Towards Optimising Programs with Sketch-Guided Polyhedral Compilation

Juliette Maytié Reuben Carolan Christophe Alias Thomas Kœhler Cedric Bastoul

28 January 2026 IMPACT

Objective: Semi-automatic optimisation

Naive → How ? → Fast

```
void multiplyMatrix(int m1[][C1], int m2[][C2])
{
    int result[R1][C2];

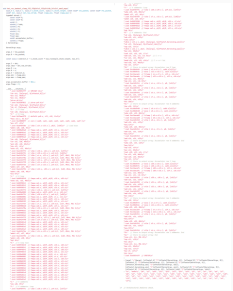
    printf("Resultant Matrix is:\n");

    for (int i = 0; i < R1; i++) {
        for (int j = 0; j < C2; j++) {
            result[i][j] = 0;

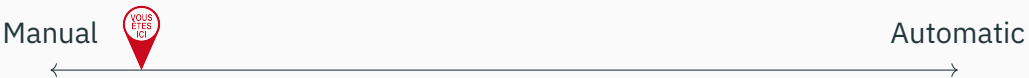
            for (int k = 0; k < R2; k++) {
                result[i][j] += m1[i][k] * m2[k][j];
            }

            printf("%d\t", result[i][j]);
        }

        printf("\n");
    }
}
```



Manual Optimization



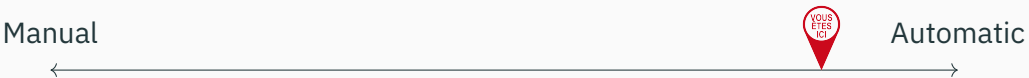
Examples :

- BLAS
- CuDNN (Nvidia)
- KleidiAI (ARM)
- RocBlas (AMD)
- OneDNN (Intel)
- MPS (Apple)

\+ Highly effective code
\+ Human expertise

\- Closed ecosystem
\- Very expensive

Automatic Optimisation



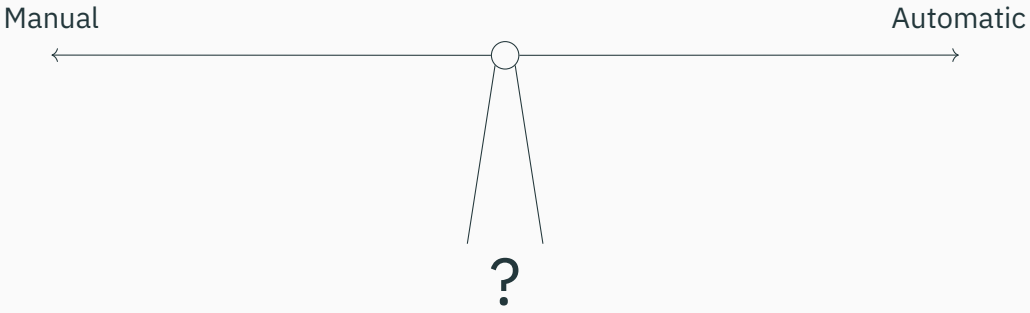
Examples :

- GCC/Clang
- Pluto
- TorchInductor

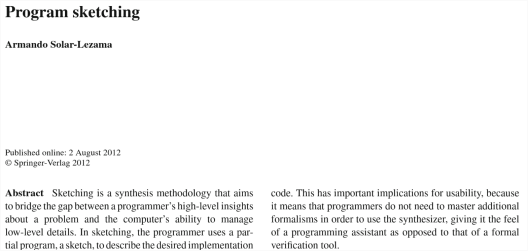
\+ Time saving
\+ Correct by construction

\- Little flexibility
\- Performance not guaranteed

Semi-automatic optimisation?



Synthesis-based



Program sketching

Armando Solar-Lezama

Published online: 2 August 2012
© Springer-Verlag 2012

Abstract Sketching is a synthesis methodology that aims to bridge the gap between a programmer's high-level insights about a problem and the computer's ability to manage low-level details. In sketching, the programmer uses a partial program, a sketch, to describe the desired implementation code. This has important implications for usability, because it means that programmers do not need to master additional formalisms in order to use the synthesizer, giving it the feel of a programming assistant as opposed to that of a formal verification tool.

```
void main(int x) {
    int y = x * ??;
    assert y == x + x;
}
```

```
void swap (ref bit[W] x, ref bit[W] y) {
    minrepeat
      {| x | y |} = x ^ y;
}
```

Synthesis-based

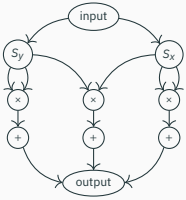
```
generator bit[W] gen(bit[W] x, int bnd) {
  assert bnd >= 0;
  if (??) return x;
  if (??) return ??;
  if (??) return ~gen(x, bnd - 1);
  if (??)
    return {| gen(x, bnd - 1) (+ | & | ^)
              gen (x, bnd - 1)|}
}
```

Limits:

- SAT solver -> not Scalable
- Search and control mixed
- Lack of construction on control flow

Schedule

Calculation specification

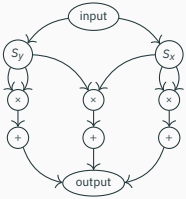


Optimisation script

```
output.split(y, y, yi, 32)
  .parallel(y)
  .vectorize(x, vec);
input.store_at(output, y)
  .compute_at(output, yi)
  .vectorize(x, vec);
Sx.store_at(output, y)
  .compute_at(output, yi)
  .vectorize(x, vec);
Sy.store_at(output, y)
  .compute_at(output, yi)
  .vectorize(x, vec);
Sx.compute_with(Sy, x);
```

Schedule

Calculation specification



Optimisation script

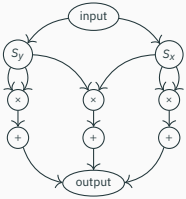
```
output.split(y, y, yi, 32)
  .parallel(y)
  .vectorize(x, vec);
input.store_at(output, y)
  .compute_at(output, yi)
  .vectorize(x, vec);
Sx.store_at(output, y)
  .compute_at(output, yi)
  .vectorize(x, vec);
Sy.store_at(output, y)
  .compute_at(output, yi)
  .vectorize(x, vec);
Sx.compute_with(Sy, x);
```

Simplified result

```
pfor y:
  store input, Sx, Sy
  for yi:
    input
    Sx, Sy
    for x:
      vfor xi:
        output
```

Schedule

Calculation specification



Optimisation script

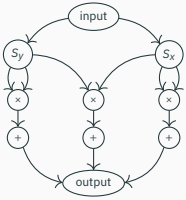
```
output.split(y, y, yi, 32)
  .parallel(y)
  .vectorize(x, vec);
input.store_at(output, y)
  .compute_at(output, yi)
  .vectorize(x, vec);
Sx.store_at(output, y)
  .compute_at(output, yi)
  .vectorize(x, vec);
Sy.store_at(output, y)
  .compute_at(output, yi)
  .vectorize(x, vec);
Sx.compute_with(Sy, x);
```

Simplified result

```
pfor y:
  store input, Sx, Sy
  for yi:
    input
    Sx, Sy
    for x:
      vfor xi:
        output
```

Schedule

Calculation specification



Optimisation script

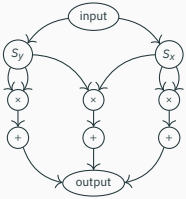
```
output.split(y, y, yi, 32)
  .parallel(y)
  .vectorize(x, vec);
input.store_at(output, y)
  .compute_at(output, yi)
  .vectorize(x, vec);
Sx.store_at(output, y)
  .compute_at(output, yi)
  .vectorize(x, vec);
Sy.store_at(output, y)
  .compute_at(output, yi)
  .vectorize(x, vec);
Sx.compute_with(Sy, x);
```

Simplified result

```
pfor y:
  store input, Sx, Sy
  for yi:
    input
    Sx, Sy
    for x:
      vfor xi:
        output
```

Schedule

Calculation specification



Optimisation script

```
output.split(y, y, yi, 32)
  .parallel(y)
  .vectorize(x, vec);
input.store_at(output, y)
  .compute_at(output, yi)
  .vectorize(x, vec);
Sx.store_at(output, y)
  .compute_at(output, yi)
  .vectorize(x, vec);
Sy.store_at(output, y)
  .compute_at(output, yi)
  .vectorize(x, vec);
Sx.compute_with(Sy, x);
```

Simplified result

```
pfor y:
  store input, Sx, Sy
  for yi:
    input
    Sx, Sy
    for x:
      vfor xi:
        output
```

Polysketch: dream

Naive

```
for (int x = 0; x < N; x++)
    for (int y = 0; y < x; y++)
        B[y][x] = ...;
for (int x = 0; x < N; x++)
    for (int y = 0; y < x; y++)
        C[y][x] = ...;
```

+

Sketch

```
for y:
    for x:
        C
```

⇒

Output

```
for (int y = 0; y < N; y++) {
    for (int x = 0; x <= y; x++) {
        B[y][x] = ...;
        C[y][x] = ...;
    }
}
```

Polysketch: This talk

Naive + Sketch ⇒ Output

```
for (y, x) in PDomain("0≤x<N∧0≤x≤y"):
    B[y, x] = ...;
for (y, x) in PDomain("0≤x<N∧0≤x≤y"):
    C[y, x] = ...;
```

SARE
python DSL

```
for y in Par():
    for x in Par():
        B[y, x];
        C[y, x];
```

Sketch
python DSL

```
for (int y = 0; y < N; y++) {
    for (int x = 0; x <= y; x++) {
        B[y][x] = ...;
        C[y][x] = ...;
    }
}
```

Executable C code

Input languages

Sketch

```
Id     ::= String
Sketch ::= Arr(Id, List(Index))
         | Sketch; Sketch
         | Pfor v { Sketch }
Index  ::= ~Linear_function | Linear_function
```

Spec

```
Id      ::= String
Spec    ::= { equation: Id -> Array,
              in:  List(Id, PDomain),
              out: List(Id) }
Array   ::= { vars: List(Id); dom: PDomain; expr: Expr }
Expr    ::= v | n | Expr ∘ Expr
          | Read(Id, List(Linear_function))
          | If PDomain Expr Expr
```

Example Blur 1d

Sketch

```
for i in Par():
  for j in Par():
     B[~(i + j)]
  C[~i]
```

Specification

$$\texttt{In}: \quad A[k] : \{0 \le k < N\}$$

$$\texttt{Out}: C$$

$$B[k] : \{0 \le k < N-2\} := \frac{A[k] + A[k+1] + A[k+2]}{3}$$

$$C[k] : \{0 \le k < N-4\} := \frac{B[k] + B[k+1] + B[k+2]}{3}$$

Blur 1d: Completion

$\vdash C[k] : 0 \leq k < N - 4$

```
for i in Par():

  for j in Par():

    B[~(i + j)]

  C[~i]
```

Blur 1d: Completion

```
for i in Par():
```
$i \in \{\} \vdash C[k] : 0 \leq k < N - 4$

```
  for j in Par():

    B[~(i + j)]


  C[~i]
```

Blur 1d: Completion

```
for i in Par():

  for j in Par():

    B[~(i + j)]
```

$i \in \{\} \vdash C[k] : 0 \leq k < N - 4$

```
C[~i]
```

C iteration Domain

$$(i), () \to \{0 \leq i < N - 4\}$$

$$f(i) = (i, i)$$

$$(i), (k) \to \{0 \leq k < N - 4\}$$

Blur 1d: Completion

```
for i in Par():


  for j in Par():

    B[~(i + j)]

  (i),() → {0 ≤ i < N − 4}
  C[i] = (B[i] + ...) / 3
  i ∈ {0 ≤ i < N − 4} ⊢ B[k] : i ≤ k ≤ i + 2
```

Blur 1d: Completion

```
for i in Par():
```

$i \in \{0 \le i < N-4\} \vdash B[k] : 0 \le k \le i+2$

```
  for j in Par():

    B[~(i + j)]
```

$(i), () \to \{0 \le i < N-4\}$

```
  C[i] = (B[i] + ...) / 3
```

Blur 1d: Completion

```
for i in Par():
```

```
  for j in Par():
```
$i,j \in \{0 \le i < N-4\} \vdash B[k] : i \le k \le i+2$
```
    B[~(i + j)]
```

$(i),() \to \{0 \le i < N-4\}$
```
  C[i] = (B[i] + ...) / 3
```

B iteration Domain

$$(i,j),() \to \{0 \le i < N-4 \wedge 0 \le j \le 2\}$$

$$f(i,j) = (i,j,i+j)$$

$$(i,j),(k) \to \{0 \le i < N-4 \wedge i \le k \le i+2\}$$

Blur 1d: Completion

```
for i in Par():
```

```
  for j in Par():
    (i,j),() → {0 ≤ i < N − 4 ∧ 0 ≤ j ≤ 2}
    B[i + j] = (A[i + j] + ...) / 3
    ⊢
  (i),() → {0 ≤ i < N − 4}
  C[i] = (B[i] + ...) / 3
```

Blur 1d: Completion

```
for i in Par():

  (i), () → {0 ≤ i < N − 4}
  for j ∈ {0 ≤ j ≤ 2}:

    B[i + j] = (A[i + j] + ...) / 3
  ⊢
  (i), () → {0 ≤ i < N − 4}
  C[i] = (B[i] + ...) / 3
```

Blur 1d: Completion

```
for i in Par():
  (i), () → {0 ≤ i < N − 4}

  for j ∈ {0 ≤ j ≤ 2}:

    B[i + j] = (A[i + j] + ...) / 3


  C[i] = (B[i] + ...) / 3
  ⊢
```

Blur 1d: Completion

$(), () \to \{\}$

```
for i ∈ {0 ≤ i < N − 4}:

  for j ∈ {0 ≤ j ≤ 2}:

    B[i + j] = (A[i + j] + ...) / 3

  C[i] = (B[i] + ...) / 3
```

⊢

Example: Shift

Sketch

```
for i in Par():
  B[i]
  C[i]
```

Specification

$$\texttt{In}: \quad A[k] : \{0 \le k < N\}$$

$$\texttt{Out}: C$$

$$B[k] : \{1 \le k < N+1\} := A[k-1]$$

$$C[k] : \{2 \le k < N+2\} := B[k-1]$$

Shift: Completion

$(), () \to \{\} \quad \vdash C[k] : 2 \le k < N + 2$

```
for i in Par():

  B[i]

  C[i]
```

Shift: Completion

```
for i in Par():
```
$(i), () \to \{\} \quad i \in \{\} \vdash C[k] : 2 \leq k < N + 2$

```
  B[i]


  C[i]
```

Shift: Completion

```
for i in Par():

  B[i]

  (i),() → {}    i ∈ {} ⊢ C[k] : 2 ≤ k < N+2
  C[i]
```

C iteration domain

$$(i, e_c), () \to \{2 \le i + e_c < N + 2\}$$

$$f(i, e_c) = (i, i + e_c)$$

$$(i), (k) \to \{2 \le k < N + 2\}$$

Shift: Completion

```
for i in Par():

  B[i]
```

$(i, e_c), () \to \{2 \le i + e_c < N + 2\}$

```
C[i + e_c] = B[i + e_c - 1];
```

$(i, e_c), () \to \{\}$

$i, e_c \in \{2 \le i + e_c < N + 2\} \vdash B[k] : k = i + e_c - 1$

Shift: Completion

B iteration domain

$(i, e_c, e_b), () \to \{1 \le i + e_b < N + 1\}$

$f(i, e_c, e_b) = (i, e_c, i + e_b)$

$(i, e_c), (k) \to \{2 \le i + e_c < N + 2 \wedge k = i + e_c - 1\}$

```
for i in Par():
```
$(i, e_c), () \to \{\}$

$i, e_c \in \{2 \le i + e_c < N + 2\} \vdash B[k] : k = i + e_c - 1$

```
  B[i]
```

$(i, e_c), () \to \{2 \le i + e_c < N + 2\}$

```
  C[i + e_c] = B[i + e_c - 1];
```

Shift: Completion

```
for i in Par():
```

$(i, e_c, e_b), () \to \{1 \le i + e_b < N + 1\}$

```
  B[i + e_b] = A[i + e_b - 1];
```

$(i, e_c, e_b), () \to \{e_b = e_c - 1\} \quad \vdash$

$(i, e_c), () \to \{2 \le i + e_c < N + 2\}$

```
  C[i + e_c] = B[i + e_c - 1];
```

Shift: Completion

```
for i in Par():
```
$(i, e_c, e_b), () \to \{1 \le i + e_b < N + 1\}$

```
  B[i + e_b] = A[i + e_b - 1];


  C[i + e_c] = B[i + e_c - 1];
```
$(i, e_c, e_b), () \to \{e_b = e_c - 1\} \quad \vdash$

Shift: Completion

$(e_c, e_b), () \to \{\}$

```
for i ∈ {1 ≤ i < N+1}:

  B[i + e_b] = A[i + e_b - 1];

  C[i + e_c] = B[i + e_c - 1];
```

$(e_c, e_b), () \to \{e_b = e_c - 1\} \quad \vdash$

Shift: Completion

$(), () \to \{\}$

evar([e_c,e_b], $\{e_b = e_c - 1\}$);

for $i \in \{1 \le i < N+1\}$:

```
  B[i + e_b] = A[i + e_b - 1];
```

```
  C[i + e_c] = B[i + e_c - 1];
```

$(), () \to \{\} \quad \vdash$

Conclusion

Future works:

- Benchmark
- Sketch → Dream
    + Sequential loop
    + Tiling