NC STATE UNIVERSITY

Applying NEON Advanced SIMD to find_chroma_matches

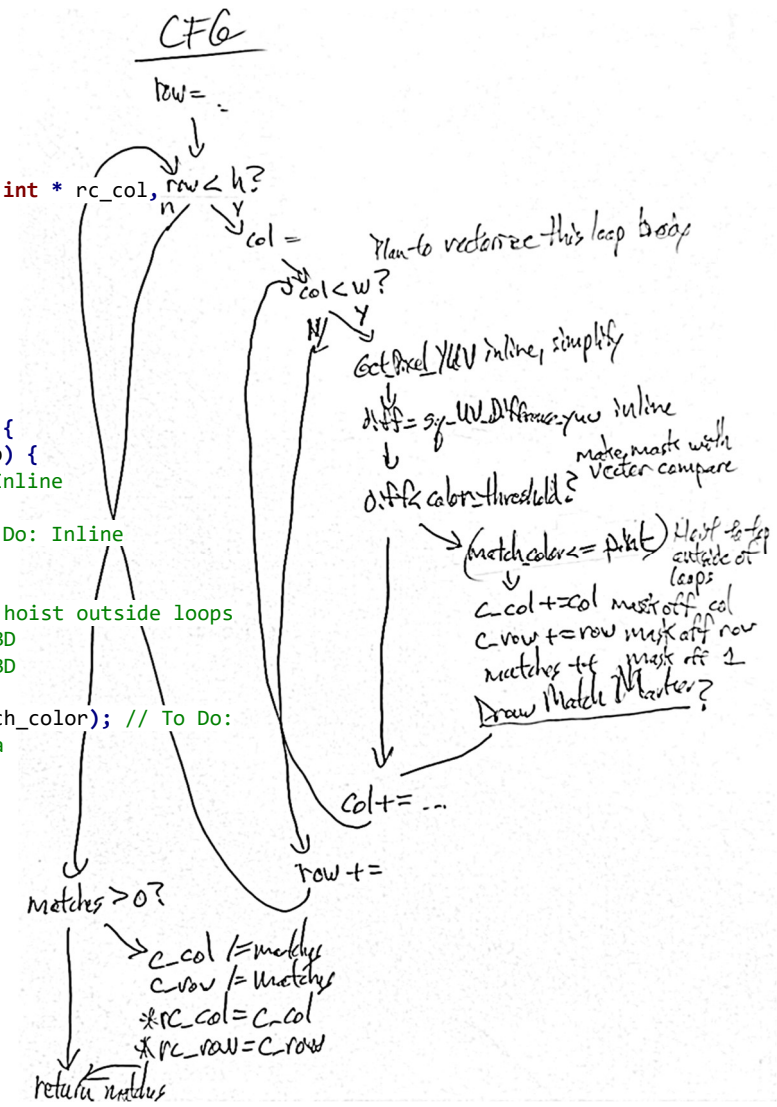
Applying Neon Advanced SIMD to find_chroma_matches

- Analyze control flow
  - Create CFG
  - Mark up CFG with plan for modifications and simplifications
    - Inline subroutine calls, perform if conversion, apply special instructions
- Analyze data flow
  - Evaluate data layout in memory and compatibility with vector registers
  - Create DFG
  - Mark up DFG with plan for instructions/features to apply
  - Evaluate data sizes and minimum precision needed
- Write code
  - prolog
  - body
  - epilog
    - reduce results

1. Analyze Control Flow

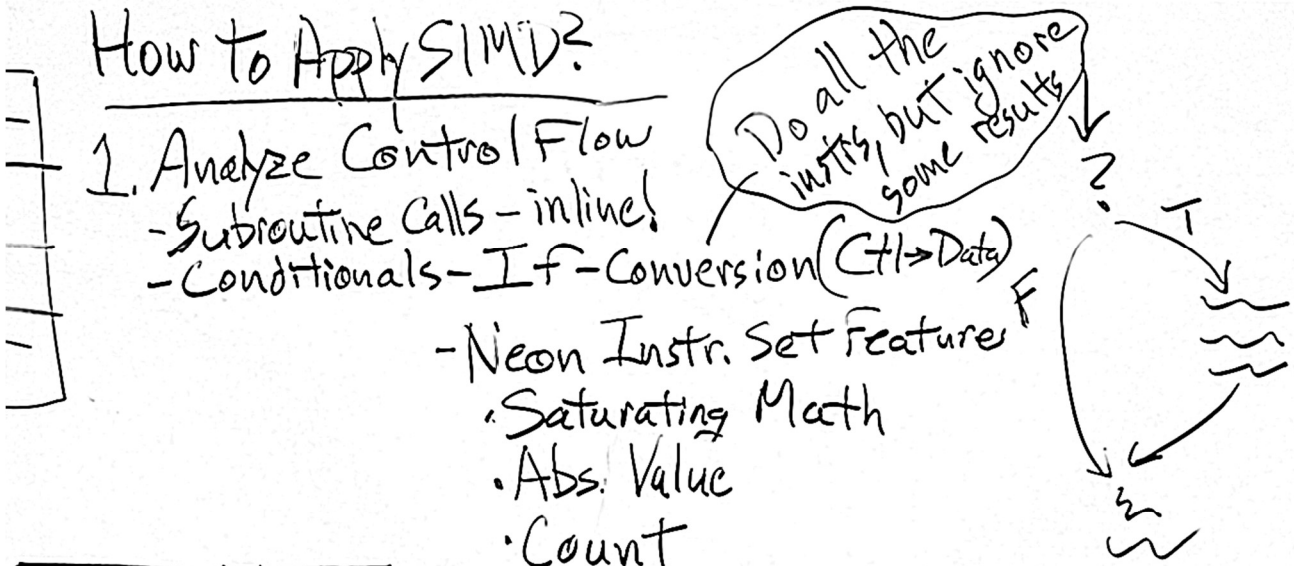
- Create plan to improve control flow
  - Inline subroutine calls
  - Apply any special instructions or features which can eliminate conditional control flow
    - Conditional instruction execution
    - Saturating math
    - Absolute value, min, max, count bits, count leading bits, etc.
    - Bitwise select/insert
  - Perform if-conversion

```
int find_chroma_matches(YUV_IMAGE_T * i, YUV_T * tc, int * rc_col,
                        int * rc_row, int sep){
  int col, row;
  int matches=0, diff;
  YUV_T color, prev_color={0,0,0};
  int c_col=0, c_row=0;
  YUV_T * match_color = &pink;
  YUV_T v_color;

  for (row = sep/2; row <= i->h - sep/2; row += sep) {
    for (col = sep/2; col <= i->w - sep/2; col += sep) {
      Get_Pixel_yuv(i, col, row, &color); // To Do: Inline
      // Identify pixels with right color
      diff = Sq_UV_Difference_yuv(&color, tc); // To Do: Inline

      if (diff < color_threshold) { // To Do:
            match_color = &pink; // To Do: Manually hoist outside loops
            c_col += col; // To Do: If-conversion TBD
            c_row += row; // To Do: If-conversion TBD
            matches++; // To Do: If-conversion
            Draw_Match_Marker(i, col, row, sep, match_color); // To Do:
            // Figure out how to merge with ex. data
      }
    } // for col
  } // for row

  if (matches > 0) {
    c_col /= matches;
    c_row /= matches;
    *rc_col = c_col;
    *rc_row = c_row;
  }

  return matches;
}
```

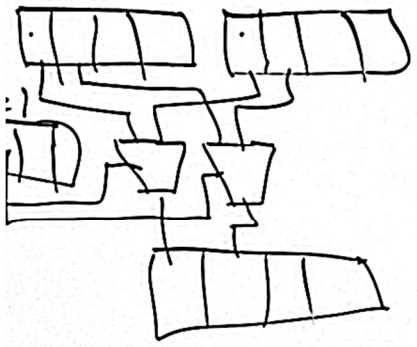


How to Apply SIMD?

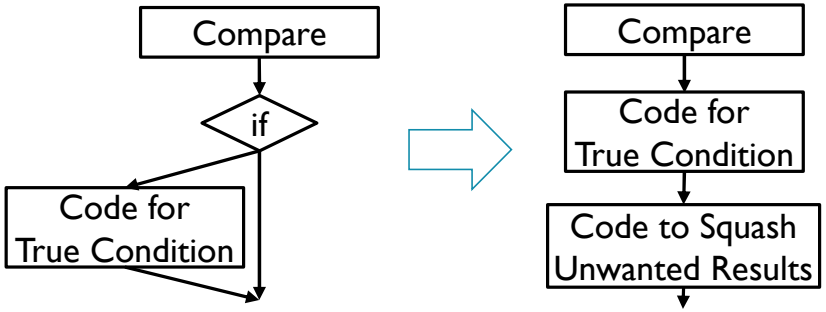
1. Analyze Control Flow
   - Subroutine Calls – inline!
   - Conditionals – If-Conversion (Ctl→Data)

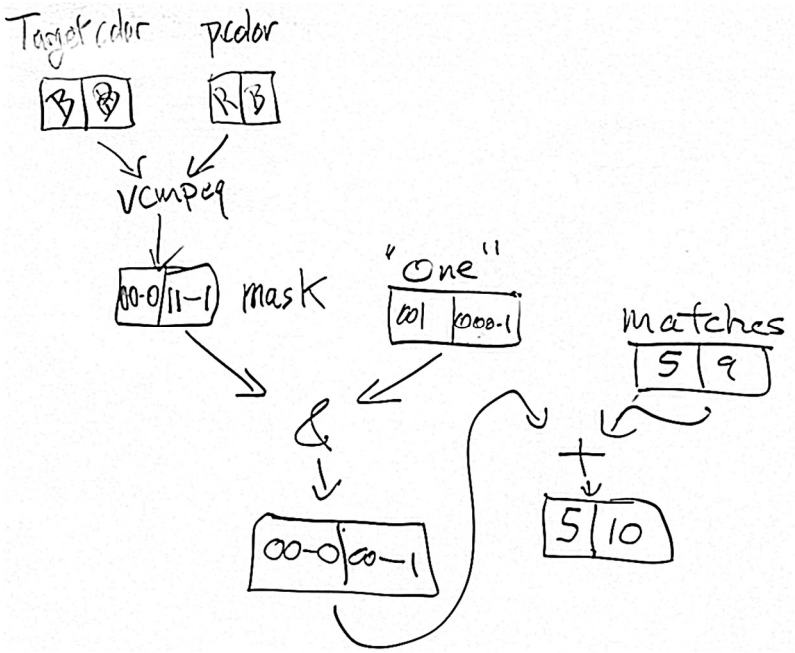


- Neon Instr. Set Features
  - Saturating Math
  - Abs. Value
  - Count
  - Table Lookup
  - Transpose, Zip, Unzip
  - Bitwise Select/insert
  - Vector Compare
  - Fused ops. Mac, sad



If Conversion – Eliminate Conditional Control Flow

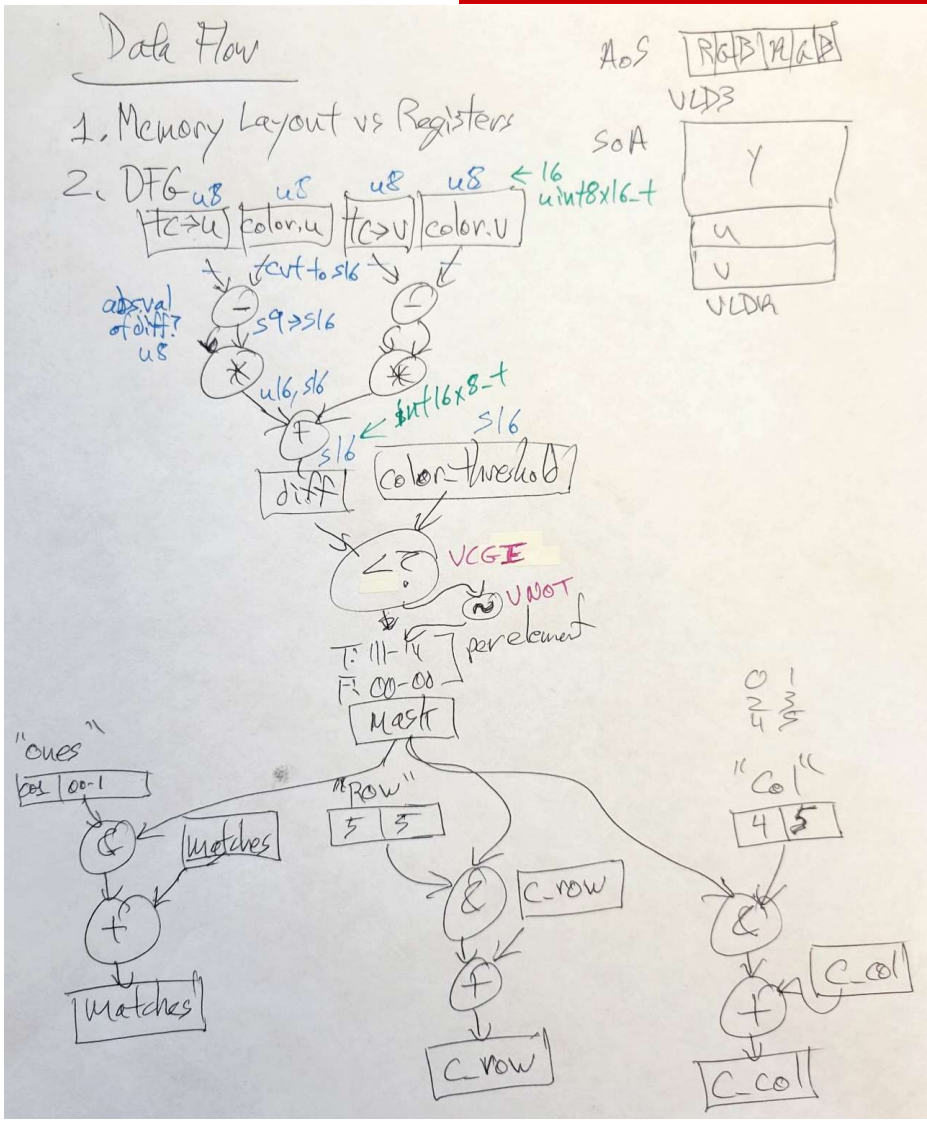




- Convert code with conditional control flow ("If") to unconditional control flow
  - Execute code from both true and false conditions
    - No code for false condition in this example.
  - Merge results, ignoring results of code from wrong condition.
- Two methods
  - ARM ISA has conditional instruction execution (predication).
    - See "ARMv7A Features for Performance" slides
    - Compiler may be able to implement
  - Masking. Squash/replace unwanted result data with math/logic/move operations
    - Sum: replace with 0 (e.g. use vector AND with #0)
    - Product: replace with 1 (e.g. use vector bit insert/select)
    - Min, max: replace with max_int or min_int for data type (e.g. use vector bit insert/select)

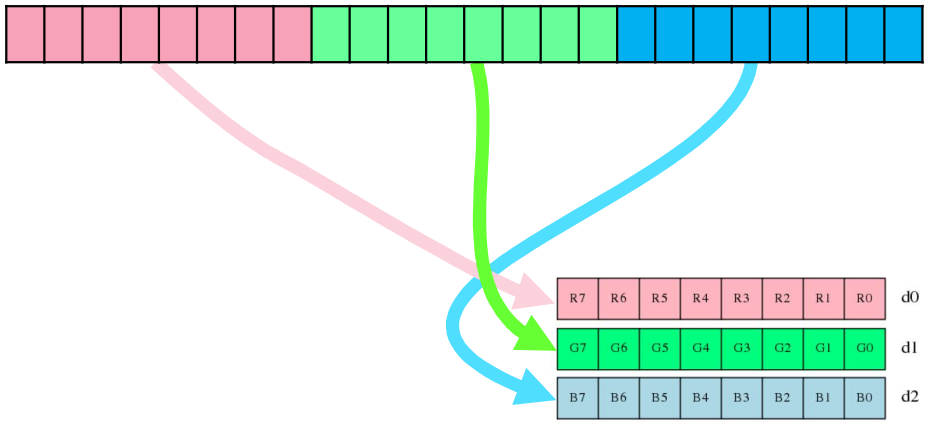
2. Analyze Data Flow

- Compare data layout in memory vs. SIMD registers
- Create data flow graph with variables and operations
- Evaluate element data precision requirements
- Plan data reduction

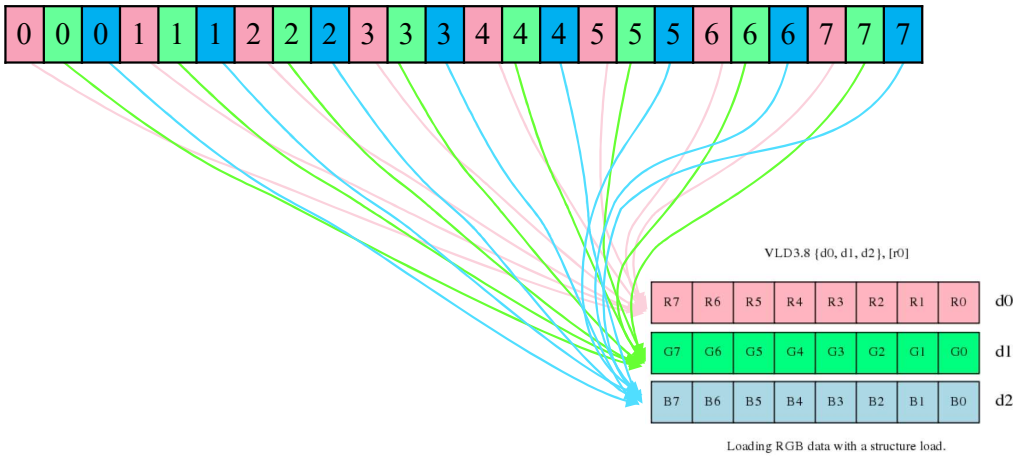


Evaluate Data Layout

- Does data layout in memory match ideal register layout?



- Yes: Structure of arrays (SoA). Use [vldr|vstr]



- No: Array of structures (AoS). Use structure load/store [vld|vst][1|2|3|4]

Create Data Flow Graph

- Show data and operations
  - Nodes: operations (instructions)
  - Edges: data variables and intermediate results
- Define if-conversion to remove code from conditionals
  - Use vector compare to create mask register
  - Use that mask to select relevant results
- Identify where to apply other useful Neon instructions
  - Fused operations: MAC (multiply accumulate), SAD (sum of absolute differences), etc.
  - Zip/unzip, transpose
  - Table lookup
  - etc…

Evaluate Precision Requirements for Data Elements

- Annotate DFG
  - Mark data types for input data values
    - signed/unsigned, size (bits), integer/float
  - Mark operations which increase minimum precision
    - <b> + <b> = <b+1>
    - <b> * <b> = <b+b>
  - Propagate data types through DFG to mark intermediate data values
  - Define data types for intermediate variables and outputs
- Consider reducing precision to increase elements per vector
  - Clip precision with saturating math
  - Reduce precision by scaling. Use halving instructions

Plan Data Reduction

- Combine each element (lane) in vector to determine overall data value

- Use pairwise operations to merge lanes
  - Pairwise add, multiply, minimum, maximum, etc.