Halide

A Language and Compiler for Image Processing Pipelines

Jonathan Ragan-Kelley (Stanford, MIT CSAIL)

with Andrew Adams (MIT CSAIL, Google)
Sylvain Paris, Connelly Barnes (Adobe)
Marc Levoy (Stanford, Google)
Saman Amarasinghe, Frédo Durand (MIT CSAIL)

We are surrounded by computational cameras

Enormous opportunity,
demands extreme optimization

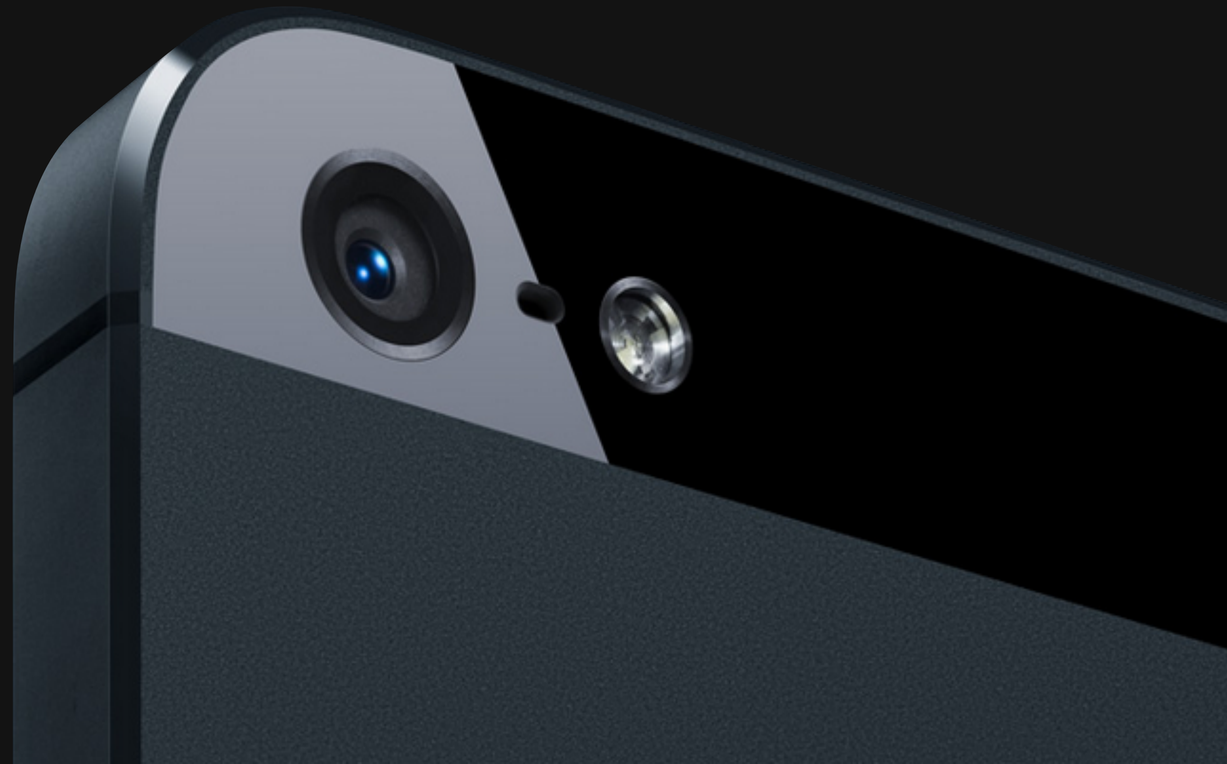
parallelism & locality limit
performance and energy

Camera: 8 Mpixels
(96MB/frame as *float*)

CPUs: 8 GFLOP/sec

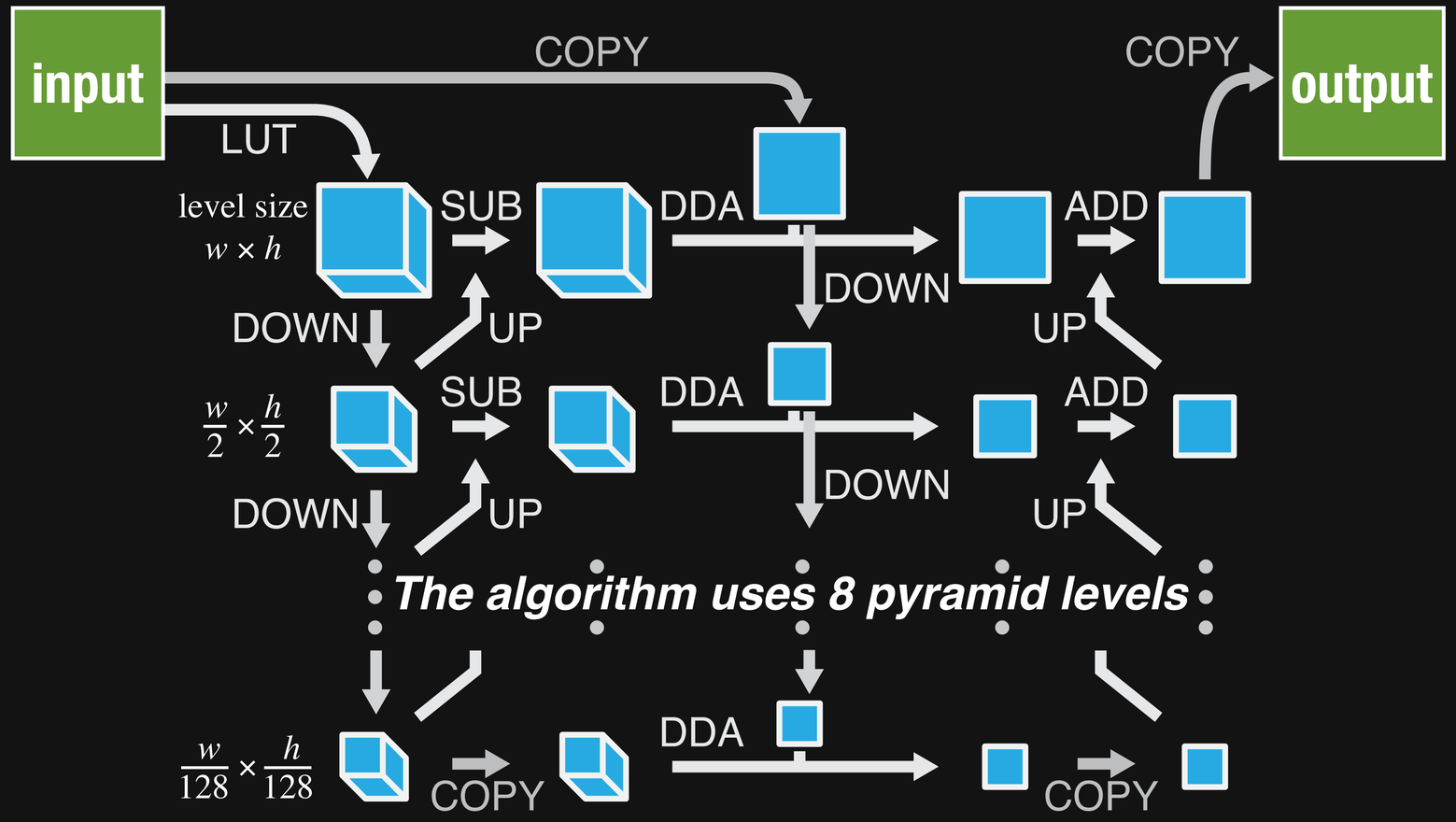
GPU: 10 GFLOP/sec

Required arithmetic intensity > 20:1



A realistic pipeline: Local Laplacian Filters

[Paris et al. 2010, Aubry et al. 2011]



LUT: look-up table

$O(x,y,k) \leftarrow \text{lut}(I(x,y) - k\sigma)$

ADD: addition

$O(x,y) \leftarrow I_1(x,y) + I_2(x,y)$

SUB: subtraction

$O(x,y) \leftarrow I_1(x,y) - I_2(x,y)$

DDA: data-dependent access

$k \leftarrow \text{floor}(I_1(x,y) / \sigma)$

$\alpha \leftarrow (I_1(x,y) / \sigma) - k$

$O(x,y) \leftarrow (1-\alpha)\, I_2(x,y,k) + \alpha\, I_2(x,y,k+1)$

UP: upsample

$T_1(2x,2y) \leftarrow I(x,y)$

$T_2 \leftarrow T_1 \otimes_x [1\ 3\ 3\ 1]$

$O \leftarrow T_2 \otimes_y [1\ 3\ 3\ 1]$

DOWN: downsample

$T_1 \leftarrow I \otimes_x [1\ 3\ 3\ 1]$

$T_2 \leftarrow T_1 \otimes_y [1\ 3\ 3\ 1]$

$O(x,y) \leftarrow T_2(2x,2y)$

wide, deep, heterogeneous

stencils + stream processing

Today's methodology

C++ w/multithreading, SIMD
CUDA/OpenCL
OpenGL/RenderScript

Optimization requires manually
transforming program & data structure
for locality and parallelism.

libraries don't solve this:
BLAS, IPP, MKL, OpenCV
optimized kernels compose into
inefficient pipelines (no fusion)

Halide

a new language & compiler for image processing

1. **Decouple *algorithm* from *schedule***

   Algorithm: *what* is computed
   Schedule: *where* and *when* it's computed

2. **Single, unified model for *all* schedules**

   Simple enough to search, expose to user
   Powerful enough to beat expert-tuned code

Halide

0.9 ms/megapixel

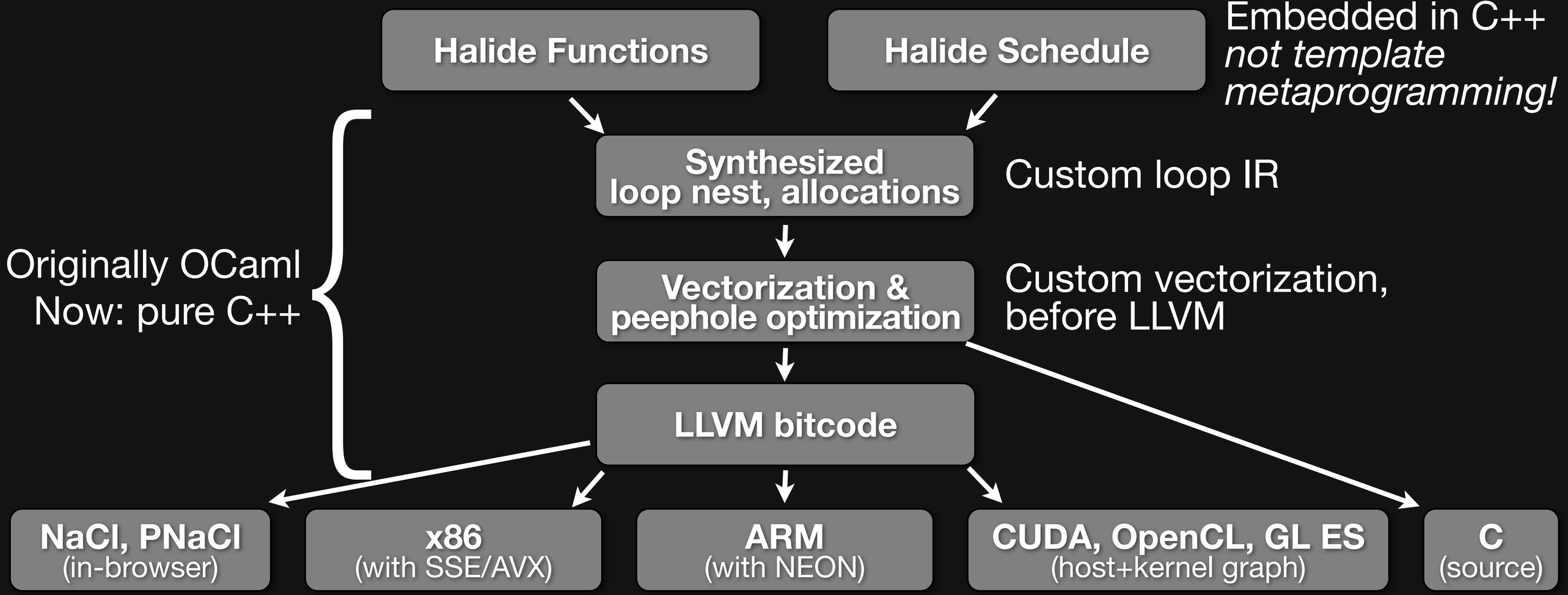
```
Func box_filter_3x3(Func in) {
  Func blurx, blury;
  Var x, y, xi, yi;

  // The algorithm - no storage, order
  blurx(x, y) = (in(x-1, y) + in(x, y) + in(x+1, y))/3;
  blury(x, y) = (blurx(x, y-1) + blurx(x, y) + blurx(x, y+1))/3;

  // The schedule - defines order, locality; implies storage
  blury.tile(x, y, xi, yi, 256, 32)
       .vectorize(xi, 8).parallel(y);
  blurx.compute_at(blury, x).store_at(blury, x).vectorize(x, 8);

  return blury;
}
```

The Halide Compiler



Current status

open source at http://halide-lang.org

Google

~ **30** developers

> **10 kLOC** in production

G+ Photos *auto-enhance*
Data center
Android
Chrome (PNaCl)

HDR+
Glass
Nexus devices

Computational photography course (6.815)
60 undergrads

Local Laplacian Filters

prototype for Adobe Photoshop Camera Raw / Lightroom

Reference: 300 lines C++

Adobe: 1500 lines
3 months of work
10x faster *(vs. reference)*

Halide: 60 lines
1 intern-day

20x faster (vs. reference)
2x faster (vs. Adobe)

GPU: 90x faster (vs. reference)
9x faster (vs. Adobe)