Jan 22, 2025

Qualcomm

Dead Iteration Elimination

Cedric Bastoul

Maxime Schmitt

Benoit Meister

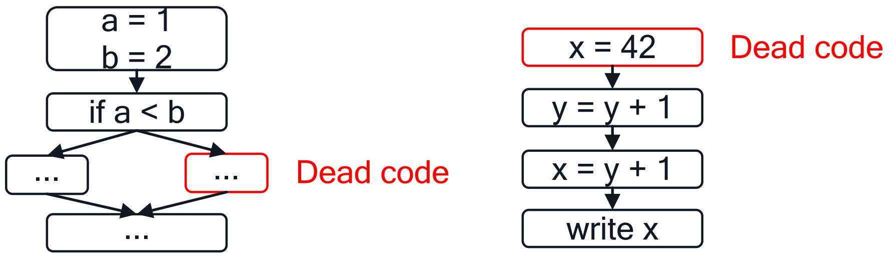
Chandan Reddy

We are not monsters

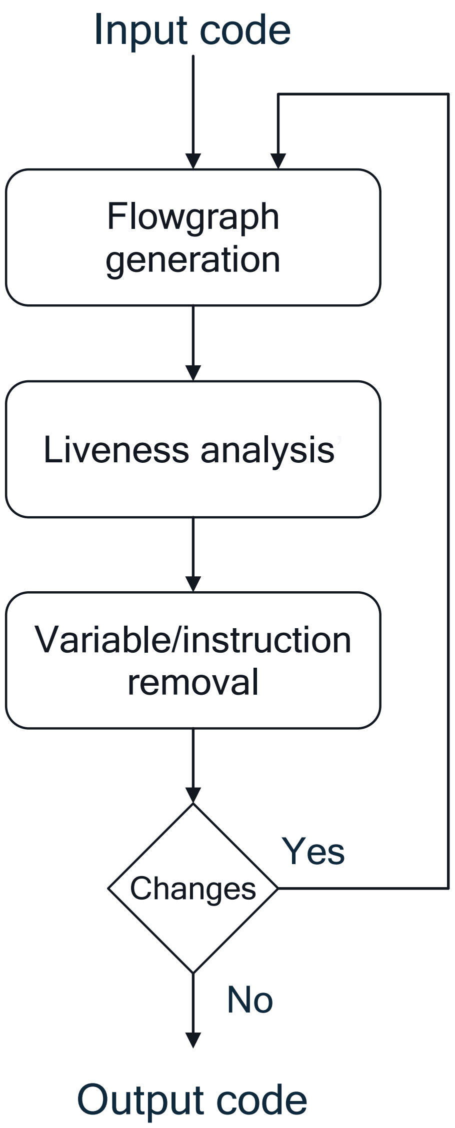
No iterations were harmed during the making of this presentation!

Prior Art

- **Dead Code Elimination (DCE)**
  - Elimination of code either never executed or producing never used data

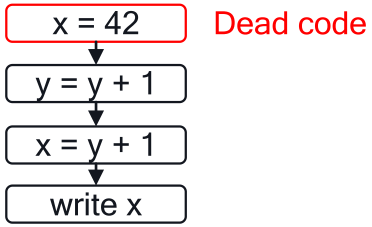


  - Exploit SSA form and data-flow analysis information [Cytron et al. TOPLAS 91]
  - Simple analysis of the uses of values and reachability of instructions
  - Iterative elimination until stability
- **Target complete removal of instructions**
  - Avoid allocating memory for unused variables
  - May ultimately remove full functions
  - No consideration of loop iteration spaces and removal of dead iterations

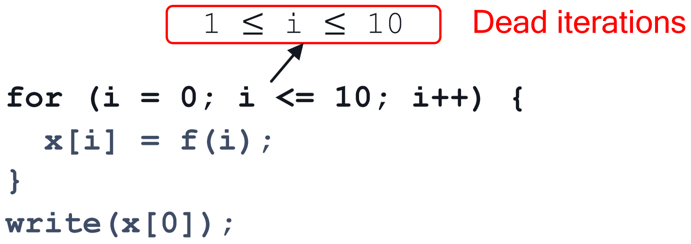


Introduction

- **Prior art: Dead Code Elimination (DCE)**
  - Classical optimization present in all compilers
  - Identification and elimination of <u>code</u> not contributing to output data
  - Increase performance and reduce executable size
  - Target complete removal of instructions



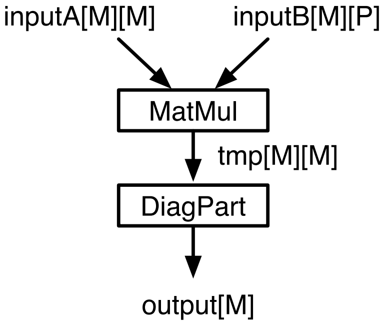
- **Our Approach: Dead Iteration Elimination (DIE)**
  - Identification and elimination of <u>loop iterations</u> not contributing to output data
  - Relevant to graph compilers compiling applications built from standard “bricks”
    - Operator composition
    - Inactivation of iterations due to, e.g., specialization/sparsification/subsampling
  - Complementary to DCE



Application Scenarios 1/4

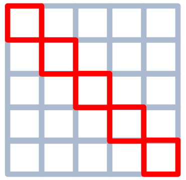
Iteration-level dead code elimination

- Remove iterations not contributing to the output (because of operator fusion, or bad programming)
- Example: MatMul-DiagPart subgraph filtering diagonal elements after a matrix multiplication




```
// MatMul operator
for (i = 0; i < M; i++) {
  for (j = 0; j < M; j++) {
    tmp[i][j] = 0.;
    for (k = 0; k < P; k++) {
      tmp[i][j] += inputA[i][k] * inputB[k][j];
    }
  }
}
// DiagPart operator
for (i = 0; i < M; i++) {
  output[i] = tmp[i][i];
}
```


```
// Fused MatMul-DiagPart operator after DIE
for (i = 0; i < M; i++) {
  temp[i][i] = 0.;
  for (k = 0; k < P; k++) {
    temp[i][i] += inputA[i][k] * inputB[k][i];
  }
}
for (i = 0; i < M; i++) {
  output[i] = temp[i][i];
}
```



Computation not necessary to the diagonal output has been filtered out

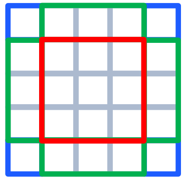
- Reduce the need for specialized custom operators in AI/DL frameworks

Application Scenarios 2/4

Tile-specialized code generation

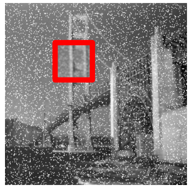
- Given a set of output tiles and a general code, generate the code computing only that set of output tiles
- Example: general mean filter specialized to a given tile

```
for (i = 0; i < height; i++) {
  for (j = 0; j < width; j++) {
    if ((i == 0) && (j == 0)) {                                                  // Case 1: top left corner
      output[i][j] = (input[i][j] + input[i][j+1] + input[i+1][j] + input[i+1][j+1]) / 4;
    }
    if ((i == 0) && (j > 0) && (j < width - 1)) {                                // Case 2: top row except corners
      output[i][j] = (input[i][j-1] + input[i][j] + input[i][j+1] + input[i+1][j-1] + input[i+1][j] + input[i+1][j+1]) / 6;
    }
    if ((i == 0) && (j == width - 1)) {                                          // Case 3: top right corner
      output[i][j] = (input[i][j-1] + input[i][j] + input[i+1][j-1] + input[i+1][j]) / 4;
    }
    if ((i > 0) && (i < height - 1) && (j == 0)) {                               // Case 4: left column except corners
      output[i][j] = (input[i-1][j] + input[i-1][j+1] + input[i][j] + input[i][j+1] + input[i+1][j] + input[i+1][j+1]) / 6;
    }
    if ((i > 0) && (i < height - 1) && (j > 0) && (j < width - 1)) {             // Case 5: out of borders, general case
      output[i][j] = (input[i-1][j-1] + input[i-1][j] + input[i-1][j+1] +
                      input[i  ][j-1] + input[i  ][j] + input[i  ][j+1] +
                      input[i+1][j-1] + input[i+1][j] + input[i+1][j+1]) / 9;
    }
    if ((i > 0) && (i < height - 1) && (j == width - 1)) {                       // Case 6: right column except corners
      output[i][j] = (input[i-1][j-1] + input[i-1][j] + input[i][j-1] + input[i][j] + input[i+1][j-1] + input[i+1][j]) / 6;
    }
    if ((i == height - 1) && (j == 0)) {                                         // Case 7: bottom left corner
      output[i][j] = (input[i-1][j] + input[i-1][j+1] + input[i][j] + input[i][j+1]) / 4;
    }
    if ((i == height - 1) && (j > 0) && (j < width - 1)) {                       // Case 8: bottom row except corners
      output[i][j] = (input[i-1][j-1] + input[i-1][j] + input[i-1][j+1] + input[i][j-1] + input[i][j] + input[i][j+1]) / 6;
    }
    if ((i == height - 1) && (j == width - 1)) {                                 // Case 9: bottom right corner
      output[i][j] = (input[i-1][j-1] + input[i-1][j] + input[i][j-1] + input[i][j]) / 4;
    }
  }
}
```




```
for (i = 64; i < 128; i++) {
  for (j = 64; j < 128; j++) {
    output[i][j] = (input[i-1][j-1] + input[i-1][j] + input[i-1][j+1] +
                    input[i  ][j-1] + input[i  ][j] + input[i  ][j+1] +
                    input[i+1][j-1] + input[i+1][j] + input[i+1][j+1]) / 9;
  }
}
```

Code specialized to the desired tile output[64..127][67..127] only



Application Scenarios 3/4

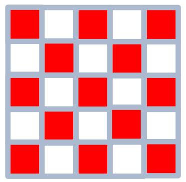
Sparsification/subsampling

- Given a dense operator and a structured sparsity information, generate the code producing only the desired data
- Example: checkerboard subsampling

```
// Roberts Edge Detection Filter
for (i = 0; i < height - 3; i++) {
  for (j = 0; j < width - 3; j++) {
    output[i+1][j+2] = abs(tmp1[i+1][j+2] - tmp1[i+2][j+1]) +
                       abs(tmp1[i+2][j+2] - tmp1[i+1][j+1]);
  }
}
```

DIE

```
for (i = 0; i < height - 3; i++) {
  for (j = 0; j < width - 3; j++) {
    if ((i + j - 3) % 2 == 0) {
      output[i+1][j+2] = abs(tmp1[i+1][j+2] - tmp1[i+2][j+1]) +
                         abs(tmp1[i+2][j+2] - tmp1[i+1][j+1]);
    }
  }
}
```



Code updated to compute only the desired data
(here, checkerboard-style: output[d1][d2] with (d1+d2)%2 == 0)

Application Scenarios 4/4

Compiler warning

- Compiler warns the user that some iterations are (missing or) not contributing to the data space
- Example: out of bound access that SOTA static checkers (e.g., cppchecker or Clang's scan-build) fail to find

```
void init(size_t size, int vector[size]) {
  for (size_t i = 0; i < size; i++) {
    vector[i + 1] = 0;
  }
}
```

DIE

```
  for (size_t i = 0; i < size; i++) {
  ^
out.c:4:3: iteration "i <- size" leads to out of bound access of vector[i + 1]
```

Non-trivial out of bound access has been detected at compilation time

Problem Statement

- **Problem: programs may include statement iterations not contributing to the desired output**
  - **Graph compilers**
    - Excess computation may be introduced by operator sequences (e.g., when some output of a given operator is not used)
    - Iteration domains of basic operators may be reduced due to specialization, sparsification, subsampling
  - **General compilers**
    - Parameters may remove some computation or output while corresponding computation (partly) remain
    - Bugs may be present in the code

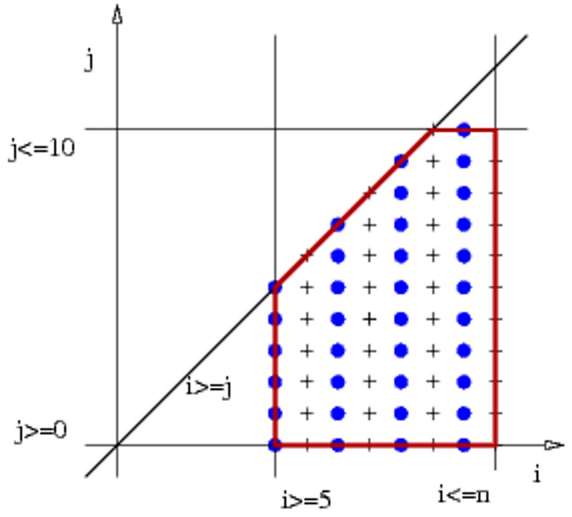
- **Relevance: applications to optimization and productivity**
  - **For computational graph compilers**
    - Enable automatic removal of excess computation not contributing to the final output, reduce the need for custom operators
    - Enable automatic specialization, sparsification and subsampling of (sequences of) operators, including custom operators
  - **For general compilers**
    - Complement dead-code elimination with dead iteration elimination
    - Extend static analysis ability to detect potential bugs, e.g., out of bound memory accesses

Solution: Polyhedral Model Background

- **Models the execution of loop nests in a vector space**
  - Loop iterations and data accesses are integer-valued points within a polyhedron
  - Analyses and transformations rely on high-level math libraries
    - ➢E.g., our algorithm relies on fusion, intersection, projection, image, preimage of polyhedra
  - Algorithms exist for many analyses and tasks
    - ➢E.g., our algorithm relies on exact iteration-wise data dependence analysis
  - Code or compiler IR can be generated from the polyhedral representation

- **A vast majority of Neural Net layers can be efficiently modeled using the polyhedral model [TACO 2019]**
  - But the technology has a broader application domain (including radar, image & signal processing, physics simulations)

```
n = f();
for (i = 5; i <= n; i += 2) {
  A[i][i] = A[i][i] / B[i];
  for (j = 0; j <= i; j++) {
    if (j <= 10) {
      ... A[i + 2*j + n][i + 3] ...
    }
  }
}
```

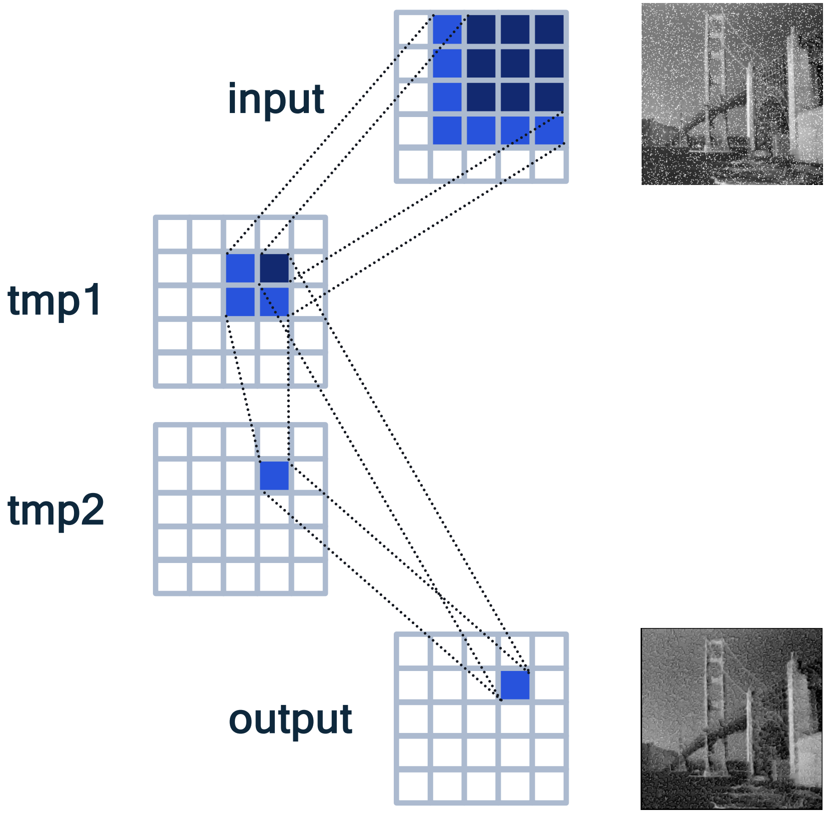


$$\{i, j \in \mathbb{Z}^2 | \exists k \in \mathbb{Z}, \\ 5 \leq i \leq n; 0 \leq j \leq i; i = 2k + 1\}$$

Solution: Introductive Example

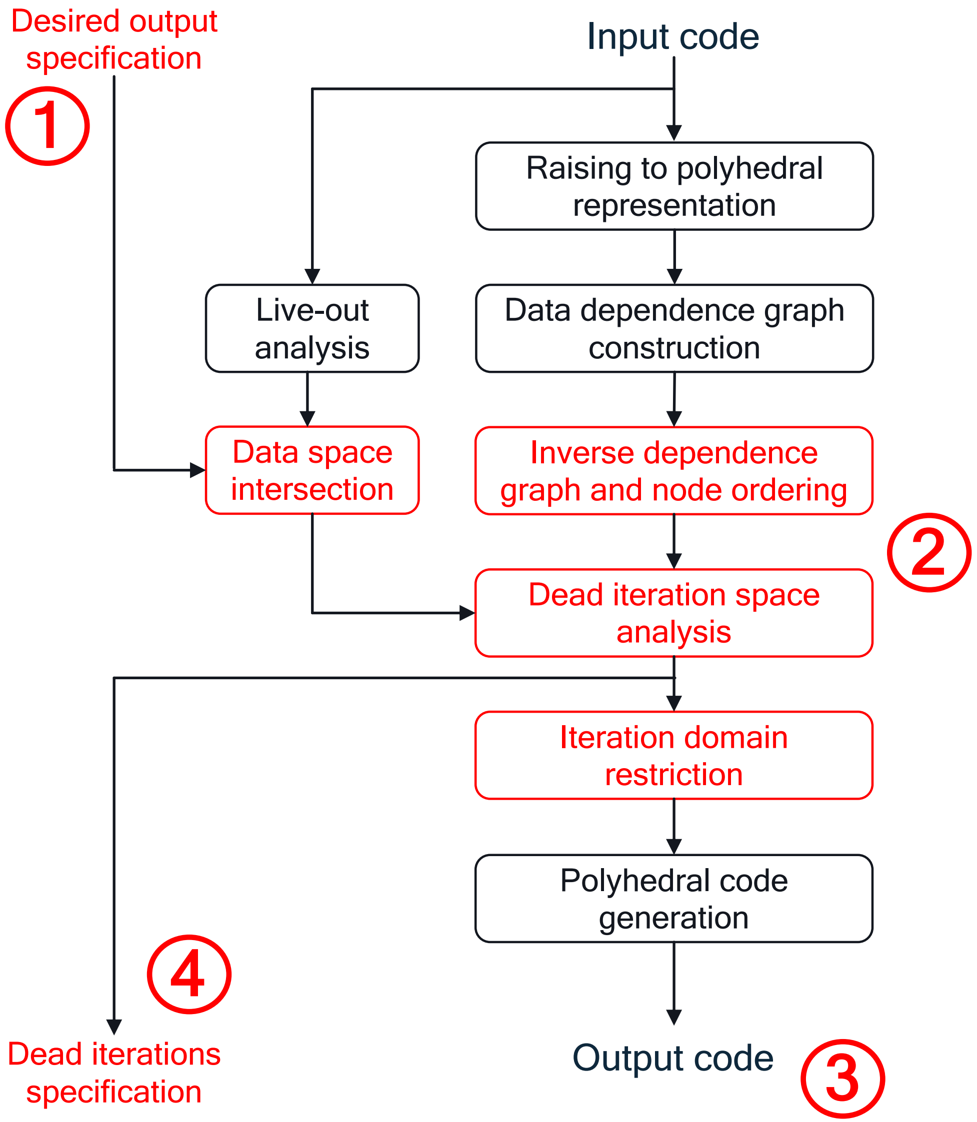
- Mean-Roberts Sharpening
  - 3-stage image processing pipeline
- To produce a given part of the final image, which data and computations are required?

```
// Mean Filter
for (i = 1; i < height - 1; i++) {
  for (j = 1; j < width - 1; j++) {
    tmp1[i][j] = (input[i-1][j-1] + input[i-1][j] + input[i-1][j+1] +
                  input[i  ][j-1] + input[i  ][j] + input[i  ][j+1] +
                  input[i+1][j-1] + input[i+1][j] + input[i+1][j+1]) / 9;
  }
}
// Roberts Edge Detection Filter
for (i = 0; i < height - 3; i++) {
  for (j = 0; j < width - 3; j++) {
    tmp2[i+1][j+2] = abs(tmp1[i+1][j+2] - tmp1[i+2][j+1]) +
                     abs(tmp1[i+2][j+2] - tmp1[i+1][j+1]);
  }
}
// Additive
for (i = 1; i < height - 2; i++) {
  for (j = 2; j < width - 1; j++) {
    output[i][j] = tmp1[i][j] - 1 * tmp2[i][j];
  }
}
```



Solution: Key Points

1. **Support for desired output data specification**
   - Enable specialization/sparsification/subsampling specification
2. **Iteration-level dead code analysis**
   - Innovative polyhedral analysis dedicated to required data-space analysis and their mapping to statement iteration domains
3. **Generation of output code cleared from dead iterations**
   - Exploit polyhedral code generation on restricted iteration domains
4. **Support for dead iteration space output**
   - Enable programmer feedback, e.g., compiler warnings



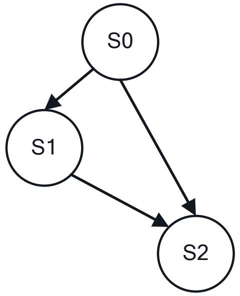
Solution: Dead Iteration Space Analysis

Principle: back-propagate data shape constraints through operations on these shapes and update iteration domains accordingly

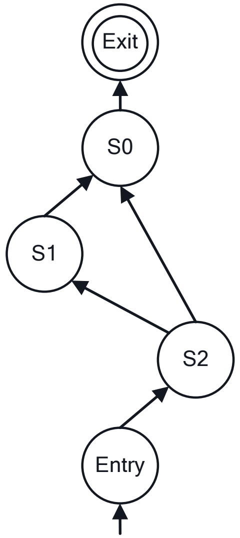
1. Build the inverted data dependence graph (inverted edges), including:
   - A new entry vertex with edges to every vertex writing the output tensors
   - A new exit vertex with edges from every vertex reading the input tensors
   - Nodes involved within a dependence cycle merged into a supernode
2. For each vertex v, initialize $R_{v,t}$ the required data space for the tensor t:
   - Entry vertex: $R_{entry,t}$ = required data space of each output tensor t
   - Other vertices: empty spaces
3. For each vertex v according to a topological ordering of the nodes (v has iteration domain D_v, writes tensor w with function f_w, and reads tensors r_i with function f_{ri})
   a. Vertex potential contribution space for the written tensor w: $P_{v,w} = \bigcup_{i \text{ in predecessor}(v)} R_{i,w}$
   b. Vertex contributing space for the written tensor w: $C_{v,w} = \text{Preimage}(P_{v,w}, f_w) \cap D_v$
   c. Required data spaces for v: $\{R_{v,ri} = \text{Image}(C_{v,w}, f_{ri})\}$
   d. Dead iteration space for v: $\text{Dead}_v = D_v - C_{v,w}$
4. For each vertex v, restrict the iteration domain:
   - $D_v = D_v - \text{Dead}_v$

```
// Mean Filter
for (i = 1; i < height - 1; i++) {
  for (j = 1; j < width - 1; j++) {
S0: tmp1[i][j] = (input[i-1][j-1] + input[i-1][j] + input[i-1][j+1] +
                  input[i  ][j-1] + input[i  ][j] + input[i  ][j+1] +
                  input[i+1][j-1] + input[i+1][j] + input[i+1][j+1]) / 9;
  }
}
// Roberts Edge Detection Filter
for (i = 0; i < height - 3; i++) {
  for (j = 0; j < width - 3; j++) {
S1: tmp2[i+1][j+2] = abs(tmp1[i+1][j+2] - tmp1[i+2][j+1]) +
                     abs(tmp1[i+2][j+2] - tmp1[i+1][j+1]);
  }
}
// Additive
for (i = 1; i < height - 2; i++) {
  for (j = 2; j < width - 1; j++) {
S2: output[i][j] = tmp1[i][j] - 1 * tmp2[i][j];
  }
}
```

Data-Dependence Graph



Inverted Dependence Graph With Entry/Exit Nodes

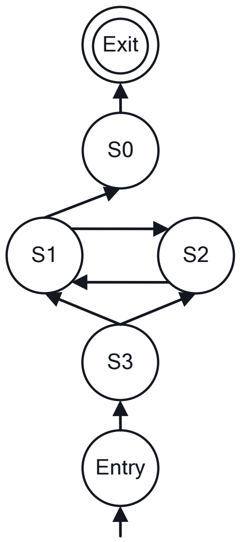


Required data space: output[LB0..UB0][LB1..UB1]
$R_{entry,output} = \{(d0,d1) \mid LB0 \leq d0 \leq UB0, LB1 \leq d1 \leq UB1\}$

Cycle Handling

- Time constrained (production compiler)
  - Use “lightweight” (complexity) set operations such as image/preimage/union/subtract
  - Only visit node once!
- Locate the strongly connected components
  - Over-approximate by requiring the node’s full iteration domain to be visited!
- Simple but very effective accuracy improvement
  - Don’t approximate self dependency cycle with distance zero
  - E.g., reduction X += A[i], or more generally X = f(X, …)

```
for (i = 0; i <= 10; i++) {
  tmp1[i] = input[i];             // S0
}
for (i = 5; i <= 10; i++) {
  tmp2[i] = tmp1[i] + tmp3[i - 5]; // S1
  tmp3[i - 4] += tmp2[i - 2];      // S2
}
for (i = 0; i <= 10; i++) {
  output[i] = tmp2[i] + tmp3[i];   // S3
}
```



Example:

- In this case with a dependence cycle, only the ordering Entry → S3 → S2 → S1 → S0 → Exit is valid, not Entry → S3 → S1 → S2 → S0 → Exit (because there is no edge to propagate information coming from S2 to S0).

Polyhedral Art

Sven Verdoolaege proposed a variant of the algorithm

- Implemented in PPCG and leveraged by Polly (LLVM) and AlphaZ

Fixed point algorithm starting from the live-out set:

1. **Apply reversed dependence map**

   Stop if no additional iterations have been added

2. **Union w/ the previous iteration set**

3. **Widening using affine hull**

4. **Intersect w/ the Scop domain**

Differences w/ Sven's Algorithm

Widening for depencency cycle handling

Most implementation of Sven's algorithm use affine hull as widening operation:

- Rather expensive to compute
- Too coarse? (at Scop level)

Our algorithm:

- Lighter; one traversal of the inverted dependance graph, one node at a time (not full scop)
- Dependence cycles are approximated (apart for self dependency)

Matmul w/ triangular output

```
for (i = 0; i < M; i++)
  for (j = 0; j < M; j++) {
    tmp[i][j] = 0.;
    for (k = 0; k < P; k++)
      tmp[i][j] += inputA[i][k] * inputB[k][j];
  }
for (i = 0; i < M; i++)
  for (j = i ; j < M; j++)
    output[i][j] = tmp[i][j];
```

```
for (i = 0; i < M; i++)
  for (j = i; j < M; j++) {
    tmp[i][j] = 0.;
    for (k = 0; k < P; k++)
      tmp[i][j] += inputA[i][k] * inputB[k][j];
  }
for (i = 0; i < M; i++)
  for (j = i; j < M; j++)
    output[i][j] = tmp[i][j];
```

Summary

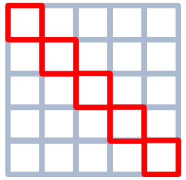
- Extracts the **iteration** and **data domains** required given an output data specification
  - For each node of the computation graph
- Applicable to a wide variety of program
  - Static Control Parts
  - + black box operations where the iteration domain and access function are approximated
- Many application scenarios
  - Code analysis and feedback/warning generation
  - Remove dead iterations from existing code base (e.g., due to fusion, composition, etc)
  - Generate code tailored to computing a subset of an original output space (tilingspecialization, etc)

Application Scenarios 1/2

- **Iteration-level dead code elimination**
  - Remove the iterations not contributing to the output (because of operator fusion, or bad programming)
  - Example: filtering out diagonal elements after a matrix multiplication

```
for (i = 0; i < M; i++) {   // Matrix multiplication kernel
  for (j = 0; j < M; j++) {
    temp[i][j] = 0.;
    for (k = 0; k < P; k++) {
      temp[i][j] += inputA[i][k] * inputB[k][j];
    }
  }
}
for (i = 0; i < M; i++) {   // Filtering of diagonal elements
  output[i] = temp[i][i];
}
```

```
// NOTE: additional analysis may entirely remove temp array
for (i = 0; i < M; i++) {
  temp[i][i] = 0.;
  for (k = 0; k < P; k++) {
    temp[i][i] += inputA[i][k] * inputB[k][i];
  }
  output[i] = temp[i][i];
}
```

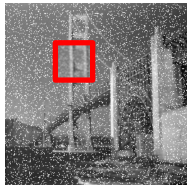


Computation not necessary to compute the diagonal has been filtered out

- **Tile-specialized code generation**
  - Given a set of output tiles and a general code, generate the code that computes only that set of output tiles
  - Example: general mean filter specialized to a given tile

```
for (i = 0; i < height; i++) {
  for (j = 0; j < width; j++) {
    if ((i == 0) && (j == 0)) {                                            // Case 1: top left corner
      output[i][j] = (input[i][j] + input[i][j+1] + input[i+1][j] + input[i+1][j+1]) / 4;
    }
    if ((i == 0) && (j > 0) && (j < width - 1)) {                          // Case 2: top row except corners
      output[i][j] = (input[i][j-1] + input[i][j] + input[i][j+1] + input[i+1][j-1] + input[i+1][j] + input[i+1][j+1]) / 6;
    }
    if ((i == 0) && (j == width - 1)) {                                    // Case 3: top right corner
      output[i][j] = (input[i][j-1] + input[i][j] + input[i+1][j-1] + input[i+1][j]) / 4;
    }
    if ((i > 0) && (i < height - 1) && (j == 0)) {                         // Case 4: left column except corners
      output[i][j] = (input[i-1][j] + input[i-1][j+1] + input[i][j] + input[i][j+1] + input[i+1][j] + input[i+1][j+1]) / 6;
    }
    if ((i > 0) && (i < height - 1) && (j > 0) && (j < width - 1)) {       // Case 5: out of borders, general case
      output[i][j] = (input[i-1][j-1] + input[i-1][j] + input[i-1][j+1] +
                      input[i  ][j-1] + input[i  ][j] + input[i  ][j+1] +
                      input[i+1][j-1] + input[i+1][j] + input[i+1][j+1]) / 9;
    }
    if ((i > 0) && (i < height - 1) && (j == width - 1)) {                 // Case 6: right column except corners
      output[i][j] = (input[i-1][j-1] + input[i-1][j] + input[i][j-1] + input[i][j] + input[i+1][j-1] + input[i+1][j]) / 6;
    }
    if ((i == height - 1) && (j == 0)) {                                   // Case 7: bottom left corner
      output[i][j] = (input[i-1][j] + input[i-1][j+1] + input[i][j] + input[i][j+1]) / 4;
    }
    if ((i == height - 1) && (j > 0) && (j < width - 1)) {                 // Case 8: bottom row except corners
      output[i][j] = (input[i-1][j-1] + input[i-1][j] + input[i-1][j+1] + input[i][j-1] + input[i][j] + input[i][j+1]) / 6;
    }
    if ((i == height - 1) && (j == width - 1)) {                           // Case 9: bottom right corner
      output[i][j] = (input[i-1][j-1] + input[i-1][j] + input[i][j-1] + input[i][j]) / 4;
    }
  }
}
```

```
for (i = 64; i < 128; i++) {
  for (j = 64; j < 128; j++) {
    output[i][j] = (input[i-1][j-1] + input[i-1][j] + input[i-1][j+1] +
                    input[i  ][j-1] + input[i  ][j] + input[i  ][j+1] +
                    input[i+1][j-1] + input[i+1][j] + input[i+1][j+1]) / 9;
  }
}
```



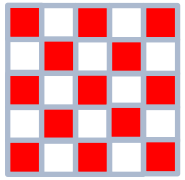
Code has been specialized to the desired tile only

Application Scenarios 2/2

- **Sparsification/subsampling**
  - Given a dense operator and a structured sparsity information, generate the code producing only the desired data
  - Example: checkerboard subsampling

```c
// Roberts Edge Detection Filter
for (i = 0; i < height - 3; i++) {
  for (j = 0; j < width - 3; j++) {
    output[i+1][j+2] = abs(tmp1[i+1][j+2] - tmp1[i+2][j+1]) +
                       abs(tmp1[i+2][j+2] - tmp1[i+1][j+1]);
  }
}
```

```c
// NOTE: additional analysis may output on subsampled matrix
for (i = 0; i < height - 3; i++) {
  for (j = 0; j < width - 3; j++) {
    if ((i + j - 3) % 2 == 0) {
      output[i+1][j+2] = abs(tmp1[i+1][j+2] - tmp1[i+2][j+1]) +
                         abs(tmp1[i+2][j+2] - tmp1[i+1][j+1]);
    }
  }
}
```



Code updated to compute only the desired data (here, checkerboard-style)

- **Compiler warning**
  - Compiler warns the user that some iterations are (missing or) not contributing to the data space
  - Example: out of bound access that advanced static checkers (e.g., cppchecker or Clang’s scan-build) are failing to find

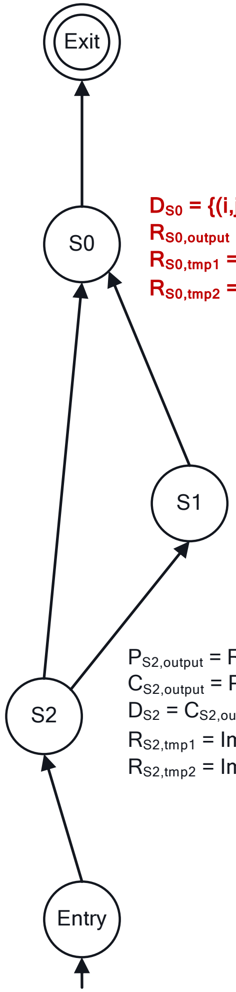
```c
void init(size_t size, int vector[size]) {
  for (size_t i = 0; i < size; i++) {
    vector[i + 1] = 0;
  }
}
```

```
  for (size_t i = 0; i < size; i++) {
  ^
out.c:4:3: iteration "i <- size" leads to out of bound access of vector[i + 1]
```

Non-trivial out of bound access has been detected at compilation time

DIE General Algorithm

Inverted Dependence Graph
With Entry/Exit Nodes



$D_{S0} = \{(i,j) \mid 1 \le i \le height, 1 \le j \le width\}$
$R_{S0,output} = \emptyset$
$R_{S0,tmp1} = \emptyset$
$R_{S0,tmp2} = \emptyset$

$P_{S1,tmp2} = R_{S2,tmp2} = \{(d_0, d_1, i_2, j_2) \mid d_0 = i_2 \;\&\&\; d_1 = j_2 \;\&\&\; 1,LB0 \le i_2 \le height - 1,UB0 \;\&\&\; 2, LB1 \le j_2 \le width, UB1\}$
$C_{S1,tmp2} = Preimage(P_{S1,tmp2}, (d_0 = i_1 + 1, d_1 = j_1 + 2)) \cap D_{S1} = \{(i_1, i_2, j_1, j_2) \mid i_1 = i_2 \;\&\&\; j_1 = j_2 \;\&\&\; 1,LB0 \le i_2 \le height - 1,UB0 \;\&\&\; 2, LB1 \le j_2 \le width, UB1\} \cap \{(i,j) \mid 0 \le i \le height - 2, 0 \le j \le width - 2\}$
$= \{(i_1, i_2, j_1, j_2) \mid i_1 = i_2 \;\&\&\; j_1 = j_2 \;\&\&\; 1,LB0 \le i_2 \le height - 2,UB0 \;\&\&\; 2, LB1 \le j_2 \le width - 2, UB1\}$
$D_{S2} = C_{S2,output} = \{(i_2, j_2) \mid 1,LB0 \le i_2 \le height - 1,UB0 \;\&\&\; 2, LB1 \le j_2 \le width, UB1\}$
$R_{S2,tmp1} = Image(C_{S2,output}, (d_0 = i_2, d_1 = j_2)) = \{(d_0, d_1, i_2, j_2) \mid d_0 = i_2 \;\&\&\; d_1 = j_2 \;\&\&\; 1,LB0 \le i_2 \le height - 1,UB0 \;\&\&\; 2, LB1 \le j_2 \le width, UB1\}$
$R_{S2,tmp2} = Image(C_{S2,output}, (d_0 = i_2, d_1 = j_2)) = \{(d_0, d_1, i_2, j_2) \mid d_0 = i_2 \;\&\&\; d_1 = j_2 \;\&\&\; 1,LB0 \le i_2 \le height - 1,UB0 \;\&\&\; 2, LB1 \le j_2 \le width, UB1\}$

$D_{S1} = \{(i,j) \mid 0 \le i \le height - 2, 0 \le j \le width - 2\}$
$R_{S1,output} = \emptyset$
$R_{S2,tmp1} = \emptyset$
$R_{S1,tmp2} = \emptyset$

$P_{S2,output} = R_{entry,output} = \{(d_0, d_1) \mid LB0 \le d_0 \le UB0 \;\&\&\; LB1 \le d_1 \le UB1\}$
$C_{S2,output} = Preimage(P_{S2,output}, (d_0 = i_2, d_1 = j_2)) \cap D_{S2} = \{(i_2, j_2) \mid LB0 \le i_2 \le UB0 \;\&\&\; LB1 \le j_2 \le UB1\} \cap \{(i_2,j_2) \mid 1 \le i_2 \le height - 1 \;\&\&\; 2 \le j_2 \le width\} = \{(i_2, j_2) \mid 1,LB0 \le i_2 \le height - 1,UB0 \;\&\&\; 2, LB1 \le j_2 \le width, UB1\}$
$D_{S2} = C_{S2,output} = \{(i_2, j_2) \mid 1,LB0 \le i_2 \le height - 1,UB0 \;\&\&\; 2, LB1 \le j_2 \le width, UB1\}$
$R_{S2,tmp1} = Image(C_{S2,output}, (d_0 = i_2, d_1 = j_2)) = \{(d_0, d_1, i_2, j_2) \mid d_0 = i_2 \;\&\&\; d_1 = j_2 \;\&\&\; 1,LB0 \le i_2 \le height - 1,UB0 \;\&\&\; 2, LB1 \le j_2 \le width, UB1\}$
$R_{S2,tmp2} = Image(C_{S2,output}, (d_0 = i_2, d_1 = j_2)) = \{(d_0, d_1, i_2, j_2) \mid d_0 = i_2 \;\&\&\; d_1 = j_2 \;\&\&\; 1,LB0 \le i_2 \le height - 1,UB0 \;\&\&\; 2, LB1 \le j_2 \le width, UB1\}$

Required data space: output[LB0..UB0][LB1..UB1]
$R_{entry,output} = \{(d0,d1) \mid LB0 \le d0 \le UB0, LB1 \le d1 \le UB1\}$

```
// Mean Filter
for (i = 1; i < height - 1; i++) {
  for (j = 1; j < width - 1; j++) {
S0: tmp1[i][j] = (input[i-1][j-1] + input[i-1][j] + input[i-1][j+1] +
                  input[i  ][j-1] + input[i  ][j] + input[i  ][j+1] +
                  input[i+1][j-1] + input[i+1][j] + input[i+1][j+1]) / 9;
  }
}
// Roberts Edge Detection Filter
for (i = 0; i < height - 3; i++) {
  for (j = 0; j < width - 3; j++) {
S1: tmp2[i+1][j+2] = abs(tmp1[i+1][j+2] - tmp1[i+2][j+1]) +
                     abs(tmp1[i+2][j+2] - tmp1[i+1][j+1]);
  }
}
// Additive
for (i = 1; i < height - 2; i++) {
  for (j = 2; j < width - 1; j++) {
S2: output[i][j] = tmp1[i][j] - 1 * tmp2[i][j];
  }
}
```

Problem Statement

Remove statement iterations not contributing to a computation output

- **Provide analyses and removal techniques to identify and eliminate non-useful statement iterations from a program**
  - Input is a computational kernel or function code
  - Output is the identification of non-useful statement iterations and/or a semantically equivalent code cleared from those iterations
- **Definitions**
  - Statement iterations:
    - Subset of the executions of a statement enclosed within an iterative loop
    - Contributing statement iterations are those writing to a memory location which is part of the output
  - Computation output:
    - Live-out data space of a computational kernel or a function as analyzed by a compiler
    - Or, output-data space (e.g., output tile, sparsified/subsampled output) as specified by the user
- **Application domain**
  - Ideal application domain is known as “static control codes”
    - Loop bounds and conditionals are affine expressions of outer loop counters and constant parameters
    - Tensor access functions are affine expressions of outer loop counters and constant parameters
  - Not respecting the above conditions is not blocking but may limit non-contributing statement iteration removal
  - Typical target codes are AI/DL operators or scientific computation kernels