Expressing **Heterogeneous** Parallelism in **C++** with Intel **Threading Building Blocks**

James Reinders
James Reinders Consulting LLC

Pablo Reble, Jim Cownie
Intel

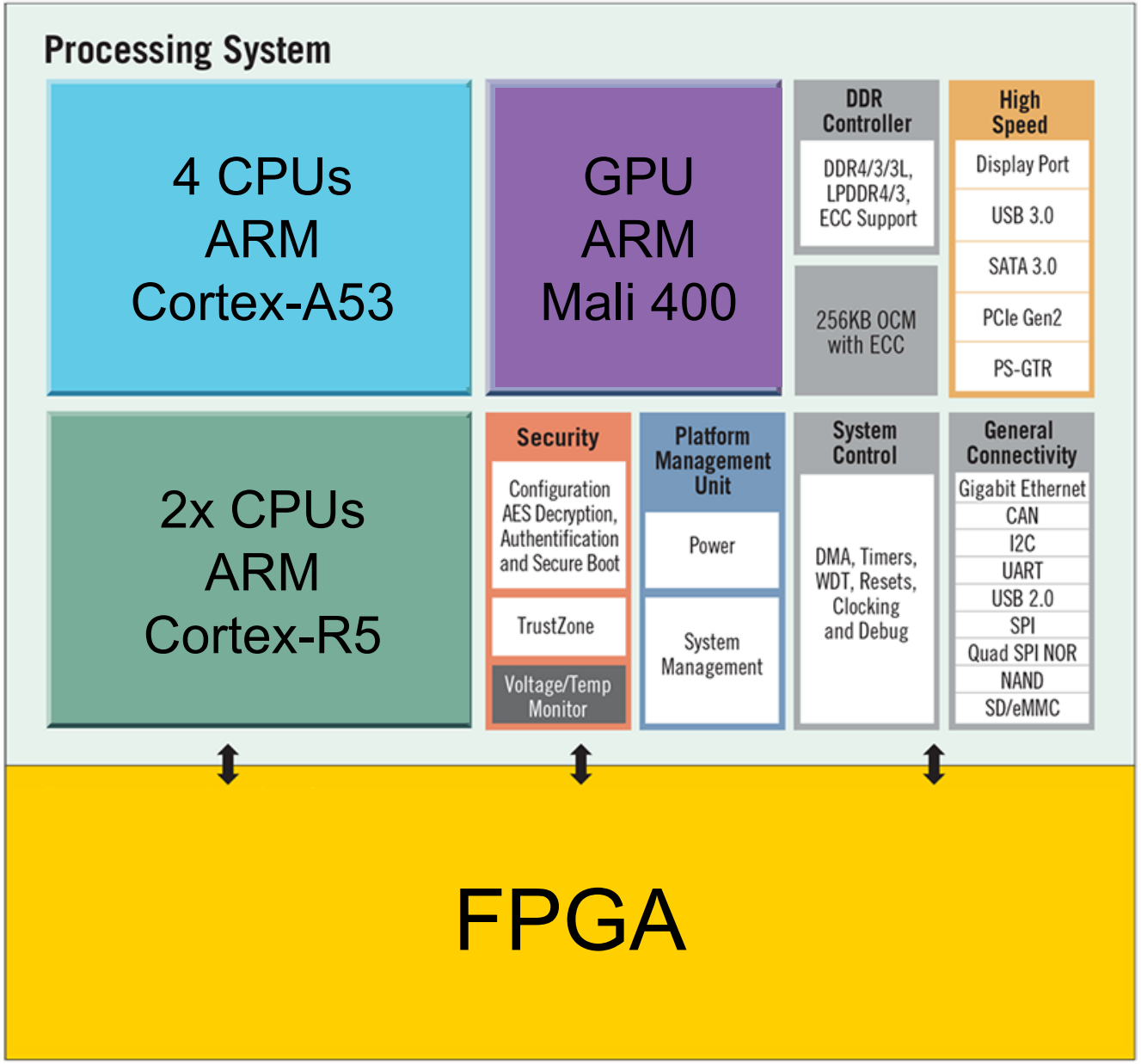
Rafael Asenjo
Dept. of Computer Architecture
University of Malaga, Spain.

UNIVERSIDAD DE MÁLAGA

Agenda

- Part I: Motivation & Background (30’)
- Part II: TBB heterogeneous features (50’)
- Part III: Heterogen. Flow Graph node (80’)
- Part IV: Templates on top of TBB (20’)

Xilinx Zynq UltraScale+ MPSoC



Note: Illustration not drawn to scale.

Part I

- **Motivation**

- Hardware
  - Integrated GPUs (iGPUs)
  - Integrated FPGAs (iFPGAs)

- Software
  - Programming models for heterogeneous chips with GPUs
  - Programming models for heterogeneous chips with FPGAs

Motivation

- A new mantra: Power and Energy saving
- In all domains

UNIVERSIDAD DE MÁLAGA

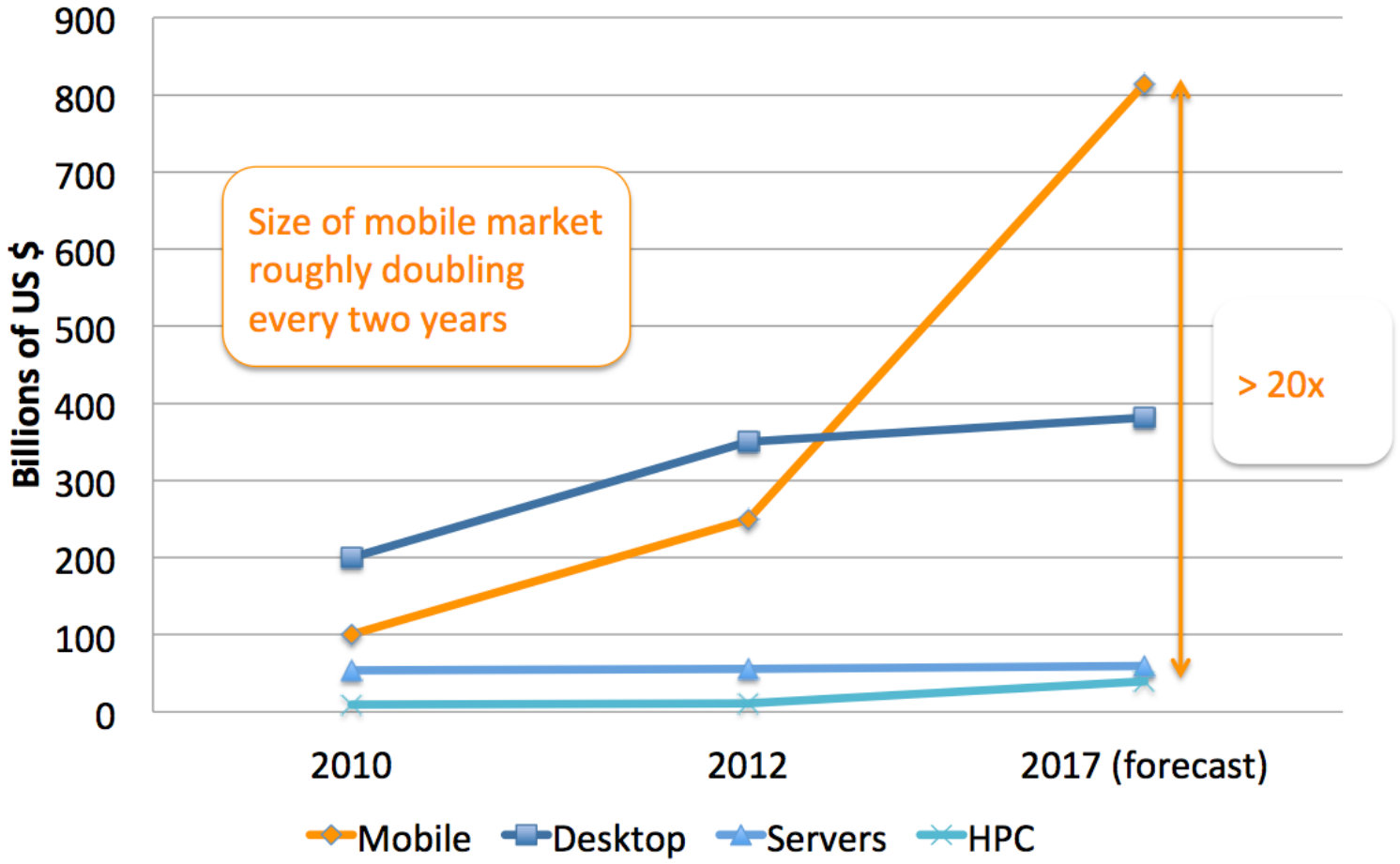
Motivation

- **GPUs came to rescue**:
  - Massive Data Parallel Code at a low price in terms of power
  - Supercomputers and servers: NVIDIA
- **GREEN500 Top 10**:
- Out of 500:
  - 65 systems w. NVIDIA
  - 23 systems w. Xeon Phi

Green500 Rank	MFLOPS/W	System
1	9462.1	NVIDIA DGX-1, Xeon E5-2698v4 20C 2.2GHz, Infiniband EDR, NVIDIA Tesla P100
2	7453.5	Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100
3	6673.8	ZettaScaler-1.6, Xeon E5-2618Lv3 8C 2.3GHz, Infiniband FDR, PEZY-SCnp
4	6051.3	Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway
5	5806.3	PRIMERGY CX1640 M1, Intel Xeon Phi 7210 64C 1.3GHz, Intel Omni-Path
6	4985.7	PRIMERGY CX1640 M1, Intel Xeon Phi 7250 68C 1.4GHz, Intel Omni-Path
7	4688.0	Cray XC40, Intel Xeon Phi 7230 64C 1.3GHz, Aries interconnect
8	4112.1	Cray CS-Storm, Intel Xeon E5-2680v2 10C 2.8GHz, Infiniband FDR, Nvidia K80
9	4086.8	Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect
10	3836.6	KOI Cluster, Intel Xeon Phi 7230 64C 1.3GHz, Intel Omni-Path

Motivation

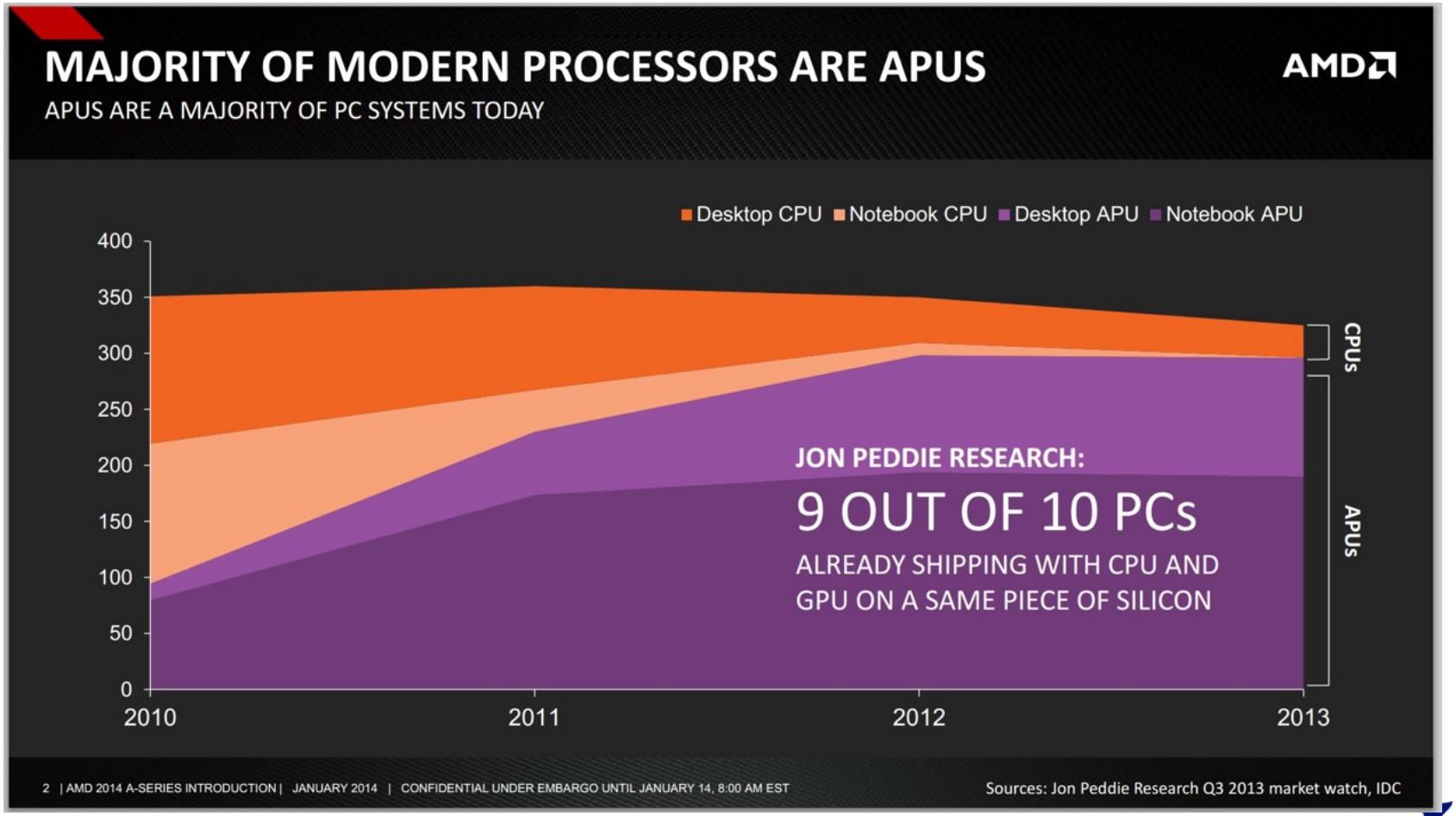
- There is (parallel) life beyond supercomputers:



Source: IDC. HPC forecast Intersect360

Motivation

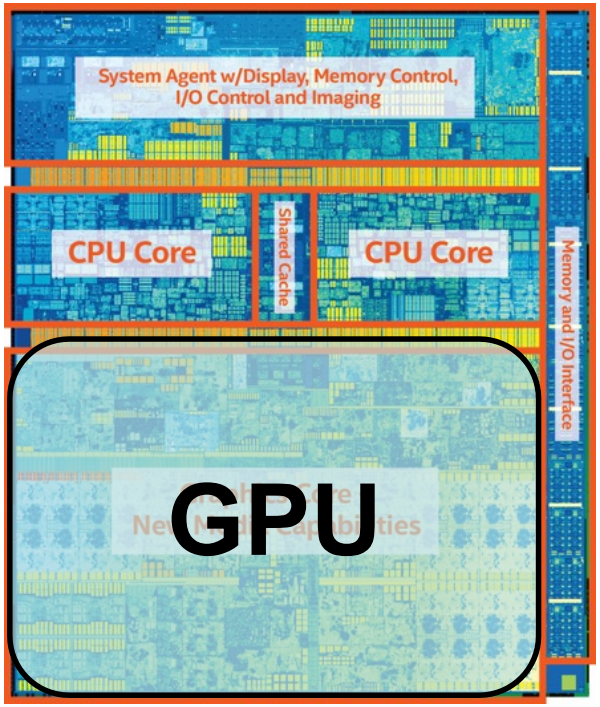
- Plenty of GPUs elsewhere:
  - Integrated GPUs **on more than 90%** of shipped processors



Motivation

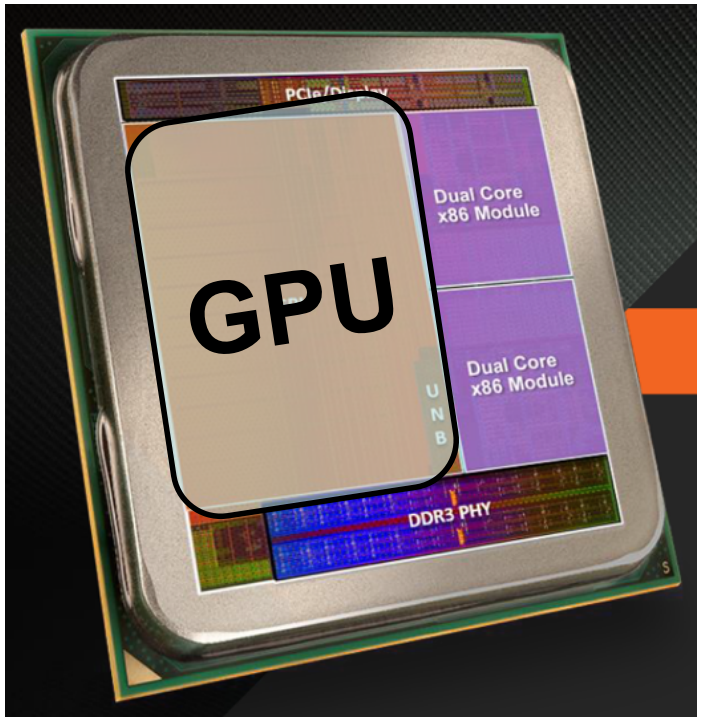
- Plenty of GPUs on desktops and laptops:
  - Desktops (35 – 130 W) and laptops (15 – 60 W):

Intel Kaby Lake



http://www.anandtech.com/show/10610/intel-announces-7th-gen-kaby-lake-14nm-plus-six-notebook-skus-desktop-coming-in-january/2

AMD APU Kaveri



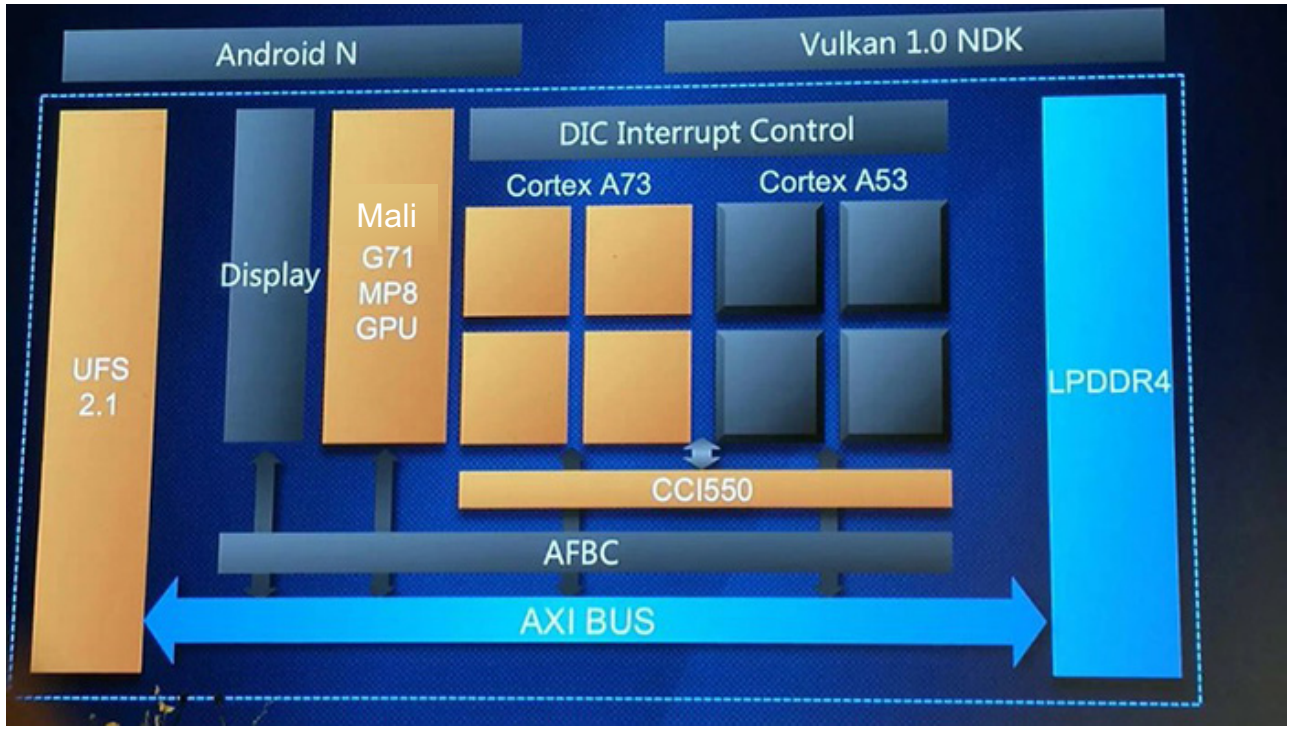
http://www.techspot.com/photos/article/770-amd-a8-7600-kaveri/

Motivation

- Plenty of integrated GPUs in mobile devices too.

Huawei HiSilicon Kirin 960 (2 - 6 W)

Huawei Mate 9



http://www.anandtech.com/show/10766/huawei-announces-hisilicon-kirin-960-a73-g71

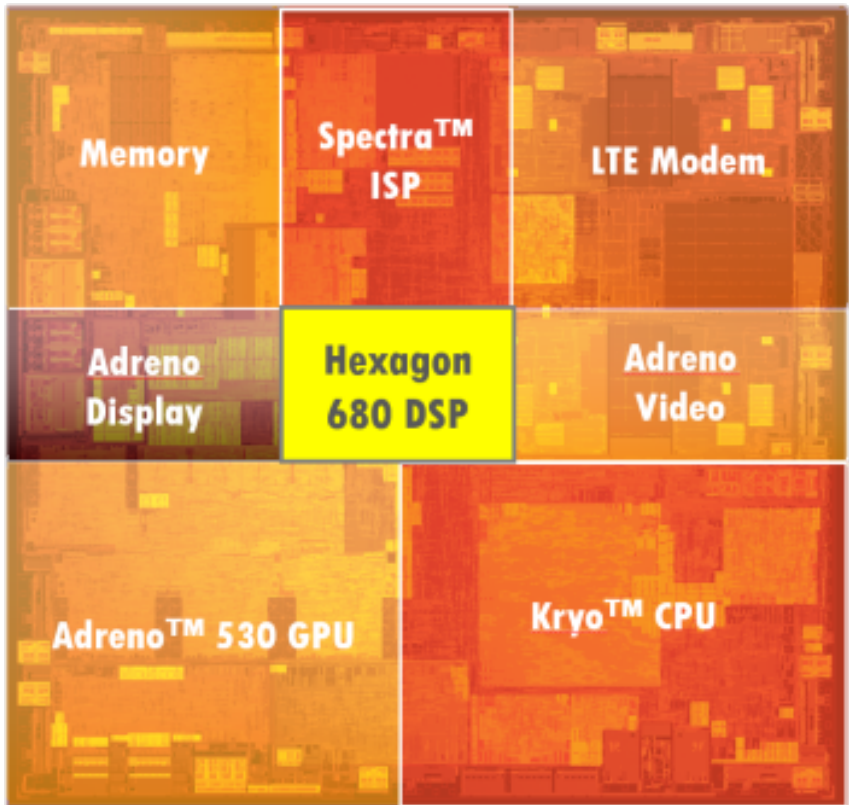
Motivation

- And user-programmable DSPs as well:

Qualcomm Snapdragon 820/821 (2 - 6 W)

Samsung Galaxy S7

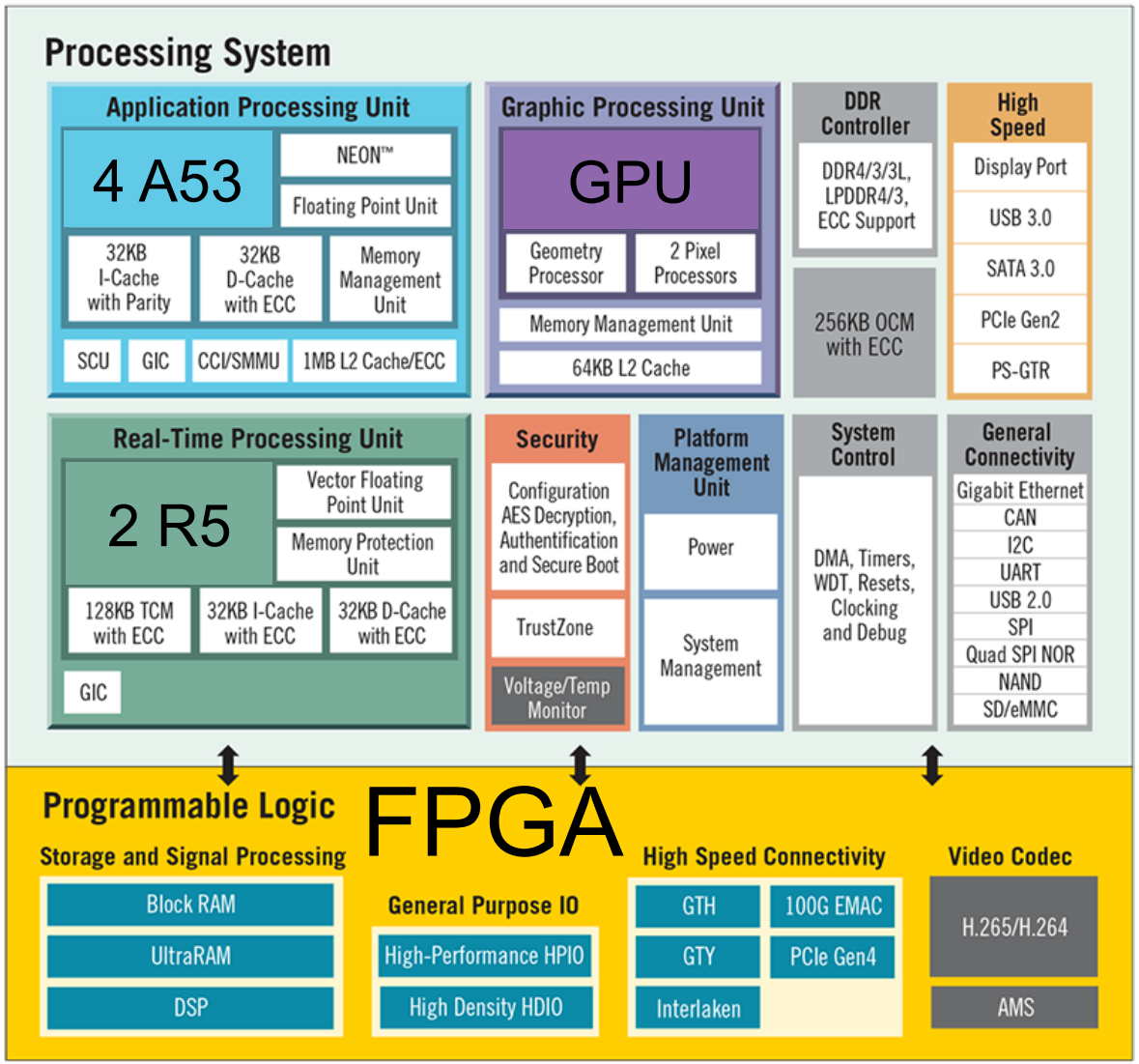
Google Pixel



https://www.qualcomm.com/products/snapdragon/processors/820

Motivation

- And **FPGAs** too…
  - ARM + iFPGA
    - Altera **Cyclon V**
    - Xilinx **Zynq**-7000
  - **Intel + Altera**'s FPGA
    - Heterogeneous Architecture Research Platform
- CPU + iGPU + iFPGA
  - Xilinx Zynq **UltraScale+**
    - 4 Cortex A53
    - 2 Cortex R5
    - GPU Mali 400
    - FPGA

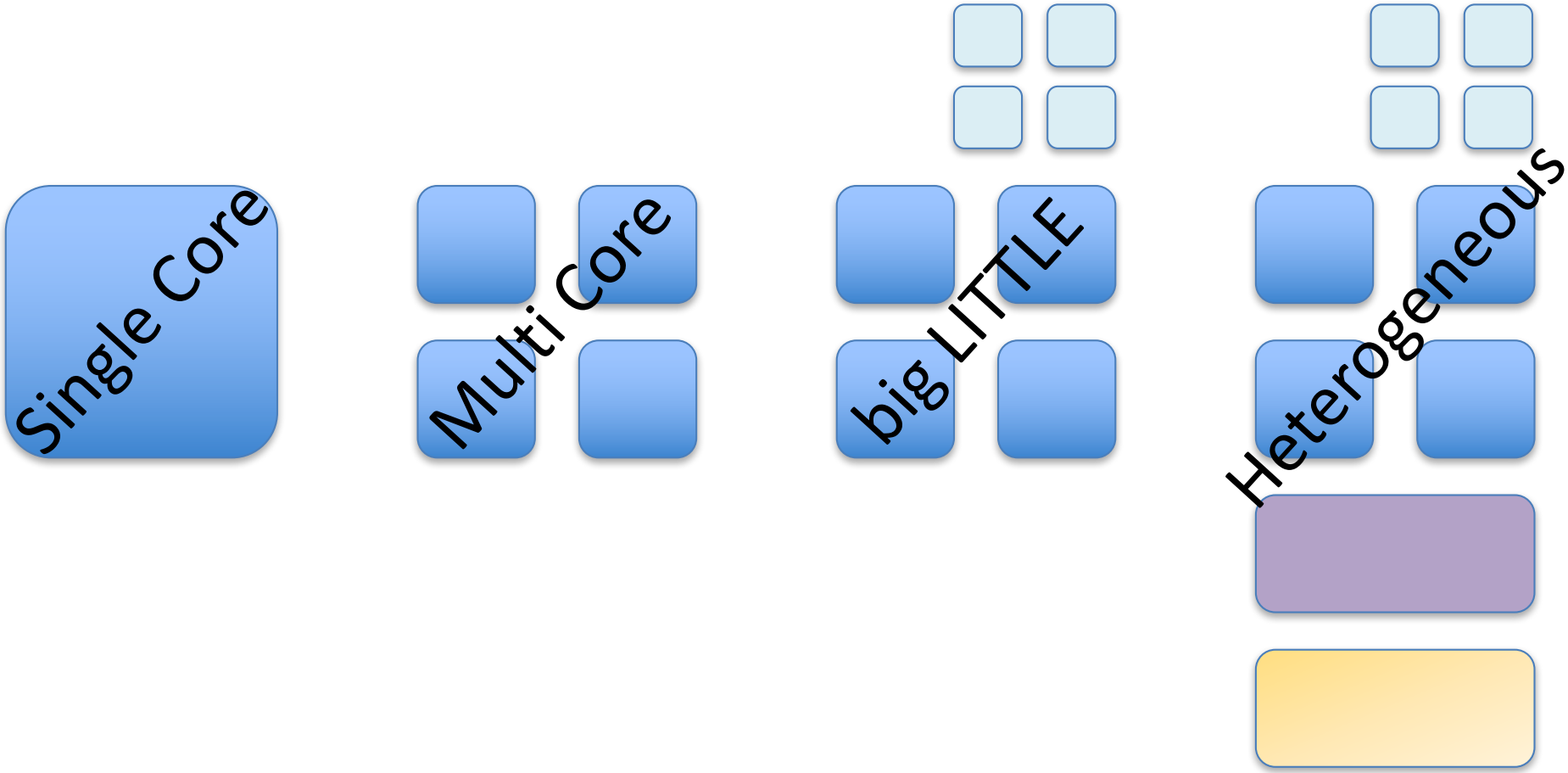


Note: Illustration not drawn to scale.

http://www.analyticsengines.com/developer-blog/xilinx-announce-new-zynq-architecture/

Motivation

- Overwhelming heterogeneity:



Motivation

- Plenty of room for improvements
  - Want to **make the most** out of the CPU, the GPU and the FPGA
  - Taking **productivity** into account:

    Productivity = Performance - Pain

    - High level abstractions in order to hide HW details
    - Lack of high-level productive programming models
    - “Homogeneous programming of heterogeneous architectures”
  - Taking **energy consumption** into account:

    ~~Performance ➔ Performance / Watt~~

    Performance ➔ Performance / Joule

Productivity = Performance / Joule - Pain

Part I

- Motivation

- **Hardware**
  - Integrated GPUs (**iGPUs**)
  - Integrated FPGAs (iFPGAs)

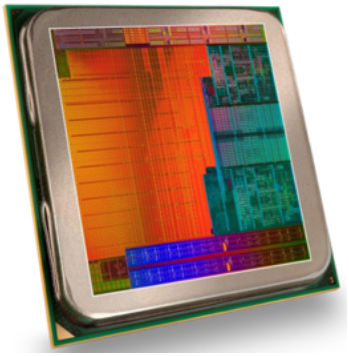
- Software
  - Programming models for heterogeneous chips with GPUs
  - Programming models for heterogeneous chips with FPGAs

Hardware: iGPUs

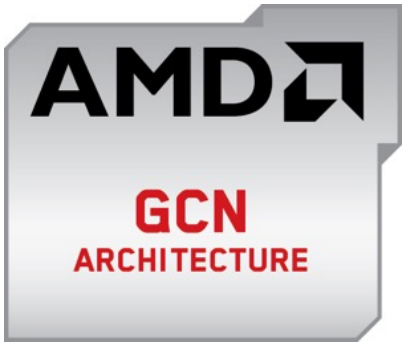
Intel Skylake

Iris Pro Graphics

AMD Kaveri

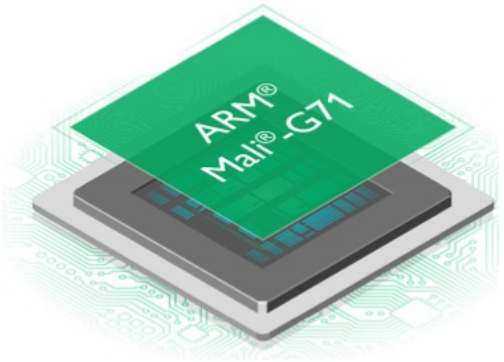


Graphic Core Next



HiSilicon Kirin 960

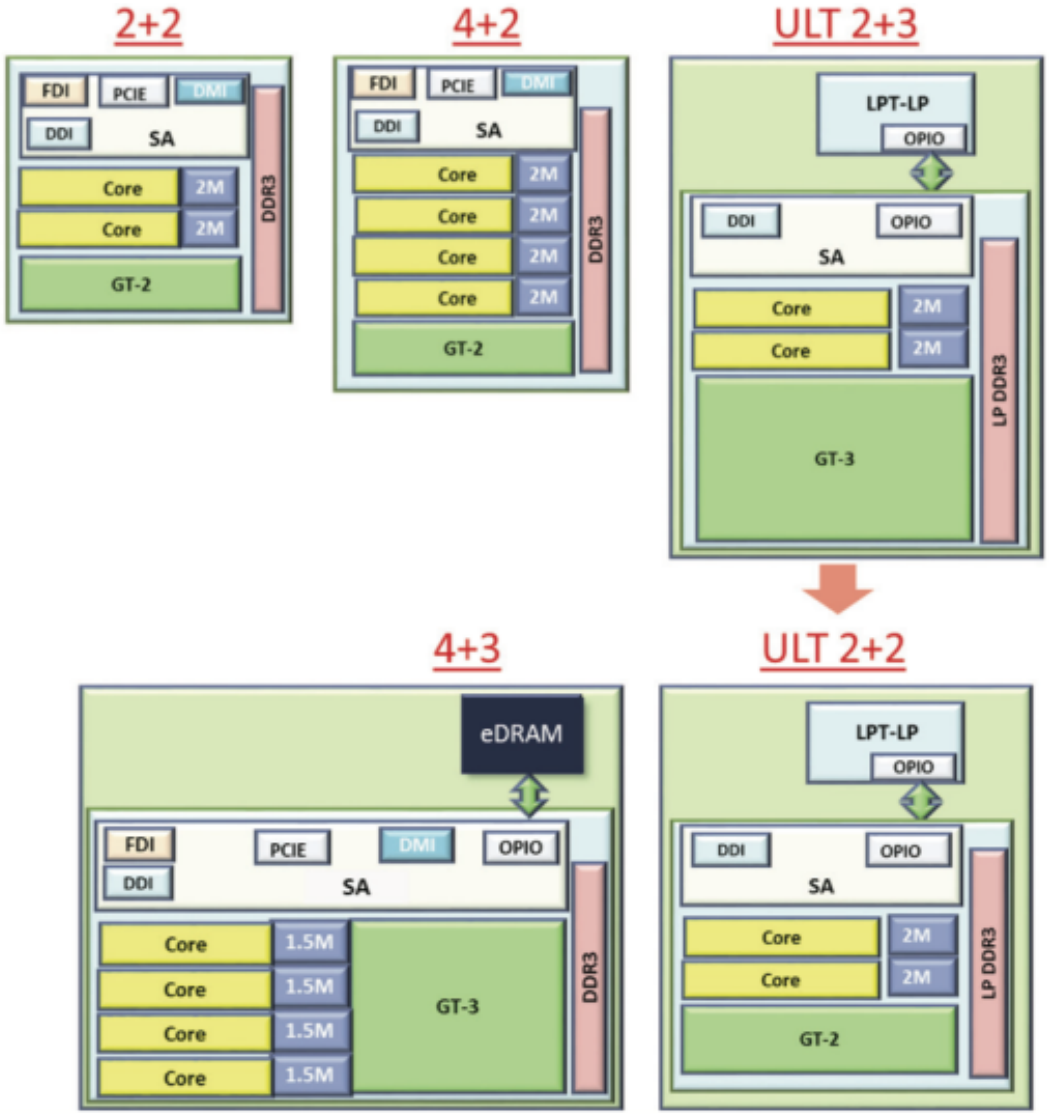
ARM Mali-G71



Intel Skylake – Kaby Lake

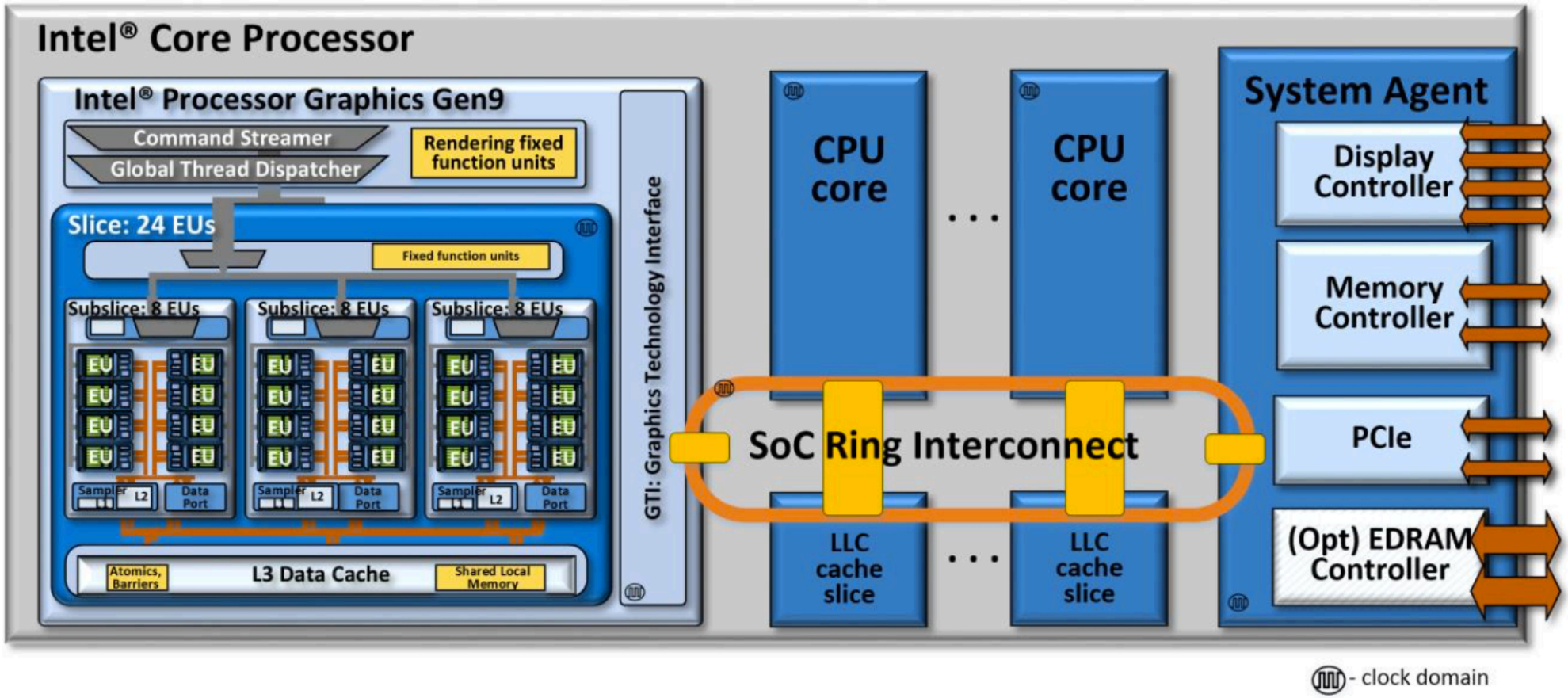
- Modular design
  - 2 or 4 cores
  - GPU
    - GT-1: 12 EU
    - GT-2: 24 EU
    - GT-3: 48 EU
    - GT-4: 72 EU

- SKU options
  - 2+2 (4.5W, 15W)
  - 4+2 (35W -- 91W)
  - 2+3 (15W, 28W,...)
  - 4+4 (65W)
  - ....



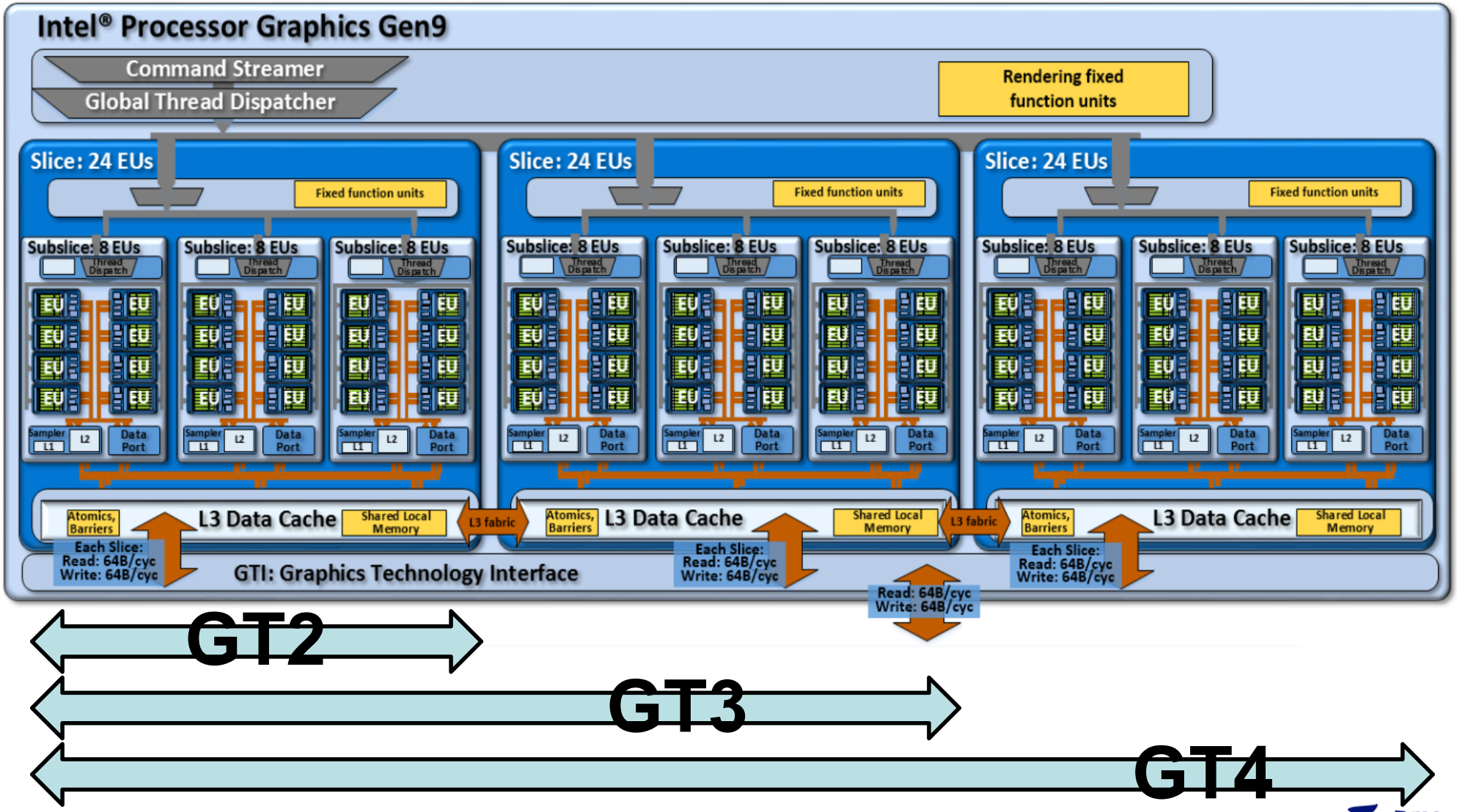
http://www.anandtech.com/show/6355/intels-haswell-architecture

Intel Skylake – Kaby Lake



https://software.intel.com/sites/default/files/managed/c5/9a/The-Compute-Architecture-of-Intel-Processor-Graphics-Gen9-v1d0.pdf

Intel Processor Graphics

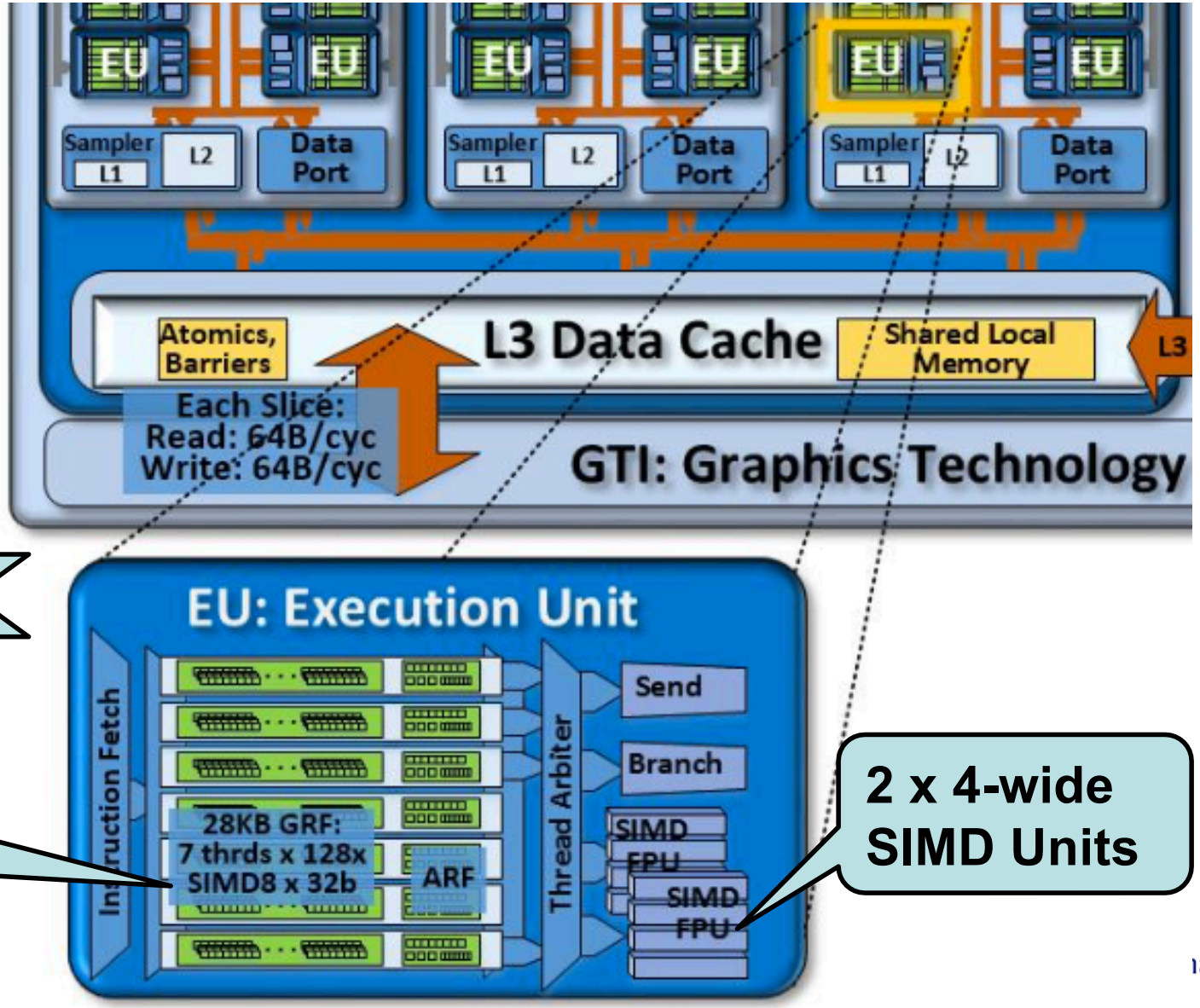


UNIVERSIDAD DE MÁLAGA

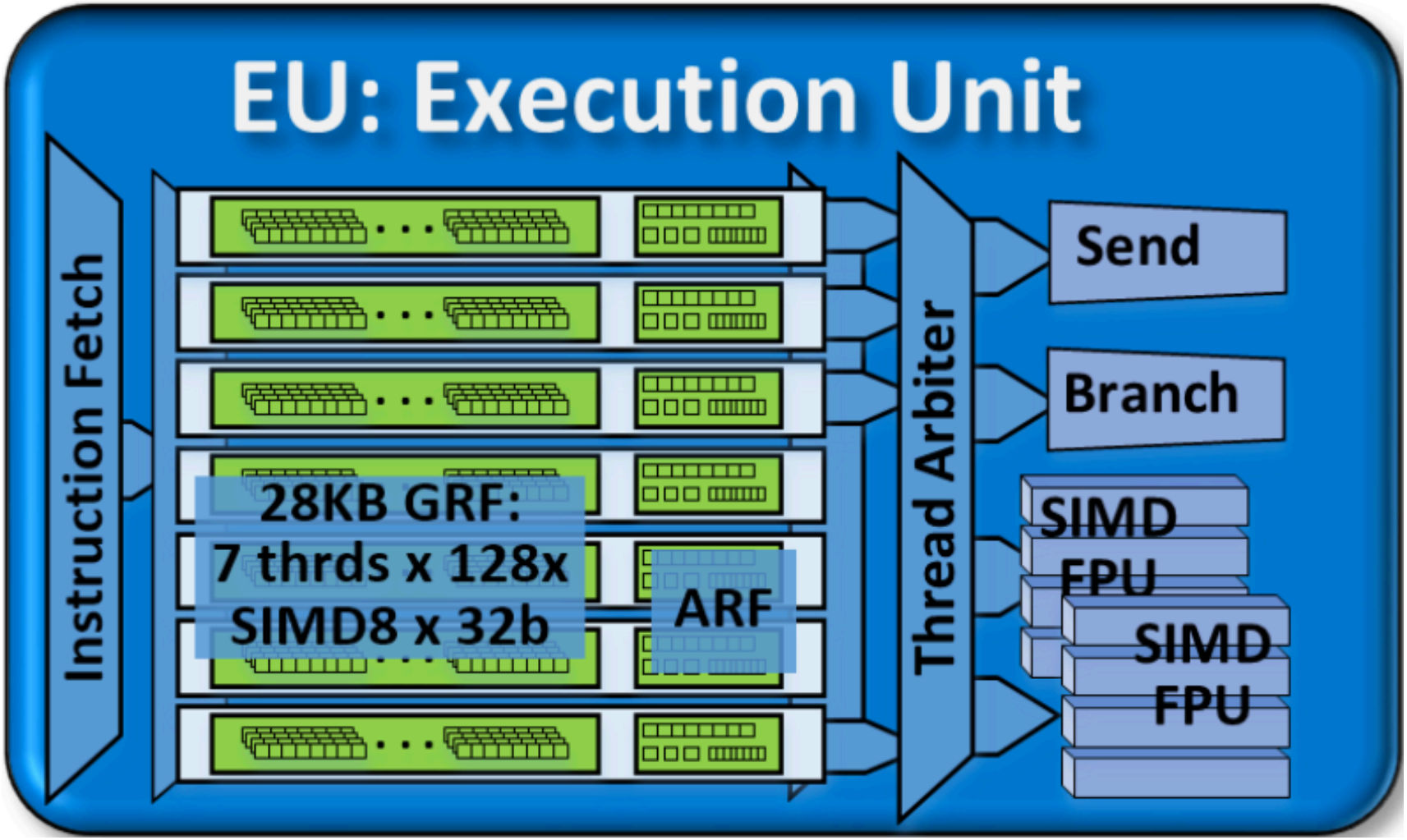
Intel Processor Graphics

- Peak performance:
  - 72 EUs
  - 2 x 4-wide SIMD
  - 2 flop/ck (fmadd)
  - 1GHz

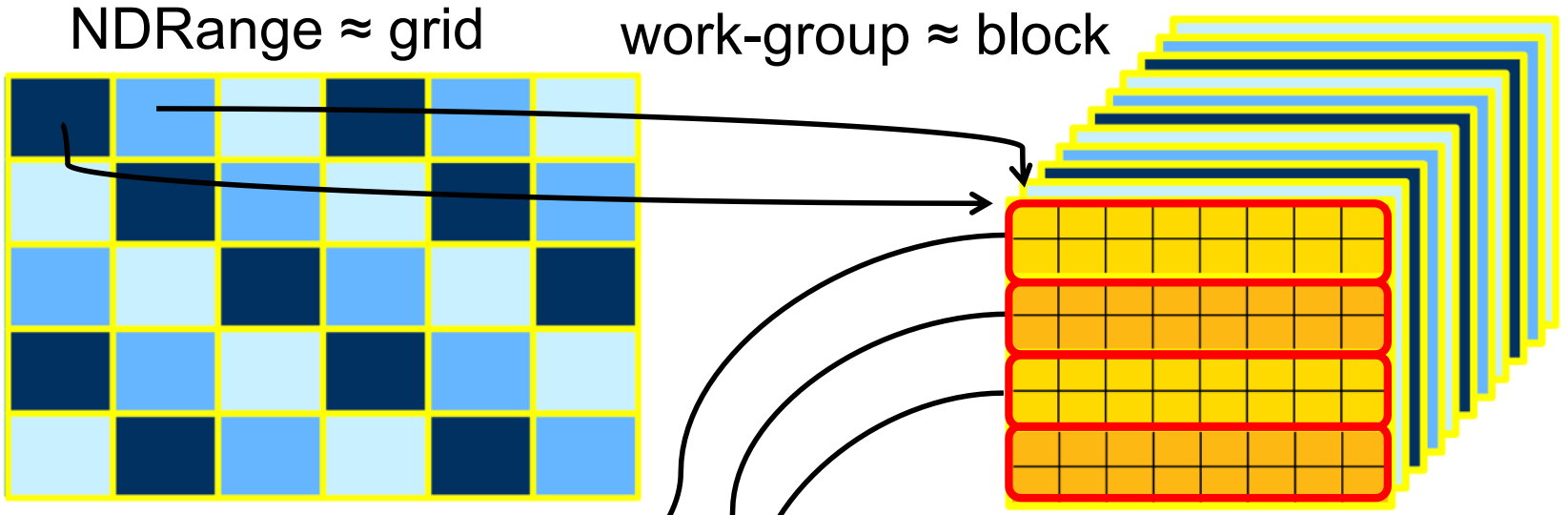
1.15 TFlops

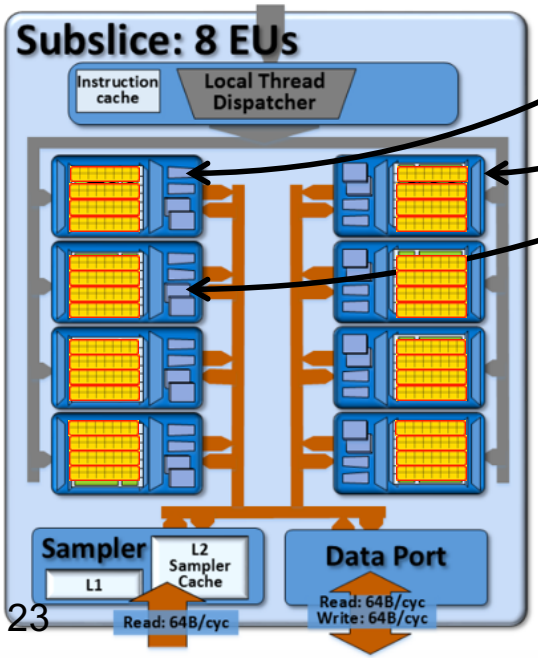


Intel Processor Graphics



Intel Processor Graphics

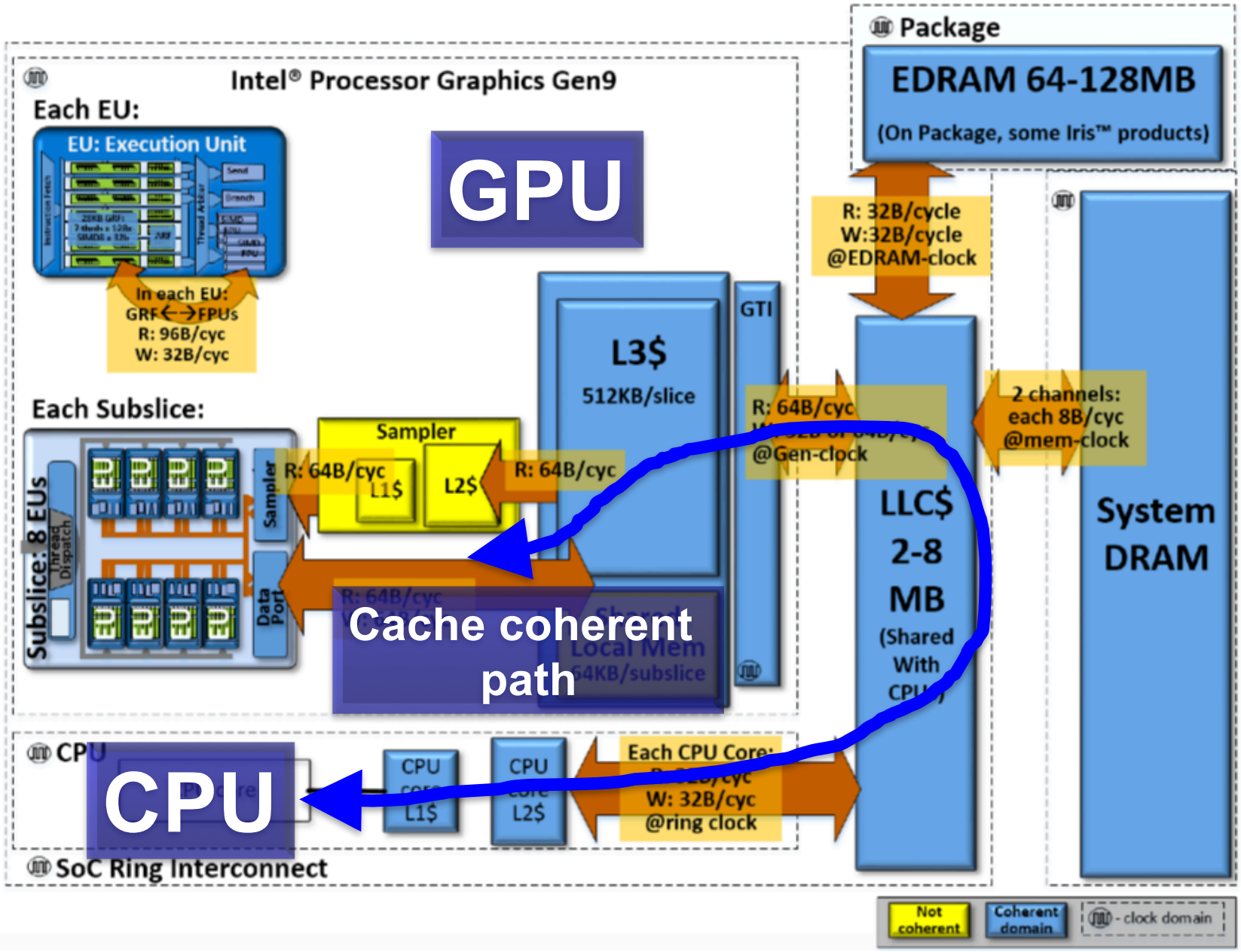




= EU-thread (SIMD16) ≈ wavefront ≈ warp: 16 work-items ≈ 16 thrds

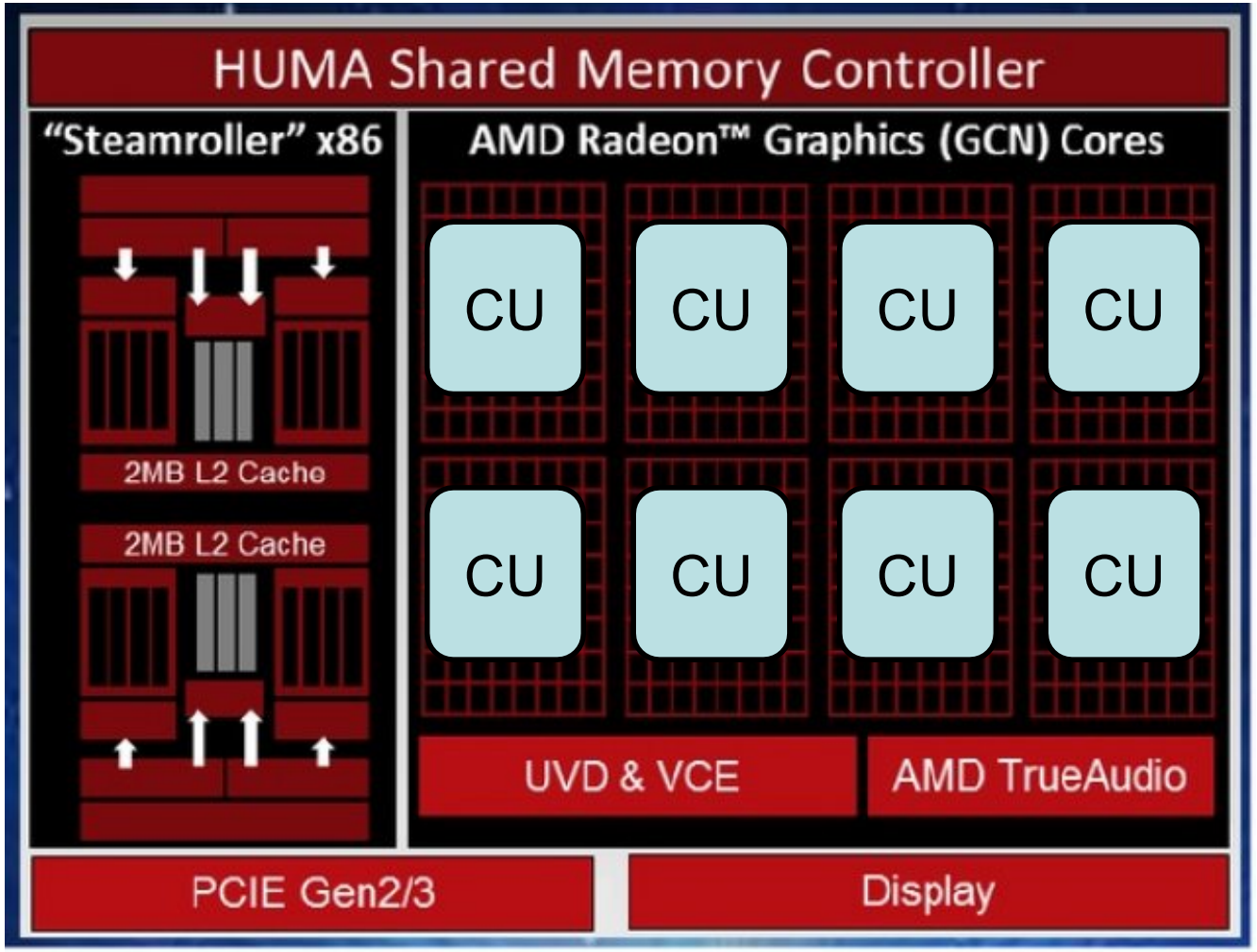
- Each SIMD16 wavefront takes 2cks
- IMT: fine-grained multi-threading
  - Latency hiding mechanism

Intel Virt. Technology for Directed I/O (Intel VT-d)



AMD Kaveri

- Steamroller microarch (2 – 4 “Cores”) + 8 GCN Cores.

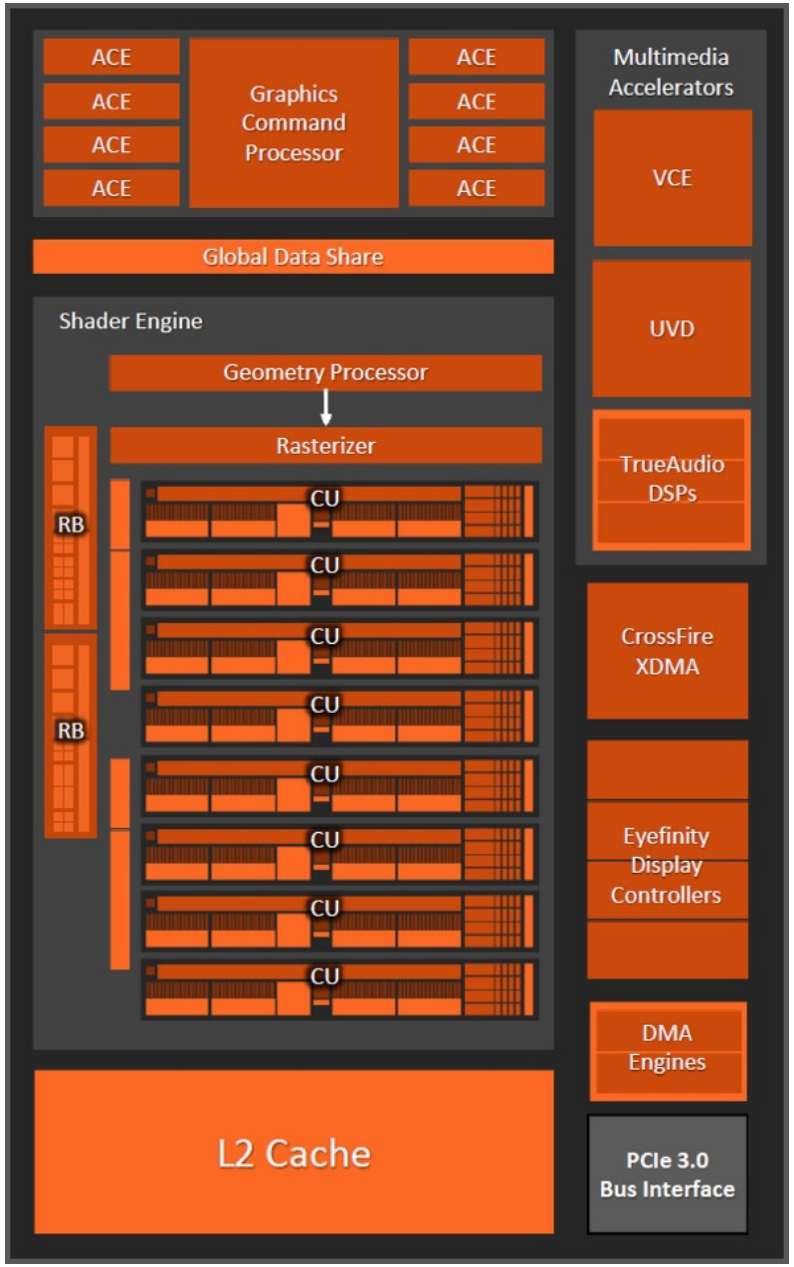


http://wccftech.com/

AMD Graphics Core Next (GCN)

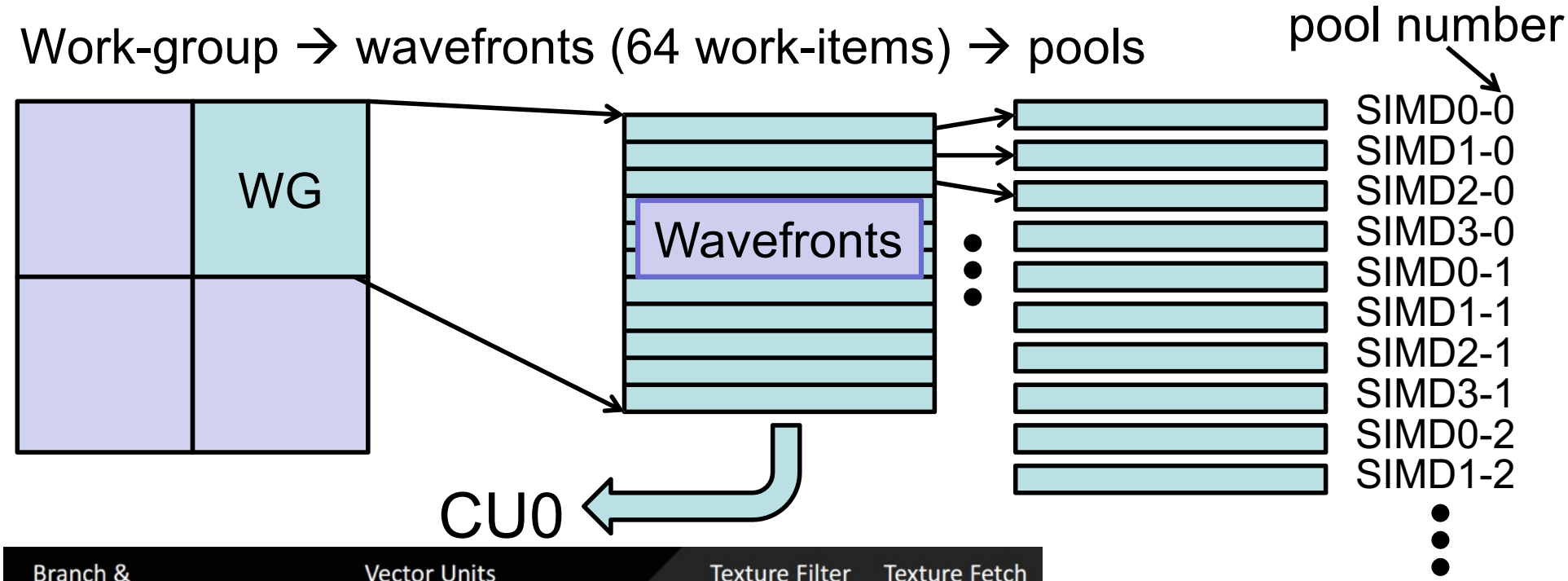
- In Kaveri,
  - 8 Compute Units (CU)
  - Each CU: **4 x 16-wide SIMD**
  - Total: 512 FPUs
  - 866 MHz

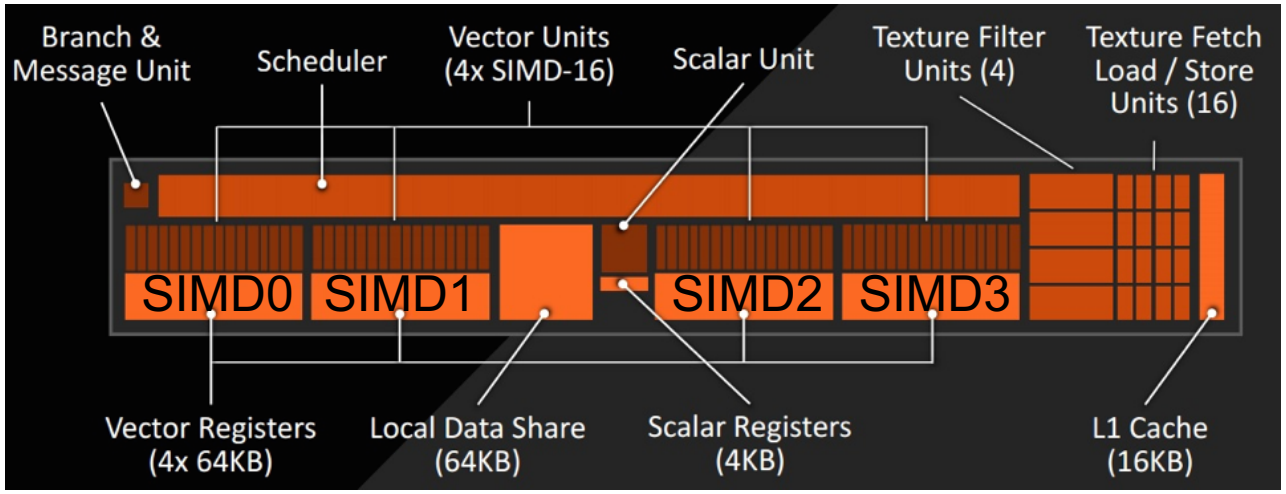
- Max GFLOPS=
  - 0.86 GHz x
  - 512 FPUs x
  - 2 fmad =
  - **880 GFLOPS**



OpenCL execution on GCN

Work-group → wavefronts (64 work-items) → pools

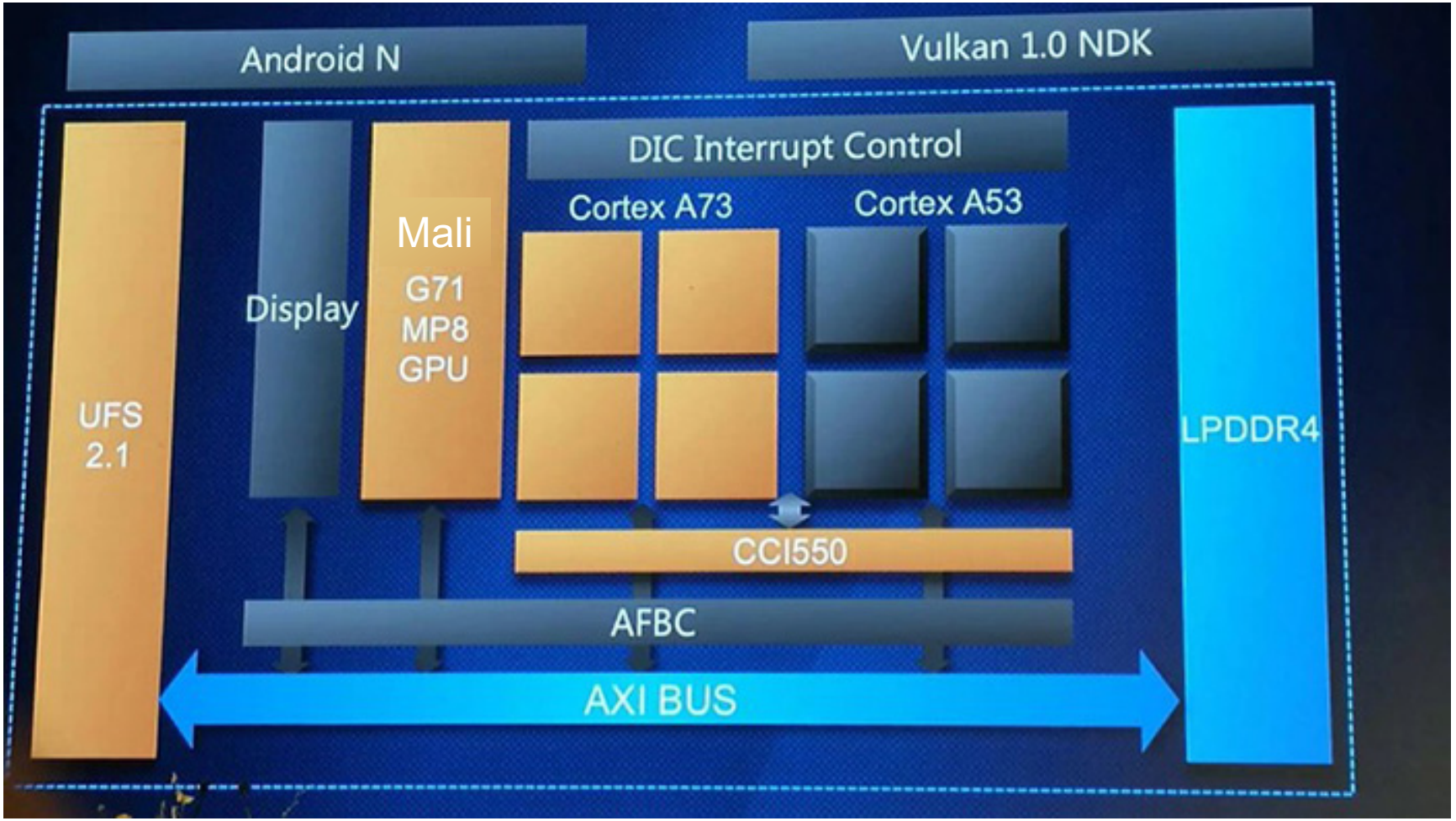




4 pools:
4 wavefronts in flight per SIMD

4 ck to execute each wavefront

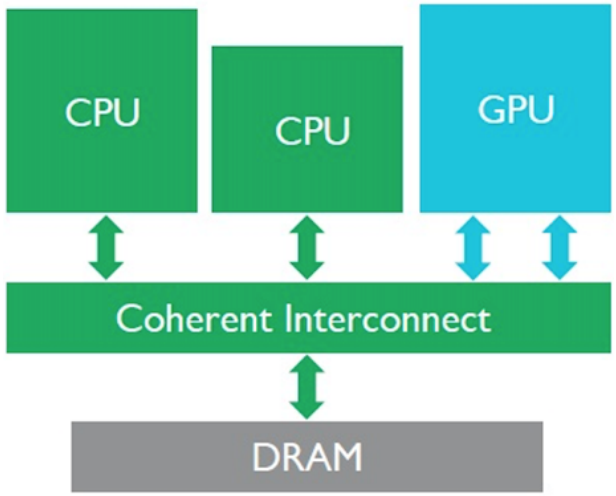
HiSilicon Kirin 960

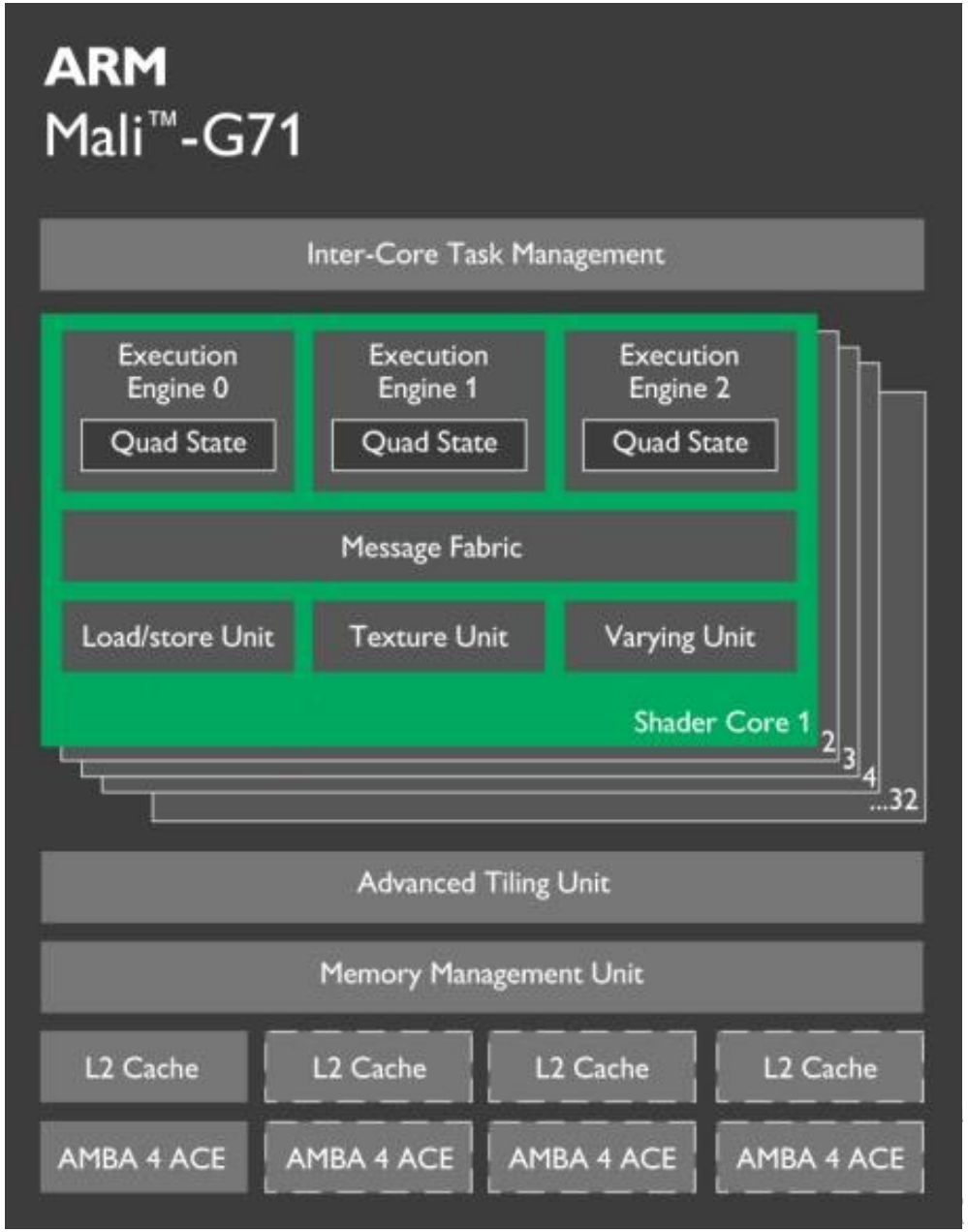


- ARM64 big.LITTLE (Cortex A73+Cortex A53) + Mali G71

Mali G71

- Supports OpenCL 2.0
- **4, 6, 8, 16, 32** Cores
- Each core:
  - 3 x 4-wide SIMD units
- 32 x 12 x 2 fmad x 0.85GHz=**652 GFLOPS**



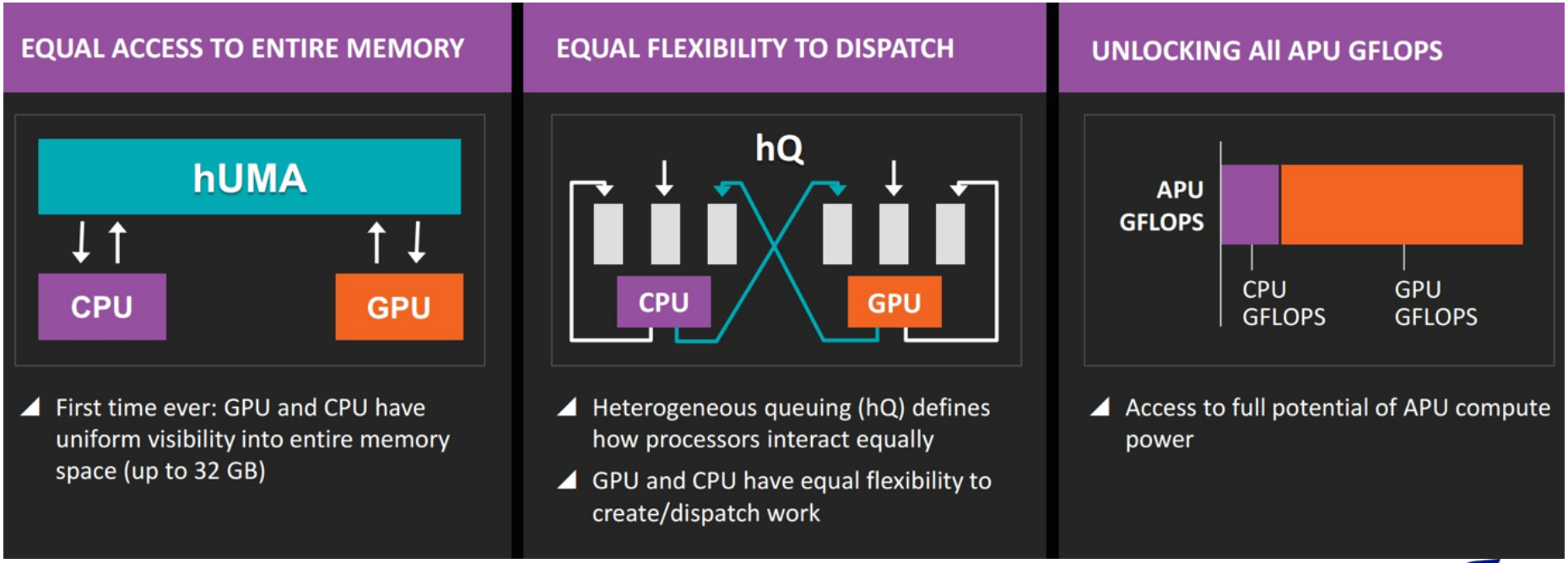


HSA (Heterogeneous System Architecture)

- HSA Foundation's goal: Productivity on heterogeneous HW
  - CPU, GPU, DSPs,...
- Founders:

AMD ARM Imagination MEDIATEK QUALCOMM SAMSUNG

HSA FOUNDATION



UNIVERSIDAD DE MÁLAGA

Advantages of iGPUs

- Discrete and integrated GPUs: different goals
  - NVIDIA Pascal: 3584 CUDA cores, 250W, 10 TFLOPS
  - Intel Iris Pro Graphics 580: 72EU x 8 SIMD, ~15W, 1.15 TFLOPS
  - Mali-G71: 32 EU x 3 x 4 SIMD, ~ 2W, 0.652 TFLOPS
- CPU and GPU are both **first-class citizens**.
  - **SVM** and **platform atomics** allows for closer collaboration
    - Avoid PCI data transfer and associated POWER dissipation
  - Operating System doesn't get in the way → less overhead
- CPU and GPU may have similar performance
  - It's more likely that they can collaborate
- Cheaper!

Part I

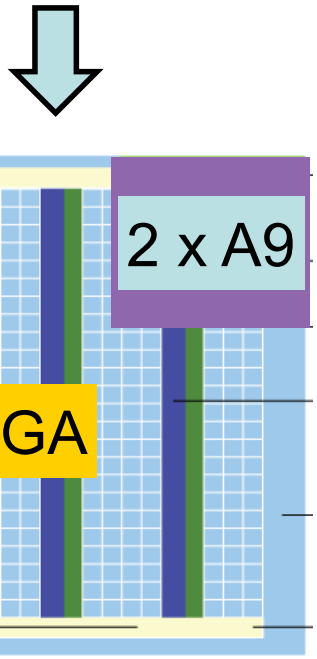
- Motivation

- **Hardware**
  - Integrated GPUs (iGPUs)
  - Integrated FPGAs (**iFPGAs**)

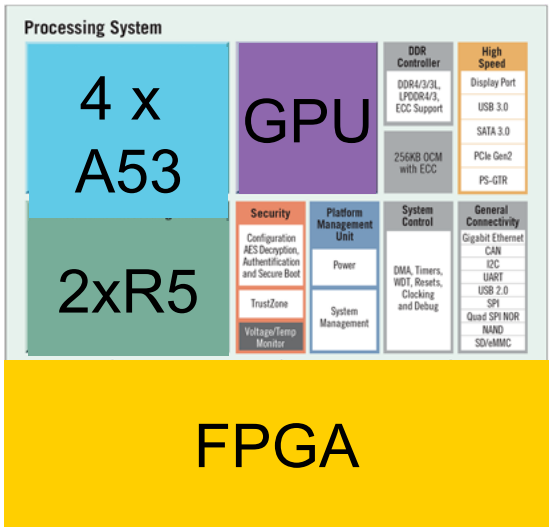
- Software
  - Programming models for heterogeneous chips with GPUs
  - Programming models for heterogeneous chips with FPGAs

Hardware: iFPGAs

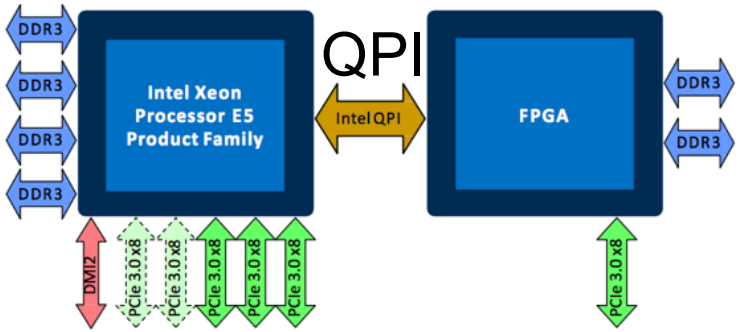
Altera Cyclone V



Xilinx Zynq UltraScale+

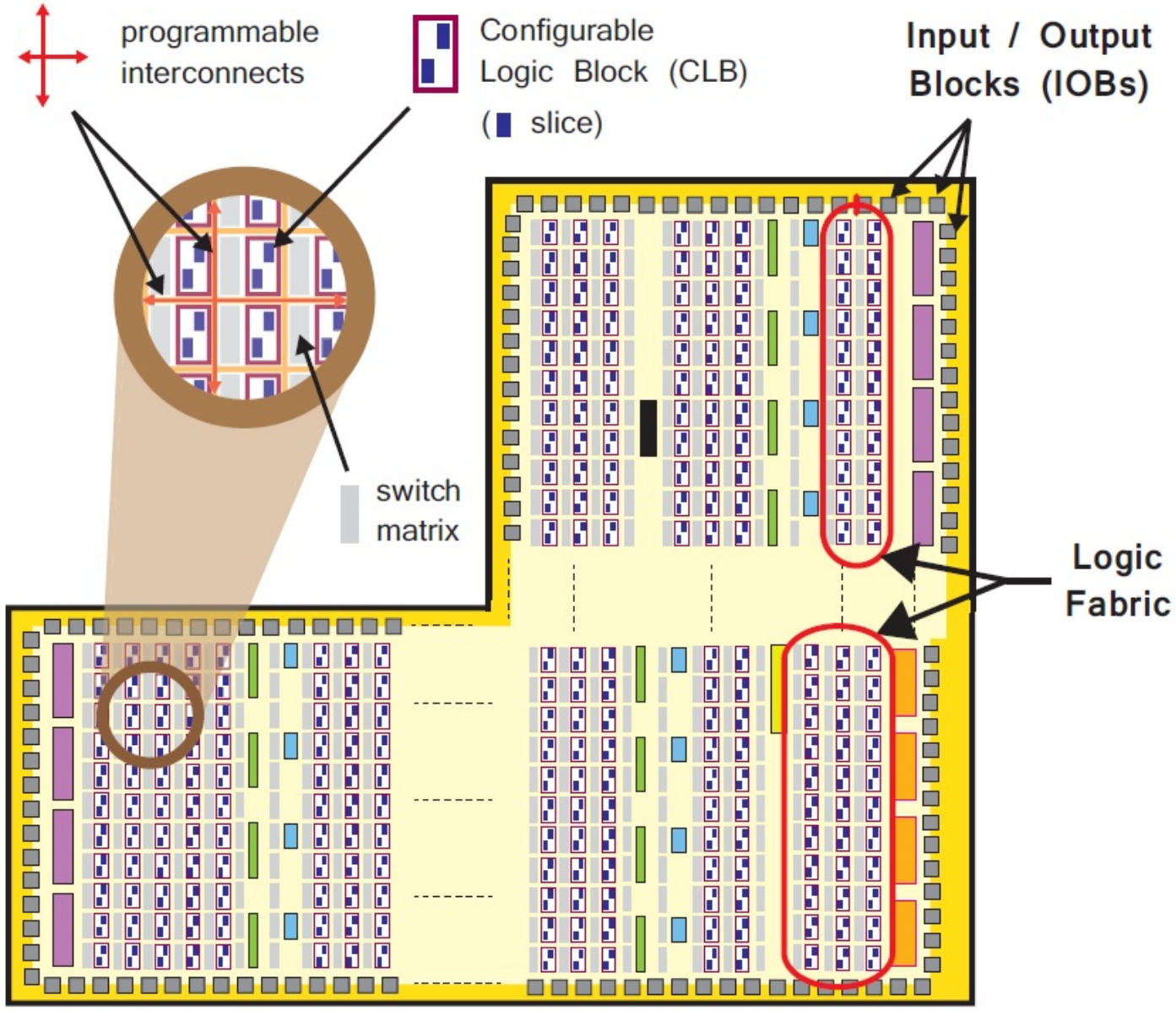


Intel Xeon+Arria10



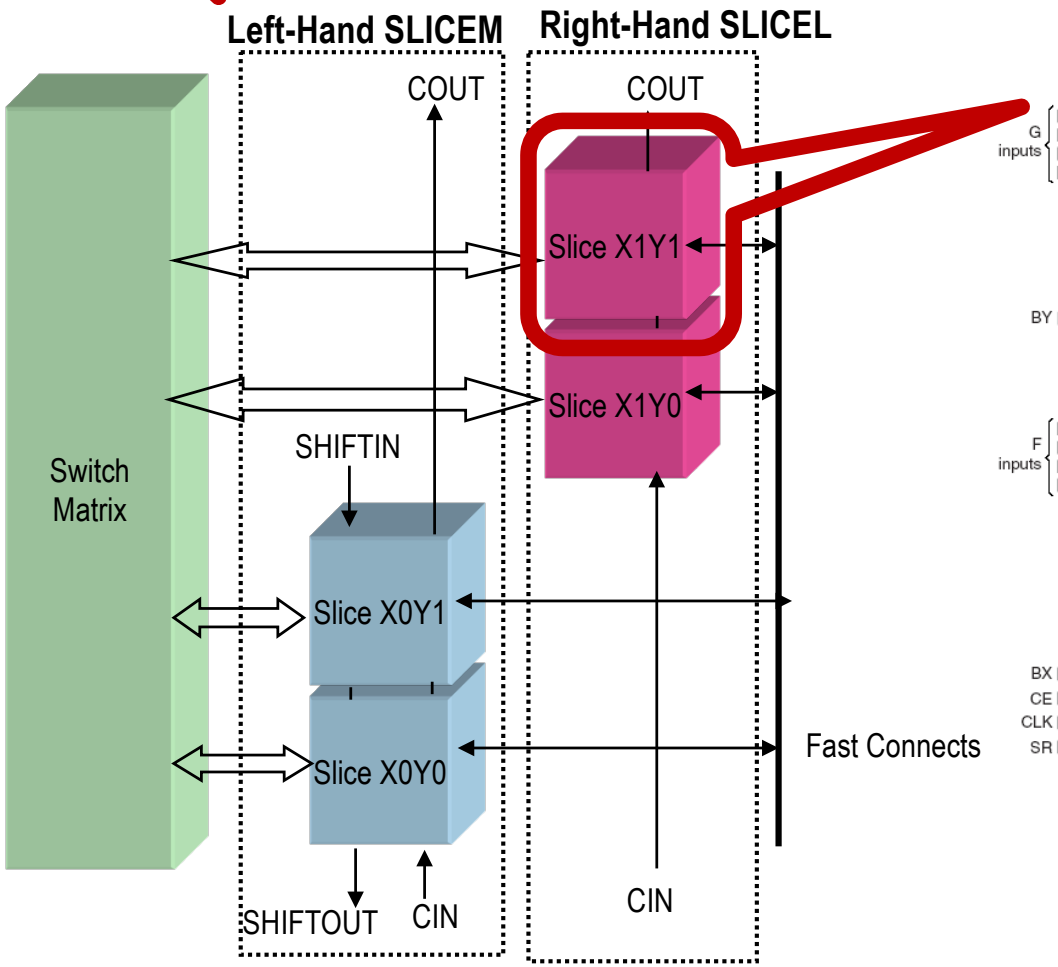
UNIVERSIDAD DE MÁLAGA

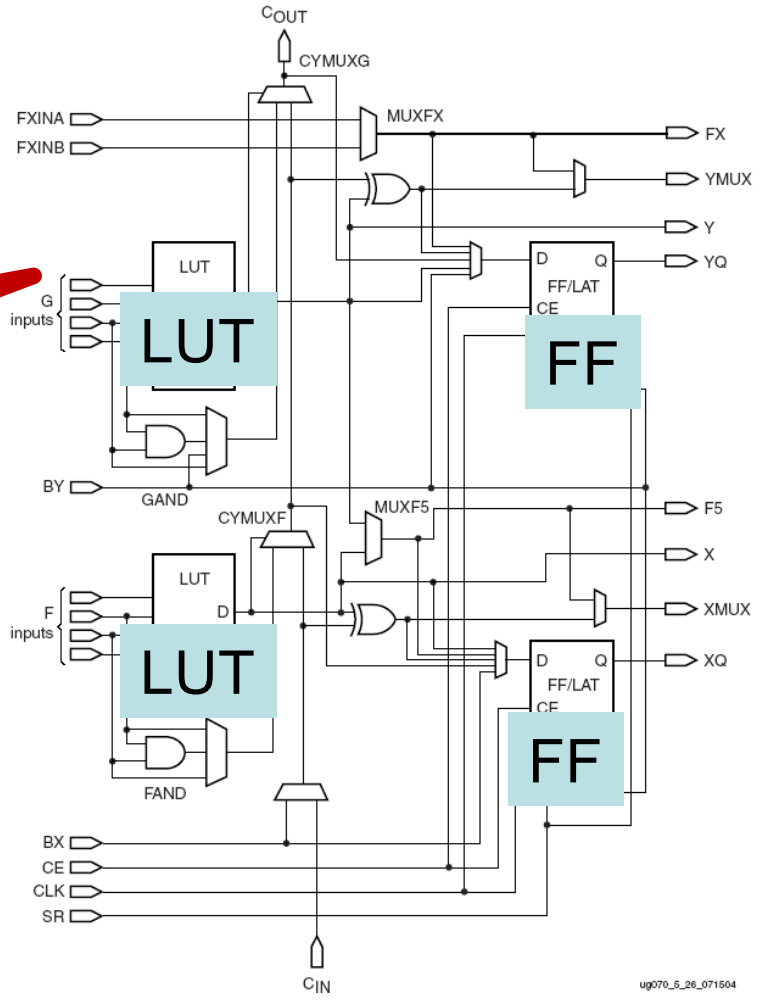
FPGA Architecture



Courtesy: Javier Navaridas, U. Manchester.

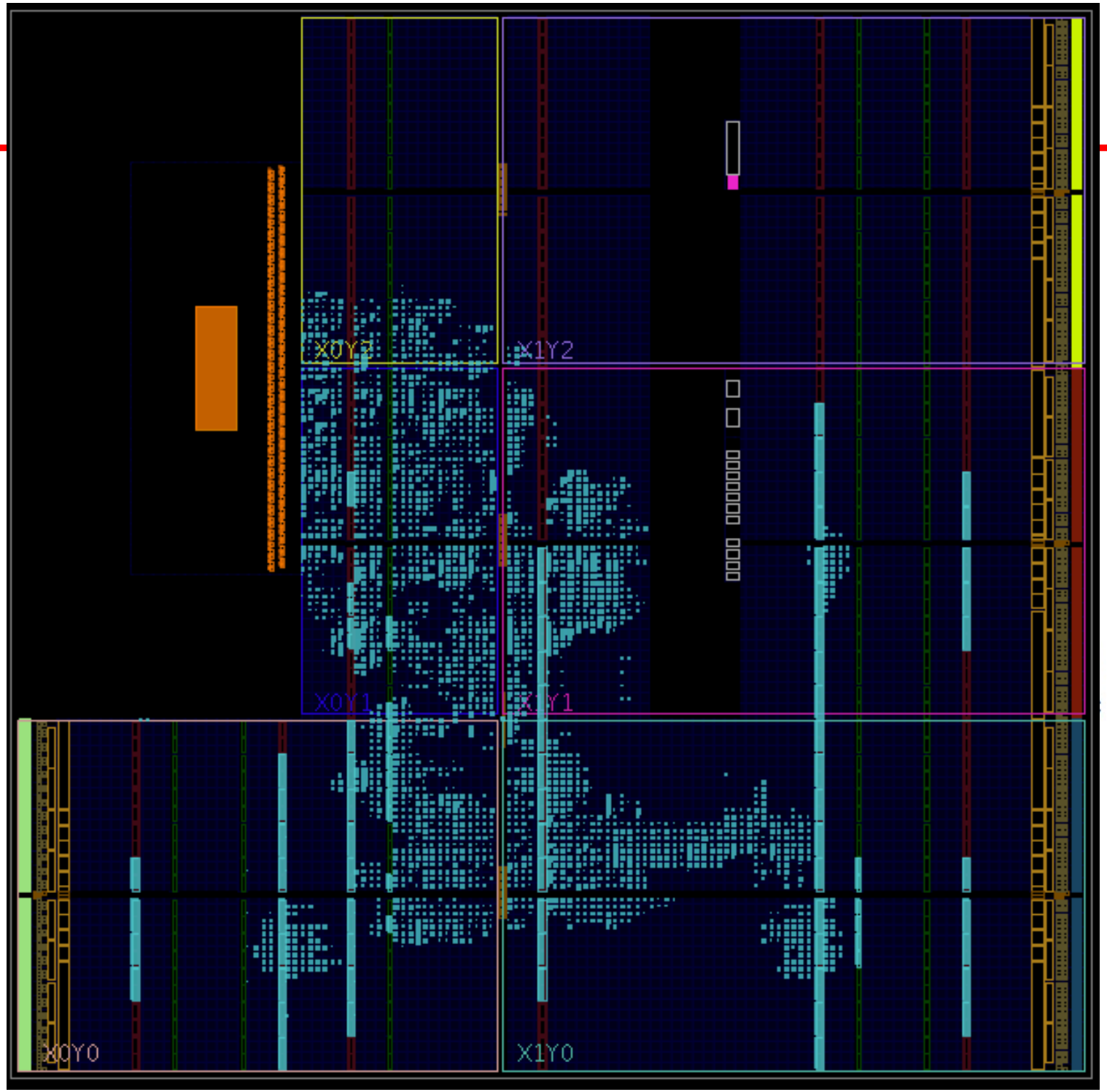
CLB: Configurable Logic Block





Courtesy: Jose Luis Nunez-Yanez, U. Bristol.

UNIVERSIDAD DE MÁLAGA

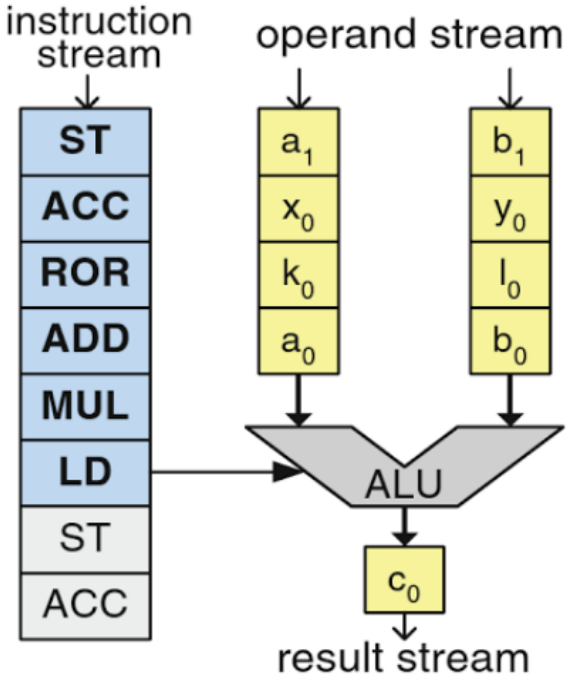


UNIVERSIDAD DE MÁLAGA

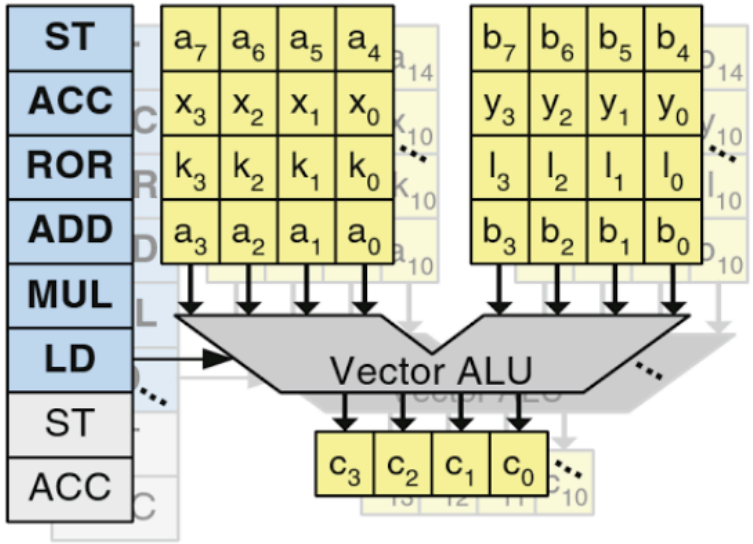
How do we “mold this liquid silicon”?

- Hardware overlays
  - Map a CPU or GPU onto the FPGA
    - ✓ 100s of simple CPU can fit on large FPGAs
    - ✓ The CPU or GPU overlay can change adapt
    - 👎 Resulting overlay is not as efficient as a “real” one
    - 👎 Same drawbacks as “general purpose” processors

CPU:



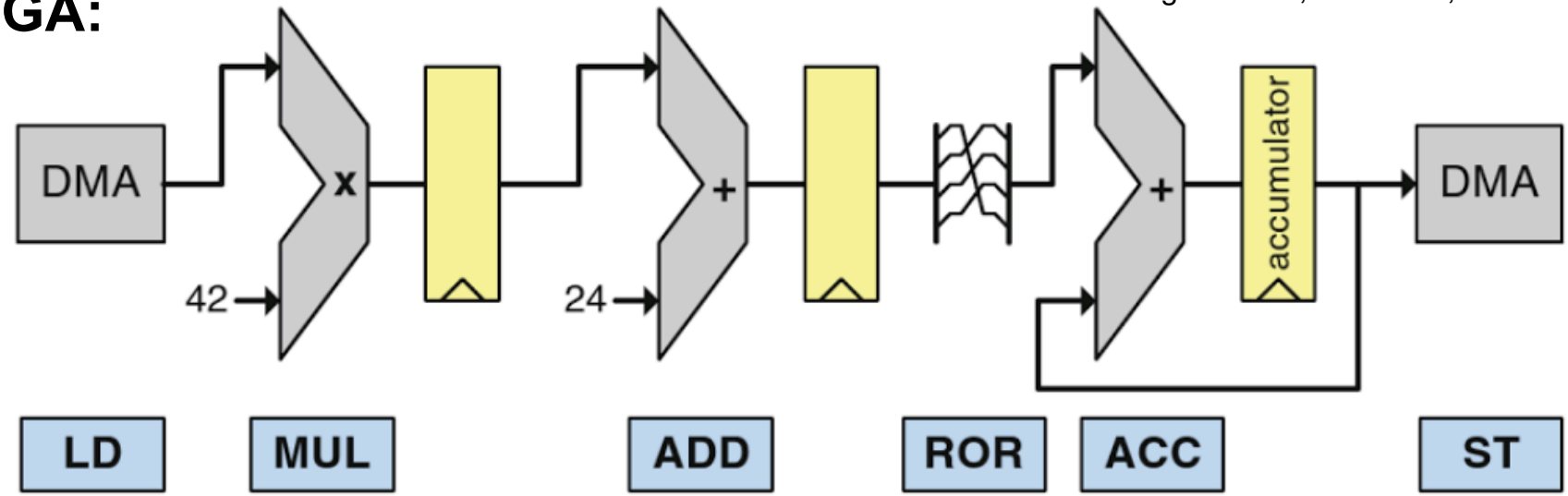
GPU:



How do we “mold this liquid silicon”?

FPGA:

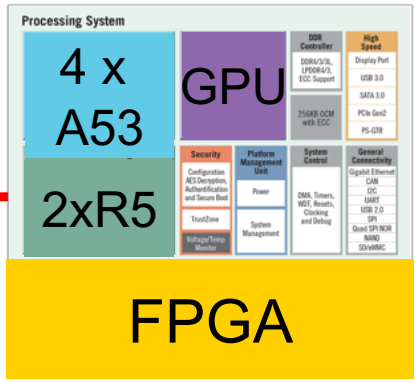
FPGAs for Software Programmers, Dirk Kock, Daniel Ziener



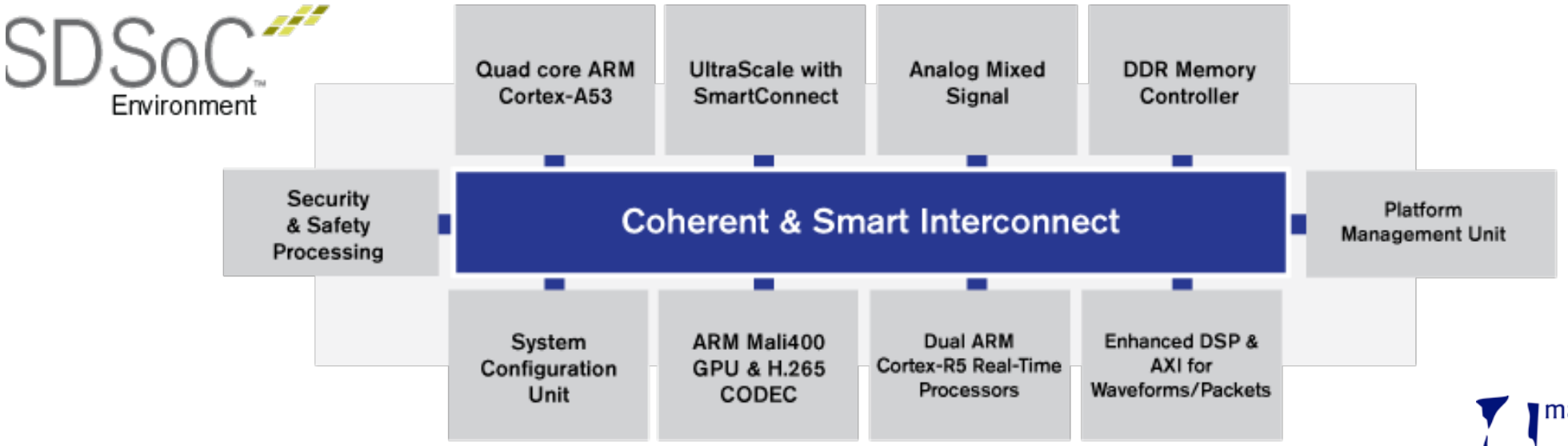
- ✓ No FETCH nor DECODE of instructions → already hardwired
- ✓ Data movement (Exec. Units ⇔ MEM) reduction → Power save
- ✓ Not constrained by a fixed ISA:
  - ✓ Bit-level parallelism; shift, bit mask, ...; variable precision arith.
- 👎 Less frequency (hundreds of MHz)
- 👎 In order execution (proposal from David Kaeli’s group to OoO)
- 👎 Programming effort

Hardware thread reordering to boost OpenCL throughput on FPGAs, ICCD’16

Xilinx Zynq UltraScale+ MPSoC



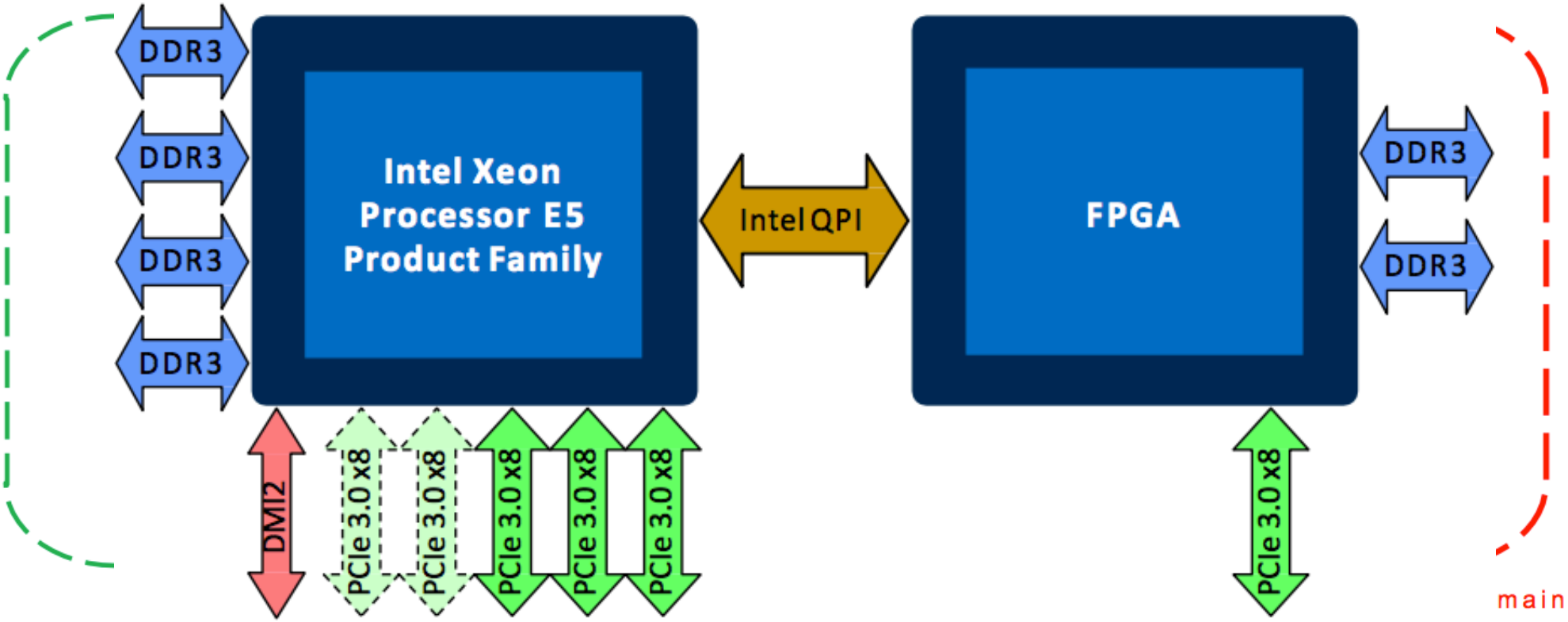
- Available:
  - SW coherence (cache flush) → Slow
  - HW coherence (MESI coherence protocol) → Fast. Ports:
    - Accelerator Coherency Port, ACP
    - Cache-coherent interconnect (CCI) ACE-Lite ports
      - Part of ARM® AMBA® AXI and ACE protocol specification



UNIVERSIDAD DE MÁLAGA

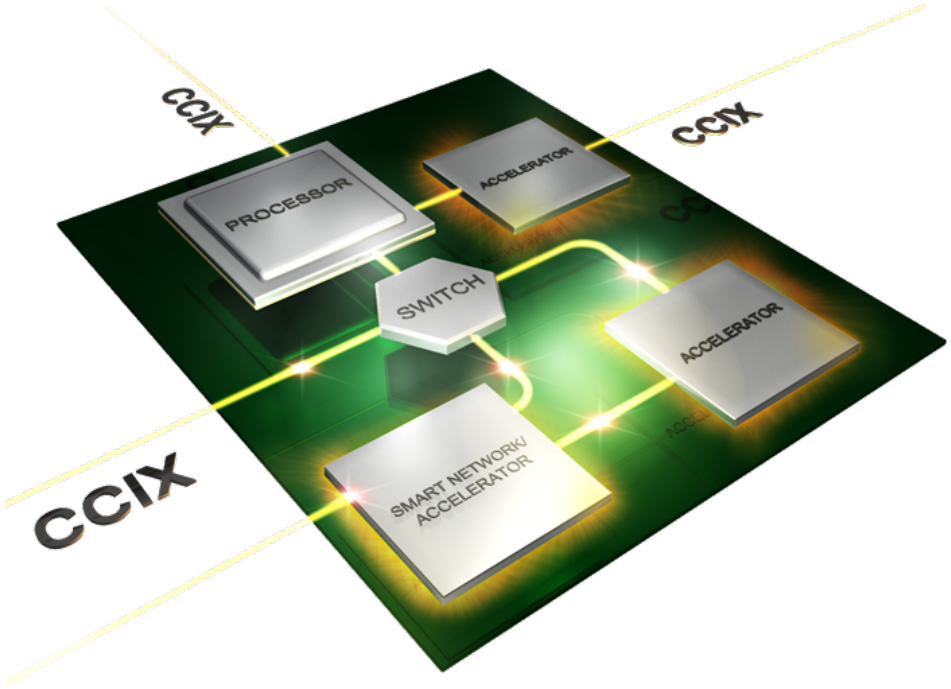
Intel Broadwell+Arria10

- Cache Coherent Interconnect (CCI)
  - New IP inside the FPGA: Intel Quick Path Interconnect with CCI
  - AFU (Accelerated Function Unit) can access cached data



Towards CCI everywhere

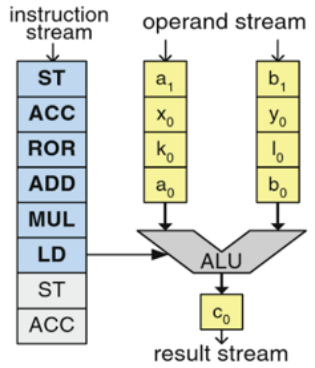
- CCIX consortium (http://www.ccixconsortium.com)
  - New chip-to-chip interconnect operating at 25Gbps
  - Scalable to higher speeds.
  - Protocol built for FPGAs, GPUs, network/storage adapters, ...
  - Already in Xilinx Virtex UltraScale+

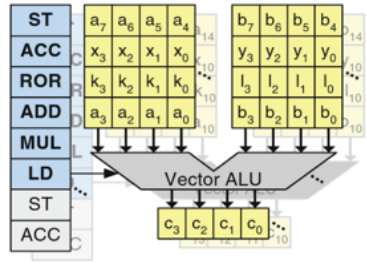


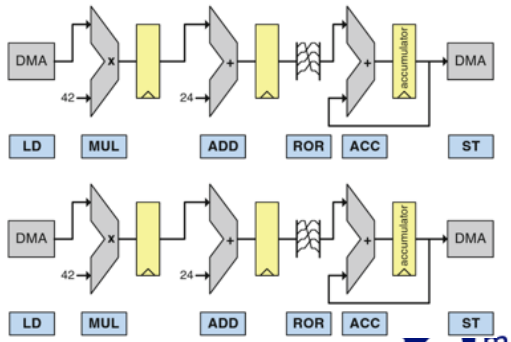
AMD · Amphenol · ARM · Arteris · avery design systems · BROADCOM · Bull atos technologies · cadence · CAVIUM · HUAWEI · IBM · IDT · KEYSIGHT TECHNOLOGIES · Mellanox TECHNOLOGIES · Micron · NETSPEED SYSTEMS · QUALCOMM · redhat · SYNOPSYS · tsmc · TELEDYNE LECROY Everywhereyoulook · XILINX ALL PROGRAMMABLE

Which architecture suits me best?

- All of them are Turing complete, but
- Depending on the problem:
  - CPU:
    - For control-dominant problems
    - Frequent context switching
  - GPU:
    - Data-parallel problems (vectorized/SIMD)
    - Little control flow and little synchronization
  - FPGA:
    - Stream processing problems
    - Large data sets processed in a pipelined fashion
    - Low power constraints







UNIVERSIDAD DE MÁLAGA

Part I

- Motivation

- Hardware
  - Integrated GPUs (iGPUs)
  - Integrated FPGAs (iFPGAs)

- **Software**
  - Programming models for heterogeneous chips with **GPUs**
  - Programming models for heterogeneous chips with FPGAs

http://imgs.xkcd.com/comics/standards.png

HOW STANDARDS PROLIFERATE

see: A/C chargers, Character encodings, Instant Messaging, etc)

Programming models for heterogeneous: GPUs

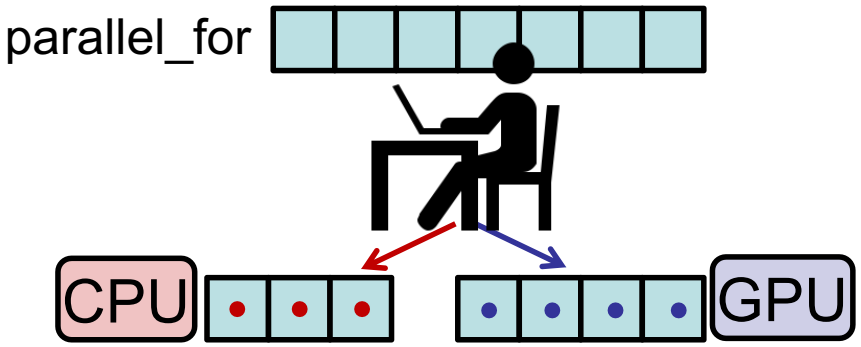
- Targeted at exploiting **one device** at a time
  - CUDA (NVIDIA)
  - **OpenCL** (Khronos Group Standard)
  - OpenACC (C, C++ or Fortran + Directives → OpenMP 4.0)
  - C++AMP (Windows' extension of C++. Recently HSA announced own ver.)
  - Sycl (Khronos Gruop's specification for "Single-source C++ progr.)
  - ROCm (Radeon Open Compute, HCC, HIP, for AMD GCN –HSA–)
  - RenderScript (Google's Java API for Android)
  - ParallDroid (Java + Directives from ULL, Spain)
  - Many more (Numba Python, IBM Java, Matlab, R, JavaScript, …)

- Targeted at exploiting **both devices** simultaneously (**discrete** GPUs)
  - Qilin (C++ and Qilin API compiled to TBB+CUDA)
  - OmpSs (OpenMP-like directives + Nanos++ runtime + Mercurium compiler)
  - XKaapi
  - StarPU

- Targeted at exploiting **both devices** simultaneously (**integrated** GPUs)
  - Qualcomm Symphony
  - Intel Concord

Programming models for heterogeneous: GPUs

Targeted at exploiting **ONE** device at a time



CUDA

OpenCL

Sycl

C++AMP

OpenACC

Matlab

ROCm

R

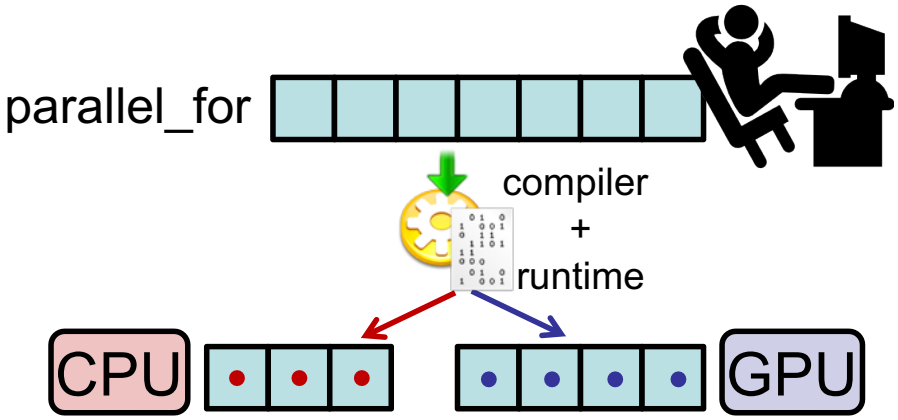
JavaScript

Numba Python

RenderScript

ParallDroid

Targeted at exploiting **SEVERAL** devices at the same time



Qilin

OmpSs

XKaapi

StarPU

Qualcomm Symphony

Concord

Programming models for heterogeneous: FPGAs

- HDL-like languages
  - SystemVerilog, Bluespec System Verilog
  - Extend RTL languages by adding new features
- Parallel libraries
  - CUDA, **OpenCL**
  - Instrument existing parallel libraries to generate RTL code
- C-based languages
  - **SDSoC**, LegUp, ROCCC, Impulse C, Vivado, Calypto Catapult C
  - Compile* C into intermediate representation and from there to HDL
  - Functions become Finite State Machines and variables mem. blocks
- Higher level languages
  - Kiwi (C#), Lime (Java), Chisel (Scala)
  - Translate* input language into HDL

* Normally a subset thereof

Courtesy: Javier Navaridas, U. Manchester.

OpenCL 2.0

- OpenCL 2.0 supports:
  - Coherent **SVM** (Shared Virtual Memory) & **Platform atomics**
  - Example: allocating an array of atomics:

CPU Host Code

```
clContex = ...
clCommandQueue = ...
clKernel = ...

cl_svm_mem_flags flags = CL_MEM_SVM_FINE_GRAIN_BUFFER | CL_MEM_SVM_ATOMICS;
atomic_int * data;
data = (atomic_int *) clSVMAlloc(clContext, flags, NUM * sizeof(atomic_int), 0);

clSetKernelArgSVMPointer(clKernel, 0, data);
clSetKernelArg(clKernel, 1, sizeof(int), &NUM);
clEnqueueNDRangeKernel(clCommandQueue, clKernel, .... );

atomic_store_explicit ((atomic_int*)&data[0], 99, memory_order_release);
```

GPU Kernel Code

```
kernel void myKernel(global int *data, global int* NUM){

int i=atomic_load_explicit((global atomic_int *)&data[0], memory_order_acquire);
....
}
```

C++AMP

- C++ Accelerated Massive Parallelism
- Example: Vector Add (`C[i]=A[i]+B[i]`)

```
#include <amp.h>

int main(int argc, char *argv[])
{
    // Allocate auto-managed host/device views of data:
    concurrency::array_view<float> A(sizeElements);
    concurrency::array_view<float> B(sizeElements);
    concurrency::array_view<float> C(sizeElements);

    // Initialize host data
    for (int i=0; i<sizeElements; i++) {
        A[i] = 1.618f * i;
        B[i] = 3.142f * i;
    }
    C.discard_data(); // tell runtime not to copy CPU host data.

    // Launch kernel onto default accelerator
    // The HCC runtime will ensure that A and B are available on the accelerator
    concurrency::parallel_for_each(concurrency::extent<1> (sizeElements),
      [=] (concurrency::index<1> idx) restrict(amp) {
        int i = idx[0];
        C[i] = A[i] + B[i];
    });
}
```

A, B and C buffers

Initialize A and B

Run on GPU

SYCL's flavour: C[i]=A[i]+B[i]

```
#include <sycl.hpp>
using namespace cl::sycl
int main() {
  std::vector h_a(LENGTH);                // a vector
  std::vector h_b(LENGTH);                // b vector
  std::vector h_c(LENGTH);                // c vector
  // Fill vectors a and b with random float values
  ....
  { // Device buffers
    buffer d_a(h_a); buffer d_b(h_b); buffer d_c(h_c);
    queue myQueue;
    command_group(myQueue, [&]()
    {
      // Data accessors
     auto a = d_a.get_access<access::read>();
     auto b = d_b.get_access<access::read>();
     auto c = d_c.get_access<access::write>();
      // Kernel
      parallel_for(count, kernel_functor([ = ](id<> item) {
        int i = item.get_global(0);
        c[i] = a[i] + b[i];
      }));
    });}
}
```

codeplay®

ComputeCpp™

Automatically targets the first OpenCL-enabled device

Command group to define an asynchronous task

SYCL Parallel STL

- Parallel STL aimed at C++17 standard:
  - std::sort(vec.begin(), vec.end()); //Vector sort
  - std::sort(**seq**, vec.begin(), vec.end()); // Explicit SEQUENTIAL
  - std::sort(**par**, vec.begin(), vec.end()); // Explicit PARALLEL
- SYCL exposes the execution policy (user-defined)

```
std::vector<int>(N);
float ratio = 0.5;
sycl::het_sycl_policy<class ForEach> policy(ratio);
std::for_each(policy, v.begin(), v.end(),
       [=](int& val) {
        val= val*val;
        }
 );
```

50% of the iterations on **CPU**
50% of the iterations on **GPU**

https://github.com/KhronosGroup/SyclParallelSTL

 Courtesy: Antonio Vilches

Qualcomm Symphony

- **Point Kernel**: write once, run everywhere
  - Pattern tuning extends to heterogeneous load hints

```
SYMPHONY_POINT_KERNEL_3(vadd, int, i, float*, a, int, na,
                        float*, b, int, nb, float*, c, int, nc,
{ c[i] = a[i] + b[i];}); …

symphony::buffer_ptr buf_a(1024), buf_b(1024), buf_c(1024);
...
symphony::range<1>(1024) r;
symphony::pfor_each(r, vadd_pk, buf_a, buf_b, buf_c,
            symphony::pattern::tuner().set_cpu_load(10)
                                      .set_gpu_load(50)
                                      .set_dsp_load(40)
);
```

A kernel for each device is automatically generated

Executed across CPU(10%) + GPU(50%) + DSP(40%)

Courtesy: Calin Cascaval. Qualcomm

Intel Concord

- C++ heterogeneous programming framework

```
class Foo {
   int *a, *b, *c;
   void operator(int i) { c[i] = a[i]+b[i]; }

Foo *f = new Foo();

parallel_for(0, N, *f);
```

Executed across CPU+GPU, dynamic partition

- Papers:
  - Rashid Kaleem et al. Adaptive heterogeneous scheduling on integrated GPUs. **PACT 2014**.
    - Intel Heterogeneous Research Compiler (iHRC) at https://github.com/IntelLabs/iHRC/
  - Naila Farooqui et al. Affinity-aware work-stealing for integrated CPU-GPU processors. **PPoPP '16**. PhD dissertation: https://smartech.gatech.edu/bitstream/handle/1853/54915/FAROOQUI-DISSERTATION-2016.pdf

Part I

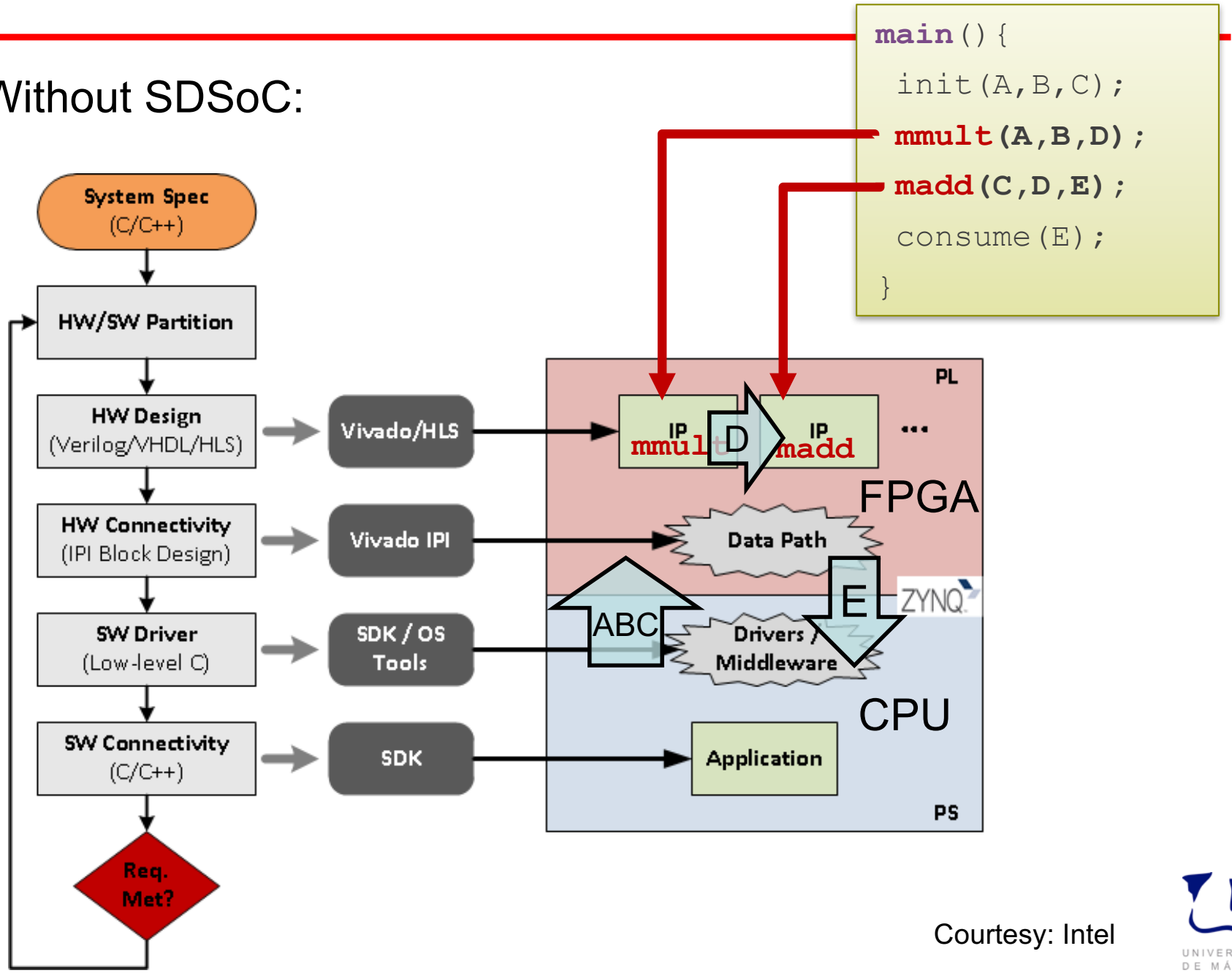
- Motivation

- Hardware
  - Integrated GPUs (iGPUs)
  - Integrated FPGAs (iFPGAs)

- **Software**
  - Programming models for heterogeneous chips with GPUs
  - Programming models for heterogeneous chips with **FPGAs**

Xilinx SDSoC

- Without SDSoC:



Courtesy: Intel

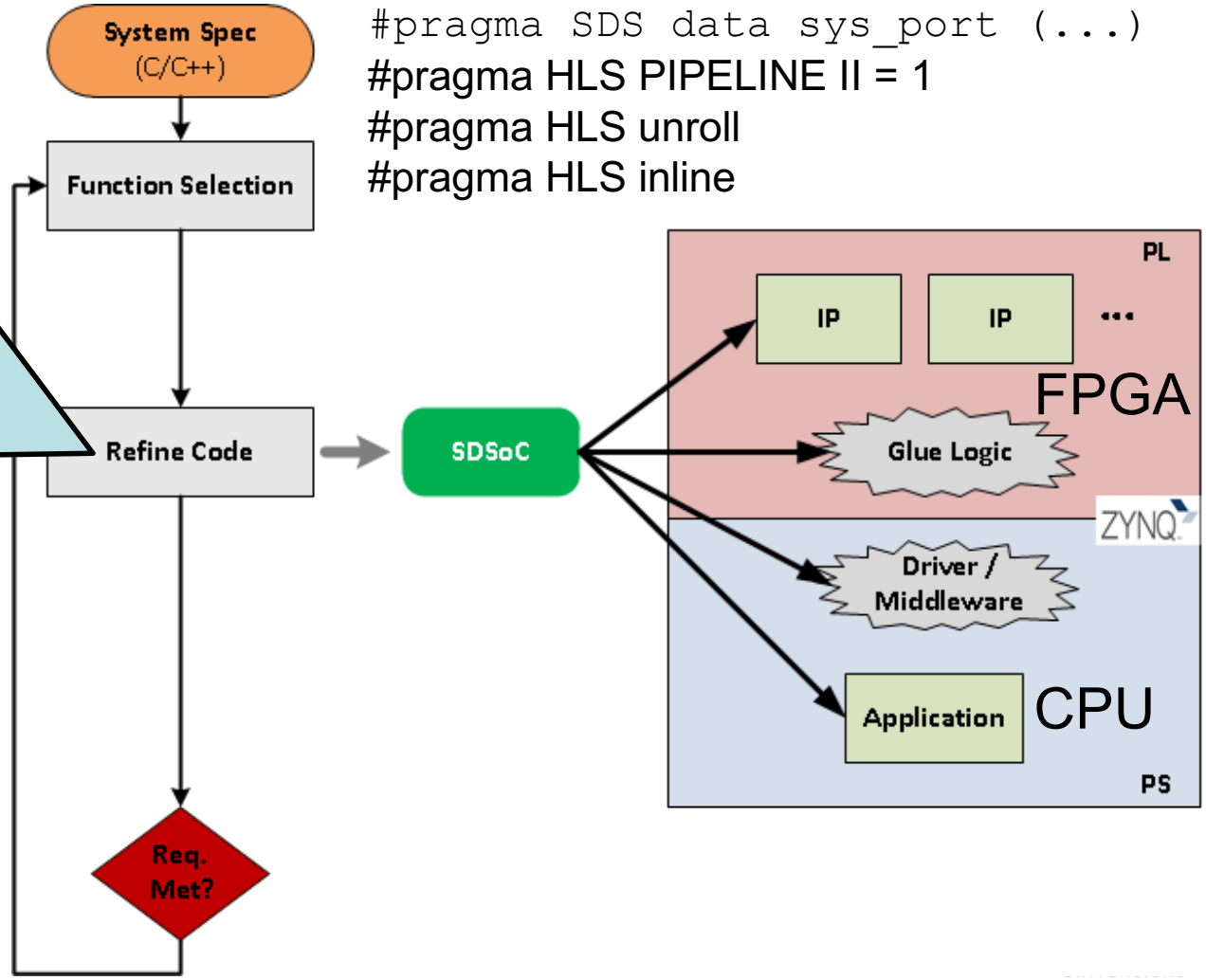
Xilinx SDSoC

- With SDSoC:
  - Eclipse-based SDK

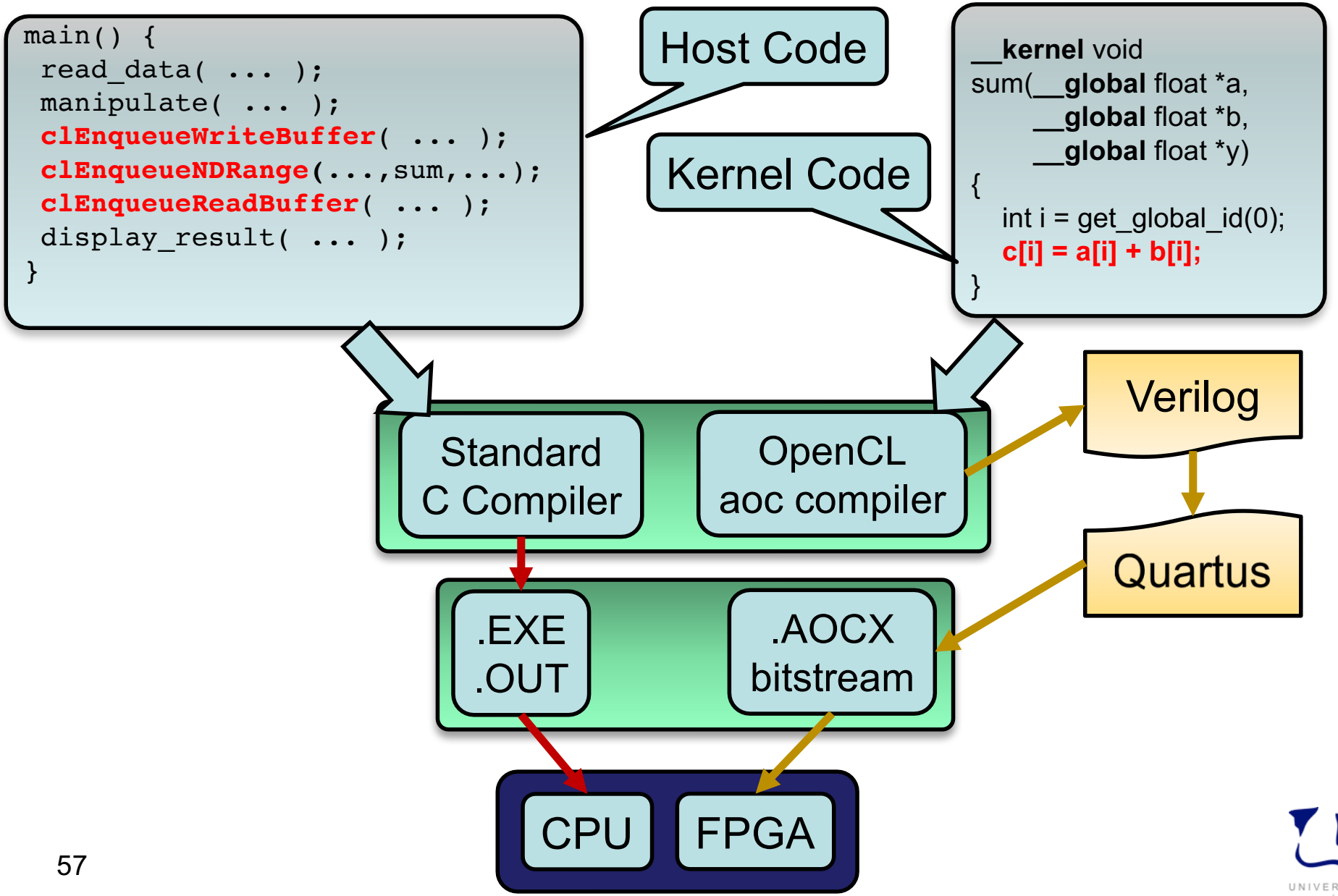
```
#pragma SDS data copy(Array)
#pragma SDS zero_copy(Array)
#pragma SDS data access_pattern
#pragma SDS data sys_port (...)
#pragma HLS PIPELINE II = 1
#pragma HLS unroll
#pragma HLS inline
```

Refine Code:

- Re-factoring code to be more HLS-friendly
- Adding code annotations (i.e., pragmas)

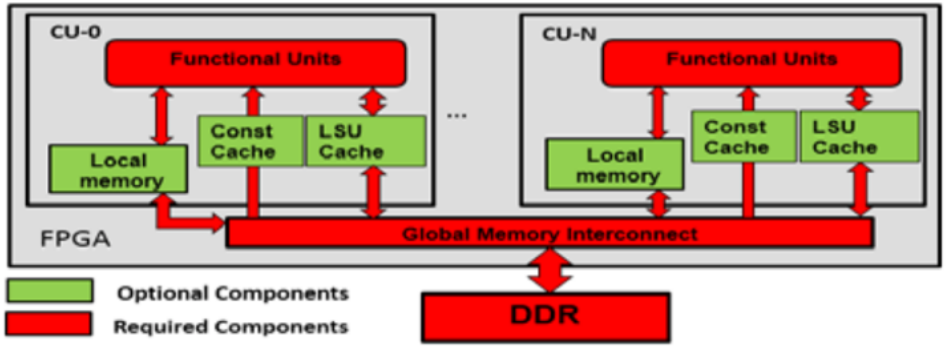


Altera OpenCL



Altera OpenCL optimizations

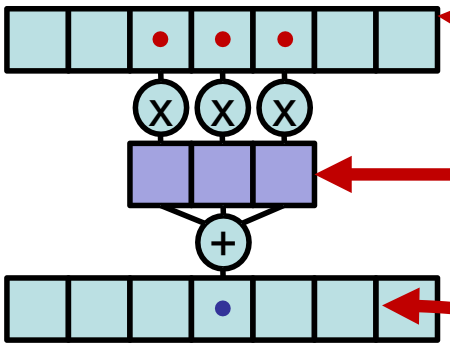
- Locality:
  - Shift-Register Pattern (SRP)
  - Local Memory (LM)
  - Constant Memory (CM)
  - Load-Store Units Buffers (LSU)
- Loop Unrolling (LU)
  - Deeper pipelines
  - Optimized tree-like reductions
- Kernel vectorization (SIMD)
  - Wider ALUs
- Compute Unit Replication (CUR)
  - Replicate Pipelines
  - Downsides: memory divergence ⬆, complexity ⬆, frequency ⬇



Shift Register Pattern

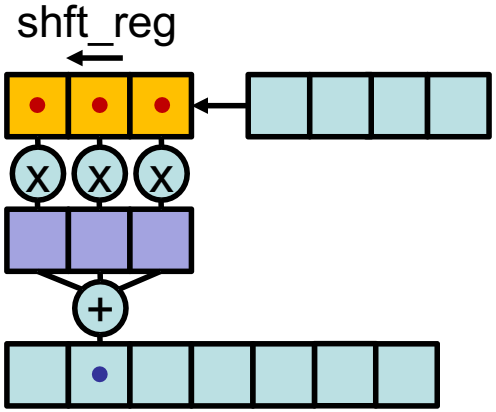
- Example: 1D stencil computation

NDRange
No SRP



```
float acc;
int i = get_global_id(0);
acc  = inArray[i-1] * filterArray[i-1];
acc += inArray[i]   * filterArray[i];
acc += inArray[i+1] * filterArray[i+1];
outArray[i] = acc;
```

SingleTask
SRP




```
float shft_reg[3];
shft_reg[1] = inArray[0];
shft_reg[2] = inArray[1];

for (i=1; i<N-1; i++){
  shft_reg[0] = shft_reg[1];
  shft_reg[1] = shft_reg[2];
  shft_reg[2] = inArray[i+1];

  acc  = shft_reg[0] * filterArray[i-1];
  acc += shft_reg[1] * filterArray[i];
  acc += shft_reg[2] * filterArray[i+1];
  outArray[i] = acc;
}
```

Shift !!

64x faster than Naive (17 elem. filter)
“Tuning Stencil Codes in OpenCL for FPGAs”, ICCD’16

UNIVERSIDAD DE MÁLAGA

Agenda

- Part I: Motivation & Background (30’)
- Part II: TBB heterogeneous features (50’)
- Part III: Heterogen. Flow Graph node (80’)
- Part IV: Templates on top of TBB (20’)