Halide

By: *Shameer, Gurkirat, Marcus, & Jonathan*

What is Halide?

- Halide is a domain-specific language (DSL) that's embedded in C++
- Designed for image processing, computer vision, and scientific computation
- Algorithms are efficient and achieve optimal performance across hardware platforms

Why was Halide created?

- Halide was developed to solve the trade-off between productivity and performance.
- Writing fast image processing code in C++ is very difficult and requires manual optimization for different hardware, which is time-consuming and prone to errors
- Halide allows developers to write clean, concise algorithms

History of Halide

- Created in 2012 at MIT, helped by Adobe and Google
- Prominent creators are Jonathan Ragan-Kelley and Andrew Adams
- The name “Halide” comes from silver halides which are a light sensitive compound used in photography
- The 2013 Research Paper “*Decoupling Algorithms from Schedules for Easy Optimization of Image Processing Pipelines”* won the ACM SIGPLAN award for best paper

What is Halide Used For?

- Halide is primarily used in applications that require high-speed, complex image and signal processing
- **Computational Photography:** Noise reduction, HDR imaging, panorama stitching, and focus stacking
- **Machine Learning:** Efficient implementation of layers in deep neural networks, particularly on specialized hardware
- **Scientific and Medical Imaging:** Processing large datasets for research and diagnostic purposes
- **Video Processing:** Real-time video filters and effects
- **Real-world examples:** Adobe Photoshop, Google's HDR+ feature on Pixel phones

Strengths and Weaknesses

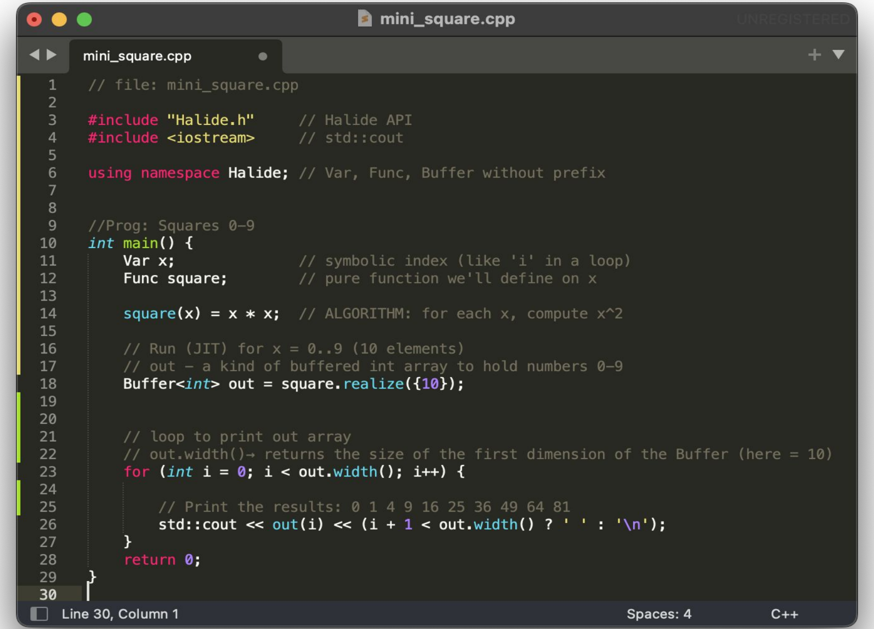
Strengths

- **Performance:** Halide can generate code that is often much faster than general-purpose languages
- **Productivity:** Separating algorithm from schedule improves code simplicity and developers can quickly experiment with different optimization strategies
- **Portability:** The same algorithm can be compiled for different hardware by simply changing the schedule

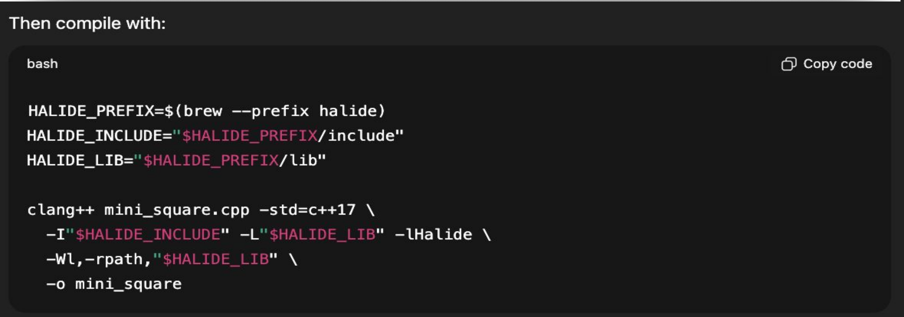
Weaknesses

- **Steep Learning Curve:** The concept of scheduling and understanding Halide's optimization model can be challenging
- **Not a General-Purpose Language:** It's specialized for array and image processing and not suitable for general-purpose programming tasks
- **Debugging:** Debugging can be difficult because the generated machine code is highly transformed from the source

Implementation



```cpp
// file: mini_square.cpp

#include "Halide.h"      // Halide API
#include <iostream>      // std::cout

using namespace Halide; // Var, Func, Buffer without prefix


//Prog: Squares 0-9
int main() {
    Var x;               // symbolic index (like 'i' in a loop)
    Func square;         // pure function we'll define on x

    square(x) = x * x;  // ALGORITHM: for each x, compute x^2

    // Run (JIT) for x = 0..9 (10 elements)
    // out - a kind of buffered int array to hold numbers 0-9
    Buffer<int> out = square.realize({10});


    // loop to print out array
    // out.width()→ returns the size of the first dimension of the Buffer (here = 10)
    for (int i = 0; i < out.width(); i++) {

        // Print the results: 0 1 4 9 16 25 36 49 64 81
        std::cout << out(i) << (i + 1 < out.width() ? ' ' : '\n');
    }
    return 0;
}

```



Then compile with:

```bash
HALIDE_PREFIX=$(brew --prefix halide)
HALIDE_INCLUDE="$HALIDE_PREFIX/include"
HALIDE_LIB="$HALIDE_PREFIX/lib"

clang++ mini_square.cpp -std=c++17 \
  -I"$HALIDE_INCLUDE" -L"$HALIDE_LIB" -lHalide \
  -Wl,-rpath,"$HALIDE_LIB" \
  -o mini_square
```

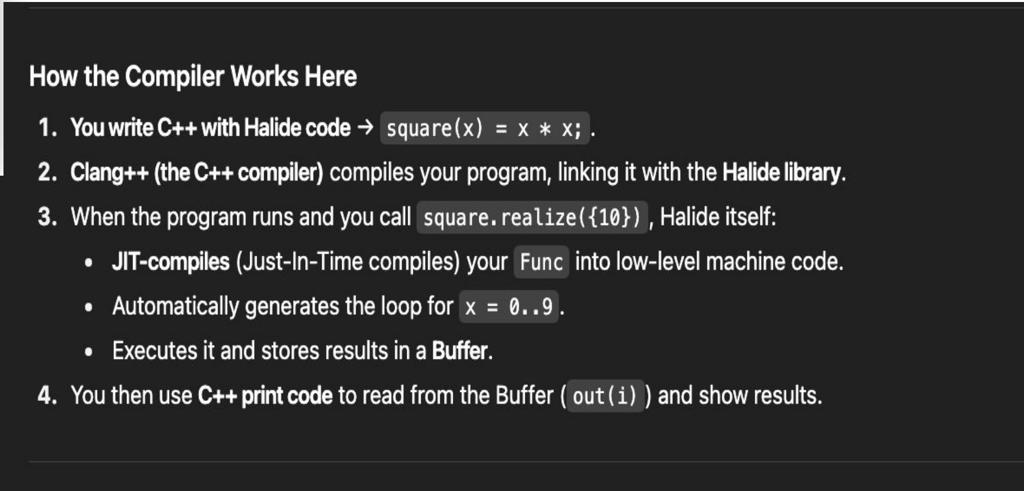
Squares 0-9:

square.realize({10}):

- square is the defined Halide Func (a pure formula).
- .realize({10}) means: “evaluate this function over a domain of size 10.”
  - That is, compute values for x = 0, 1, 2, ..., 9.
- The result is returned as a Halide Buffer (a kind of array object).

out.width():

- out is a Buffer (Halide’s array type).
- “**.width()**” gives the size of the first dimension (the x-dimension).
  - In a 1-D example, .width() is just the length of the array.
  - In a 2-D buffer (like an image), .width() = number of columns, .height() = number of rows.



How the Compiler Works Here

1. **You write C++ with Halide code** → `square(x) = x * x;` .
2. **Clang++ (the C++ compiler)** compiles your program, linking it with the **Halide library**.
3. When the program runs and you call `square.realize({10})` , Halide itself:
   - **JIT-compiles** (Just-In-Time compiles) your `Func` into low-level machine code.
   - Automatically generates the loop for `x = 0..9` .
   - Executes it and stores results in a **Buffer**.
4. You then use **C++ print code** to read from the Buffer (`out(i)`) and show results.

Conclusion

- Halide is a powerful tool for developing high-performance image and array processing pipelines
- Its core principle of separating algorithm from schedule allows developers to write clean, productive code while still achieving optimal performance
- While it has a learning curve and is not for general-purpose tasks, it is an invaluable tool for specialists in computational photography, computer vision, and machine learning.