Automatically Scheduling Halide Image Processing Pipelines

Ravi Teja Mullapudi (CMU)
Andrew Adams (Google)
Dillon Sharlet (Google)
Jonathan Ragan-Kelley (Stanford)
Kayvon Fatahalian (CMU)
High demand for efficient image processing
Scheduling image processing algorithms

Algorithm description

Var x, y;
Func f, g;
g(x,y) = f(x,y) + ...
h(x) = g(x,y) + ...
Scheduling image processing algorithms

Algorithm description

```c
Var x, y;
Func f, g;
g(x,y) = f(x,y) + ...
h(x) = g(x,y) + ...
```

Schedule (machine mapping)

- parallelize y loop
- tile output dims
- vectorize y loop

Implementation
Scheduling image processing algorithms

Image processing algorithm developers

Algorithm description

Var x, y;
Func f, g;
g(x,y) = f(x,y) + ...
h(x) = g(x,y) + ...

Schedule (machine mapping)

parallelize y loop
tile output dims
vectorize y loop

Implementation
Few developers have the skill set to author highly optimized schedules

Image processing algorithm developers

Algorithm description

Var x, y;
Func f, g;
g(x,y) = f(x,y) + ...
h(x) = g(x,y) + ...

Schedule (machine mapping)

parallelize y loop
tile output dims
vectorize y loop

> 10x Faster Implementation
Contribution: automatic scheduling of image processing pipelines

Image processing algorithm developers

Algorithm description

Var x, y;
Func f, g;
g(x,y) = f(x,y) + ...
h(x) = g(x,y) + ...

Generates expert-quality schedules in seconds

> 10x Faster Implementation

Scheduling Algorithm
Why is it challenging to schedule image processing pipelines?
Algorithm: 3x3 box blur
Algorithm: 3x3 box blur

\[ bx(x, y) = \frac{(in(x-1, y) + in(x, y) + in(x+1, y))}{3} \]
Algorithm: 3x3 box blur

\[
bx(x, y) = \frac{\text{in}(x-1, y) + \text{in}(x, y) + \text{in}(x+1, y)}{3}
\]

\[
\text{out}(x, y) = \frac{bx(x, y-1) + bx(x, y) + bx(x, y+1)}{3}
\]
A basic (slow) schedule

compute all pixels of bx, in parallel
compute all pixels of by, in parallel
A basic (slow) schedule

compute all pixels of bx, in parallel
compute all pixels of by, in parallel
A basic (slow) schedule

compute all pixels of bx, in parallel
compute all pixels of by, in parallel
A basic (slow) schedule

compute all pixels of bx, in parallel
compute all pixels of by, in parallel

in

Intermediate buffer

bx

out
A basic (slow) schedule

compute all pixels of bx, in parallel
compute all pixels of by, in parallel
A basic (slow) schedule

1. Compute all pixels of bx, in parallel.
2. Compute all pixels of by, in parallel.

Diagram:
- 'in' input grid
- 'bx' intermediate buffer
- 'out' output grid
Low performance: bandwidth bound

Large in-memory buffer

in → bx → out
Tiling to improve data locality

For each 3x3 tile, in parallel compute required pixels of bx compute pixels of out in tile
Tiling to improve data locality

For each 3x3 tile, compute required pixels of bx and compute pixels of out in the tile.
Tiling to improve data locality

For each 3x3 tile, in parallel compute required pixels of bx compute pixels of out in tile
Tiling to improve data locality

Intermediate buffer: fits in fast on-chip storage

for each 3x3 tile, in parallel compute required pixels of bx compute pixels of out in tile
Tiling to improve data locality

for each 3x3 tile, in parallel compute required pixels of bx compute pixels of out in tile
Tiling to improve data locality

for each 3x3 tile, in parallel
compute required pixels of bx
compute pixels of out in tile
Tiling to improve data locality

for each 3x3 tile, in parallel compute required pixels of bx
compute pixels of out in tile
Tiling to improve data locality

for each 3x3 tile, in parallel compute required pixels of bx compute pixels of out in tile
Tiling introduces redundant work
Tiling introduces redundant work

Pixels computed twice
Tiling introduces redundant work

Pixels computed twice
Larger tiles reduce redundant work

for each 3x6 tile, in parallel
compute required pixels of bx
compute pixels in tile of out
Goal: balance parallelism, locality, work

for each 3x6 tile, in parallel compute required pixels of bx compute pixels in tile of out
Goal: balance parallelism, locality, work

for each 3x6 tile, in parallel compute required pixels of bx compute pixels in tile of out
Represent image processing pipelines as graphs

DAG representation of the two-stage blur pipeline
Real world pipelines are complex graphs

Local Laplacian filters
[Paris et al. 2010, Aubry et al. 2011]

100 stages

Google Nexus HDR+ mode: over 2000 stages!
Key aspects of scheduling
Key aspects of scheduling

Deciding which stages to interleave for better data locality
Key aspects of scheduling

Deciding which stages to interleave for better data locality

Picking tiles sizes to trade-off locality and re-computation
Key aspects of scheduling

Deciding which stages to interleave for better data locality

Picking tiles sizes to trade-off locality and re-computation

Maintain ability to execute in parallel
An Algorithm for Scheduling Image Processing Pipelines
Algorithm

Input: DAG of pipeline stages
Algorithm

Input: DAG of pipeline stages

Output: Optimized schedule

for each 8x128 tile in parallel
compute required pixels of A
compute pixels in tile of B

for each 8x8 tile in parallel
compute required pixels of C
compute required pixels of D
compute pixels in tile of E
Algorithm

Input: DAG of pipeline stages

Output: Optimized schedule

- for each 8x128 tile in parallel
  - compute required pixels of A
  - compute pixels in tile of B

- for each 8x8 tile in parallel
  - compute required pixels of C
  - compute required pixels of D
  - compute pixels in tile of E
Algorithm

Input: DAG of pipeline stages

Output: Optimized schedule

- For each 8x128 tile in parallel, compute required pixels of A and compute pixels in tile of B.
- For each 8x8 tile in parallel, compute required pixels of C, required pixels of D, and compute pixels in tile of E.
Algorithm

Input: DAG of pipeline stages

Output: Optimized schedule

for each 8x128 tile in parallel
compute required pixels of A
compute pixels in tile of B

for each 8x8 tile in parallel
compute required pixels of C
compute required pixels of D
compute pixels in tile of E

Tile size: 8 x 128
Tile size: 8 x 8
Scheduling the DAG for better locality

Determine which stages to group together?

How to tile stages in each group?
When to group stages?

Grouping A and B together can either improve or degrade performance.
Quantifying the cost of a group

Tile size: 3 x 3

for each 3x3 tile in parallel
compute required pixels of A
compute pixels in tile of B
compute all pixels of C, in parallel
compute all pixels of D, in parallel
compute all pixels of E, in parallel

Cost = Cost of arithmetic + Cost of memory
Quantifying the cost of a group

Cost = (Number of arithmetic operations) + (Number of memory accesses) x (LOAD COST)

for each 3x3 tile in parallel
compute required pixels of A
compute pixels in tile of B
compute all pixels of C, in parallel
compute all pixels of D, in parallel
compute all pixels of E, in parallel
Quantifying the cost of a group

Cost = (Number of arithmetic operations) +
(Number of memory accesses) x (LOAD COST)

Tile size: 3 x 3

for each 3x3 tile in parallel
compute required pixels of A
compute pixels in tile of B
Estimating cost using interval analysis

Tile size: 3 x 3

\[
\text{Cost} = (\text{Number of arithmetic operations}) + (\text{Number of memory accesses}) \times (\text{LOAD COST})
\]
Estimating cost using interval analysis

Cost = (Number of arithmetic operations) + (Number of memory accesses) x (LOAD COST)
Estimating cost using interval analysis

Cost = (Number of arithmetic operations) + (Number of memory accesses) x (LOAD COST)
Estimating cost using interval analysis

Cost = (Number of arithmetic operations) + (Number of memory accesses) x (LOAD COST)
Estimating cost using interval analysis

Cost = (Number of arithmetic operations) + (Number of memory accesses) x (LOAD COST)
Estimating cost using interval analysis

Cost = Number of tiles \times \text{Cost per tile}
Search for best tile sizes

Tile size: 1 x 6
Search for best tile sizes

Tile size: 6 x 1
Search for best tile sizes

Tile size: 2 x 2
When to group stages?

Benefit(\(A, B\)) = Cost(\(A\)) + Cost(\(B\)) - Cost(\(A, B\))
Exhaustive search is infeasible

Exponential number of possible groupings
Greedy grouping algorithm

compute all pixels of A, in parallel
compute all pixels of B, in parallel
compute all pixels of C, in parallel
compute all pixels of D, in parallel
compute all pixels of E, in parallel
Greedy grouping algorithm

compute all pixels of A, in parallel
compute all pixels of B, in parallel
compute all pixels of C, in parallel
compute all pixels of D, in parallel
compute all pixels of E, in parallel
Greedy grouping algorithm

- Compute all pixels of A, in parallel.
- Compute all pixels of B, in parallel.
- Compute all pixels of C, in parallel.
- Compute all pixels of D, in parallel.
- Compute all pixels of E, in parallel.
Greedy grouping algorithm

- Compute all pixels of A, in parallel
- Compute all pixels of B, in parallel
- Compute all pixels of D, in parallel

For each 8x8 tile in parallel:
- Compute required pixels of C
- Compute pixels in tile of E

Tile size: 8 x 8
Greedy grouping algorithm

for each 8x128 tile in parallel
compute required pixels of A
compute pixels in tile of B

compute all pixels of D, in parallel

for each 8x8 tile in parallel
compute required pixels of C
compute pixels in tile of E

Tile size: 8 x 128
Tile size: 8 x 8
Greedy grouping algorithm

- For each 8x128 tile in parallel:
  - Compute required pixels of A
  - Compute pixels in tile of B
- For each 8x8 tile in parallel:
  - Compute required pixels of C
  - Compute required pixels of D
  - Compute pixels in tile of E

Tile size: 8 x 128
Tile size: 8 x 8
Auto scheduler implementation details

- Multi-core parallelism, vectorization, loop reordering, and unrolling

for each 8x128 tile in parallel
  vectorize compute required pixels of A unroll x by 4
  vectorize compute required pixels of B
  vectorize compute pixels in tile of D

for each 8x8 tile in parallel
  vectorize compute required pixels of C unroll y by 2
  vectorize compute pixels in tile of E
Evaluation
### Benchmarks of varying complexity and structure

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Stages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blur</td>
<td>3</td>
</tr>
<tr>
<td>Unsharp mask</td>
<td>9</td>
</tr>
<tr>
<td>Harris corner detection</td>
<td>13</td>
</tr>
<tr>
<td>Camera RAW processing</td>
<td>30</td>
</tr>
<tr>
<td>Non-local means denoising</td>
<td>13</td>
</tr>
<tr>
<td>Max-brightness filter</td>
<td>9</td>
</tr>
<tr>
<td>Multi-scale interpolation</td>
<td>52</td>
</tr>
<tr>
<td>Local-laplacian filter</td>
<td>103</td>
</tr>
<tr>
<td>Synthetic depth-of-field</td>
<td>74</td>
</tr>
<tr>
<td>Bilateral filter</td>
<td>8</td>
</tr>
<tr>
<td>Histogram equalization</td>
<td>7</td>
</tr>
<tr>
<td>VGG-16 deep network eval</td>
<td>64</td>
</tr>
</tbody>
</table>
## Auto scheduler generates schedules in seconds

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Stages</th>
<th>Compile time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blur</td>
<td>3</td>
<td>&lt;1</td>
</tr>
<tr>
<td>Unsharp mask</td>
<td>9</td>
<td>&lt;1</td>
</tr>
<tr>
<td>Harris corner detection</td>
<td>13</td>
<td>&lt;1</td>
</tr>
<tr>
<td>Camera RAW processing</td>
<td>30</td>
<td>&lt;1</td>
</tr>
<tr>
<td>Non-local means denoising</td>
<td>13</td>
<td>&lt;1</td>
</tr>
<tr>
<td>Max-brightness filter</td>
<td>9</td>
<td>&lt;1</td>
</tr>
<tr>
<td>Multi-scale interpolation</td>
<td>52</td>
<td>2.6</td>
</tr>
<tr>
<td>Local-laplacian filter</td>
<td>103</td>
<td>3.9</td>
</tr>
<tr>
<td><strong>Synthetic depth-of-field</strong></td>
<td>74</td>
<td><strong>55</strong></td>
</tr>
<tr>
<td>Bilateral filter</td>
<td>8</td>
<td>&lt;1</td>
</tr>
<tr>
<td>Histogram equalization</td>
<td>7</td>
<td>&lt;1</td>
</tr>
<tr>
<td>VGG-16 deep network eval</td>
<td>64</td>
<td>6.9</td>
</tr>
</tbody>
</table>
Auto scheduler performs comparably to experts

Performance relative to experts (6 core Xeon CPU)
Auto scheduler performs comparably to experts

On 8 of the 14 benchmarks performance within 10% of experts or better

Performance relative to experts (6 core Xeon CPU)
## Auto scheduler performs comparably to experts

Auto scheduler performs comparably to experts on 8 of the 14 benchmarks. Baseline schedules exploit multi-core and vector parallelism but no grouping.

### Performance relative to experts (6 core Xeon CPU)

<table>
<thead>
<tr>
<th>Task</th>
<th>Auto scheduler</th>
<th>Baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bilateral grid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blur</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Camera pipe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Convolution layer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Harris corner</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Histogram equal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mscale interpolate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lens blur</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Local laplacian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Matrix multiply</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max filter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non-local means</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unsharp mask</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VGG-16 evaluation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

On 8 of the 14 benchmarks, performance is within 10% of experts or better.

Baseline schedules exploit multi-core and vector parallelism but no grouping.

*Performance relative to experts (6 core Xeon CPU)*
Auto scheduler can save time for experts

Throughput vs. Time (min) for:
- Non-local means
- Lens blur
- Max filter

- Dillon
- Andrew
Auto scheduler can save time for experts

![Graphs showing throughput over time for different filters: Non-local means, Max filter, Lens blur. The graphs compare the performance of Auto scheduler, Dillon, and Andrew.](image)
Exploring cost model parameters

<table>
<thead>
<tr>
<th>Component</th>
<th>Auto scheduler</th>
<th>3-day auto tuning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bilateral grid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blur</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Camera pipe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Convolution layer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Harris corner</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Histogram equal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mscale interpolate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lens blur</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Local laplacian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Matrix multiply</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max filter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non-local means</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unsharp mask</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VGG-16 evaluation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Performance relative to experts (6 core Xeon CPU)
Exploring cost model parameters

Performance relative to experts (6 core Xeon CPU)
Quad core ARM performance

Performance relative to experts (ARM CPU)
K40 GPU performance

Performance relative to experts (K40)
K40 GPU performance

Performance relative to experts (K40)
Prior work

Optimizing Halide via auto-tuning and stochastic search [Ragan-Kelley 13, Ansel 14]:
- Compilation time: hours to days
- Output up to 5-10x slower than hand-tuned implementations

Darkroom [Hegarty 14]:
- Auto-scheduling assuming applications restricted to fixed-size stencils

PolyMage [Mullapudi 15]: polyhedral-based optimization
- Greedy group-and-tile algorithm was inspired by PolyMage
- Polyhedral approach cannot analyze non-affine and data-dependent computations
Limitations

Restricted space of schedules
- Does not consider sliding windows and multi-level tiling

No human interaction with the auto scheduler
- Enable experts to guide the scheduling process
Summary

Algorithm that generates Halide schedules
- Competitive with experts
- Generated in seconds
- Practical implementation

In the process of being merged into mainline Halide
https://github.com/halide/Halide/tree/auto_scheduler
Generalizing the auto scheduler for other DSLs

Abstract analysis and scheduling techniques into components that can be used across languages
Thank you

https://github.com/halide/Halide/tree/auto_scheduler