

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

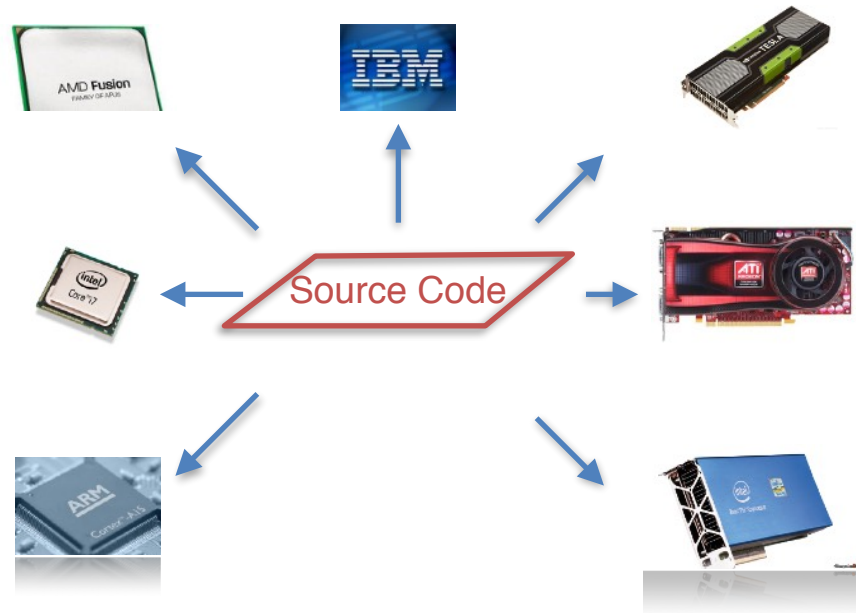
- Very complex machines
- Gap between performance of hand-tuned and compiler-generated code has grown substantially
- Platform-specific optimizations are required

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

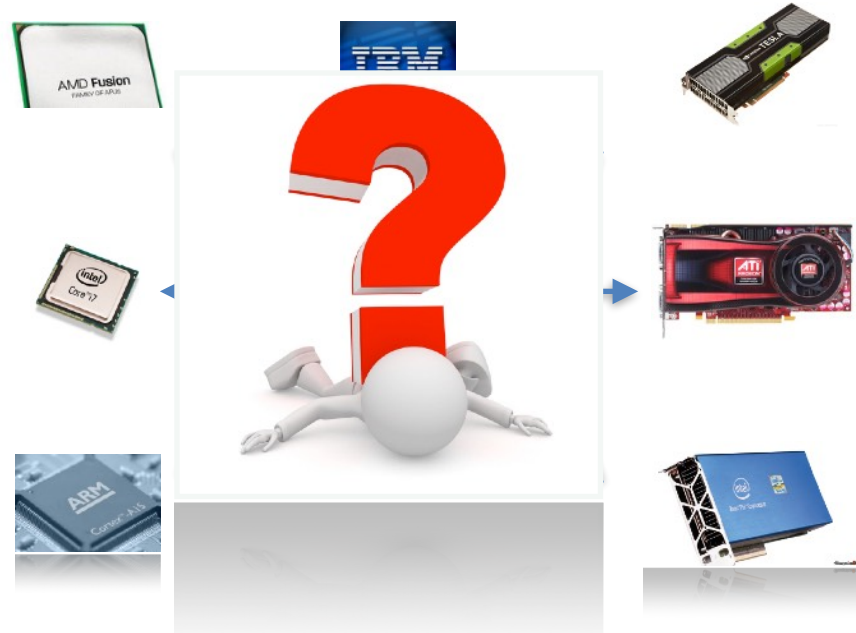
Introduction

- 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

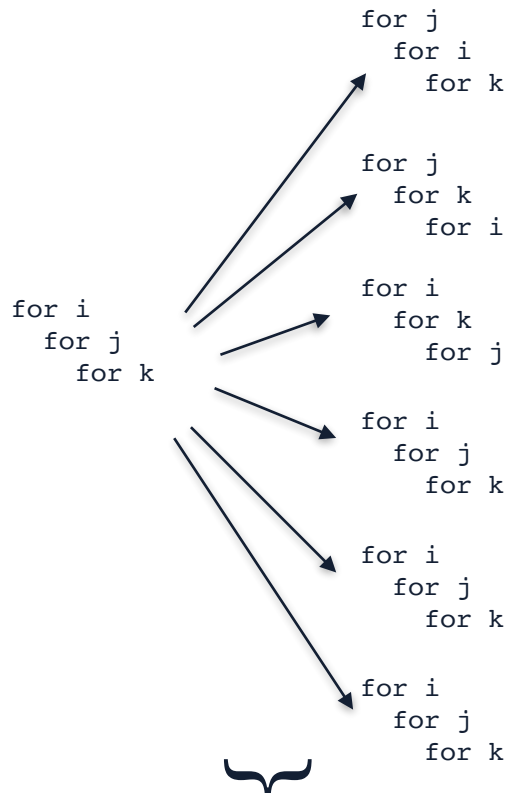
6 variants

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

Optimization Space

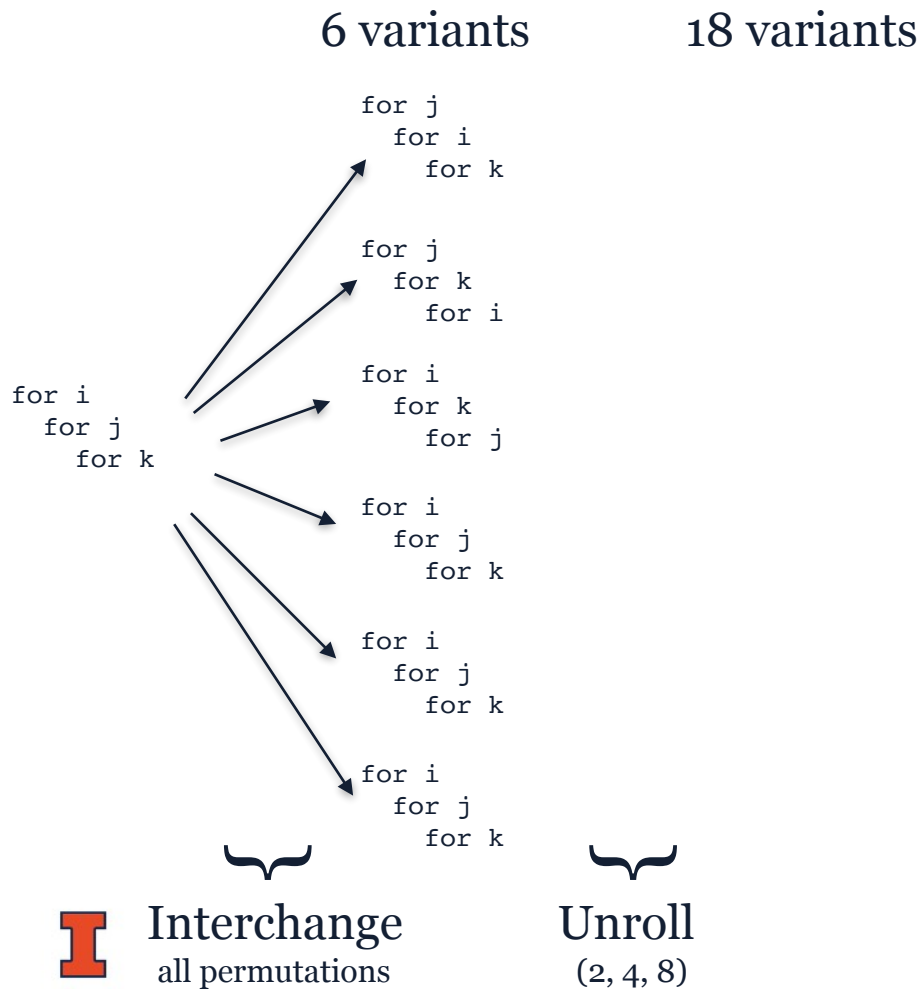
- triple nested loop

6 variants



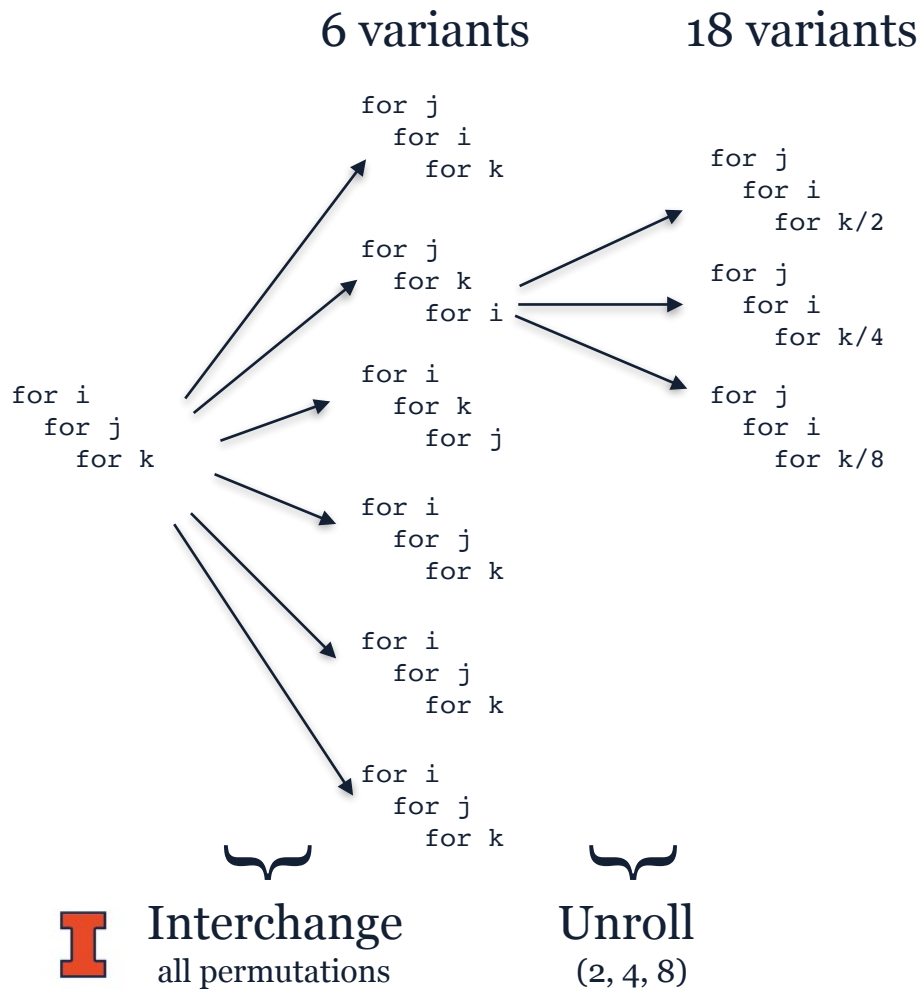
Optimization Space

- triple nested loop



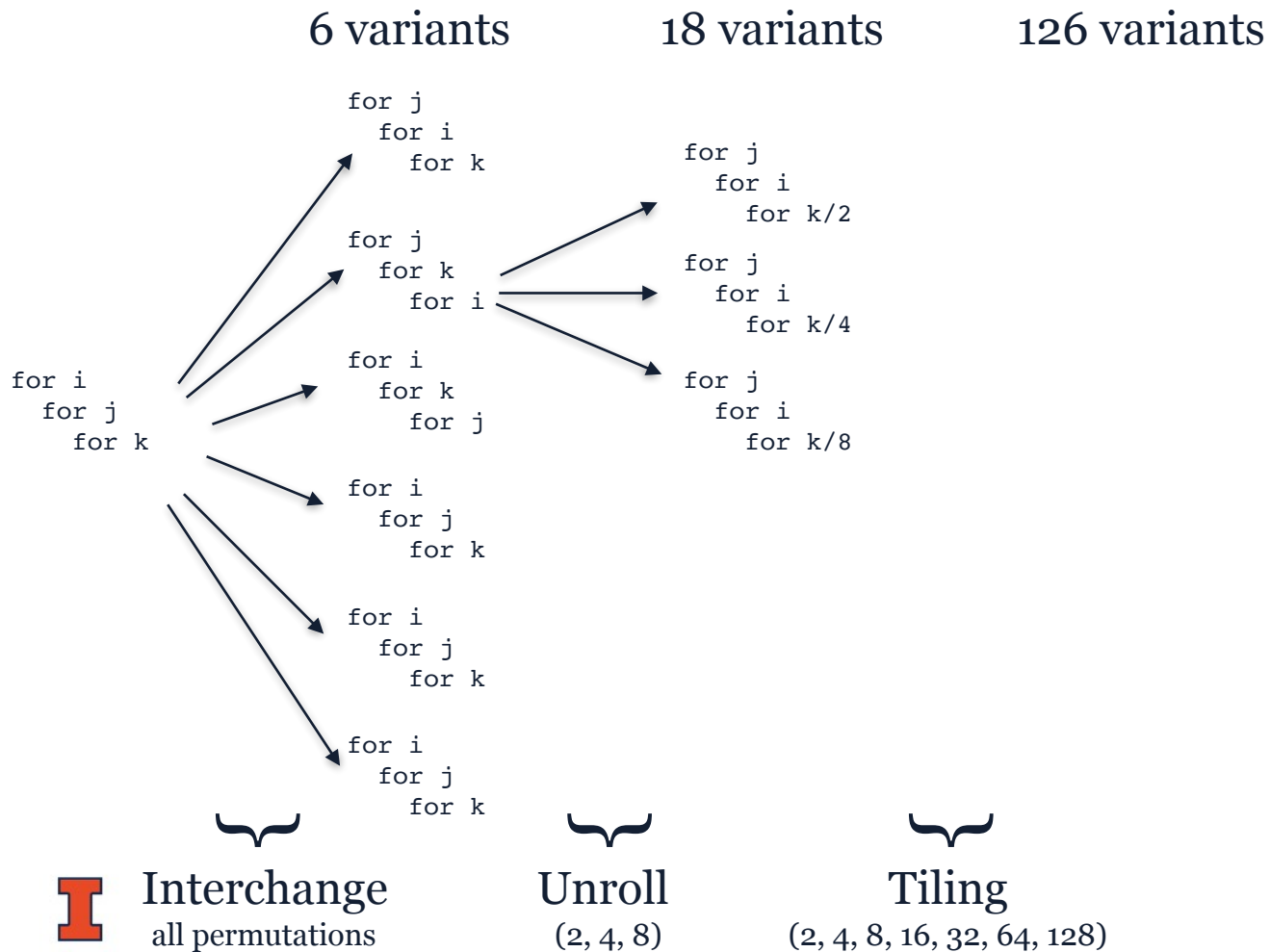
Optimization Space

- triple nested loop



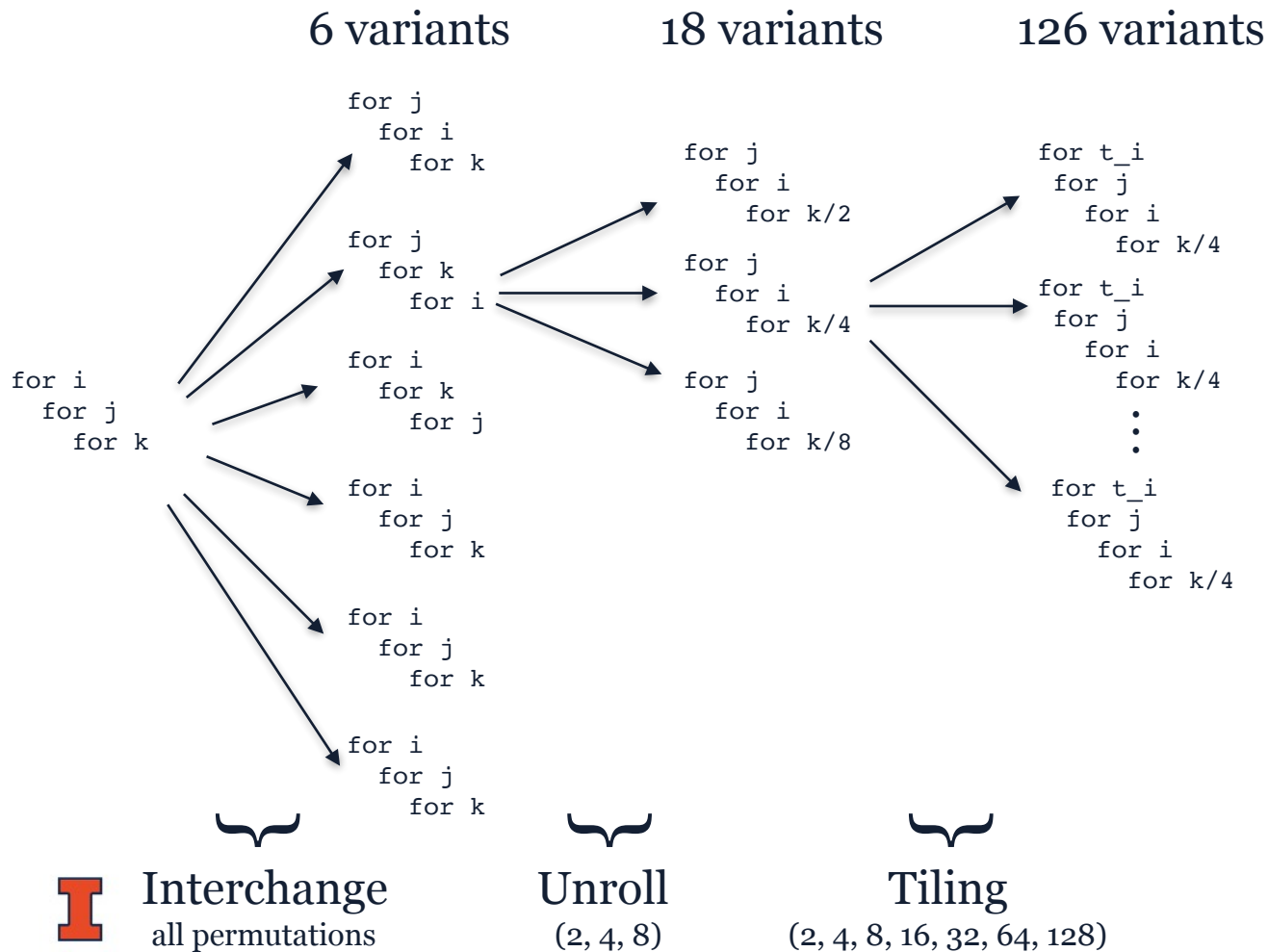
Optimization Space

- triple nested loop



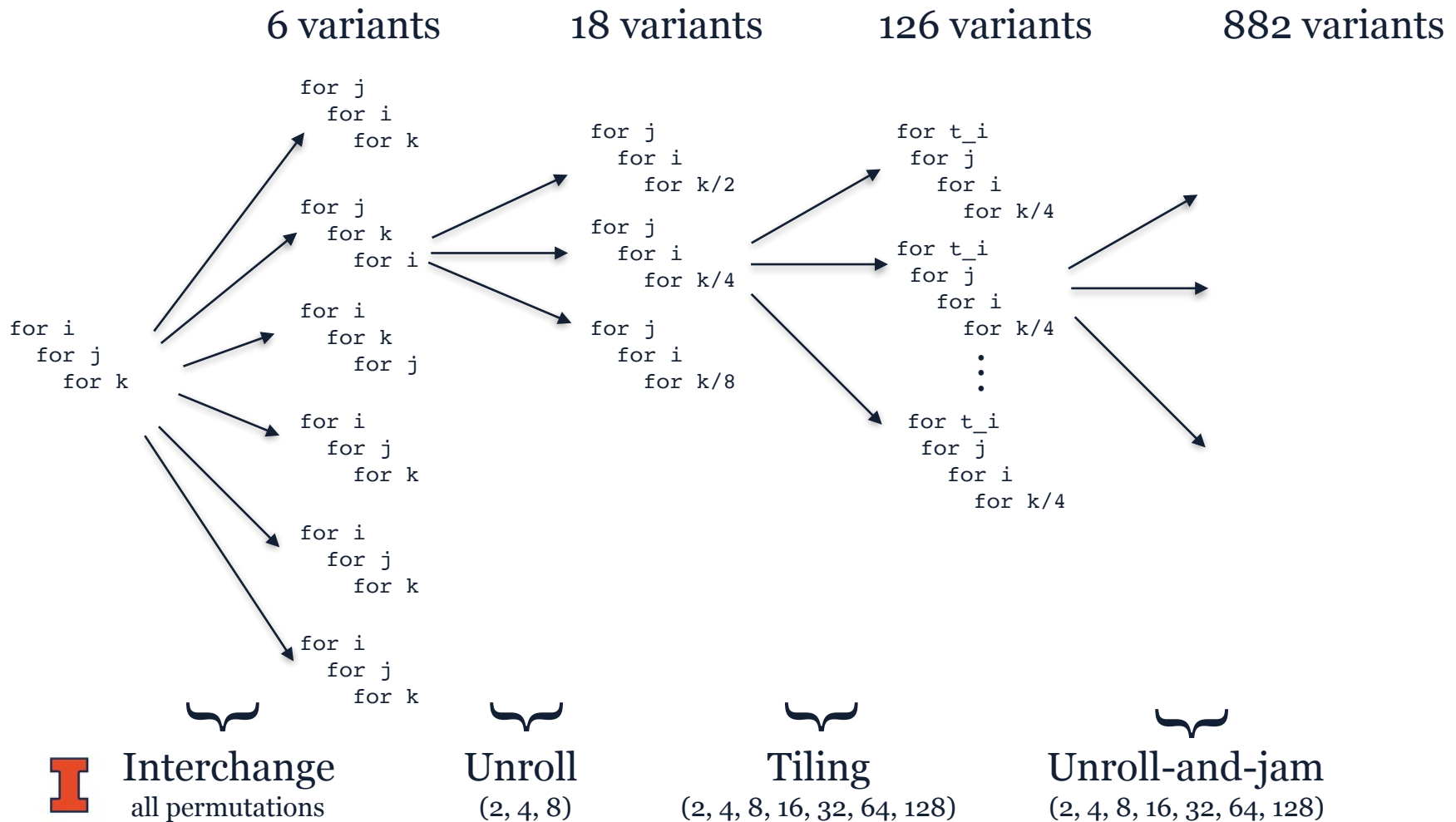
Optimization Space

- triple nested loop

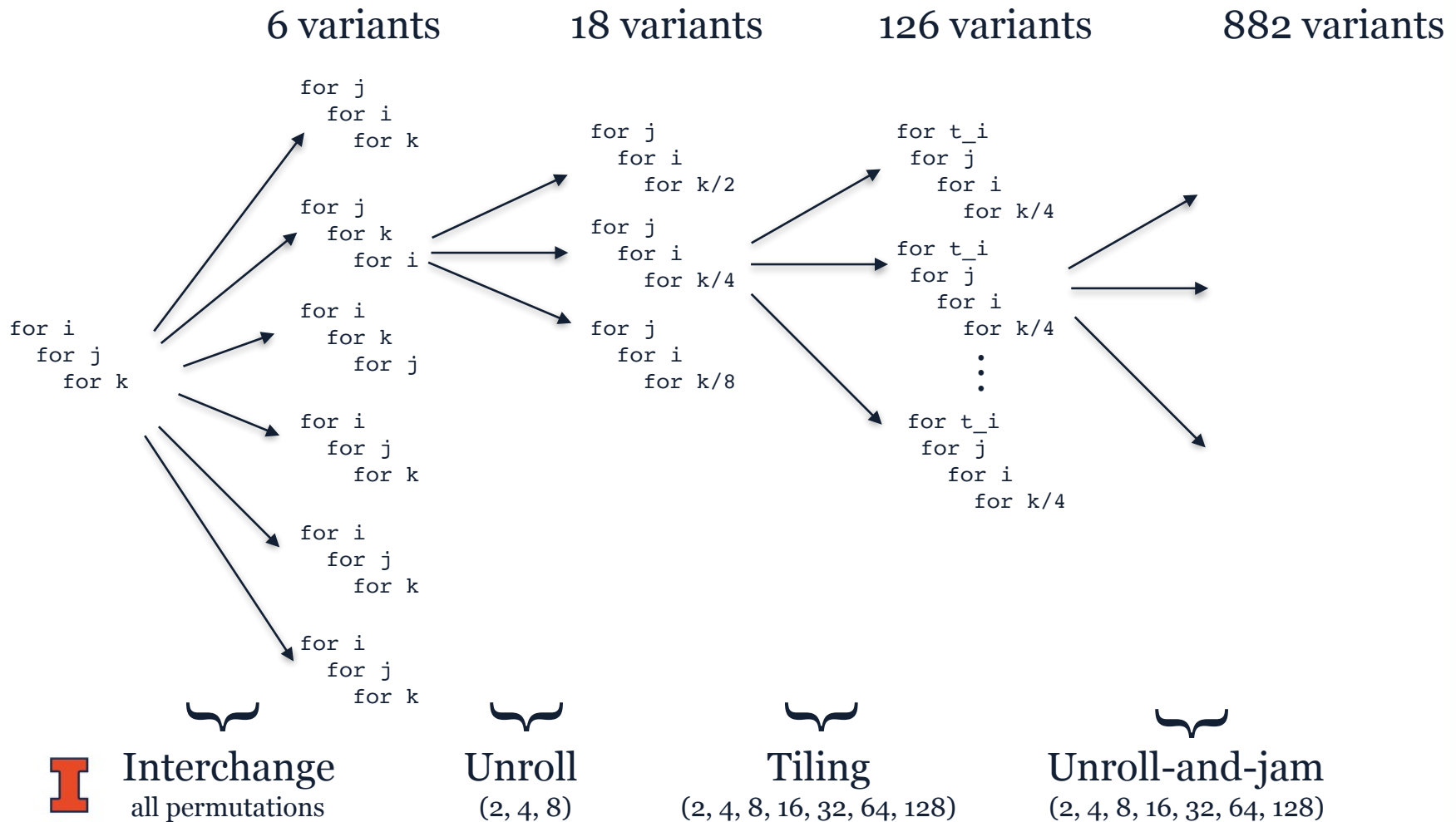


Optimization Space

- triple nested loop



Locus



Locus

Locus program

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

 + A black and white icon of a clipboard with a sheet of paper. The paper contains several lines of code, including the 'for' loops seen to the left of the plus sign, and some arrows pointing left.

Locus

Locus program

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

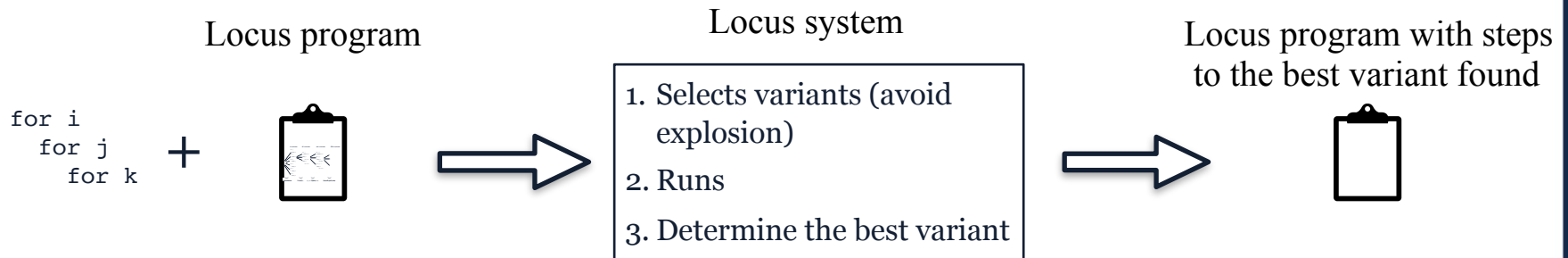
+



Locus system

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

Locus



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

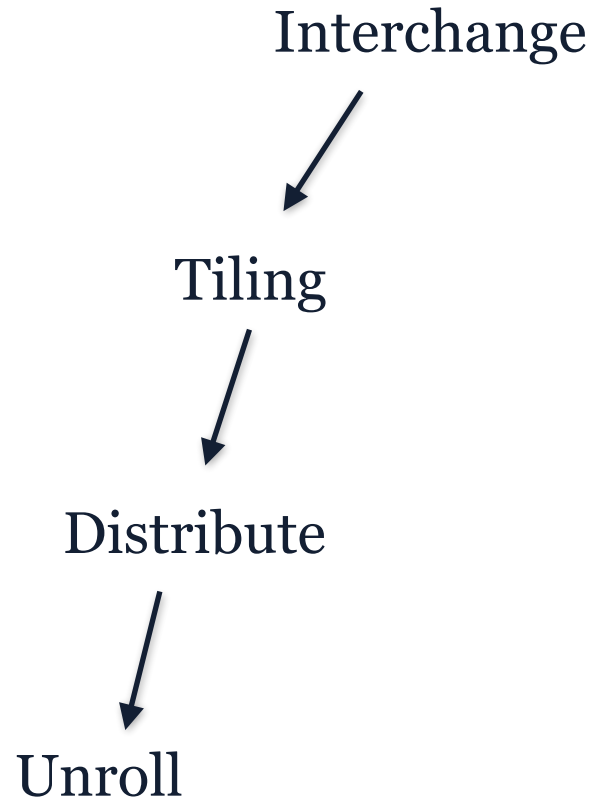


Distribute

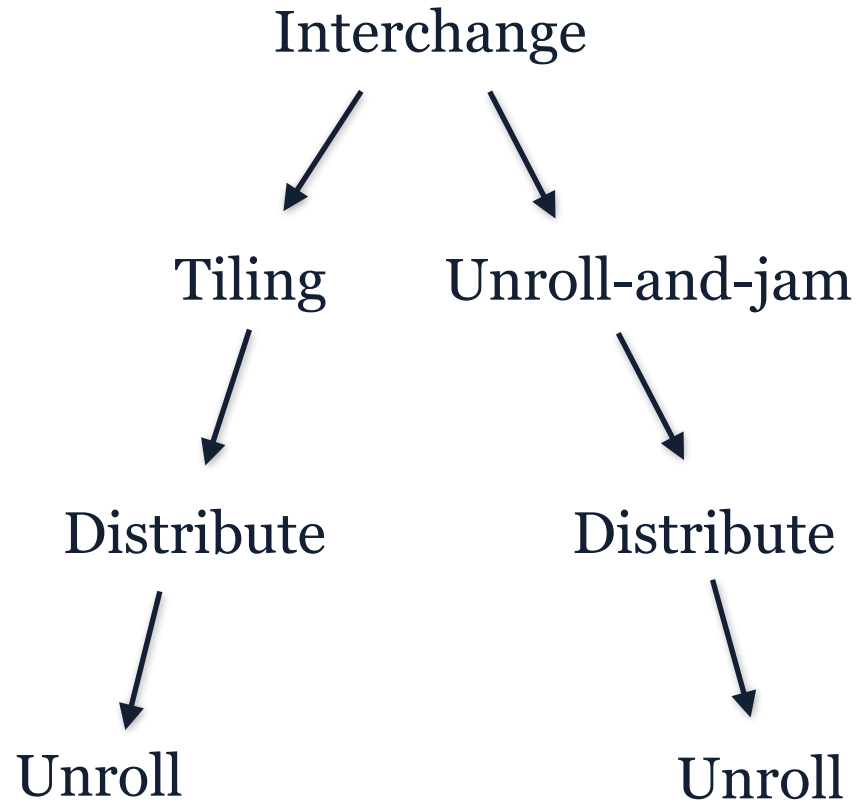


Unroll

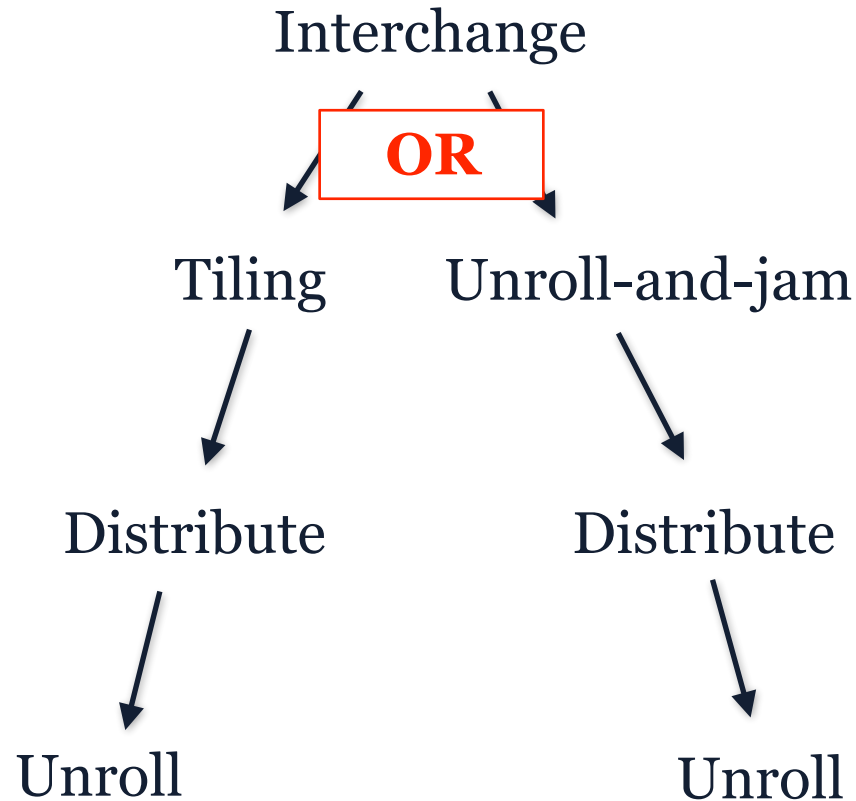
Locus Optimization Language



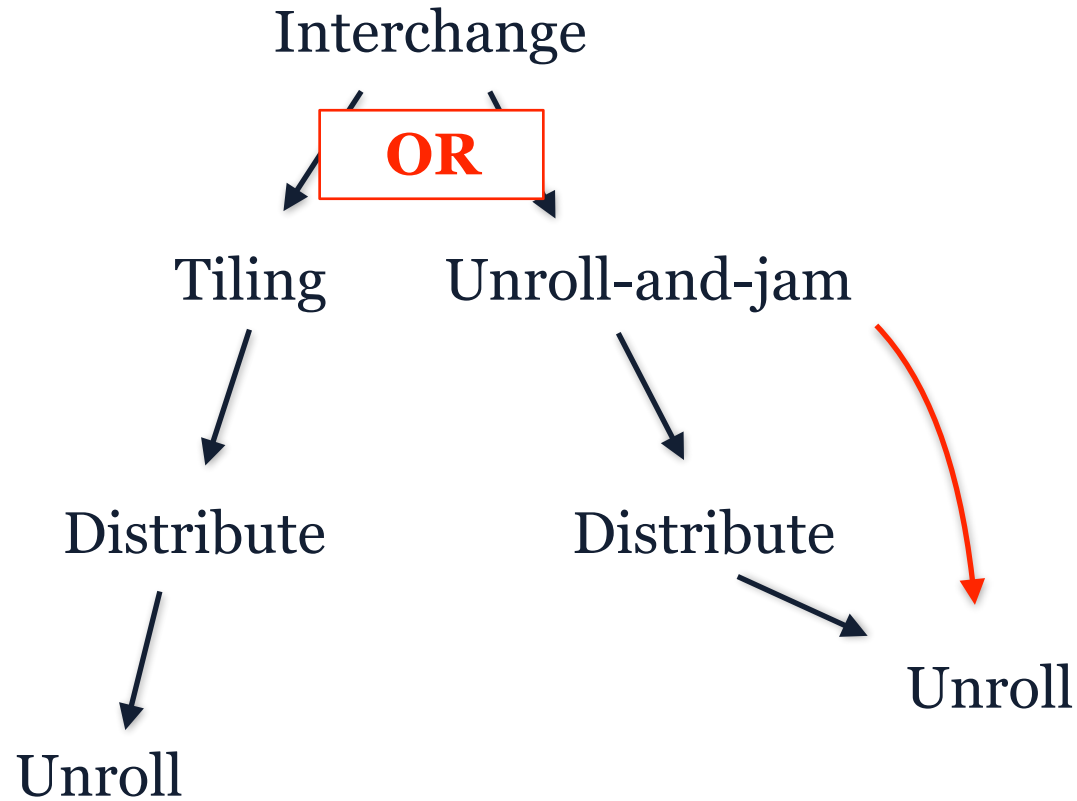
Locus Optimization Language



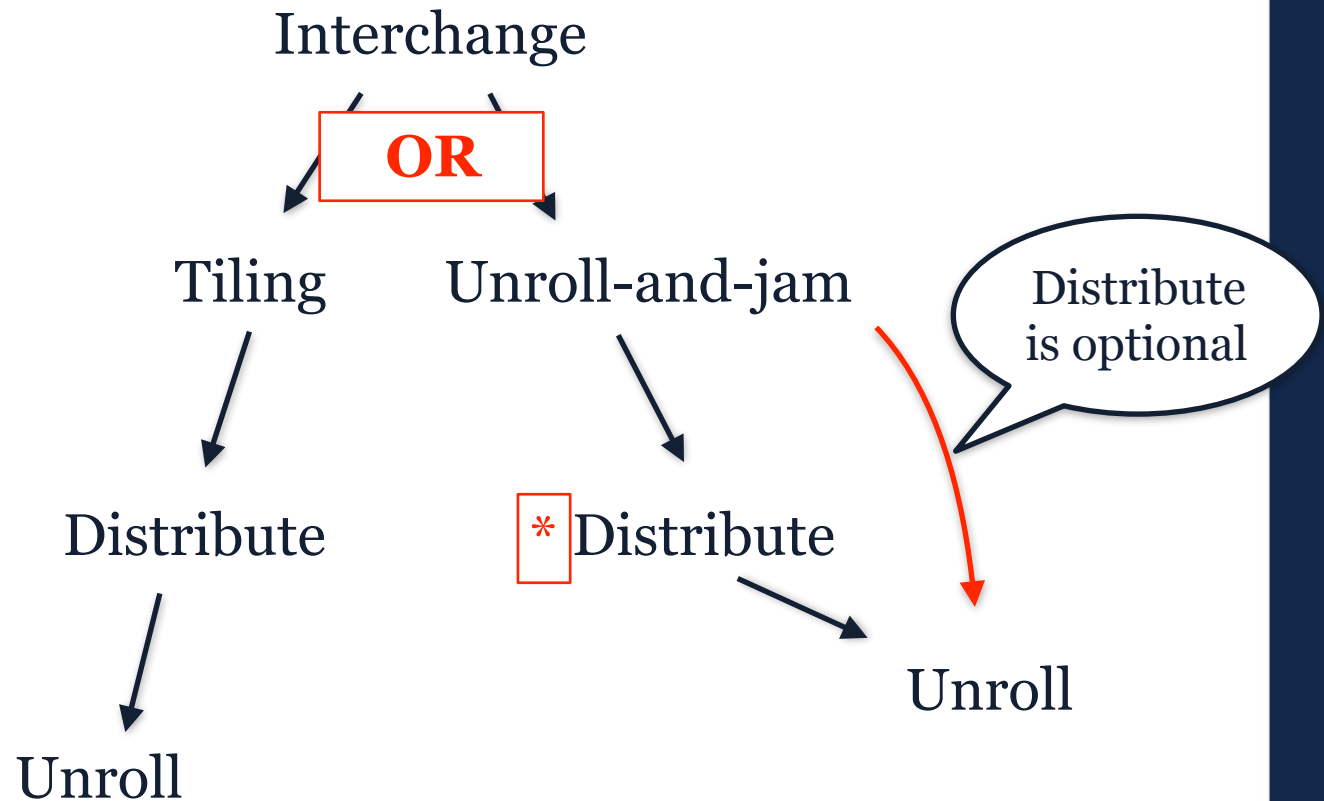
Locus Optimization Language



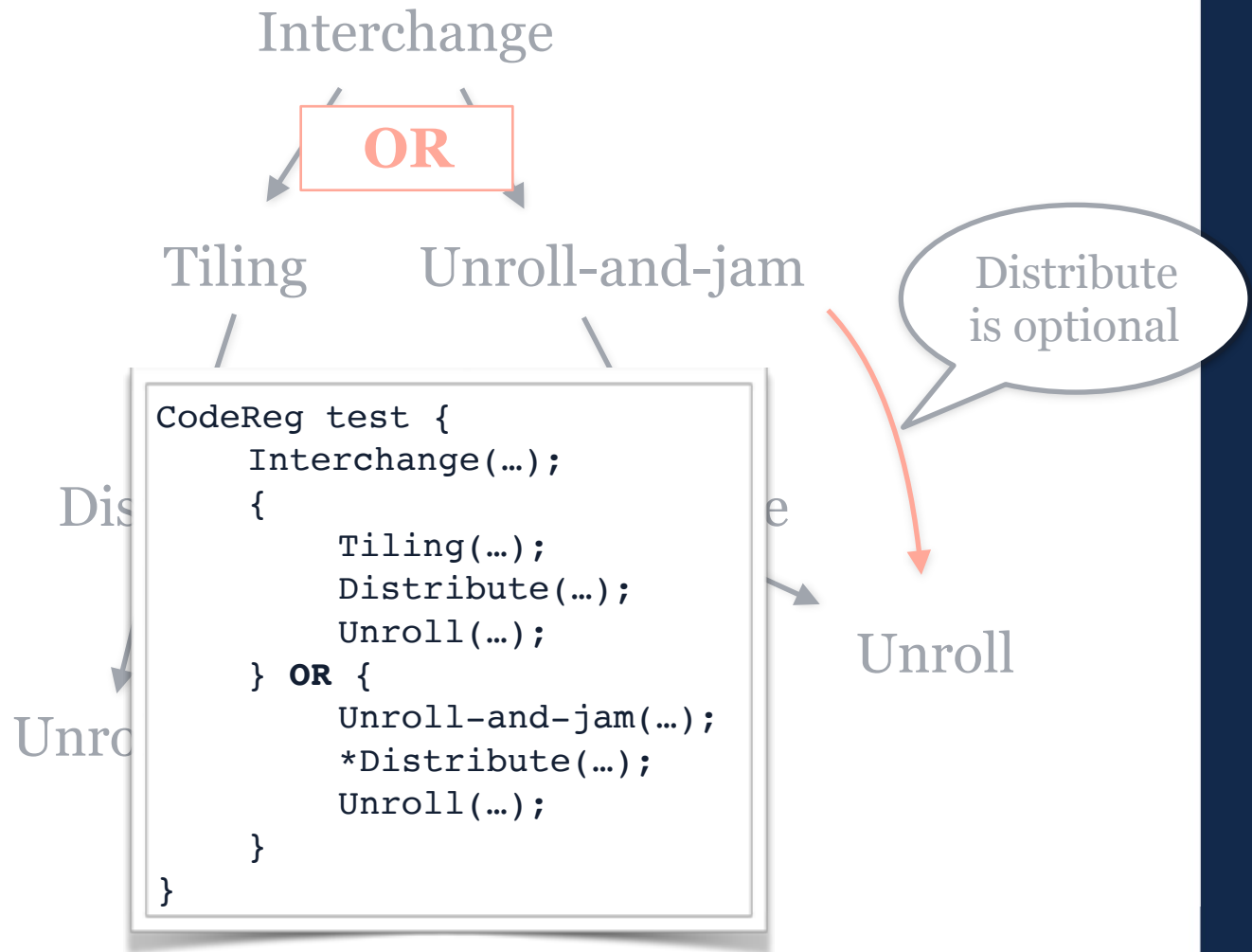
Locus Optimization Language



Locus Optimization Language



Locus Optimization Language



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}

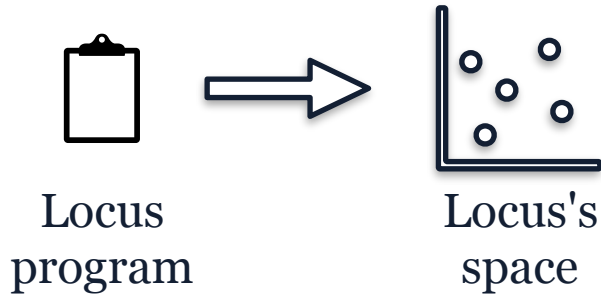
- Search modules (OpenTuner, HyperOpt):



Locus
program

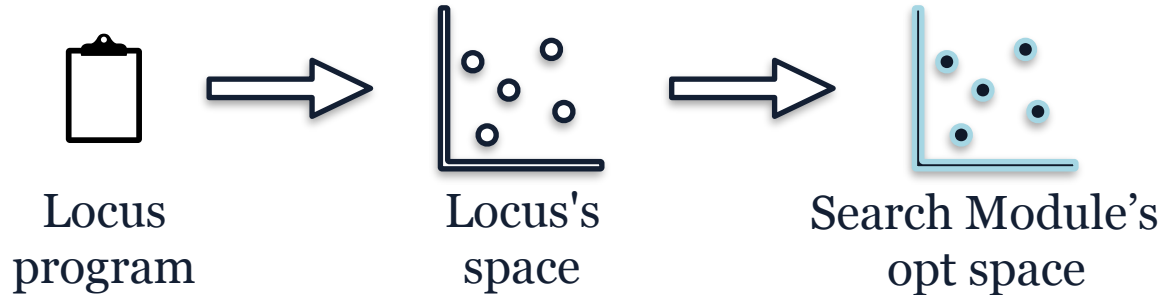
Modules Integration ^{2/3}

- Search modules (OpenTuner, HyperOpt):



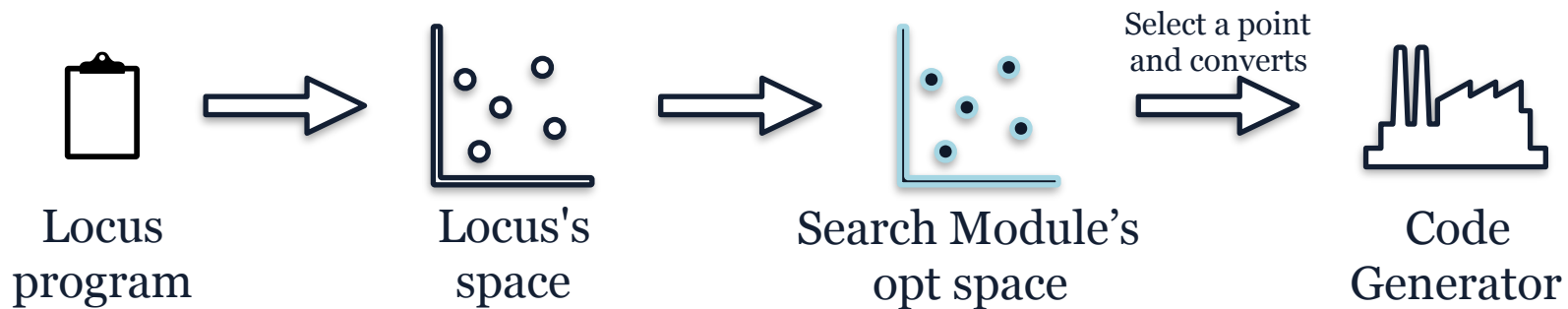
Modules Integration ^{2/3}

- Search modules (OpenTuner, HyperOpt):



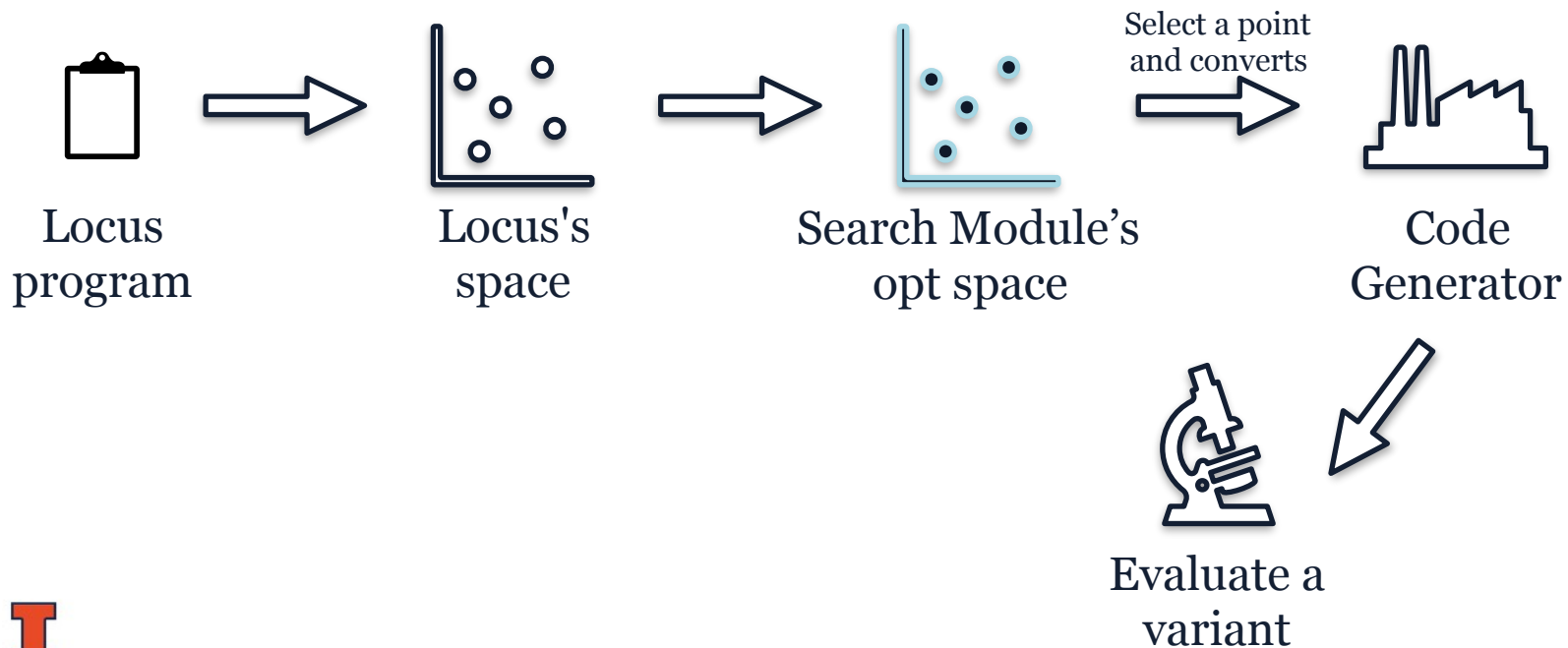
Modules Integration ^{2/3}

- Search modules (OpenTuner, HyperOpt):



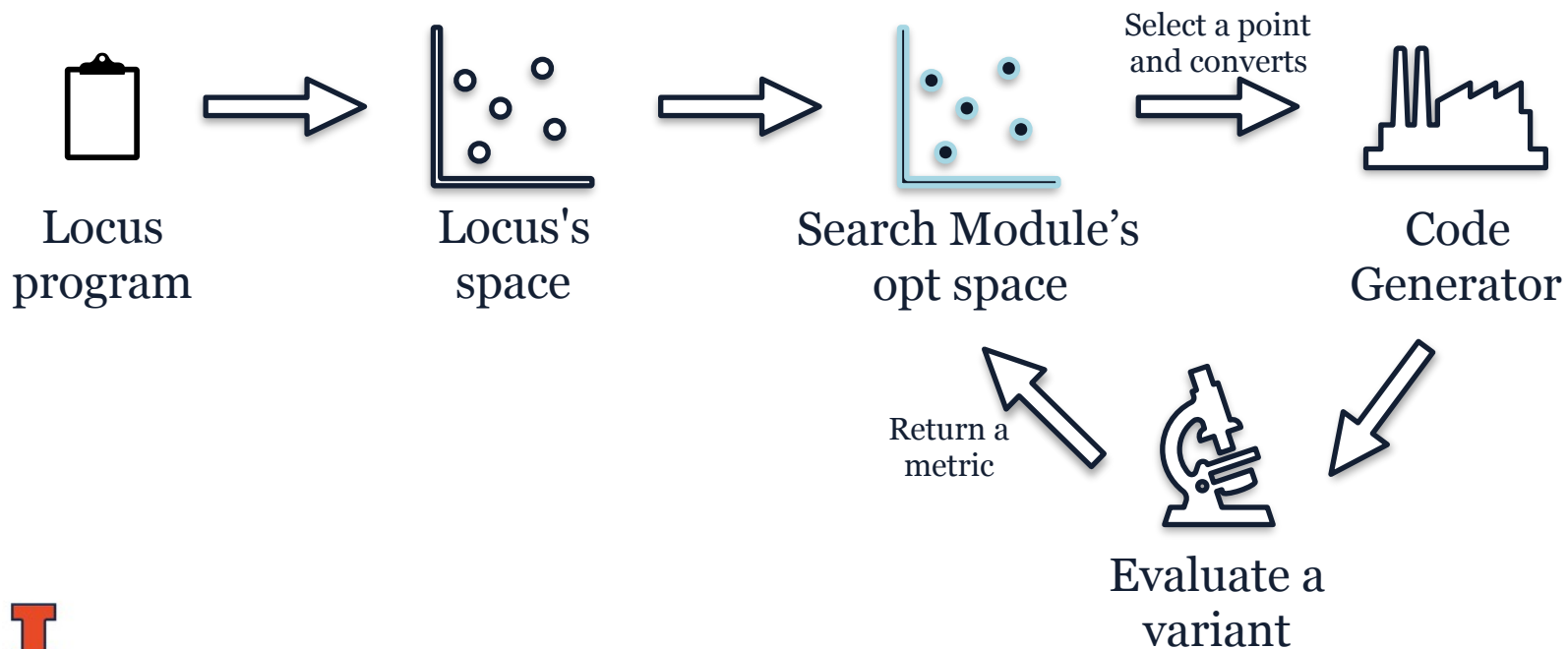
Modules Integration ^{2/3}

- Search modules (OpenTuner, HyperOpt):



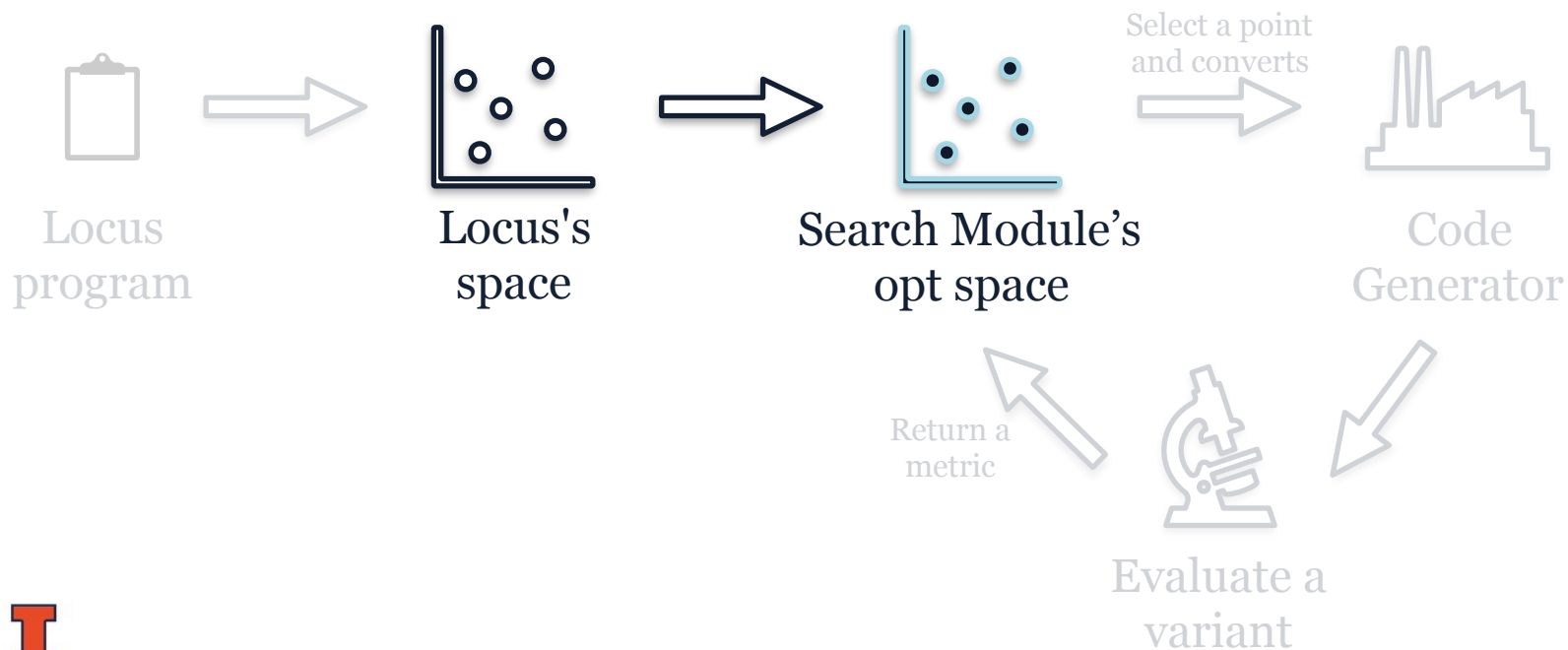
Modules Integration ^{2/3}

- Search modules (OpenTuner, HyperOpt):



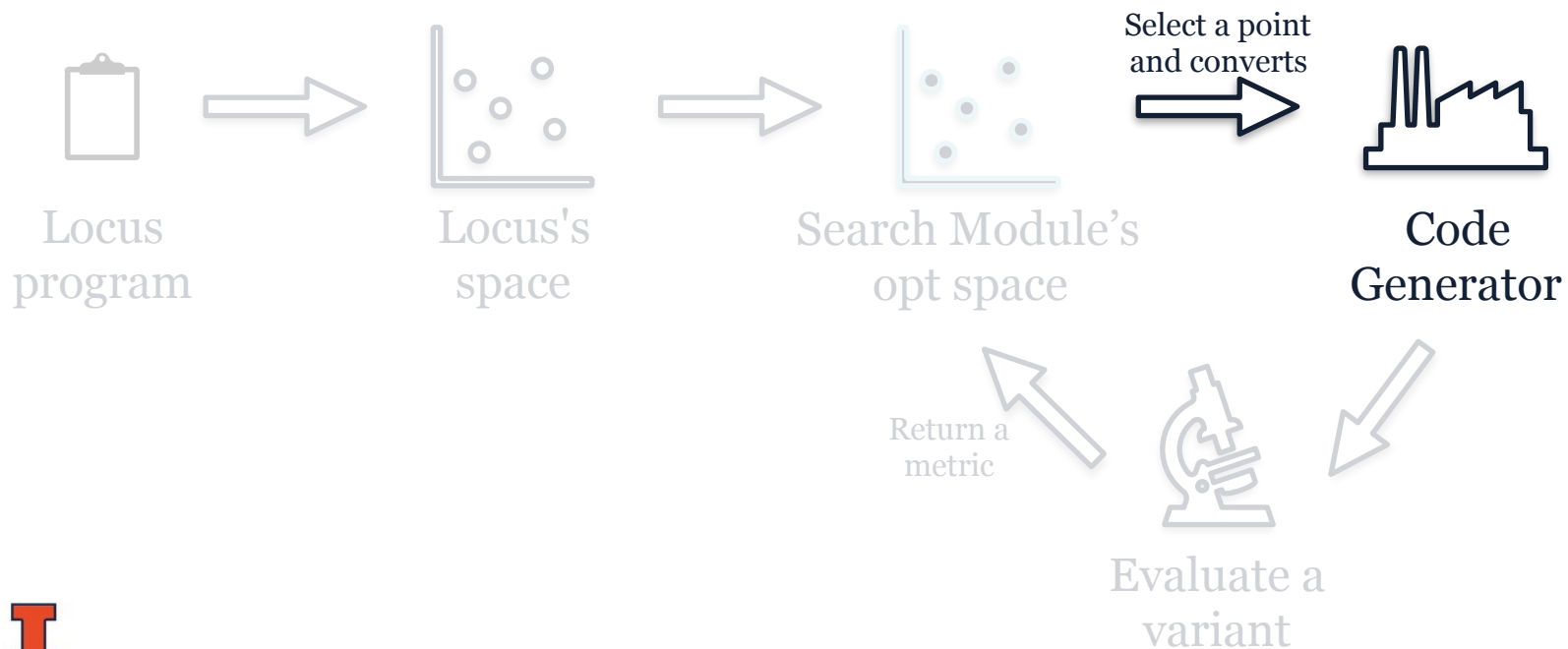
Modules Integration ^{2/3}

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



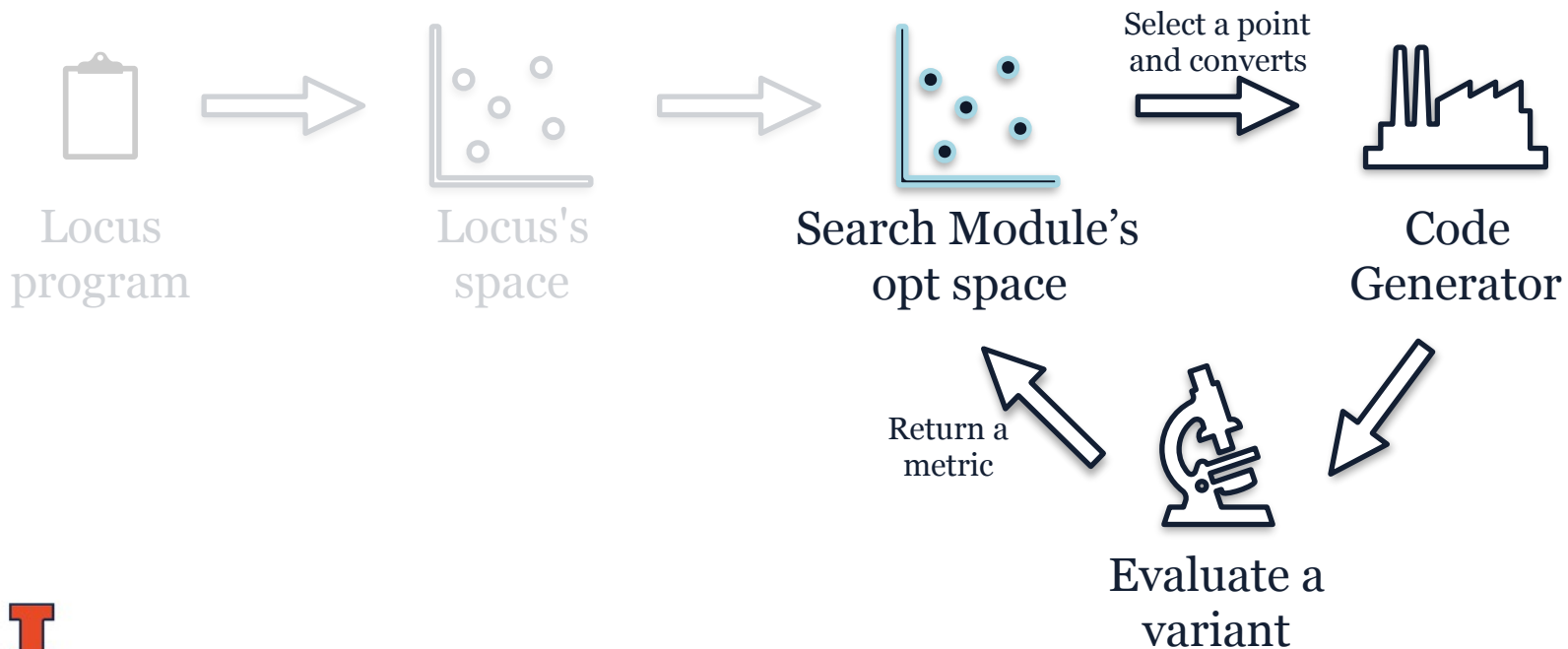
Modules Integration ^{2/3}

- 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}

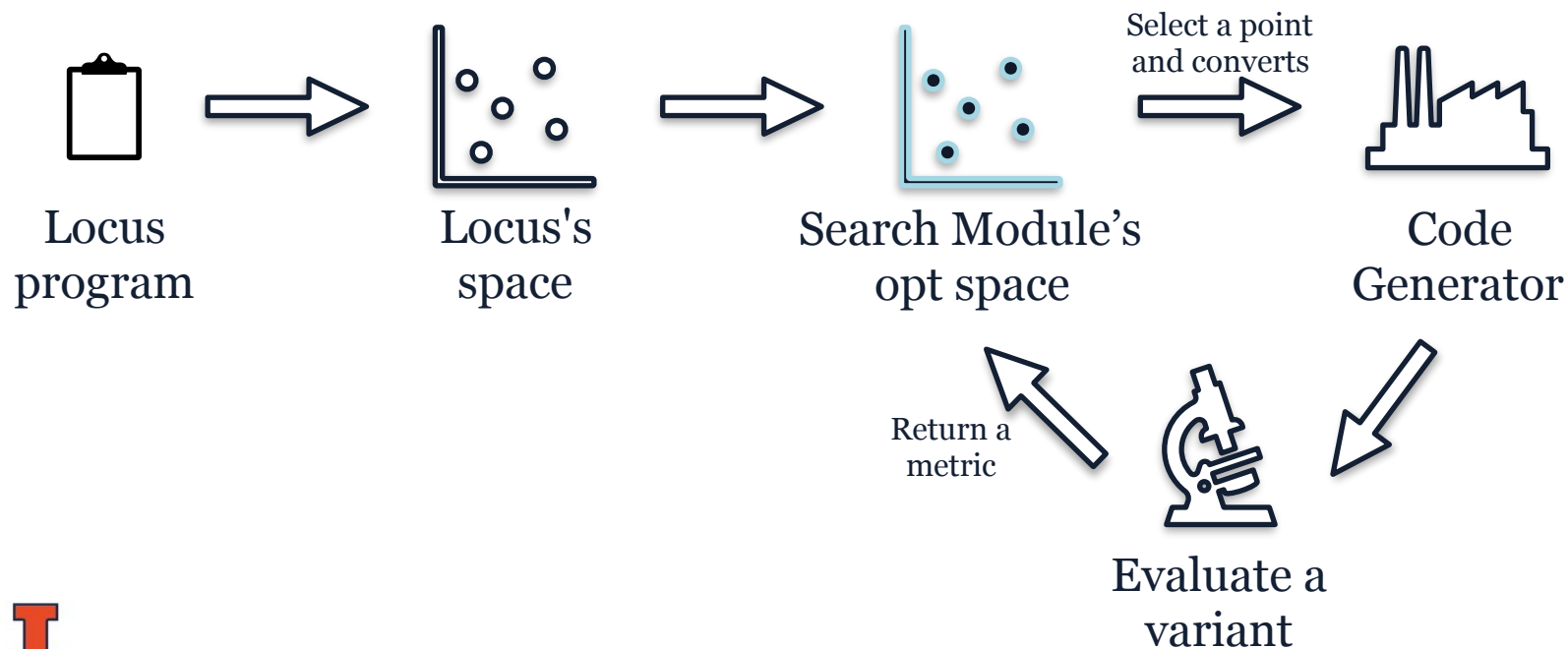
- Transformation modules (Pips, RoseLocus, Pragmas, BuiltIn):
 - Allows for fine-grain selection
 - Can pick a different module for each transformation (e.g., Interchange, Tiling)
 - Work on code region level
 - Workflow:
 - Locus transforms to modules notation
 - Module applies the optimization
 - Locus 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

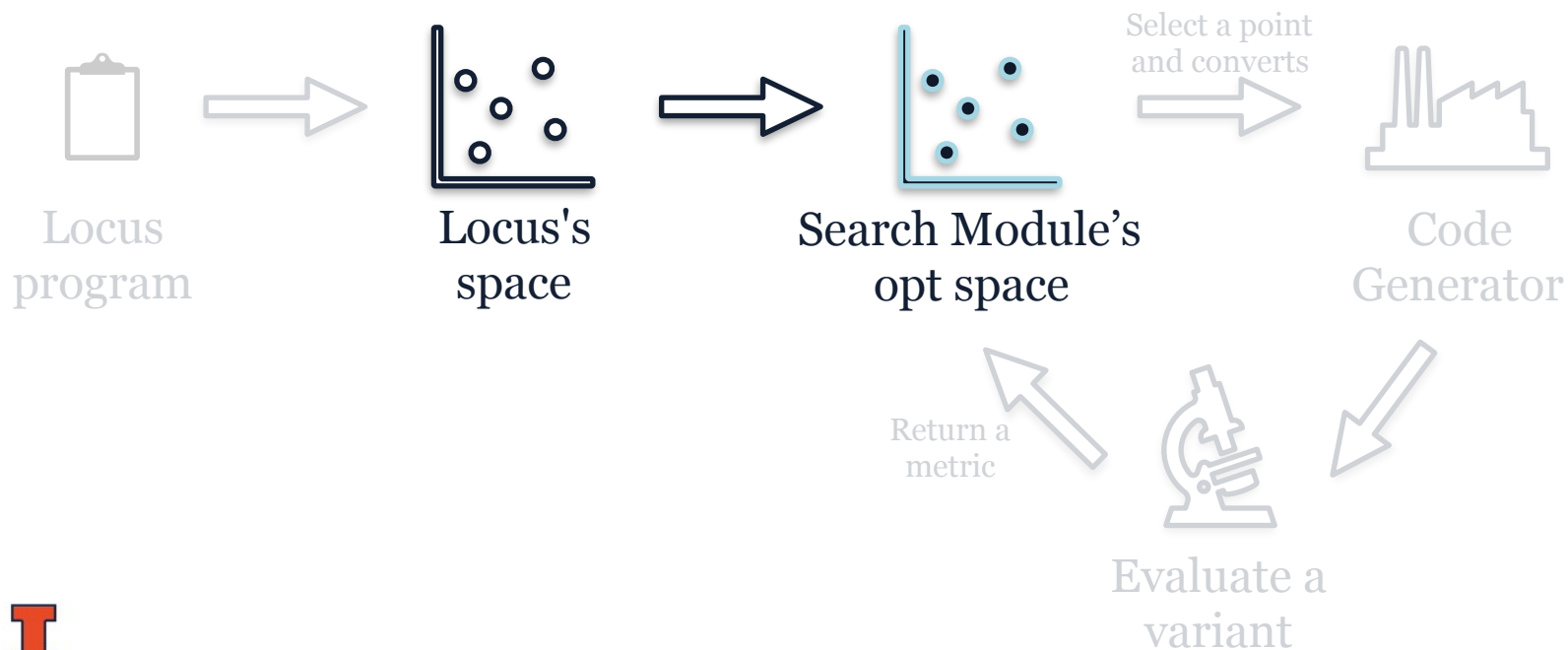
Optimizations for Pruning

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



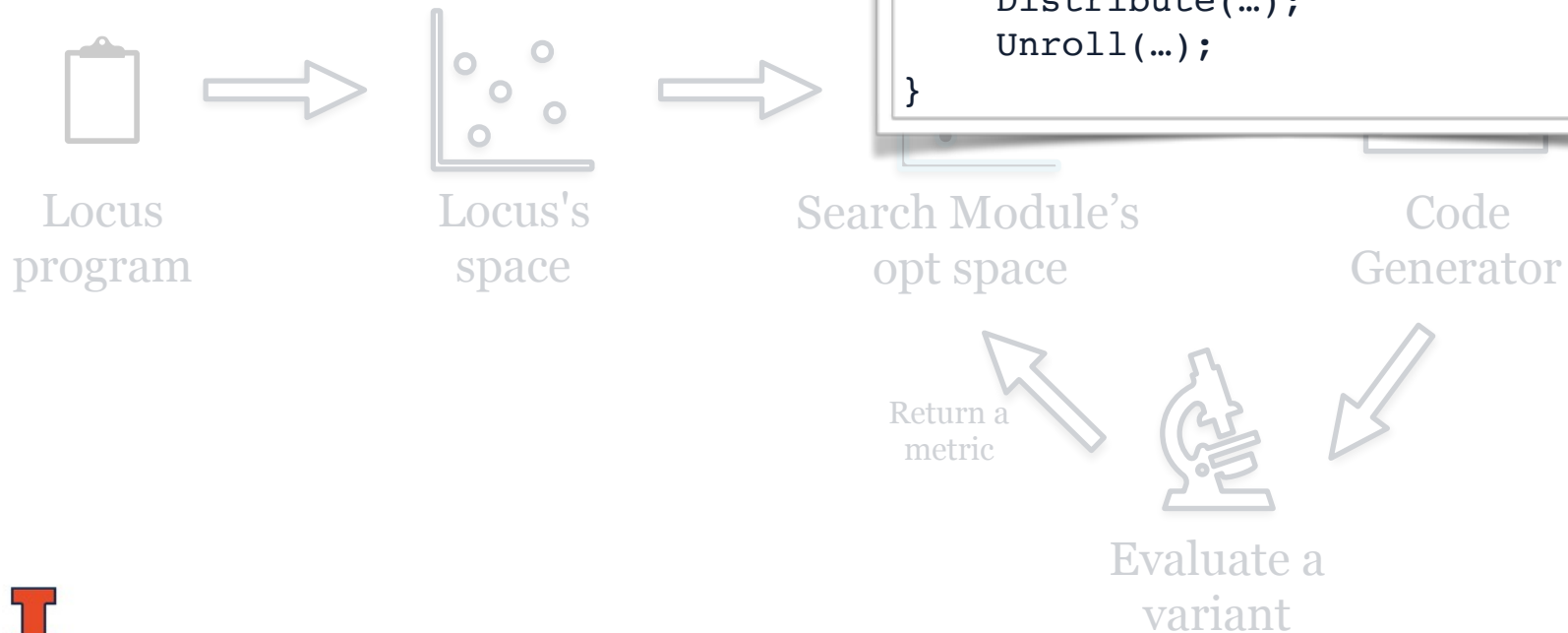
Optimizations for Pruning

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



Optimizations for Pruning

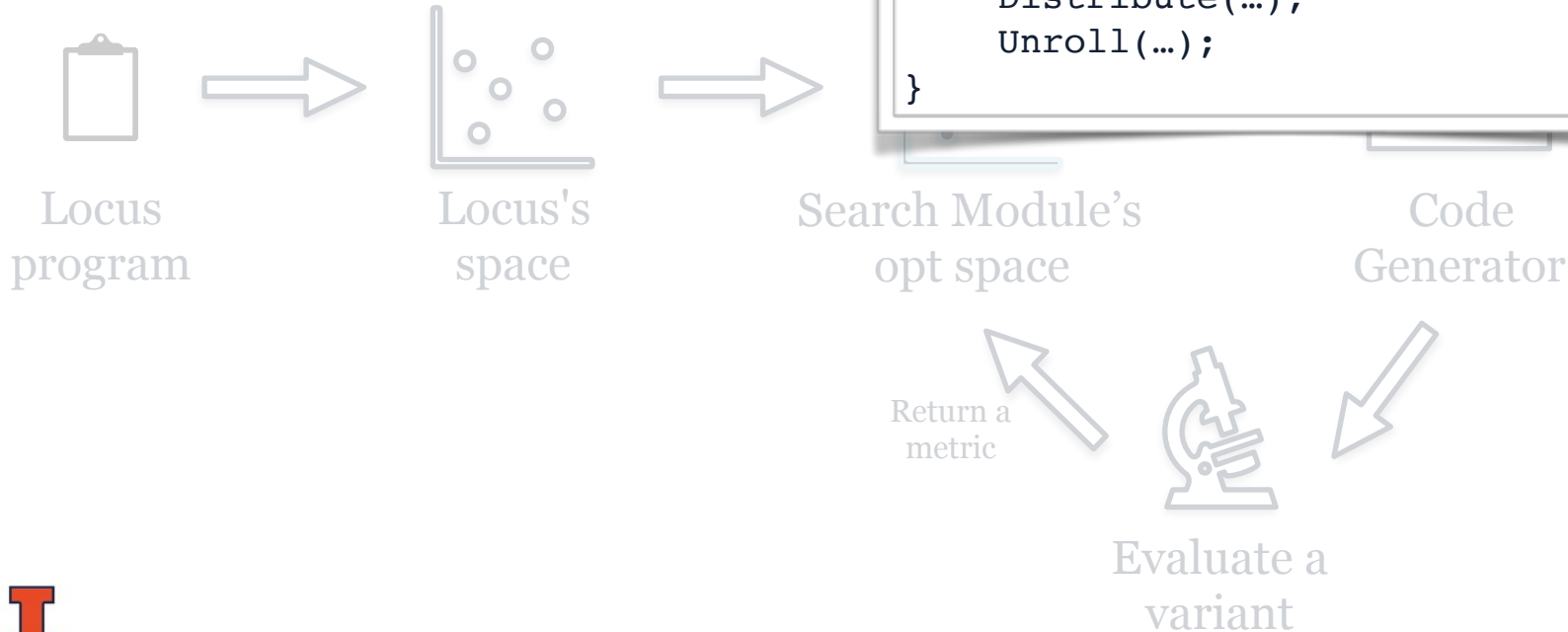
- During conversion:
 - Dead code elimination
 - Constant folding
 - Constant propagation



Optimizations for Pruning

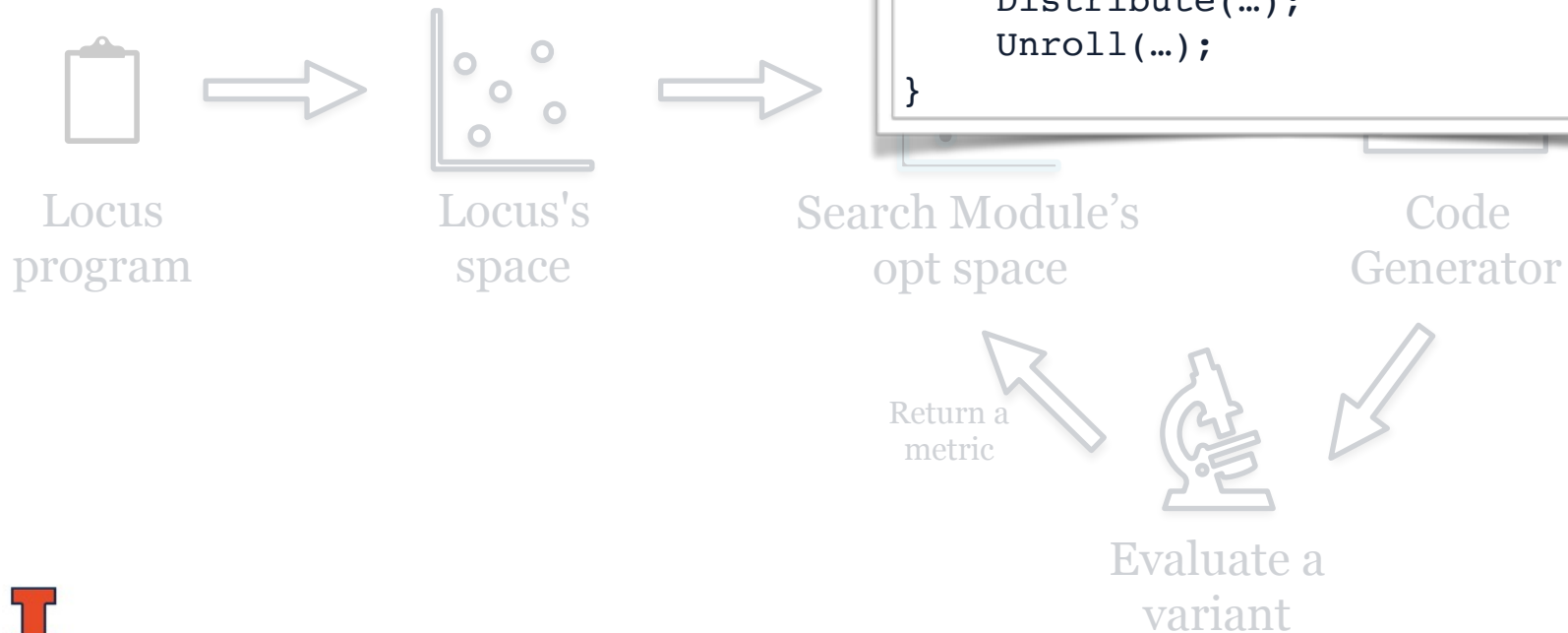
- During conversion:
 - Dead code elimination
 - Constant folding
 - Constant propagation

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



Optimizations for Pruning

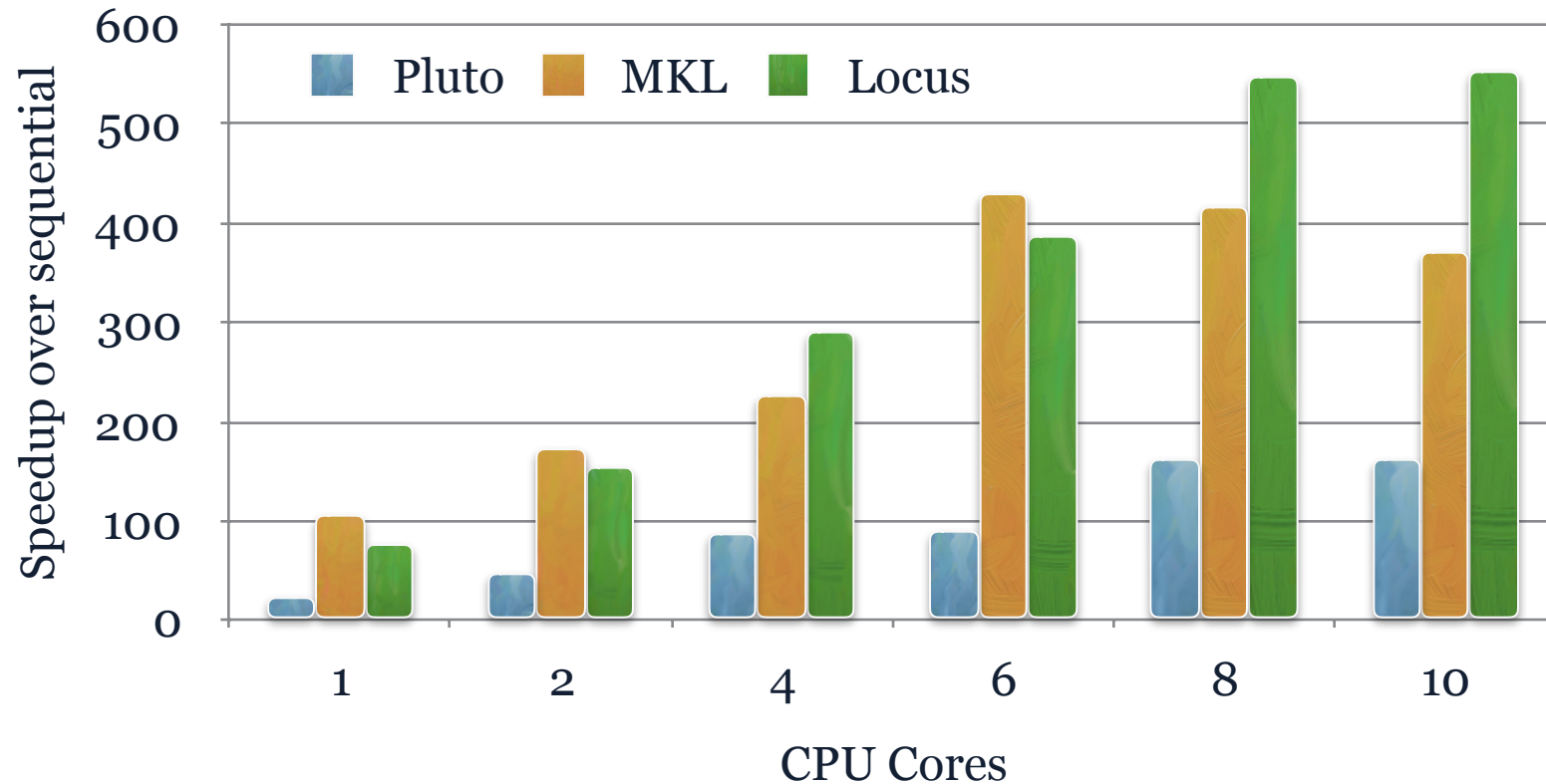
- During conversion:
 - Dead code elimination
 - Constant folding
 - Constant propagation



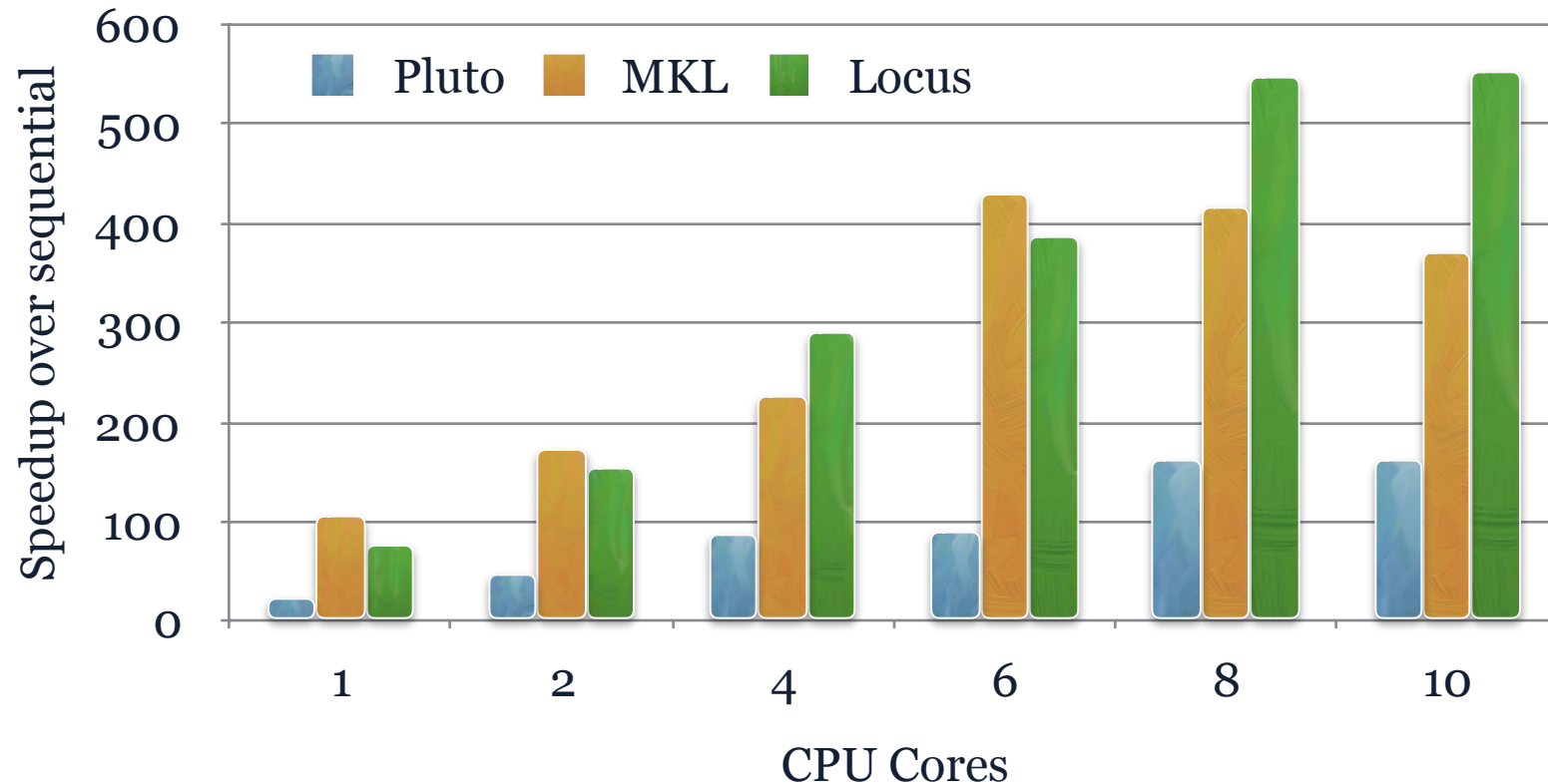
Experimental Results

- 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

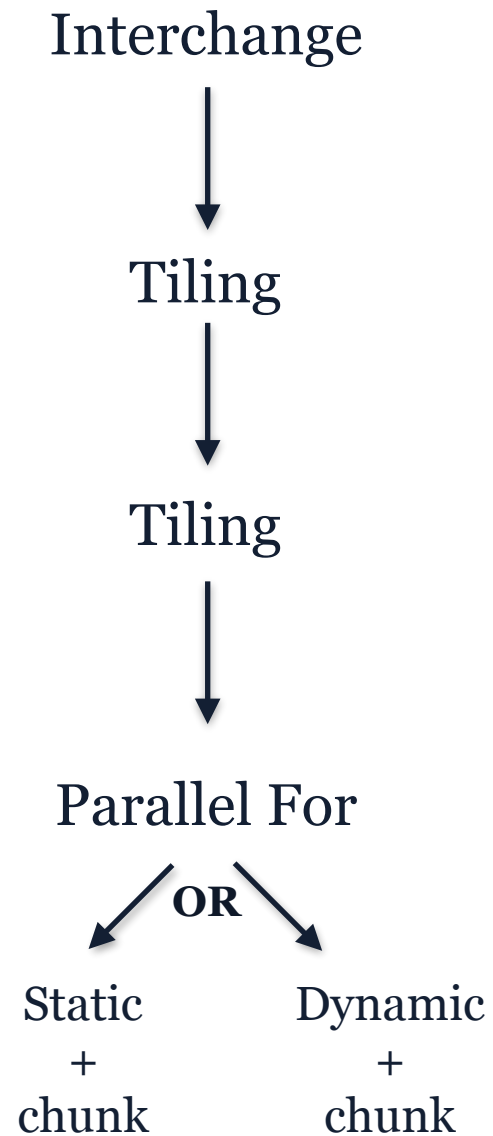


Tiling



Parallel For

Matrix-Matrix Multiplication



Matrix-Matrix Multiplication

- Large space of optimization

Interchange



Tiling



Tiling



Parallel For



Static
+
chunk

Dynamic
+
chunk

Matrix-Matrix Multiplication

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

Interchange



Tiling



Tiling



Parallel For



Static
+
chunk

Dynamic
+
chunk

Matrix-Matrix Multiplication

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

Interchange



Tiling



Tiling



Parallel For



Static
+
chunk

Dynamic
+
chunk

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



Static
+
chunk

Dynamic
+
chunk

Stencils



Stencils

- 6 different stencils

Stencils

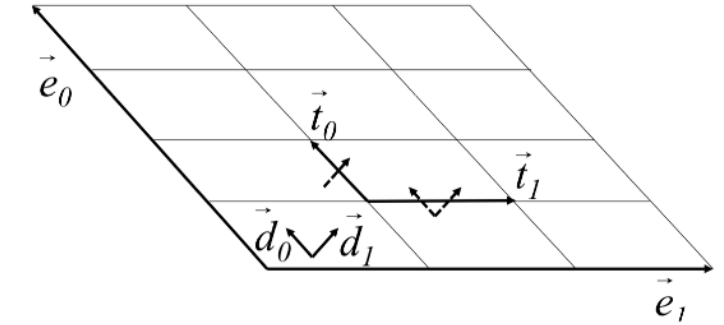
- 6 different stencils
- Skew tiling accross time-space

Stencils

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

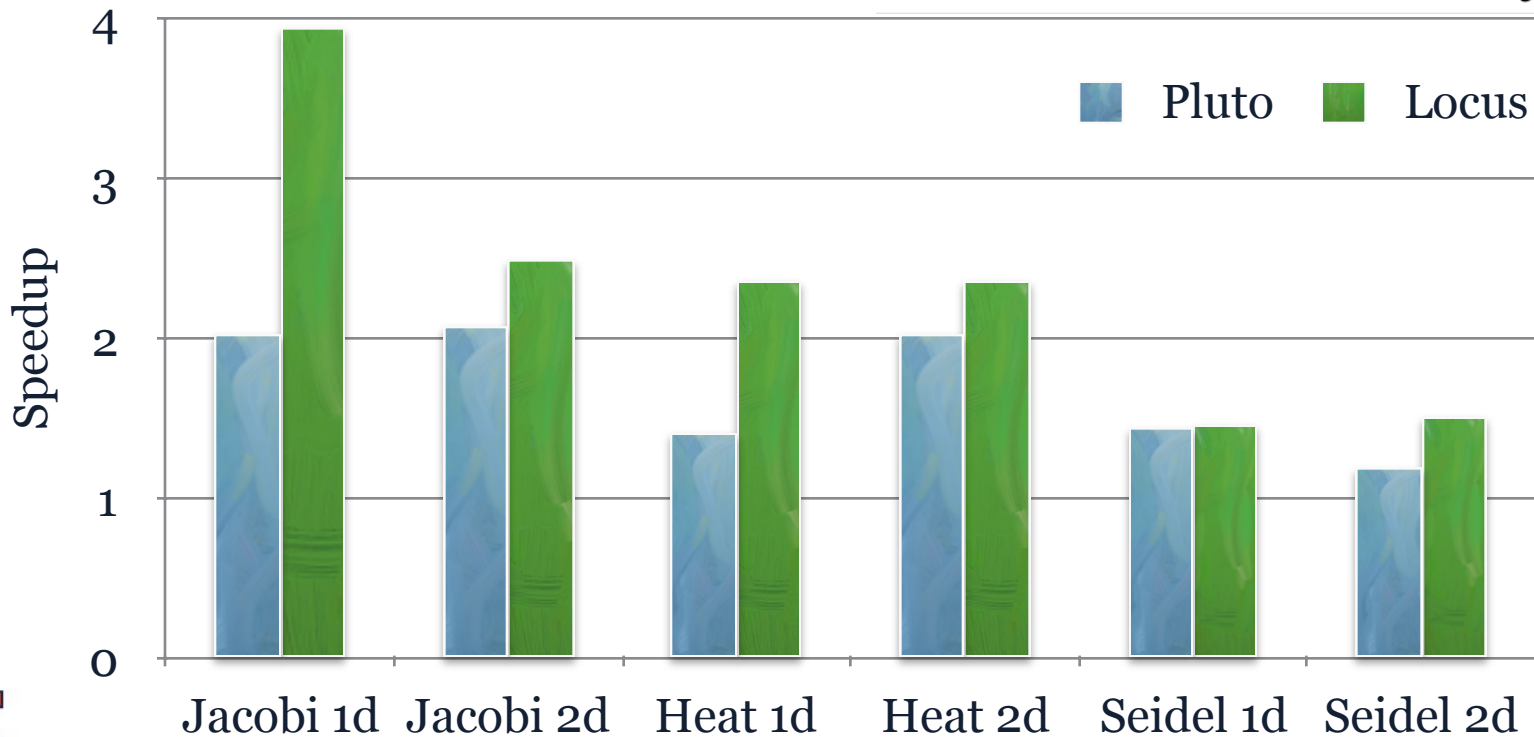
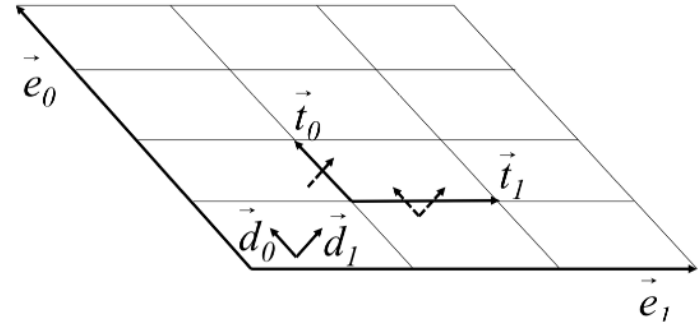
Stencils

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



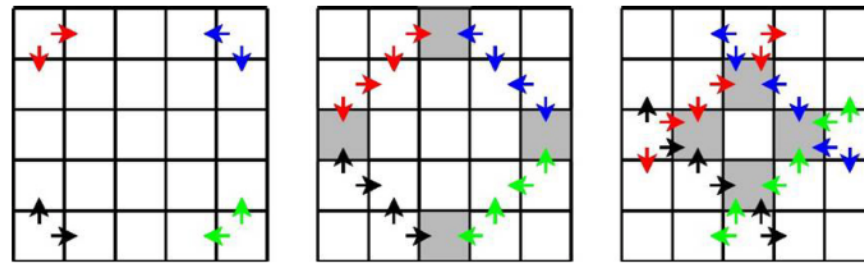
Stencils

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

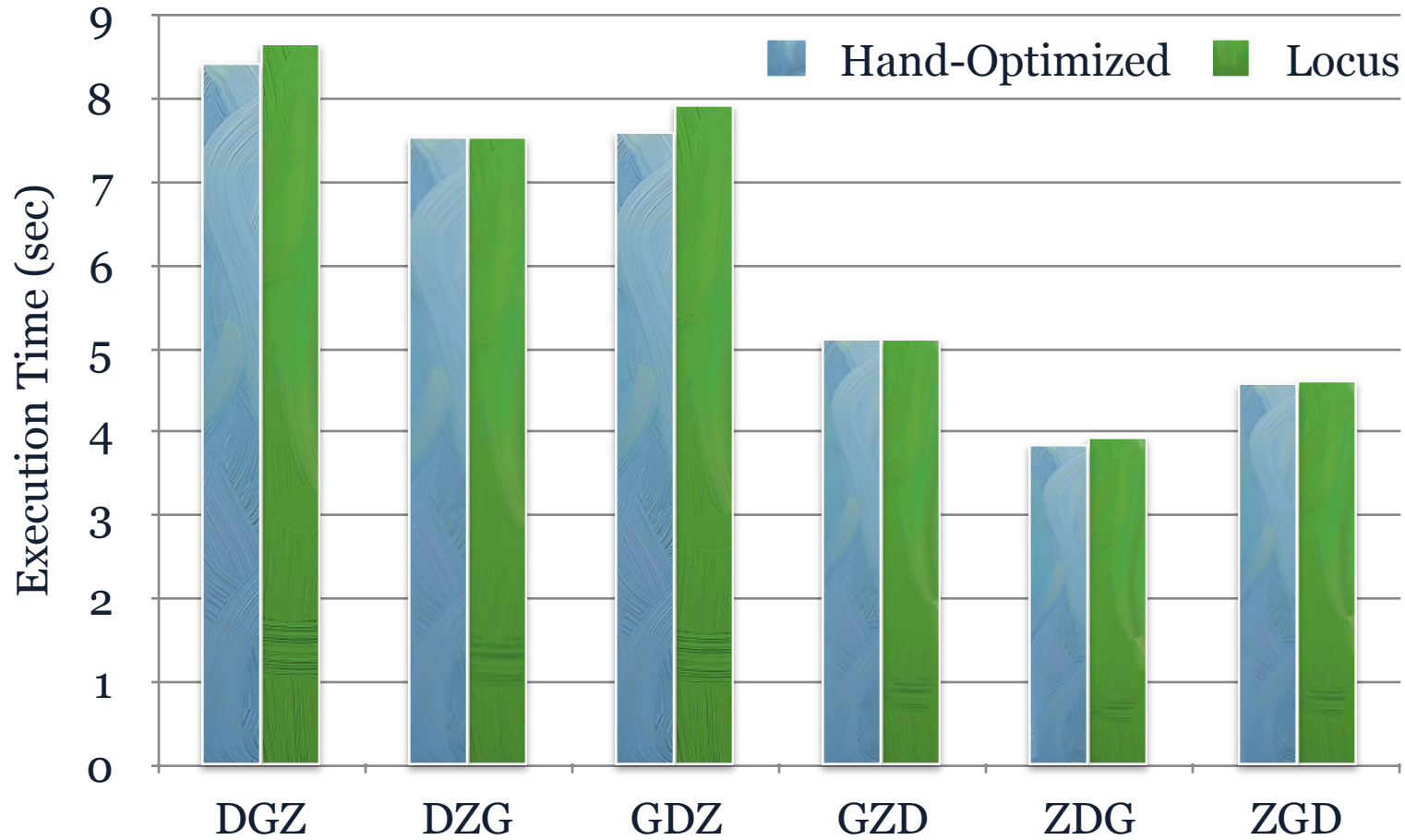


Kripke

- 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



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=omplloop);
}
```

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=ompleop);
}
```

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=omplloop);
}
```

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=omplloop);
}
```

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=omplloop);
}
```

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];

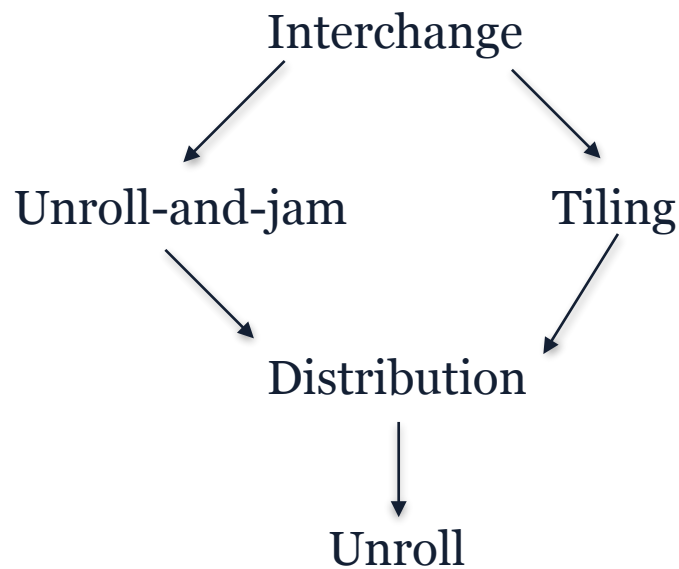
          #####
          # Address calculation to be inc
          #####

          *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=omplloop);
}
```

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
Cortextsuite [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

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));
}

```

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040

```

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

```

1  CodeReg scop {
2    perfect = BuiltIn.IsPerfectLoopNest();
3    depth = BuiltIn.LoopNestDepth();
4    if (RoseLocus.IsDepAvailable()) {
5      if (perfect && depth > 1) {
6        permorder = permutation(seq(0,depth));
7        RoseLocus.Interchange(order=permorder);
8      }
9    }
10   if (perfect) {
11     indexT1 = integer(1..depth);
12     T1fac = poweroftwo(2..32);
13     RoseLocus.Tiling(loop=indexT1, factor=T1fac);
14   }
15   OR {
16     if (depth > 1) {
17       indexUAJ = integer(1..depth-1);
18       UAJfac = poweroftwo(2..4);
19       RoseLocus.UnrollAndJam(loop=indexUAJ,
20                            factor=UAJfac);
21     }
22   } OR {
23     None; # No tiling, interchange, or unroll and jam.
24   }
25   innerloops = BuiltIn.ListInnerLoops();
26   *RoseLocus.Distribute(loop=innerloops);
27 }
28 innerloops = BuiltIn.ListInnerLoops();
29 RoseLocus.Unroll(loop=innerloops,
30                  factor=poweroftwo(2..8));
31 }

```

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

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



1200+ lines of code

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!

I ILLINOIS

CGO - Washington, DC - Feb 2019