SIGGRAPH Asia 2019

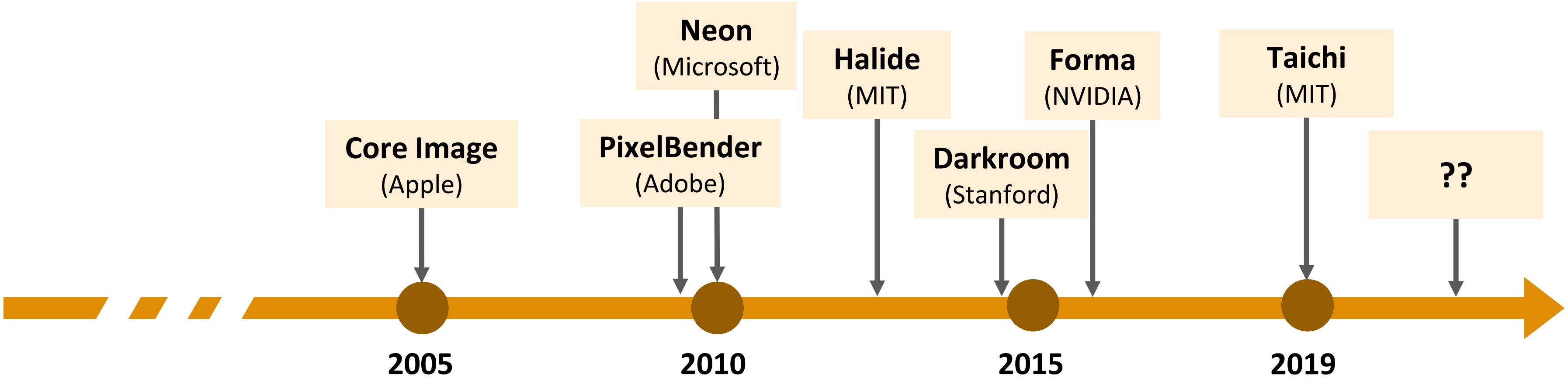
Automatically Translating Image Processing Libraries to Halide

Maaz Bin Safeer Ahmad
(University of Washington)

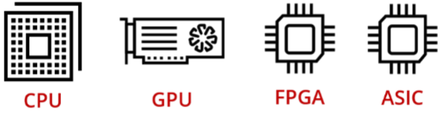
Jonathan Ragan-Kelley & Alvin Cheung
(UC Berkeley)

Shoaib Kamil
(Adobe Research)

Domain Specific Languages



High Performance



Portability

Algorithm + Schedule + Data Structure

Maintainability

Ps

OpenCV

Legacy C++ Implementation

```
void blur(Buffer<uint16_t> in, Buffer<uint16_t> out) {
  __m128i one_third = _mm_set1_epi16(21846);
#pragma omp parallel for
  for (int yTile = 0; yTile < out.height(); yTile += 32) {
    __m128i tmp[(128/8) * (32 + 2)];
    for (int xTile = 0; xTile < out.width(); xTile += 128) {
      __m128i *tmpPtr = tmp;
      for (int y = 0; y < 32+2; y++) {
        const uint16_t *inPtr = &(in(xTile, yTile+y));
        for (int x = 0; x < 128; x += 8) {
          __m128i a = _mm_load_si128((const __m128i*)(inPtr));
          __m128i b = _mm_loadu_si128((const __m128i*)(inPtr+1));
          __m128i c = _mm_loadu_si128((const __m128i*)(inPtr+2));
          __m128i sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
          __m128i avg = _mm_mulhi_epi16(sum, one_third);
          _mm_store_si128(tmpPtr++, avg);
          inPtr+=8;
        }
      }
      tmpPtr = tmp;
      for (int y = 0; y < 32; y++) {
        __m128i *outPtr = (__m128i *)(&(out(xTile, yTile+y)));
        for (int x = 0; x < 128; x += 8) {
          __m128i a = _mm_load_si128(tmpPtr+(2*128)/8);
          __m128i b = _mm_load_si128(tmpPtr+128/8);
          __m128i c = _mm_load_si128(tmpPtr++);
          __m128i sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
         __m128i avg = _mm_mulhi_epi16(sum, one_third);
         _mm_store_si128(outPtr++, avg);
        }
   }}}
}
```

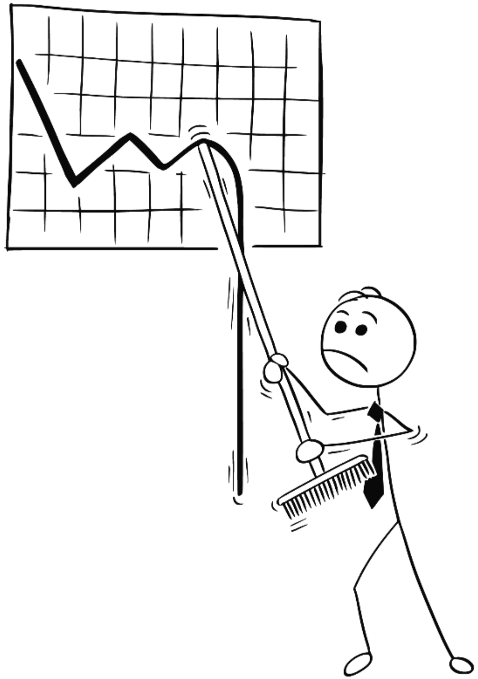
Legacy C++ Implementation

Tiles of 32 x 128

```
void blur(Buf                              nt16_t> out) {
  __m128i one_thir                    pi16(2
#pragma omp parallel for
  for (int yTile = 0; yTile < out.height(); yTile += 32) {
    __m128i tmp[(128/8) * (32 + 2)];
    for (int xTile = 0; xTile < out.width(); xTile += 128) {
      __m128i *tmpPtr = tmp;
      for (int y = 0; y < 32+2; y++) {
        const uint16_t *inPtr = &(in(xTile, yTile+y));
        for (int x = 0; x < 128; x += 8) {
          __m128i a = _mm_load_si128((const __m128i*)(inPtr));
          __m128i b = _mm_loadu_si128((const __m128i*)(inPtr+1));
          __m128i c = _mm_loadu_si128((const __m128i*)(inPtr+2));
          __m128i sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
          __m128i avg = _mm_mulhi_epi16(sum, one_third);
          _mm_store_si128(tmpPtr++, avg);
          inPtr+=8;
        }
      }
      tmpPtr = tmp;
      for (int y = 0; y < 32; y++) {
        __m128i *outPtr = (__m128i *)(&(out(xTile, yTile+y))
        for (int x = 0; x < 128; x += 8) {
          __m128i a = _mm_load_si128(tmpPtr+(2*128)/8);
          __m128i b = _mm_load_si128(tmpPtr+128/8);
          __m128i c = _mm_load_si128(tmpPtr++);
          __m128i sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
         __m128i avg = _mm_mulhi_epi16(sum, one_third);
         _mm_store_si128(outPtr++, avg);
        }
   }}}
}
```

SSE2 Instructions

Performance & Portability Deteriorate Over-time

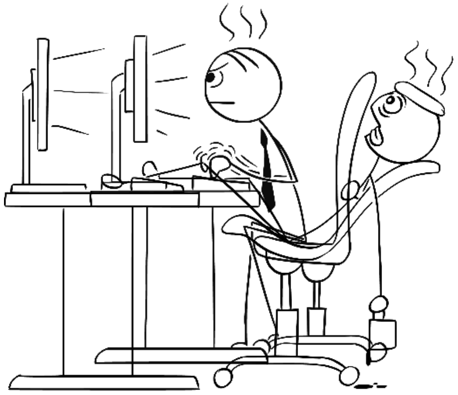


Legacy C++ Implementation

```
void blur(Buffer<uint16_t> in, Buffer<uint16_t> out) {
  __m128i one_third = _mm_set1_epi16(21846);
#pragma omp parallel for
  for (int yTile = 0; yTile < out.height(); yTile += 32) {
    __m128i tmp[(128/8) * (32 + 2)];
    for (int xTile = 0; xTile < out.width(); xTile += 128) {
      __m128i *tmpPtr = tmp;
      for (int y = 0; y < 32+2; y++) {
        const uint16_t *inPtr = &(in(xTile, yTile+y));
        for (int x = 0; x < 128; x += 8) {
          __m128i a = _mm_load_si128((const __m1 8i*)(inPtr));
          __m128i b = _mm_loadu_si128((con 8i*)(inPtr+1));
          __m128i c = _mm_loadu_si128( i*)(inPtr+2));
          __m128i sum = _mm_add a, b), c);
          __m128i avg = _m _third);
          _mm_store_s
          inPtr
        }
      }
      tmpPt
      for (i y = 0; y < 32; y++) {
        __m128i *outPtr = (__m128i *)(&(out(xTile, yTile+y)));
        for (int x = 0; x < 128; x += 8) {
          __m128i a = _mm_load_si128(tmpPtr+(2*128)/8);
          __m128i b = _mm_load_si128(tmpPtr+128/8);
          __m128i c = _mm_load_si128(tmpPtr++);
          __m128i sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
         __m128i avg = _mm_mulhi_epi16(sum, one_third);
         _mm_store_si128(outPtr++, avg);
        }
    }}}
}
```

3x3 Box Blur

Rewrite??



- Large Code Bases
- Obfuscated Code
- Requires Expertise
- Risk Introducing Bugs

Legacy C++ Implementation

```
void blur(Buffer<uint16_t> in, Buffer<uint16_t> out) {
  __m128i one_third = _mm_set1_epi16(21846);
#pragma omp parallel for
  for (int yTile = 0; yTile < out.height(); yTile += 32) {
    __m128i tmp[(128/8) * (32 + 2)];
    for (int xTile = 0; xTile < out.width(); xTile += 128) {
      __m128i *tmpPtr = tmp;
      for (int y = 0; y < 32+2; y++) {
        const uint16_t *inPtr = &(in(xTile, yTile+y));
        for (int x = 0; x < 128; x += 8) {
          __m128i a = _mm_load_si128((const __m128i*)(inPtr));
          __m128i b = _mm_loadu_si128((const __m128i*)(inPtr+1));
          __m128i c = _mm_loadu_si128((const __m128i*)(inPtr+2));
          __m128i sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
          __m128i avg = _mm_mulhi_epi16(sum, one_third);
          _mm_store_si128(tmpPtr++, avg);
          inPtr+=8;
        }
      }
      tmpPtr = tmp;
      for (int y = 0; y < 32; y++) {
        __m128i *outPtr = (__m128i *)(&(out(xTile, yTile+y)));
        for (int x = 0; x < 128; x += 8) {
          __m128i a = _mm_load_si128(tmpPtr+(2*128)/8);
          __m128i b = _mm_load_si128(tmpPtr+128/8);
          __m128i c = _mm_load_si128(tmpPtr++);
          __m128i sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
          __m128i avg = _mm_mulhi_epi16(sum, one_third);
          _mm_store_si128(outPtr++, avg);
        }
   }}}
}
```

Program Synthesis & Verification


```
Func blur(Func in) {
  Func temp, out;
  Var x, y;

  temp(x,y) = (in(x-1,y)+in(x,y)+in(x+1,y))/3;
  out(x,y) = (temp(x,y-1)+temp(x,y)+temp(x,y+1))/3;

  return out;
}
```

Auto Schedule

Equivalent Halide Algorithm

Program Synthesis

{ C++ Implementation }

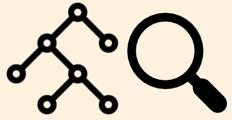
```cpp
void blur(Buffer<uint16_t> in, Buffer<uint16_t> out) {
  __m128i one_third = _mm_set1_epi16(21846);
#pragma omp parallel for
  for (int yTile = 0; yTile < out.height(); yTile += 32) {
    __m128i tmp[(128/8) * (32 + 2)];
    for (int xTile = 0; xTile < out.width(); xTile += 128) {
      __m128i *tmpPtr = tmp;
      for (int y = 0; y < 32+2; y++) {
        const uint16_t *inPtr = &(in(xTile, yTile+y));
        for (int x = 0; x < 128; x += 8) {
          __m128i a = _mm_load_si128((const __m128i*)(inPtr));
          __m128i b = _mm_loadu_si128((const __m128i*)(inPtr+1));
          __m128i c = _mm_loadu_si128((const __m128i*)(inPtr+2));
          __m128i sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
          __m128i avg = _mm_mulhi_epi16(sum, one_third);
          _mm_store_si128(tmpPtr++, avg);
          inPtr+=8;
        }
      }
      tmpPtr = tmp;
      for (int y = 0; y < 32; y++) {
        __m128i *outPtr = (__m128i *)(&(out(xTile, yTile+y)));
        for (int x = 0; x < 128; x += 8) {
          __m128i a = _mm_load_si128(tmpPtr+(2*128)/8);
          __m128i b = _mm_load_si128(tmpPtr+128/8);
          __m128i c = _mm_load_si128(tmpPtr++);
          __m128i sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
              __m128i avg = _mm_mulhi_epi16(sum, one_third);
              _mm_store_si128(outPtr++, avg);
        }
    }}}
}
```

Dexter' Search Process

Equivalence Checker

$$\forall inp \in inputs.$$
$$func_{cpp}(inp) == func_{halide}(inp)$$

Search Algorithm



Search Space

$$E \rightarrow D + D$$
$$D \rightarrow F * F$$
$$F \rightarrow var \mid const$$

{ Halide Algorithm }

```cpp
Func blur(Func in) {
  Func temp, out;
  Var x, y;

  temp(x,y) = (in(x-1,y)+in(x,y)+in(x+1,y))/3;
  out(x,y) = (temp(x,y-1)+temp(x,y)+temp(x,y+1))/3;

  return out;
}
```

Program Synthesis

{ C++ Implementation }

```
void blur(Buffer<uint16_t> in, Buffer<uint16_t> out) {
  __m128i one_third = _mm_set1_epi16(21846);
#pragma omp parallel for
  for (int yTile = 0; yTile < out.height(); yTile += 32) {
    __m128i tmp[(128/8) * (32 + 2)];
    for (int xTile = 0; xTile < out.width(); xTile += 128) {
      __m128i *tmpPtr = tmp;
      for (int y = 0; y < 32+2; y++) {
        const uint16_t *inPtr = &(in(xTile, yTile+y));
        for (int x = 0; x < 128; x += 8) {
          __m128i a = _mm_load_si128((const __m128i*)(inPtr));
          __m128i b = _mm_loadu_si128((const __m128i*)(inPtr+1));
          __m128i c = _mm_loadu_si128((const __m128i*)(inPtr+2));
          __m128i sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
          __m128i avg = _mm_mulhi_epi16(sum, one_third);
          _mm_store_si128(tmpPtr++, avg);
          inPtr+=8;
        }
      }
      tmpPtr = tmp;
      for (int y = 0; y < 32; y++) {
        __m128i *outPtr = (__m128i *)(&(out(xTile, yTile+y)));
        for (int x = 0; x < 128; x += 8) {
          __m128i a = _mm_load_si128(tmpPtr+(2*128)/8);
          __m128i b = _mm_load_si128(tmpPtr+128/8);
          __m128i c = _mm_load_si128(tmpPtr++);
          __m128i sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
              __m128i avg = _mm_mulhi_epi16(sum, one_third);
              _mm_store_si128(outPtr++, avg);
        }
    }}}
}
```

Dexter' Search Process

Equivalence Checker

$$\forall inp \in inputs.\quad func_{cpp}(inp) == func_{halide}(inp)$$

Search Algorithm

Search Space

$$E \rightarrow D + D$$
$$D \rightarrow F * F$$
$$F \rightarrow var \mid const$$

Large set of candidate algorithms

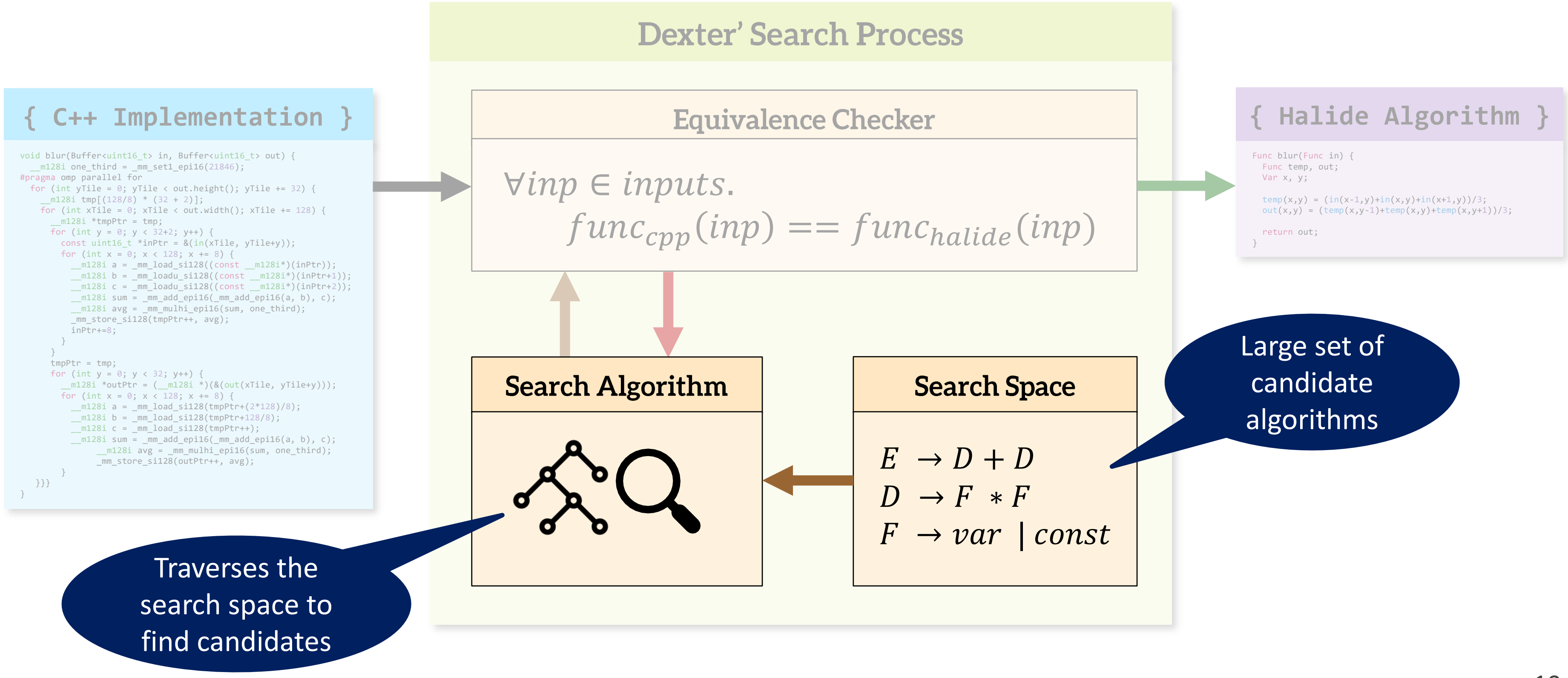
{ Halide Algorithm }

```
Func blur(Func in) {
  Func temp, out;
  Var x, y;

  temp(x,y) = (in(x-1,y)+in(x,y)+in(x+1,y))/3;
  out(x,y) = (temp(x,y-1)+temp(x,y)+temp(x,y+1))/3;

  return out;
}
```

Program Synthesis



Program Synthesis

Ensures semantic equality between original and candidate

{ C++ Implementation }

```
void blur(Buffer<uint16_t> in, Buffer<uint16_t> out) {
  __m128i one_third = _mm_set1_epi16(21846);
#pragma omp parallel for
  for (int yTile = 0; yTile < out.height(); yTile += 32) {
    __m128i tmp[(128/8) * (32 + 2)];
    for (int xTile = 0; xTile < out.width(); xTile += 128) {
      __m128i *tmpPtr = tmp;
      for (int y = 0; y < 32+2; y++) {
        const uint16_t *inPtr = &(in(xTile, yTile+y));
        for (int x = 0; x < 128; x += 8) {
          __m128i a = _mm_load_si128((const __m128i*)(inPtr));
          __m128i b = _mm_loadu_si128((const __m128i*)(inPtr+1));
          __m128i c = _mm_loadu_si128((const __m128i*)(inPtr+2));
          __m128i sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
          __m128i avg = _mm_mulhi_epi16(sum, one_third);
          _mm_store_si128(tmpPtr++, avg);
          inPtr+=8;
        }
      }
      tmpPtr = tmp;
      for (int y = 0; y < 32; y++) {
        __m128i *outPtr = (__m128i *)(&(out(xTile, yTile+y)));
        for (int x = 0; x < 128; x += 8) {
          __m128i a = _mm_load_si128(tmpPtr+(2*128)/8);
          __m128i b = _mm_load_si128(tmpPtr+128/8);
          __m128i c = _mm_load_si128(tmpPtr++);
          __m128i sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
              __m128i avg = _mm_mulhi_epi16(sum, one_third);
              _mm_store_si128(outPtr++, avg);
        }
    }}}
}
```

Dexter' Search Process

Equivalence Checker

$$\forall inp \in inputs.$$
$$func_{cpp}(inp) == func_{halide}(inp)$$

Search Algorithm

Search Space

$$E \rightarrow D + D$$
$$D \rightarrow F * F$$
$$F \rightarrow var \mid const$$

{ Halide Algorithm }

```
Func blur(Func in) {
  Func temp, out;
  Var x, y;

  temp(x,y) = (in(x-1,y)+in(x,y)+in(x+1,y))/3;
  out(x,y) = (temp(x,y-1)+temp(x,y)+temp(x,y+1))/3;

  return out;
}
```

Large set of candidate algorithms

Traverses the search space to find candidates

Program Synthesis

Ensures semantic equality between original and candidate

{ C++ Implementation }

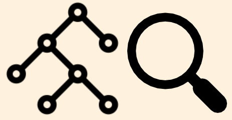
```cpp
void blur(Buffer<uint16_t> in, Buffer<uint16_t> out) {
  __m128i one_third = _mm_set1_epi16(21846);
#pragma omp parallel for
  for (int yTile = 0; yTile < out.height(); yTile += 32) {
    __m128i tmp[(128/8) * (32 + 2)];
    for (int xTile = 0; xTile < out.width(); xTile += 128) {
      __m128i *tmpPtr = tmp;
      for (int y = 0; y < 32+2; y++) {
        const uint16_t *inPtr = &(in(xTile, yTile+y));
        for (int x = 0; x < 128; x += 8) {
          __m128i a = _mm_load_si128((const __m128i*)(inPtr));
          __m128i b = _mm_loadu_si128((const __m128i*)(inPtr+1));
          __m128i c = _mm_loadu_si128((const __m128i*)(inPtr+2));
          __m128i sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
          __m128i avg = _mm_mulhi_epi16(sum, one_third);
          _mm_store_si128(tmpPtr++, avg);
          inPtr+=8;
        }
      }
      tmpPtr = tmp;
      for (int y = 0; y < 32; y++) {
        __m128i *outPtr = (__m128i *)(&(out(xTile, yTile+y)));
        for (int x = 0; x < 128; x += 8) {
          __m128i a = _mm_load_si128(tmpPtr+(2*128)/8);
          __m128i b = _mm_load_si128(tmpPtr+128/8);
          __m128i c = _mm_load_si128(tmpPtr++);
          __m128i sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
               __m128i avg = _mm_mulhi_epi16(sum, one_third);
               _mm_store_si128(outPtr++, avg);
        }
    }}}
}
```

Dexter' Search Process

Equivalence Checker

$$\forall inp \in inputs. \quad func_{cpp}(inp) == func_{halide}(inp)$$

Search Algorithm



Search Space

$$\begin{aligned} E &\rightarrow D + D \\ D &\rightarrow F * F \\ F &\rightarrow var \mid const \end{aligned}$$

Large set of candidate algorithms

Traverses the search space to find candidates

{ Halide Algorithm }

```
Func blur(Func in) {
  Func temp, out;
  Var x, y;

  temp(x,y) = (in(x-1,y)+in(x,y)+in(x+1,y))/3;
  out(x,y) = (temp(x,y-1)+temp(x,y)+temp(x,y+1))/3;

  return out;
}
```

Program Synthesis

Ensures semantic equality between original and candidate

{ C++ Implementation }

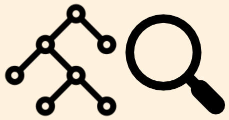
```
void blur(Buffer<uint16_t> in, Buffer<uint16_t> out) {
  __m128i one_third = _mm_set1_epi16(21846);
#pragma omp parallel for
  for (int yTile = 0; yTile < out.height(); yTile += 32) {
    __m128i tmp[(128/8) * (32 + 2)];
    for (int xTile = 0; xTile < out.width(); xTile += 128) {
      __m128i *tmpPtr = tmp;
      for (int y = 0; y < 32+2; y++) {
        const uint16_t *inPtr = &(in(xTile, yTile+y));
        for (int x = 0; x < 128; x += 8) {
          __m128i a = _mm_load_si128((const __m128i*)(inPtr));
          __m128i b = _mm_loadu_si128((const __m128i*)(inPtr+1));
          __m128i c = _mm_loadu_si128((const __m128i*)(inPtr+2));
          __m128i sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
          __m128i avg = _mm_mulhi_epi16(sum, one_third);
          _mm_store_si128(tmpPtr++, avg);
          inPtr+=8;
        }
      }
      tmpPtr = tmp;
      for (int y = 0; y < 32; y++) {
        __m128i *outPtr = (__m128i *)(&(out(xTile, yTile+y)));
        for (int x = 0; x < 128; x += 8) {
          __m128i a = _mm_load_si128(tmpPtr+(2*128)/8);
          __m128i b = _mm_load_si128(tmpPtr+128/8);
          __m128i c = _mm_load_si128(tmpPtr++);
          __m128i sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
              __m128i avg = _mm_mulhi_epi16(sum, one_third);
              _mm_store_si128(outPtr++, avg);
        }
    }}}
}
```

Dexter' Search Process

Equivalence Checker

$$\forall inp \in inputs.\quad func_{cpp}(inp) == func_{halide}(inp)$$

Search Algorithm



Search Space

$$\begin{aligned} E &\rightarrow D + D \\ D &\rightarrow F * F \\ F &\rightarrow var \mid const \end{aligned}$$

{ Halide Algorithm }

```
Func blur(Func in) {
  Func temp, out;
  Var x, y;

  temp(x,y) = (in(x-1,y)+in(x,y)+in(x+1,y))/3;
  out(x,y) = (temp(x,y-1)+temp(x,y)+temp(x,y+1))/3;

  return out;
}
```

Large set of candidate algorithms

Traverses the search space to find candidates

Search Space

$$
\begin{aligned}
Expr &:= terms \mid iden \mid Expr\ BOp\ Expr \mid UOp\ Expr \\
&\mid (Expr\ ?\ Expr\ :\ Expr) \mid f(Expr, \ldots) \\
&\mid cast{<}Type{>}(Expr) \\
Type &:= float \mid uint8\_t \mid int8\_t \mid uint16\_t \mid \ldots \\
BOps &:= + \mid - \mid * \mid / \mid << \mid \& \mid != \mid \ldots \\
UOps &:= \sim \mid - \mid !
\end{aligned}
$$

Grammar of Halide Expressions

1D, 2D and 3D Operations	Pixel Transforms, Convolutions, Gather Ops	Boundary Conditions	Arithmetic Ops, Type-casts, Conditionals

Equivalence Verification

{ C++ Implementation } ≡ { Halide Candidate }

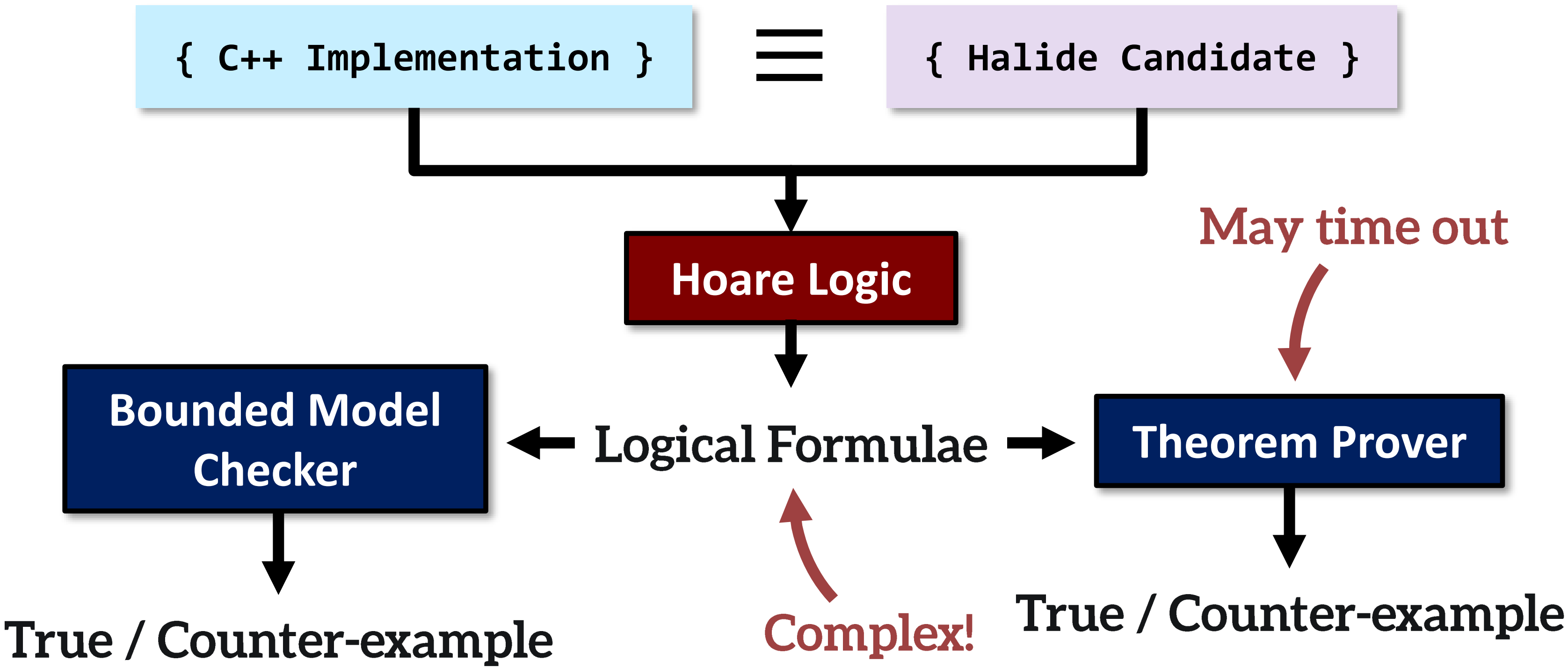
Hoare Logic (Hoare 1969)

An Axiomatic Basis for Computer Programming

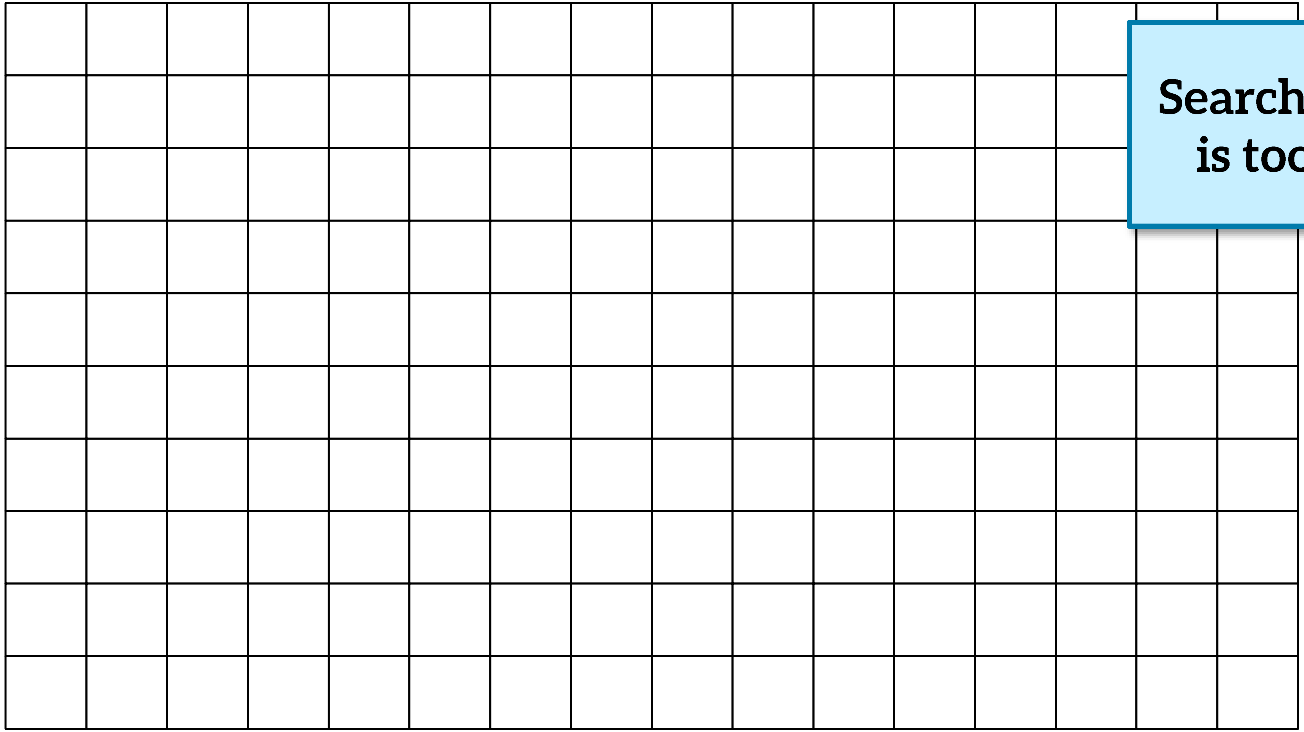
C. A. R. Hoare

The Queen's University of Belfast, Northern Ireland*

Equivalence Verification

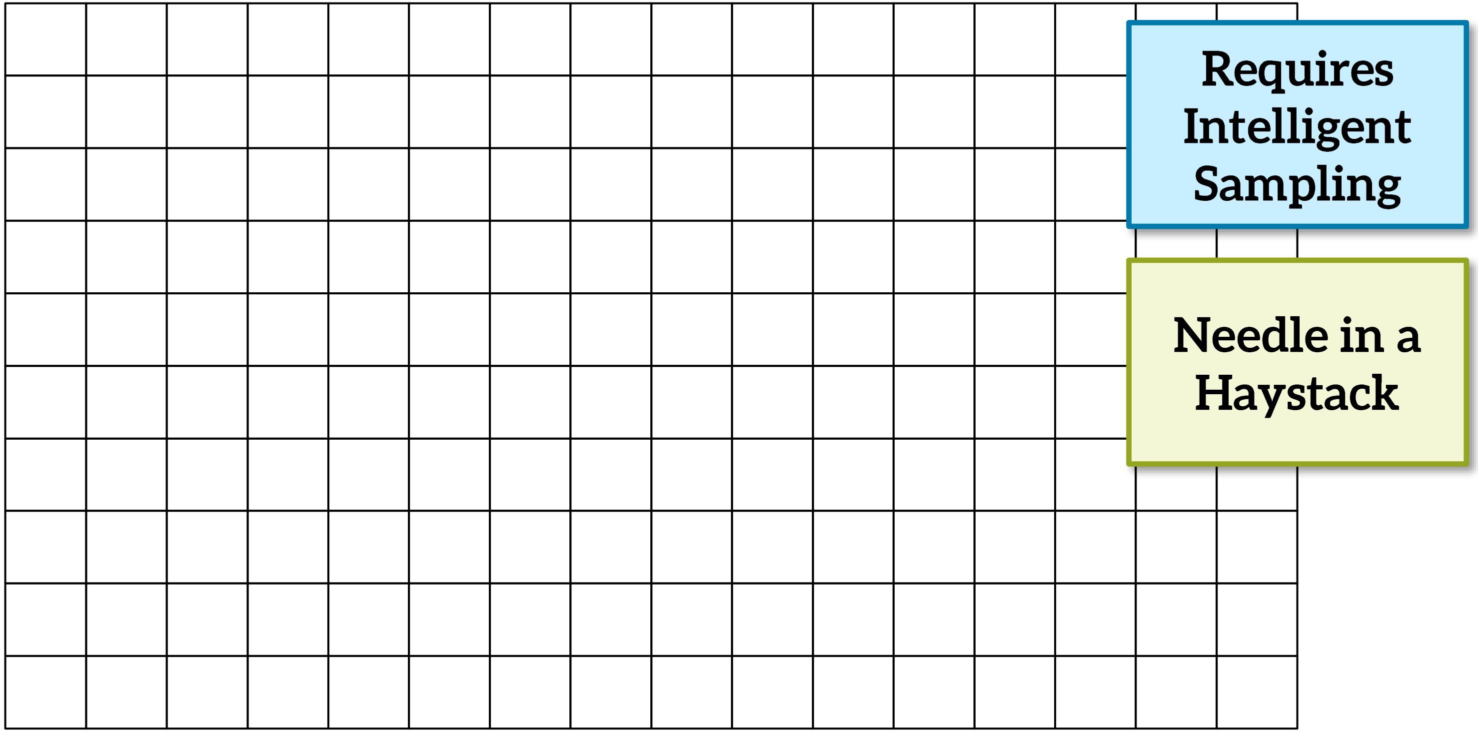


Search Algorithm: Enumeration



Search space is too big!

Search Algorithm: Stochastic



Search Algorithm: CEGIS

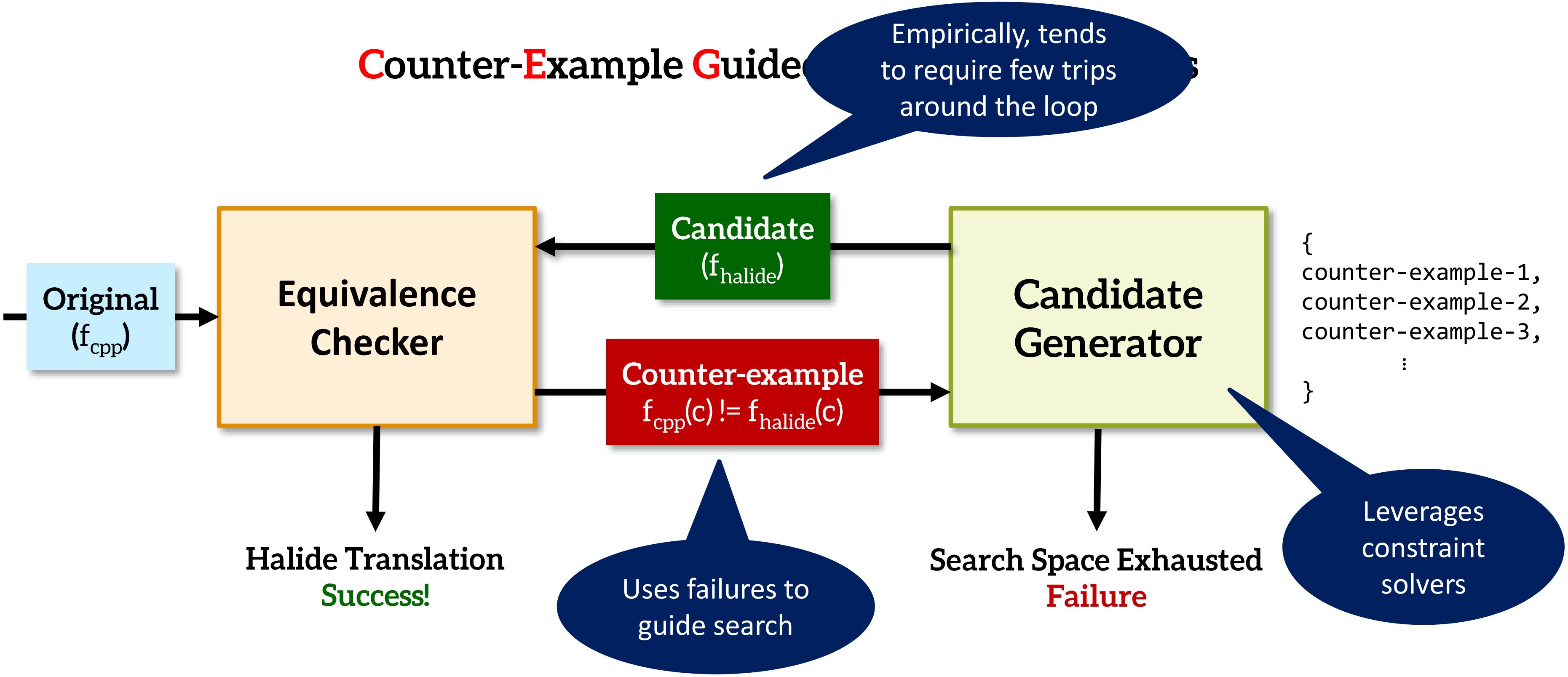


Image Processing Algorithms

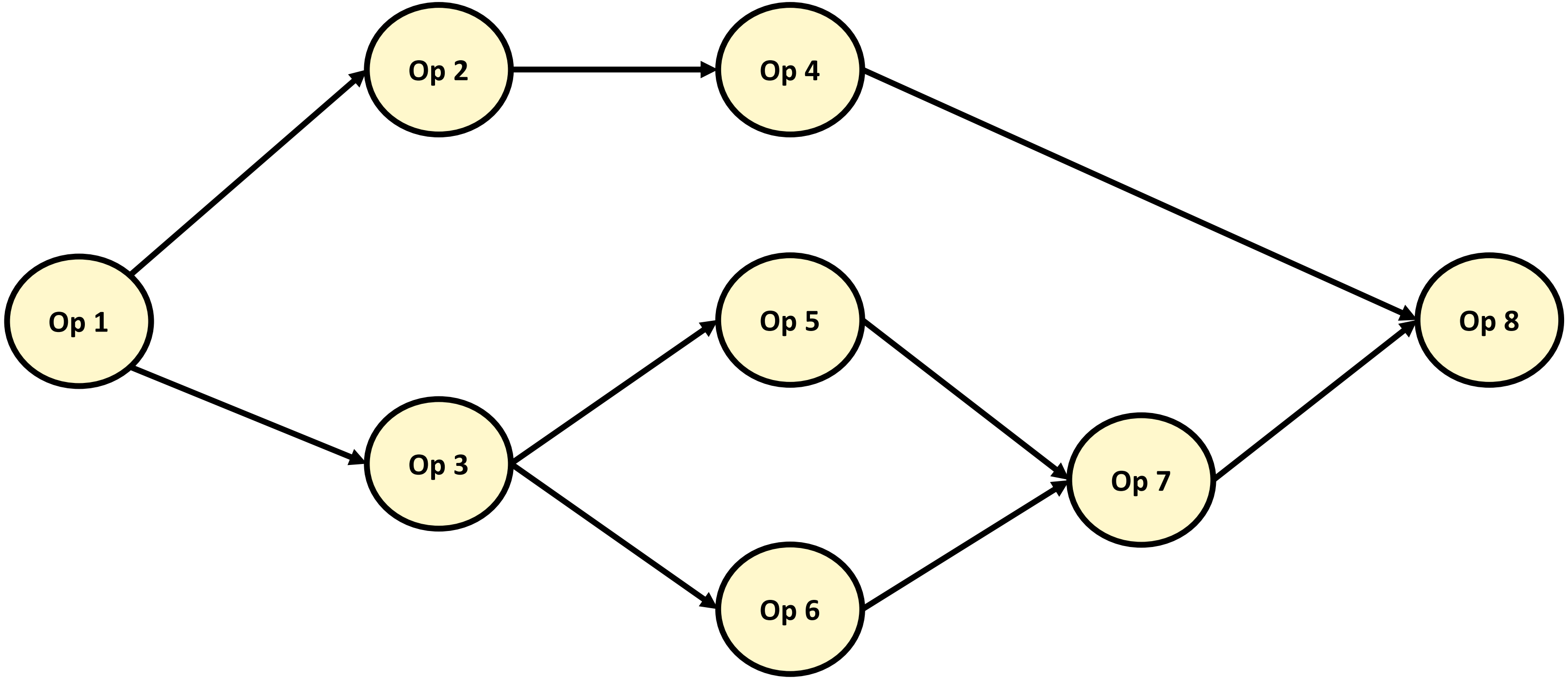


Image Processing Operations

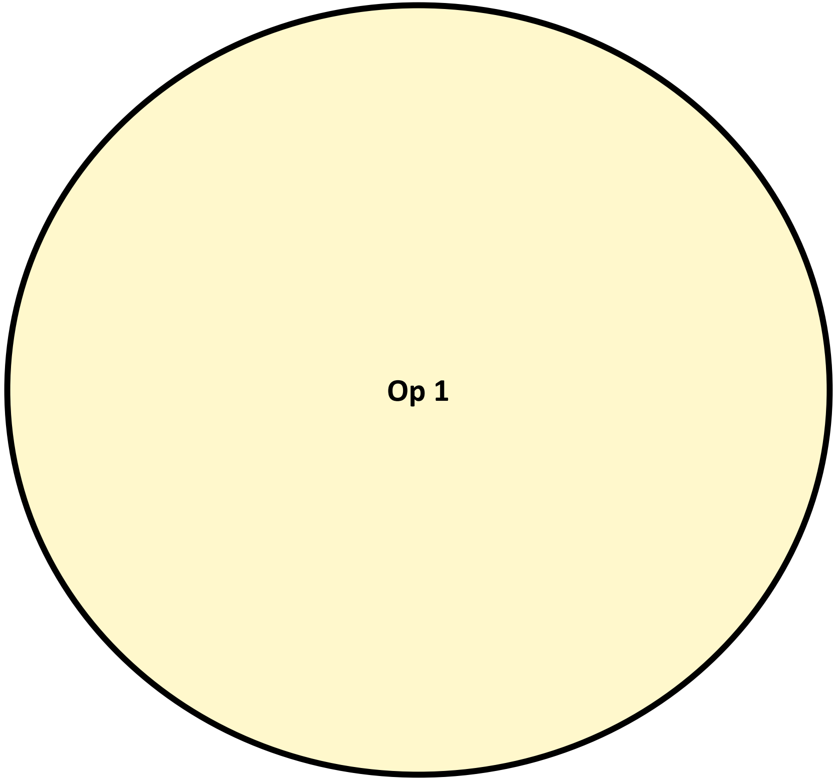


Image Processing Operations

1. *The **Region of Interest***
   (ROI) of the operation.

Output Image

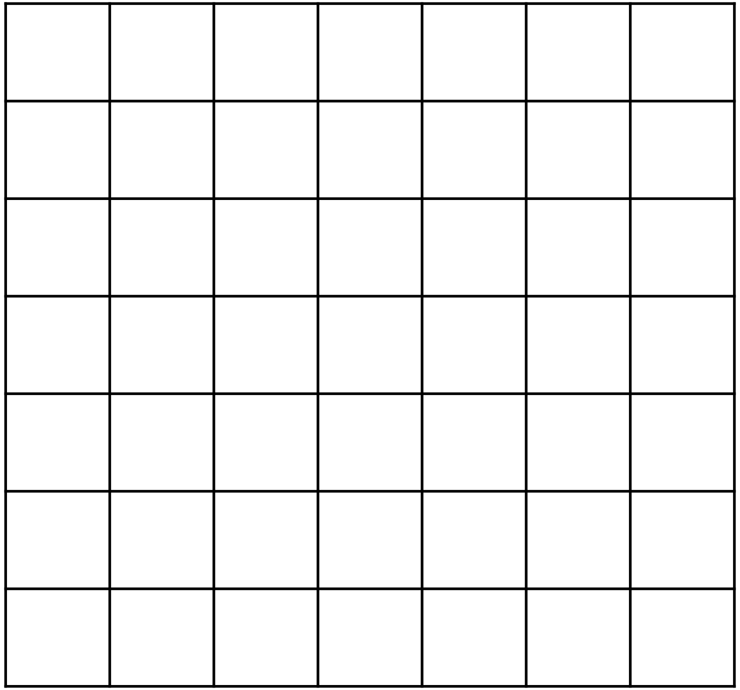


Image Processing Operations

1. *The **Region of Interest*** (ROI) of the operation.

Output Image

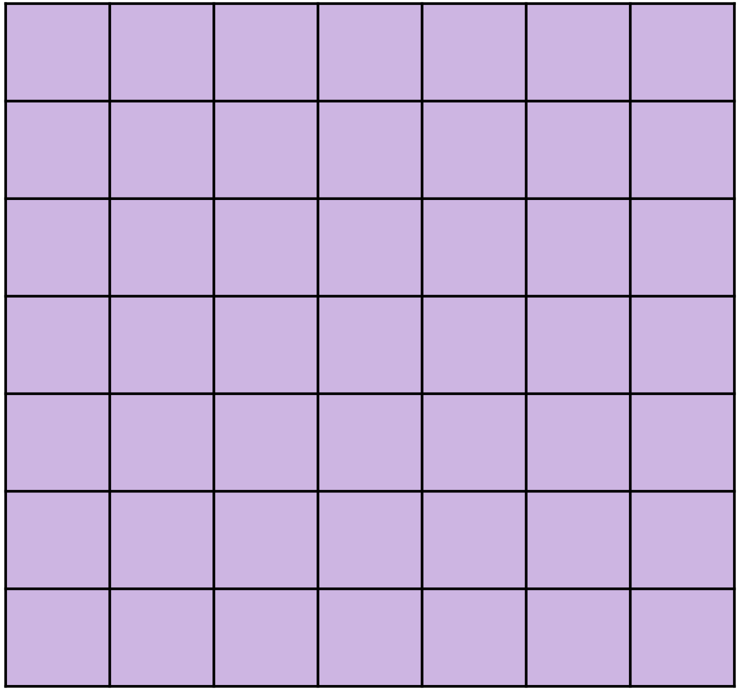


Image Processing Operations

1. *The **Region of Interest*** (ROI) of the operation.

Output Image

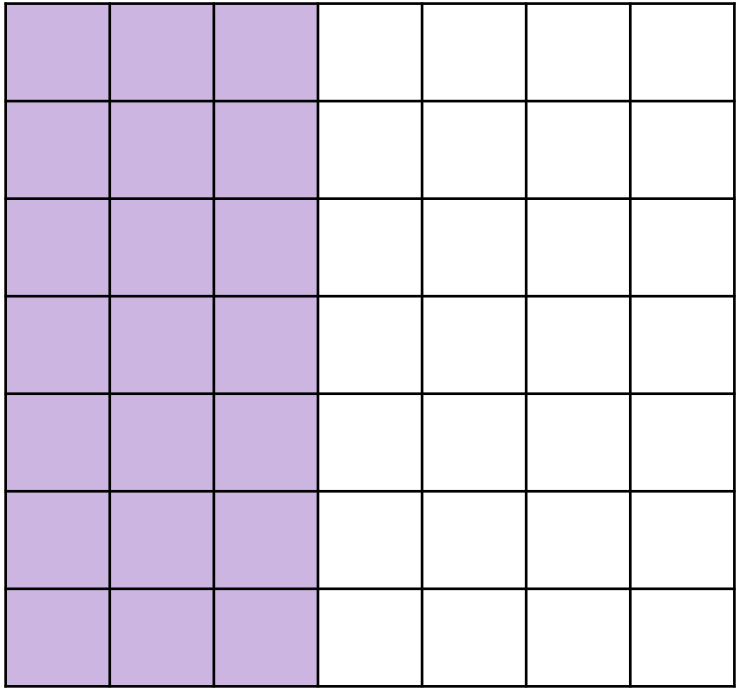


Image Processing Operations

1. *The **Region of Interest*** (ROI) of the operation.

Output Image

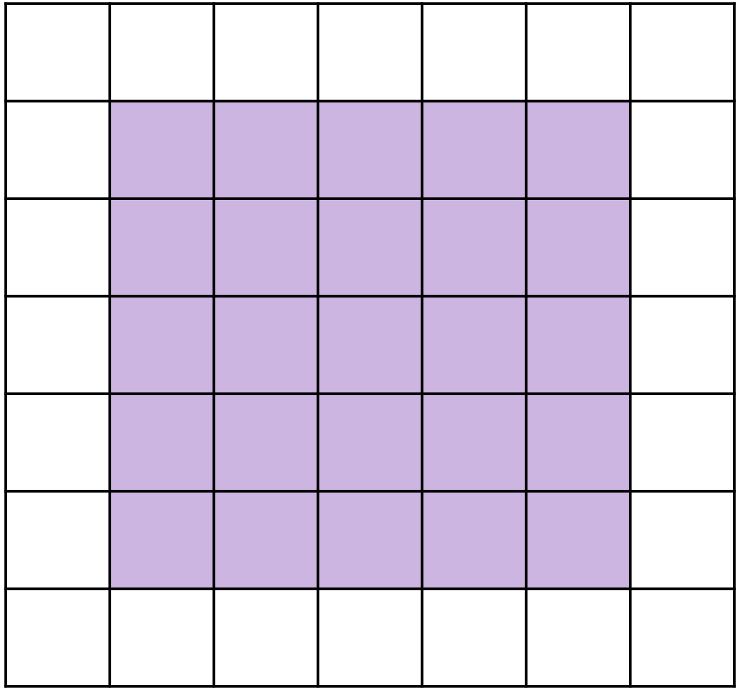
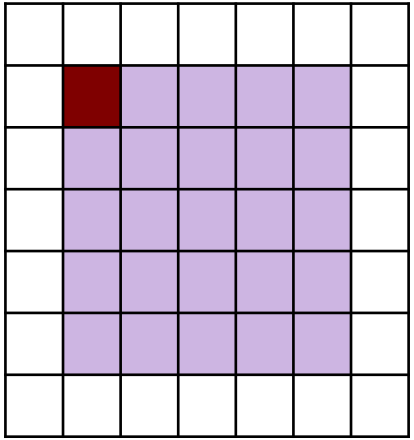


Image Processing Operations

1. *The* ***Region of Interest*** (ROI) of the operation.
2. The ***terminals*** used to compute the value of each pixel.

Output Image



Input Image

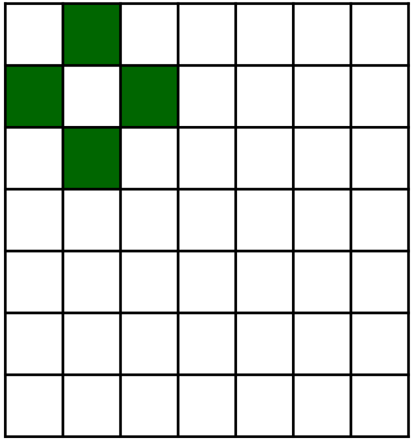
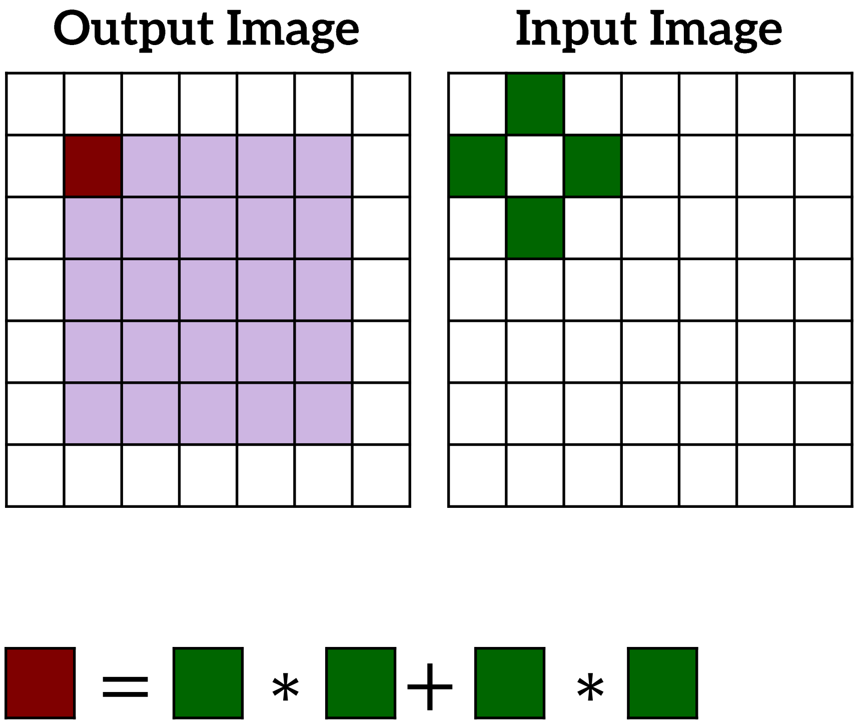
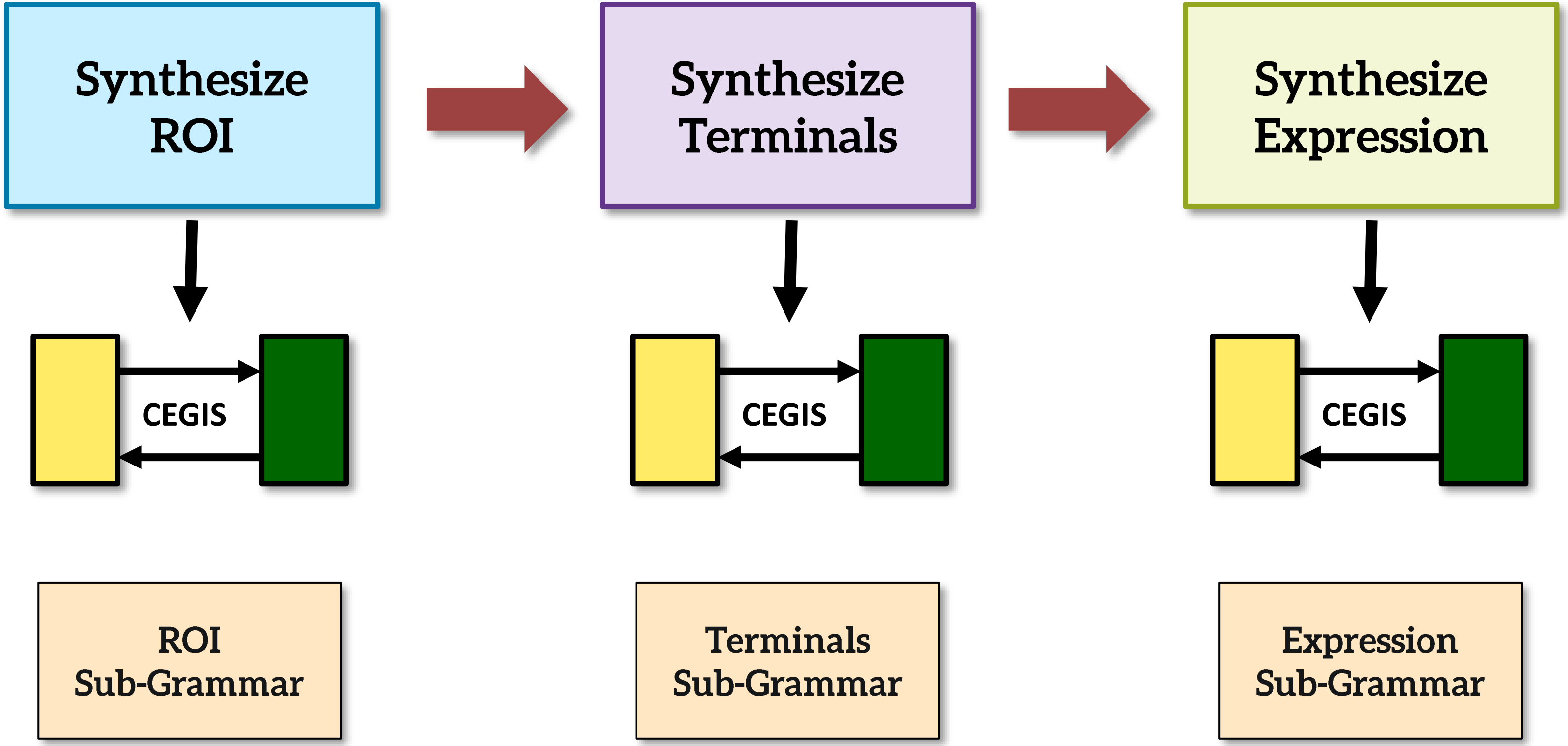


Image Processing Operations

1. *The **Region of Interest*** (ROI) of the operation.
2. The ***terminals*** used to compute the value of each pixel.
3. The **computation expression** over the set of terminals.



Dexter's 3-Stage Search



Verifying Region of Interest

Original Code

```
for (int i = 0; i < pixels; ++i) {
    out_cpp[i] = inp[i] * 0.5f;
}
```

Reduced Version

```
for (int i = 0; i < pixels; ++i) {
    out_cpp[i] = ⊥;
}
```

RHS of the assignment is abstracted away

Verifying Terminals

Synthesize ROI → **Synthesize Terminals** → **Synthesize Expression**

Original Code

```
for (int i = 0; i < pixels; ++i) {
    out_cpp[i] = inp[i] * 0.5f;
}
```

Reduced Version

```
for (int i = 0; i < pixels; ++i) {
    out_cpp[i] = ⊥(inp[i], 0.5f);
}
```

Computation Expression
is abstracted away

Does it work?

Evaluation: Adobe Photoshop

⏳ Lots of legacy code! (Ver 1.0 released in 1990)

353 performance-critical functions

- Compositing layers, rotations, blurs etc.
- Over 30,000 lines of code!

Complex and highly optimized code

Functions up to 150 lines of C++, containing:

- Vectorization,
- Bit-twiddling,
- Loop-unrolling etc.

Many operations a part of file format

Evaluation: Feasibility Results

Translated **264 (74.7%)** successfully!

57% Failures: Lack of supported C++ features

43% Failures: Search timed out

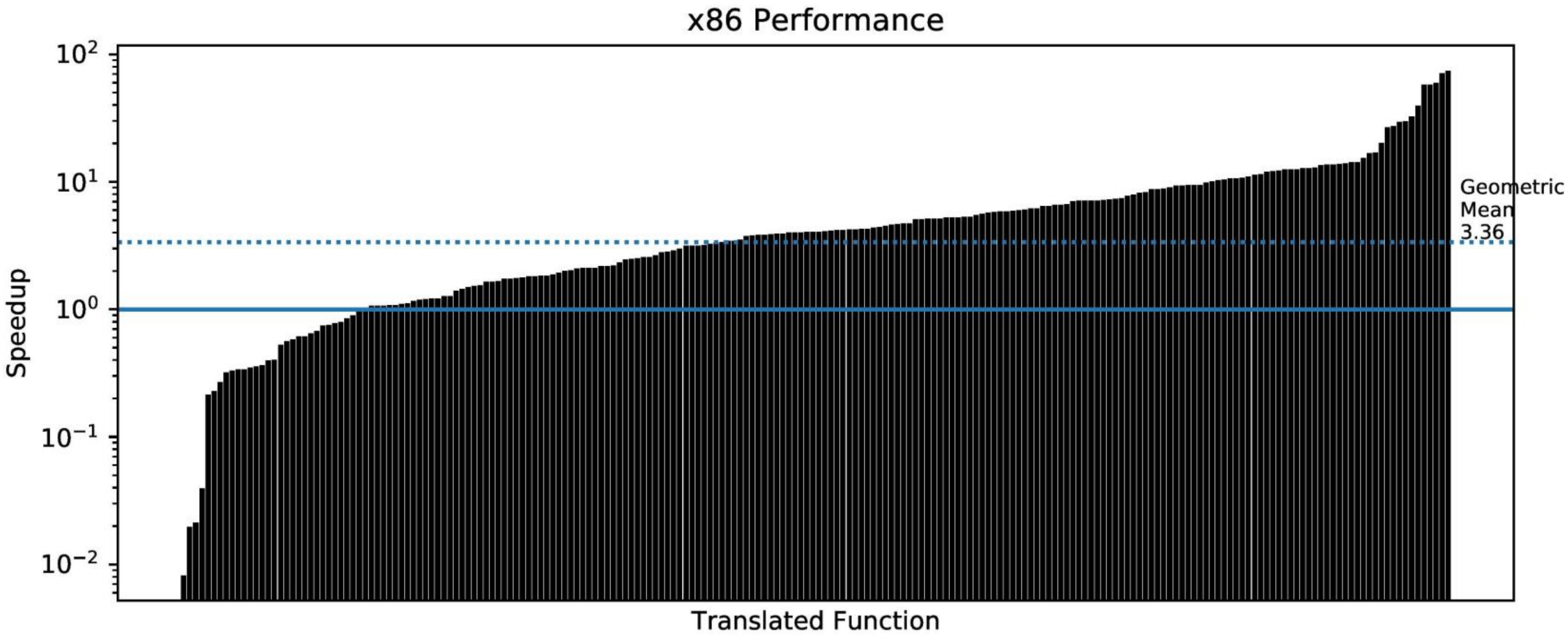
Total Compile time:
200 hours on 60 cores

Max time / function:
6 hours

Evaluation: Impact

The first of Dexter translated algorithms just shipped with the latest Photoshop release (Nov 11th, 2019)

Evaluation: Runtime Performance

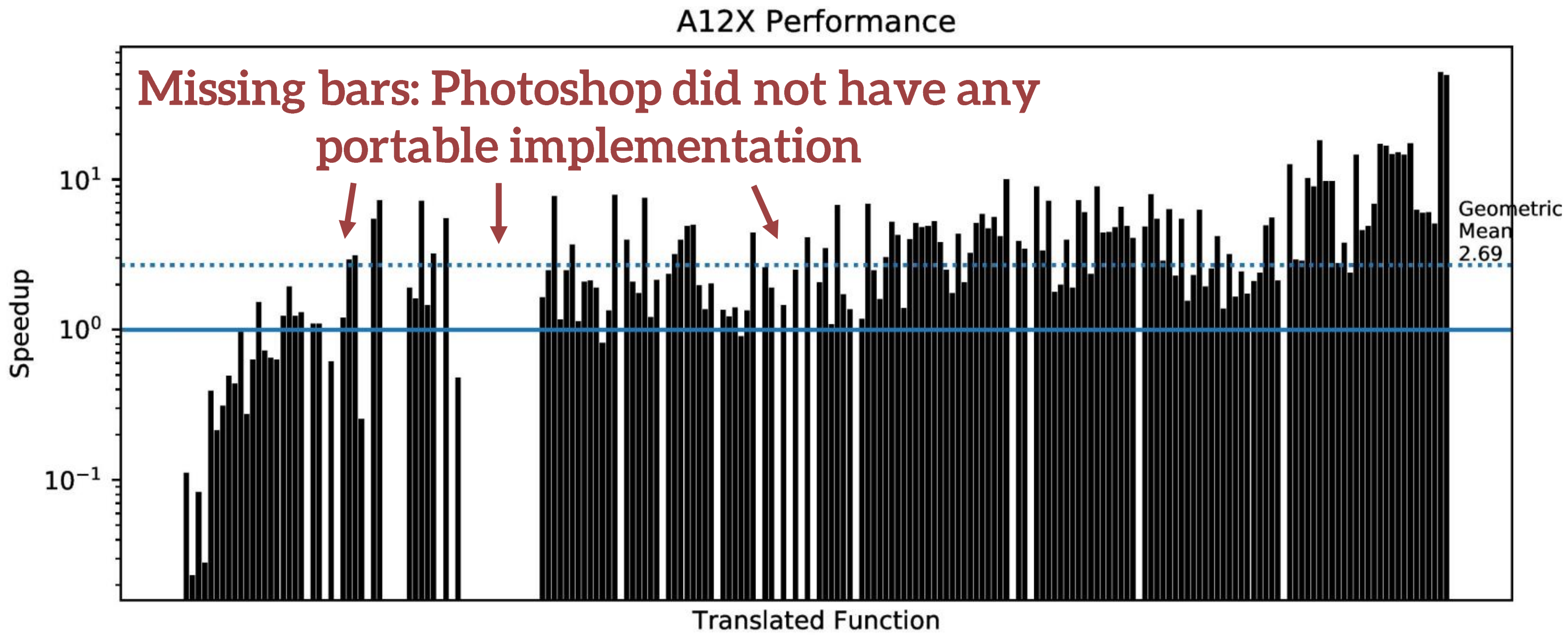


Median speedup
7.03x

86% benchmarks over 1x faster

70% benchmarks over 2x faster

Evaluation: Portability



Median speedup
4.52x

Future Work

- Scale synthesis to support more classes of algorithms
- Demonstrate feasibility for other source / target languages (e.g. CUDA → Halide)
- Port schedule from the legacy code

Conclusion

- Dexter can rejuvenate legacy image processing code by re-writing it to Halide.
- Our 3-stage synthesis algorithm accelerates synthesis of image processing algorithms.
- Our technique is robust and scalable enough to be applied to complex real-world code.

dexter.uwplse.org


THANK YOU

Conclusion

- Dexter can rejuvenate legacy image processing code by re-writing it to Halide.
- Our 3-stage synthesis algorithm accelerates synthesis of image processing algorithms.
- Our technique is robust and scalable enough to be applied to complex real-world code.

dexter.uwplse.org