SIMD Introduction
Application, Hardware & Software

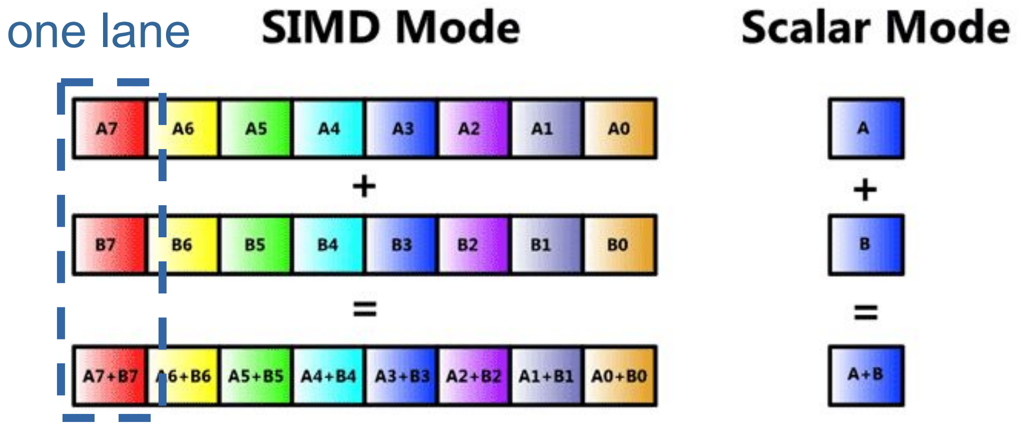
Champ Yen (嚴梓鴻)
champ.yen@gmail.com
http://champyen.blogspot.tw

Agenda

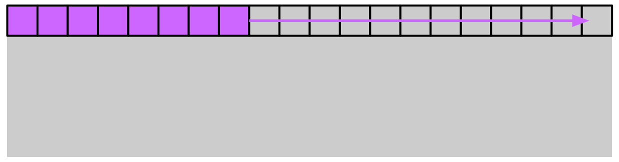
- What & Why SIMD
- SIMD in different Processors
- SIMD for Software Optimization
- What are important in SIMD?
- Q & A

Link of This Slides
https://goo.gl/Rc8xPE

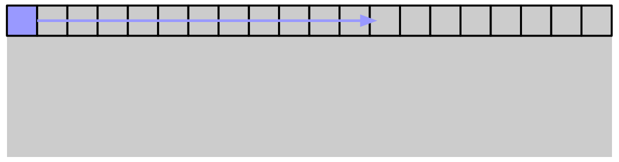
What is SIMD (**S**ingle **I**nstruction **M**ultiple **D**ata)



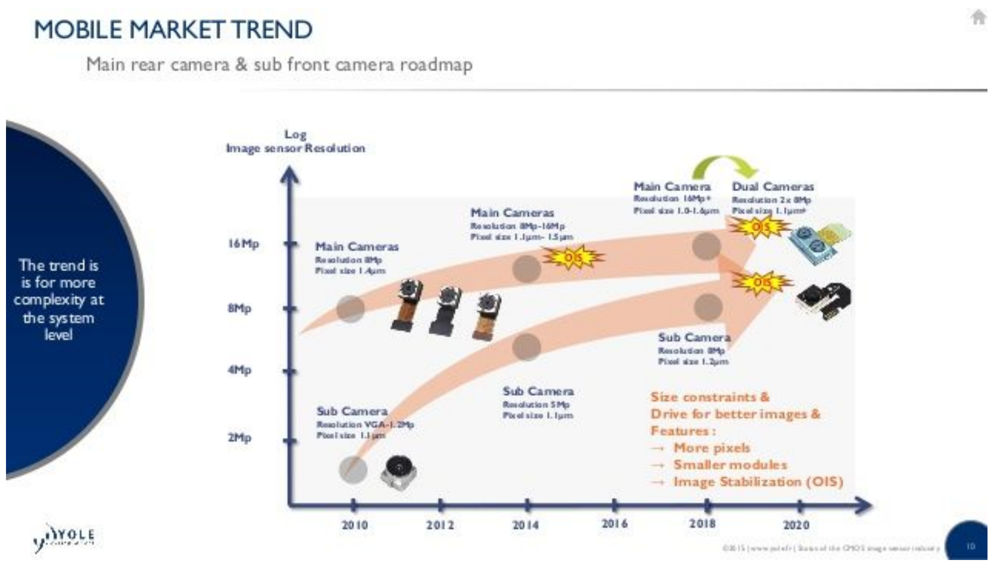
```
for(y = 0; y < height; y++){
     for(x = 0; x < width; x+=8){
          //process 8 point simutaneously
          uin16x8_t va, vb, vout;
          va = vld1q_u16(a+x);
          vb = vld1q_u16(b+x);
          vout = vaddq_u16(va, vb);
          vst1q_u16(out, vout);
     }
     a+=width; b+=width; out+=width;
}
```

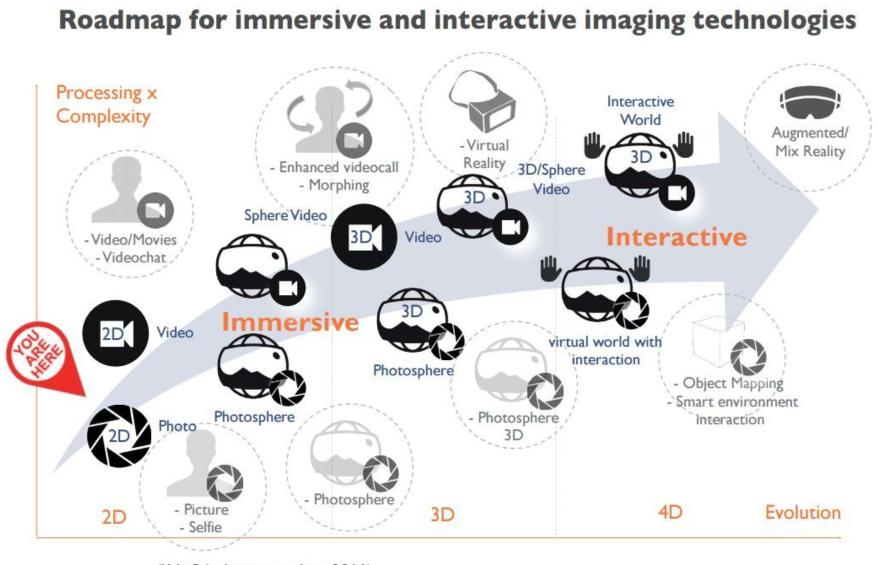


```
for(y = 0; y < height; y++){
     for(x = 0; x < width; x++){
          //process 1 point
          out[x] = a[x]+b[x];
     }
     a+=width; b+=width; out+=width;
}
```

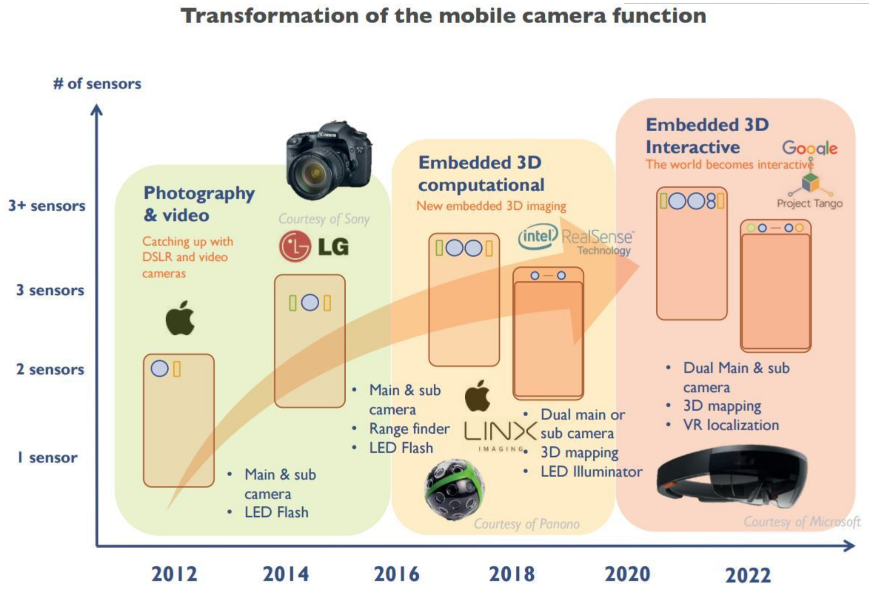


Why do we need to use SIMD?





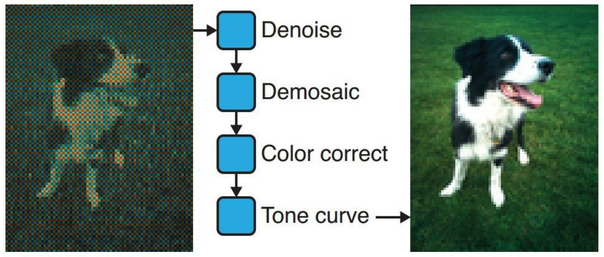
(Yole Développement, June 2016)



(Yole Développement, June 2016)

Why & How do we use SIMD?

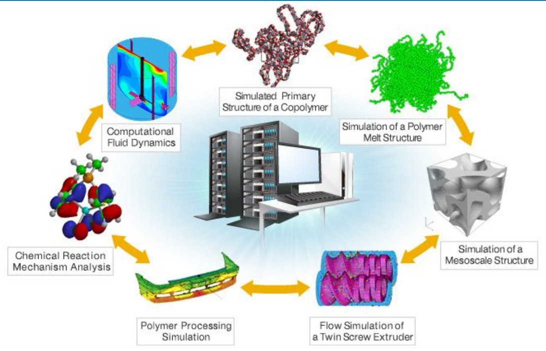
Image Processing



OpenVX

OpenCV

Scientific Computing


Simulated Primary Structure of a Copolymer
Simulation of a Polymer Melt Structure
Computational Fluid Dynamics
Simulation of a Mesoscale Structure
Chemical Reaction Mechanism Analysis
Polymer Processing Simulation
Flow Simulation of a Twin Screw Extruder

NVIDIA CUDA

OpenCL™

Gaming

OpenGL

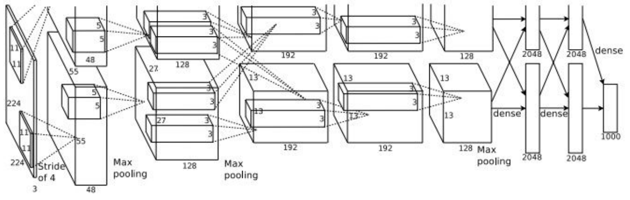
Vulkan™

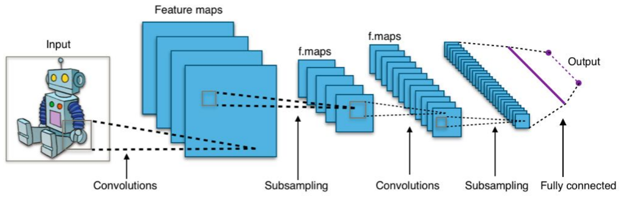
Deep Neural Network

TensorFlow™

PYTORCH

OpenAI





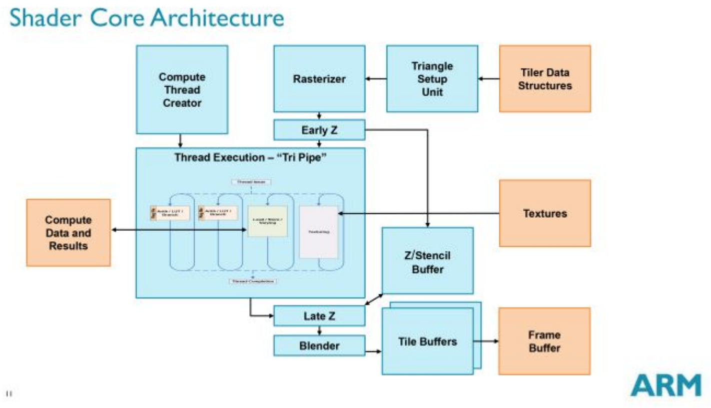
SIMD in different Processor - CPU

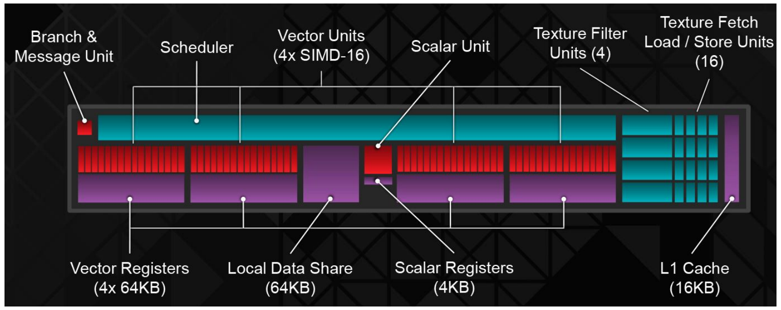
- x86
  - MMX
  - SSE
  - AVX
  - AVX-512
- ARM - Application
  - v5 DSP Extension
  - v6 SIMD
  - v7 NEON
  - v8 Advanced SIMD (NEON)
  - SVE

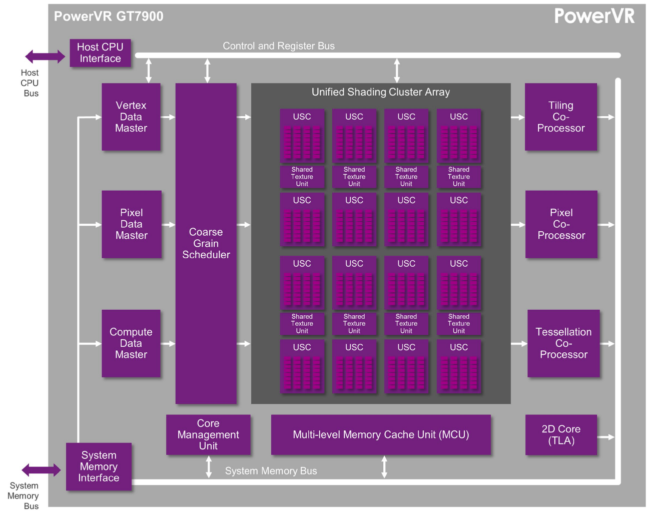
https://software.intel.com/sites/landingpage/IntrinsicsGuide/
http://infocenter.arm.com/help/topic/com.arm.doc.ihi0073a/IHI0073A_arm_neon_intrinsics_ref.pdf

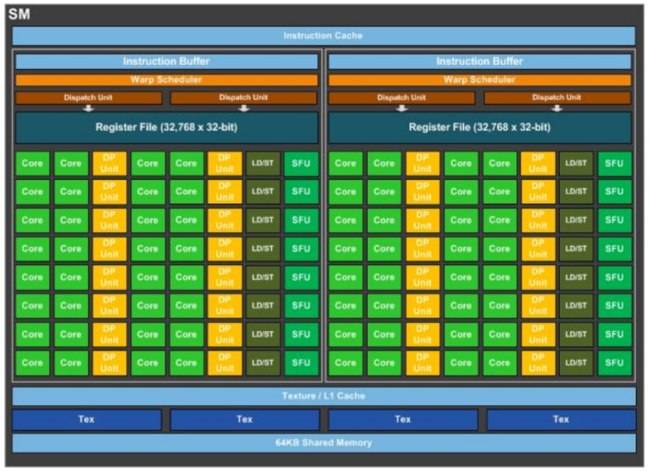
SIMD in different Processor - GPU

- SIMD
  - AMD GCN
  - ARM Mali
- WaveFront
  - Nvidia
  - Imagination PowerVR Rogue



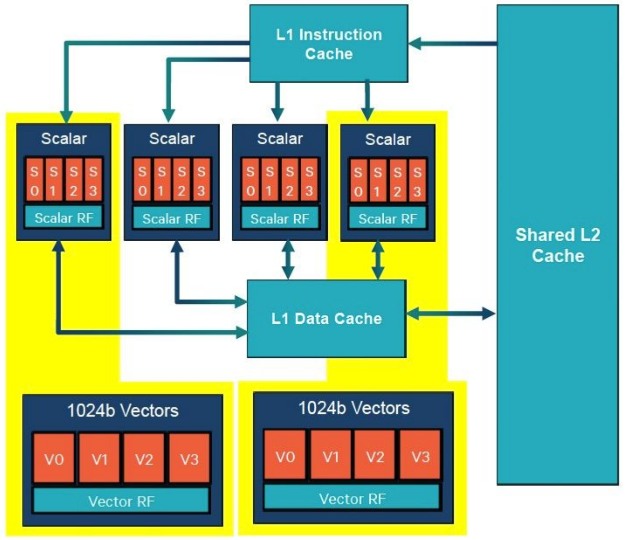


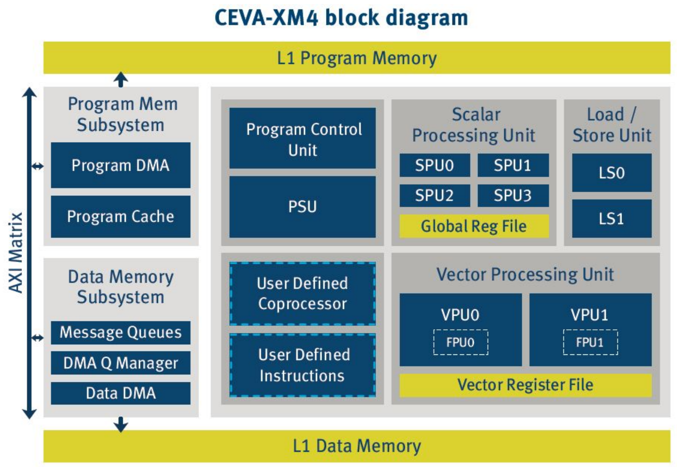


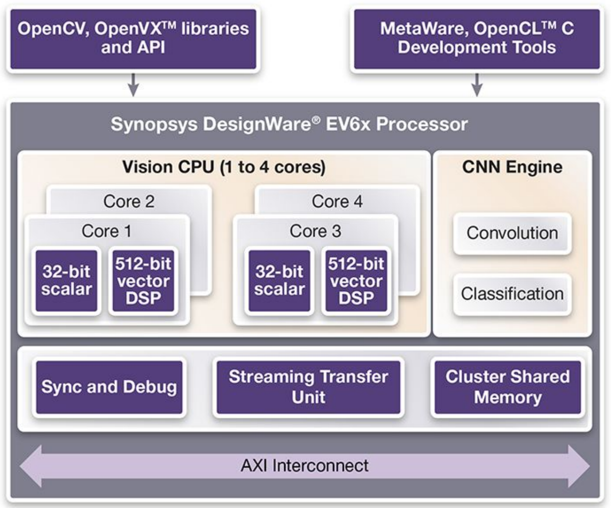


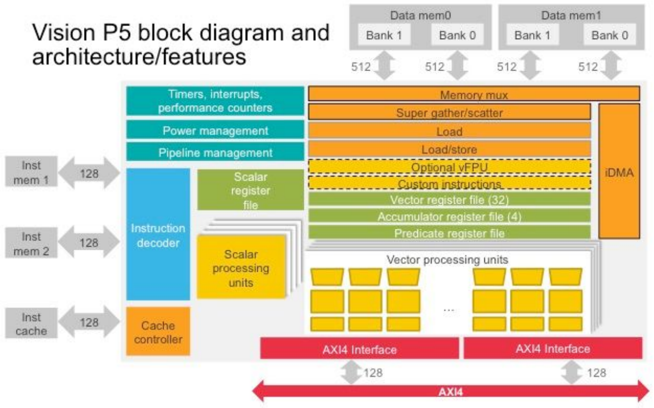
SIMD in different Processor - DSP

- Qualcomm Hexagon 600 HVX
- Cadence IVP P5
- Synopsys EV6x
- CEVA XM4

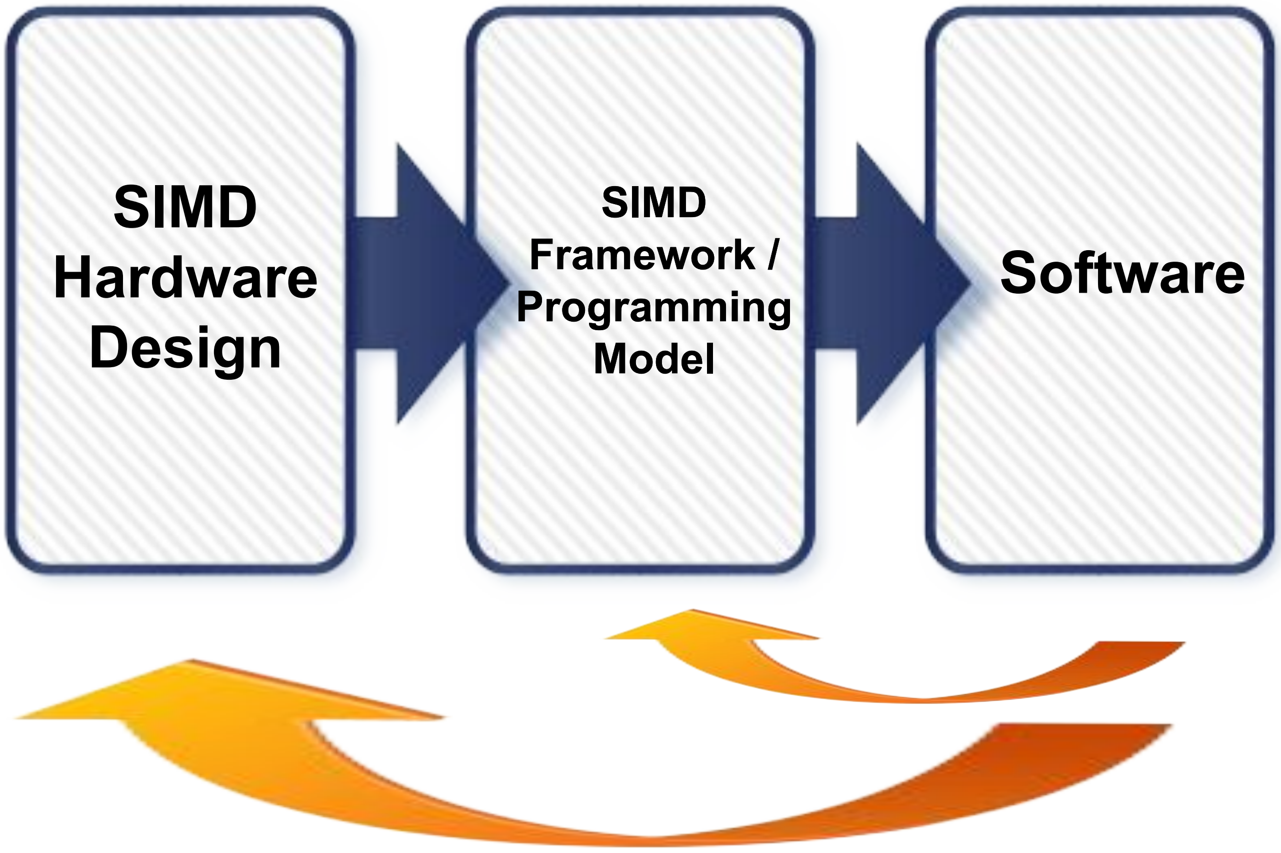








SIMD Optimization



SIMD Optimization

- Auto/Semi-Auto Method
- Compiler Intrinsics
- Specific Framework/Infrastructure
- Coding in Assembly
- What are the difficult parts of SIMD Programming?

SIMD Optimization – Auto/SemiAuto

- compiler
  - auto-vectorization optimization options
  - #pragma
  - IR optimization
- CilkPlus/OpenMP/OpenACC

Serial Code

```
for(i = 0; i < N; i++){
  A[i] = B[i] + C[i];
}
```

SIMD Pragma

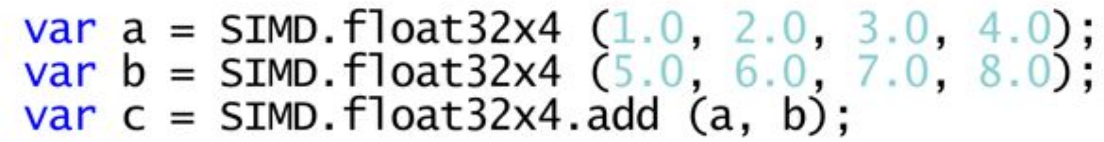
```
#pragma omp simd
for(i = 0; i < N; i++){
  A[i] = B[i] + C[i];
}
```

https://software.intel.com/en-us/articles/performance-essentials-with-openmp-40-vectorization

SIMD Optimization – Intrinsics

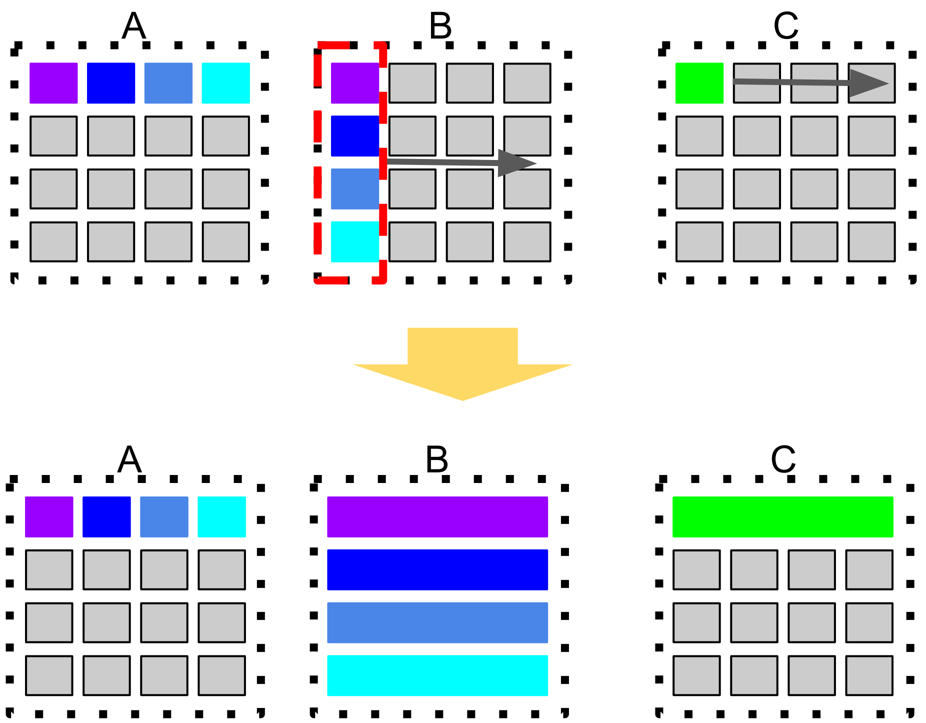
- Arch-Dependent Intrinsics
  - Intel SSE/AVX
  - ARM NEON/MIPS ASE
  - Vector-based DSP
- Common Intrisincs
  - GCC/Clang vector intrinsics
  - OpenCL / SIMD.js
- take Vector Width into consideration
- Portability between compilers

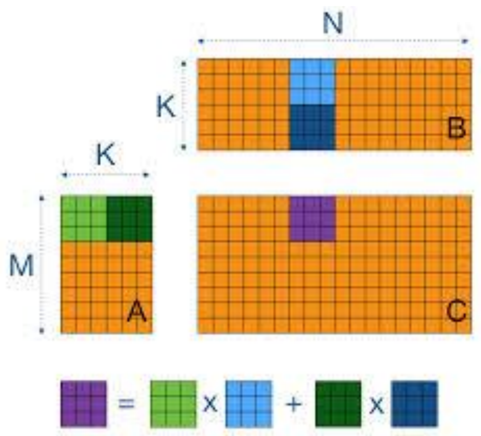
SIMD.js example from: https://01.org/node/1495



```
var a = SIMD.float32x4 (1.0, 2.0, 3.0, 4.0);
var b = SIMD.float32x4 (5.0, 6.0, 7.0, 8.0);
var c = SIMD.float32x4.add (a, b);
```

SIMD Optimization – Intrinsics



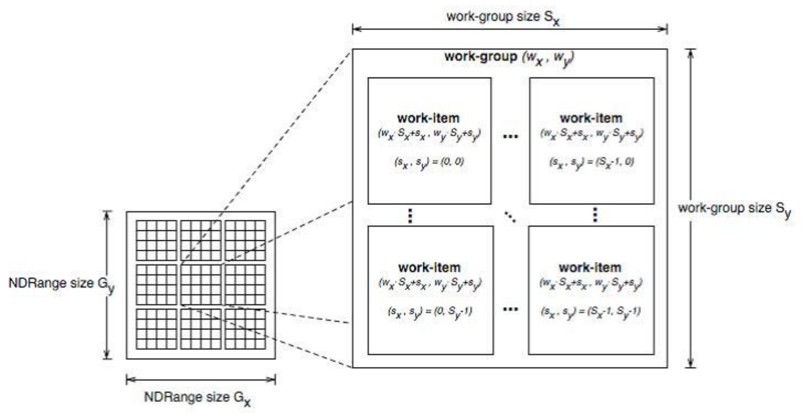


4x4 Matrix Multiplication ARM NEON Example
http://www.fixstars.com/en/news/?p=125

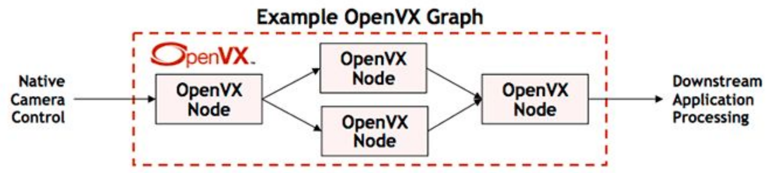
```
//...
   //Load matrixB into four vectors
   uint16x4_t vectorB1, vectorB2, vectorB3, vectorB4;

   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);
   }
//...
```

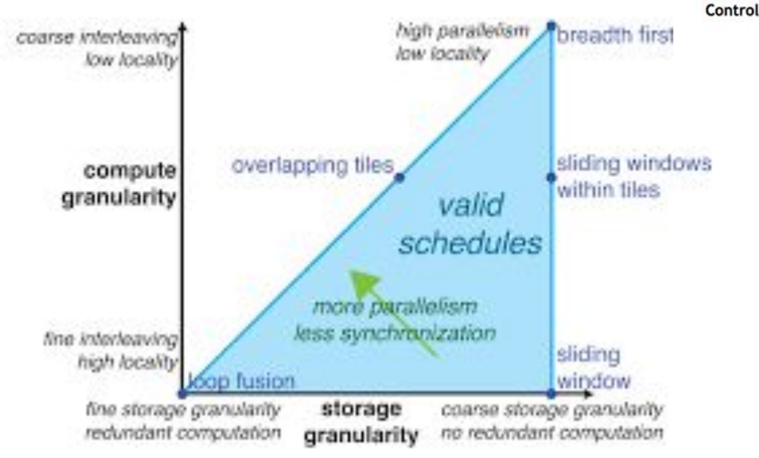
SIMD Optimization – Data Parallel Frameworks

- OpenCL/Cuda/C++AMP
- OpenVX/Halide
- SIMD-Optimized Libraries
  - Apple Accelerate
  - OpenCV
  - ffmpeg/x264
  - fftw/Ne10



workitems in OpenCL

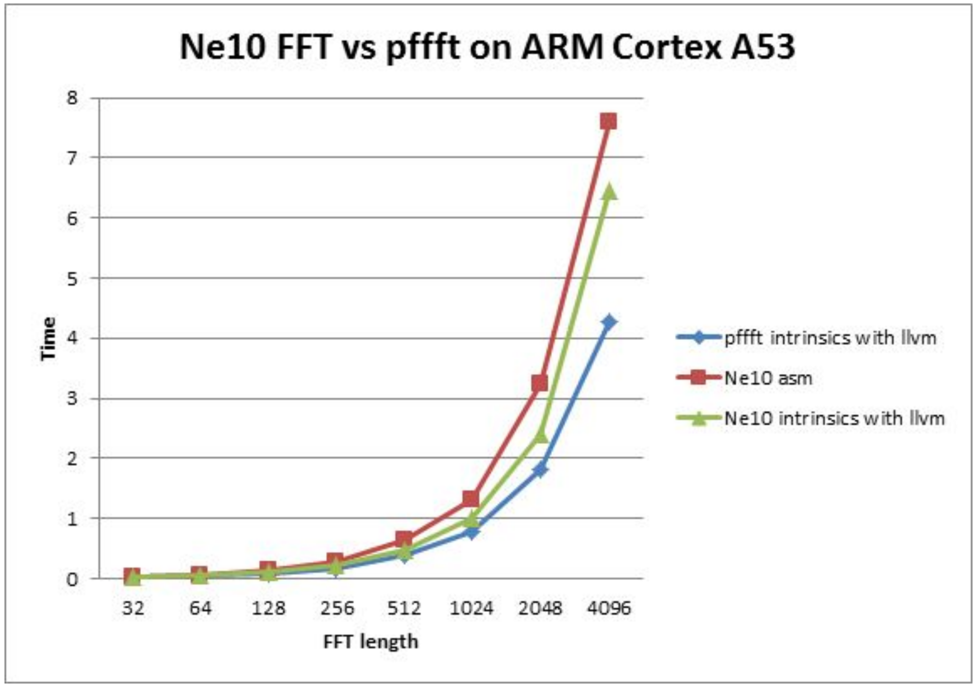



coarse interleaving low locality
compute granularity
fine interleaving high locality
high parallelism low locality
breadth first
overlapping tiles
sliding windows within tiles
valid schedules
more parallelism less synchronization
sliding window
loop fusion
fine storage granularity redundant computation
storage granularity
coarse storage granularity no redundant computation

core idea of Halide Lang

SIMD Optimization – Coding in Assembly

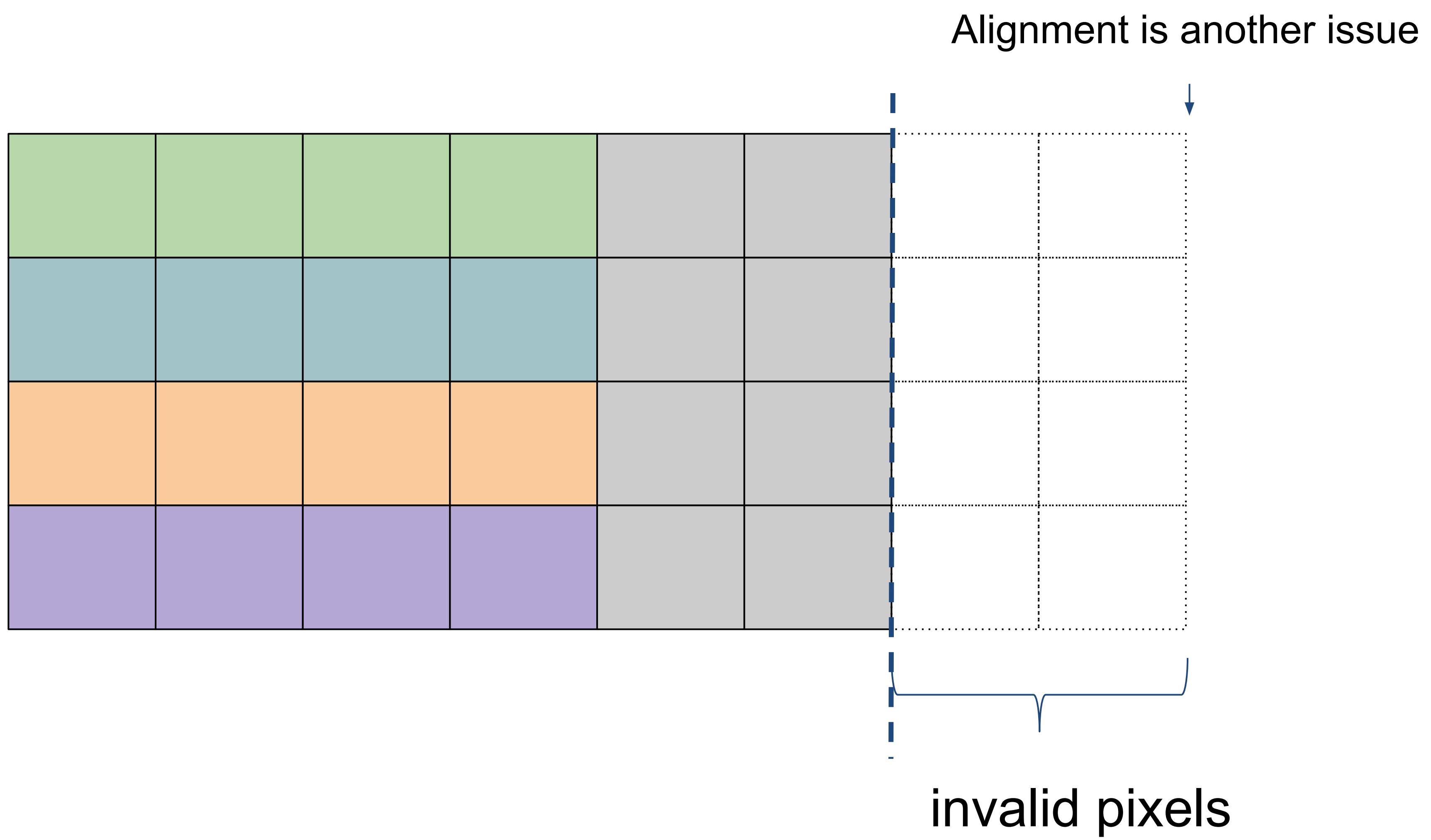
- WHY !!!!???
- Extreme Performance Optimization
  - precise code size/cycle/register usage
- The difficulty depends on ISA design and assembler



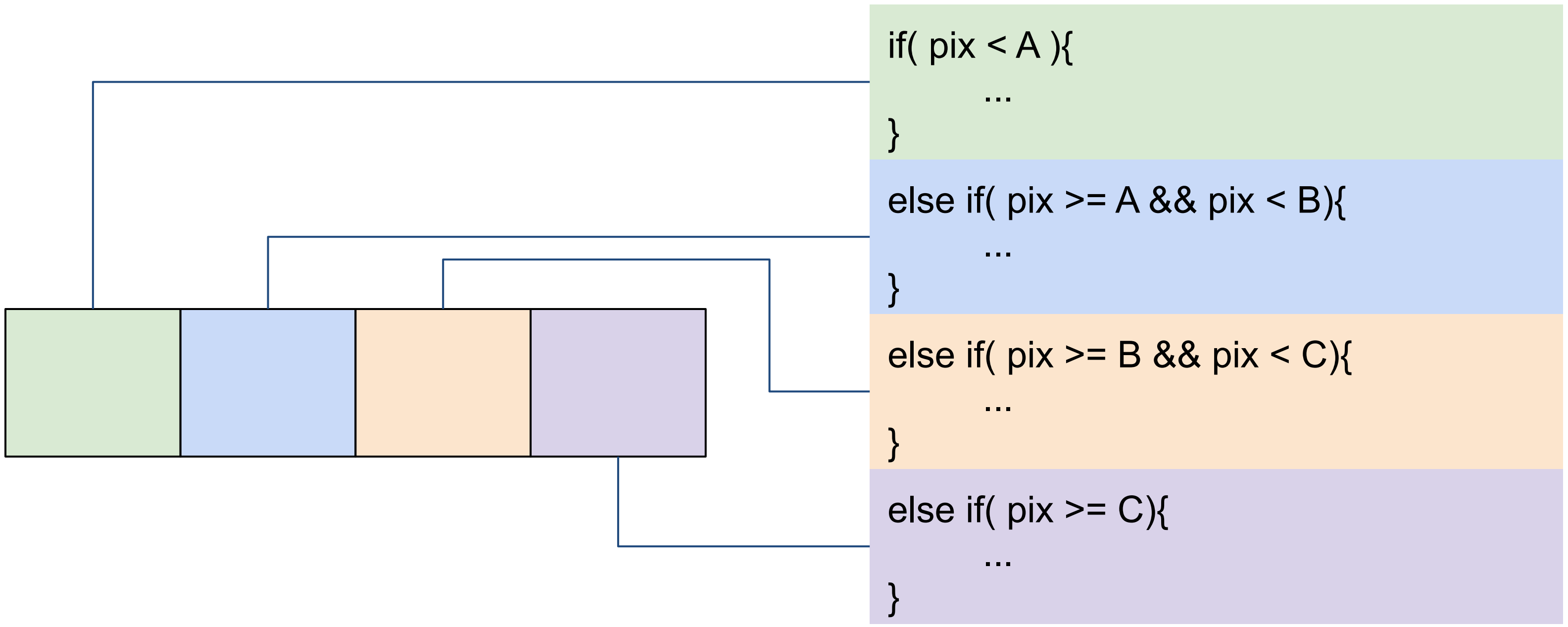
SIMD Optimization – The Difficult Parts

- Finding Parallelism in Algorithm
- Portability between different intrinsics
  - sse-to-neon
  - neon-to-sse
- Unsupported Operations
  - Division, High-level function (eg: math functions)
- Floating-Point
  - Unsupported/cross-deivce Compatiblilty
- Boundary handling
  - Padding, Predication, Fallback
- Divergence
  - Predication, Fallback to scalar
- Register Spilling
  - multi-stages, # of variables + braces, assembly
- Non-Regular Access/Processing Pattern/Dependency
  - LUT, AoS(Array of Structure), content dependent flow
  - Multi-stages/Reduction ISA/Enhanced DMA

Boundaries

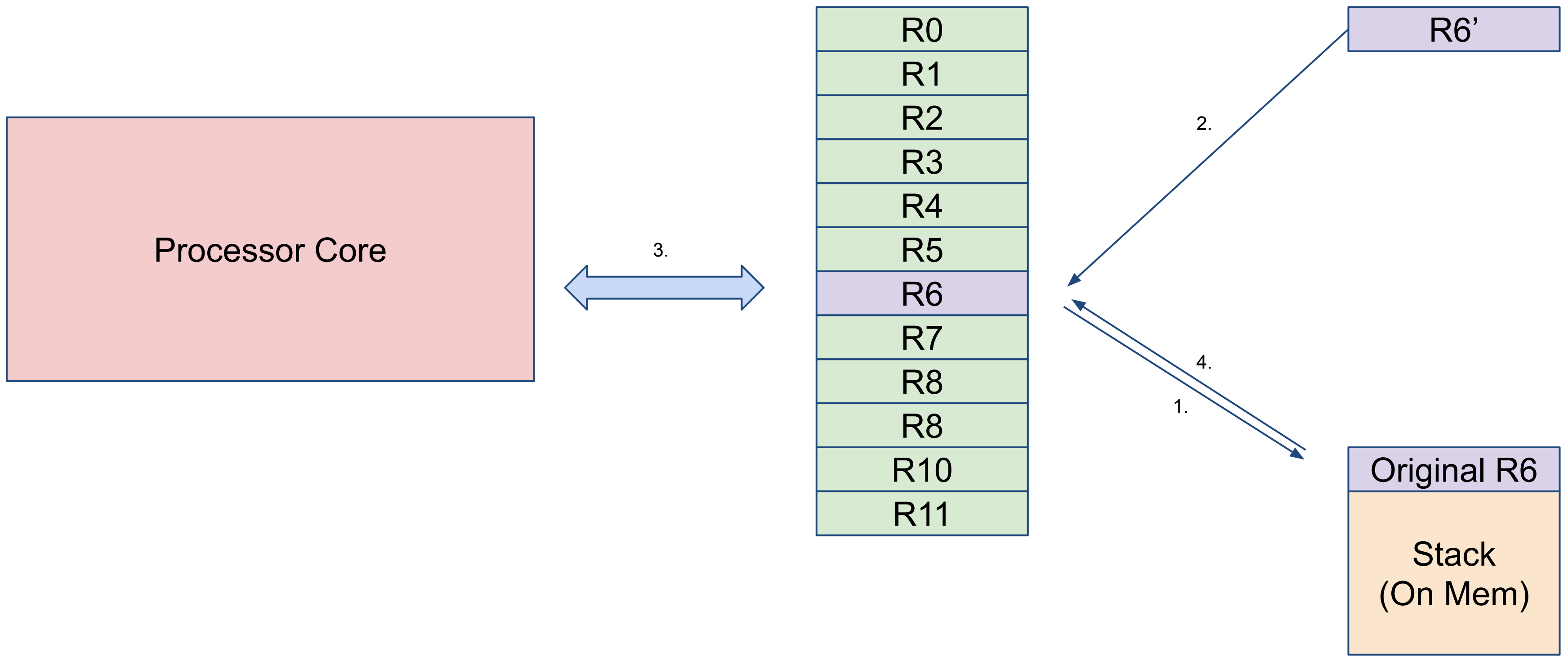


Divergence

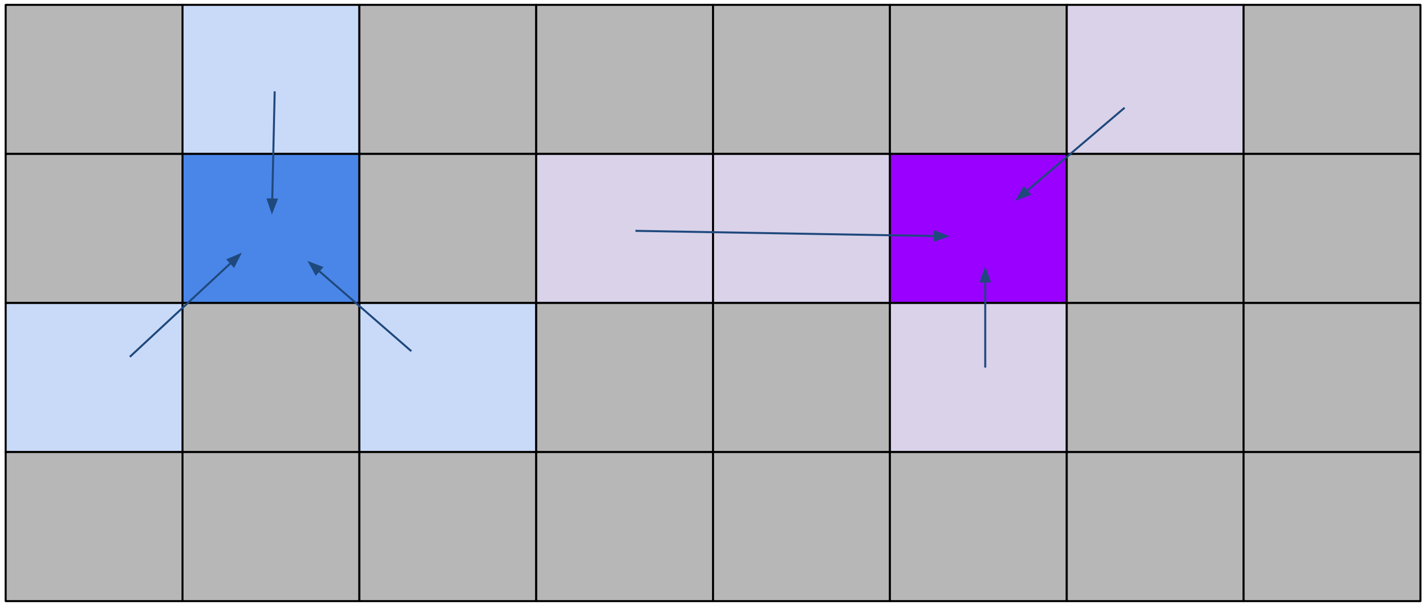


```
if( pix < A ){
        ...
}
else if( pix >= A && pix < B){
        ...
}
else if( pix >= B && pix < C){
        ...
}
else if( pix >= C){
        ...
}
```

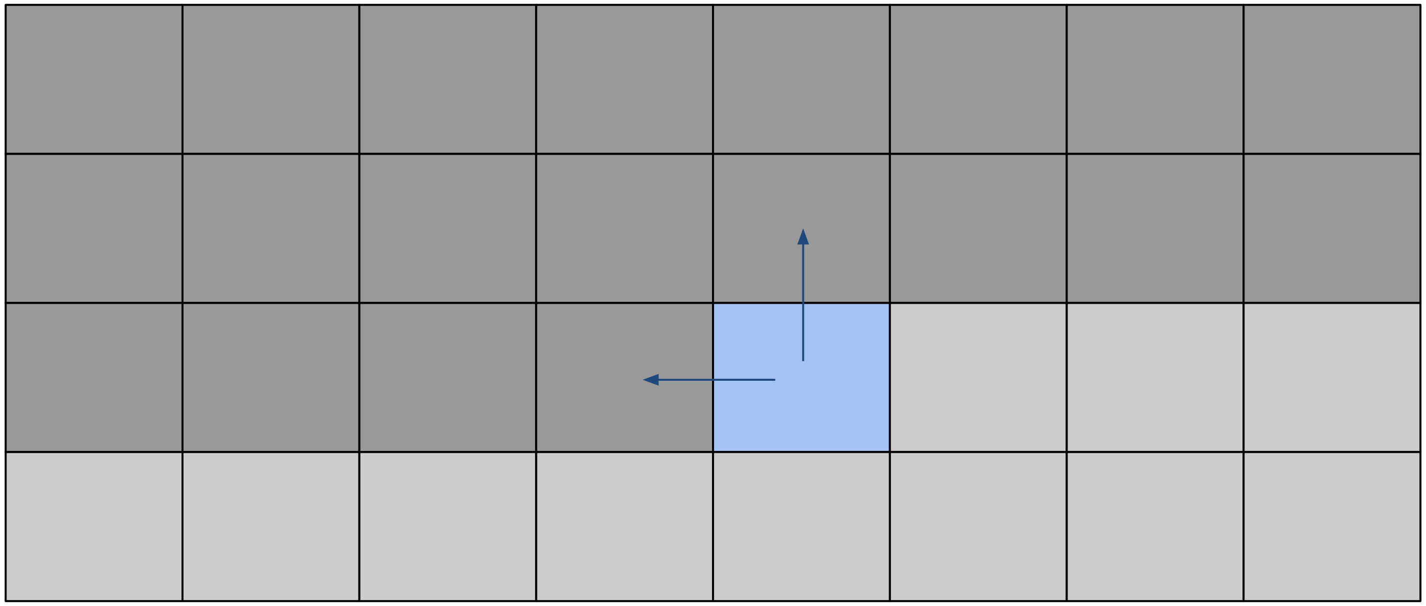
Register Spilling



Non-Regular Access Pattern



Dependencies



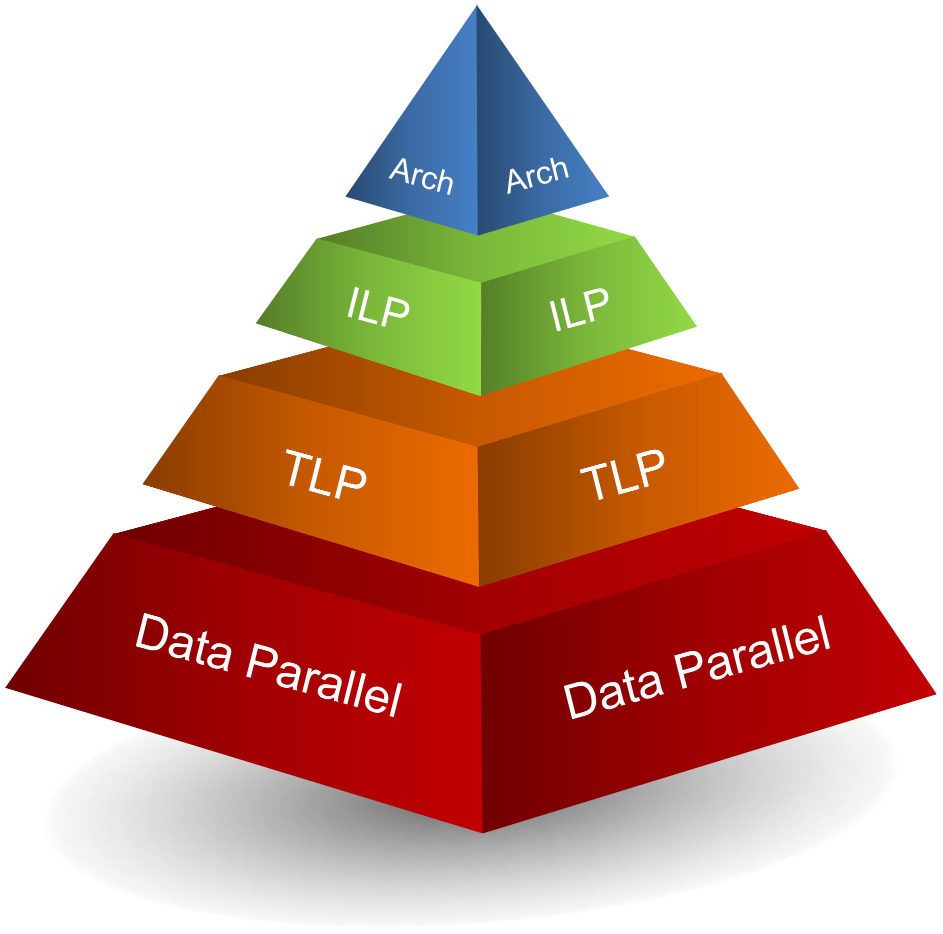
What are important in SIMD?

Architecture

Core, ALUs, pipeline, memory (registers, cache, local mem …), ISA latency

ILP

Instruction-Level Parallelism: SuperScalar, VLIW, SIMD, Vector Machine



TLP

Thread-Level Parallelism: Hardware Threads, Software Threads

Data Parallel

Per-Entry parallelism friendly algorithm.

What are important in SIMD?

- ISA design
- Memory Model
- Thread / Execution Model
- Scalability / Extensibility
- The Trends

What are important in SIMD? ISA Design

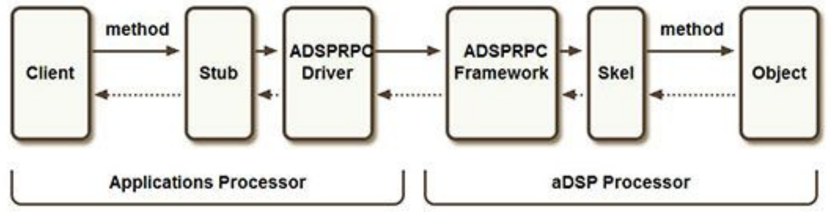
- SuperScalar vs VLIW
- vector register design
  - Unified or Dedicated float/predication/accumulators registers
- ALU
- Inter-Lane
  - reduction, swizzle, pack/unpack
- Multiply
- Memory Access
  - Load/Store, Scatter/Gather, LUT, DMA ...
- Scalar $\leftrightarrow$ Vector

What are important in SIMD? Memory Model

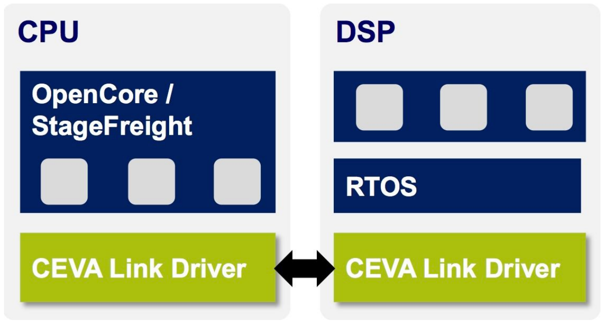
- DATA! DATA! DATA!
  - it takes great latency to pull data from DRAM to register!
- Buffers Between Host Processor and SIMD Processor
  - Handled in driver/device-side
- TCM
  - DMA
  - relative simple hw/control/understand
  - fixed cycle (good for simulation/estimation)
- Cache
  - prefetch
  - transparent
  - portable
  - performance scalability
  - simple code flow
- Mixed
  - powerful

What are important in SIMD? Thread / Execution Model

- The flow to trigger DSP to work



Qualcomm FastRPC
w/ IDL Compiler



CEVA Link

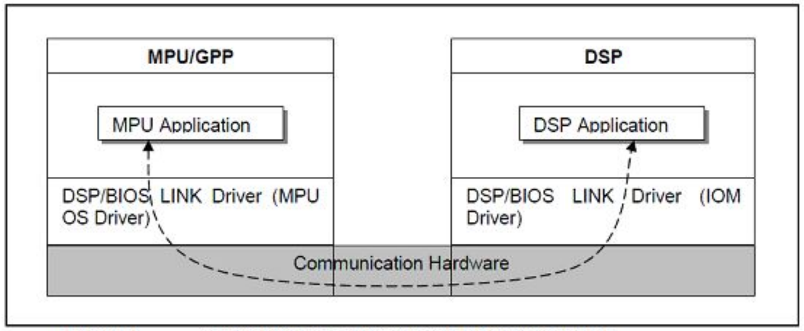
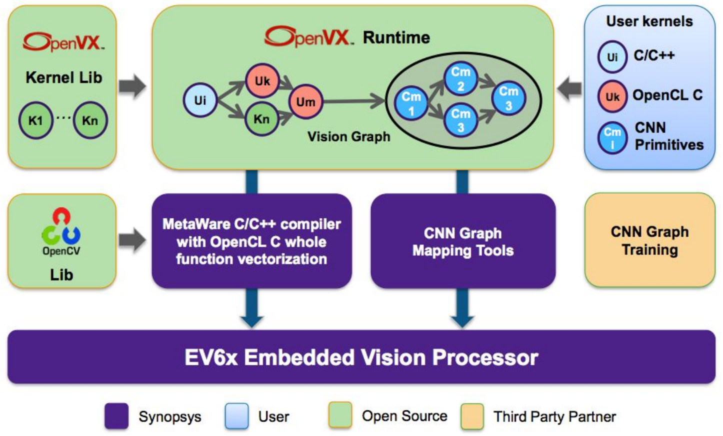

MPU/GPP
MPU Application
DSP/BIOS LINK Driver (MPU OS Driver)
DSP
DSP Application
DSP/BIOS LINK Driver (IOM Driver)
Communication Hardware

Figure 1. GPP-DSP connectivity through DSP/BIOS™ LINK

TI DSP Link

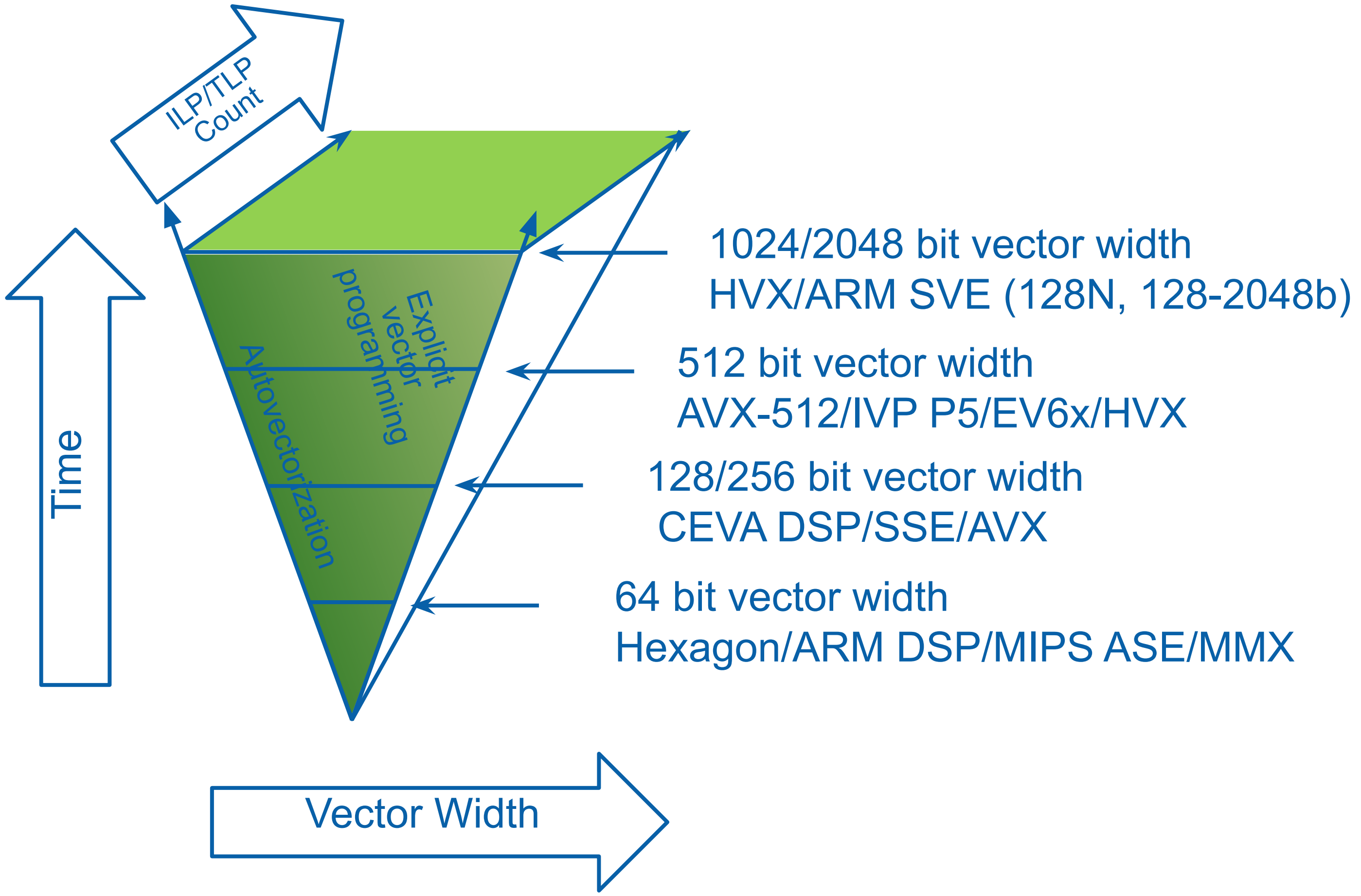


Synopsys EV6x SW ecosystem

What are important in SIMD? Scalability/Extensibility

- What to do when ONE core doesn’t meet performance requirement?
- HWA/Co-processor Interface
- Add another core?
  - Cost
  - Multicore Programming Mode

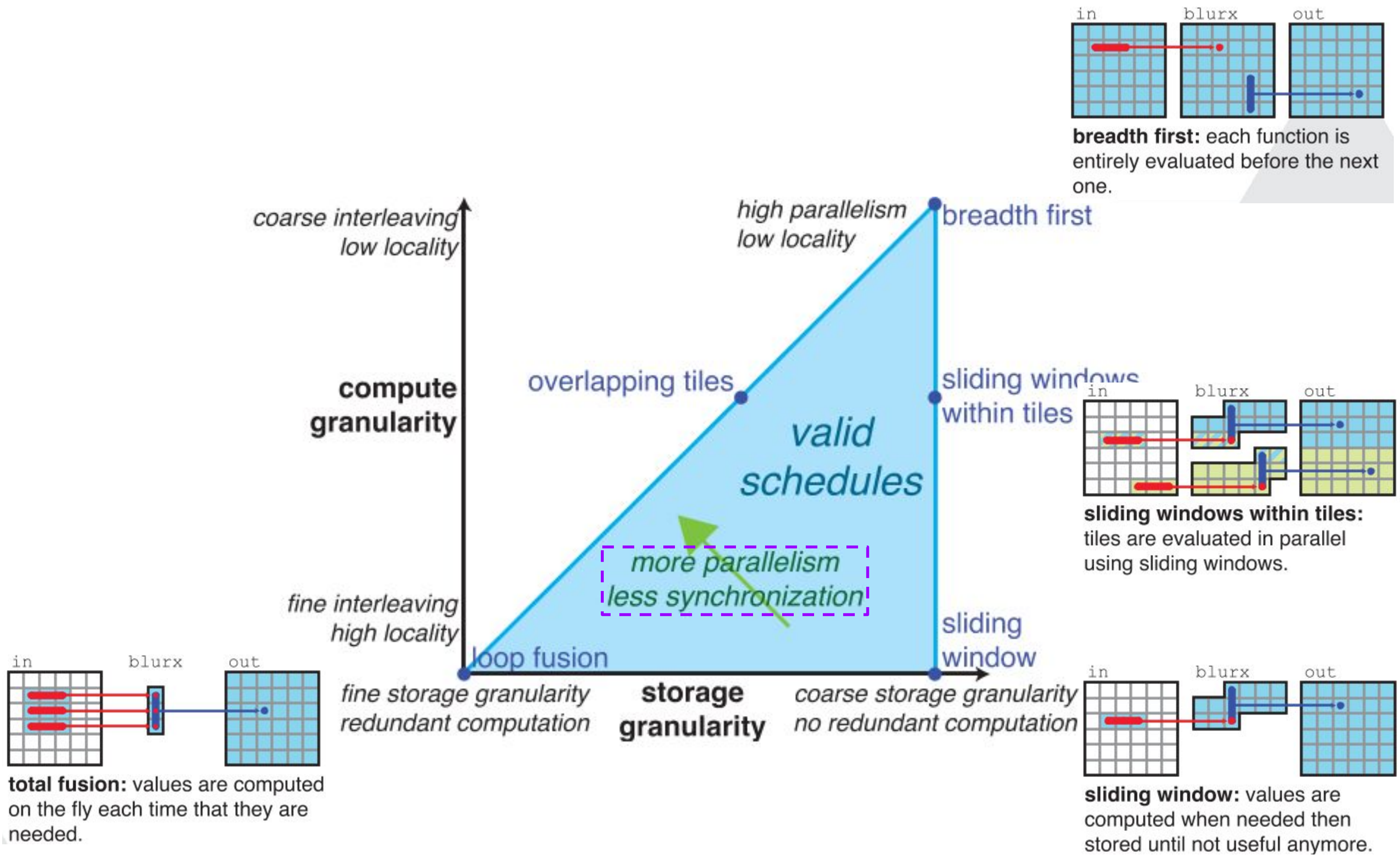
What are important in SIMD? Trends



Thank You, Q & A

What does make optimization works?

Halide – Optimization Concepts



in
blurx
out
breadth first: each function is entirely evaluated before the next one.
coarse interleaving
low locality
high parallelism
low locality
breadth first
compute granularity
overlapping tiles
sliding windows within tiles
valid schedules
in
blurx
out
sliding windows within tiles: tiles are evaluated in parallel using sliding windows.
more parallelism
less synchronization
fine interleaving
high locality
sliding window
loop fusion
in
blurx
out
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.
sliding window: values are computed when needed then stored until not useful anymore.

Domain-Specific Architecture (DSA)

Pixel Visual Core as Example

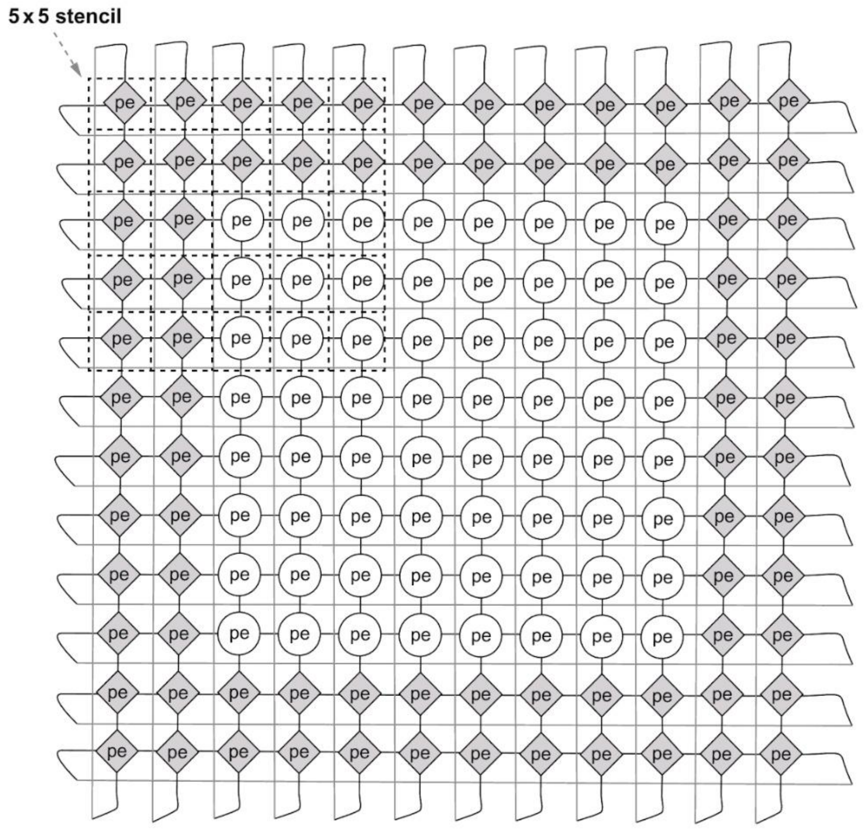
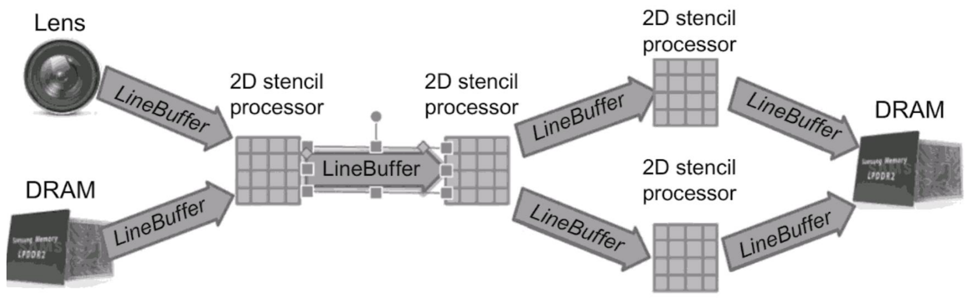


Figure 7.34 The two-dimensional array of full processing elements *(shown as unshaded circles)* surrounded by two layers of simplified processing elements *(shaded diamonds)* called a *halo*. In this figure, there are 8 × 8 or 64 full PEs with 80 simplified PEs in the halo. (Pixel Visual Core actually has 16 × 16 or 256 full PEs and two layers in its halo and thus 144 simplified PEs.) The edges of the halo are connected *(shown as gray lines)* to form a torus. Pixel Visual Core does a series of two-dimensional shifts across all processing elements to move the neighbor portions of each stencil computation into the center PE of the stencil. An example 5 × 5 stencil is shown in the upper-left corner. Note that 16 of the 25 pieces of data for this 5 × 5 stencil location come from halo processing elements.


Lens
DRAM
LineBuffer
LineBuffer
2D stencil processor
LineBuffer
2D stencil processor
LineBuffer
2D stencil processor
LineBuffer
2D stencil processor
LineBuffer
LineBuffer
LineBuffer
DRAM