Exploring Halide library/language with demonstrations

Halide- An introduction:

Halide is a programming language and compiler friendly that focuses on efficiently expressing and compiling array computation for image processing, computer vision, scientific computation, and machine learning. It has a simple syntax and powerful code optimizations to achieve ease of use and high performance. It also includes features for automatic gradient generation for machine learning and optimization applications.

Halide Vs PyTorch:

Halide is 27.8 times faster on the CPU and 7.5 times faster on the GPU. Halide is not meant to be a replacement for PyTorch or Tensorflow. It is a complement to them. It is used in cases like where an efficient implementation in PyTorch or Tensorflow is required but difficult. One must consider using Halide before diving into the low-level C++/CUDA implementation.

Halide is embedded language:

Halide is embedded in C++. This means that we write code in C++ but the code builds an in-memory representation of a Halide pipeline using Halide's C++ API. We can then compile this representation to an object file, or JIT-compile it and run it in the same process.

Advantages of Halide:

- The syntax of Halide is easy.and cross platform.
- It produces very fast executables.
- Halide supports many targets like x86,ARM, GPU.
- It's quite easy to embed it in C++.

Disadvantages of Halide:

- Halide is not turing complete. So, it can't represent everything.
- Everytime, it doesn't generate useful error messages.

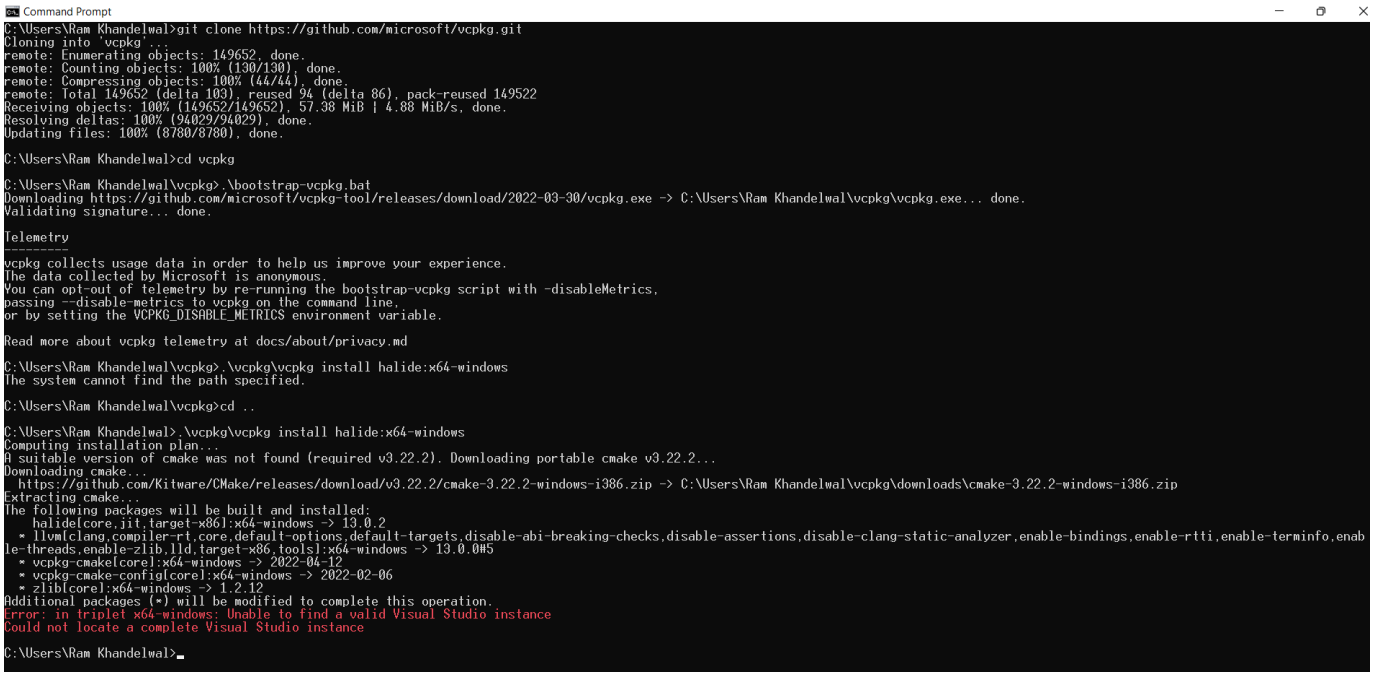
Installing Halide:

- First, Download the Visual Studio Professional 2022 from Microsoft .
- Then make sure to select Desktop development with C++ option while installing..
- Now, Download the latest binary release of halide file for Windows.
  [https://github.com/halide/Halide/releases
- Now run the below command in windows command prompt for installing vcpkg.

```
git clone https://github.com/microsoft/vcpkg
```

- Next, run the below command in command prompt.
  ```
  .\vcpkg\bootstrap-vcpkg.bat
  ```
- After that, run the below command to install the libraries

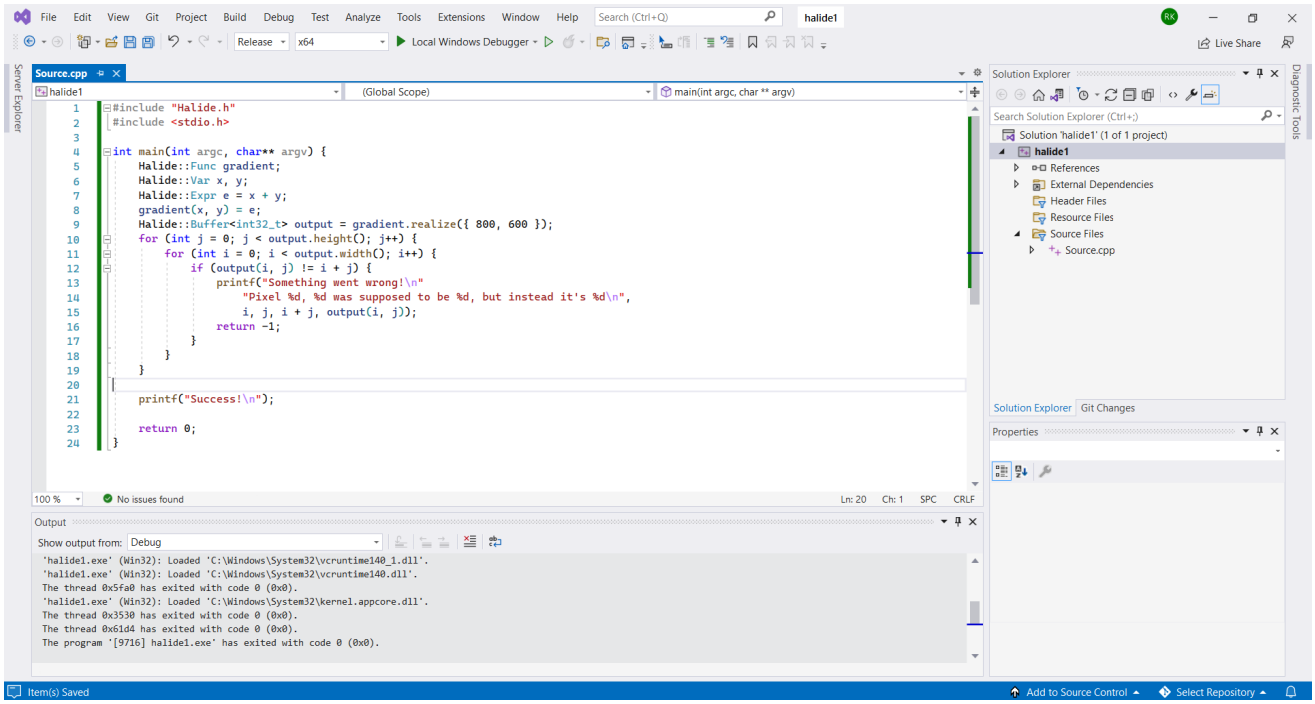
  .\vcpkg\vcpkg install halide
- This marks the end of halide setup .



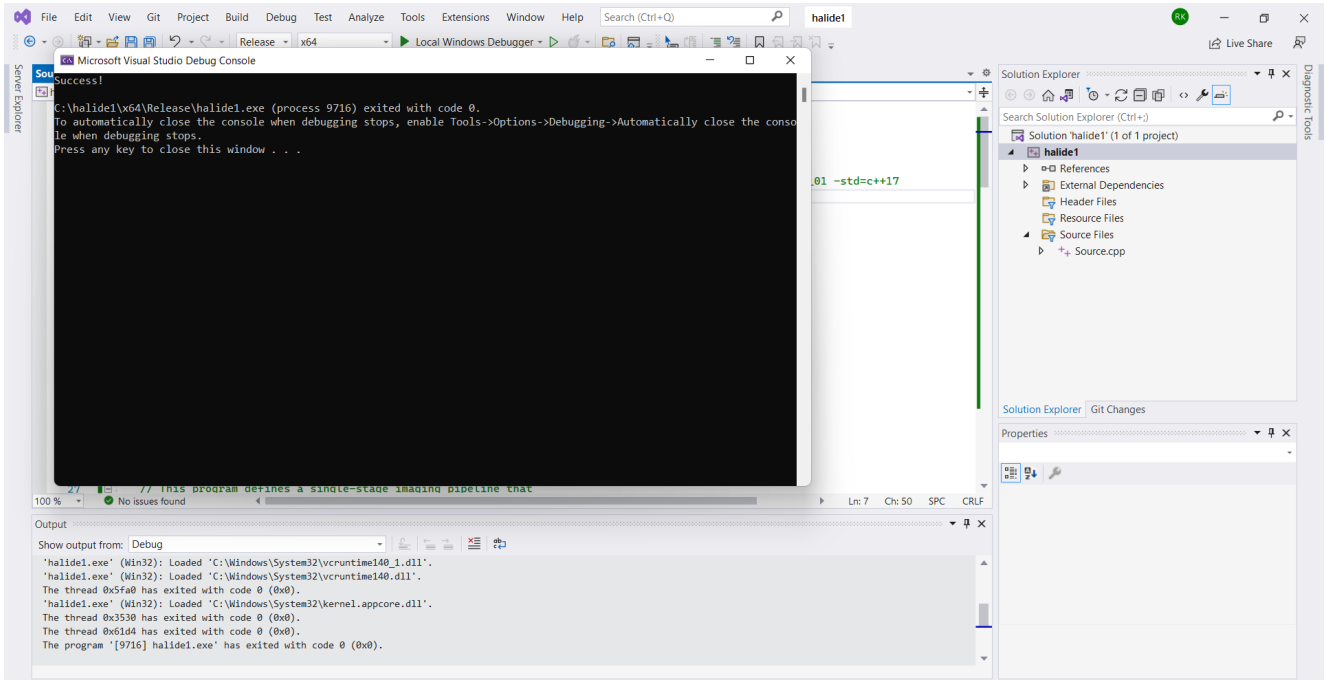
Main Elements of Halide:

- Variables are represented by Keyword **Var**.
- Functions are denoted by Keyword **Func**. These are defined over integer domain.
- Expressions are denoted by the Keyword **Expr**. These are combination of Halide Functions and Variables.
- Images are represented by the Keyword **Image**. This denote input and output images. We can create from numpy arrays.

Halide Basics:



The code of the above program is attached along with this report. So when we build it on Visual Studio, it got executed as “Success”:

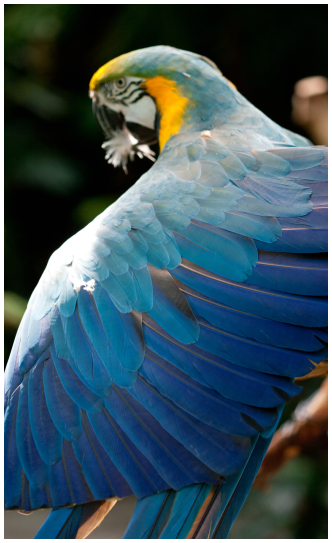


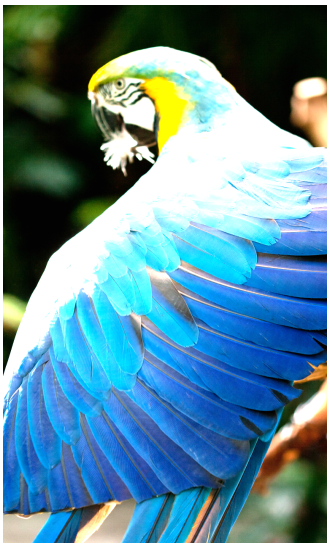
An Example Halide Program to Brighten the Image

```
1   #include <Halide.h>
2   #include <halide_image_io.h>
3
4   using namespace Halide;
5   using namespace Halide::Tools;
6
7   int main() {
8       // Constructing Halide functions statically.
9       ImageParam input(Float(32), 3);
10      Func f("f");
11      Var x("x"), y("y"), c("c");
12      f(x, y, c) = min(2 * input(x, y, c), 1.f); // Double the values and clamp them by 1.
13      // Actually compiling/executing the Halide functions.
14      Buffer<float> input_buffer = load_and_convert_image("images/rgb.png"); // Setup the input by loading an image.
15      input.set(input_buffer);
16      // Process the input by calling f.realize
17      Buffer<float> output_buffer = f.realize(input_buffer.width(), input_buffer.height(), input_buffer.channels());
18      convert_and_save_image(output_buffer, "output.png"); // Save the image to a file.
19  }
```

Now, Lets understand the above program line by line.

- Firstly, we are using necessary header files in C++. We are also using namespace for our ease in writing the program.
- Line 9 means that we are giving input a multidimensional array having Float type and we declared this using the keyword ImageParam. The dimension of the array is included in parenthesis.
- Line 10 and 11 means that we are declaring function f, variables x,y and c. Keywords for defining function in Func and for defining a variable is Var. Letters in the parenthesis are useful for debugging and printing.
- In Line 12, we are defining the function f in terms of x,y and c. It is minimum of 1.f (i.e float 1) and double the value of input(x,y,c).
- Line 14 means that we are loading the image and storing that input in a multidimensional array with name input_buffer. Buffers in halide are used to store the inputs and keyword used is Buffer<data-type>. data type can be int, float etc.
- Line 15 means that input value is set from input_buffer created using keyword input and it is set to buffer. Buffers stores the input values.
- In Line 17, To realize the values, we use the keyword realize and store it in output_buffer. Even the output is stored and buffer and buffer allocates the exact space needed.
- Finally, In Line 18, we are using the values from output_buffer and convert and save to an image.





Input Image **Output Image**

Applications in Halide

Here we would describe how to write Halide algorithms using Halide functions. Halide works by separating the code into algorithms and schedules. The below algorithms are based on default schedules, this means that if one tries to execute these programs they will run slow.

i) Matrix Multiplication:

```
ImageParam A(Float(32), 2, "A"), B(Float(32), 2, "B");
Func C("C");
Var i("i"), j("j");
C(i, j) = 0.f;
RDom k(0, A.dim(1).extent(), "k"); // inner loop
C(i, j) += A(i, k) * B(k, j);
```

The pseudo code for the above algorithm is:

```
for i = 0 to A.rows()
    for j = 0 to B.cols()
        C(i, j) = 0.f
        for k = 0 to A.cols()
            C(i, j) += A(i, k) * B(k, j)
```

Explanation:

- First line takes input two 2D matrix A and B of type float 32.
- Second line declares a Halide function which is named as “C” which is not defined yet.
- Third line defines the function that it has two dimensions and declares coordinates (i,j).
- Fourth line does two works. C(i,j) means that i and j will traverse over C’s elements and all elements are firstly initialized to 0 (float value).
- RDom constructs a variable k that loops over A’s column size.
- Finally the last line finds the value of each element of C where i and k are traversing in A’s row and column and k and j are traversing in B’s row and column respectively.

ii) Image Convolution

Image convolution also uses matrix multiplication for transformation of images. The code is:

```
ImageParam input(Float(32), 2, "in"), kernel(Float(32), 2, "k");
Var x("x"), y("y");
Func bounded_input("input");
bounded_input(x, y) = input(clamp(x, 0, input.dim(0).extent()),
                            clamp(y, 0, input.dim(1).extent()));
Func output("output");
RDom r(0, kernel.dim(0).extent(), 0, kernel.dim(1).extent());
output(x, y) += bounded_input(x - r.x + kernel.dim(0).extent() / 2,
                              y - r.y + kernel.dim(1).extent() / 2) *
                kernel(r.x, r.y);
```

Explanation:

- Firstly two arrays are taken as input of each of 2 dimension. The arrays are “in” and “k”.
- Two variables x and y are declared.
- A Halide function input is declared.
- We define a bounded_input function of which boundaries are defined. Here we need to define out-of-bound access to the closest pixel which is achieved through clamping of coordinates.
- Another function output is defined to store the transformed array.Note that output is automatically initialized to 0.
- Similar to the matrix multiplication , reduction domain variable r is defined.

- Lastly, we calculate each value of output(x,y) using bounded input values and kernel function.

iii) <u>Histograms</u>

We could generate histograms in halide using the following code given below:

```
Param<int> num_bins;
Param<float> hist_min, hist_max;
ImageParam input(Float(32), 1, "in");
Var x("x");
Func hist("hist");
RDom r(0, input.dim(0).extent());
Expr hist_index = cast<int>(num_bins * ((input(r) - hist_min) / (hist_max
- hist_min)));
hist(hist_index) += 1;
```

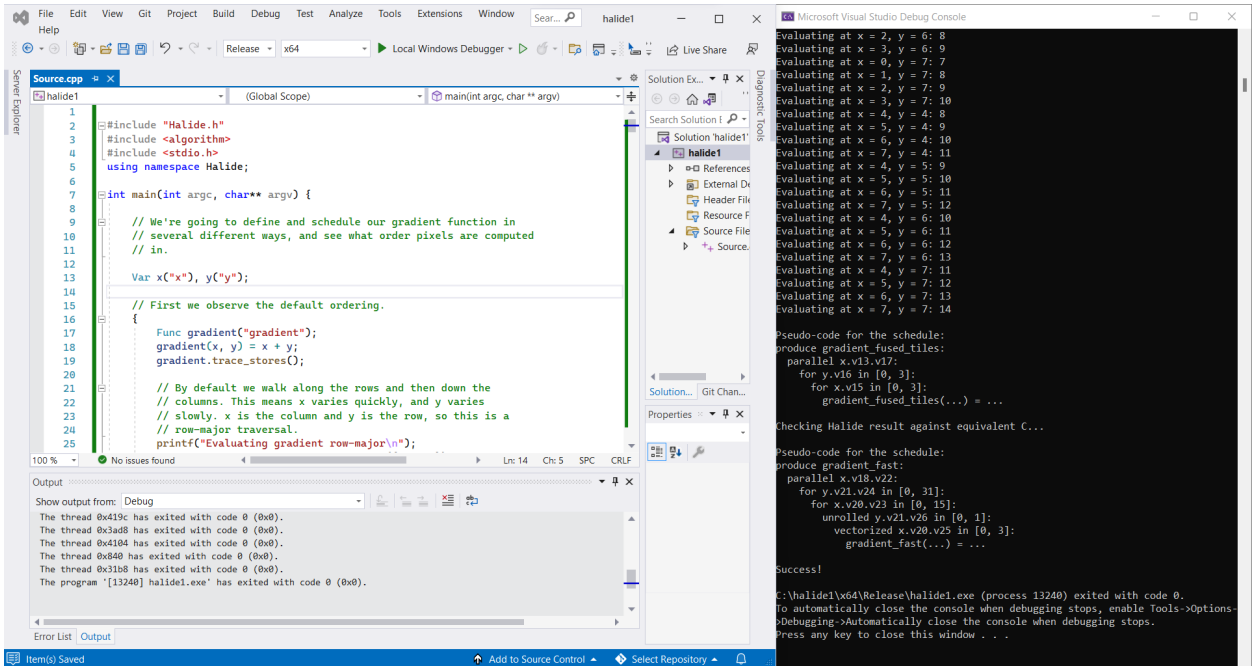
We define the parameters which are number of bins, max and min values. We declare a input halide function ImageParam which takes input values. “hist” function is used to calculate the scaled value of the bin which is casted onto the particular index. The index is incremented by 1 each time.

<u>Schedules</u>:

Halide Schedules define the organization of computation in an image processing pipeline by specifying two things namely for each stage, what is the order to compute the values and when do we need to compute the inputs.

Let’s first understand two terms namely producer and consumer. Assume that there are two functions blur_x and blur_y. If blur_y function uses blur_x function computed in the previous step, then blur_x is called producer and blur_y is called consumer. In simple terms, producer is the one which is computed at earlier stage and consumer is the one computed at later stage.

The below code and execution is of Scheduling1.cpp . It shows how scheduling happens and also prints the pseudo code of Scheduling used on the console. The program compiled without any error. The program covers all concepts of scheduling, tiling, fusion etc,.



Below are two types of schedules

(i) Root Schedule:

In a Root schedule, the producer is computed on the entire image before computing the consumer. The syntax for root schedule is Func.compute_root(). Advantage is that there is no redundant work involved in computation.

(ii) Inline Schedule:

In an Inline schedule, for every value of the consumer, we find the producer value right when needed. This can be thought of as opposite to root schedule. The syntax for inline schedule is Func.compute_inline(). Advantage of inline schedule is that it makes use of locality.

Multi stage pipelines:

In multi stage Halide pipeline, there are multiple recursive functions. Each function calls another function and so multiple dependencies exist.

The below program expresses a multi stage pipeline that blurs an image first horizontally and then vertically.

Tiling & Fusion:

The consumer can be split into tiles and find the value of producer for each tile before finding out the consumer tile. Function used is .tile() on consumer and .compute_at() on producer. This concept makes use of parallelism. For fusion, we insert producer loop at required level of consumer loop.

Clamping:

The “clamp” function in halide is a built in function used to introduce clamping or padding.It clamps the array on the edges to introduce padding.
The syntax to introduce clamping at the time of taking input is :

```
Func clamped("clamped");
clamped(x, y, c) = input(clamp(x, 0, im.width()-1),
clamp(y, 0, im.height()-1),
c);
```

Reductions:

They are used to aggregate multiple values such as for case where we want to use a for loop over input pixels. It has various applications:

- average/max of image
- Histograms
- Convolution of image
- Max over windows

About Reductions:

- Loops are implicit in halide
- RDom is Reduction Domain which is used to initialize a variable that could traverse in the array (or used as a loop).

  ```
  RDom(baseX, extentX, baseY, extentY,...)
  ```

- Initializing of functions.

  ```
  myFunc(Var1, Var2)=initialValue;
  ```

- Updating functions:

  ```
  myFunc(Expr,Expr) = f ( myFunc(Expr,Expr), rdom );
  ```

Conclusion

In this report we have explored Halide library and its applications. We have also ran the codes which are attached along with this report. The codes are easy to understand as they are well commented and few of those explanations are also given in the report itself. The report covers all the necessary topics that should be covered while learning halide and is well fit for anyone who is a beginner to learn this language.

References:

[1] https://halide-lang.org/
[2] https://github.com/halide/Halide
[3] https://dl.acm.org/doi/fullHtml/10.1145/3406117
[4] https://people.csail.mit.edu/tzumao/tmp/learning_halide/what_is_halide.html
[5] https://www.youtube.com/watch?v=1ir_nEfKQ7A