# Verified Lifting of Stencil Computations

**Shoaib Kamil (Adobe), Alvin Cheung (UWashington),
Shachar Itzhaky (MIT), Armando Solar-Lezama (MIT)**

**Presented by Alexa VanHattum**

Adobe


adobe.com/products/photoshop

Adobe    Support Home    Forums Home    News    Product    Explore    Sign

More discussions in Photoshop

**7 Replies** | Latest reply on Sep 1, 2017 1:12 AM by Trevor.Dennis

tomaugerdotcom Jun 3, 2013 7:41 AM

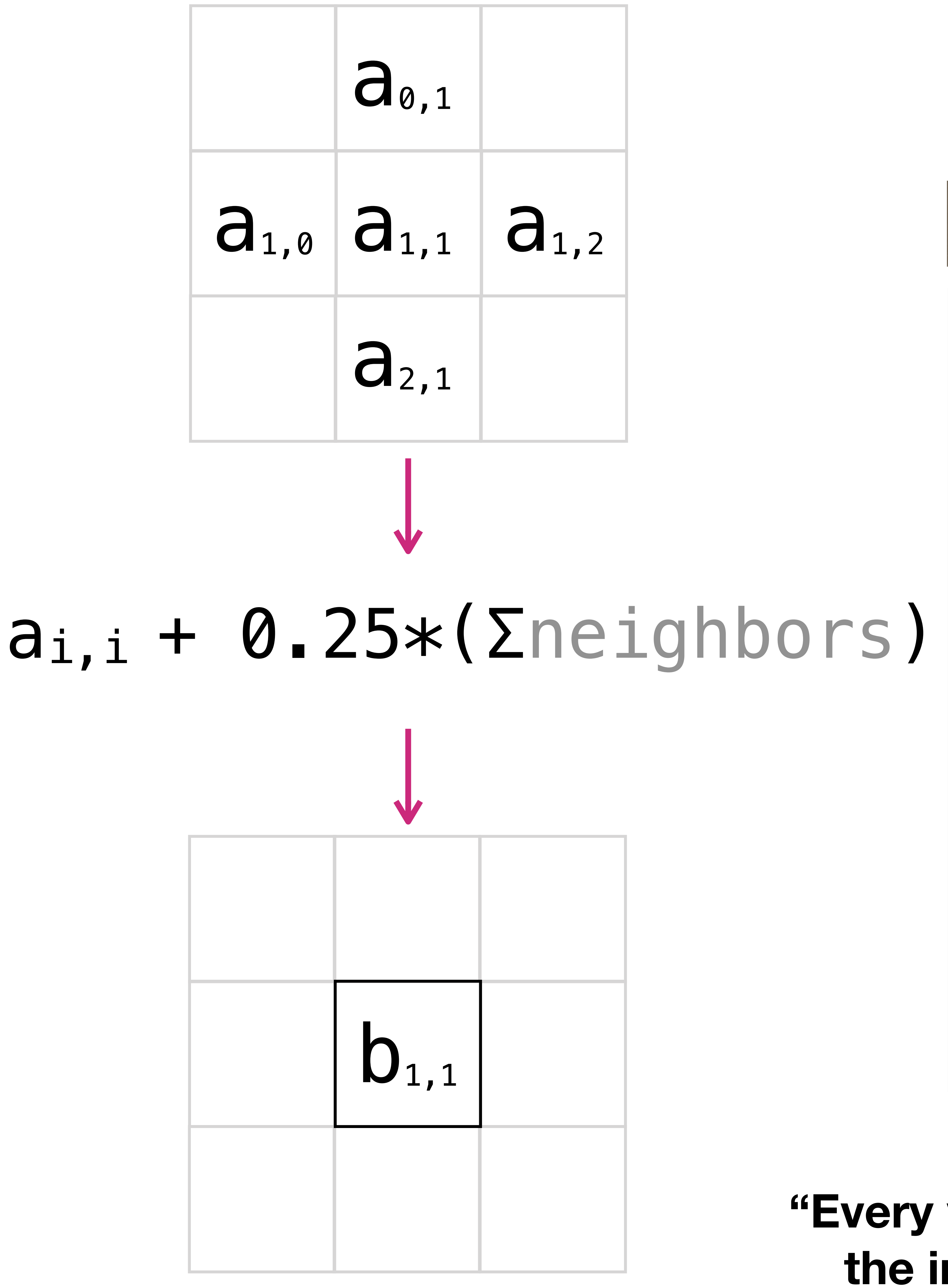
# Why is Blur Gallery so slow?

This question is **Not Answered.**

I have a Windows 7 beast with 16Gb RAM and 8 Processors and a kickass GPU as well. Generally Photoshop smokes through just about anything. But I just ran Blur Gallery and the thing crawled. 20 seconds between UI updates - I'm not even talking applying the filter!

*forums.adobe.com/thread/1224637*

```fortran
do ib=istart,iend,2*ibsize
  do i=ib, min(ib+ibsize,iend)
    do j=jstart,jend,2
      b(i,j) = a(i,j) + 0.25*(a(i+1,j+a(i-1,j+a(i,j+1)+a(i,j-1))
      b(i,(j+1)) = a(i,(j+1)) +
         0.25*(a(i+1,(j+1))+a(i-1,(j+1))+a(i,(j+1)+1)+a(i,(j+1)-1))
    enddo
  do i=ib+ibsize, min(ib+2*ibsize,iend)
    do j=jstart,jend,2
      b(i,j) = a(i,j) + 0.25*(a(i+1,j+a(i-1,j+a(i,j+1)+a(i,j-1))
      b(i,(j+1)) = a(i,(j+1)) +
         0.25*(a(i+1,(j+1))+a(i-1,(j+1))+a(i,(j+1)+1)+a(i,(j+1)-1))
    enddo
  enddo
enddo
```

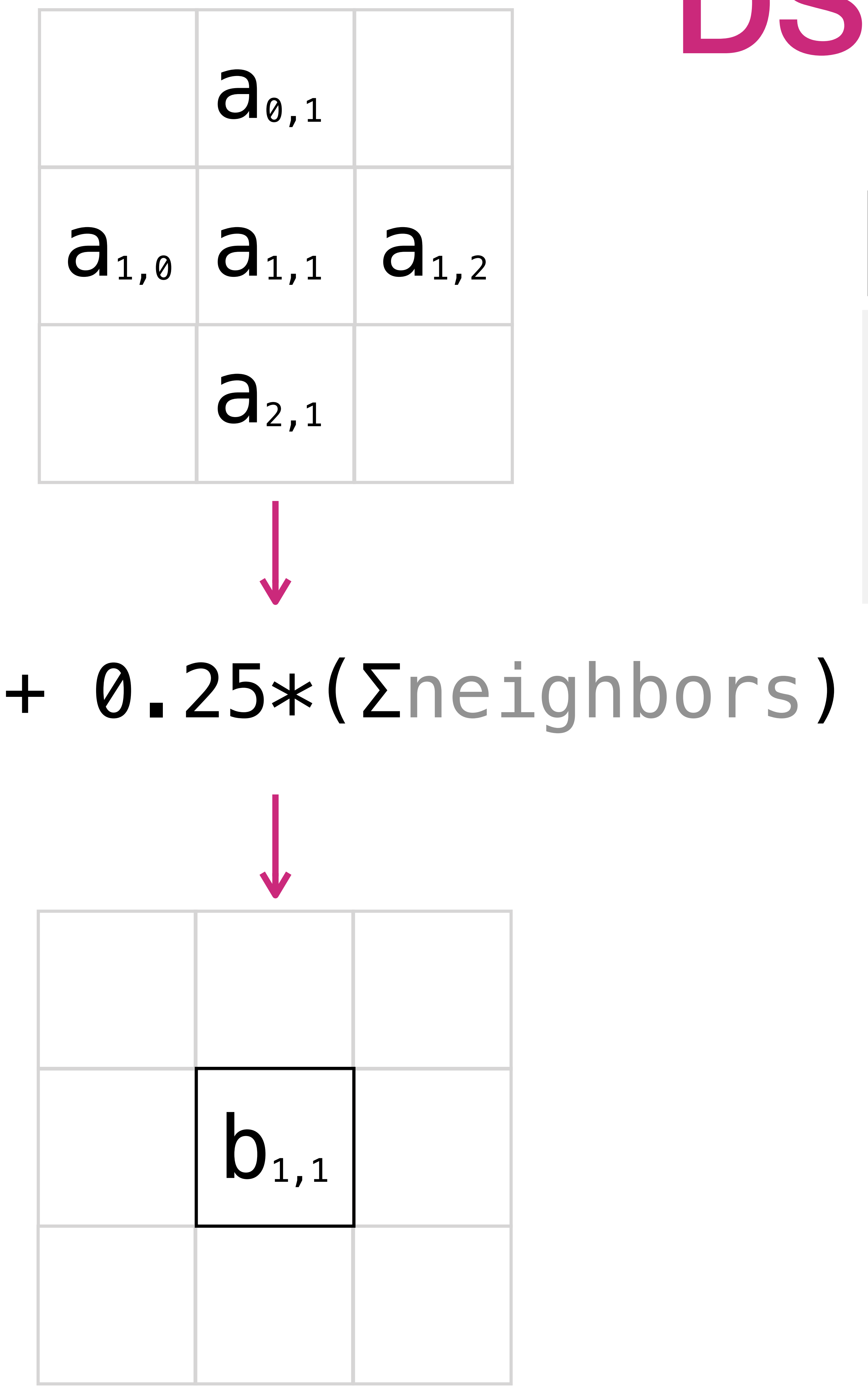



## Fortran

```fortran
do ib=istart,iend,2*ibsize
  do i=ib, min(ib+ibsize,iend)
    do j=jstart,jend,2
      b(i,j) = a(i,j) + 0.25*(a(i+1,j+a(i-1,j+a(i,j+1)+a(i,j-1))
      b(i,(j+1)) = a(i,(j+1)) +
         0.25*(a(i+1,(j+1))+a(i-1,(j+1))+a(i,(j+1)+1)+a(i,(j+1)-1))
    enddo
  do i=ib+ibsize, min(ib+2*ibsize,iend)
    do j=jstart,jend,2
      b(i,j) = a(i,j) + 0.25*(a(i+1,j+a(i-1,j+a(i,j+1)+a(i,j-1))
      b(i,(j+1)) = a(i,(j+1)) +
         0.25*(a(i+1,(j+1))+a(i-1,(j+1))+a(i,(j+1)+1)+a(i,(j+1)-1))
    enddo
  enddo
enddo
```

**“Every value in the output array should be the value in the input plus the weighted sum of its neighbors.**

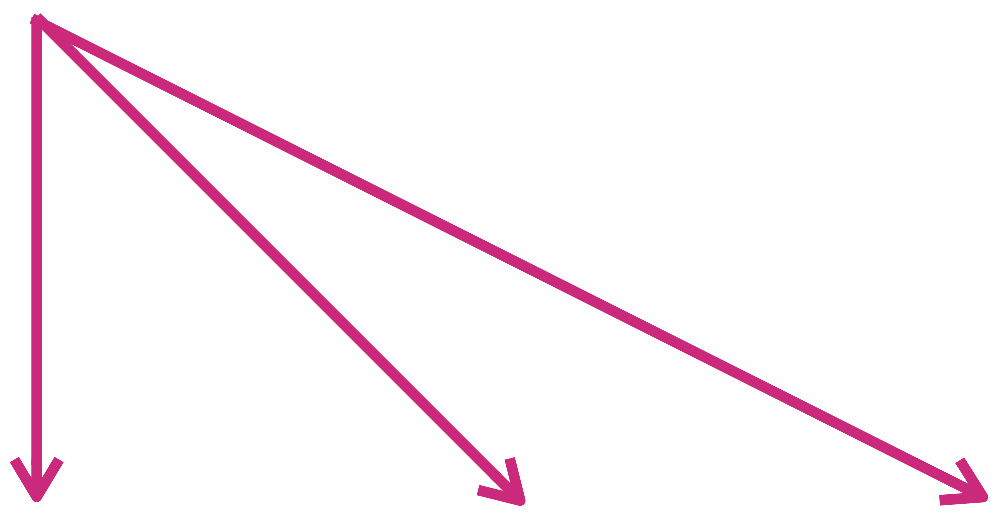
# DSLs to the rescue?




**Halide** : image & tensor processing DSL embedded in C++

```
ImageParam b(type_of<double>(),2);
Func func;
Var i, j;
func(i,j) = b(i,j) + 0.25*(b(i+1,j+b(i-1,j+b(i,j+1)+b(i,j-1))
func.compile_to_file("ex1", b);
```

**// ... schedule elided**

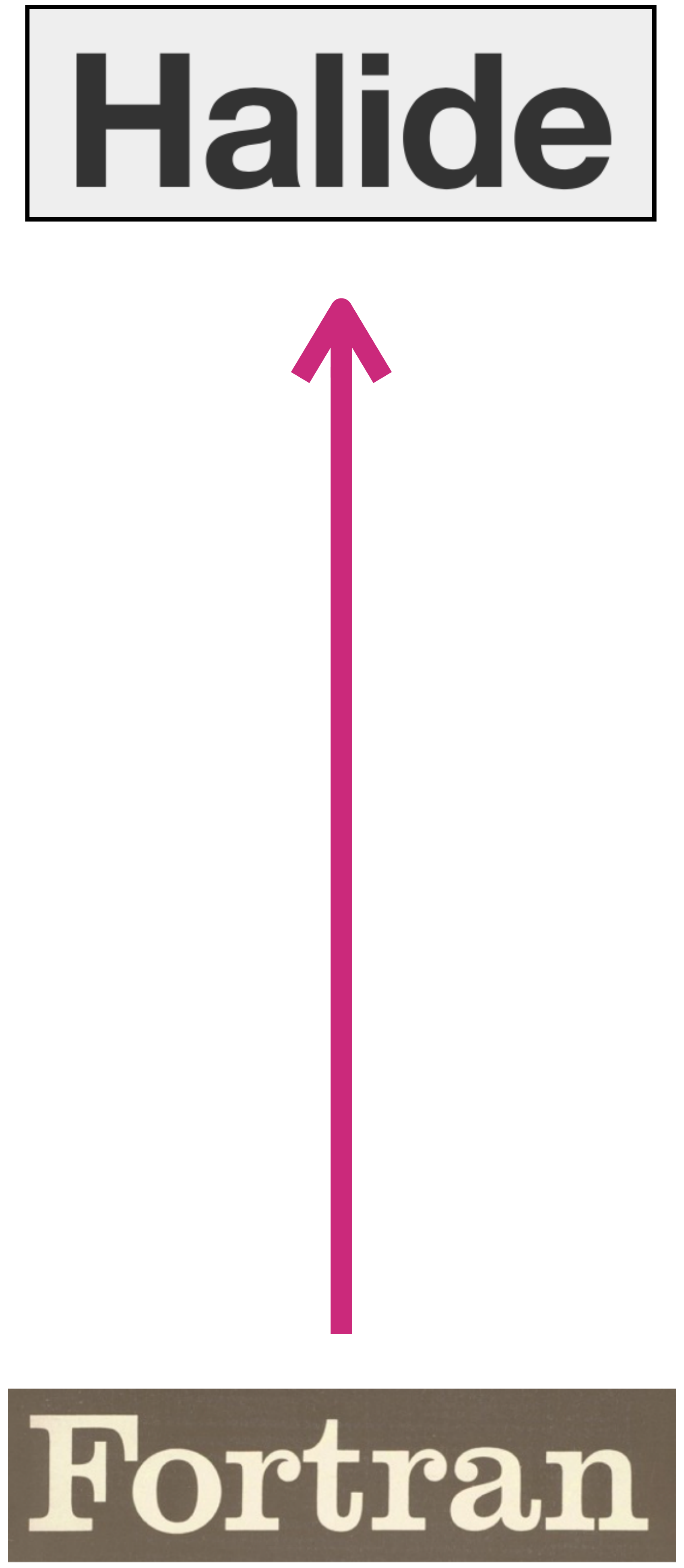


**CPUs** **GPUs** **new architectures!**

| | | |
|---|---|---|
| $a_{0,0}$ | $a_{0,1}$ | $a_{0,2}$ |
| $a_{1,0}$ | $a_{1,1}$ | $a_{1,2}$ |
| $a_{2,0}$ | $a_{2,1}$ | $a_{2,2}$ |

$$b_{1,1} = f(a_{0,0},\dots,a_{2,2})$$

| | | |
|---|---|---|
| | | |
| | $\boxed{b_{1,1}}$ | |
| | | |




**Verified lifting:**
Given a stencil computation, find a high-level summary that fully captures the algorithm

**Solution goals:**
- Extracts algorithm from optimization
- Fully automated
- Verified soundness

# “STNG”

Fortran

Identify stencil computations

Key contribution!

Synthesize Hoare-style verification conditions

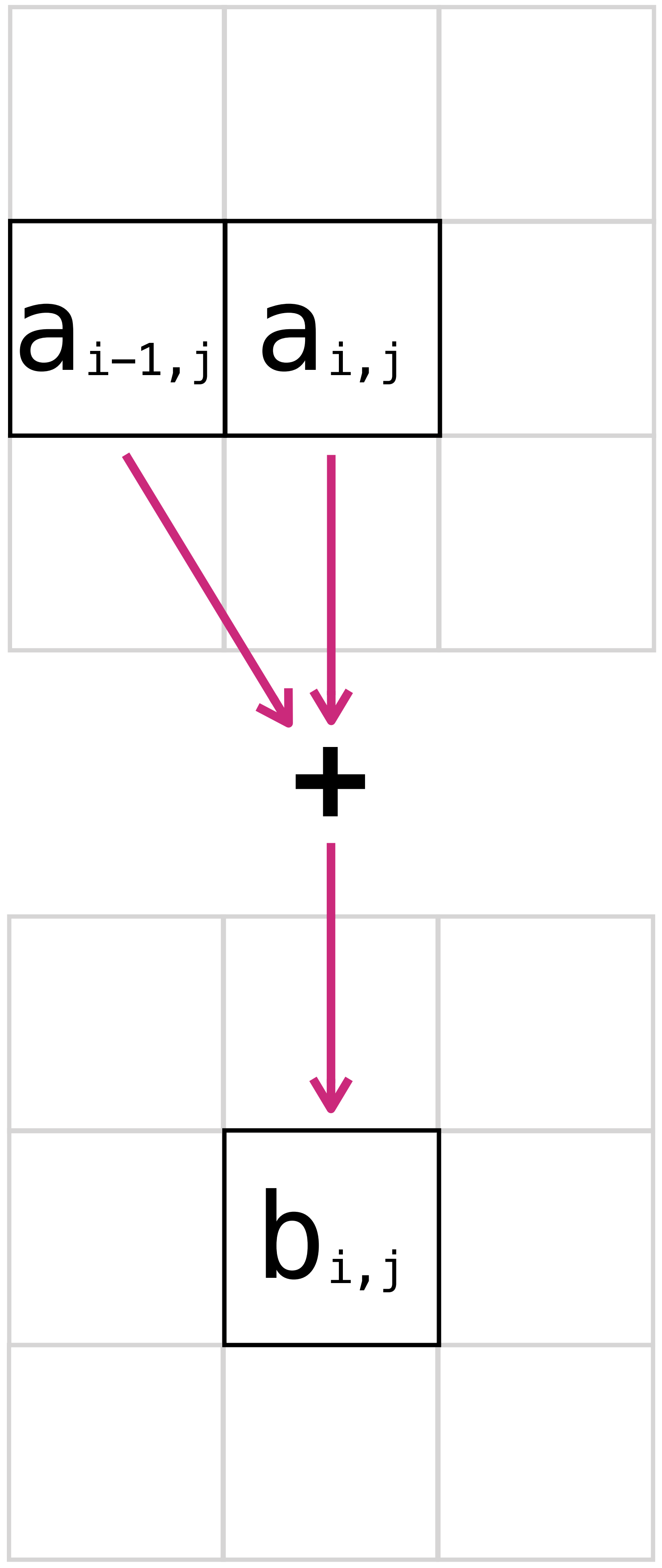
Verify lifted summary

Generate “glue code” +

Halide

# Hoare-style verification conditions

```
integer imin,imax,jmin,jmax,i,j,q,t
real (kind=4), dimension(imin:imax,jmin:jmax) :: a
real (kind=4), dimension(imin:imax,jmin:jmax) :: b
do j = jmin,jmax
    t = a(imin, j)
    do i = imin + 1,imax
        q = a(i,j)
        b(i,j) = q + t
        t = q
    enddo
enddo
```

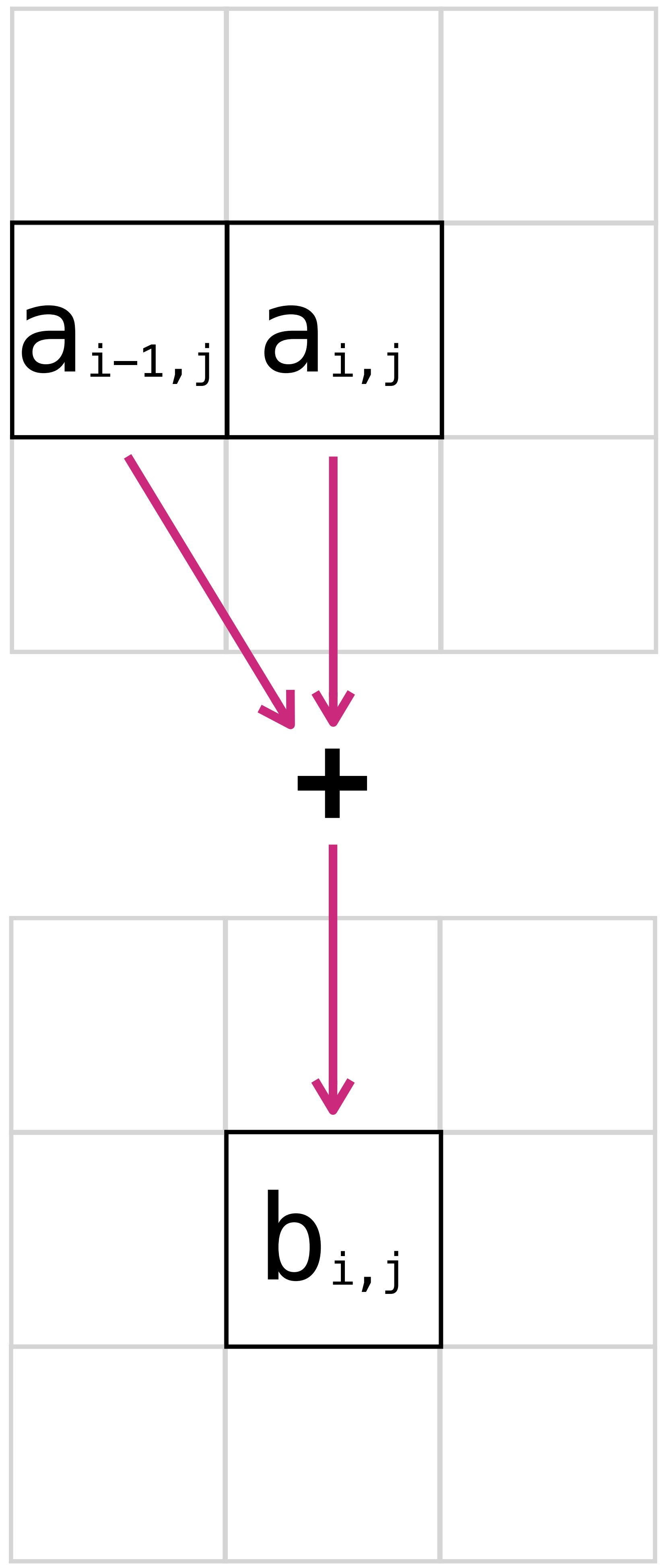



**Verification conditions (VC):**

$$
\begin{array}{ll}
(1) & \forall s.\ pre(s) \rightarrow invariant(s) \\
(2) & \forall s.\ invariant(s) \wedge cond(s) \rightarrow invariant(body(s)) \\
(3) & \forall s.\ invariant(s) \wedge \neg cond(s) \rightarrow post(s)
\end{array}
$$

# Hoare-style verification conditions

```
integer imin,imax,jmin,jmax,i,j,q,t
real (kind=4), dimension(imin:imax,jmin:jmax) :: a
real (kind=4), dimension(imin:imax,jmin:jmax) :: b
do j = jmin,jmax
    t = a(imin, j)
    do i = imin + 1,imax
        q = a(i,j)
        b(i,j) = q + t
        t = q
    enddo
enddo
```

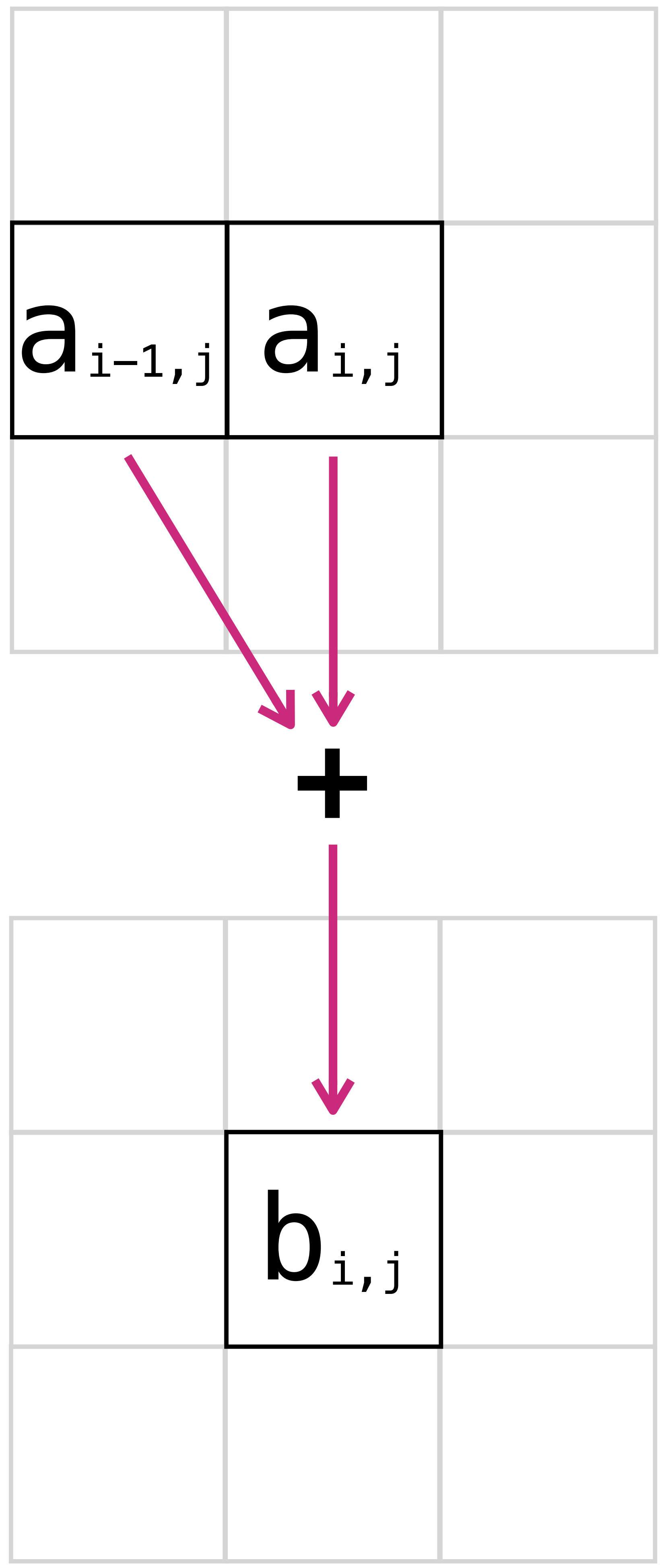



**Outer loop invariant**

$$\forall i, j' \in [imin+1, imax] \times [jmin, j)$$
$$b(i, j') = a(i-1, j') + a(i, j')$$

# Hoare-style verification conditions

```
integer imin,imax,jmin,jmax,i,j,q,t
real (kind=4), dimension(imin:imax,jmin:jmax) :: a
real (kind=4), dimension(imin:imax,jmin:jmax) :: b
do j = jmin,jmax
    t = a(imin, j)
    do i = imin + 1,imax
        q = a(i,j)
        b(i,j) = q + t
        t = q
    enddo
enddo
```

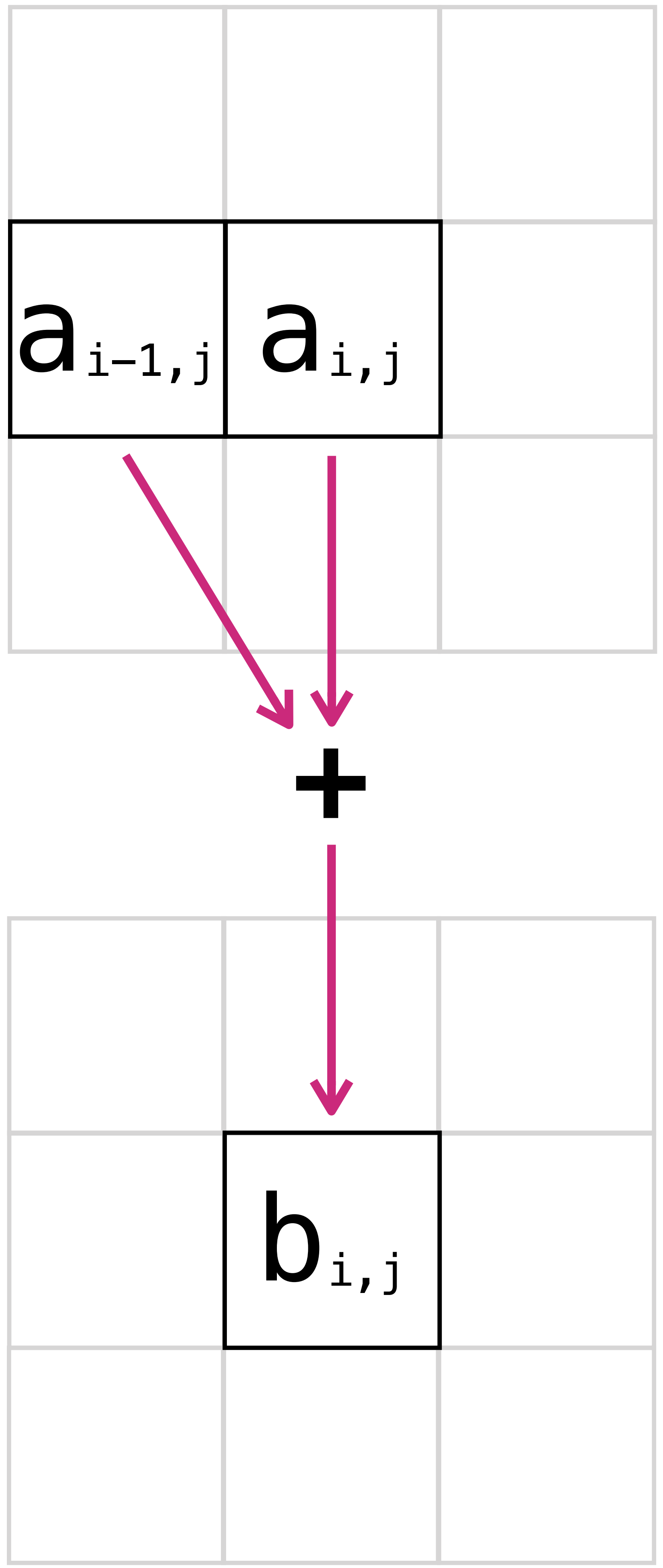



**Inner loop invariant**

$$\forall i, j' \in [imin+1, imax] \times [jmin, j)$$
$$b(i, j') = a(i-1, j') + a(i, j')$$
$$\forall i', j' \in [imin+1, i) \times [j, j)$$
$$b(i', j') = a(i'-1) + a(i', j')$$

# Hoare-style verification conditions

```
integer imin,imax,jmin,jmax,i,j,q,t
real (kind=4), dimension(imin:imax,jmin:jmax) :: a
real (kind=4), dimension(imin:imax,jmin:jmax) :: b
do j = jmin,jmax
    t = a(imin, j)
    do i = imin + 1,imax
        q = a(i,j)
        b(i,j) = q + t
        t = q
    enddo
enddo
```




**Postcondition:**

$$\boxed{\begin{array}{c}\forall i,j \in [imin+1, imax] \times [jmin, jmax] \\ b(i,j) = a(i-1,j) + a(i,j)\end{array}}$$

**High level summary!**

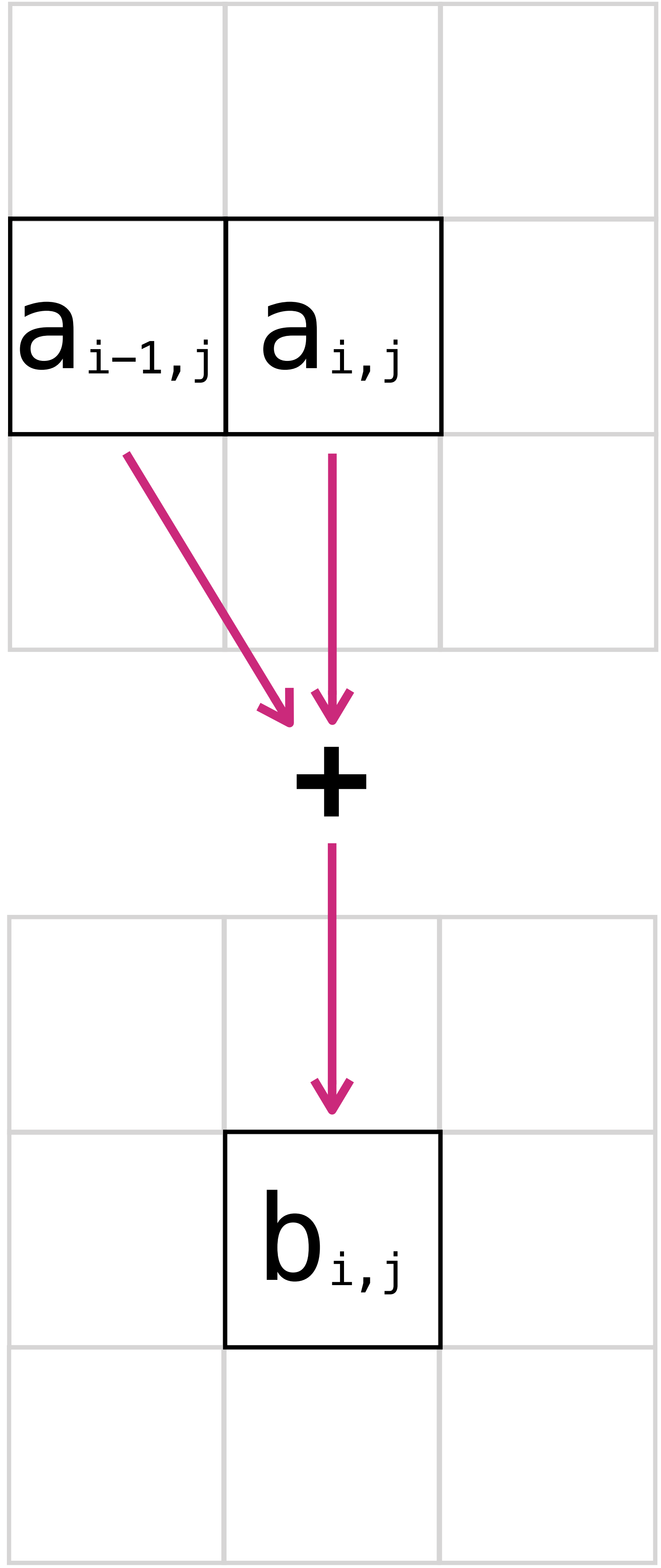
# Postconditions as summaries

Restrictions:

- Conjunction (per output matrix) of:

$$\forall \vec{x} \in dom(b).\ b[\vec{x}] = expr_{halide}(\vec{x})$$

- Right hand side must include non-output term




**Postcondition:**

$$\boxed{\begin{array}{c}\forall i, j \in [imin + 1, imax] \times [jmin, jmax] \\ b(i,j) = a(i-1,j) + a(i,j)\end{array}}$$

**High level summary!**

# Postconditions as summaries

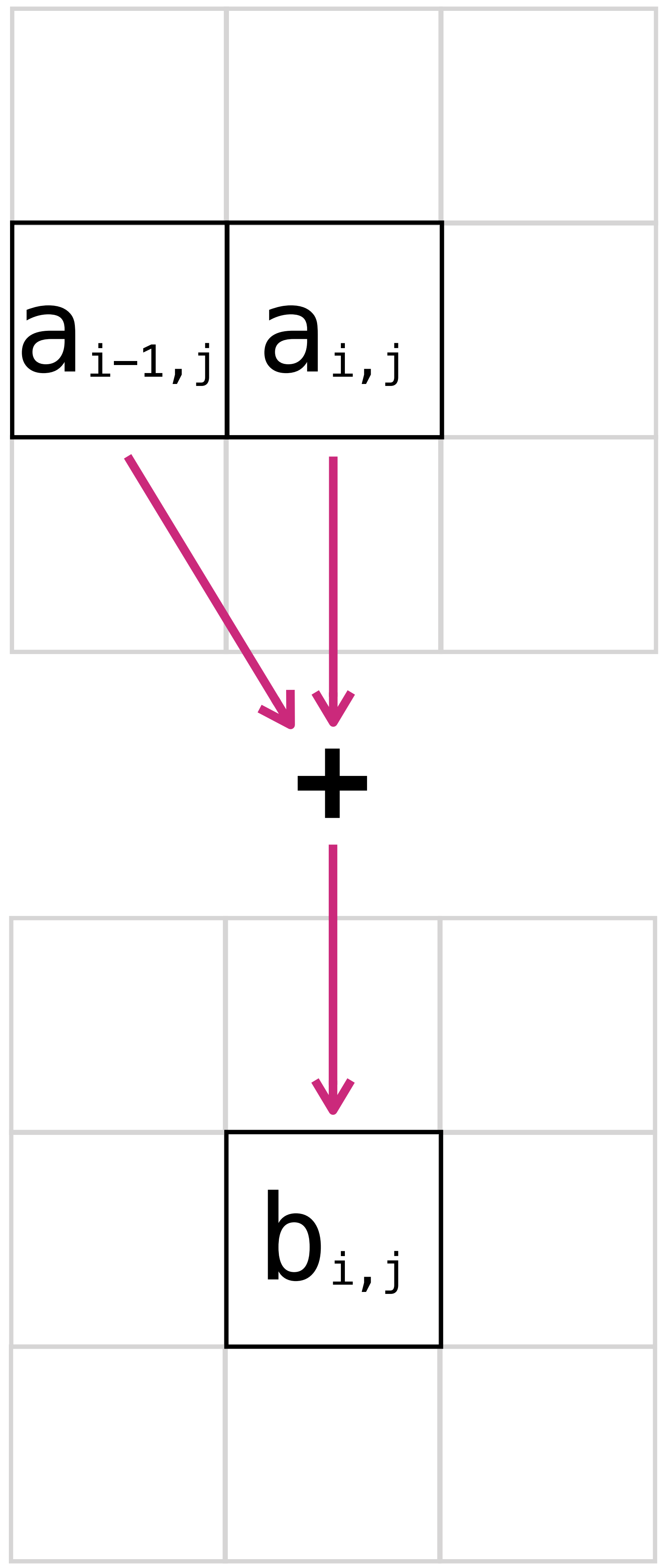
Limitations:

- ❌ Conditionals*
- ❌ Boundary conditions
- ❌ Decrementing loops
- ❌ Calls to impure functions
- ❌ In-place input modification

**Postcondition:**

$$\boxed{\begin{array}{c}\forall i,j \in [imin+1, imax] \times [jmin, jmax] \\ b(i,j) = a(i-1,j) + a(i,j)\end{array}}$$

**High level summary!**



Key idea:

Synthesizing a postcondition is most of the work to lift

**Postcondition:**

$$\boxed{\begin{array}{c}\forall i, j \in [imin + 1, imax] \times [jmin, jmax] \\ b(i, j) = a(i - 1, j) + a(i, j)\end{array}}$$

**High level summary!**

# VC generation as syntax guided synthesis

**Post condition:** $\forall \vec{x} \in dom(b).\ b[\vec{x}] = expr_{halide}(\vec{x})$

**Verification conditions (VC):**

$$
\begin{array}{ll}
\multicolumn{2}{l}{VC(post, invariant, s):} \\
(1) & \forall s.\ pre(s) \rightarrow invariant(s) \\
(2) & \forall s.\ invariant(s) \wedge cond(s) \rightarrow invariant(body(s)) \\
(3) & \forall s.\ invariant(s) \wedge \neg cond(s) \rightarrow post(s)
\end{array}
$$

**Synthesis problem:** $\exists post, invariant.\ \forall s.\ VC(post, invariant, s)$

# Scaling with inductive template generation

```
do j = jmin,jmax
  t = a(imin, j)
  do i = imin + 1,imax
    q = a(i,j)
    b(i,j) = q + t
    t = q
  enddo
enddo
```

↓

```
do j = 1,3
  t = a(1, j)
  do i = 1 + 1,6
    q = a(i,j)
    b(i,j) = q + t
    t = q
  enddo
enddo
```

Motivation:

- Search space still huge after syntax restrictions
- Leverage mixed concrete-symbolic execution
- Set bounds/indices to small, random values

# Scaling with inductive template generation

```
do j = jmin,jmax
  t = a(imin, j)
  do i = imin + 1,imax
    q = a(i,j)
    b(i,j) = q + t
    t = q
  enddo
enddo
```

↓

```
do j = 1,3
  t = a(1, j)
  do i = 1 + 1,6
    q = a(i,j)
    b(i,j) = q + t
    t = q
  enddo
enddo
```

↓

$$b[i,j] = a[hole, hole] + a[hole, hole]$$

| | | | | | |
|---|---|---|---|---|---|
| | a[1,1]+ a[2,1] | a[2,1]+ a[3,1] | a[3,1]+ a[4,1] | a[4,1]+ a[5,1] | a[5,1]+ a[6,1] |
| | a[1,2]+ a[2,2] | a[2,2]+ a[3,2] | a[3,2]+ a[4,2] | a[4,2]+ a[5,2] | a[5,2]+ a[6,2] |
| | a[1,3]+ a[2,3] | a[2,3]+ a[3,3] | a[3,3]+ a[4,3] | a[4,3]+ a[5,3] | a[5,3]+ a[6,3] |

## Anti-unification:

Generalizing from concrete instances to general structure

$$\forall e_1, e_2 \in b. \quad \sqcap(e_1, e_2) := \begin{cases} e_1 & e_1 = e_2 \\ & leaf\,(e_1) \\ \left(op\ \{\sqcap(e_{1i}, e_{2i})\}_i\right) & e_1 = \left(op\ \{e_{1i}\}_i\right) \\ & e_2 = \left(op\ \{e_{2i}\}_i\right) \\ MakeHole(e_1, e_2) & (otherwise) \end{cases}$$

$$op := + \mid - \mid \times \mid /$$

# Scaling with partial Skolemization

Motivation:

- Synthesis is typically a "∃.∀" problem

$$\exists post, invariant.\ \forall s.\ VC(post, invariant, s)$$

- VCs can introduce negation of invariants

**Loop break:** $$\forall s.\ invariant(s) \wedge \neg cond(s) \rightarrow post(s)$$

$$\Downarrow$$

$$\forall s.\ \neg invariant(s) \vee cond(s) \vee post(s)$$

$$\Downarrow$$

$$\forall \mathtt{a}, \mathtt{b}, \mathtt{j}, \mathtt{jmax}.\ \neg I_j(\mathtt{a}, \mathtt{b}, \mathtt{j})\ \vee\ \neg(\mathtt{j} > \mathtt{jmax})\ \vee\ post(\mathtt{a}, \mathtt{b})$$

- ∀ in negated invariant becomes an additional ∃ : "∃.∀.∃" problem

$$I_j(\mathtt{a}, \mathtt{b}, \mathtt{j}) = \begin{array}{c} \forall i, j' \in [imin+1, imax] \times [jmin, j) \\ a\,(i, j') = b\,(i-1, j') + b\,(i, j') \end{array}$$

# Scaling with partial Skolemization

Skolemization:
Producing an equi-satisfiable formula without an existential quantifier

**(Full) Skolemization:**

$$\exists x.\forall y.\exists z.\ (\ldots z \ldots)$$

$$\downarrow$$

$$\exists x.\forall y.\ (\ldots f(y) \ldots)$$
$$f \text{ maps } y \text{ to } z$$

**"Partial" Skolemization:**

$$\exists x.\forall y.\exists z.\ (\ldots z \ldots)$$

$$\downarrow$$

$$f = \begin{cases} f_1 & ? \\ f_2 & ? \end{cases}$$

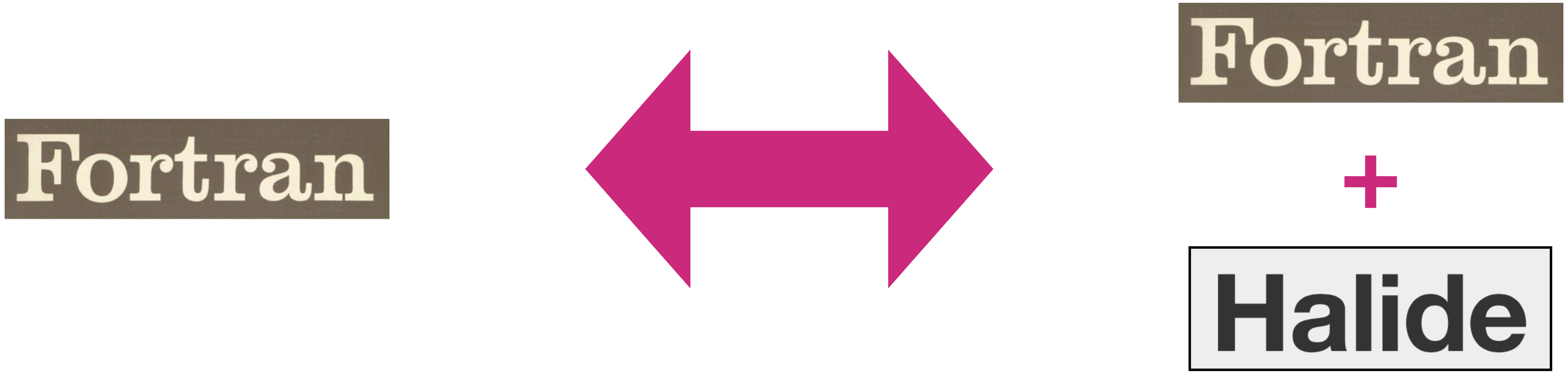
*** note: details not explicated in the paper**

$$\downarrow$$

$$\exists x.\forall y.\ (\ldots f_1(y) \ldots) \vee (\ldots f_2(y) \ldots)$$

# Evaluation

- Compare Fortran code before and after lifting stencil computations
- Compare CPU-CPU and CPU-GPU
- Halide schedules found with 6 hours of OpenTuner



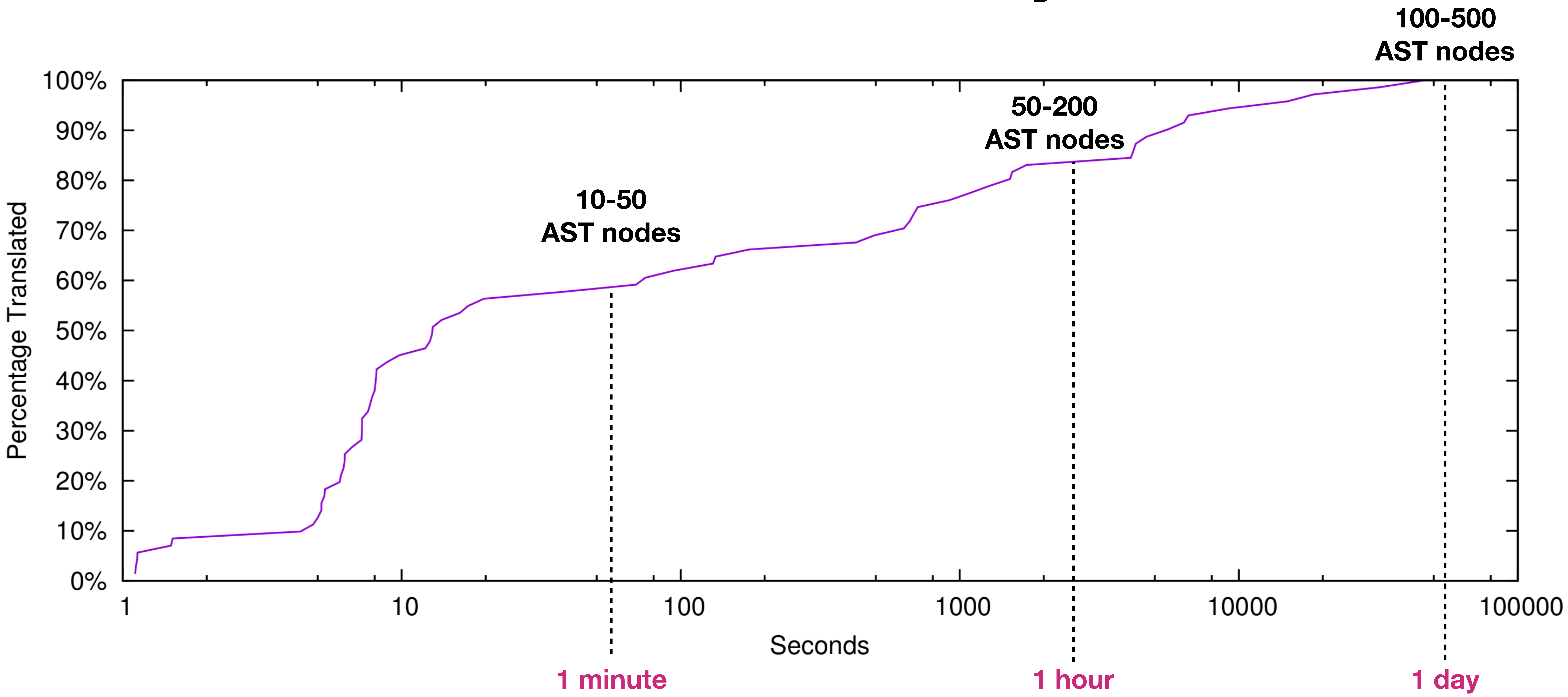

① **Lifted code performance**

② **Synthesizer performance**

# Evaluation

| | Candidates | Success Rate | CPU Speedup | GPU Speedup |
|---|---|---|---|---|
| **StencilMark** (hand-ported μ-benchmarks) | 4 | 75% | 6.5X - 11.3X | Max 4.5X |
| **NAS MG** (solving Poisson equation) | 9 | 33% | 5.7X - 17.5X | Max 3.7X |
| **Cloverleaf** (hydrodynamic Euler eqs) | 45 | 89% | 2.5X - 7.5X | Max 1400X |
| **NFFS-FVM** (fluid mechanics simulation) | 29 | 86% | 3.0X - 24.1X | Max 16.9X |
| **Challenge** (optimized 3D stencils) | 5 | 100% | 1.8X - 12.3X | Max 1.1X |

# Cumulative distribution of synthesis time






github.com/emeryberger/PLDI-2016/raw/master/presentations/pldi16-presentation227.pdf

# Summary

Verified lifting:
Given a stencil computation, find a high-level summary that fully captures the algorithm

Halide

↑

Verified lifted summary

↑

Synthesized verification conditions

↑

Fortran