SIMD Tutorial

Compiler Vector, SIMD Intrinsics, Halide and OpenCL

Champ Yen (champ.yen@gmail.com)

About Me - Champ Yen

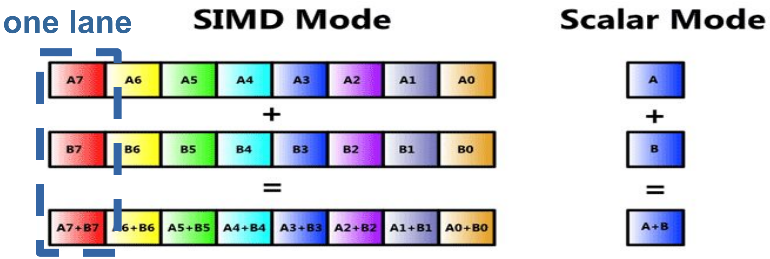
- Career
  - Sunplus mMedia - NAND Driver/FTL/uCOS
  - Alpha Image Tech. - Linux Kernel/GPU driver
  - Novatek - Video Codec
  - Mediatek - Heterogeneous Computing, Camera Features optimization
  - OnePlus - Camera Features optimization
  - **Qualcomm - RICA Application Development**
- Personal Channel
  - Facebook - https://www.facebook.com/champ.yen
  - Medium - https://medium.com/@champ.yen
  - Blogger - https://champyen.blogspot.tw

Agenda

- What is SIMD
- SIMD Programming Tools
  - Compiler Vector Extensions
  - Architecture Specific Intrinsics
  - OpenCL
  - Halide
- What are the difficulties in SIMD?
- Q & A

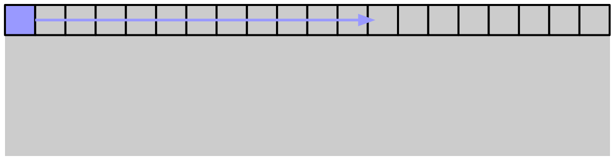
What Is SIMD?

Single Instruction Multiple Data




```
for(y = 0; y < height; y++){
	for(x = 0; x < width; x+=8){
		//process 8 point simutaneously
		uint16x8_t va, vb, vout;
		va = vld1q_u16(a+x);
		vb = vld1q_u16(b+x);
		vout = vaddq_u16(va, vb);
		vst1q_u16(out+x, vout);
	}
	a+=width; b+=width; out+=width;
}
```

```
for(y = 0; y < height; y++){
	for(x = 0; x < width; x++){
		//process 1 point
		out[x] = a[x]+b[x];
	}
	a+=width; b+=width; out+=width;
}
```



SIMD Optimization Tools

- Automatic Vectorization
- Compiler Vector Extension
- Compiler Intrinsics
- OpenCL
- Halide

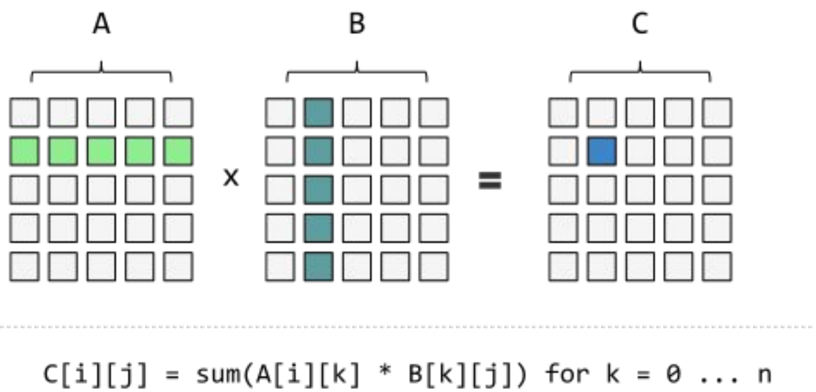
Automatic Vectorization

Automatic Vectorization

- clang -mllvm -force-vector-width=N ...
- Heavily depends on **[] array operations**
- performance may vary between versions
- no guarantee of performance gain
- difficult for further optimization

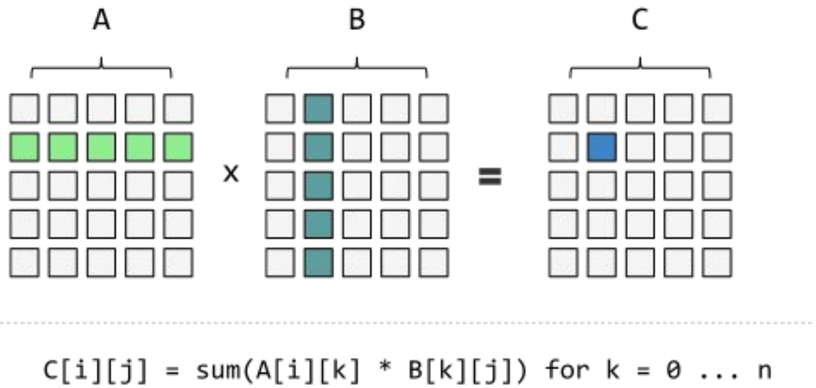
Today's Lab - Matrix Multiplication

Matrix Multiplication - Naive



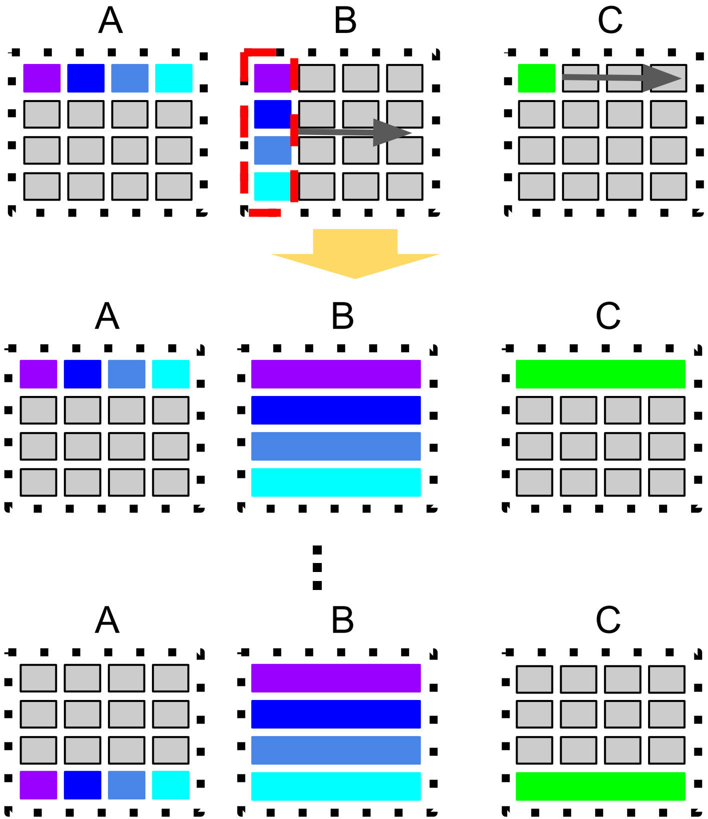

```
for(int i = 0; i < M; i++){
    for(int j = 0; j < N; j++){

        float c = 0;
        for(int k = 0; k < K; k++){
            c = ma[i*K + k]*mb[k*N + j];
        }
        mc[i*N + j] = c;

    }
}
```

Tiled Matrix Multiplication




```
#define TSIZE  4
for(int i = 0; i < M; i+=TSIZE){
    for(int j = 0; j < N; j+=TSIZE){
        for(int k = 0; k < K; k+=TSIZE){
            //tc[i][j] += ta[i][k]*tb[k][j];
            MM_4x4(
                &(ma[i*sa + k]), sa,
                &(mb[k*sb + j]), sb,
                &(mc[i*sc + j]), sc
            );
        }
    }
}
```

Vectorized 4x4 Matrix Multiplication




```
//i range from 0 to 3
vc[i] += (
            va[i].s0 * vb[0] +
            va[i].s1 * vb[1] +
            va[i].s2 * vb[2] +
            va[i].s3 * vb[3] +
        );
```

Let's Starting From

https://github.com/champyen/simd_2018

Compiler Vector Extension

Compiler Vector Extension - gcc & clang

- http://releases.llvm.org/6.0.0/tools/clang/docs/LanguageExtensions.html#vectors-and-extended-vectors
- typedef TYPE VECNAME __attribute__((ext_vector_type(VEC_LENGTH)));
  - eg. : typedef float float4 __attribute__((ext_vector_type(4)));
- (very similar) OpenCL vector types
  - swizzle (.sN or .xyzw)
  - wide range operations
- __builtin_convertvector, __builtin_shufflevector
- **vectors can be used as short array**
- **better performance (than autovector)**
- **near c readability**
- **easy to use**
- **good portability**

Operator	OpenCL	AltiVec	GCC	NEON
[]	yes	yes	yes	–
unary operators +, –	yes	yes	yes	–
++, – –	yes	yes	yes	–
+,–,*,/,%	yes	yes	yes	–
bitwise operators &,|,^,~	yes	yes	yes	–
>>,<<	yes	yes	yes	–
!, &&, ||	yes	–	–	–
==, !=, >, <, >=, <=	yes	yes	–	–
=	yes	yes	yes	yes
:?	yes	–	–	–
sizeof	yes	yes	yes	yes
C-style cast	yes	yes	yes	no
reinterpret_cast	yes	no	yes	no
static_cast	yes	no	yes	no
const_cast	no	no	no	no

Example: C = A + B

```
typedef unsigned short ushort8 __attribute__((ext_vector_type(8)));

for(y = 0; y < height; y++){
    for(x = 0; x < width; x+=8){
        //process 8 point simutaneously
        ushort8 va, vb, vout;
        va = *(ushort8*)(a+x);
        vb = *(ushort8*)(b+x);
        vout = va + vb;
        *(ushort8*)(out+x) = vout;
    }
    a+=width; b+=width; out+=width;
}
```

Architecture Specific SIMD Intrinsics

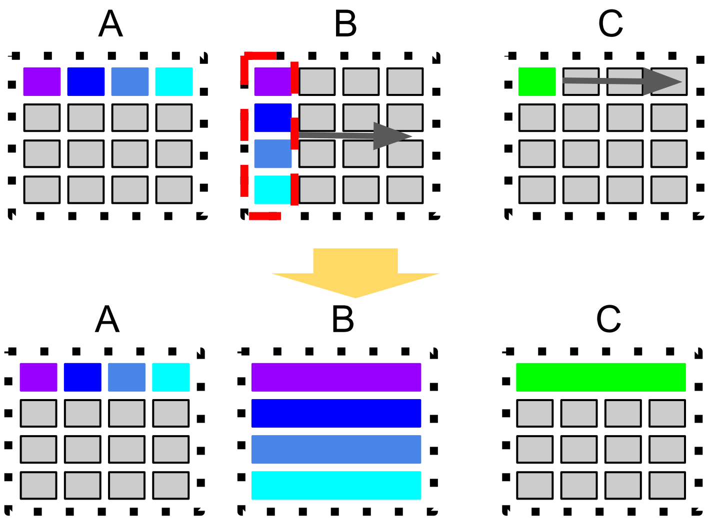
Architecture SIMD intrinsics

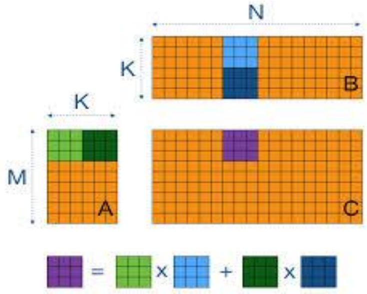
- Why do we need Arch-Specific SIMD intrinsics?
    - not normal operations (16*16 => 16b of High Part)
    - more precise control (specific instruction usage, eg: Mul-Add)
    - advanced intrinsics (LUT, shuffle)
    - difficult for compiler to figure out ILP
- X86
    - MMX/SSE/AVX,AVX2/AVX-512
    - https://software.intel.com/sites/landingpage/IntrinsicsGuide/
- ARM
    - DSP ext/NEON
    - http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0073a/index.html
- DSP
    - Hexagon V6x, HVX - https://developer.qualcomm.com/software/hexagon-dsp-sdk/tools
    - CEVA XM*/Cadence IVP/Synopsys Arc EV6x

SIMD instruction types

- Load/Store
- Per-Lane:
  - Arithmetic,
  - Bitwise, Logical
- Cross/Inter-Lane:
  - Permute, Select,Shuffle(LUT)
  - Alignment
  - Pack, Unpack
- Reduction (xxx of a vector):
  - Minimum
  - Maximum
  - Average
- Special (eg: NN specific ISA, inter-lane + per-lane attributes)

ARM NEON Example





4x4 Matrix Multiplication ARM NEON Example
http://www.fixstars.com/en/news/?p=125

```
//...
    //Load matrixB into four vectors
    uint16x4_t vectorB1, vectorB2, vectorB3, vectorB4;

    vectorB1 = vld1_u16 (B[0]);
    vectorB2 = vld1_u16 (B[1]);
    vectorB3 = vld1_u16 (B[2]);
    vectorB4 = vld1_u16 (B[3]);

    //Temporary vectors to use with calculating the dotproduct
    uint16x4_t vectorT1, vectorT2, vectorT3, vectorT4;

    // For each row in A...
    for (i=0; i<4; i++){
        //Multiply the rows in B by each value in A's row
        vectorT1 = vmul_n_u16(vectorB1, A[i][0]);
        vectorT2 = vmul_n_u16(vectorB2, A[i][1]);
        vectorT3 = vmul_n_u16(vectorB3, A[i][2]);
        vectorT4 = vmul_n_u16(vectorB4, A[i][3]);

        //Add them together
        vectorT1 = vadd_u16(vectorT1, vectorT2);
        vectorT1 = vadd_u16(vectorT1, vectorT3);
        vectorT1 = vadd_u16(vectorT1, vectorT4);

        //Output the dotproduct
        vst1_u16 (C[i], vectorT1);
    }
//...
```

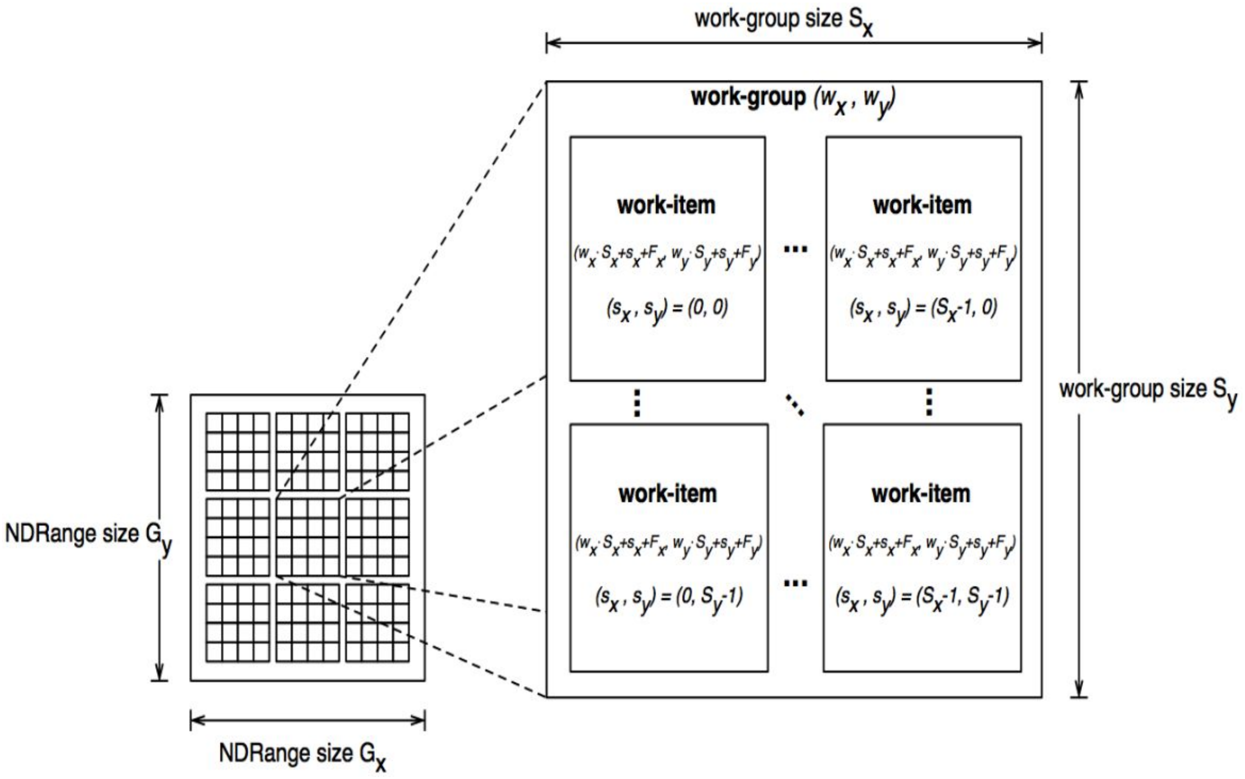
OpenCL/GPU

OpenCL/GPU

- make use of GPU computation resource
    - Throughput Oriented
- Task Parallelism / Data Parallelism
- Hardware Acceleration (OpenCL built-in functions)
    - math functions
    - image manipulation
    - OpenGL interoperability

OpenCL/GPU (Cont.)

- The IDs of workitems in dimensions are similar to loop indices.
- There is **NO** necessary link between **data geometry** and **workitem indices.**
- **Task Partitioning**
- **NOT Data Partitioning**



Short Intro. of CLTK

- CLTK provides simple & glue-code-free OpenCL Programming.
- OpenCL Programming Obstacles:
  - Initialization
  - Buffer/Image Allocation
  - Queue Manipulation
  - Kernel Execution
- **Let programmer focus on kernel implementation.**
- https://github.com/champyen/cltk/blob/master/example/cltk_test.c

Example: Gradient Filling Kernel

```
__kernel void gradient(
      __global int* buf
){
      int gidx = get_global_id(0);
      int gidy = get_global_id(1);
      buf[gidy*get_global_size(0) + gidx] = gidx + gidy;
}

// the width/height is specified by Host code
```

```
for(int y = 0; y < height; y++){
    for(int x = 0; x < width; x++){
      int gidx = x;
      int gidy = y;
      buf[gidy*width + gidx] = gidx + gidy;
    }
}
```

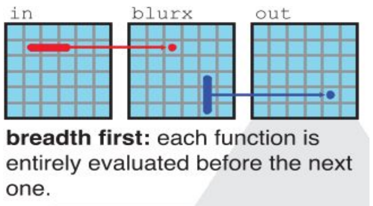
Host part of the CL example is coding with CLTK
https://github.com/champyen/cltk

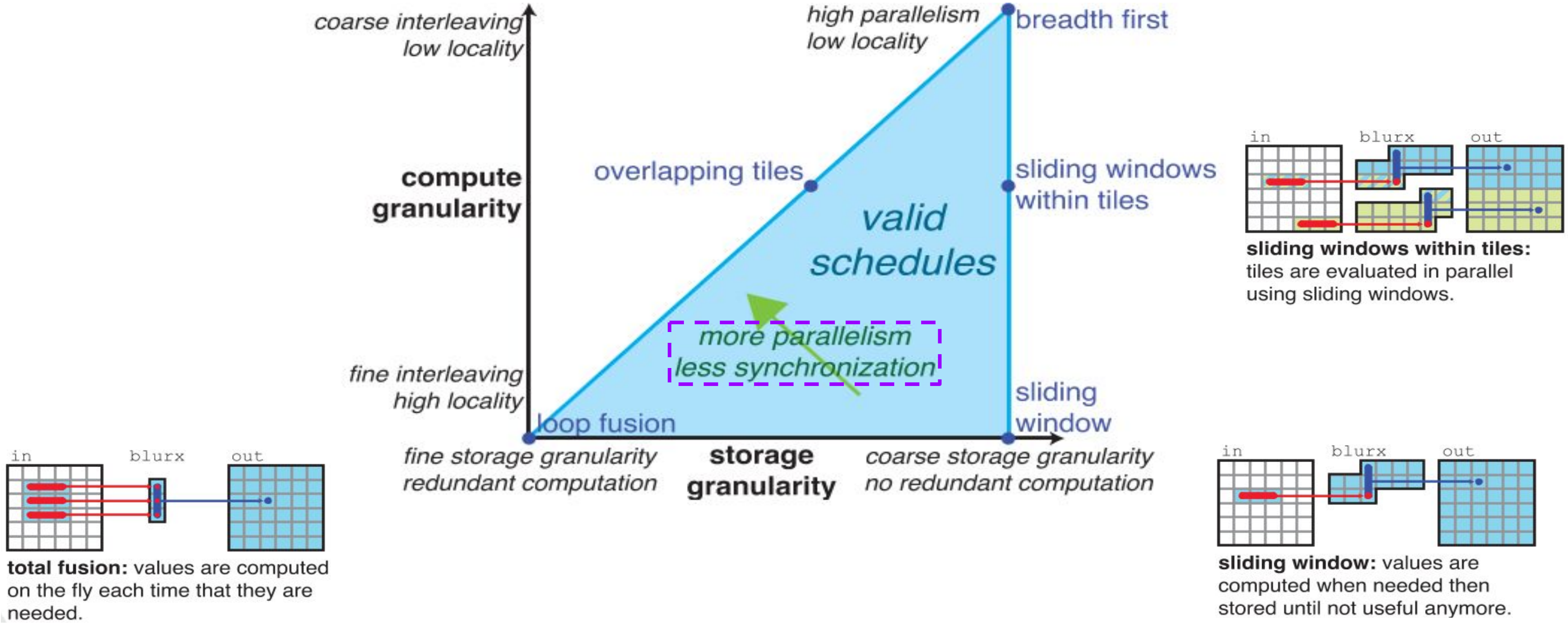
Halide

Halide

- Decouple Algorithm, Scheduling
- In-C++ DSL (implemented w/ operator override)
- Fundamentals
  - Func - Processing Stage, only Func is schedulable
  - Var - Input argument of Func
  - Exp - The computation in a Func
  - RDom - Reduction Domain
- The structure of loops have great influence on performance
- Auto-Scheduler
- Let's Halide From Here:
  - https://docs.google.com/presentation/d/1S-MnTQGpLhhtax5L7QRXtMDS-GIQdsu_DYSCetNbinY

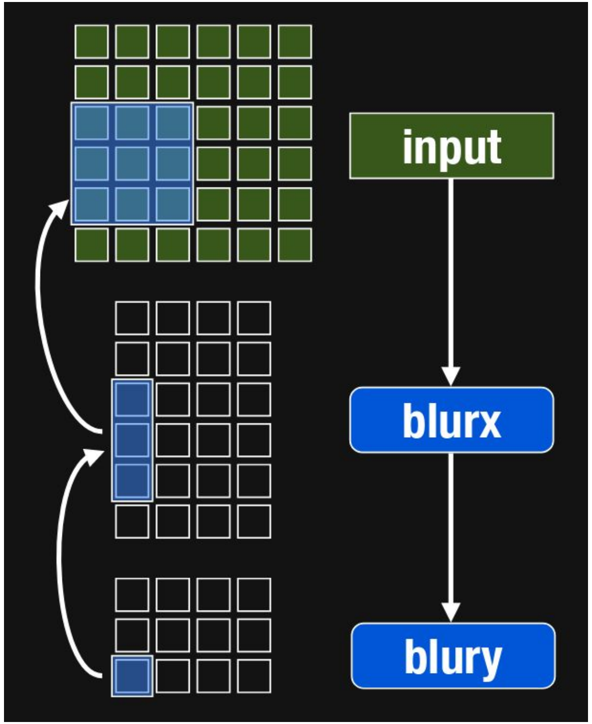
Halide (Cont.)


in
blurx
out
breadth first: each function is entirely evaluated before the next one.


coarse interleaving
low locality
compute
granularity
fine interleaving
high locality
high parallelism
low locality
breadth first
overlapping tiles
sliding windows
within tiles
valid
schedules
more parallelism
less synchronization
sliding
window
loop fusion
fine storage granularity
redundant computation
storage
granularity
coarse storage granularity
no redundant computation
in
blurx
out
total fusion: values are computed on the fly each time that they are needed.
in
blurx
out
sliding windows within tiles: tiles are evaluated in parallel using sliding windows.
in
blurx
out
sliding window: values are computed when needed then stored until not useful anymore.

Example 3x3 Box Blur

```
Var x, y;

Func blurx, blury;

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;

// =========== schedule ==========
Var xo, yo, xi, yi;

blury.tile(x, y, xo, yo, xi, yi, 16, 16)
.vectorized(xi, 4)
.parallel(yo);
```

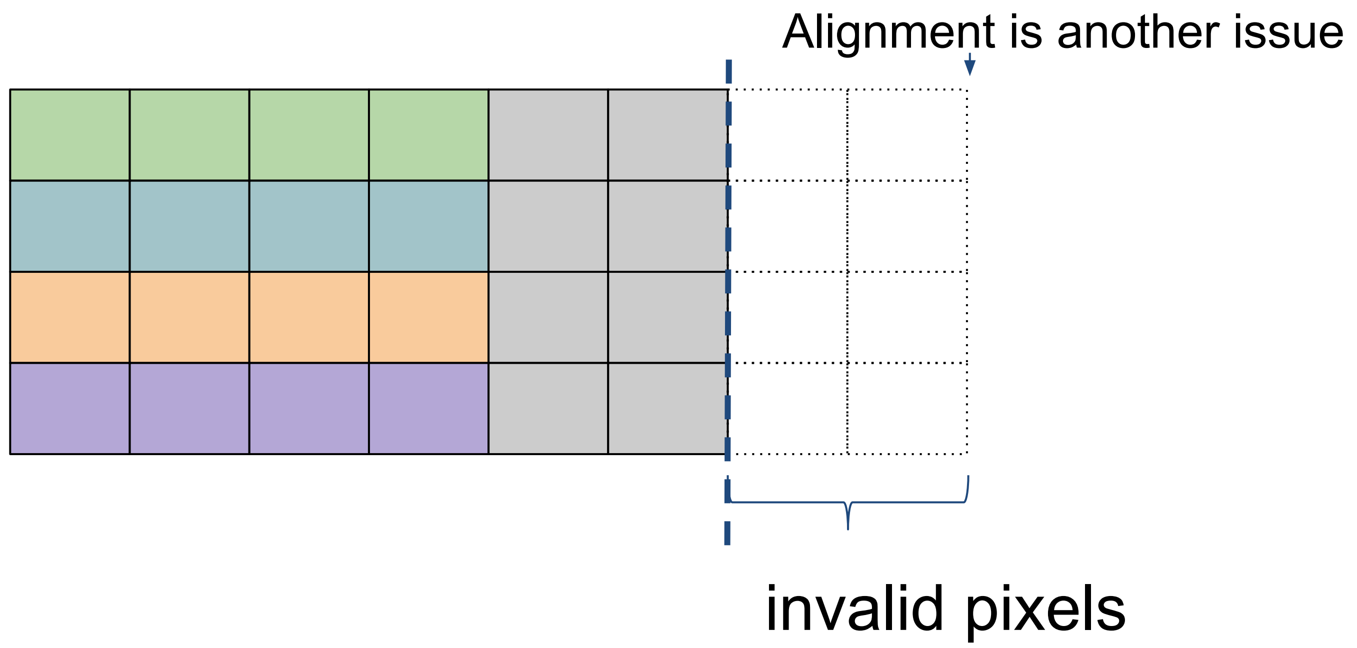


What Are The Difficulties with SIMD?

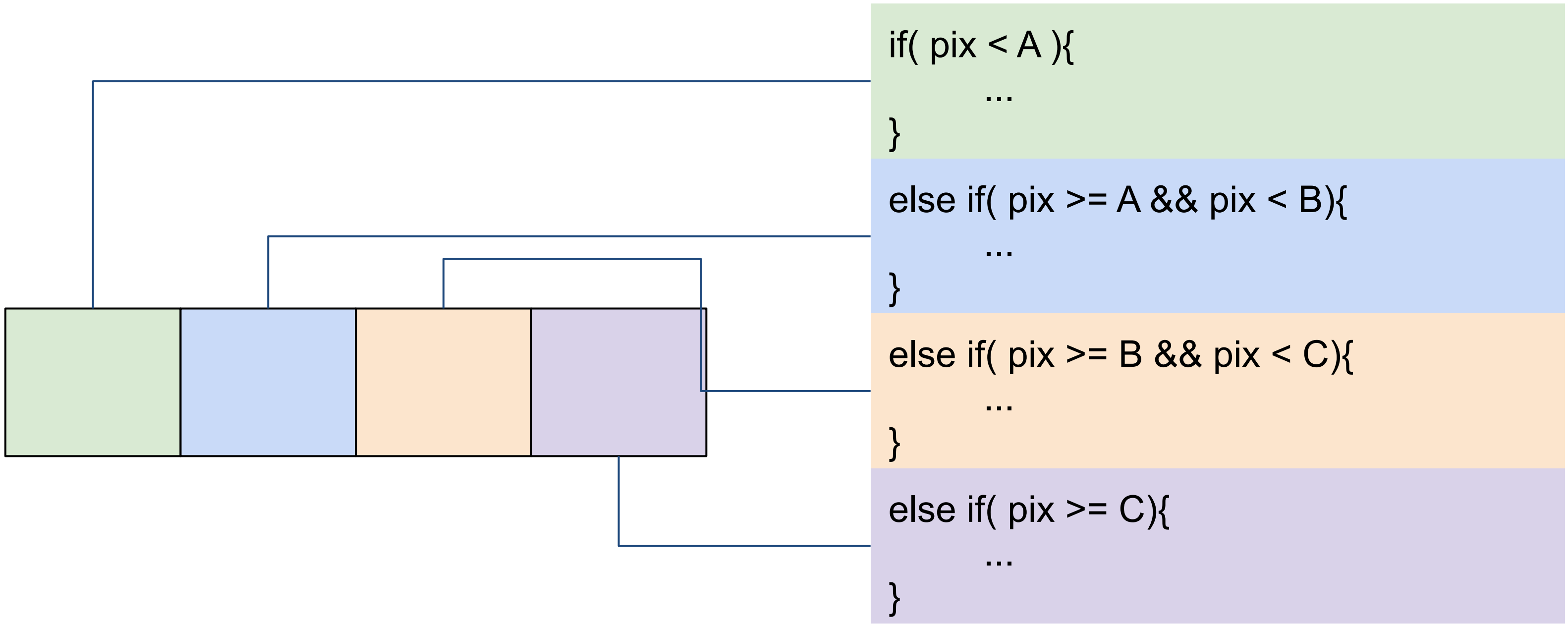
Difficulties In SIMD

- Finding Parallelism
- Portabilities
- Boundary Handling
- Divergence
- Register Spilling
- Non-Regular (Memory, Computation) Pattern, Dependencies
  - LUT, AoS(Array of Structure), content dependent flow
  - Multi-stages/Reduction ISA/Enhanced DMA
- Unsupported Operations
  - Division, High-level function (eg: math functions)
- Floating Point
  - Unsupported/cross-deivce Compatiblilty

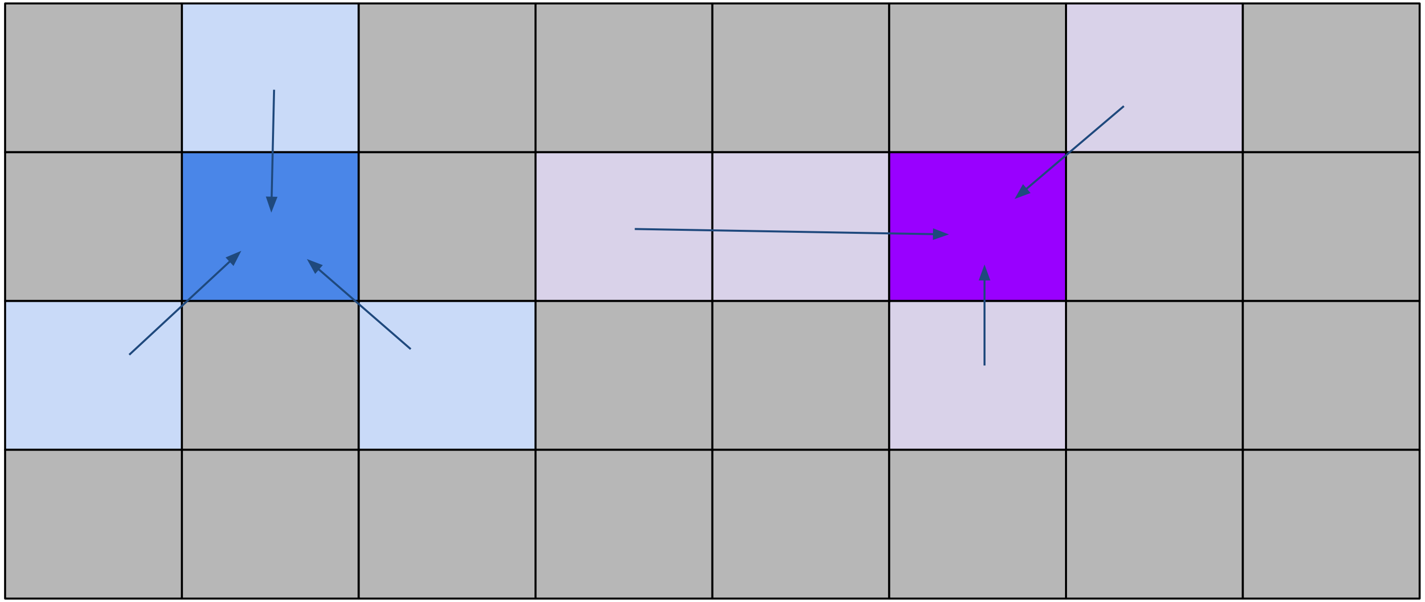
Not-Aligned Boundaries



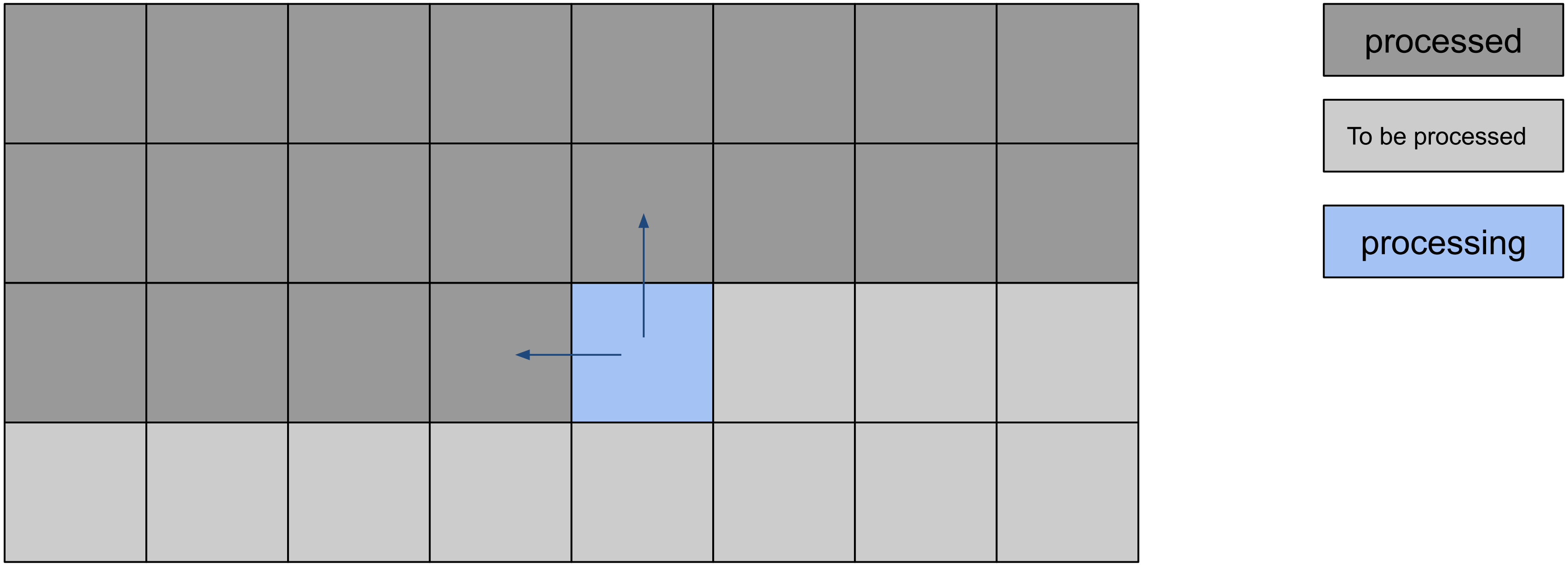
Divergence



Irregular Pattern



Dependencies



Q & A