performance of hand-tuned and compiler-generated code has grown substantially
- Platform-specific optimizations are required

Introduction

- Very complex machines
- Gap between performance of hand-tuned and compiler-generated code has grown substantially
- Platform-specific optimizations are required
- Platforms change, and new ones are introduced

Introduction

- Very complex machines
- Gap between performance of hand-tuned and compiler-generated code has grown substantially
- Platform-specific optimizations are required
- Platforms change, and new ones are introduced



Introduction

- Very complex machines
- Gap between performance of hand-tuned and compiler-generated code has grown substantially
- Platform-specific optimizations are required
- Platforms change, and new ones are introduced
- As you add them the code becomes less and less maintainable and understandable

Goal

- Improve performance automatically
- Target multiple platforms
- Keep the code maintainable in the long term

Goal

- Improve performance automatically
- Target multiple platforms
- Keep the code maintainable in the long term
- How?

Goal

- Improve performance automatically
- Target multiple platforms
- Keep the code maintainable in the long term
- How?

We use empirical search

Goal

- Improve performance automatically
- Target multiple platforms
- Keep the code maintainable in the long term
- How?

We use empirical search

Automatically generate and evaluate a collection of optimized variants by executing them

Challenges

Challenges

1. How to describe a collection of optimized variants (opt space) concisely?
    - modify and extend the use of optimizations

Challenges

1. How to describe a collection of optimized variants (opt space) concisely?
    - modify and extend the use of optimizations
2. Generate the variants automatically:
    - often needs multiple techniques
    - a lot tools out there
    - tools are not prepared to work with each other
    - compose a diverse set of transformations into a final code is not trivial

Challenges

1. How to describe a collection of optimized variants (opt space) concisely?
   - modify and extend the use of optimizations
2. Generate the variants automatically:
   - often needs multiple techniques
   - a lot tools out there
   - tools are not prepared to work with each other
   - compose a diverse set of transformations into a final code is not trivial
3. Select relevant variants
   - optimization space too large to be fully evaluated

Challenges

1. How to describe a collection of optimized variants (opt space) concisely?
   - modify and extend the use of optimizations
2. Generate the variants automatically:
   - often needs multiple techniques
   - a lot tools out there
   - tools are not prepared to work with each other
   - compose a diverse set of transformations into a final code is not trivial
3. Select relevant variants
   - optimization space too large to be fully evaluated
4. Manage platform-specific recipes of transformations
   - how and where to store
   - make it available to non-experts

Optimization Space

- triple nested loop

```
for i
  for j
    for k
```

Optimization Space

- triple nested loop

6 variants

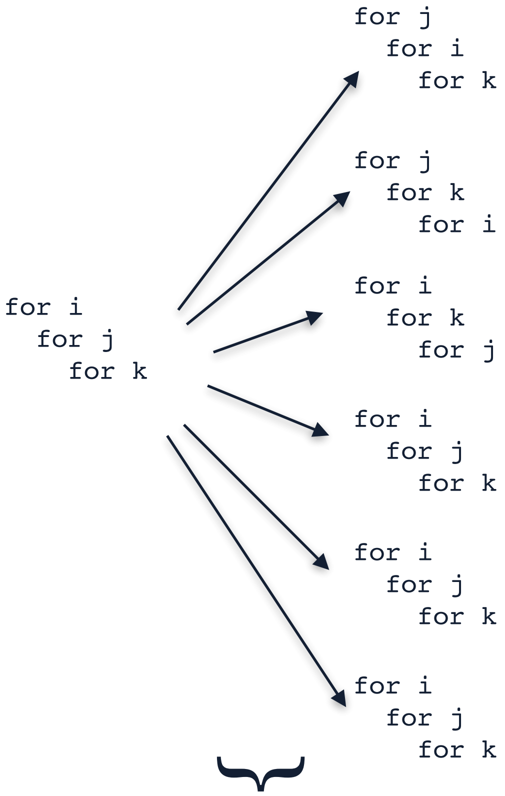
```
for i
  for j
    for k
```

Interchange
all permutations

Optimization Space

- triple nested loop

6 variants



```
for i
  for j
    for k
```

```
for j
  for i
    for k
```

```
for j
  for k
    for i
```

```
for i
  for k
    for j
```

```
for i
  for j
    for k
```

```
for i
  for j
    for k
```

```
for i
  for j
    for k
```

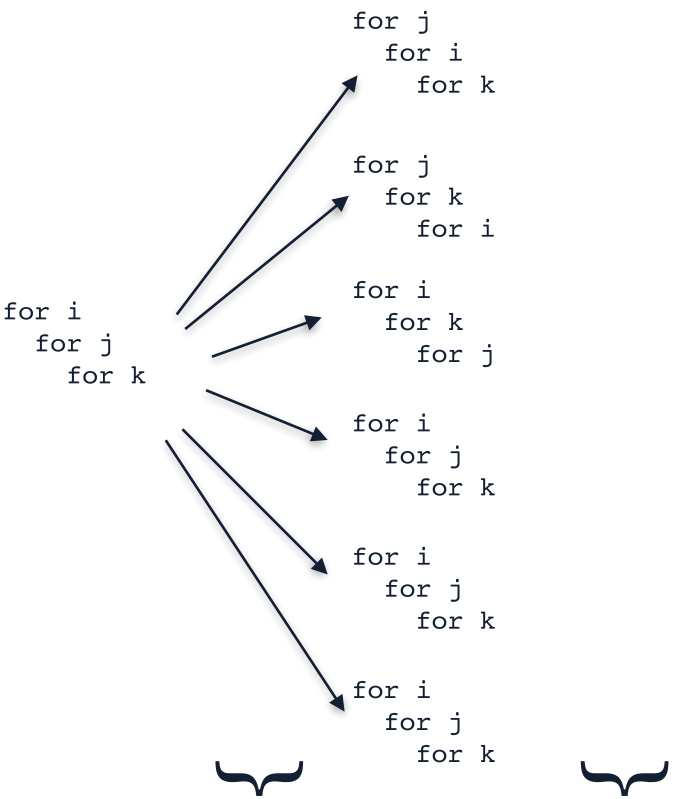
Interchange

all permutations

Optimization Space

- triple nested loop

6 variants 18 variants



```
for i
  for j
    for k
```

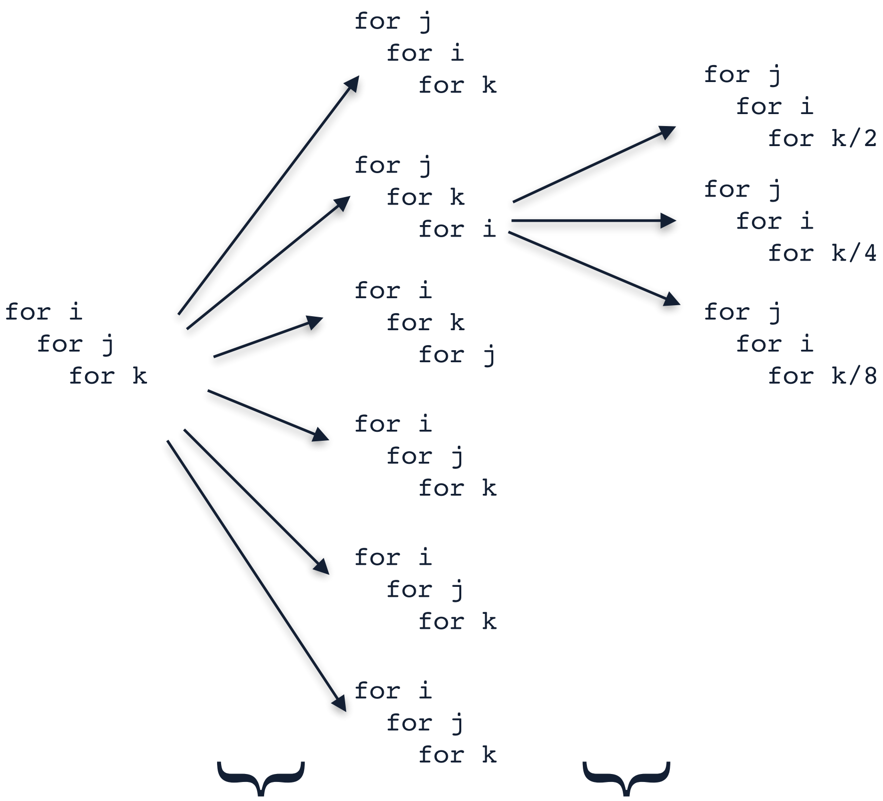
```
for j
  for i
    for k

for j
  for k
    for i

for i
  for k
    for j

for i
  for j
    for k

for i
  for j
    for k

for i
  for j
    for k
```

Interchange
all permutations

Unroll
(2, 4, 8)

Optimization Space

- triple nested loop



6 variants

18 variants

```
for i
  for j
    for k
```

```
for j
  for i
    for k
```

```
for j
  for k
    for i
```

```
for i
  for k
    for j
```

```
for i
  for j
    for k
```

```
for i
  for j
    for k
```

```
for i
  for j
    for k
```

```
for j
  for i
    for k/2
```

```
for j
  for i
    for k/4
```

```
for j
  for i
    for k/8
```

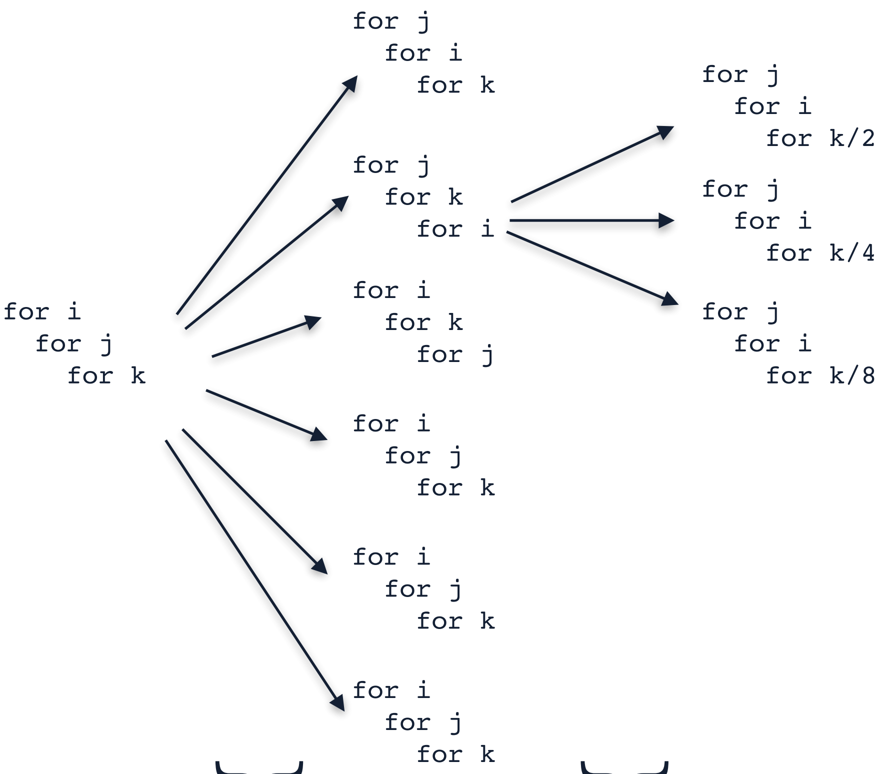
Interchange
all permutations

Unroll
(2, 4, 8)

Optimization Space

- triple nested loop

6 variants 18 variants 126 variants



```
for i
  for j
    for k
```

```
for j
  for i
    for k
```

```
for j
  for k
    for i
```

```
for i
  for k
    for j
```

```
for i
  for j
    for k
```

```
for i
  for j
    for k
```

```
for i
  for j
    for k
```

```
for j
  for i
    for k/2
```

```
for j
  for i
    for k/4
```

```
for j
  for i
    for k/8
```

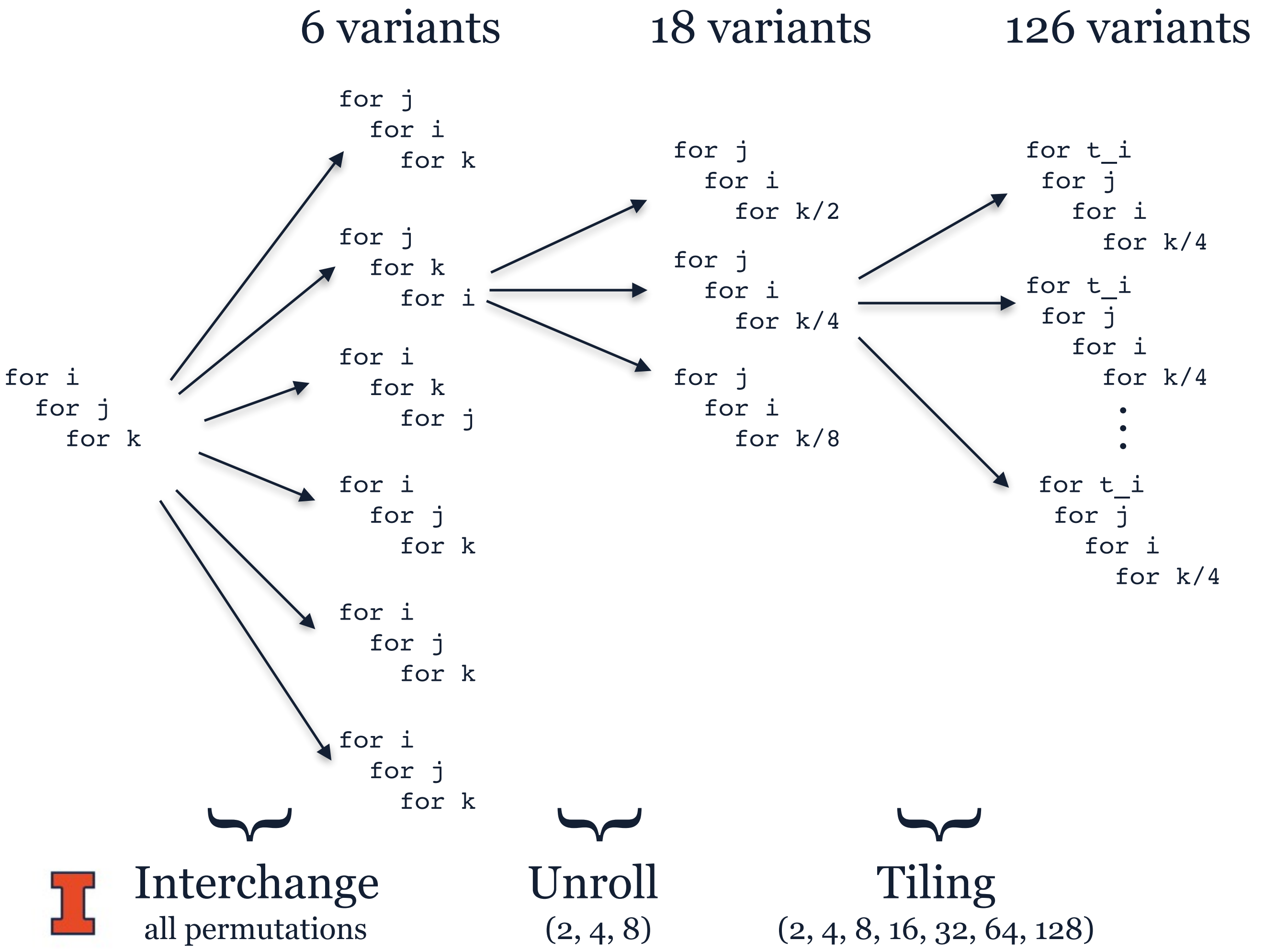
Interchange
all permutations

Unroll
(2, 4, 8)

Tiling
(2, 4, 8, 16, 32, 64, 128)

Optimization Space

- triple nested loop



6 variants

18 variants

126 variants

```
for i
  for j
    for k
```

```
for j
  for i
    for k
```

```
for j
  for k
    for i
```

```
for i
  for k
    for j
```

```
for i
  for j
    for k
```

```
for i
  for j
    for k
```

```
for i
  for j
    for k
```

```
for j
  for i
    for k/2
```

```
for j
  for i
    for k/4
```

```
for j
  for i
    for k/8
```

```
for t_i
 for j
   for i
     for k/4
```

```
for t_i
 for j
   for i
     for k/4
```

⋮

```
for t_i
 for j
   for i
     for k/4
```

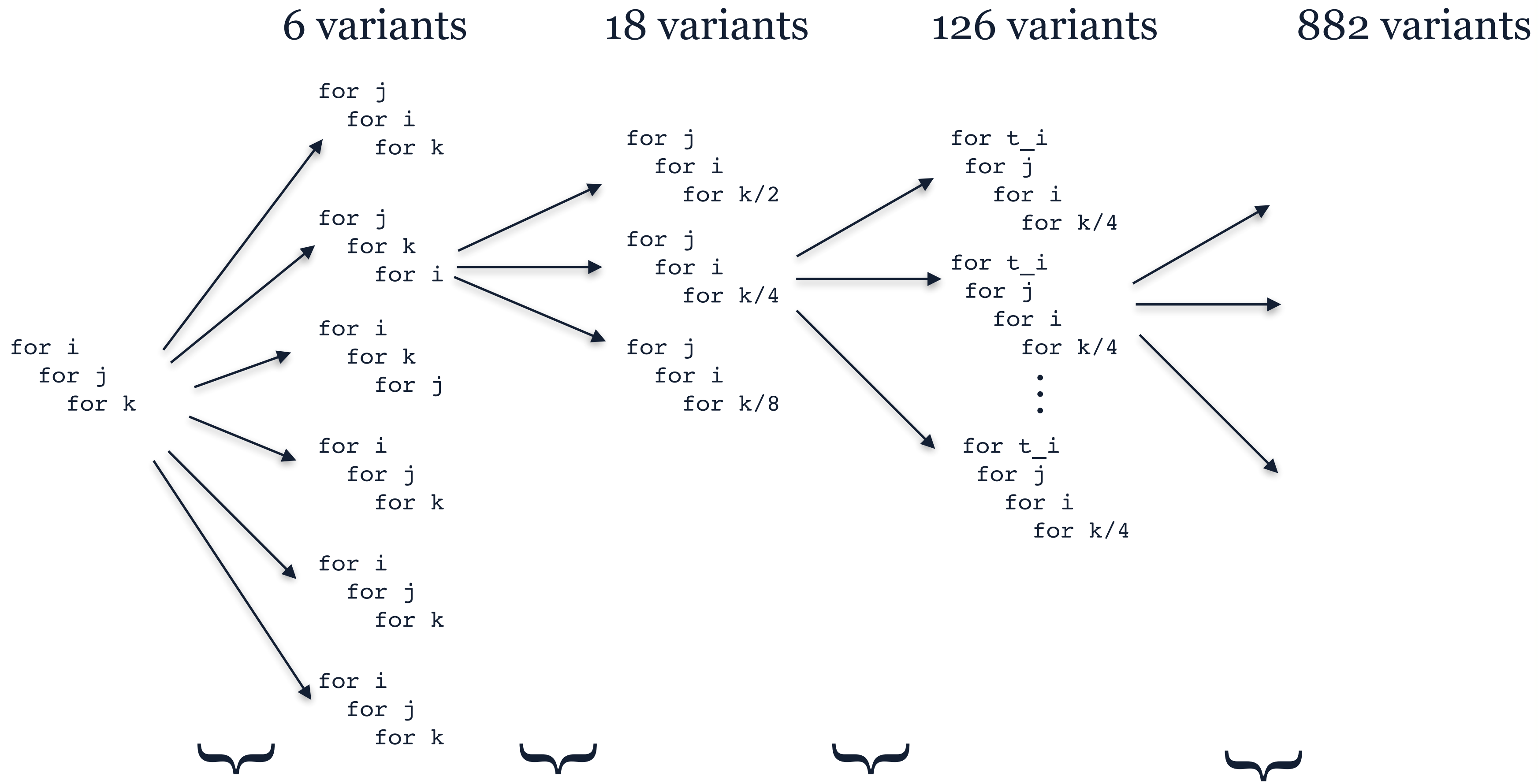
Interchange
all permutations

Unroll
(2, 4, 8)

Tiling
(2, 4, 8, 16, 32, 64, 128)

Optimization Space

- triple nested loop



6 variants 18 variants 126 variants 882 variants

```
for i
  for j
    for k
```

```
for j
  for i
    for k

for j
  for k
    for i

for i
  for k
    for j

for i
  for j
    for k

for i
  for j
    for k

for i
  for j
    for k
```

```
for j
  for i
    for k/2

for j
  for i
    for k/4

for j
  for i
    for k/8
```

```
for t_i
 for j
   for i
     for k/4

for t_i
 for j
   for i
     for k/4
       ⋮
 for t_i
  for j
    for i
      for k/4
```

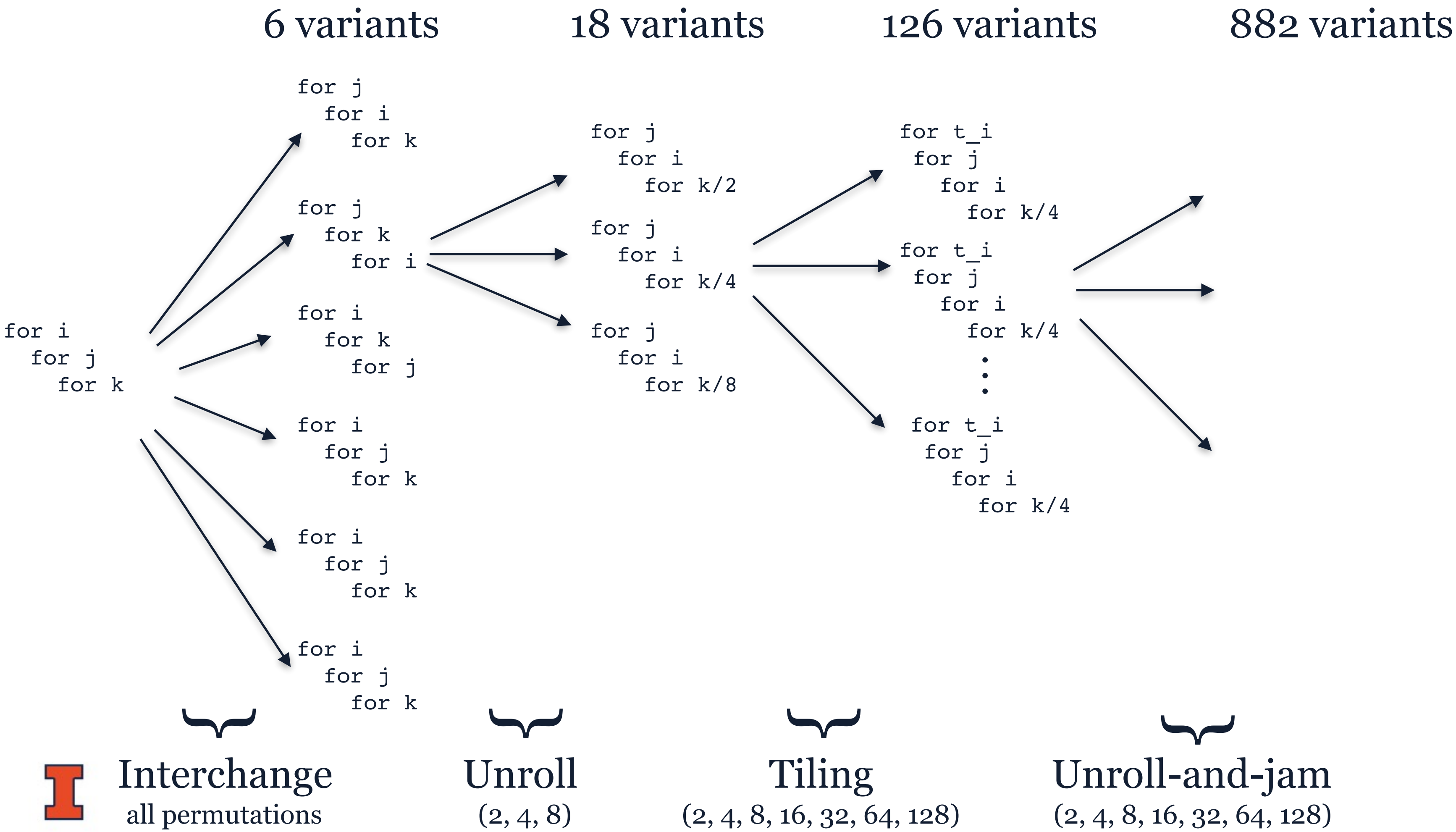
Interchange
all permutations

Unroll
(2, 4, 8)

Tiling
(2, 4, 8, 16, 32, 64, 128)

Unroll-and-jam
(2, 4, 8, 16, 32, 64, 128)

Locus



```
for i
  for j
    for k
```

6 variants

```
for j
  for i
    for k

for j
  for k
    for i

for i
  for k
    for j

for i
  for j
    for k

for i
  for j
    for k

for i
  for j
    for k
```

18 variants

```
for j
  for i
    for k/2

for j
  for i
    for k/4

for j
  for i
    for k/8
```

126 variants

```
for t_i
 for j
   for i
     for k/4

for t_i
 for j
   for i
     for k/4
       ⋮
for t_i
 for j
   for i
     for k/4
```

882 variants

Interchange
all permutations

Unroll
(2, 4, 8)

Tiling
(2, 4, 8, 16, 32, 64, 128)

Unroll-and-jam
(2, 4, 8, 16, 32, 64, 128)

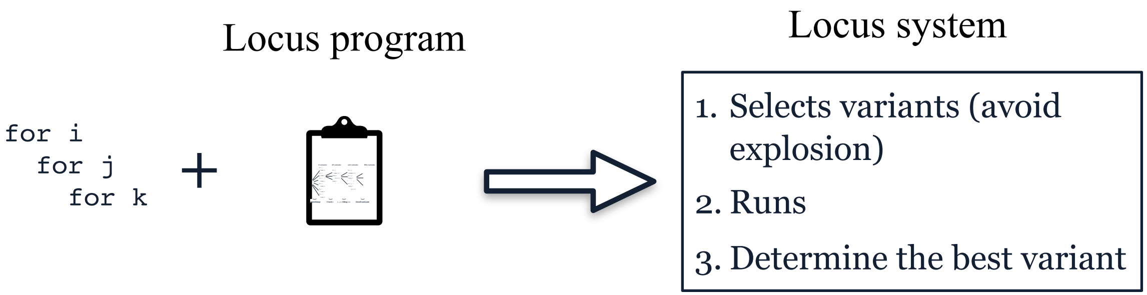
Locus

Locus program

```
for i
  for j
    for k
```

+

Locus



Locus program

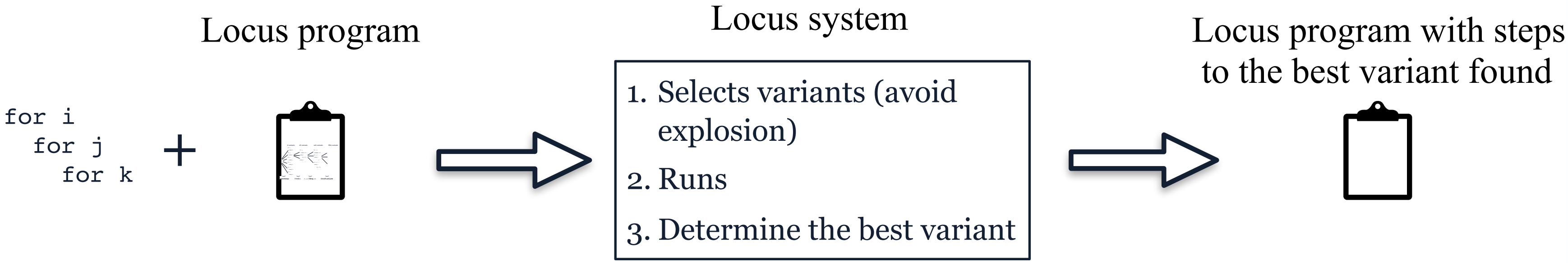
```
for i
  for j
    for k
```

+

Locus system

1. Selects variants (avoid explosion)
2. Runs
3. Determine the best variant

Locus



```
for i
  for j
    for k
```

+

Locus program

Locus system

1. Selects variants (avoid explosion)
2. Runs
3. Determine the best variant

Locus program with steps to the best variant found

Locus

- Semi-automatic approach to assist performance experts and code developers in the performance optimization of programs in C, C++, and Fortran

- Orchestrates the application of transformations to a baseline version of the code

- Specially for optimizing complex, long-lived applications running on different environments

Contributions

Contributions

- Defined Locus language:
  - describe *concisely* complex space of optimizations
  - *agnostic* of any specific traversal method
  - *decouple* performance expert role from application expert role

Contributions

- Defined Locus language:
  - describe *concisely* complex space of optimizations
  - *agnostic* of any specific traversal method
  - *decouple* performance expert role from application expert role

- Implemented a system with flexible API for plugging in:
  - *different* variant selection techniques (optimization space traversal)
  - *collection* of transformations developed internally and externally

Contributions

- Defined Locus language:
  - describe *concisely* complex space of optimizations
  - *agnostic* of any specific traversal method
  - *decouple* performance expert role from application expert role

- Implemented a system with flexible API for plugging in:
  - *different* variant selection techniques (optimization space traversal)
  - *collection* of transformations developed internally and externally

- Optimizer and interpreter for the Locus programs:
  - *prune* the space automatically
  - speeds-up the empirical search

Locus Approach

- Baseline code: defined by the developer, no platform- or compiler-specific optimizations
- Annotated regions of interest (i.e., code regions)
- Program the application of the optimizations for each code region

Locus System

Annotated Source Code

```
#pragma @Locus loop = matmul
  for (i=0; i<M; i++)
    for (j=0; j<N; j++)
      for (k=0; k<K; k++)
      C[i][j] = beta*C[i][j]
                + alpha*A[i][k]*B[k][j];
```

Locus System

Annotated Source Code

```
#pragma @Locus loop = matmul
 for (i=0; i<M; i++)
  for (j=0; j<N; j++)
   for (k=0; k<K; k++)
   C[i][j] = beta*C[i][j]
        + alpha*A[i][k]*B[k][j];
```

Locus System

Annotated Source Code

```
#pragma @Locus loop = matmul
  for (i=0; i<M; i++)
    for (j=0; j<N; j++)
      for (k=0; k<K; k++)
      C[i][j] = beta*C[i][j]
                + alpha*A[i][k]*B[k][j];
```

Locus Program

```
CodeReg matmul {
  tiledim = 4;
  tiletype = Tiling2D() OR Tiling3D();
  printstatus(tiletype);
  if (tiletype == "2D") {
    RoseLocus.Unroll(loop=innermost, factor=tiledim);
  }
}
```

Locus System

Annotated Source Code

```
#pragma @Locus loop = matmul
  for (i=0; i<M; i++)
    for (j=0; j<N; j++)
      for (k=0; k<K; k++)
      C[i][j] = beta*C[i][j]
                + alpha*A[i][k]*B[k][j];
```

Locus Program

```
CodeReg matmul {
  tiledim = 4;
  tiletype = Tiling2D() OR Tiling3D();
  printstatus(tiletype);
  if (tiletype == "2D") {
    RoseLocus.Unroll(loop=innermost, factor=tiledim);
  }
}
```

Locus System

Annotated Source Code

```
#pragma @Locus loop = matmul
  for (i=0; i<M; i++)
    for (j=0; j<N; j++)
      for (k=0; k<K; k++)
      C[i][j] = beta*C[i][j]
                + alpha*A[i][k]*B[k][j];
```

Locus Program

```
CodeReg matmul {
  tiledim = 4;
  tiletype = Tiling2D() OR Tiling3D();
  printstatus(tiletype);
  if (tiletype == "2D") {
    RoseLocus.Unroll(loop=innermost, factor=tiledim);
  }
}
```

- Optimizations are target-specific and region-specific
- Separated from the application’s code

Locus Optimization Language

Locus Optimization Language

- Optimization recipes for each code region (CodeReg, OptSeq)

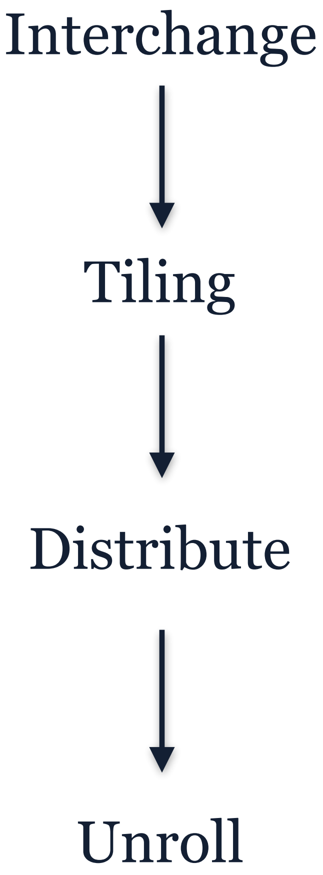
Locus Optimization Language

- Optimization recipes for each code region (CodeReg, OptSeq)
- Loops, If-then-else

Locus Optimization Language

- Optimization recipes for each code region (CodeReg, OptSeq)
- Loops, If-then-else
- Special Search Constructs:
  - OR blocks and statements;
  - Optional statements;
  - *enum*, *integer*, *permutation*, *poweroftwo*…

Locus Optimization Language



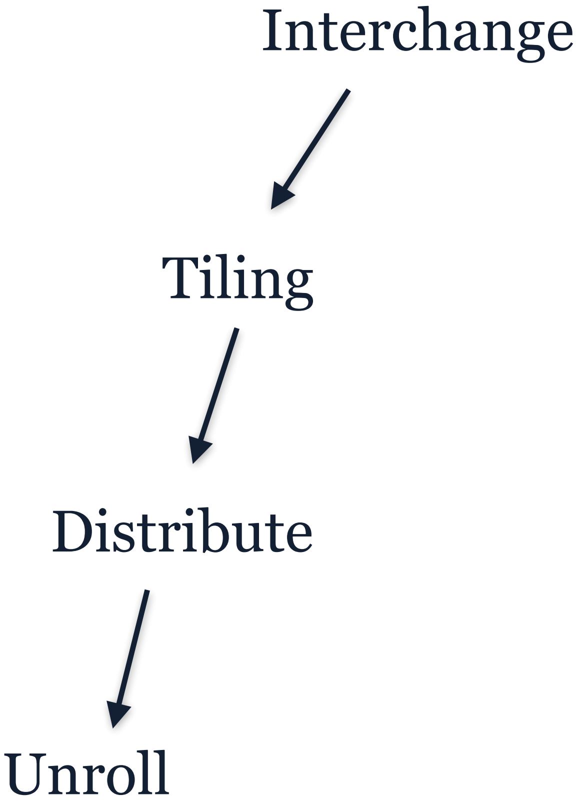
Interchange

Tiling

Distribute

Unroll

Locus Optimization Language



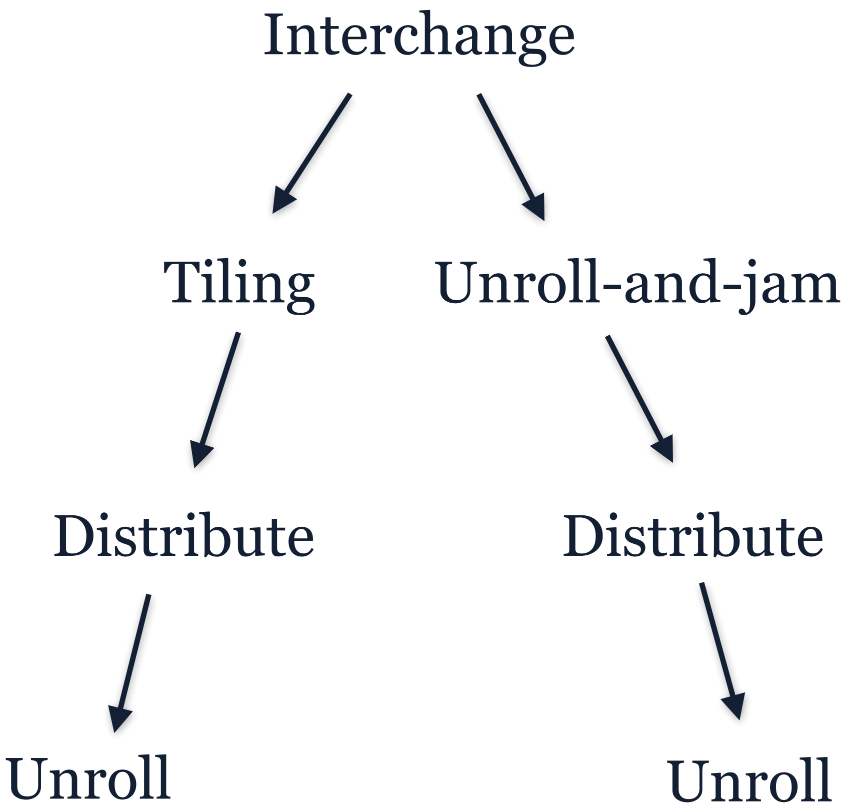
Interchange

Tiling

Distribute

Unroll

Locus Optimization Language



Interchange

Tiling

Unroll-and-jam

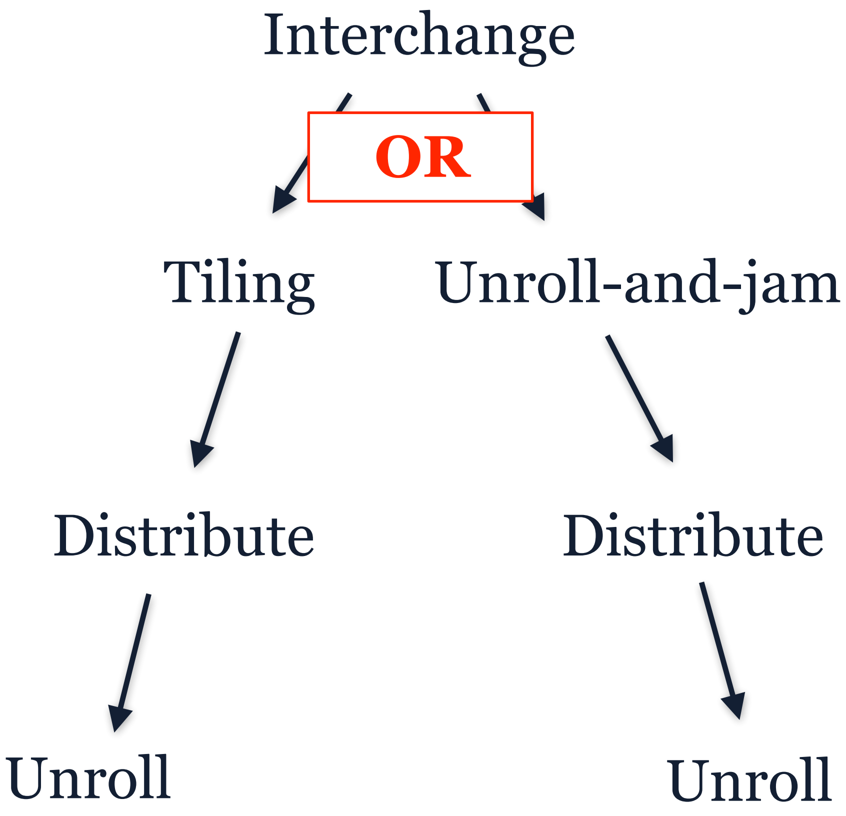
Distribute

Distribute

Unroll

Unroll

Locus Optimization Language



Interchange

OR

Tiling

Unroll-and-jam

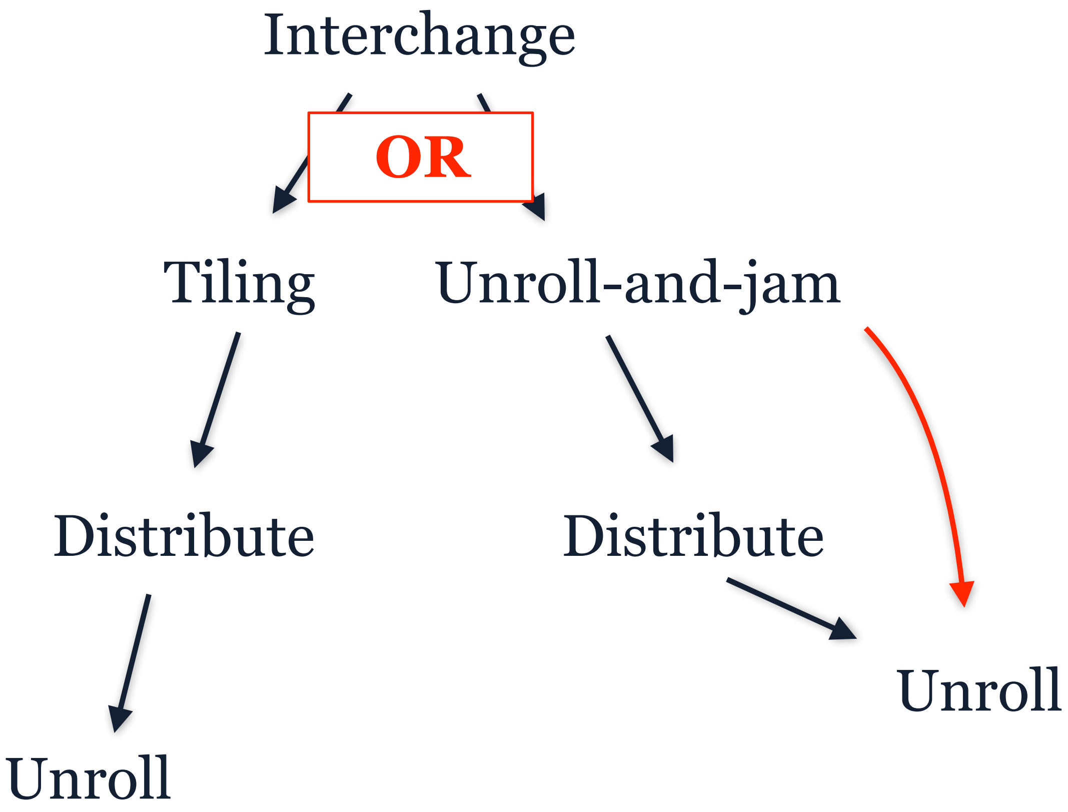
Distribute

Distribute

Unroll

Unroll

Locus Optimization Language



Interchange

OR

Tiling

Unroll-and-jam

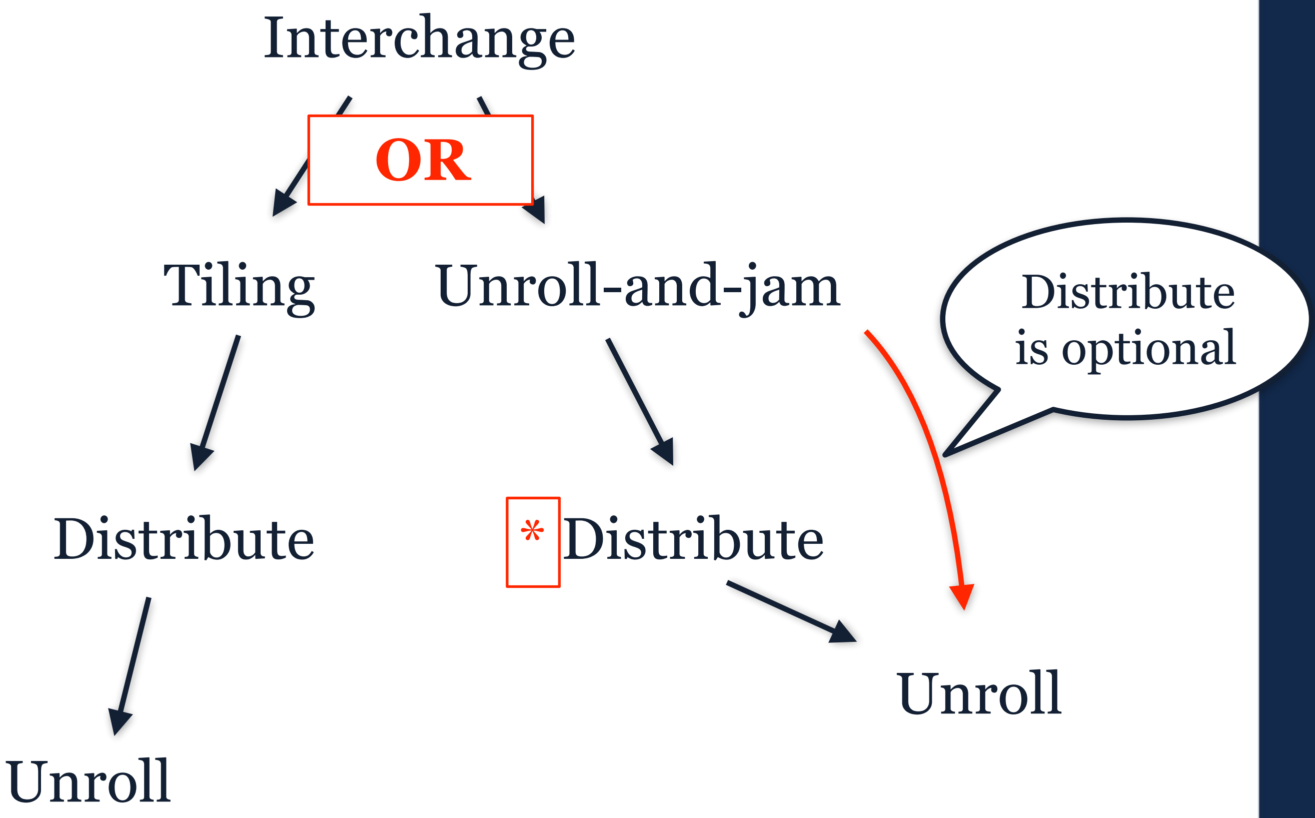
Distribute

Distribute

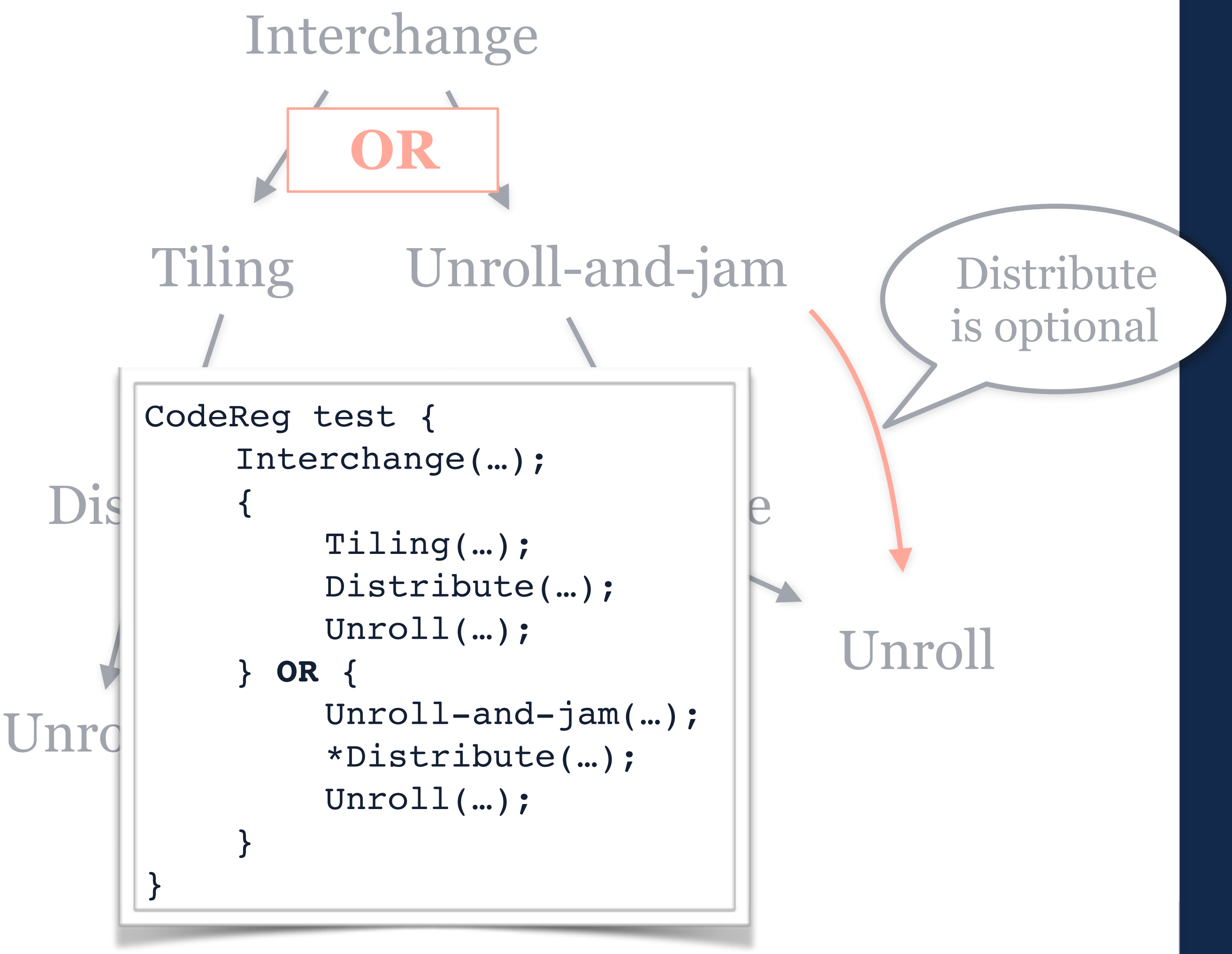
Unroll

Unroll

Locus Optimization Language



Locus Optimization Language



```
CodeReg test {
    Interchange(…);
    {
        Tiling(…);
        Distribute(…);
        Unroll(…);
    } OR {
        Unroll-and-jam(…);
        *Distribute(…);
        Unroll(…);
    }
}
```

Modules Integration 1/3

Modules Integration 1/3

- Collaborative environment, reuse other’s work

Modules Integration 1/3

- Collaborative environment, reuse other’s work
- Locus defines an entire search space

Modules Integration 1/3

- Collaborative environment, reuse other’s work
- Locus defines an entire search space
- Locus allows for both multiple search and transformation modules

Modules Integration 1/3

- Collaborative environment, reuse other’s work
- Locus defines an entire search space
- Locus allows for both multiple search and transformation modules
- Given the search space, one must:
  - decide which variants to evaluate (search module)
  - use tools to generate code that follows each variant’s transformation plan (transformation module)

Modules Integration 2/3

Modules Integration 2/3

- Search modules (OpenTuner, HyperOpt):

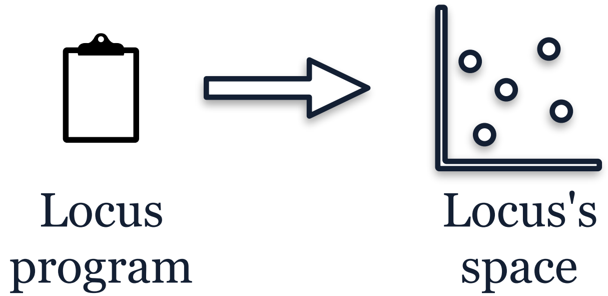
Modules Integration 2/3

- Search modules (OpenTuner, HyperOpt):

Locus
program

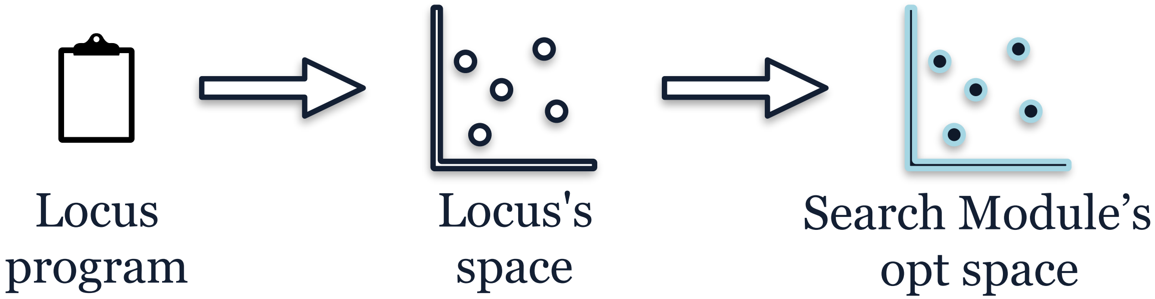
Modules Integration 2/3

- Search modules (OpenTuner, HyperOpt):



Modules Integration 2/3

- Search modules (OpenTuner, HyperOpt):



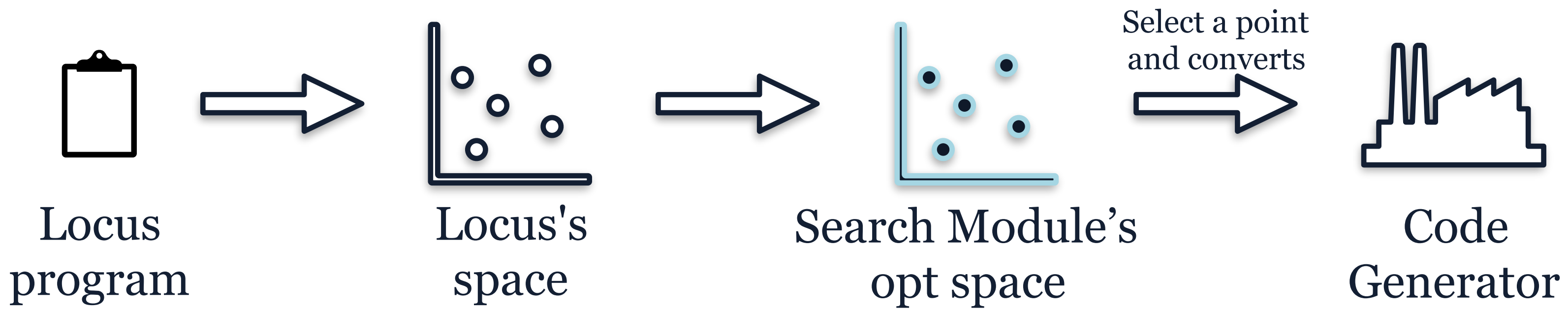
Locus program

Locus's space

Search Module’s opt space

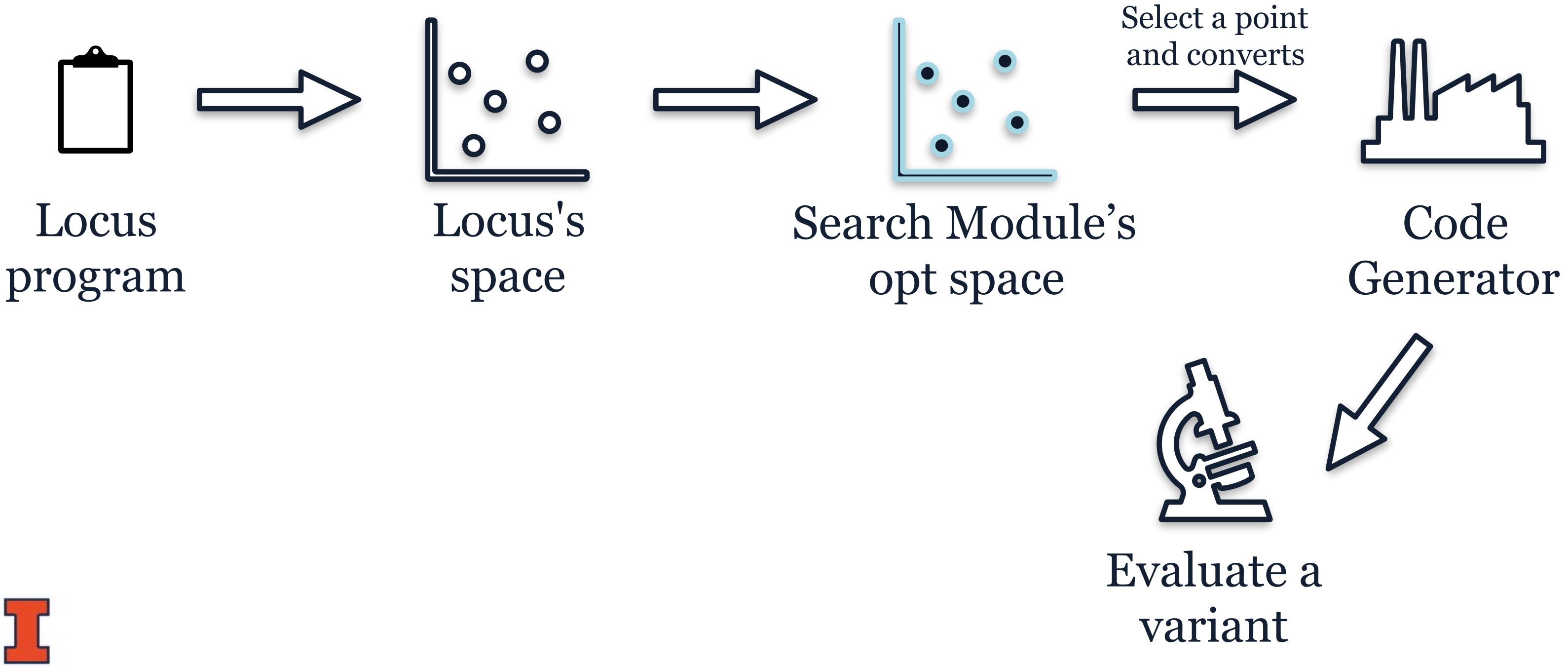
Modules Integration 2/3

- Search modules (OpenTuner, HyperOpt):



Modules Integration 2/3

- Search modules (OpenTuner, HyperOpt):



Locus program

Locus's space

Search Module’s opt space

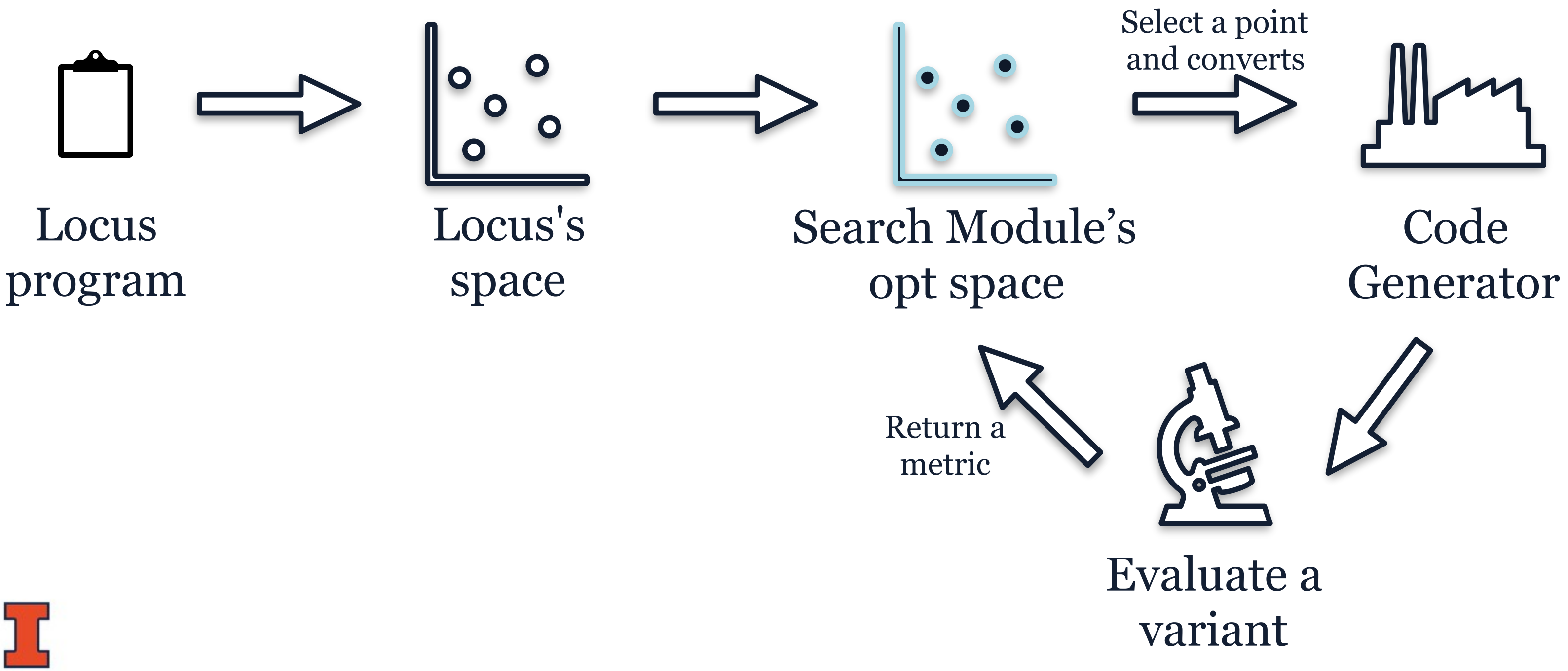
Select a point and converts

Code Generator

Evaluate a variant

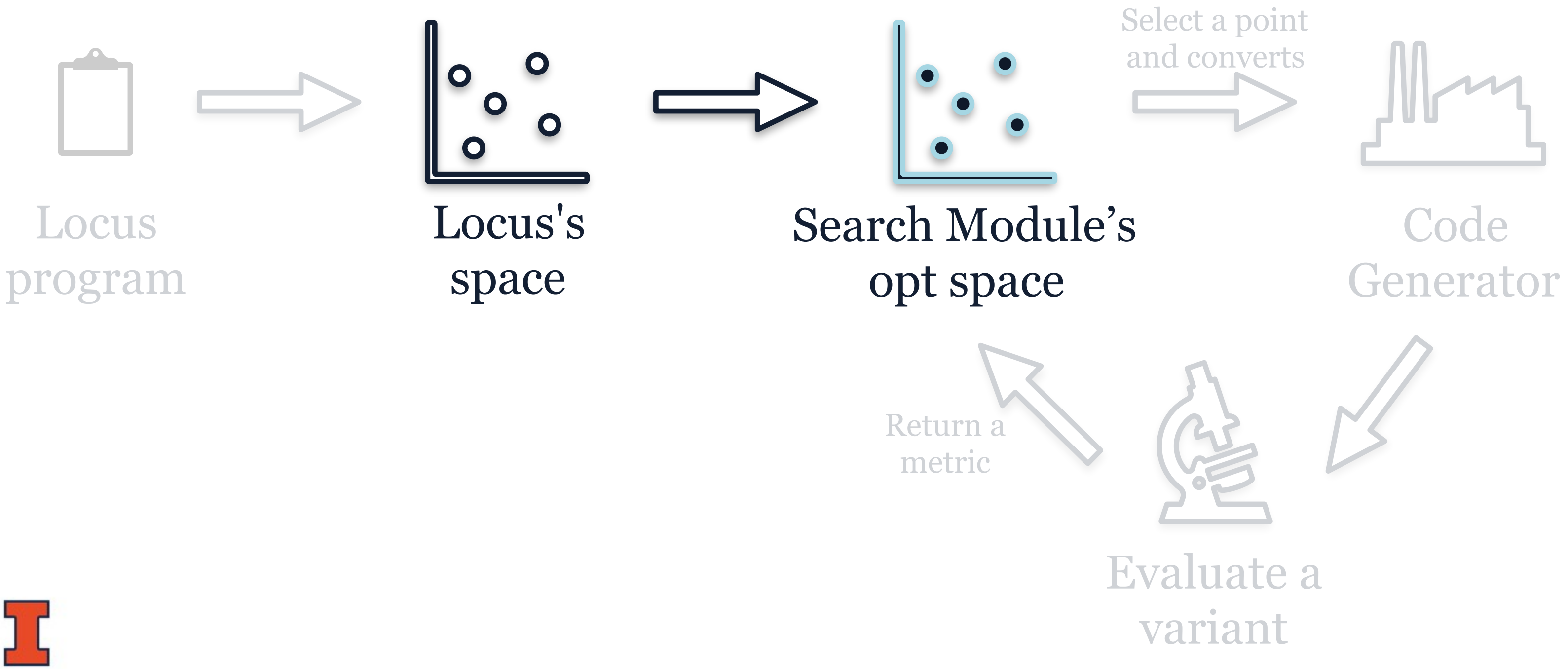
Modules Integration 2/3

- Search modules (OpenTuner, HyperOpt):



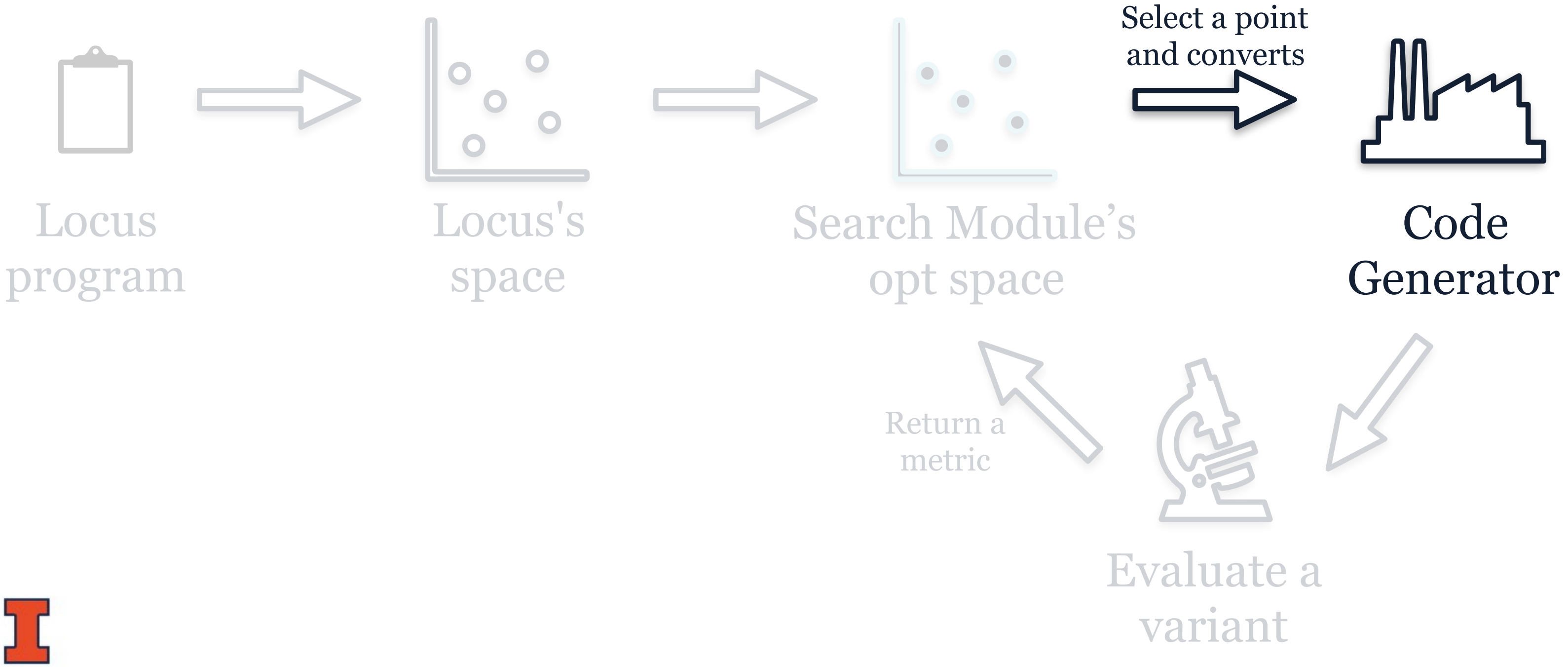
Modules Integration 2/3

- Search modules (OpenTuner, HyperOpt):
  - Convert the Locus' space to module's space
    - parameters, OR statements and blocks, conditionals



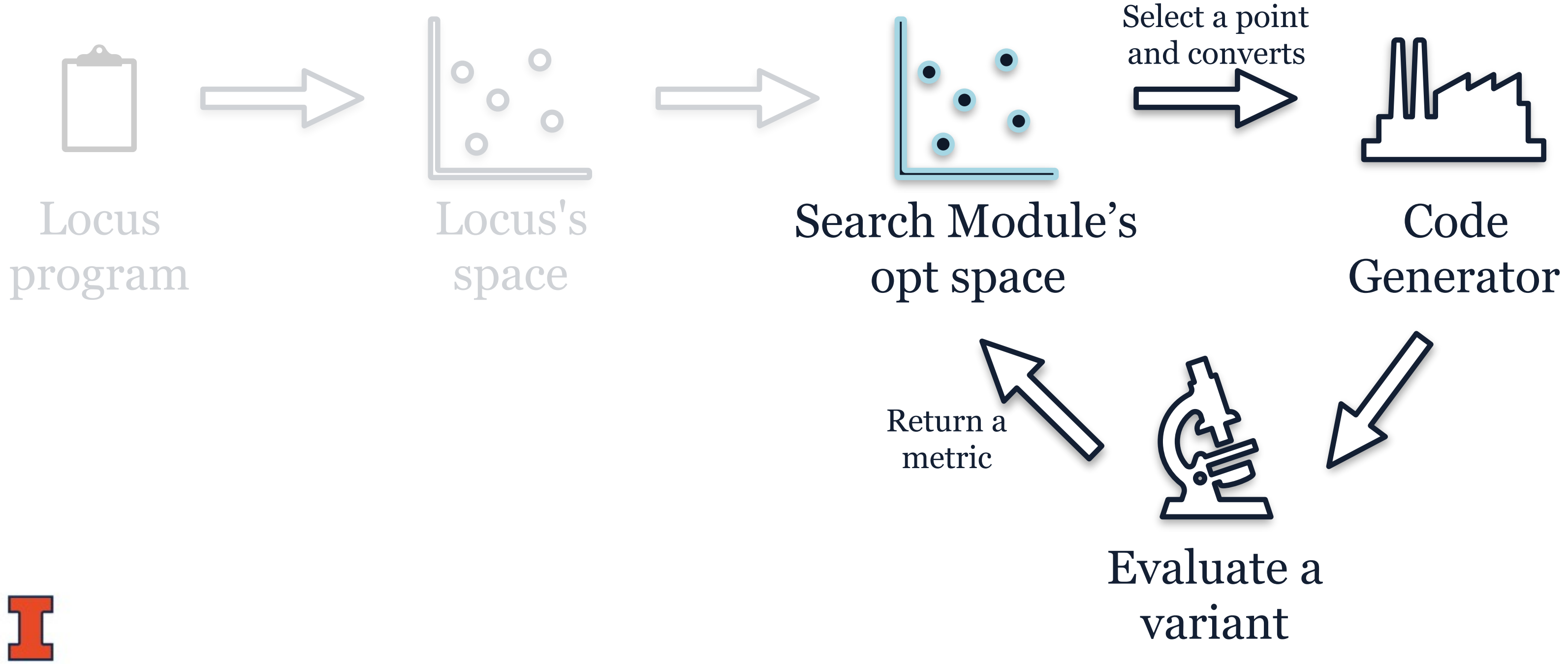
Modules Integration 2/3

- Search modules (OpenTuner, HyperOpt):
  - Convert the Locus' space to module's space
    - parameters, OR statements and blocks, conditionals
  - For each point converts it back to Locus representation, and invokes the interpreter



Modules Integration 2/3

- Search modules (OpenTuner, HyperOpt):
  - Convert the Locus' space to module's space
    - parameters, OR statements and blocks, conditionals
  - For each point converts it back to Locus representation, and invokes the interpreter
  - Search: start process



Modules Integration 3/3

Modules Integration 3/3

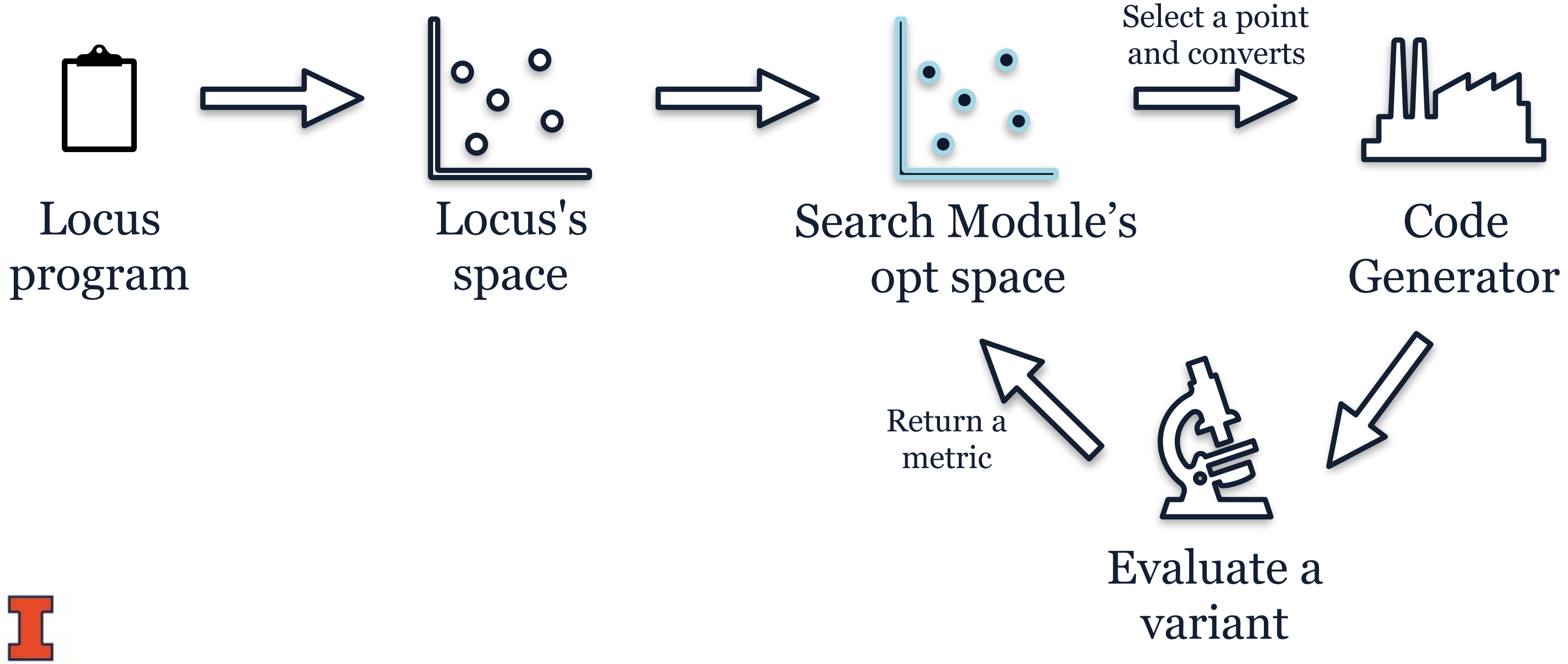
- Transformation modules (Pips, RoseLocus, Pragmas, BuiltIn):
  - Allows for fine-grain selection
    - Can pick a different module for each transformation (e.g., Interchange, Tiling)
  - Work on code region level
  - Workflow:
    - Locus transforms to modules notation
    - Module applies the optimization
    - Locus transforms the resulting code into its internal representation (AST and code region structure)
  - Flexible enough to integrate other transformations if needed

Optimizations for Pruning

- During conversion:
  - Dead code elimination
  - Constant folding
  - Constant propagation

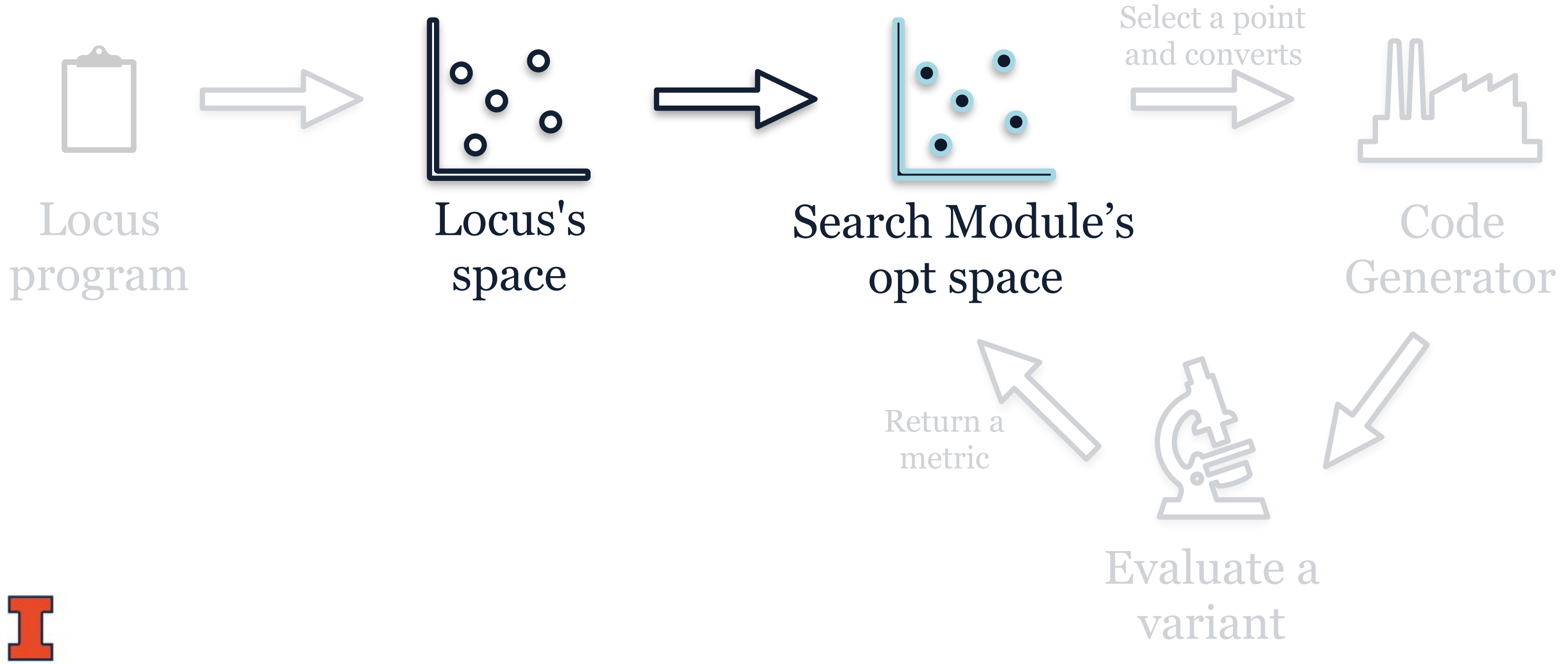
Optimizations for Pruning

- During conversion:
  - Dead code elimination
  - Constant folding
  - Constant propagation



Optimizations for Pruning

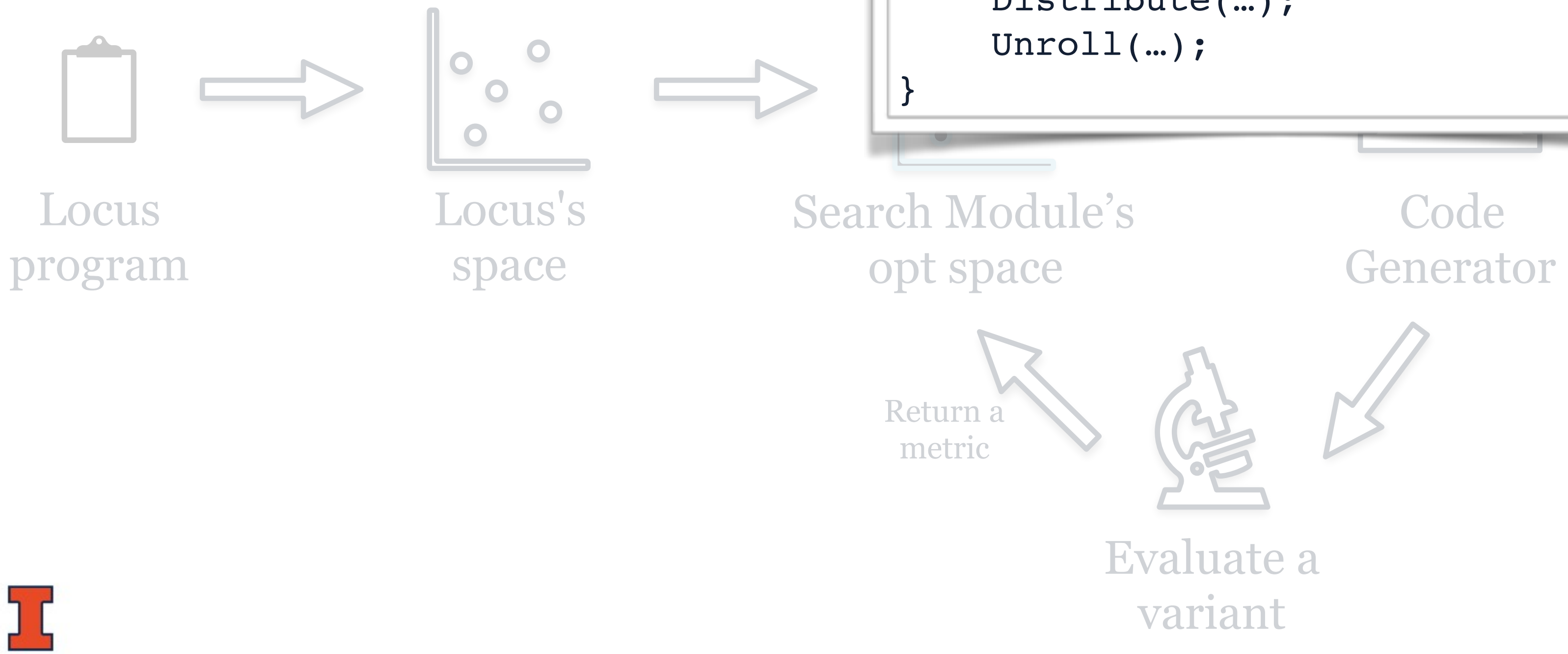
- During conversion:
  - Dead code elimination
  - Constant folding
  - Constant propagation



Optimizations for Pruning

- During conversion:
  - Dead code elimination
  - Constant folding
  - Constant propagation

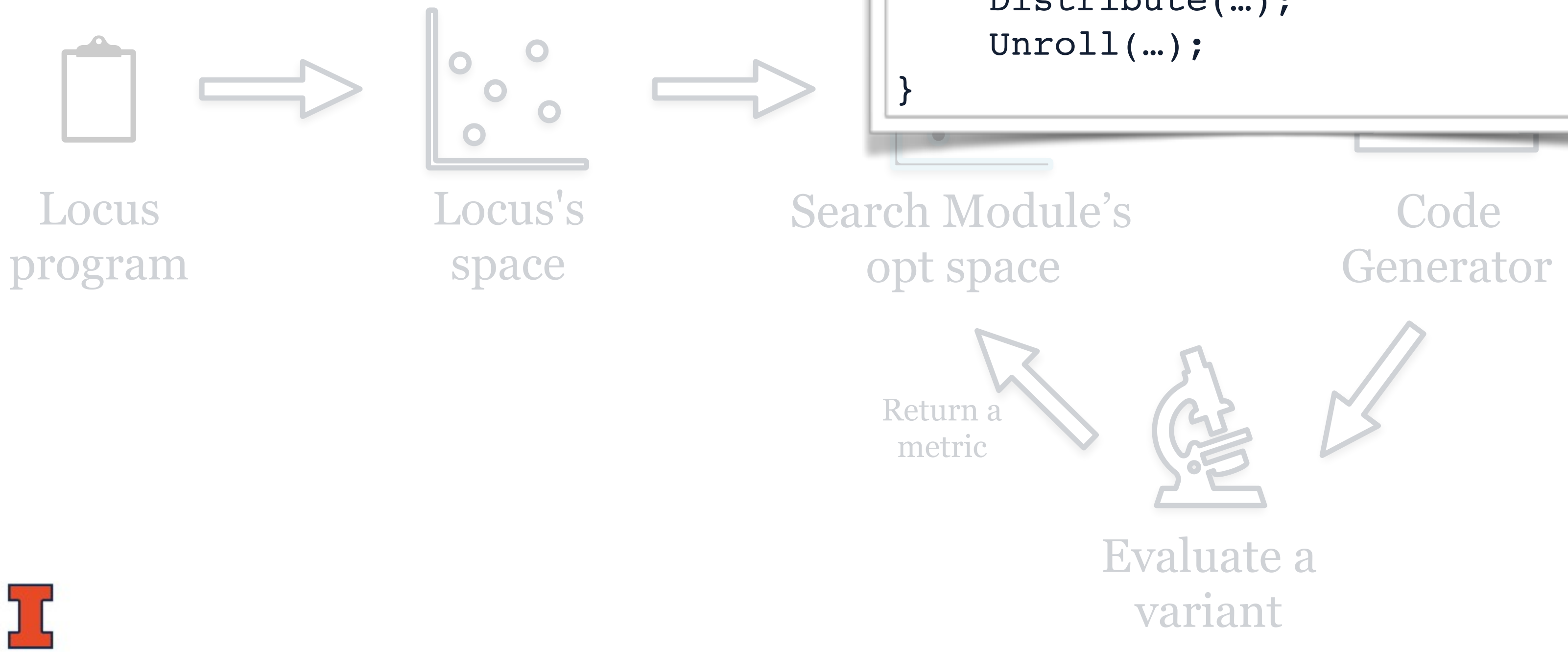
```
CodeReg test {
      perfect = IsPerfectLoopNest();
      if (perfect)
      {
            Interchange(…);
      }
      Tiling(…);
      Distribute(…);
      Unroll(…);
}
```



Optimizations for Pruning

- During conversion:
  - Dead code elimination
  - Constant folding
  - Constant propagation

```
CodeReg test {
     perfect =          False          ;
     if (perfect)
     {
          Interchange(…);
     }
     Tiling(…);
     Distribute(…);
     Unroll(…);
}
```



Locus program

Locus's space

Search Module's opt space

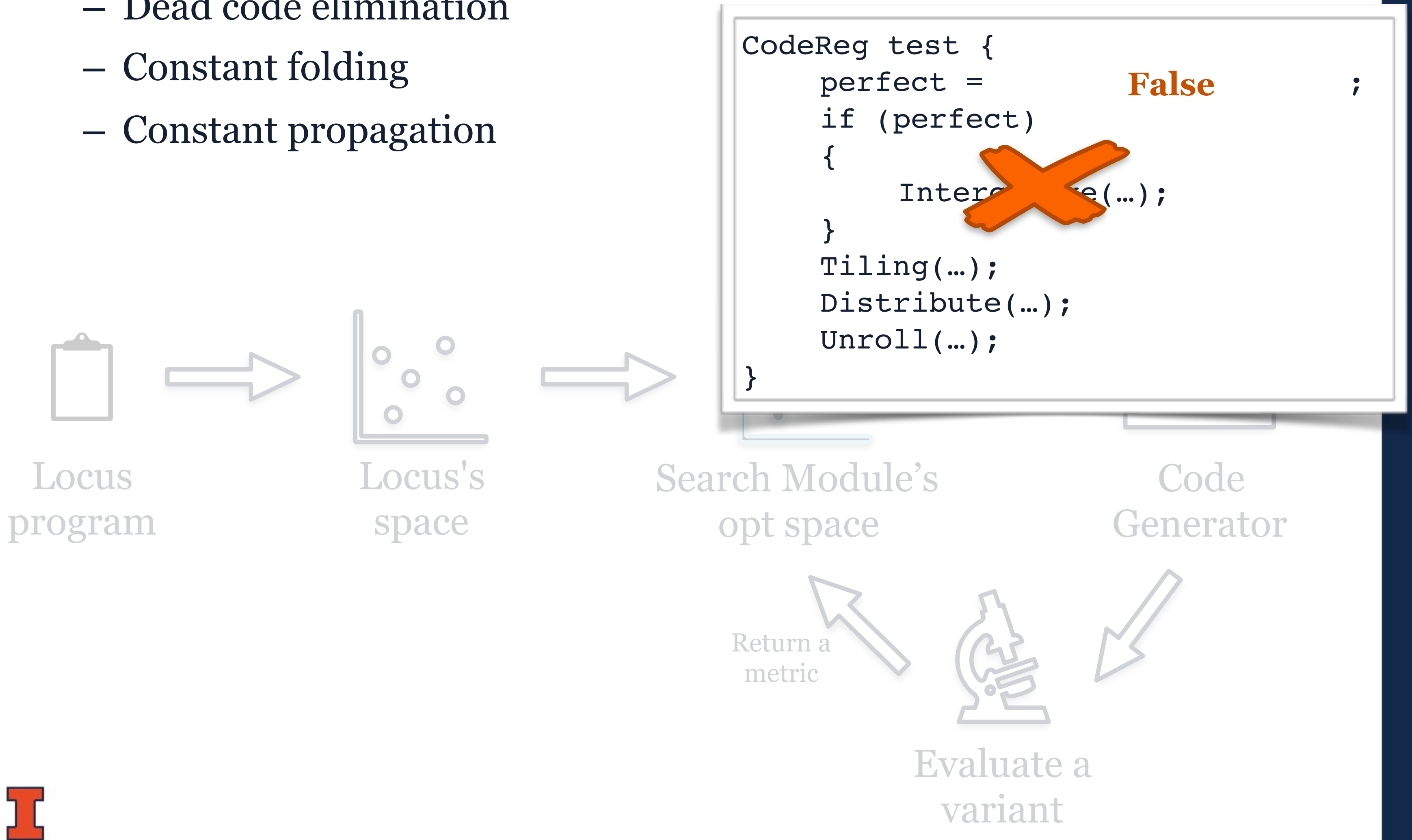
Code Generator

Return a metric

Evaluate a variant

Optimizations for Pruning

- During conversion:
  - Dead code elimination
  - Constant folding
  - Constant propagation

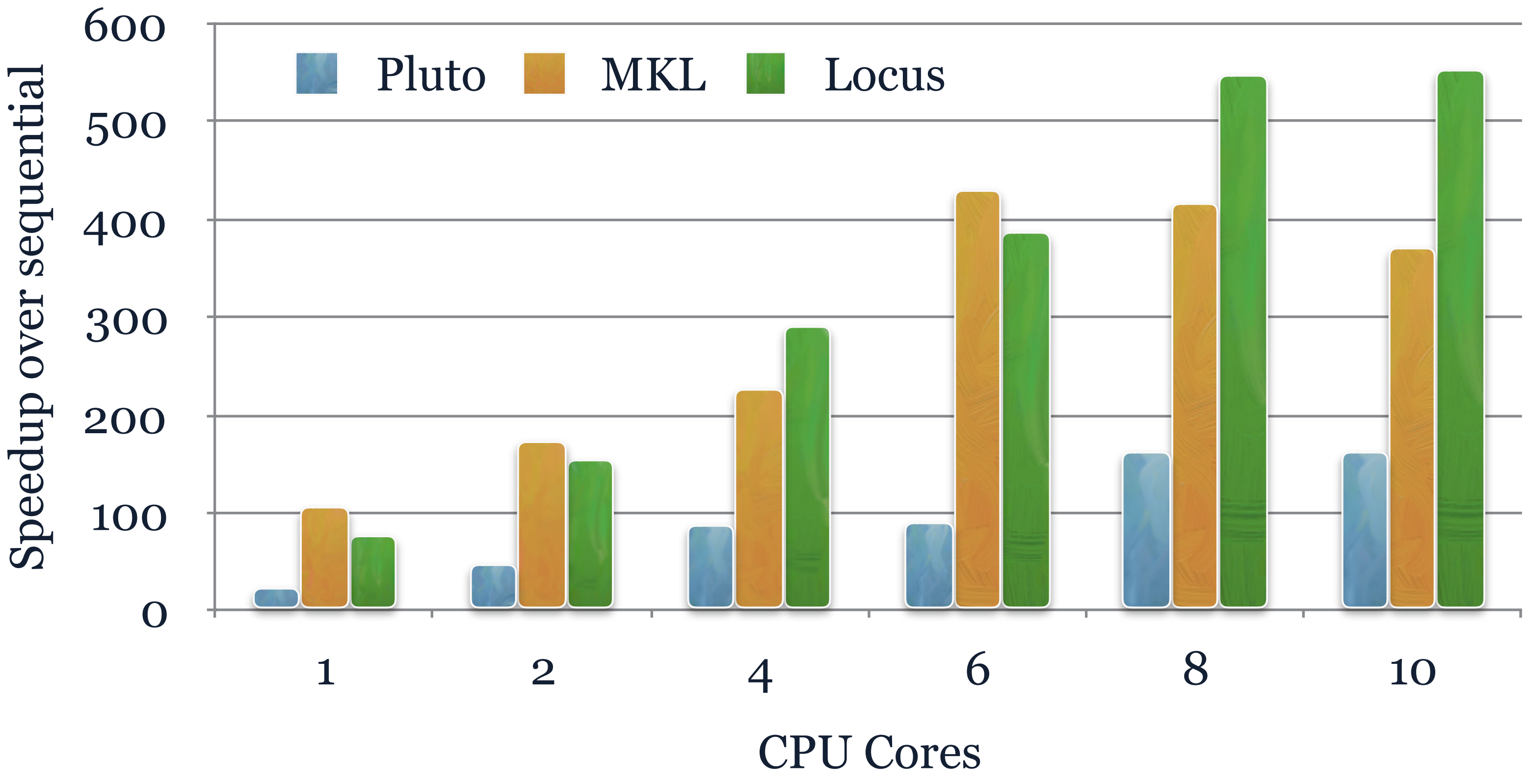


```
CodeReg test {
     perfect =           False         ;
     if (perfect)
     {
          Interchange(…);
     }
     Tiling(…);
     Distribute(…);
     Unroll(…);
}
```

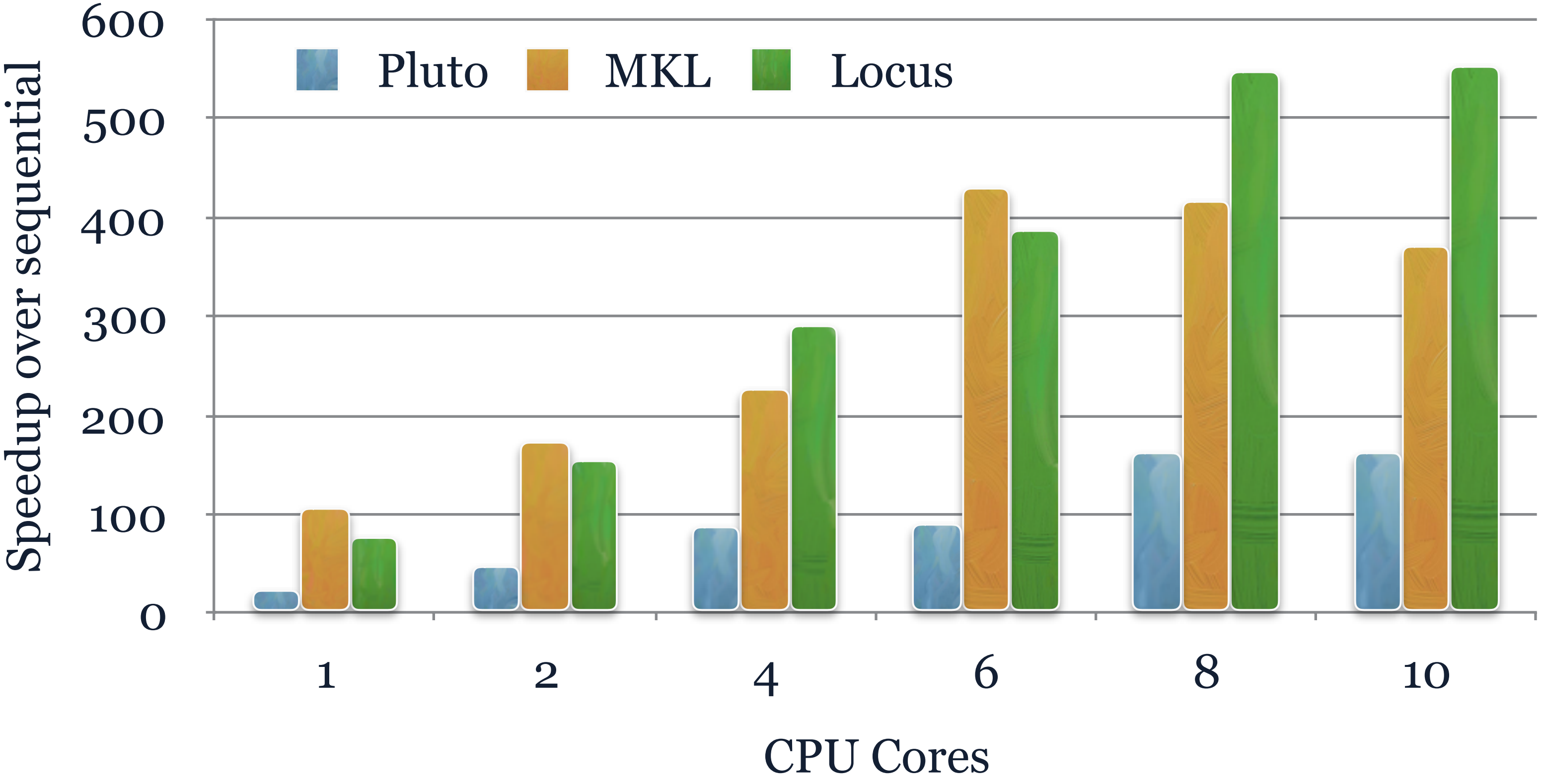
Experimental Results

- Intel Xeon E5-2660 10-Core 2.60 GHz
- Compared to Pluto and Intel MKL
  - Default values for parameters, no search
- Examples:
  - Matrix-Matrix Multiplication
  - Stencil Kernels
  - Kripke
  - Arbitrary Loop Nests
- Generic enough to be applied on known and unknown code applications

Matrix-Matrix Multiplication



Matrix-Matrix Multiplication



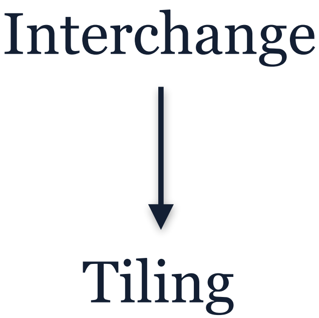
- Empirical search could find very efficient variants
- Comparable with Intel MKL performance

Matrix-Matrix Multiplication

Matrix-Matrix Multiplication

Interchange

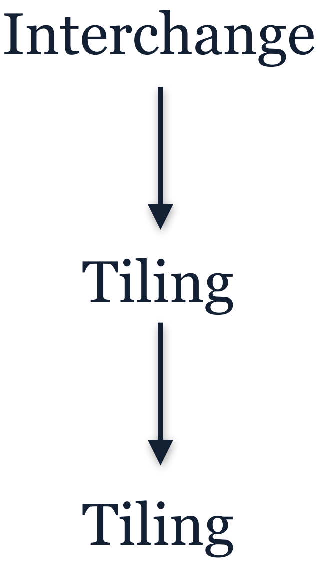
Matrix-Matrix Multiplication



Interchange

Tiling

Matrix-Matrix Multiplication

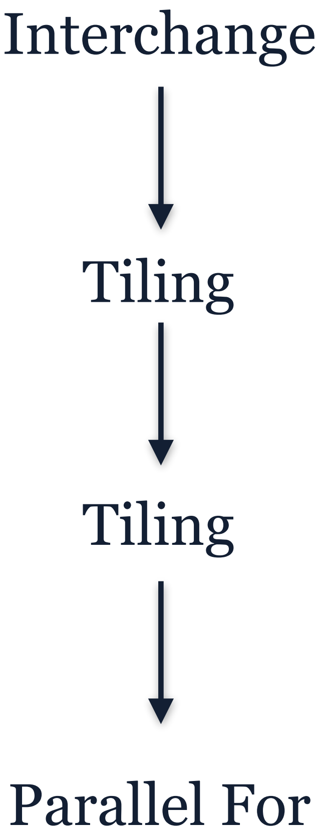


Interchange

Tiling

Tiling

Matrix-Matrix Multiplication



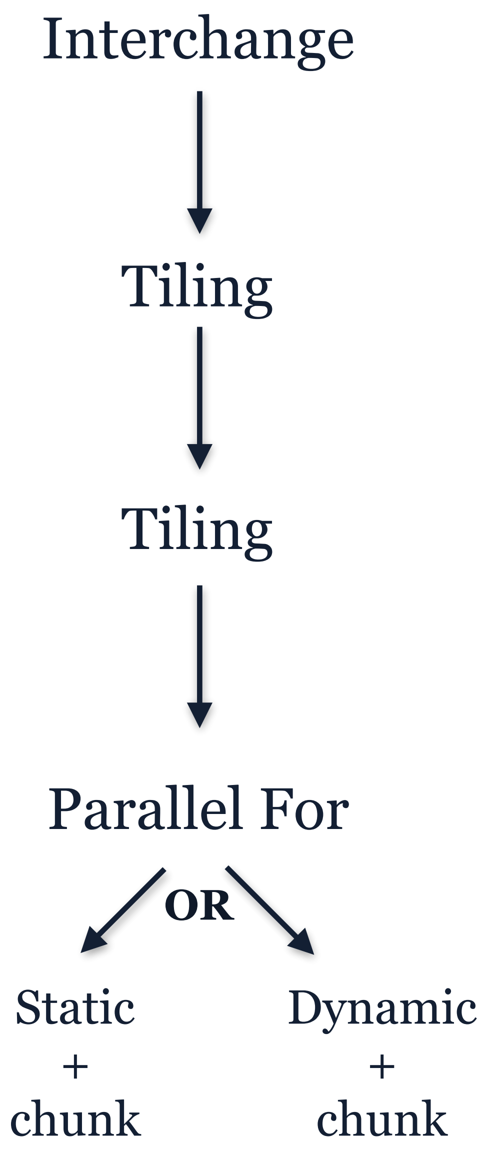
Interchange

Tiling

Tiling

Parallel For

Matrix-Matrix Multiplication



Interchange

Tiling

Tiling

Parallel For

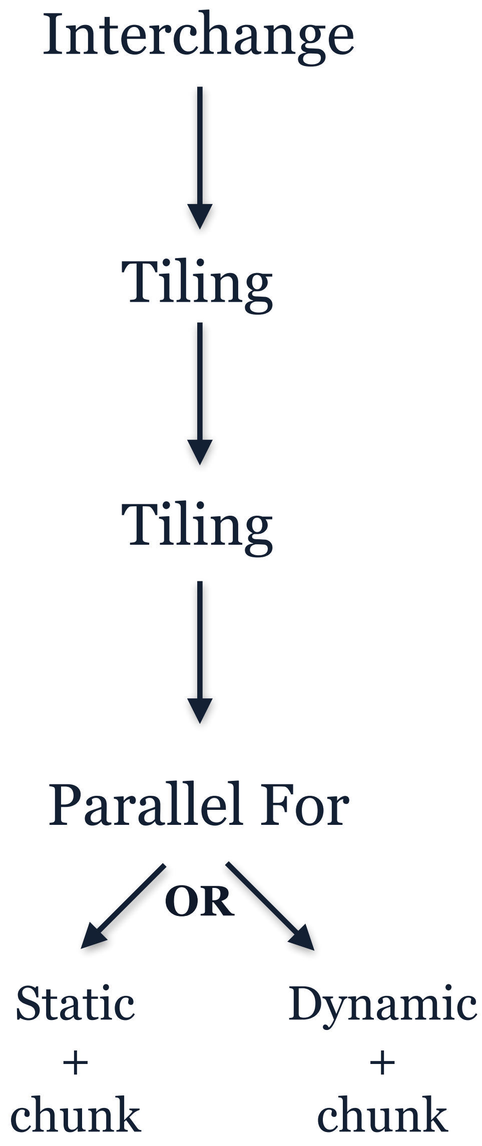
OR

Static + chunk

Dynamic + chunk

Matrix-Matrix Multiplication

- Large space of optimization



Interchange

Tiling

Tiling

Parallel For

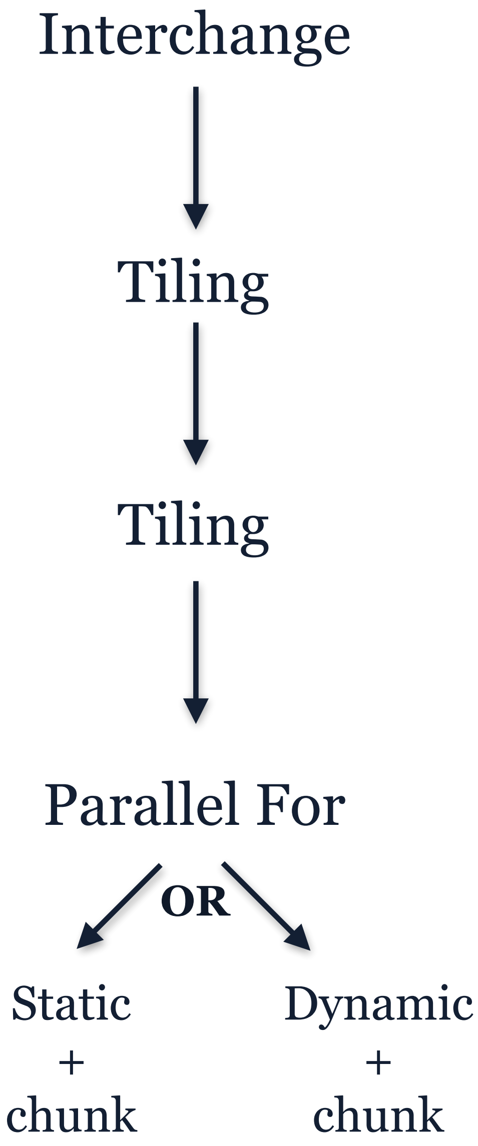
OR

Static + chunk

Dynamic + chunk

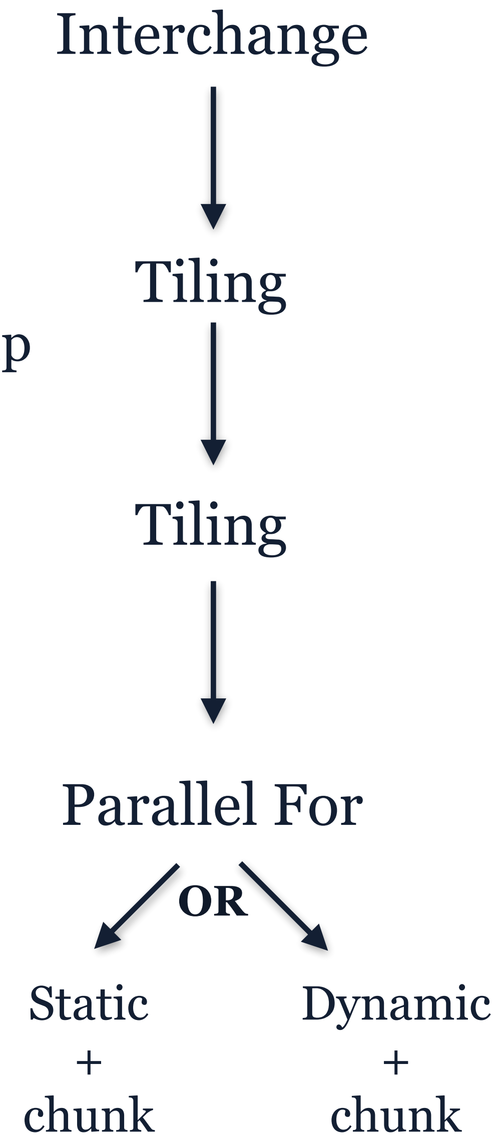
Matrix-Matrix Multiplication

- Large space of optimization
- 34,012,224 possible variants



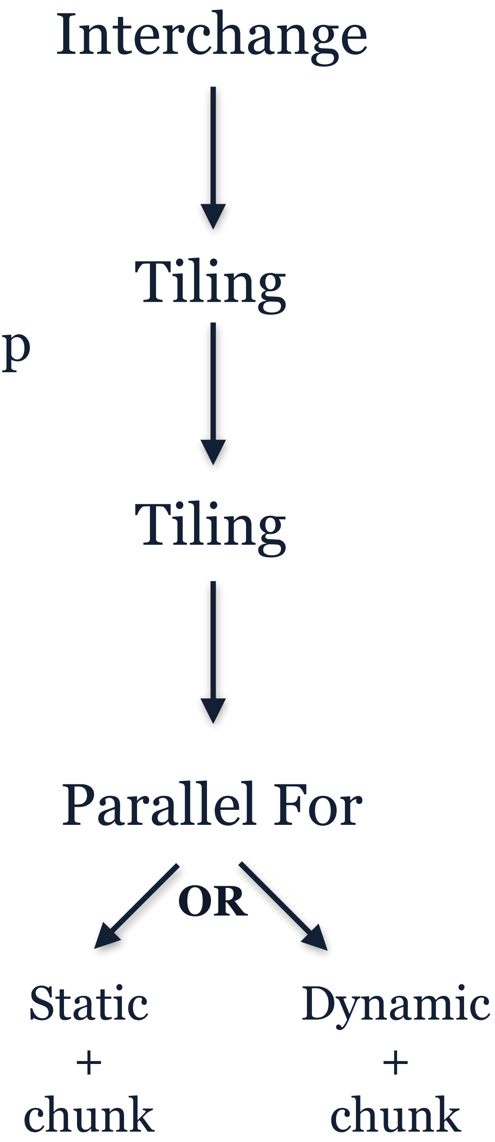
Matrix-Matrix Multiplication

- Large space of optimization
- 34,012,224 possible variants
- Average of ~450 variants evaluated per setup



Matrix-Matrix Multiplication

- Large space of optimization
- 34,012,224 possible variants
- Average of ~450 variants evaluated per setup
- 80 minutes search per setup



Interchange

Tiling

Tiling

Parallel For

OR

Static
+
chunk

Dynamic
+
chunk

Stencils

Stencils

- 6 different stencils

Stencils

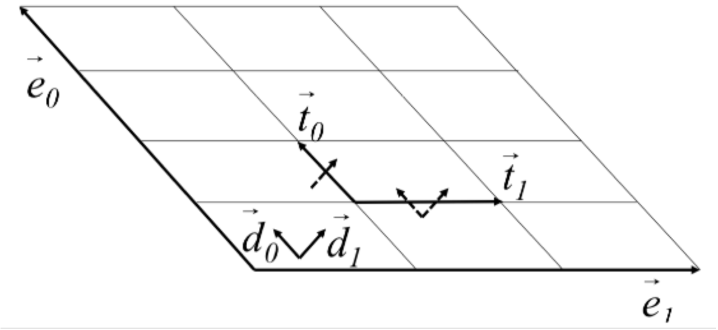
- 6 different stencils
- Skew tiling accross time-space

Stencils

- 6 different stencils
- Skew tiling accross time-space
- Found better tiling shapes

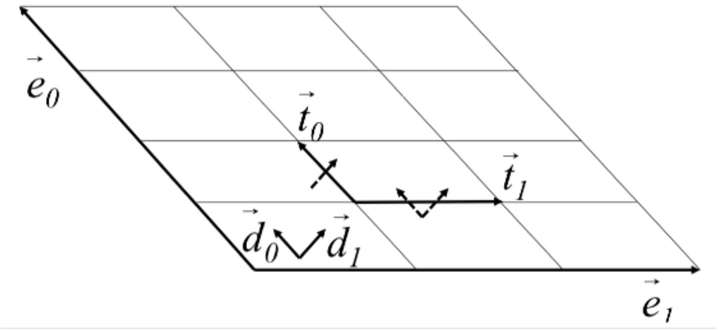
Stencils

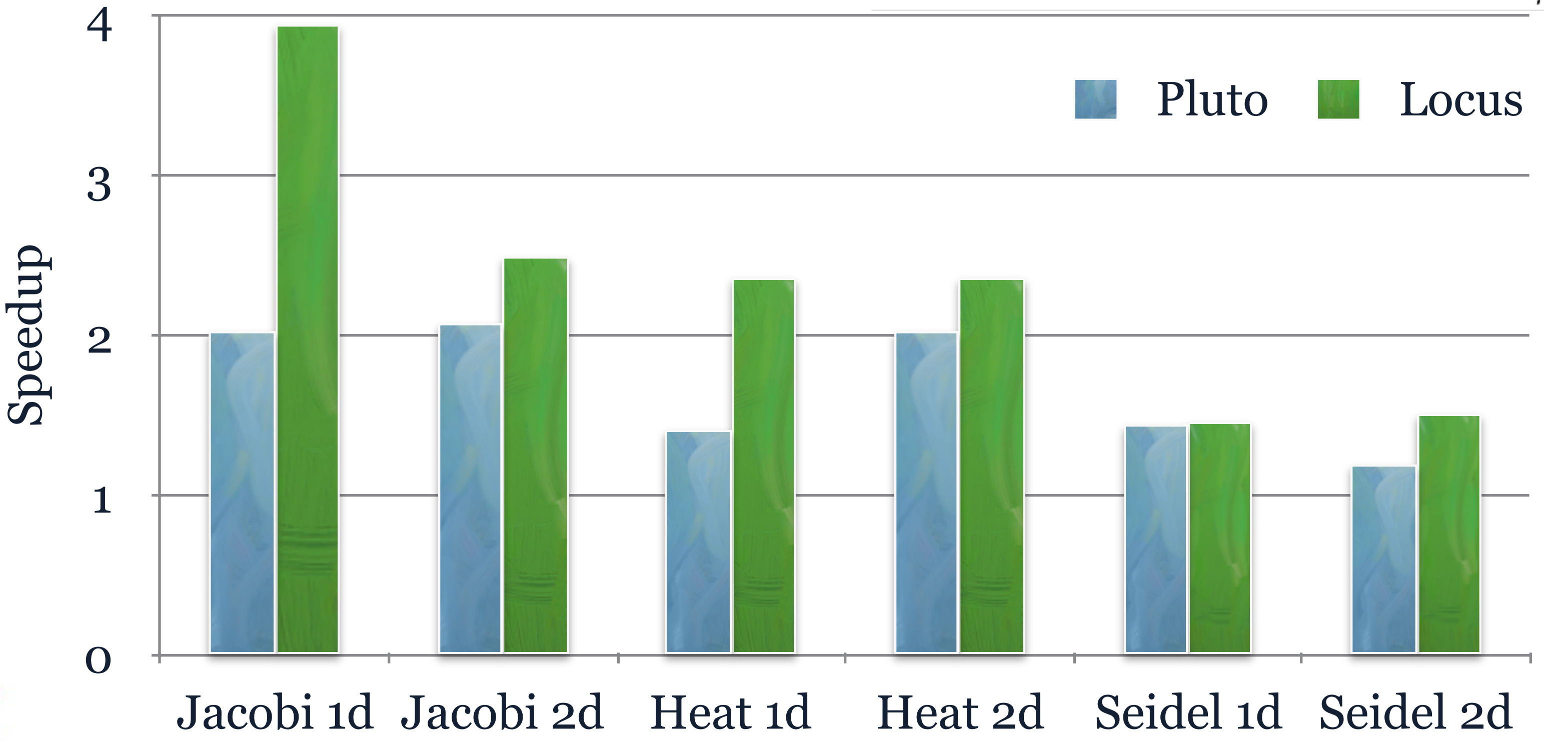
- 6 different stencils
- Skew tiling accross time-space
- Found better tiling shapes



Stencils

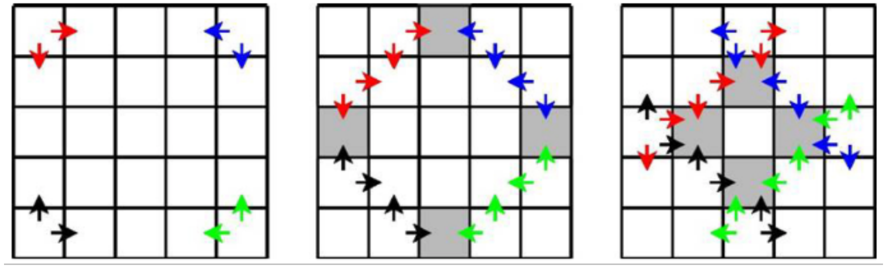
- 6 different stencils
- Skew tiling accross time-space
- Found better tiling shapes



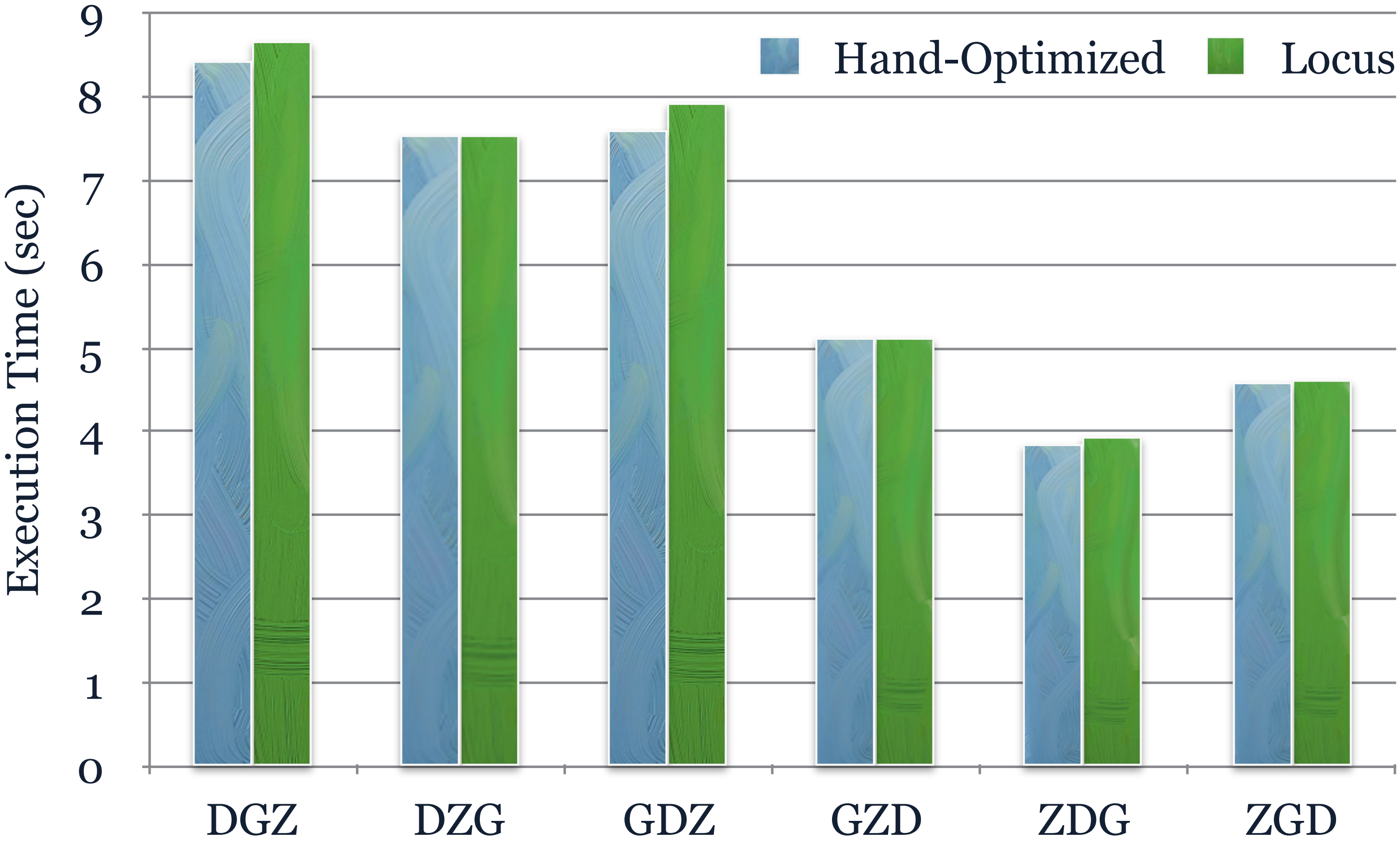


Kripke

- Deterministic particle transport code and proxy-app for the Ardra project developed at LLNL
- 5 kernels: LTimes, LPlusTimes, Scattering , Source, and Sweep
- 6 hand-optimized versions (6 angular fluxes using a 3D array indexed by direction D, group G and zone Z)
- From a single source code generate the 6 hand-optimized versions using Locus

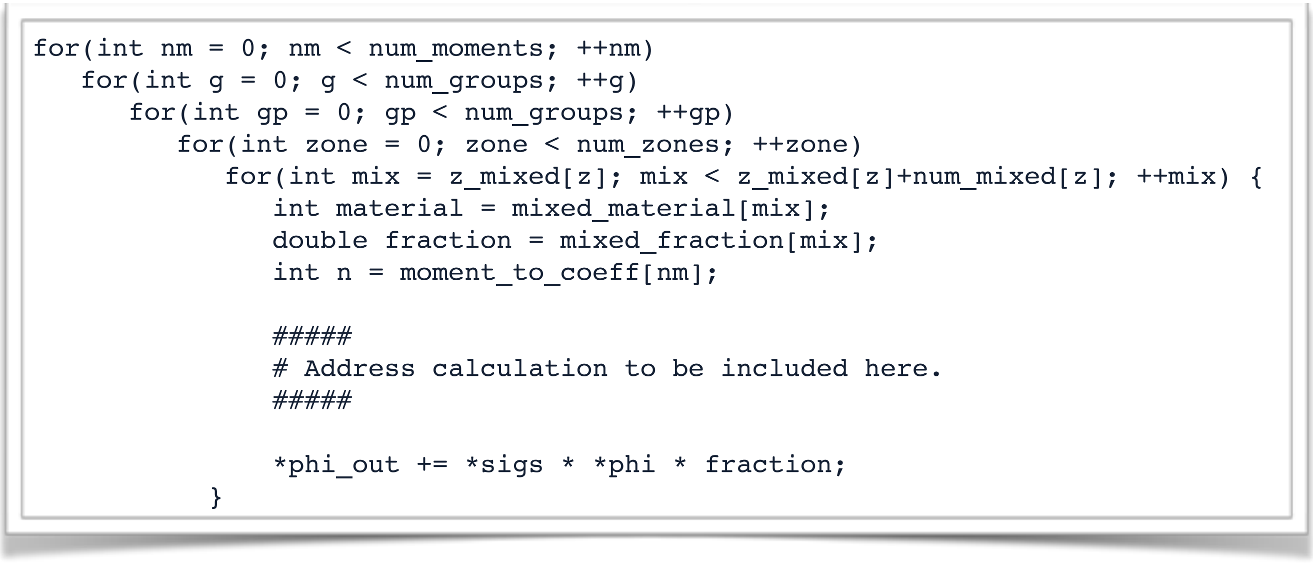


Kripke



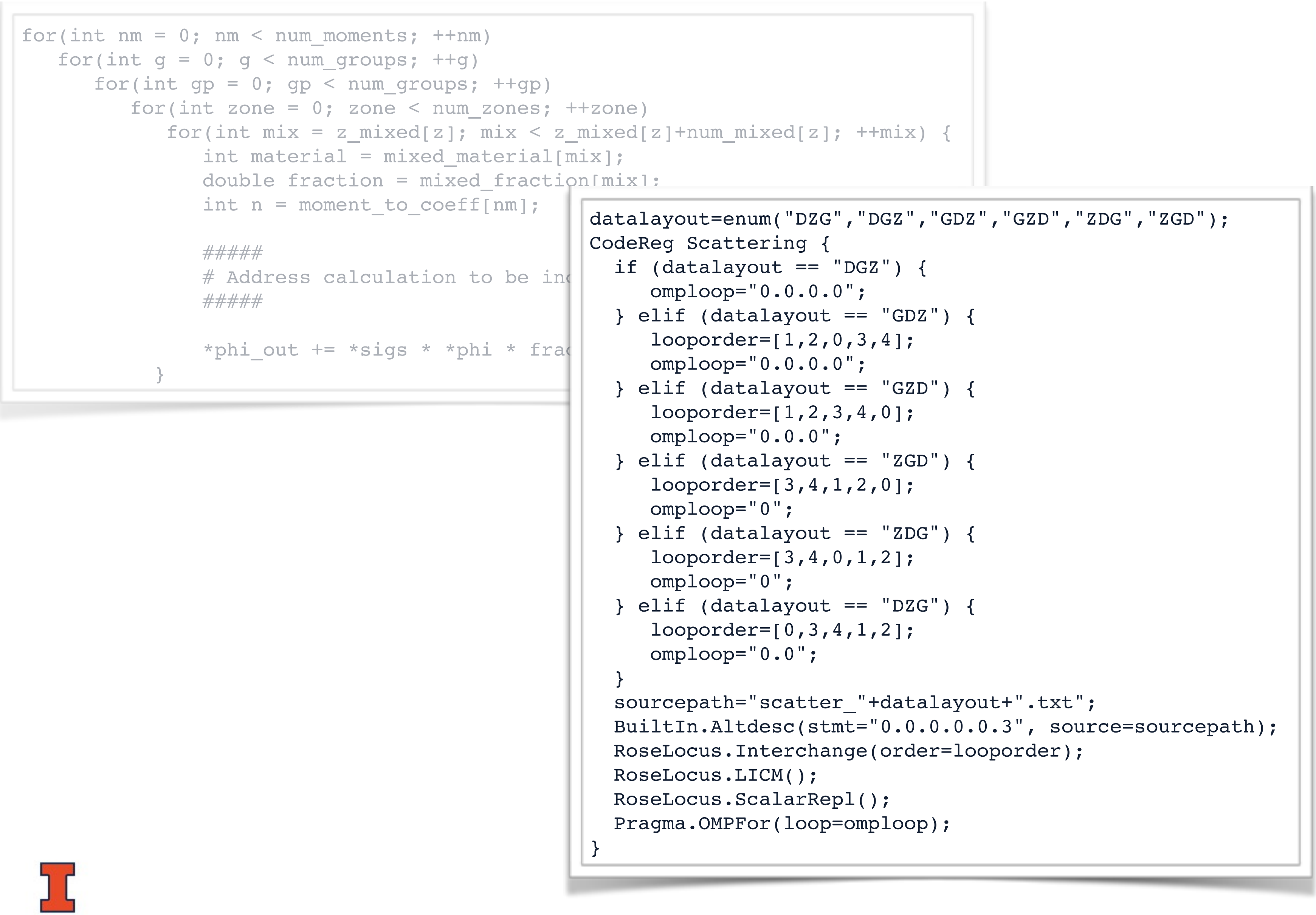
Execution Time (sec)
0
1
2
3
4
5
6
7
8
9
Hand-Optimized
Locus
DGZ
DZG
GDZ
GZD
ZDG
ZGD

Kripke - Scattering Kernel



```
for(int nm = 0; nm < num_moments; ++nm)
   for(int g = 0; g < num_groups; ++g)
      for(int gp = 0; gp < num_groups; ++gp)
         for(int zone = 0; zone < num_zones; ++zone)
            for(int mix = z_mixed[z]; mix < z_mixed[z]+num_mixed[z]; ++mix) {
               int material = mixed_material[mix];
               double fraction = mixed_fraction[mix];
               int n = moment_to_coeff[nm];

               #####
               # Address calculation to be included here.
               #####

               *phi_out += *sigs * *phi * fraction;
           }
```

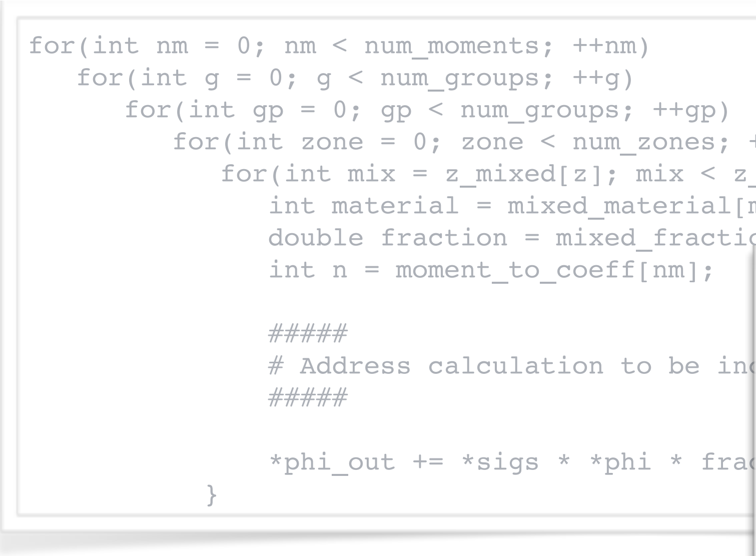
Kripke - Scattering Kernel



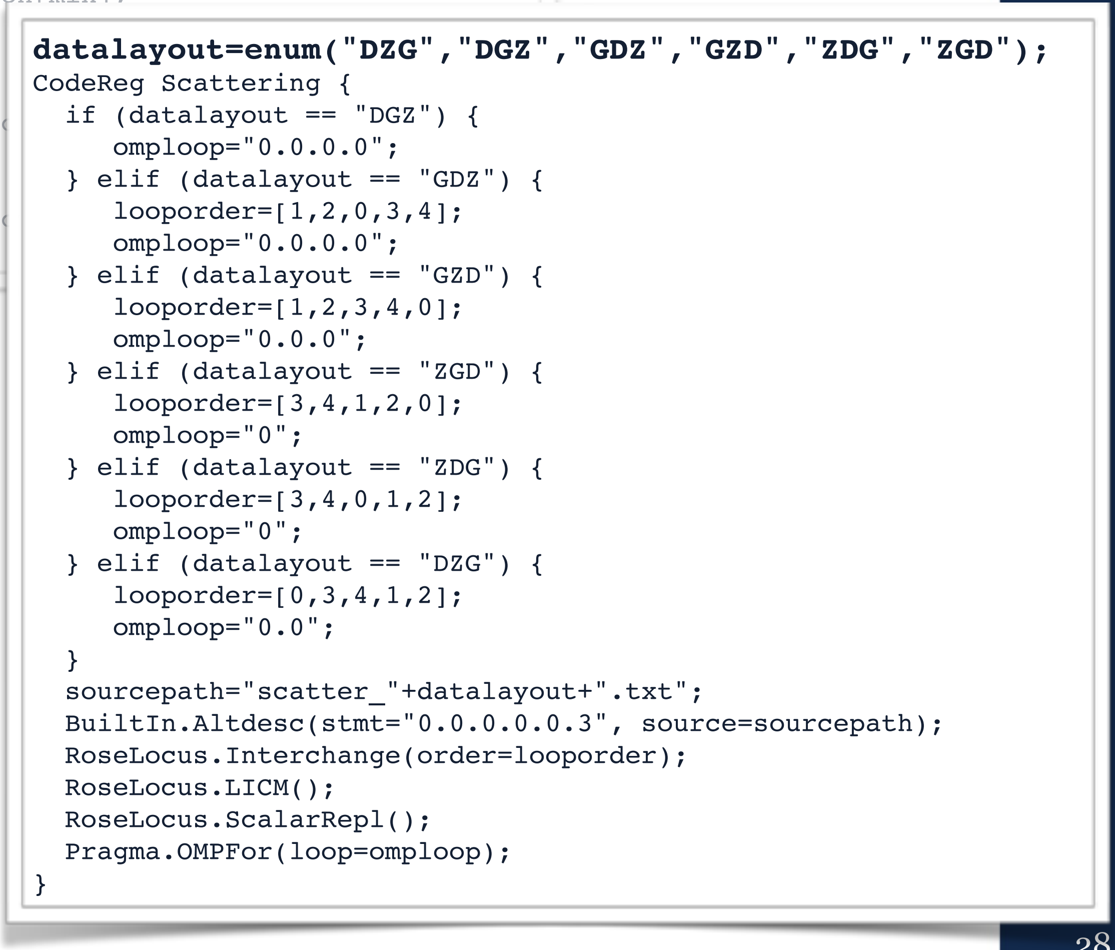
```
for(int nm = 0; nm < num_moments; ++nm)
   for(int g = 0; g < num_groups; ++g)
      for(int gp = 0; gp < num_groups; ++gp)
         for(int zone = 0; zone < num_zones; ++zone)
            for(int mix = z_mixed[z]; mix < z_mixed[z]+num_mixed[z]; ++mix) {
               int material = mixed_material[mix];
               double fraction = mixed_fraction[mix];
               int n = moment_to_coeff[nm];

               #####
               # Address calculation to be in
               #####

               *phi_out += *sigs * *phi * frac
           }
```

```
datalayout=enum("DZG","DGZ","GDZ","GZD","ZDG","ZGD");
CodeReg Scattering {
  if (datalayout == "DGZ") {
     omploop="0.0.0.0";
  } elif (datalayout == "GDZ") {
     looporder=[1,2,0,3,4];
     omploop="0.0.0.0";
  } elif (datalayout == "GZD") {
     looporder=[1,2,3,4,0];
     omploop="0.0.0";
  } elif (datalayout == "ZGD") {
     looporder=[3,4,1,2,0];
     omploop="0";
  } elif (datalayout == "ZDG") {
     looporder=[3,4,0,1,2];
     omploop="0";
  } elif (datalayout == "DZG") {
     looporder=[0,3,4,1,2];
     omploop="0.0";
  }
  sourcepath="scatter_"+datalayout+".txt";
  BuiltIn.Altdesc(stmt="0.0.0.0.0.3", source=sourcepath);
  RoseLocus.Interchange(order=looporder);
  RoseLocus.LICM();
  RoseLocus.ScalarRepl();
  Pragma.OMPFor(loop=omploop);
}
```

Kripke - Scattering Kernel

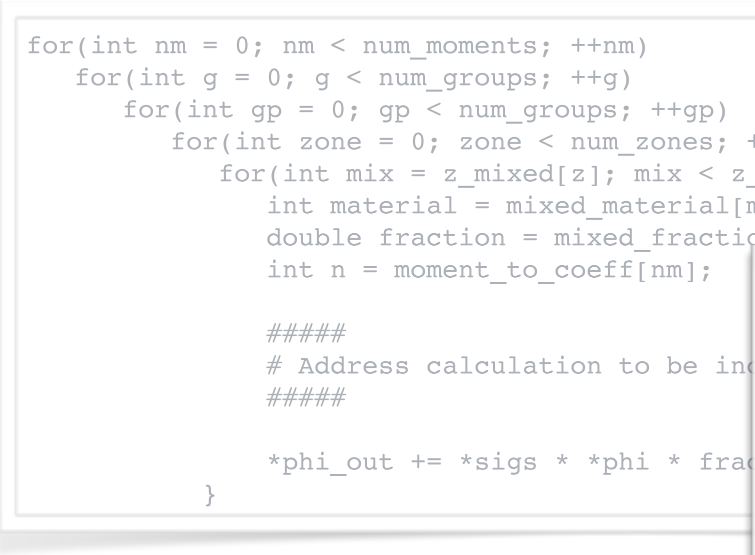


```
for(int nm = 0; nm < num_moments; ++nm)
   for(int g = 0; g < num_groups; ++g)
      for(int gp = 0; gp < num_groups; ++gp)
         for(int zone = 0; zone < num_zones; ++zone)
            for(int mix = z_mixed[z]; mix < z_mixed[z]+num_mixed[z]; ++mix) {
               int material = mixed_material[mix];
               double fraction = mixed_fraction[mix];
               int n = moment_to_coeff[nm];

               #####
               # Address calculation to be in
               #####

               *phi_out += *sigs * *phi * frac
            }
```

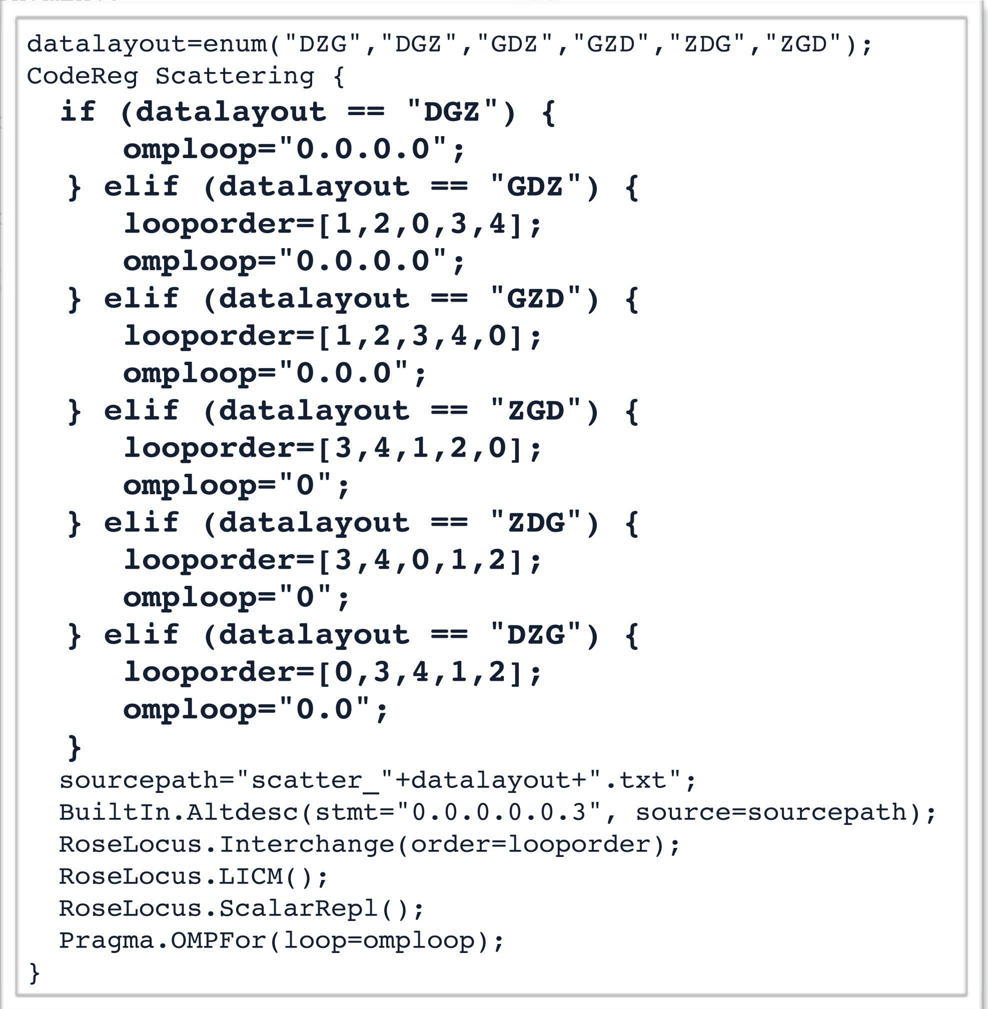


```
datalayout=enum("DZG","DGZ","GDZ","GZD","ZDG","ZGD");
CodeReg Scattering {
  if (datalayout == "DGZ") {
     omploop="0.0.0.0";
  } elif (datalayout == "GDZ") {
     looporder=[1,2,0,3,4];
     omploop="0.0.0.0";
  } elif (datalayout == "GZD") {
     looporder=[1,2,3,4,0];
     omploop="0.0.0";
  } elif (datalayout == "ZGD") {
     looporder=[3,4,1,2,0];
     omploop="0";
  } elif (datalayout == "ZDG") {
     looporder=[3,4,0,1,2];
     omploop="0";
  } elif (datalayout == "DZG") {
     looporder=[0,3,4,1,2];
     omploop="0.0";
  }
  sourcepath="scatter_"+datalayout+".txt";
  BuiltIn.Altdesc(stmt="0.0.0.0.0.3", source=sourcepath);
  RoseLocus.Interchange(order=looporder);
  RoseLocus.LICM();
  RoseLocus.ScalarRepl();
  Pragma.OMPFor(loop=omploop);
}
```

Kripke - Scattering Kernel

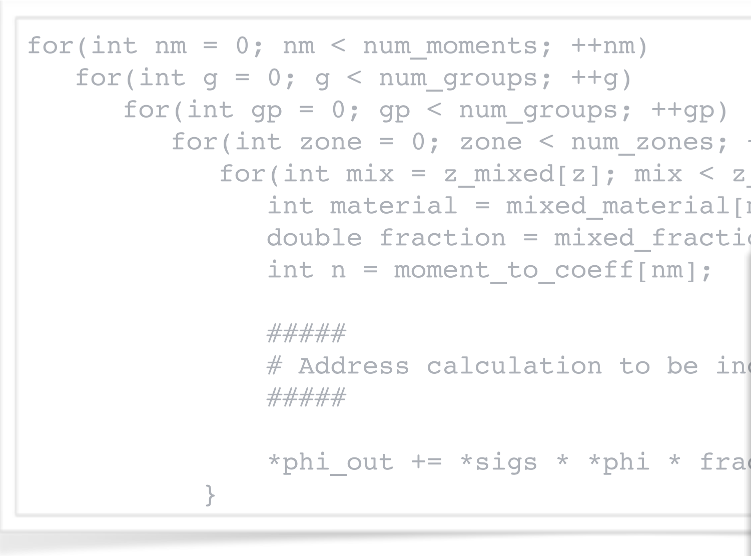


```
for(int nm = 0; nm < num_moments; ++nm)
   for(int g = 0; g < num_groups; ++g)
      for(int gp = 0; gp < num_groups; ++gp)
         for(int zone = 0; zone < num_zones; ++zone)
            for(int mix = z_mixed[z]; mix < z_mixed[z]+num_mixed[z]; ++mix) {
               int material = mixed_material[mix];
               double fraction = mixed_fraction[mix];
               int n = moment_to_coeff[nm];

               #####
               # Address calculation to be in
               #####

               *phi_out += *sigs * *phi * frac
            }
```

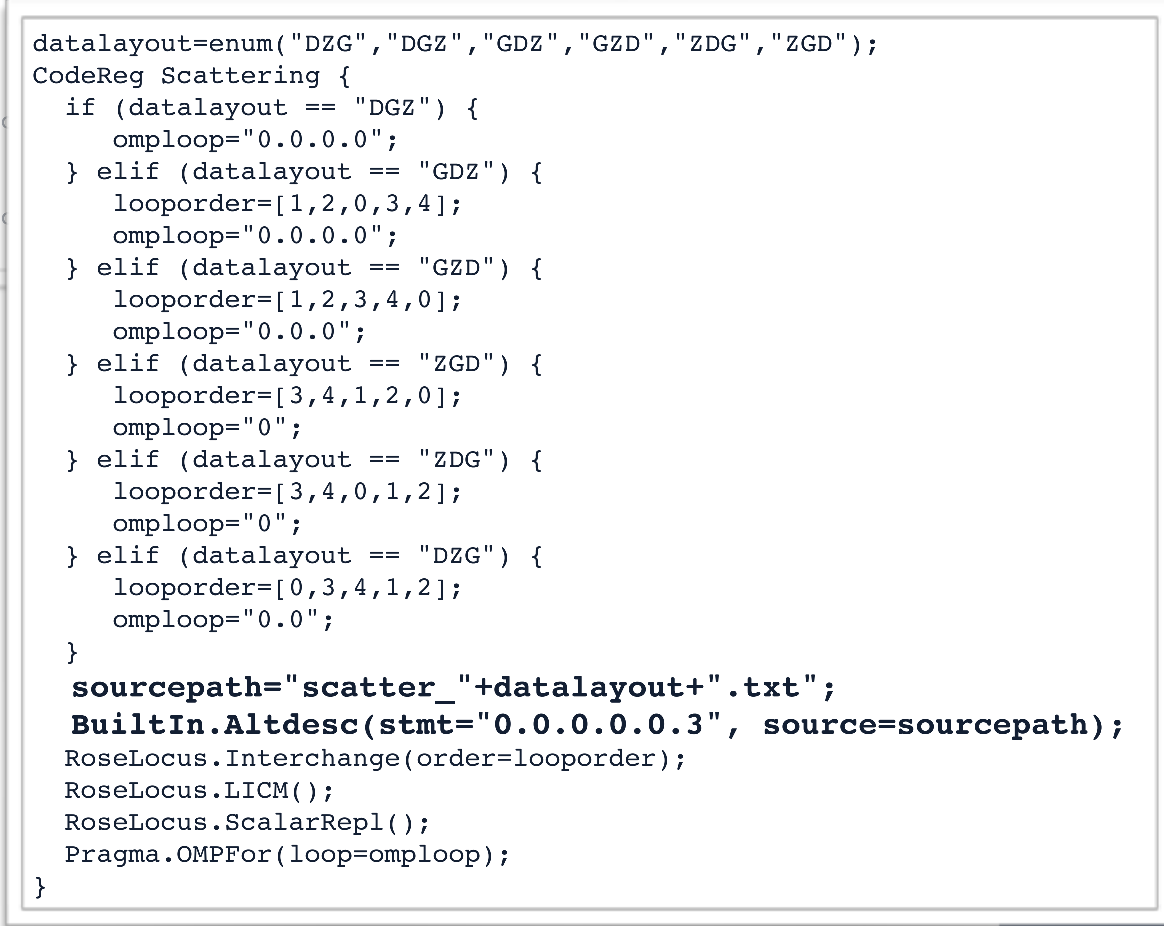


```
datalayout=enum("DZG","DGZ","GDZ","GZD","ZDG","ZGD");
CodeReg Scattering {
  if (datalayout == "DGZ") {
      omploop="0.0.0.0";
   } elif (datalayout == "GDZ") {
      looporder=[1,2,0,3,4];
      omploop="0.0.0.0";
   } elif (datalayout == "GZD") {
      looporder=[1,2,3,4,0];
      omploop="0.0.0";
   } elif (datalayout == "ZGD") {
      looporder=[3,4,1,2,0];
      omploop="0";
   } elif (datalayout == "ZDG") {
      looporder=[3,4,0,1,2];
      omploop="0";
   } elif (datalayout == "DZG") {
      looporder=[0,3,4,1,2];
      omploop="0.0";
   }
  sourcepath="scatter_"+datalayout+".txt";
  BuiltIn.Altdesc(stmt="0.0.0.0.0.3", source=sourcepath);
  RoseLocus.Interchange(order=looporder);
  RoseLocus.LICM();
  RoseLocus.ScalarRepl();
  Pragma.OMPFor(loop=omploop);
}
```

Kripke - Scattering Kernel

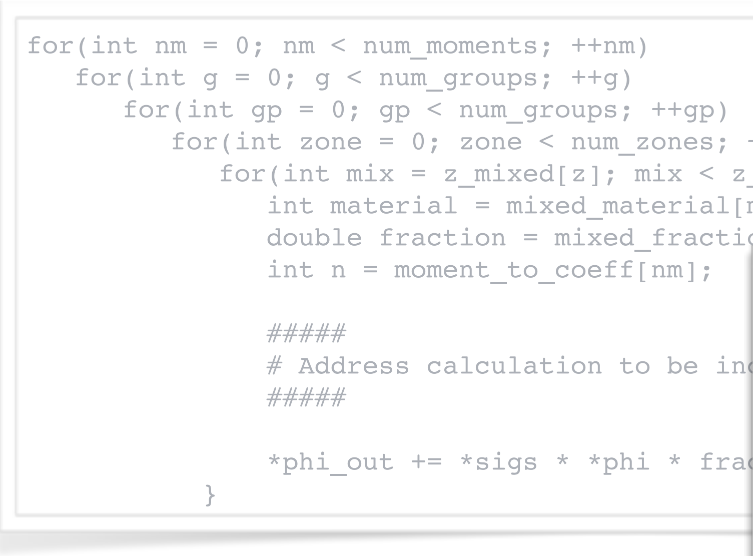


```
for(int nm = 0; nm < num_moments; ++nm)
   for(int g = 0; g < num_groups; ++g)
      for(int gp = 0; gp < num_groups; ++gp)
         for(int zone = 0; zone < num_zones; ++zone)
            for(int mix = z_mixed[z]; mix < z_mixed[z]+num_mixed[z]; ++mix) {
               int material = mixed_material[mix];
               double fraction = mixed_fraction[mix];
               int n = moment_to_coeff[nm];

               #####
               # Address calculation to be in
               #####

               *phi_out += *sigs * *phi * frac
           }
```

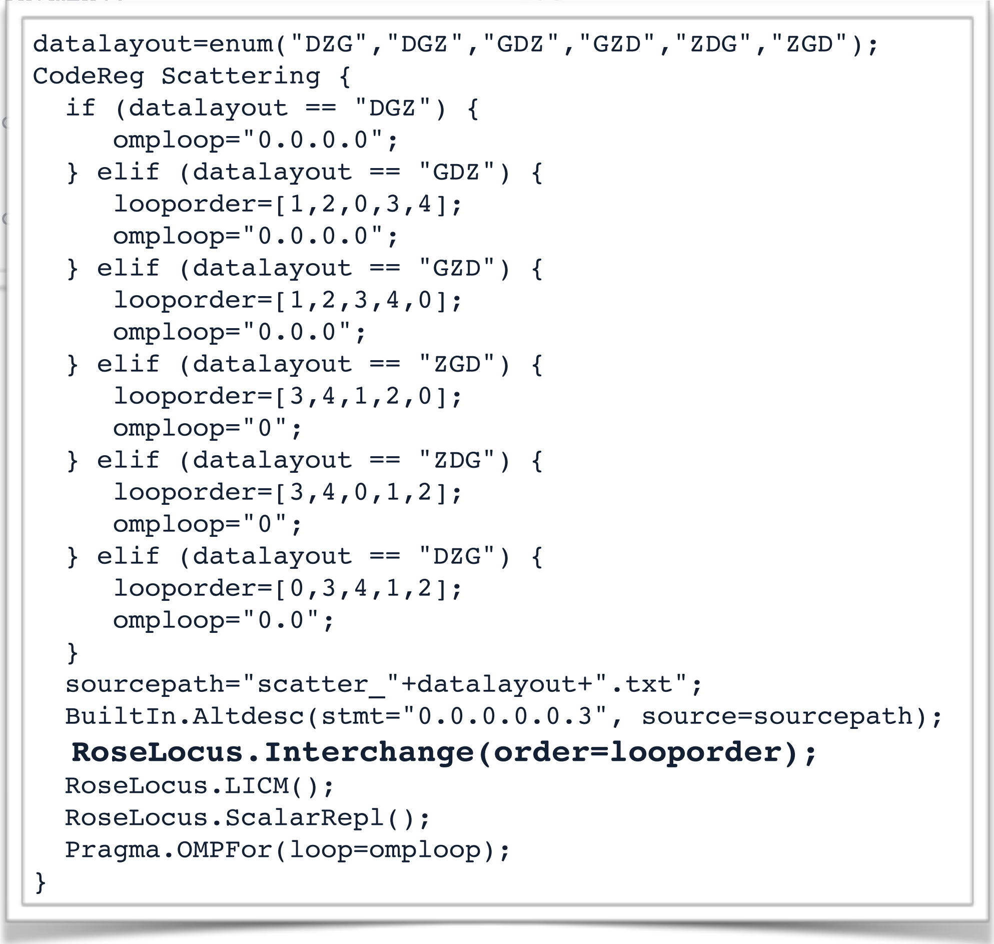


```
datalayout=enum("DZG","DGZ","GDZ","GZD","ZDG","ZGD");
CodeReg Scattering {
  if (datalayout == "DGZ") {
     omploop="0.0.0.0";
  } elif (datalayout == "GDZ") {
     looporder=[1,2,0,3,4];
     omploop="0.0.0.0";
  } elif (datalayout == "GZD") {
     looporder=[1,2,3,4,0];
     omploop="0.0.0";
  } elif (datalayout == "ZGD") {
     looporder=[3,4,1,2,0];
     omploop="0";
  } elif (datalayout == "ZDG") {
     looporder=[3,4,0,1,2];
     omploop="0";
  } elif (datalayout == "DZG") {
     looporder=[0,3,4,1,2];
     omploop="0.0";
  }
  sourcepath="scatter_"+datalayout+".txt";
  BuiltIn.Altdesc(stmt="0.0.0.0.0.3", source=sourcepath);
  RoseLocus.Interchange(order=looporder);
  RoseLocus.LICM();
  RoseLocus.ScalarRepl();
  Pragma.OMPFor(loop=omploop);
}
```

Kripke - Scattering Kernel

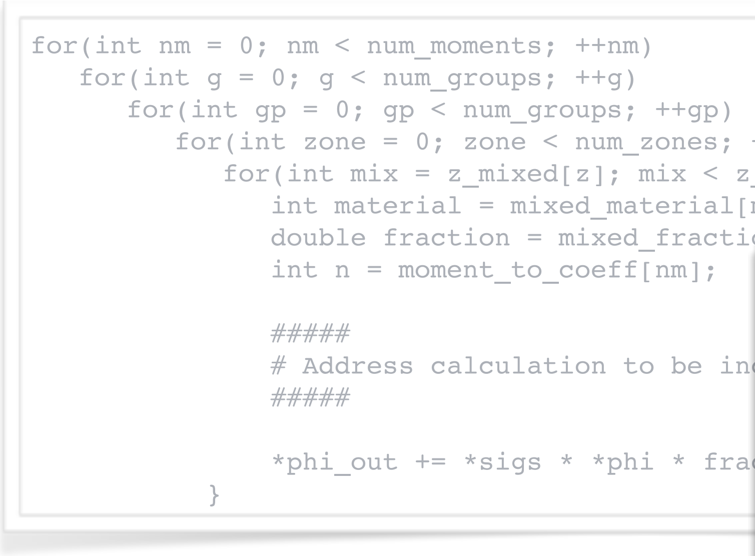


```
for(int nm = 0; nm < num_moments; ++nm)
   for(int g = 0; g < num_groups; ++g)
      for(int gp = 0; gp < num_groups; ++gp)
         for(int zone = 0; zone < num_zones; ++zone)
            for(int mix = z_mixed[z]; mix < z_mixed[z]+num_mixed[z]; ++mix) {
               int material = mixed_material[mix];
               double fraction = mixed_fraction[mix];
               int n = moment_to_coeff[nm];

               #####
               # Address calculation to be in
               #####

               *phi_out += *sigs * *phi * frac
            }
```

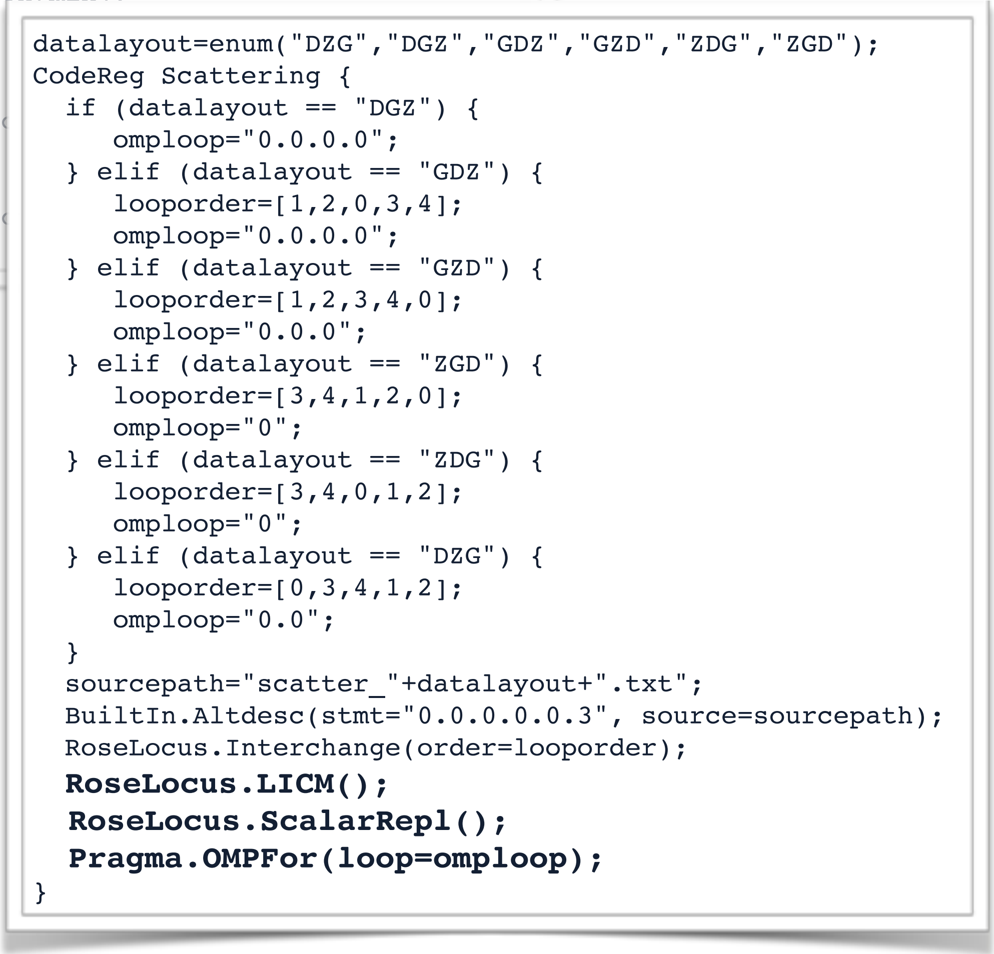


```
datalayout=enum("DZG","DGZ","GDZ","GZD","ZDG","ZGD");
CodeReg Scattering {
  if (datalayout == "DGZ") {
     omploop="0.0.0.0";
  } elif (datalayout == "GDZ") {
     looporder=[1,2,0,3,4];
     omploop="0.0.0.0";
  } elif (datalayout == "GZD") {
     looporder=[1,2,3,4,0];
     omploop="0.0.0";
  } elif (datalayout == "ZGD") {
     looporder=[3,4,1,2,0];
     omploop="0";
  } elif (datalayout == "ZDG") {
     looporder=[3,4,0,1,2];
     omploop="0";
  } elif (datalayout == "DZG") {
     looporder=[0,3,4,1,2];
     omploop="0.0";
  }
  sourcepath="scatter_"+datalayout+".txt";
  BuiltIn.Altdesc(stmt="0.0.0.0.0.3", source=sourcepath);
  RoseLocus.Interchange(order=looporder);
  RoseLocus.LICM();
  RoseLocus.ScalarRepl();
  Pragma.OMPFor(loop=omploop);
}
```

Kripke - Scattering Kernel



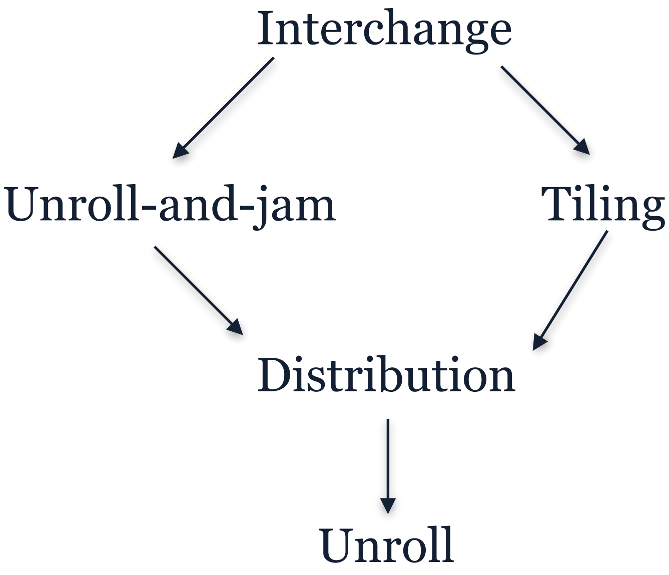
```
for(int nm = 0; nm < num_moments; ++nm)
   for(int g = 0; g < num_groups; ++g)
      for(int gp = 0; gp < num_groups; ++gp)
         for(int zone = 0; zone < num_zones; ++zone)
            for(int mix = z_mixed[z]; mix < z_mixed[z]+num_mixed[z]; ++mix) {
               int material = mixed_material[mix];
               double fraction = mixed_fraction[mix];
               int n = moment_to_coeff[nm];

               #####
               # Address calculation to be inc
               #####

               *phi_out += *sigs * *phi * frac
           }
```

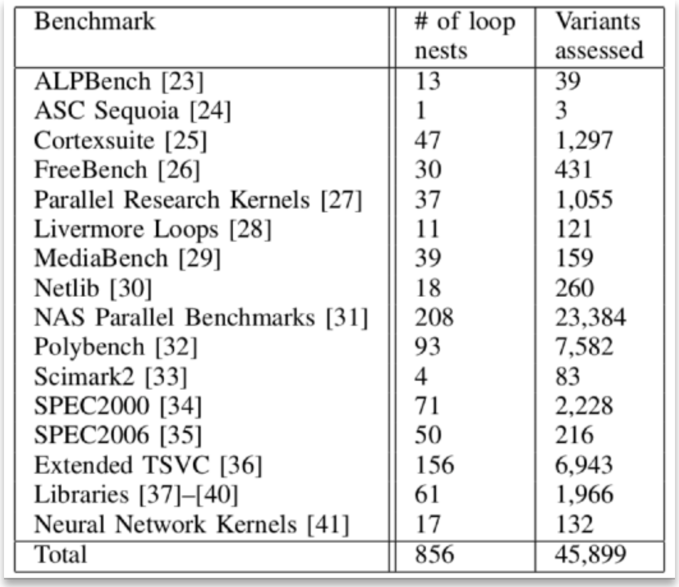


```
datalayout=enum("DZG","DGZ","GDZ","GZD","ZDG","ZGD");
CodeReg Scattering {
  if (datalayout == "DGZ") {
     omploop="0.0.0.0";
  } elif (datalayout == "GDZ") {
     looporder=[1,2,0,3,4];
     omploop="0.0.0.0";
  } elif (datalayout == "GZD") {
     looporder=[1,2,3,4,0];
     omploop="0.0.0";
  } elif (datalayout == "ZGD") {
     looporder=[3,4,1,2,0];
     omploop="0";
  } elif (datalayout == "ZDG") {
     looporder=[3,4,0,1,2];
     omploop="0";
  } elif (datalayout == "DZG") {
     looporder=[0,3,4,1,2];
     omploop="0.0";
  }
  sourcepath="scatter_"+datalayout+".txt";
  BuiltIn.Altdesc(stmt="0.0.0.0.0.3", source=sourcepath);
  RoseLocus.Interchange(order=looporder);
  RoseLocus.LICM();
  RoseLocus.ScalarRepl();
  Pragma.OMPFor(loop=omploop);
}
```

Optimization of Arbitrary Loop Nests

- Generic Locus program to optimize source codes unknown beforehand
- Goal: reproduce Gong Zhangxiaowen et al.[1] work using Locus
- Selected 856 loops from 16 benchmarks
- Transformed loops with all subsets of two sequences:

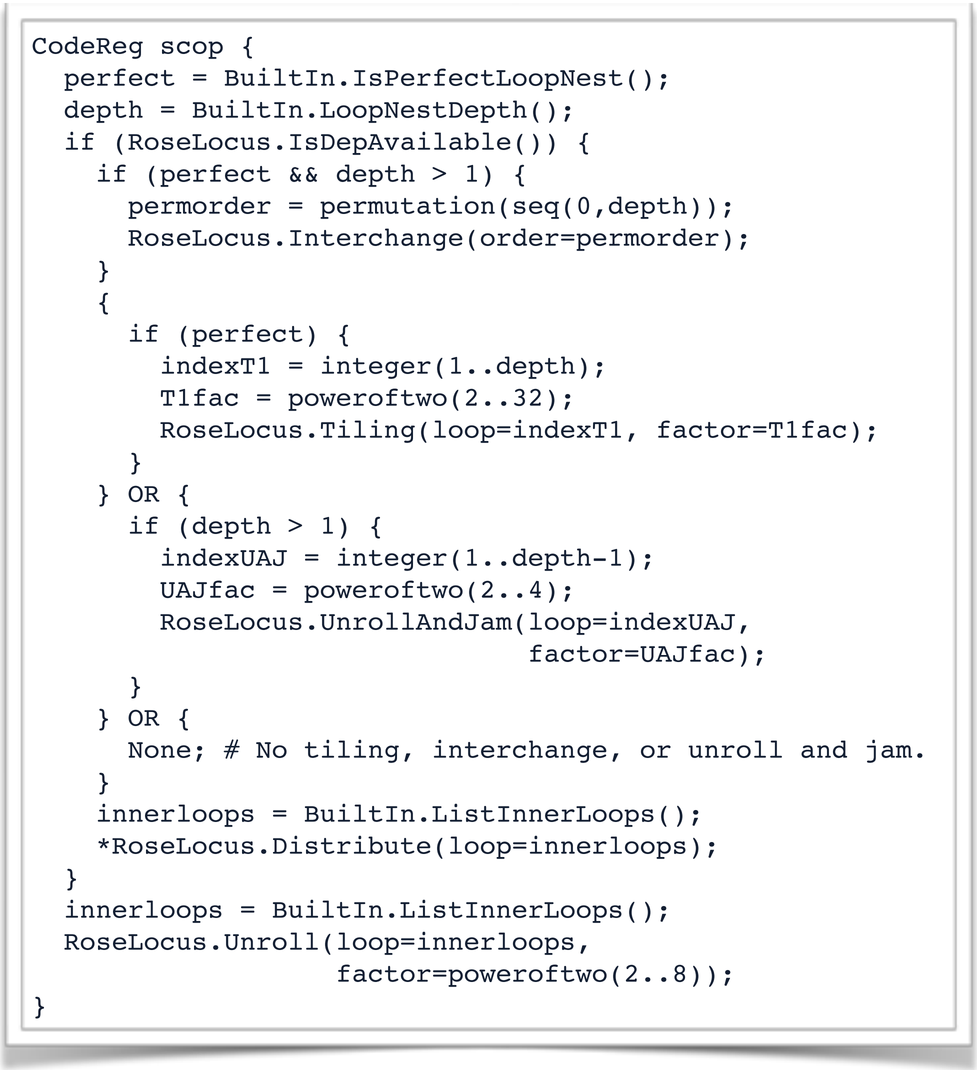




Benchmark	# of loop nests	Variants assessed
ALPBench [23]	13	39
ASC Sequoia [24]	1	3
Cortexsuite [25]	47	1,297
FreeBench [26]	30	431
Parallel Research Kernels [27]	37	1,055
Livermore Loops [28]	11	121
MediaBench [29]	39	159
Netlib [30]	18	260
NAS Parallel Benchmarks [31]	208	23,384
Polybench [32]	93	7,582
Scimark2 [33]	4	83
SPEC2000 [34]	71	2,228
SPEC2006 [35]	50	216
Extended TSVC [36]	156	6,943
Libraries [37]–[40]	61	1,966
Neural Network Kernels [41]	17	132
Total	856	45,899

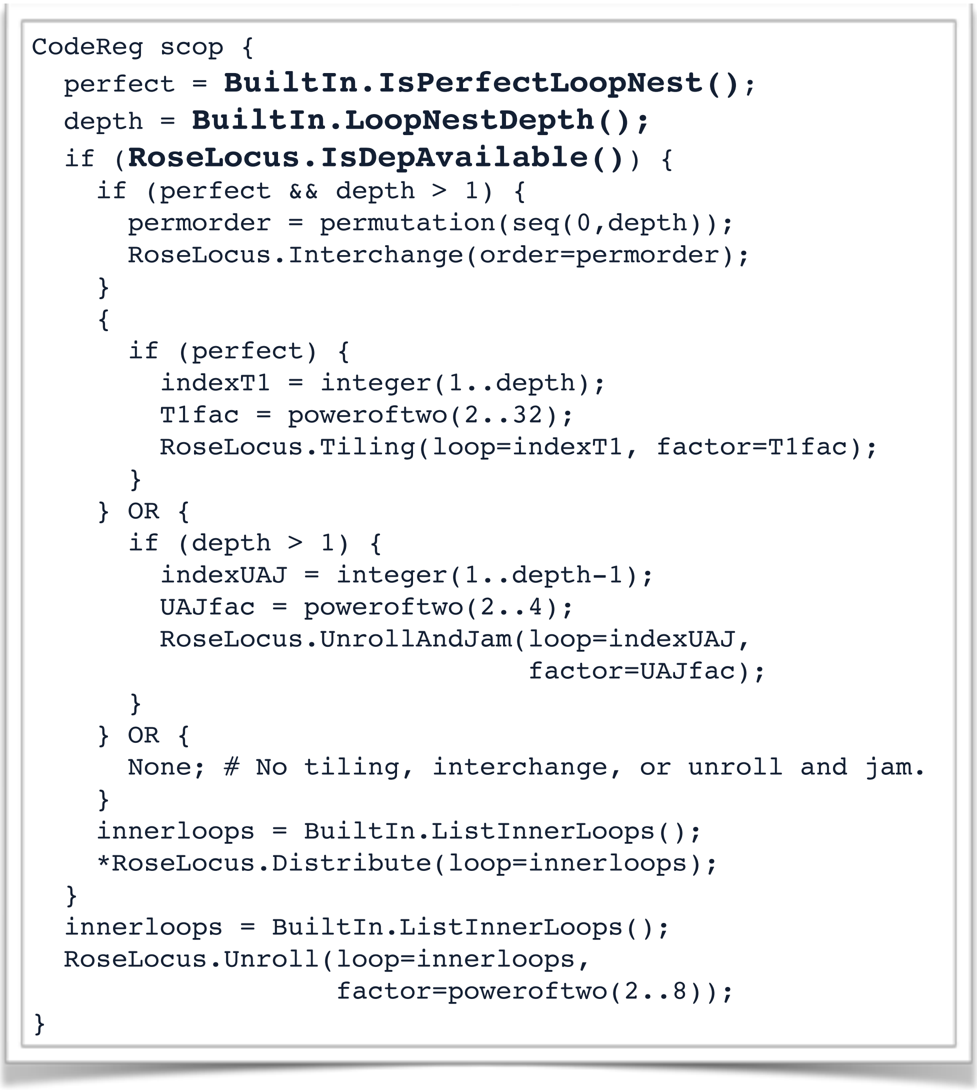
[1] Gong Zhangxiaowen et al. “An empirical study of the effect of source-level loop transformations on compiler stability”.

Optimization of Arbitrary Loop Nests



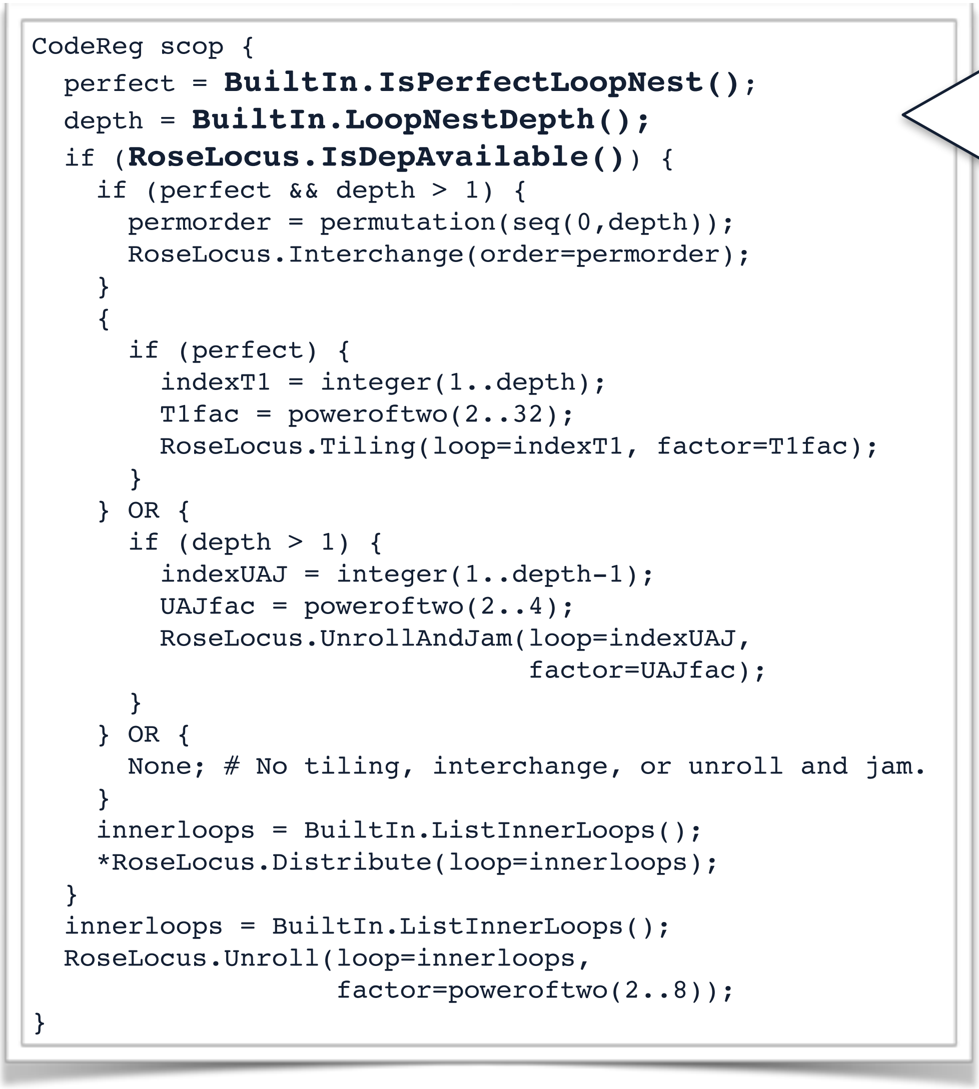
```
CodeReg scop {
  perfect = BuiltIn.IsPerfectLoopNest();
  depth = BuiltIn.LoopNestDepth();
  if (RoseLocus.IsDepAvailable()) {
    if (perfect && depth > 1) {
      permorder = permutation(seq(0,depth));
      RoseLocus.Interchange(order=permorder);
    }
    {
      if (perfect) {
        indexT1 = integer(1..depth);
        T1fac = poweroftwo(2..32);
        RoseLocus.Tiling(loop=indexT1, factor=T1fac);
      }
    } OR {
      if (depth > 1) {
        indexUAJ = integer(1..depth-1);
        UAJfac = poweroftwo(2..4);
        RoseLocus.UnrollAndJam(loop=indexUAJ,
                               factor=UAJfac);
      }
    } OR {
      None; # No tiling, interchange, or unroll and jam.
    }
    innerloops = BuiltIn.ListInnerLoops();
    *RoseLocus.Distribute(loop=innerloops);
  }
  innerloops = BuiltIn.ListInnerLoops();
  RoseLocus.Unroll(loop=innerloops,
                   factor=poweroftwo(2..8));
}
```

Optimization of Arbitrary Loop Nests



```
CodeReg scop {
  perfect = BuiltIn.IsPerfectLoopNest();
  depth = BuiltIn.LoopNestDepth();
  if (RoseLocus.IsDepAvailable()) {
    if (perfect && depth > 1) {
      permorder = permutation(seq(0,depth));
      RoseLocus.Interchange(order=permorder);
    }
    {
      if (perfect) {
        indexT1 = integer(1..depth);
        T1fac = poweroftwo(2..32);
        RoseLocus.Tiling(loop=indexT1, factor=T1fac);
      }
    } OR {
      if (depth > 1) {
        indexUAJ = integer(1..depth-1);
        UAJfac = poweroftwo(2..4);
        RoseLocus.UnrollAndJam(loop=indexUAJ,
                               factor=UAJfac);
      }
    } OR {
      None; # No tiling, interchange, or unroll and jam.
    }
    innerloops = BuiltIn.ListInnerLoops();
    *RoseLocus.Distribute(loop=innerloops);
  }
  innerloops = BuiltIn.ListInnerLoops();
  RoseLocus.Unroll(loop=innerloops,
                   factor=poweroftwo(2..8));
}
```

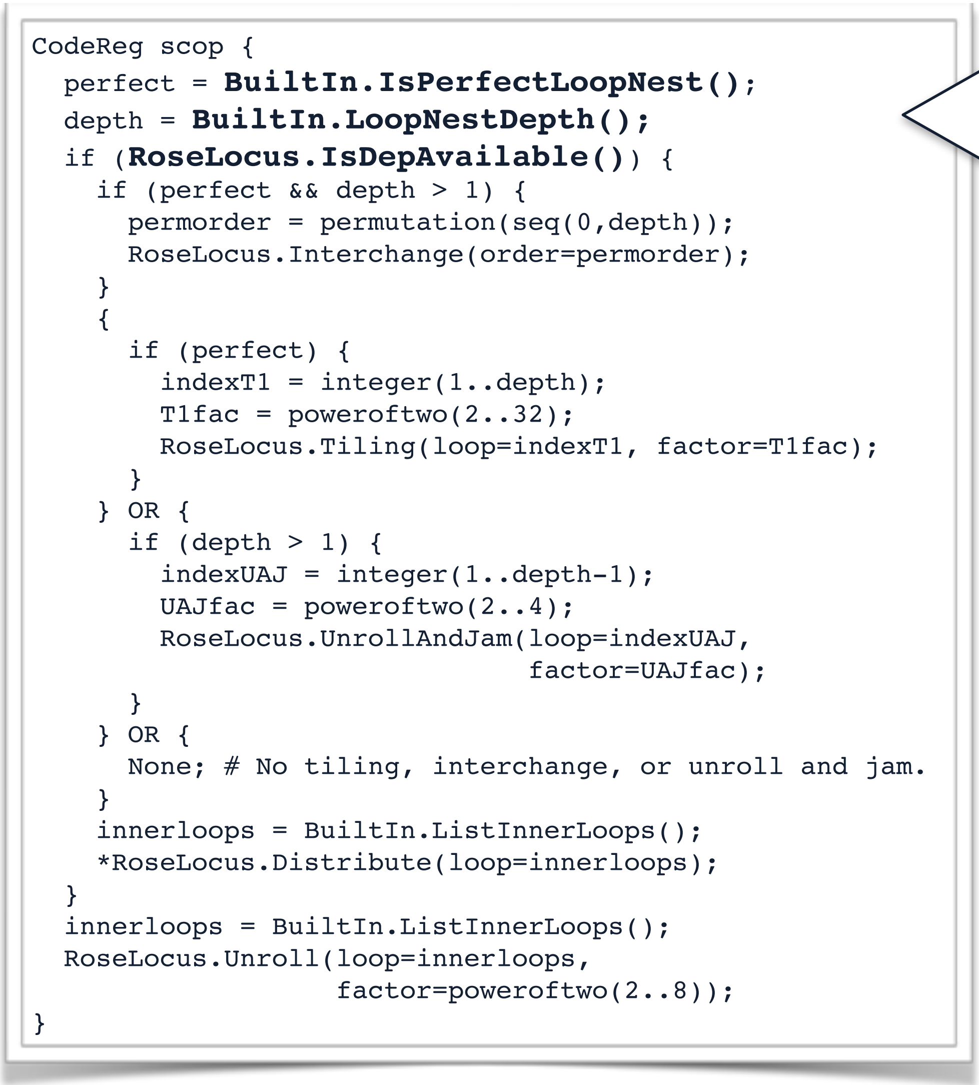
Optimization of Arbitrary Loop Nests



```
CodeReg scop {
  perfect = BuiltIn.IsPerfectLoopNest();
  depth = BuiltIn.LoopNestDepth();
  if (RoseLocus.IsDepAvailable()) {
    if (perfect && depth > 1) {
      permorder = permutation(seq(0,depth));
      RoseLocus.Interchange(order=permorder);
    }
    {
      if (perfect) {
        indexT1 = integer(1..depth);
        T1fac = poweroftwo(2..32);
        RoseLocus.Tiling(loop=indexT1, factor=T1fac);
      }
    } OR {
      if (depth > 1) {
        indexUAJ = integer(1..depth-1);
        UAJfac = poweroftwo(2..4);
        RoseLocus.UnrollAndJam(loop=indexUAJ,
                               factor=UAJfac);
      }
    } OR {
      None; # No tiling, interchange, or unroll and jam.
    }
    innerloops = BuiltIn.ListInnerLoops();
    *RoseLocus.Distribute(loop=innerloops);
  }
  innerloops = BuiltIn.ListInnerLoops();
  RoseLocus.Unroll(loop=innerloops,
                   factor=poweroftwo(2..8));
}
```

Information about the code:

Optimization of Arbitrary Loop Nests

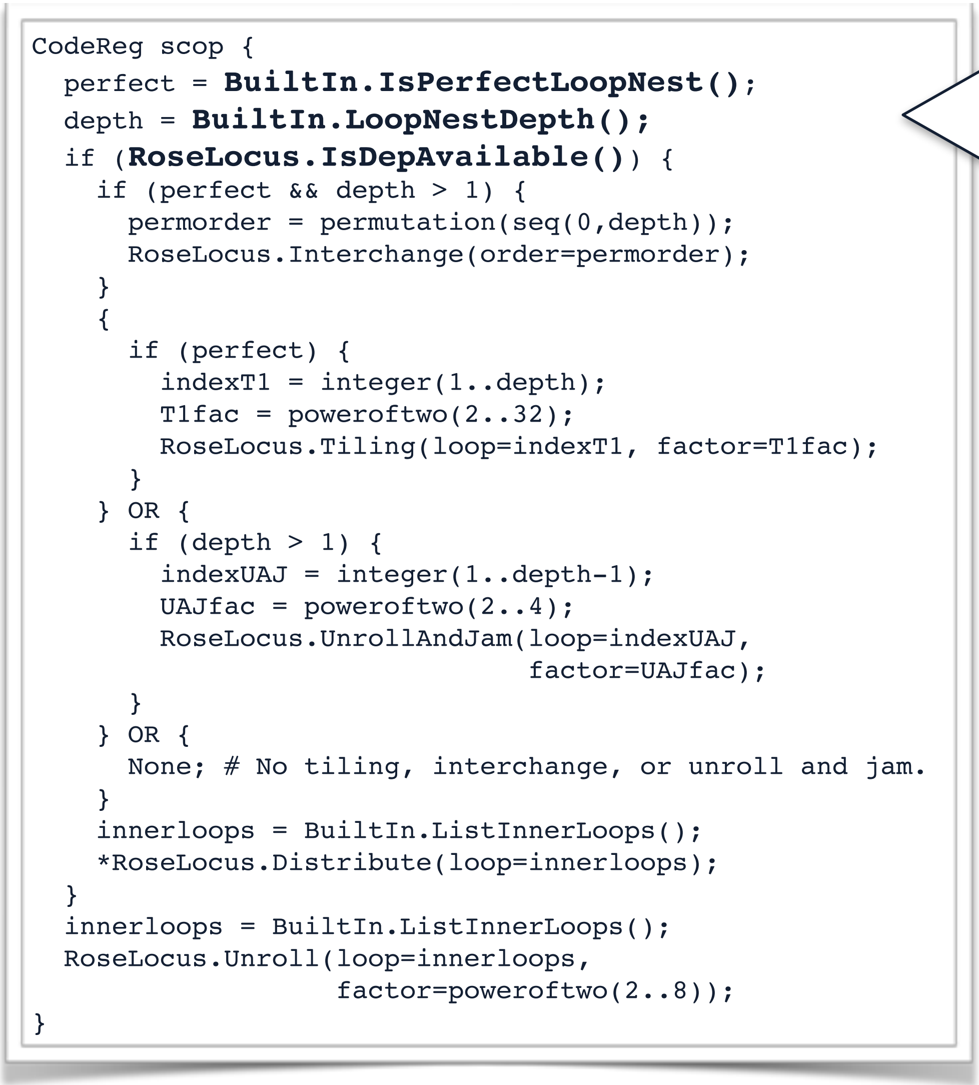


```
CodeReg scop {
  perfect = BuiltIn.IsPerfectLoopNest();
  depth = BuiltIn.LoopNestDepth();
  if (RoseLocus.IsDepAvailable()) {
    if (perfect && depth > 1) {
      permorder = permutation(seq(0,depth));
      RoseLocus.Interchange(order=permorder);
    }
    {
      if (perfect) {
        indexT1 = integer(1..depth);
        T1fac = poweroftwo(2..32);
        RoseLocus.Tiling(loop=indexT1, factor=T1fac);
      }
    } OR {
      if (depth > 1) {
        indexUAJ = integer(1..depth-1);
        UAJfac = poweroftwo(2..4);
        RoseLocus.UnrollAndJam(loop=indexUAJ,
                               factor=UAJfac);
      }
    } OR {
      None; # No tiling, interchange, or unroll and jam.
    }
    innerloops = BuiltIn.ListInnerLoops();
    *RoseLocus.Distribute(loop=innerloops);
  }
  innerloops = BuiltIn.ListInnerLoops();
  RoseLocus.Unroll(loop=innerloops,
                   factor=poweroftwo(2..8));
}
```

Information about
the code:

- Perfect loop nest?

Optimization of Arbitrary Loop Nests



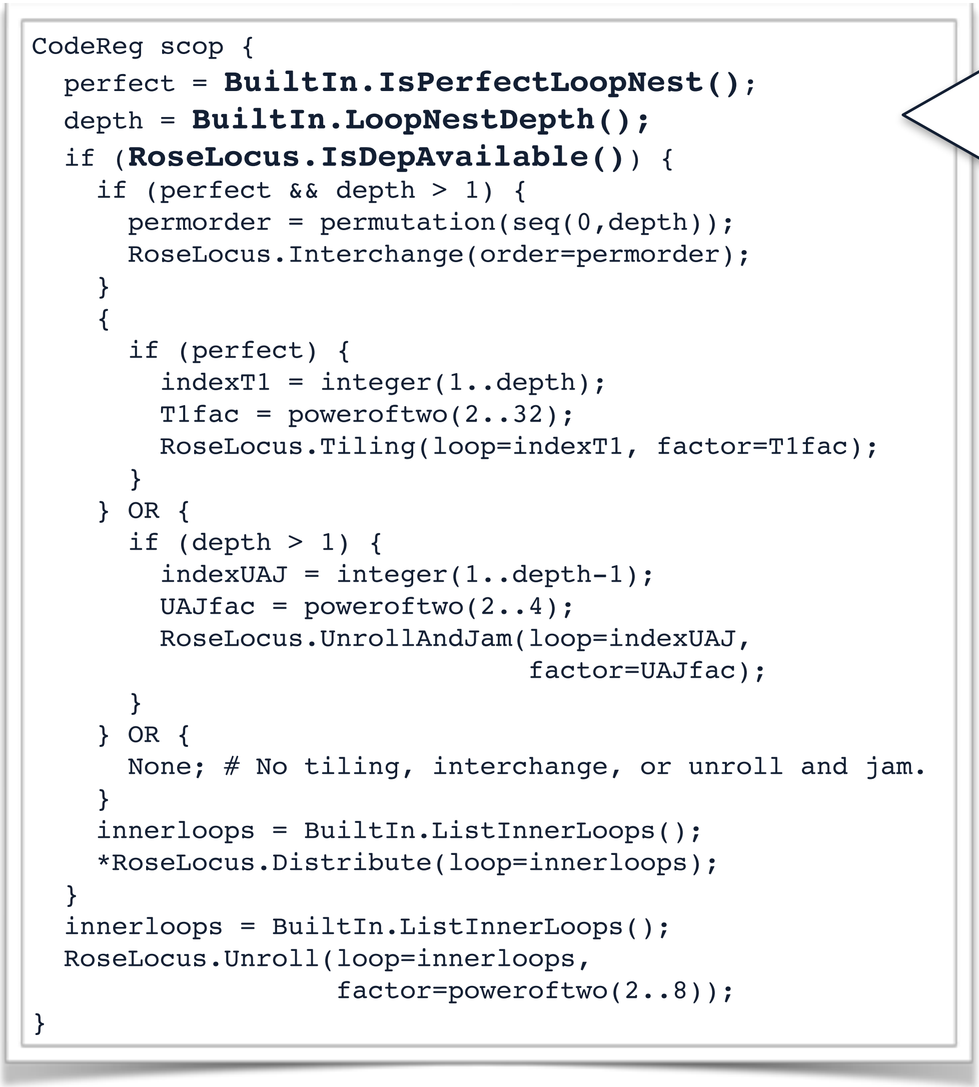
```
CodeReg scop {
  perfect = BuiltIn.IsPerfectLoopNest();
  depth = BuiltIn.LoopNestDepth();
  if (RoseLocus.IsDepAvailable()) {
    if (perfect && depth > 1) {
      permorder = permutation(seq(0,depth));
      RoseLocus.Interchange(order=permorder);
    }
    {
      if (perfect) {
        indexT1 = integer(1..depth);
        T1fac = poweroftwo(2..32);
        RoseLocus.Tiling(loop=indexT1, factor=T1fac);
      }
    } OR {
      if (depth > 1) {
        indexUAJ = integer(1..depth-1);
        UAJfac = poweroftwo(2..4);
        RoseLocus.UnrollAndJam(loop=indexUAJ,
                               factor=UAJfac);
      }
    } OR {
      None; # No tiling, interchange, or unroll and jam.
    }
    innerloops = BuiltIn.ListInnerLoops();
    *RoseLocus.Distribute(loop=innerloops);
  }
  innerloops = BuiltIn.ListInnerLoops();
  RoseLocus.Unroll(loop=innerloops,
                   factor=poweroftwo(2..8));
}
```

Information about the code:

- Perfect loop nest?

- Loop nest depth

Optimization of Arbitrary Loop Nests



```
CodeReg scop {
  perfect = BuiltIn.IsPerfectLoopNest();
  depth = BuiltIn.LoopNestDepth();
  if (RoseLocus.IsDepAvailable()) {
    if (perfect && depth > 1) {
      permorder = permutation(seq(0,depth));
      RoseLocus.Interchange(order=permorder);
    }
    {
      if (perfect) {
        indexT1 = integer(1..depth);
        T1fac = poweroftwo(2..32);
        RoseLocus.Tiling(loop=indexT1, factor=T1fac);
      }
    } OR {
      if (depth > 1) {
        indexUAJ = integer(1..depth-1);
        UAJfac = poweroftwo(2..4);
        RoseLocus.UnrollAndJam(loop=indexUAJ,
                               factor=UAJfac);
      }
    } OR {
      None; # No tiling, interchange, or unroll and jam.
    }
    innerloops = BuiltIn.ListInnerLoops();
    *RoseLocus.Distribute(loop=innerloops);
  }
  innerloops = BuiltIn.ListInnerLoops();
  RoseLocus.Unroll(loop=innerloops,
                   factor=poweroftwo(2..8));
}
```

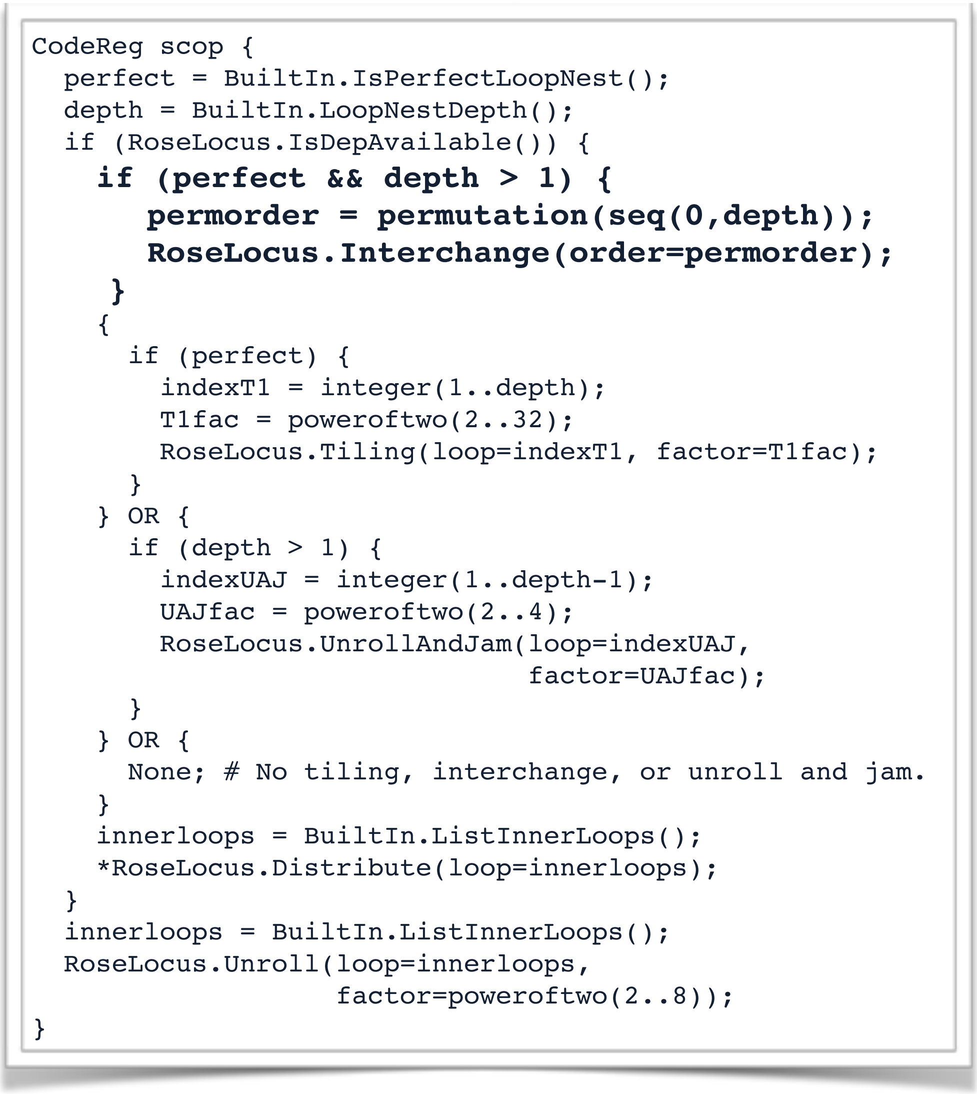
Information about
the code:

- Perfect loop nest?

- Loop nest depth

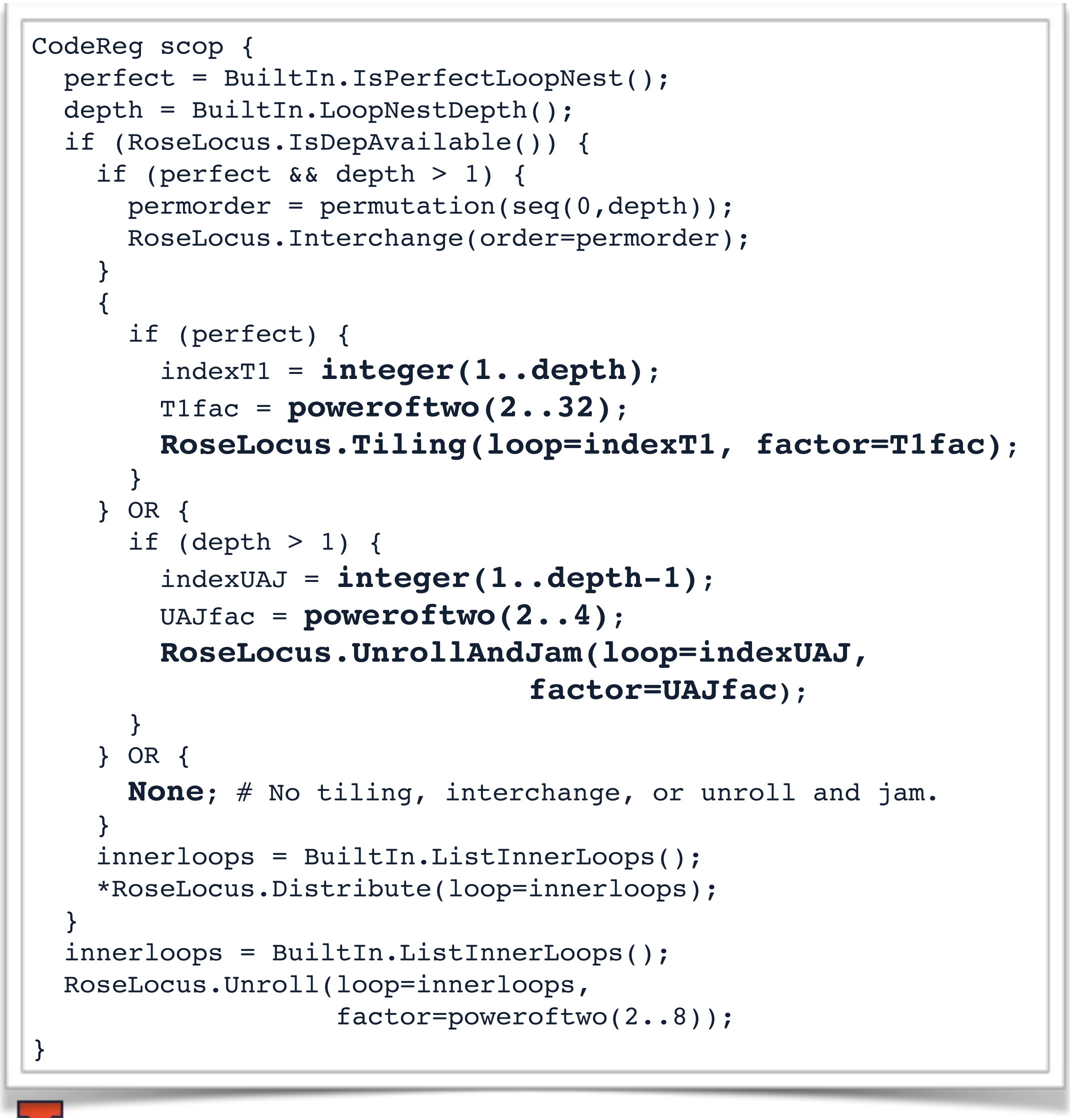
- Dependence test available?

Optimization of Arbitrary Loop Nests



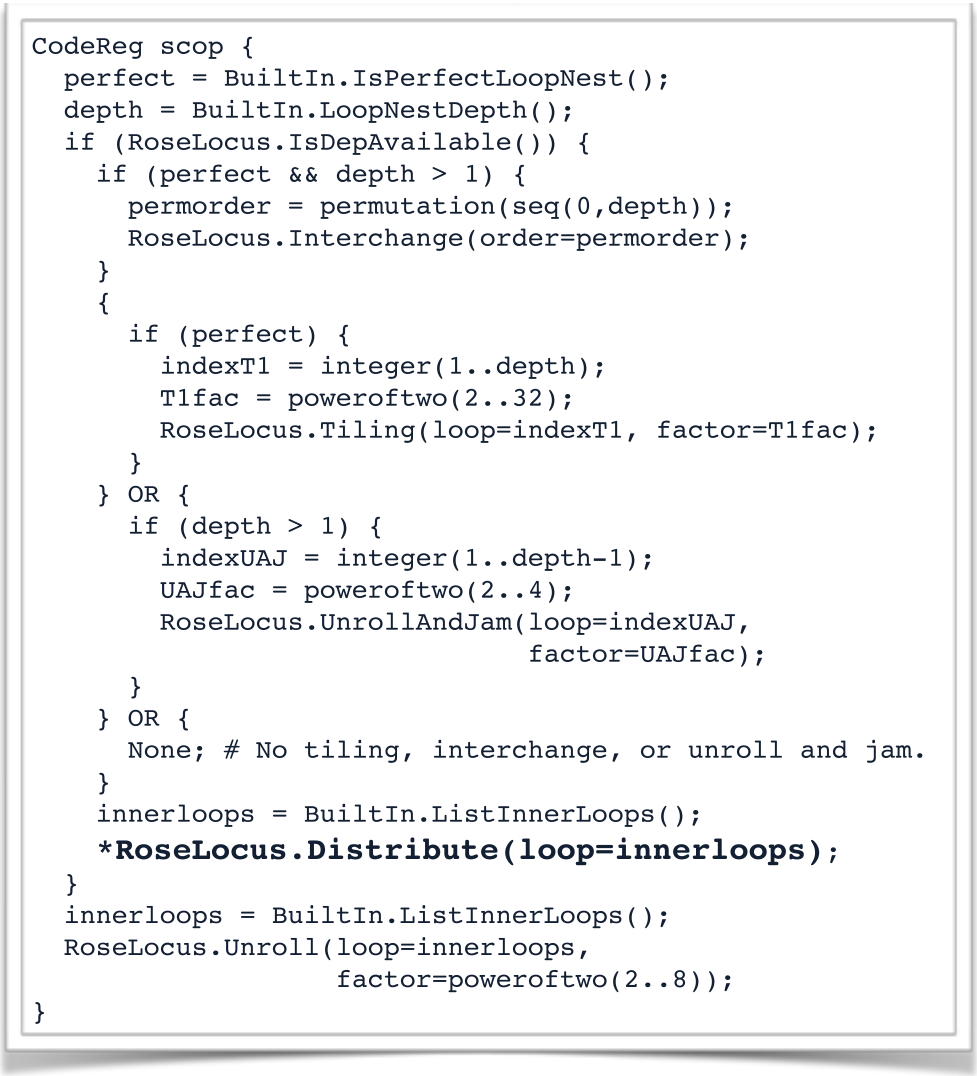
```
CodeReg scop {
  perfect = BuiltIn.IsPerfectLoopNest();
  depth = BuiltIn.LoopNestDepth();
  if (RoseLocus.IsDepAvailable()) {
    if (perfect && depth > 1) {
       permorder = permutation(seq(0,depth));
       RoseLocus.Interchange(order=permorder);
     }
    {
      if (perfect) {
        indexT1 = integer(1..depth);
        T1fac = poweroftwo(2..32);
        RoseLocus.Tiling(loop=indexT1, factor=T1fac);
      }
    } OR {
      if (depth > 1) {
        indexUAJ = integer(1..depth-1);
        UAJfac = poweroftwo(2..4);
        RoseLocus.UnrollAndJam(loop=indexUAJ,
                               factor=UAJfac);
      }
    } OR {
      None; # No tiling, interchange, or unroll and jam.
    }
    innerloops = BuiltIn.ListInnerLoops();
    *RoseLocus.Distribute(loop=innerloops);
  }
  innerloops = BuiltIn.ListInnerLoops();
  RoseLocus.Unroll(loop=innerloops,
                   factor=poweroftwo(2..8));
}
```

Optimization of Arbitrary Loop Nests



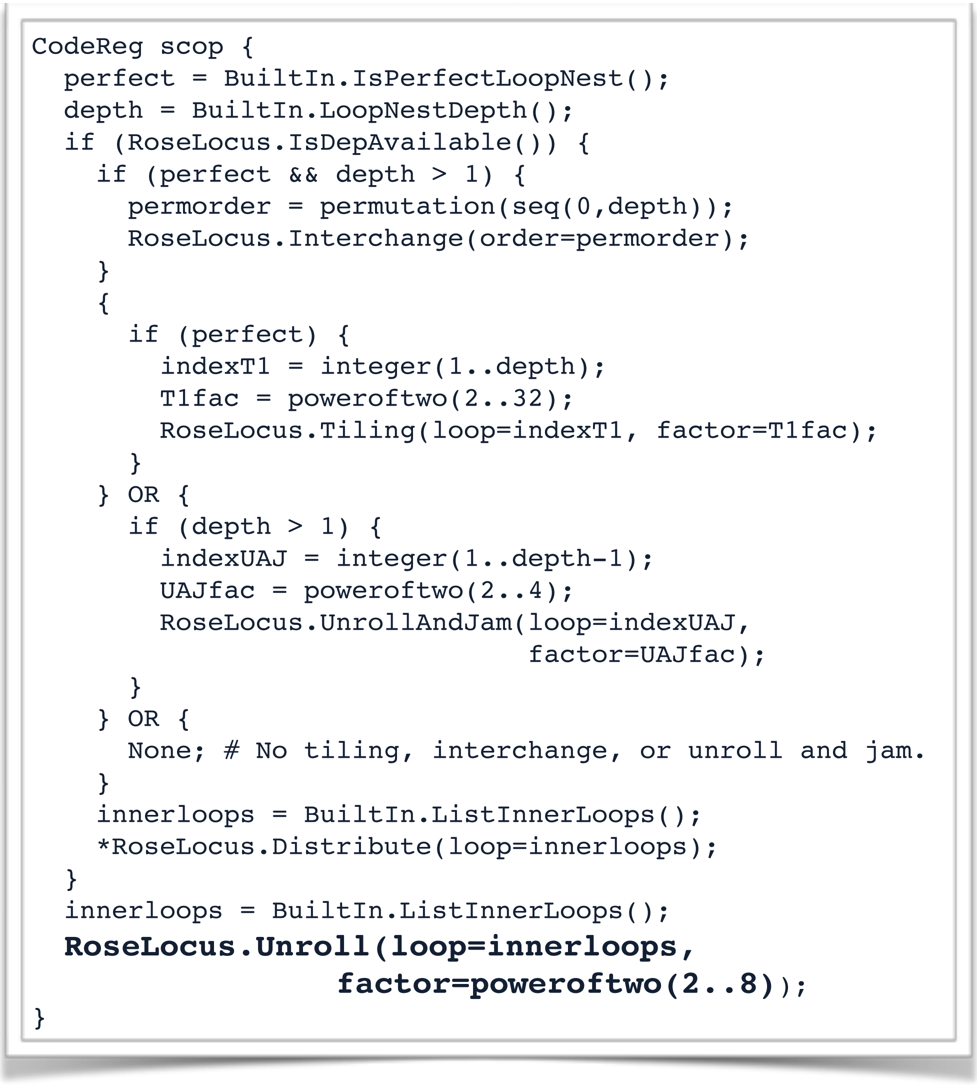
```
CodeReg scop {
  perfect = BuiltIn.IsPerfectLoopNest();
  depth = BuiltIn.LoopNestDepth();
  if (RoseLocus.IsDepAvailable()) {
    if (perfect && depth > 1) {
      permorder = permutation(seq(0,depth));
      RoseLocus.Interchange(order=permorder);
    }
    {
      if (perfect) {
        indexT1 = integer(1..depth);
        T1fac = poweroftwo(2..32);
        RoseLocus.Tiling(loop=indexT1, factor=T1fac);
      }
    } OR {
      if (depth > 1) {
        indexUAJ = integer(1..depth-1);
        UAJfac = poweroftwo(2..4);
        RoseLocus.UnrollAndJam(loop=indexUAJ,
                               factor=UAJfac);
      }
    } OR {
      None; # No tiling, interchange, or unroll and jam.
    }
    innerloops = BuiltIn.ListInnerLoops();
    *RoseLocus.Distribute(loop=innerloops);
  }
  innerloops = BuiltIn.ListInnerLoops();
  RoseLocus.Unroll(loop=innerloops,
                   factor=poweroftwo(2..8));
}
```

Optimization of Arbitrary Loop Nests



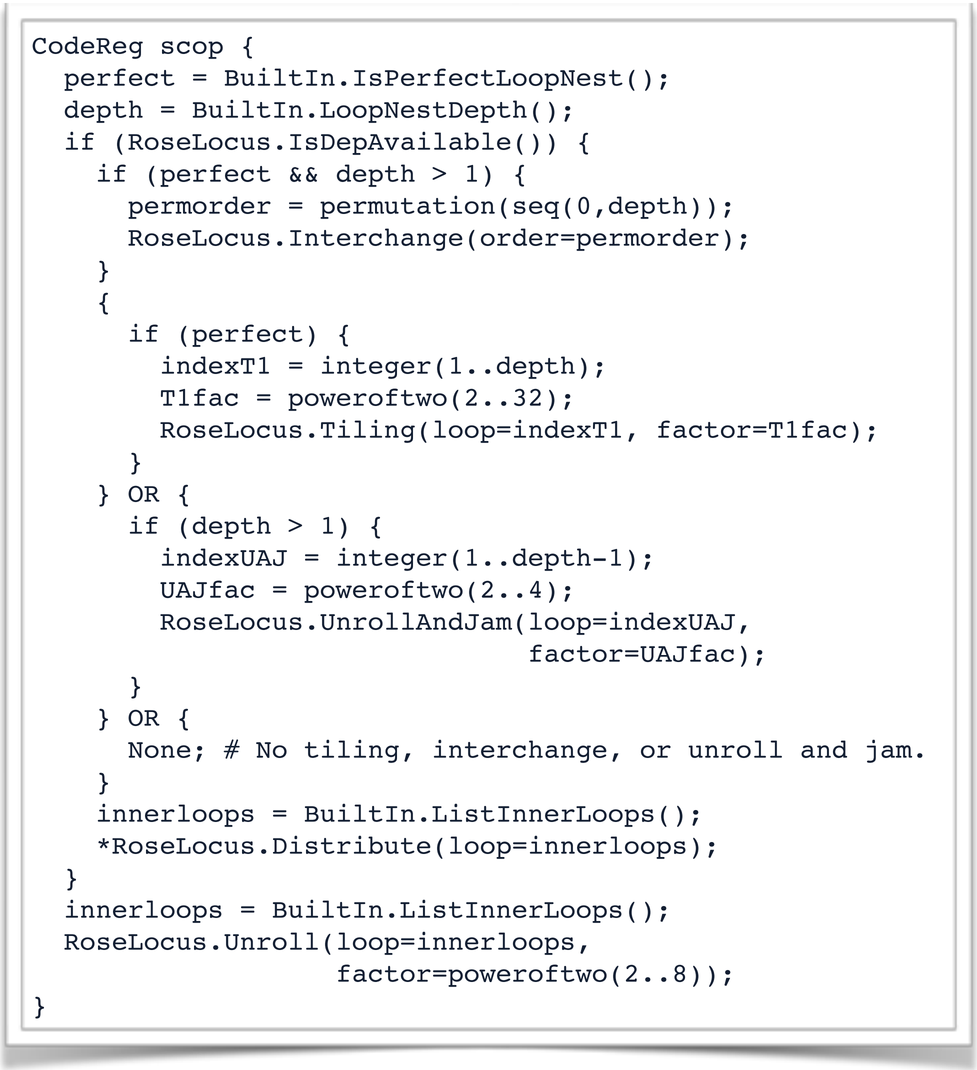
```
CodeReg scop {
  perfect = BuiltIn.IsPerfectLoopNest();
  depth = BuiltIn.LoopNestDepth();
  if (RoseLocus.IsDepAvailable()) {
    if (perfect && depth > 1) {
      permorder = permutation(seq(0,depth));
      RoseLocus.Interchange(order=permorder);
    }
    {
      if (perfect) {
        indexT1 = integer(1..depth);
        T1fac = poweroftwo(2..32);
        RoseLocus.Tiling(loop=indexT1, factor=T1fac);
      }
    } OR {
      if (depth > 1) {
        indexUAJ = integer(1..depth-1);
        UAJfac = poweroftwo(2..4);
        RoseLocus.UnrollAndJam(loop=indexUAJ,
                               factor=UAJfac);
      }
    } OR {
      None; # No tiling, interchange, or unroll and jam.
    }
    innerloops = BuiltIn.ListInnerLoops();
    *RoseLocus.Distribute(loop=innerloops);
  }
  innerloops = BuiltIn.ListInnerLoops();
  RoseLocus.Unroll(loop=innerloops,
                   factor=poweroftwo(2..8));
}
```

Optimization of Arbitrary Loop Nests



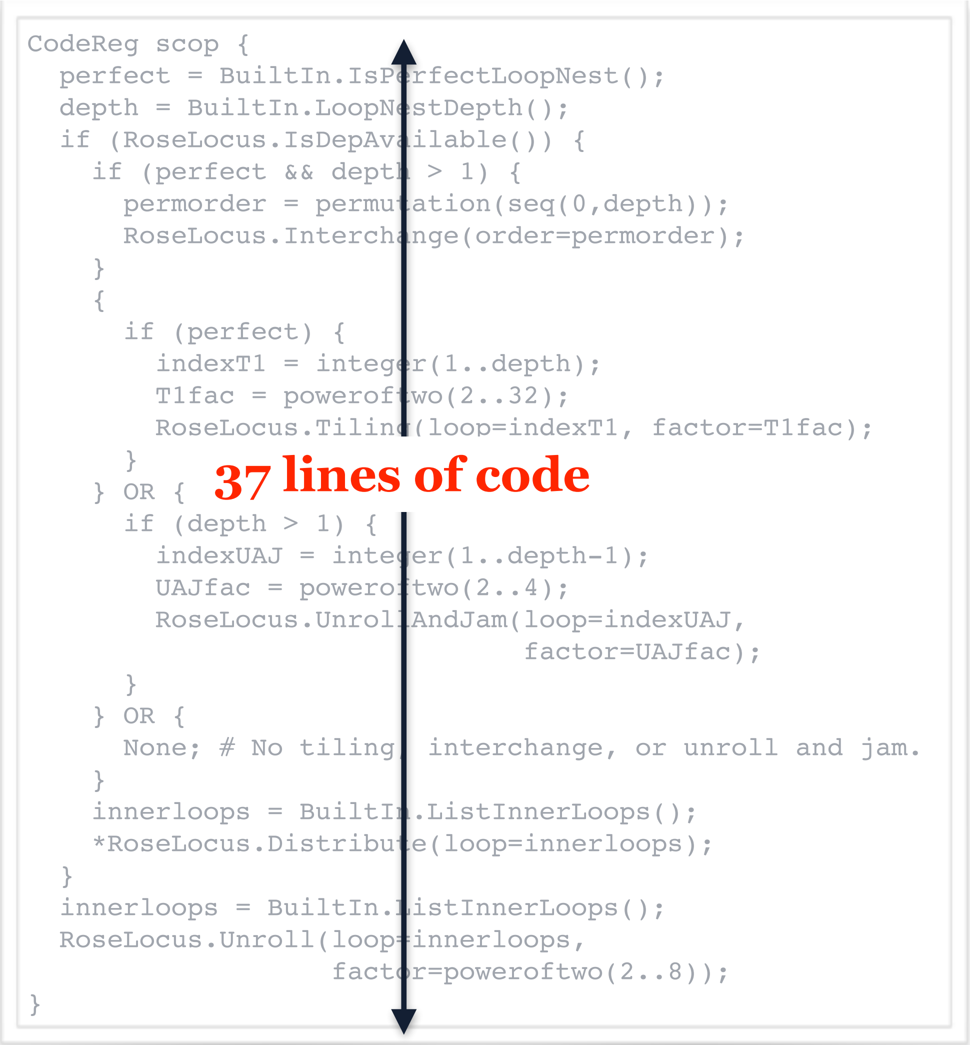
```
CodeReg scop {
  perfect = BuiltIn.IsPerfectLoopNest();
  depth = BuiltIn.LoopNestDepth();
  if (RoseLocus.IsDepAvailable()) {
    if (perfect && depth > 1) {
      permorder = permutation(seq(0,depth));
      RoseLocus.Interchange(order=permorder);
    }
    {
      if (perfect) {
        indexT1 = integer(1..depth);
        T1fac = poweroftwo(2..32);
        RoseLocus.Tiling(loop=indexT1, factor=T1fac);
      }
    } OR {
      if (depth > 1) {
        indexUAJ = integer(1..depth-1);
        UAJfac = poweroftwo(2..4);
        RoseLocus.UnrollAndJam(loop=indexUAJ,
                               factor=UAJfac);
      }
    } OR {
      None; # No tiling, interchange, or unroll and jam.
    }
    innerloops = BuiltIn.ListInnerLoops();
    *RoseLocus.Distribute(loop=innerloops);
  }
  innerloops = BuiltIn.ListInnerLoops();
  RoseLocus.Unroll(loop=innerloops,
                   factor=poweroftwo(2..8));
}
```

Optimization of Arbitrary Loop Nests



```
CodeReg scop {
  perfect = BuiltIn.IsPerfectLoopNest();
  depth = BuiltIn.LoopNestDepth();
  if (RoseLocus.IsDepAvailable()) {
    if (perfect && depth > 1) {
      permorder = permutation(seq(0,depth));
      RoseLocus.Interchange(order=permorder);
    }
    {
      if (perfect) {
        indexT1 = integer(1..depth);
        T1fac = poweroftwo(2..32);
        RoseLocus.Tiling(loop=indexT1, factor=T1fac);
      }
    } OR {
      if (depth > 1) {
        indexUAJ = integer(1..depth-1);
        UAJfac = poweroftwo(2..4);
        RoseLocus.UnrollAndJam(loop=indexUAJ,
                               factor=UAJfac);
      }
    } OR {
      None; # No tiling, interchange, or unroll and jam.
    }
    innerloops = BuiltIn.ListInnerLoops();
    *RoseLocus.Distribute(loop=innerloops);
  }
  innerloops = BuiltIn.ListInnerLoops();
  RoseLocus.Unroll(loop=innerloops,
                   factor=poweroftwo(2..8));
}
```

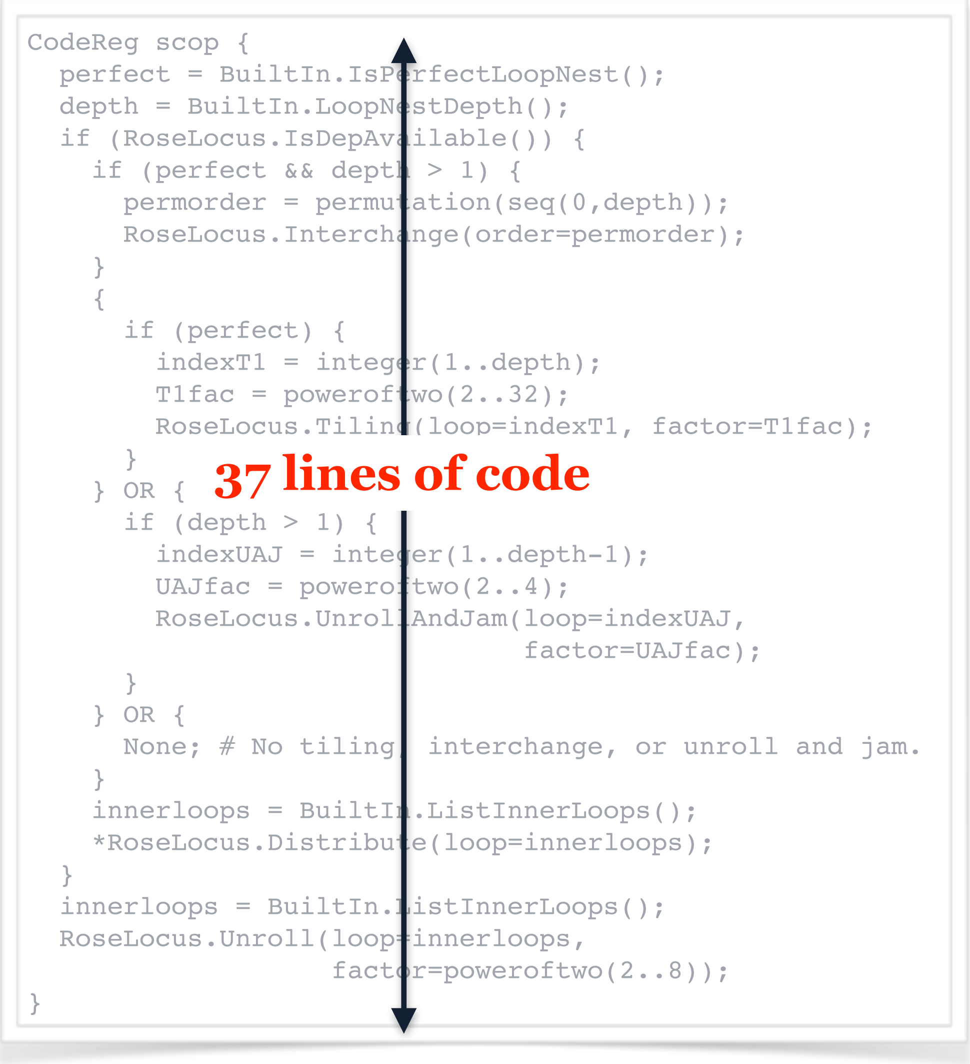
Optimization of Arbitrary Loop Nests



```
CodeReg scop {
 perfect = BuiltIn.IsPerfectLoopNest();
 depth = BuiltIn.LoopNestDepth();
 if (RoseLocus.IsDepAvailable()) {
  if (perfect && depth > 1) {
   permorder = permutation(seq(0,depth));
   RoseLocus.Interchange(order=permorder);
  }
  {
   if (perfect) {
    indexT1 = integer(1..depth);
    T1fac = poweroftwo(2..32);
    RoseLocus.Tiling(loop=indexT1, factor=T1fac);
   }
  } OR {
   if (depth > 1) {
    indexUAJ = integer(1..depth-1);
    UAJfac = poweroftwo(2..4);
    RoseLocus.UnrollAndJam(loop=indexUAJ,
               factor=UAJfac);
   }
  } OR {
   None; # No tiling, interchange, or unroll and jam.
  }
  innerloops = BuiltIn.ListInnerLoops();
  *RoseLocus.Distribute(loop=innerloops);
 }
 innerloops = BuiltIn.ListInnerLoops();
 RoseLocus.Unroll(loop=innerloops,
         factor=poweroftwo(2..8));
}
```

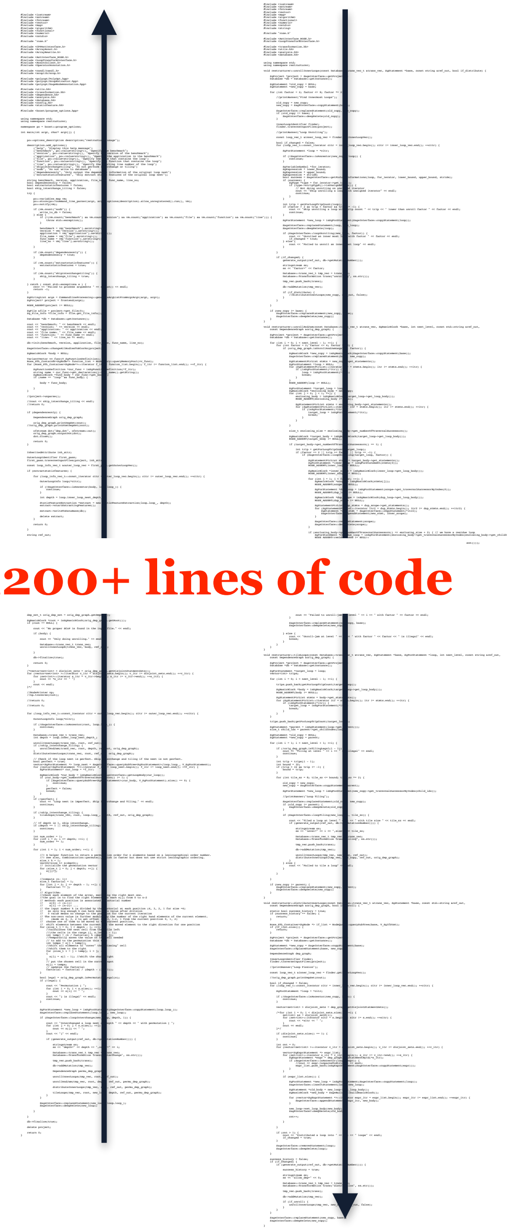
37 lines of code

Optimization of Arbitrary Loop Nests



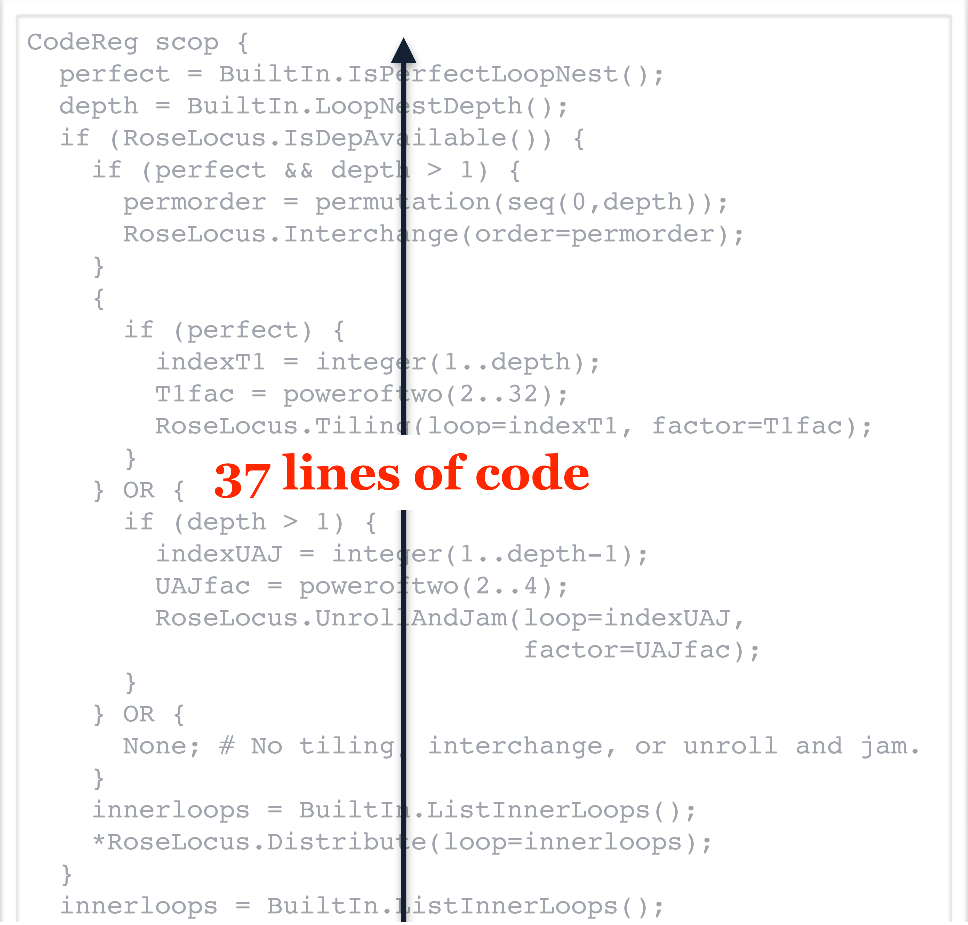
```
CodeReg scop {
  perfect = BuiltIn.IsPerfectLoopNest();
  depth = BuiltIn.LoopNestDepth();
  if (RoseLocus.IsDepAvailable()) {
    if (perfect && depth > 1) {
      permorder = permutation(seq(0,depth));
      RoseLocus.Interchange(order=permorder);
    }
    {
      if (perfect) {
        indexT1 = integer(1..depth);
        T1fac = poweroftwo(2..32);
        RoseLocus.Tiling(loop=indexT1, factor=T1fac);
      }
    } OR {
      if (depth > 1) {
        indexUAJ = integer(1..depth-1);
        UAJfac = poweroftwo(2..4);
        RoseLocus.UnrollAndJam(loop=indexUAJ,
                               factor=UAJfac);
      }
    } OR {
      None; # No tiling, interchange, or unroll and jam.
    }
    innerloops = BuiltIn.ListInnerLoops();
    *RoseLocus.Distribute(loop=innerloops);
  }
  innerloops = BuiltIn.ListInnerLoops();
  RoseLocus.Unroll(loop=innerloops,
                   factor=poweroftwo(2..8));
}
```

37 lines of code



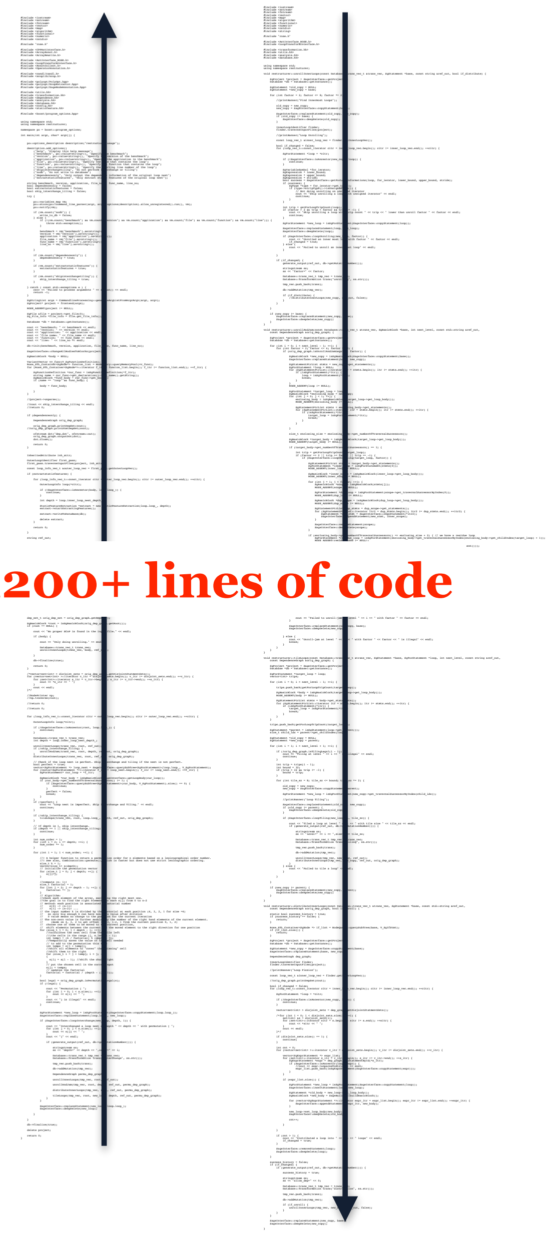
1200+ lines of code

Optimization of Arbitrary Loop Nests



```
CodeReg scop {
  perfect = BuiltIn.IsPerfectLoopNest();
  depth = BuiltIn.LoopNestDepth();
  if (RoseLocus.IsDepAvailable()) {
    if (perfect && depth > 1) {
      permorder = permutation(seq(0,depth));
      RoseLocus.Interchange(order=permorder);
    }
    {
      if (perfect) {
        indexT1 = integer(1..depth);
        T1fac = poweroftwo(2..32);
        RoseLocus.Tiling(loop=indexT1, factor=T1fac);
      }
    } OR {
      if (depth > 1) {
        indexUAJ = integer(1..depth-1);
        UAJfac = poweroftwo(2..4);
        RoseLocus.UnrollAndJam(loop=indexUAJ,
                               factor=UAJfac);
      }
    } OR {
      None; # No tiling, interchange, or unroll and jam.
    }
    innerloops = BuiltIn.ListInnerLoops();
    *RoseLocus.Distribute(loop=innerloops);
  }
  innerloops = BuiltIn.ListInnerLoops();
```

37 lines of code



1200+ lines of code

- Reproduced Gong Zhangxiaowen et al. results
- Much more concise and flexible

Conclusions

- Locus is able to represent *complex* optimization spaces for different code regions
- Easy to use fine-grain *optimizations* in fine-grain *regions of code* to improve performance
- *Share* resulting optimization programs to amortize the search time
- Keep the baseline version *cleaner* and *simpler* for the long term
- Future work:
  - Use multiple search modules concurrently to speed up the search process
  - Help users at designing optimization sequences

Acknowledgments

Project is part of the Center for Exascale Simulation of Plasma-Coupled Combustion (XPACC)
xpacc.illinois.edu

This material is based in part upon work supported by the Department of Energy, National Nuclear Security Administration, under Award Number DE-NA0002374 and by the National Science Foundation under Award 1533912.

We also gratefully acknowledge Gong Zhangxiaowen and Justin Szaday for their valuable help in setting up the experiments presented for optimizing arbitrary loop nests.

Locus: A System and a Language for Program Optimization

Thiago Teixeira[*], Corinne Ancourt[+], David Padua[*], William Gropp[*]

tteixei2@illinois.edu

[*]Department of Computer Science, University of Illinois at Urbana-Champaign, USA

[+]MINES ParisTech, PSL University, France

Thank you!

ILLINOIS

CGO - Washington, DC - Feb 2019