Making **the most** out of **Heterogeneous Chips** with CPU, **GPU and FPGA**

Rafael Asenjo

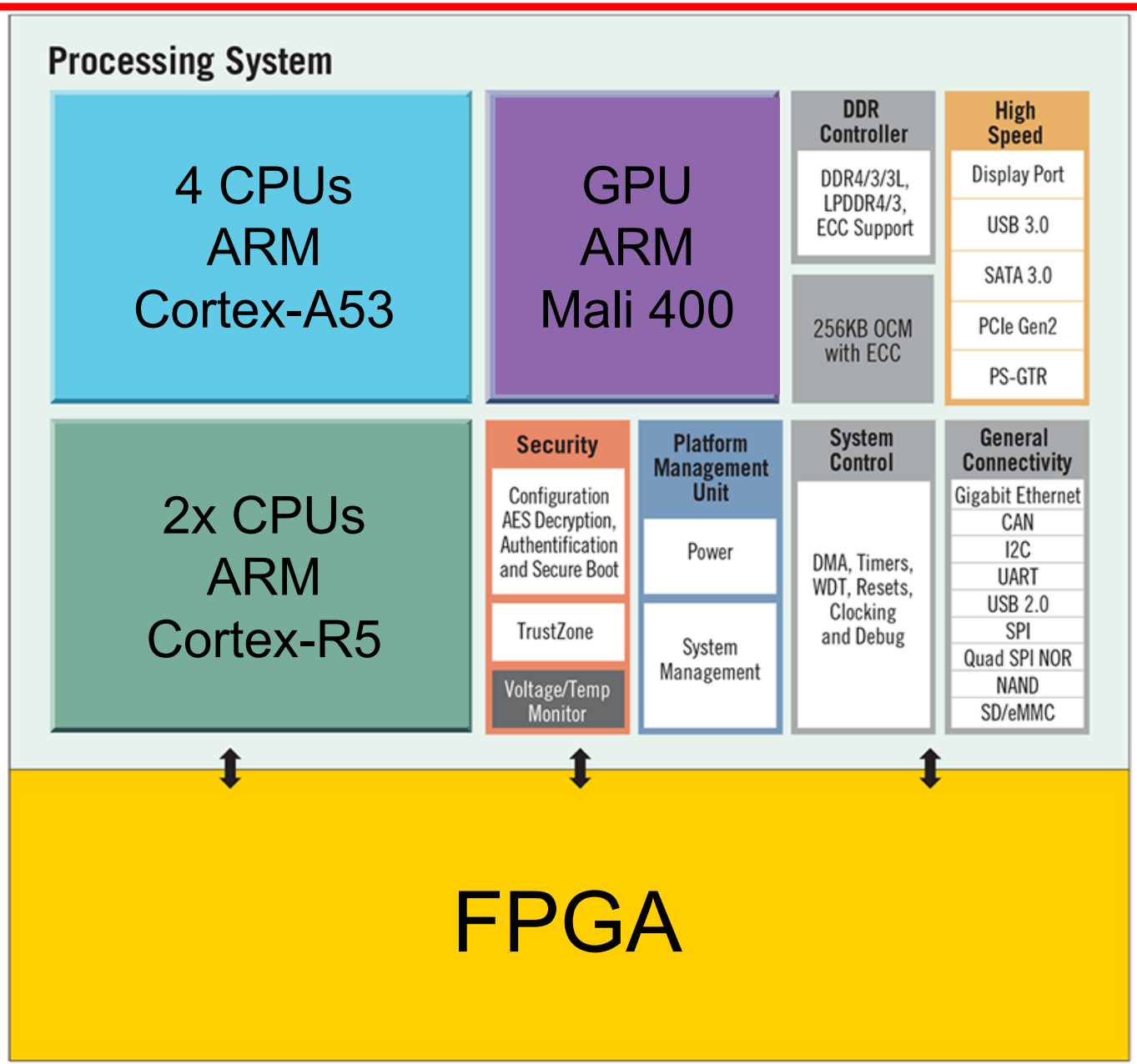
Dept. of Computer Architecture

University of Malaga, Spain.

Arquitectura de Computadores


http://motherrimmy.com

Xilinx Zynq UltraScale+ MPSoC



Note: Illustration not drawn to scale.

Agenda

- **Motivation**

- Hardware: iGPUs and iFPGAs

- Software
  - Programming models for chips with iGPUs and iFPGAs

- Our heterogeneous templates based on TBB
  - parallel_for
  - parallel pipeline

Motivation

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

Motivation

- **GPUs came to rescue**:
  - Massive Data Parallel Code at a low price in terms of power
  - Supercomputers and servers: NVIDIA
- **GREEN500 Top 5**:
  - June 2017
- Top500.org (Nov 2017):
  - 87 systems w. NVIDIA
  - 12 systems w. Xeon Phi
  - 5 systems w. PEZY

System	Cores	(TFlop/s)	(kW)	(GFlops/watts)
TSUBAME3.0 - SGI ICE XA, IP139-SXM2, Xeon E5-2680v4 14C 2.4GHz, Intel Omni-Path, NVIDIA Tesla P100 SXM2 , HPE GSIC Center, Tokyo Institute of Technology Japan	36,288	1,998.0	142	14.110
kukai - ZettaScaler-1.6 GPGPU system, Xeon E5-2650Lv4 14C 1.7GHz, Infiniband FDR, NVIDIA Tesla P100 , ExaScalar Yahoo Japan Corporation Japan	10,080	460.7	33	14.046
AIST AI Cloud - NEC 4U-8GPU Server, Xeon E5-2630Lv4 10C 1.8GHz, Infiniband EDR, NVIDIA Tesla P100 SXM2 , NEC National Institute of Advanced Industrial Science and Technology Japan	23,400	961.0	76	12.681
RAIDEN GPU subsystem - NVIDIA DGX-1, Xeon E5-2698v4 20C 2.2GHz, Infiniband EDR, NVIDIA Tesla P100 , Fujitsu Center for Advanced Intelligence Project, RIKEN Japan	11,712	635.1	60	10.603
Wilkes-2 - Dell C4130, Xeon E5-2650v4 12C 2.2GHz, Infiniband EDR, NVIDIA Tesla P100 , Dell University of Cambridge United Kingdom	21,240	1,193.0	114	10.428

Xeon + Tesla P100

Xeon + Tesla P100

Xeon + Tesla P100

Xeon + Tesla P100

Xeon + Tesla P100

Green 500 Nov 2017

Rank	TOP500 Rank	System	Cores	Rmax (TFlop/s)	Power (kW)	Power Efficiency (GFlops/watts)
1	259	**Shoubu system B** - ZettaScaler-2.2, Xeon D-1571 16C 1.3GHz, Infiniband EDR, PEZY-SC2 , PEZY Computing / Exascaler Inc. Advanced Center for Computing and Communication, RIKEN Japan	794,400	842.0	50	17.009
2	307	**Suiren2** - ZettaScaler-2.2, Xeon D-1571 16C 1.3GHz, Infiniband EDR, PEZY-SC2 , PEZY Computing / Exascaler Inc. High Energy Accelerator Research Organization /KEK Japan	762,624	788.2	47	16.759
3	276	**Sakura** - ZettaScaler-2.2, Xeon E5-2618Lv3 8C 2.3GHz, Infiniband EDR, PEZY-SC2 , PEZY Computing / Exascaler Inc. PEZY Computing K.K. Japan	794,400	824.7	50	16.657
4	149	**DGX SaturnV Volta** - NVIDIA DGX-1 Volta36, Xeon E5-2698v4 20C 2.2GHz, Infiniband EDR, NVIDIA Tesla V100 , Nvidia NVIDIA Corporation United States	22,440	1,070.0	97	15.113
5	4	**Gyoukou** - ZettaScaler-2.2 HPC system, Xeon D-1571 16C 1.3GHz, Infiniband EDR, PEZY-SC2 700Mhz , ExaScaler Japan Agency for Marine-Earth Science and Technology Japan	19,860,000	19,135.8	1,350	14.173

Xeon + PEZY

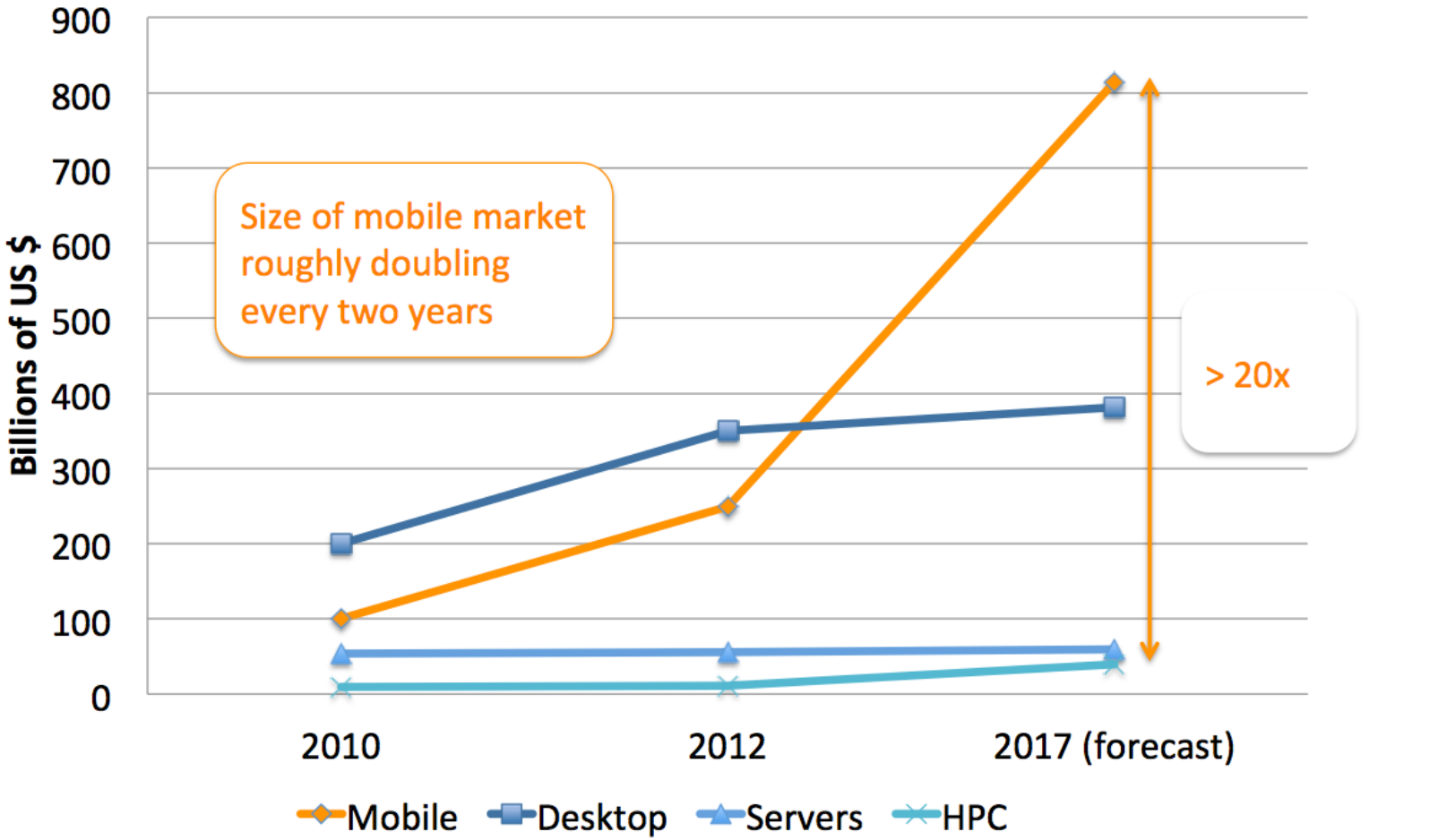
Xeon + PEZY

Xeon + PEZY

Xeon + PEZY

Motivation

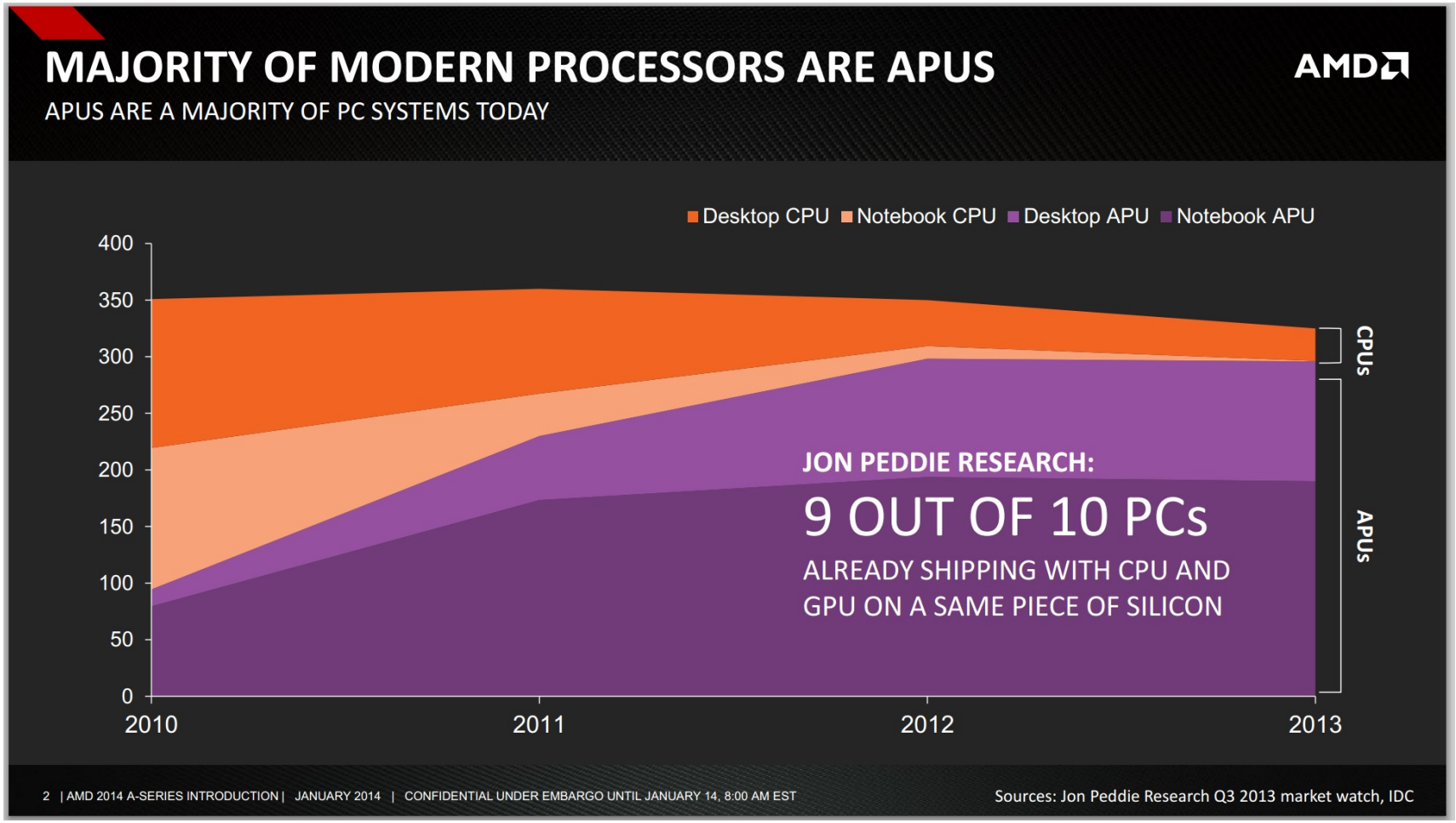
- There is (parallel) life beyond supercomputers:



Source: IDC. HPC forecast Intersect360

Motivation

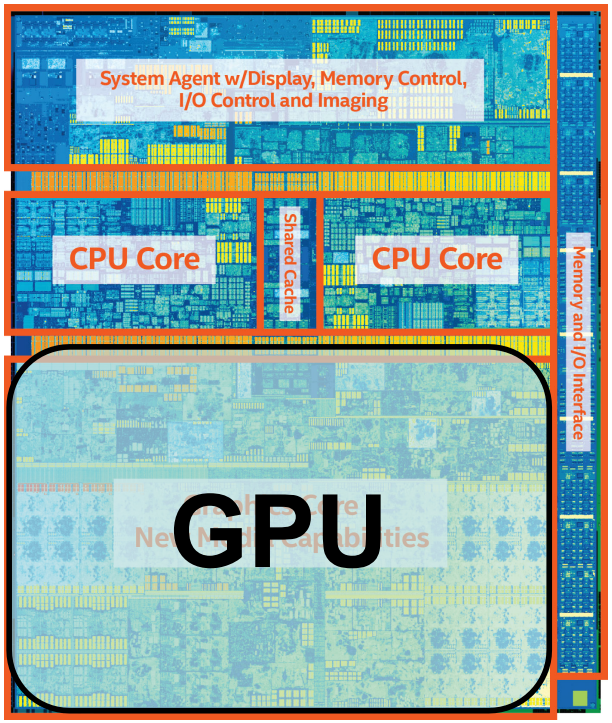
- Plenty of GPUs elsewhere:
  - Integrated GPUs **on more than 90%** of shipped processors



Motivation

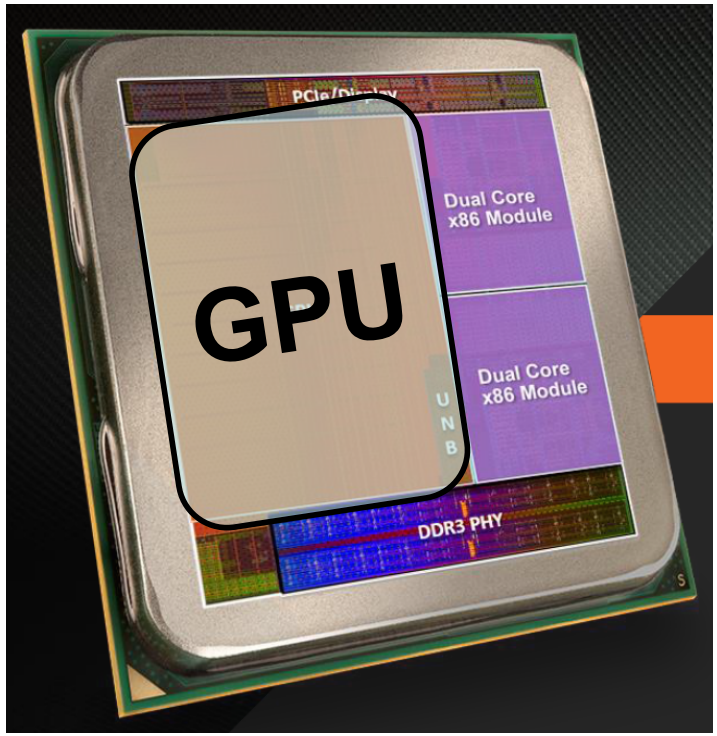
- Plenty of GPUs on desktops and laptops:
  - Desktops (35 – 130 W) and laptops (15 – 60 W):

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



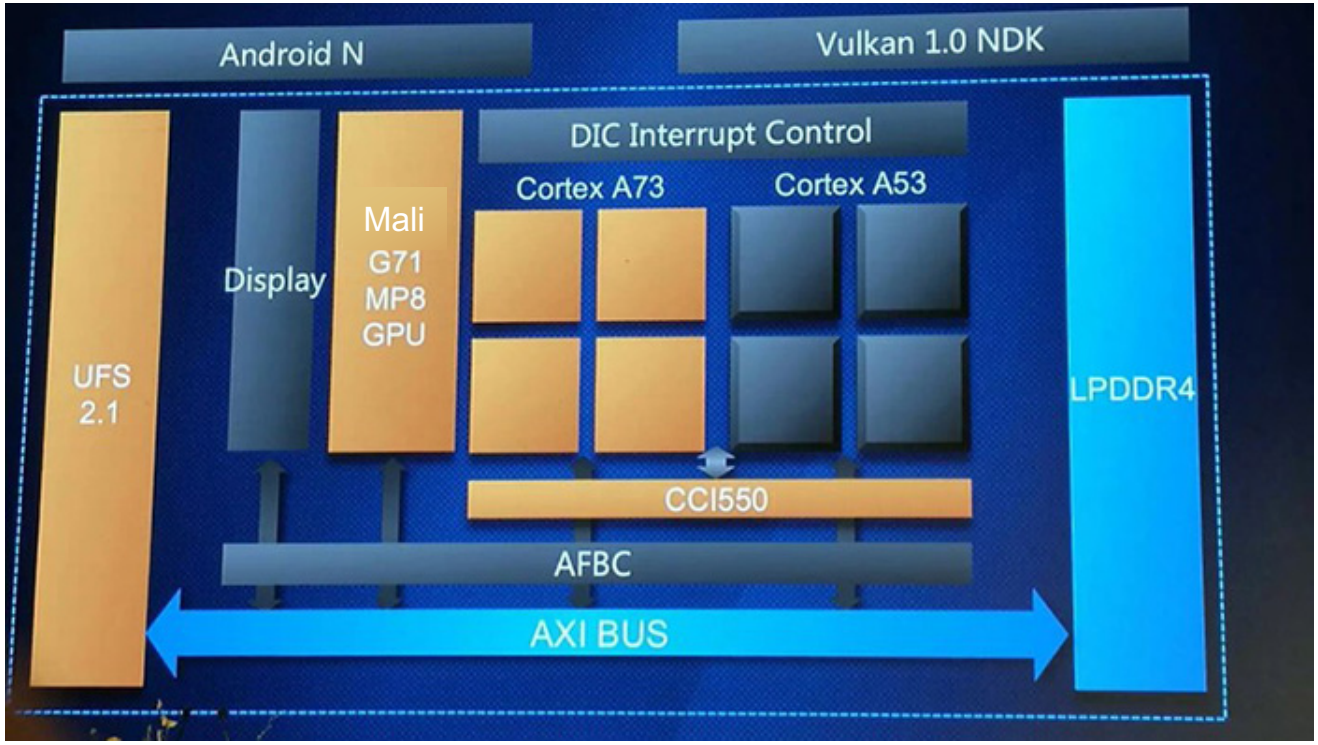
http://www.techspot.com/photos/article/770-amd-a8-7600-kaveri/

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

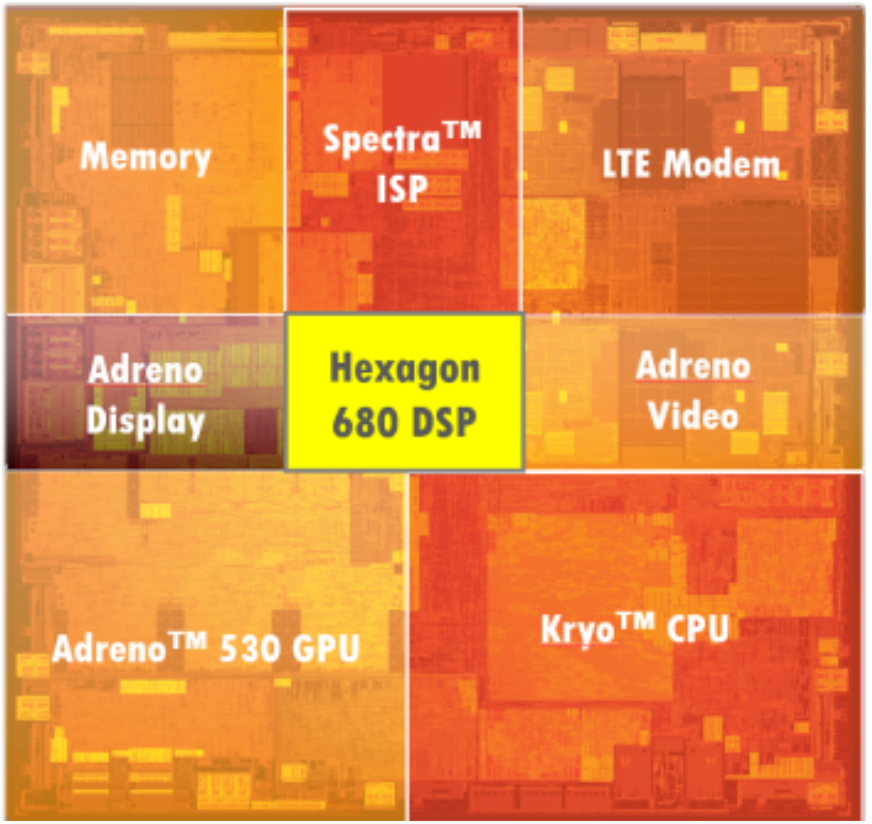
Motivation

- And user-programmable DSPs as well:

Qualcomm Snapdragon 820/821 (2 - 6 W)

Samsung Galaxy S7

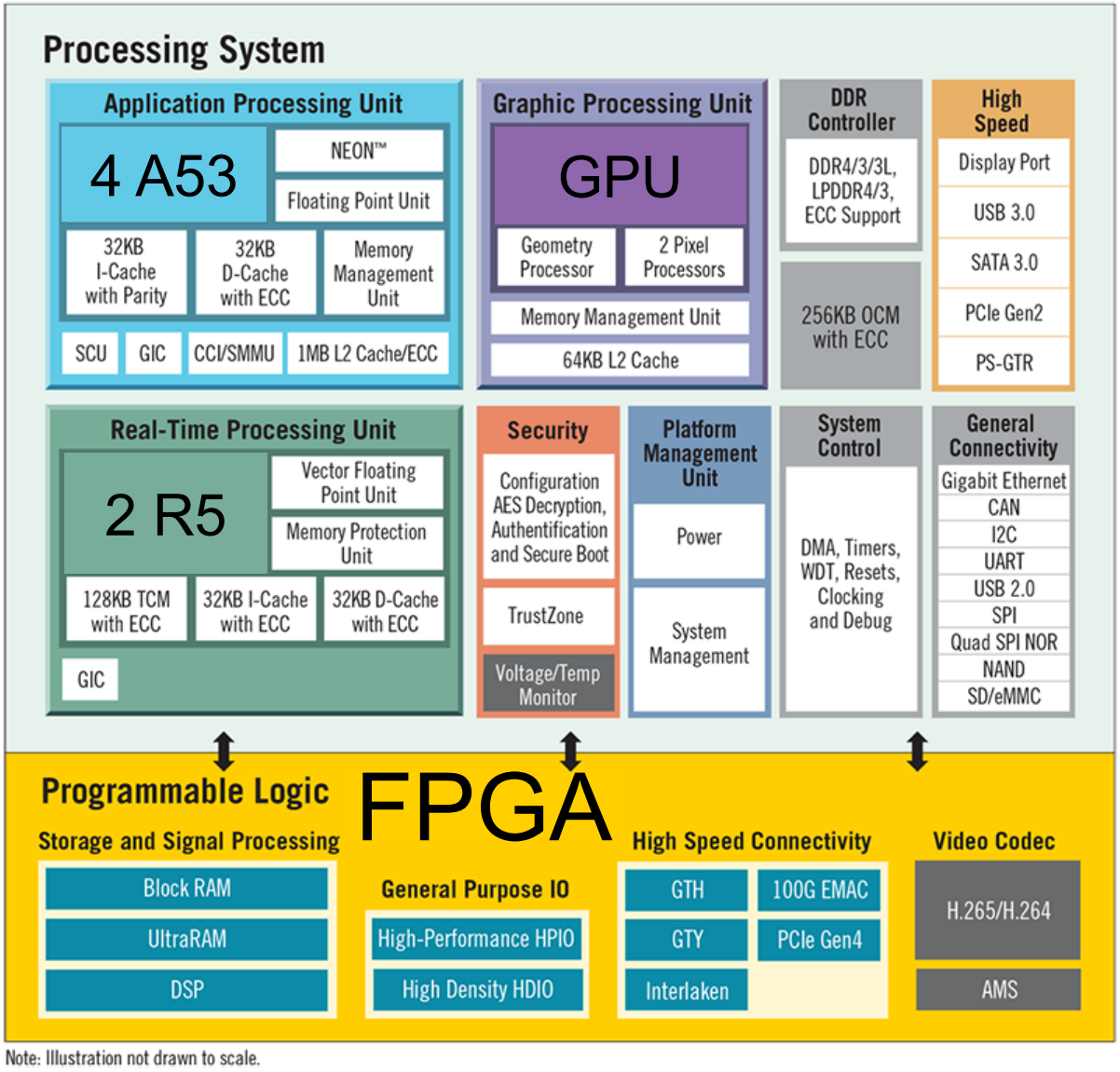
Google Pixel



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

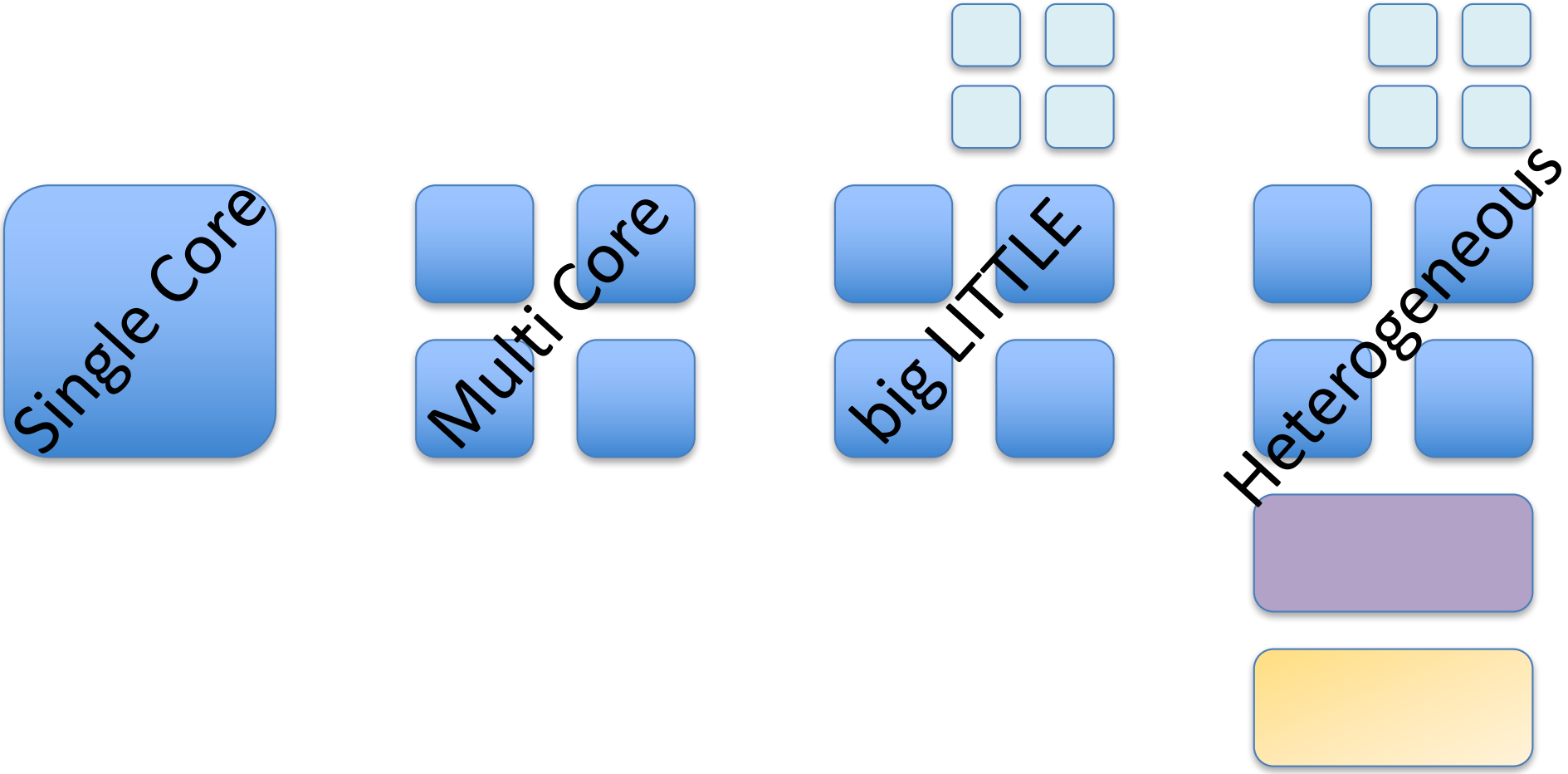


Note: Illustration not drawn to scale.

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

Agenda

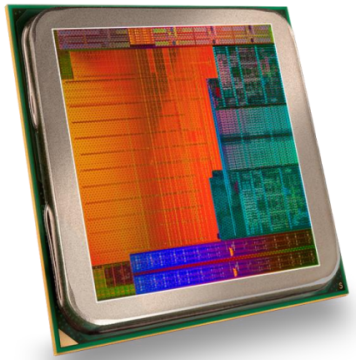
- Motivation
- **Hardware: iGPUs and iFPGAs**
- Software
  - Programming models for chips with iGPUs and iFPGAs
- Our heterogeneous templates based on TBB
  - parallel_for
  - parallel pipeline

Hardware: iGPUs

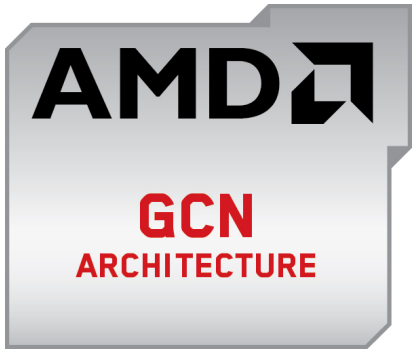
Intel Skylake

Iris Pro Graphics

AMD Kaveri

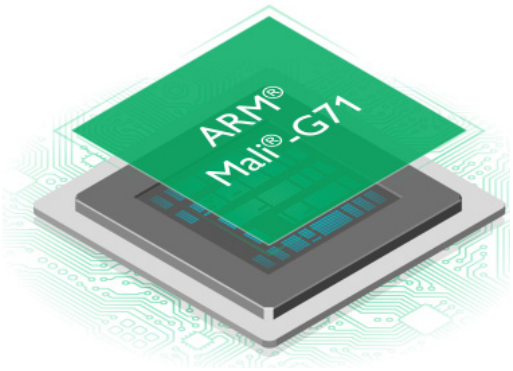


Graphic Core Next



HiSilicon Kirin 960

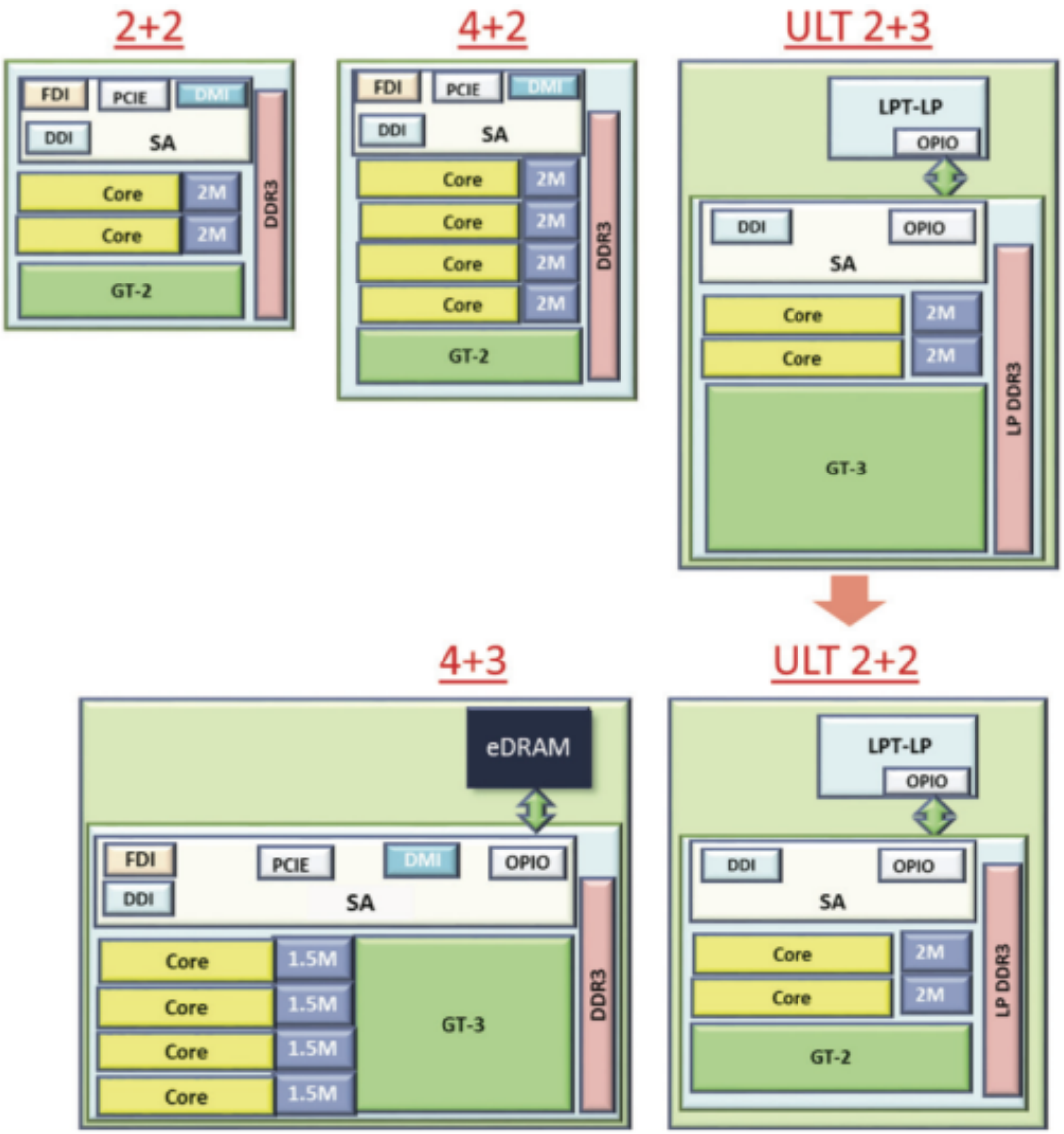
ARM Mali-G71



Intel Skylake – Kaby Lake

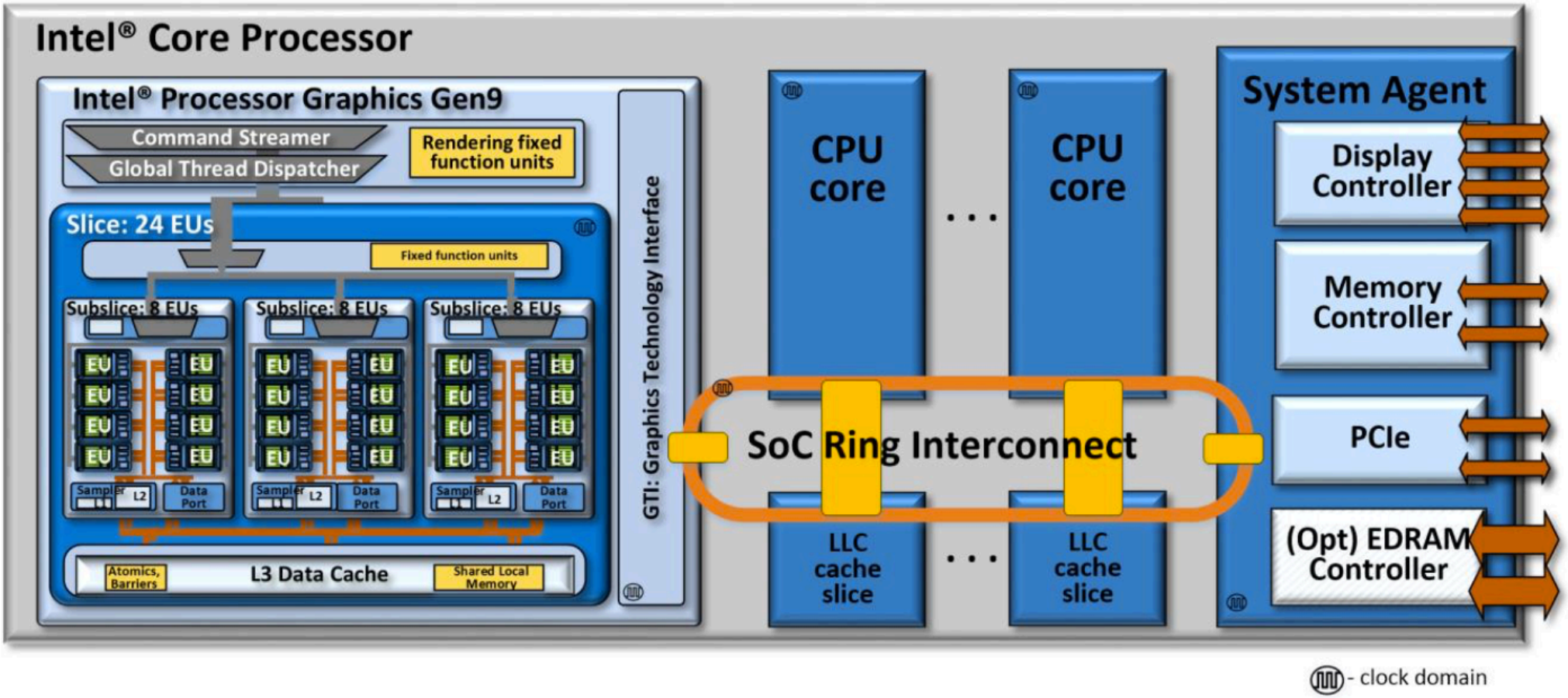
- Modular design
  - 2 or 4 cores
  - GPU
    - GT-1: 12 EU
    - GT-2: 24 EU
    - GT-3: 48 EU
    - GT-4: 72 EU

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



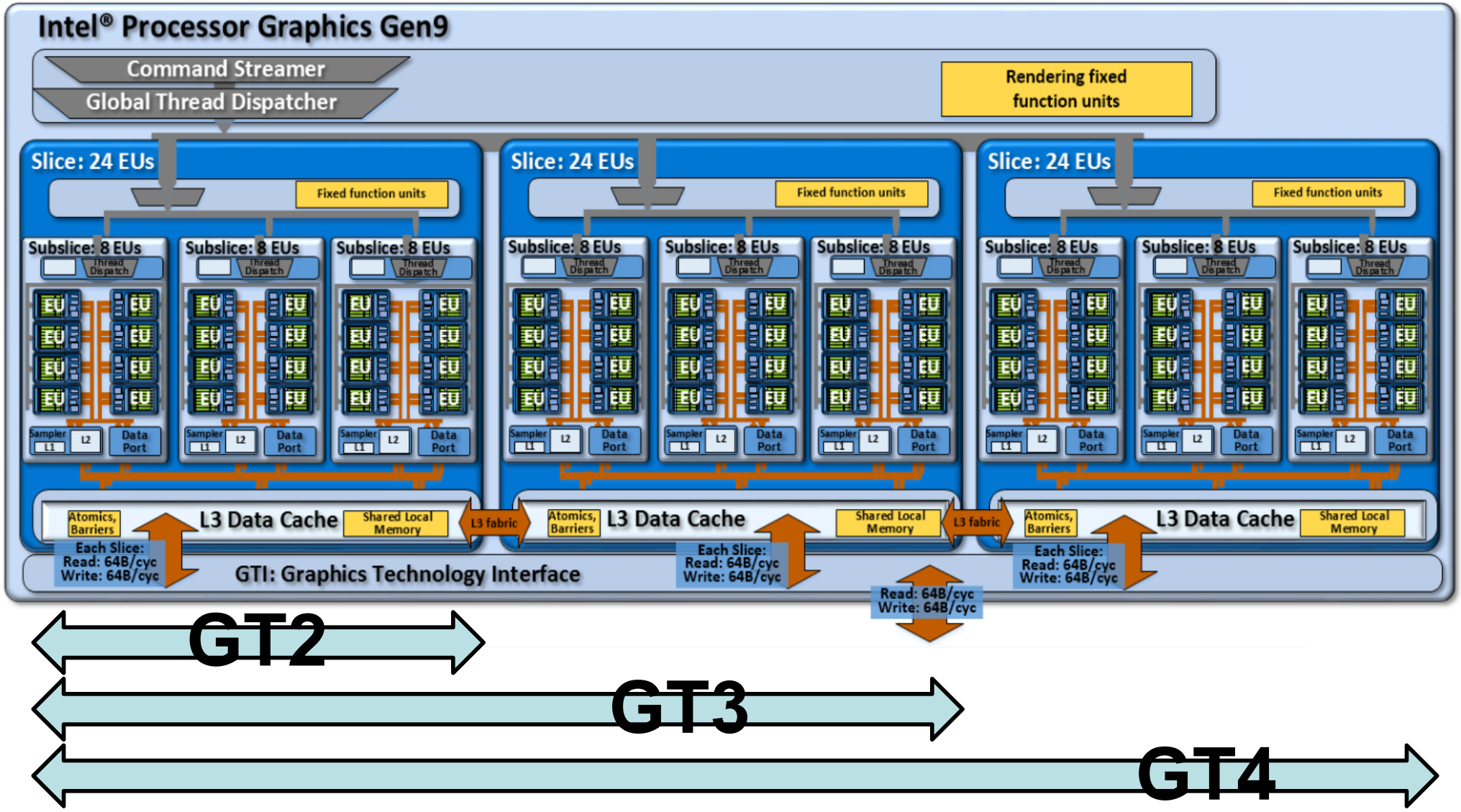
http://www.anandtech.com/show/6355/intels-haswell-architecture

Intel Skylake – Kaby Lake



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

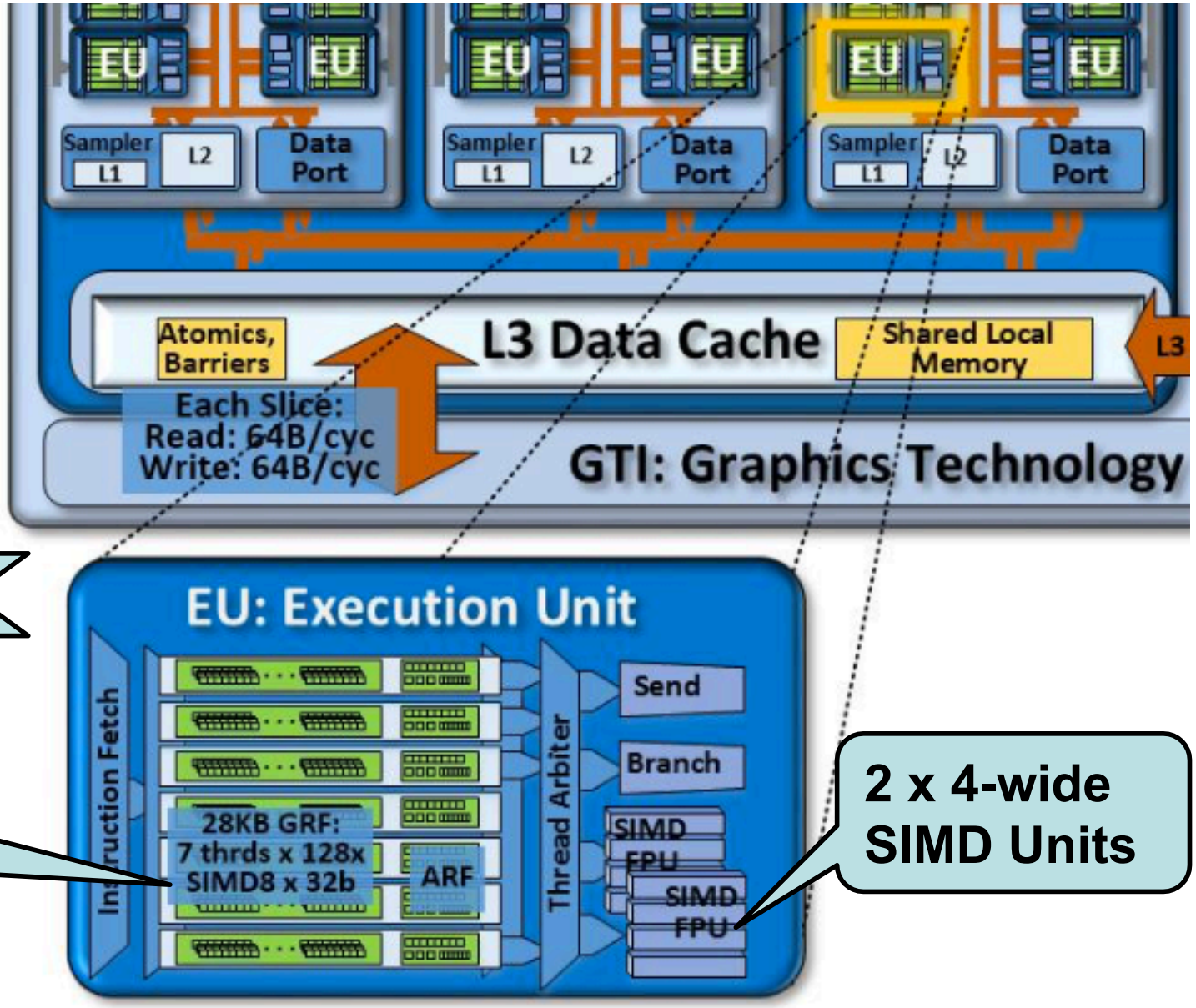
Intel Processor Graphics



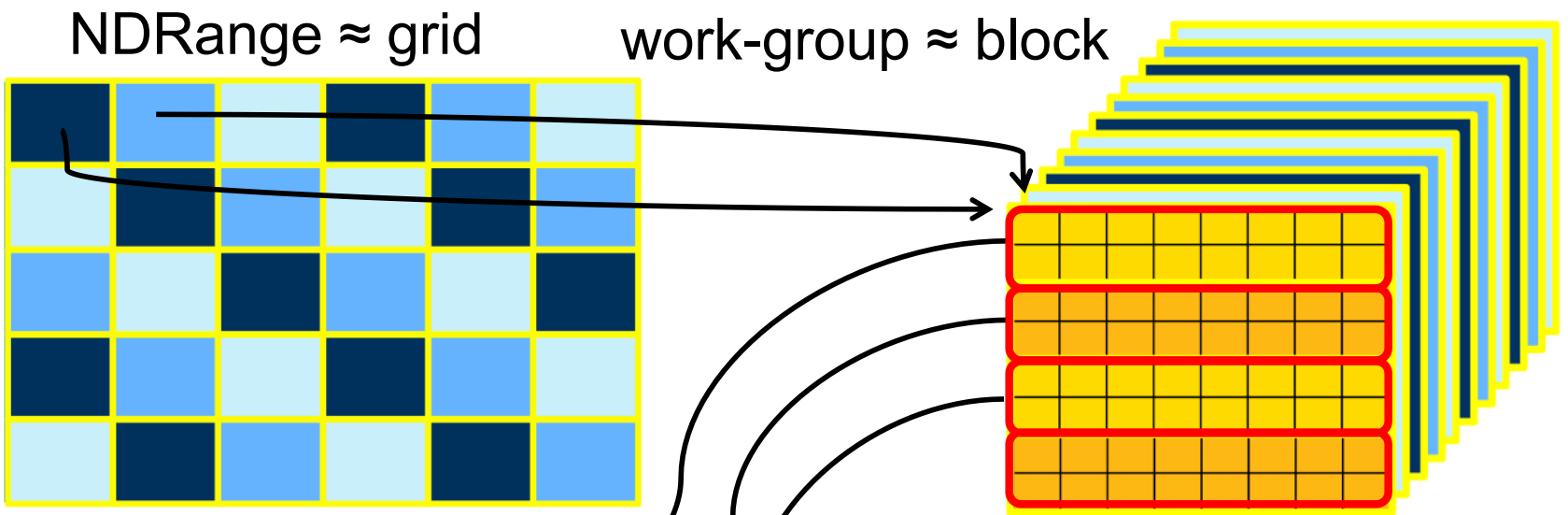
Intel Processor Graphics

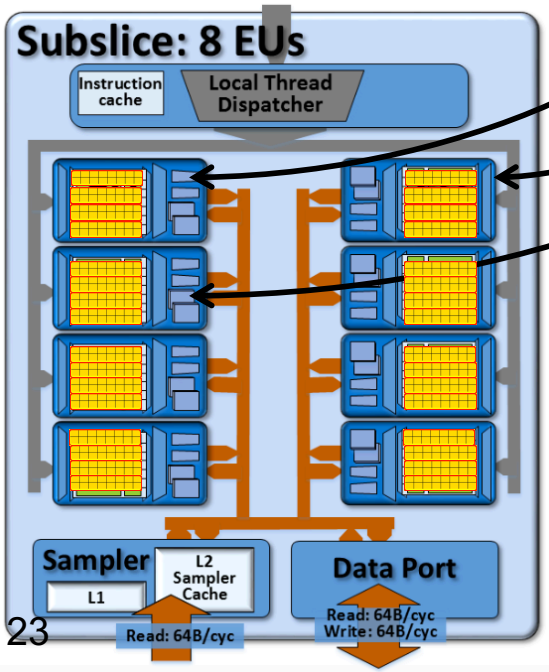
- Peak performance:
  - 72 EUs
  - 2 x 4-wide SIMD
  - 2 flop/ck (fmadd)
  - 1GHz

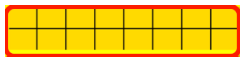
1.15 TFlops



Intel Processor Graphics



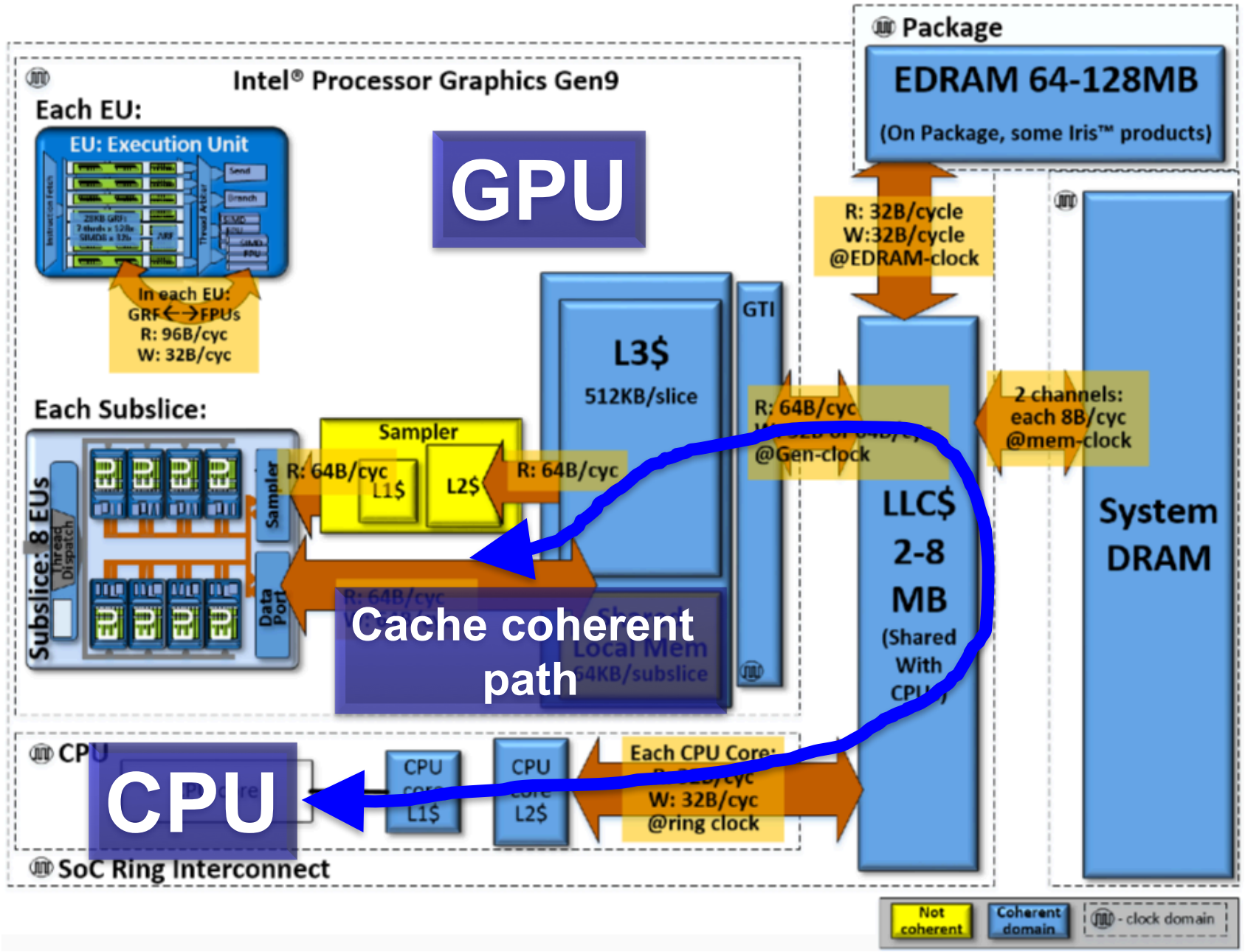




= EU-thread (SIMD16) ≈ wavefront ≈ warp:
16 work-items ≈ 16 thrds

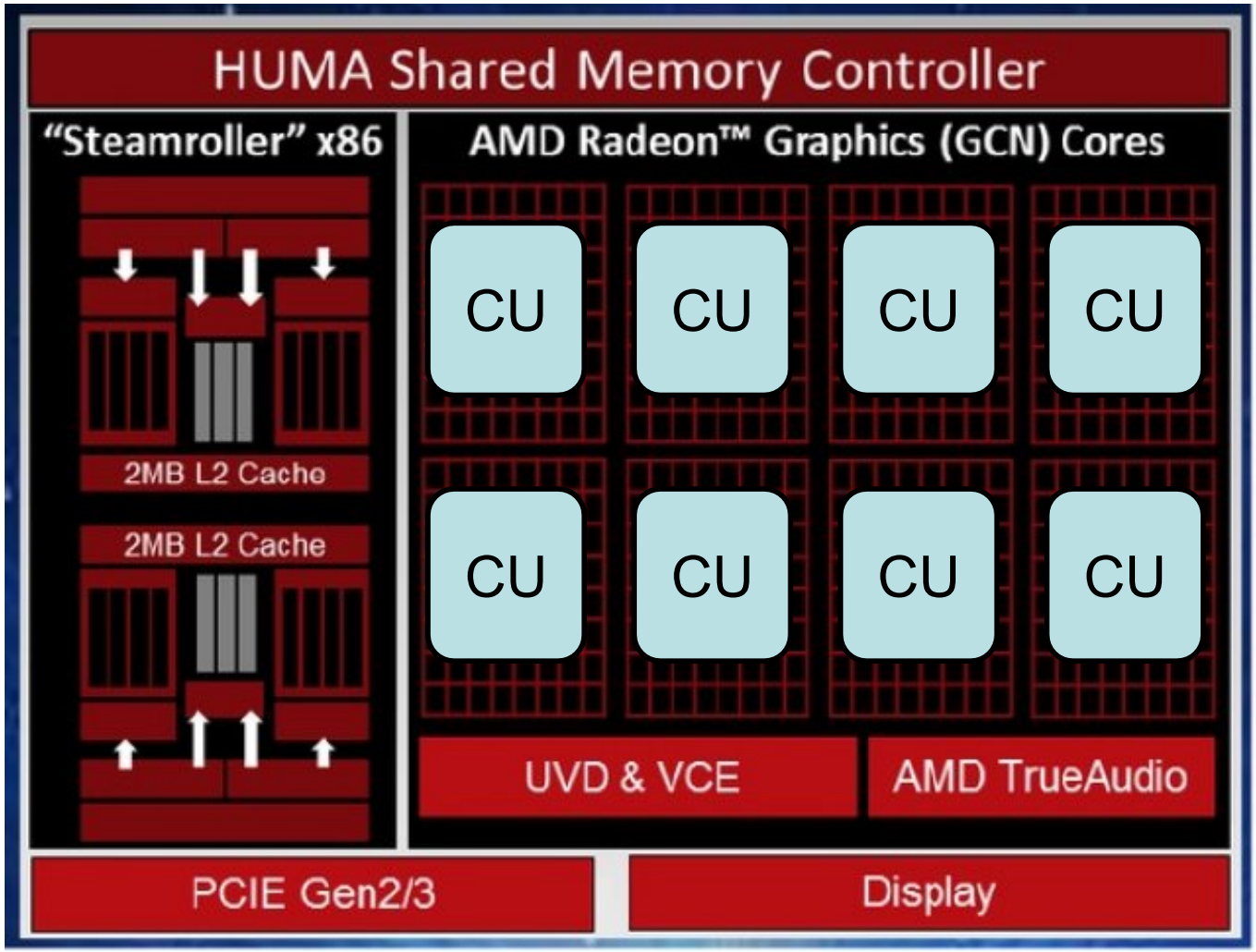
- Each SIMD16 wavefront takes 2cks
- IMT: fine-grained multi-threading
  - Latency hiding mechanism

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



AMD Kaveri

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

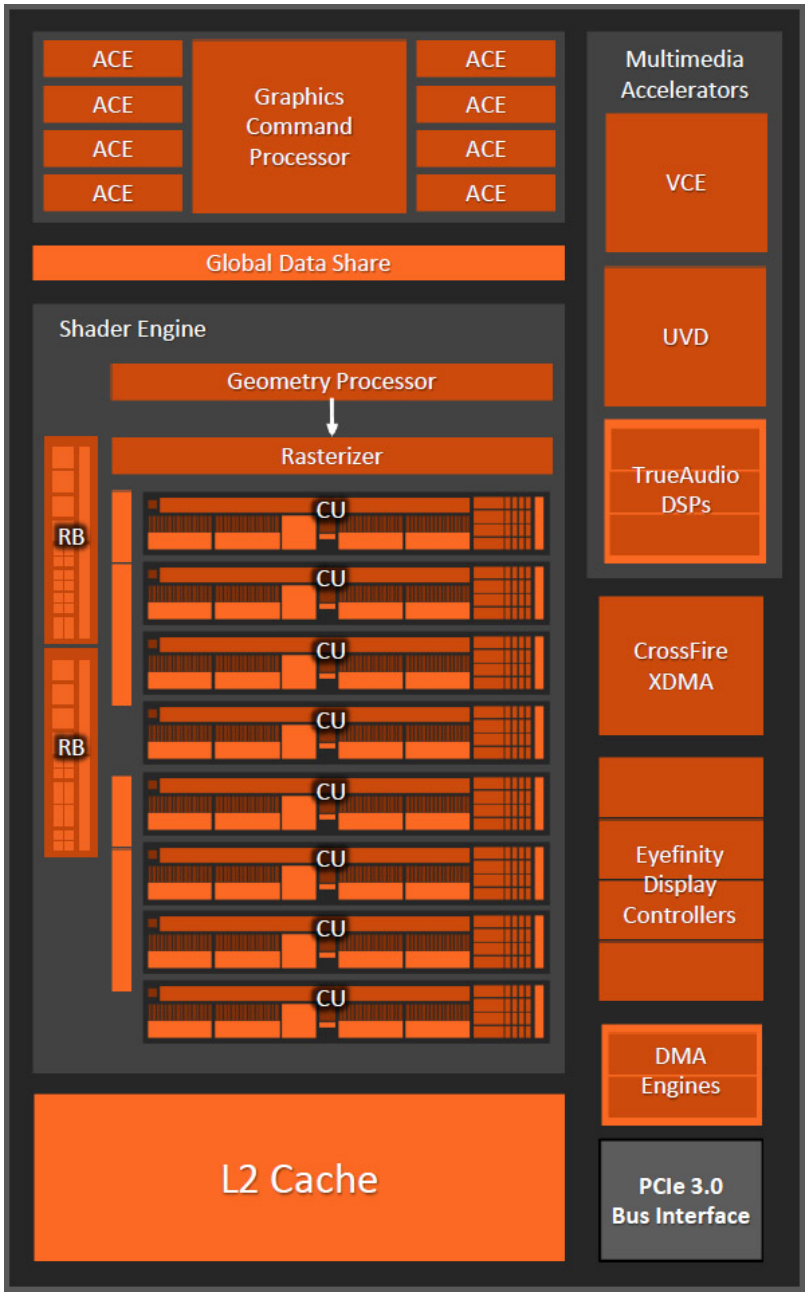


http://wccftech.com/

AMD Graphics Core Next (GCN)

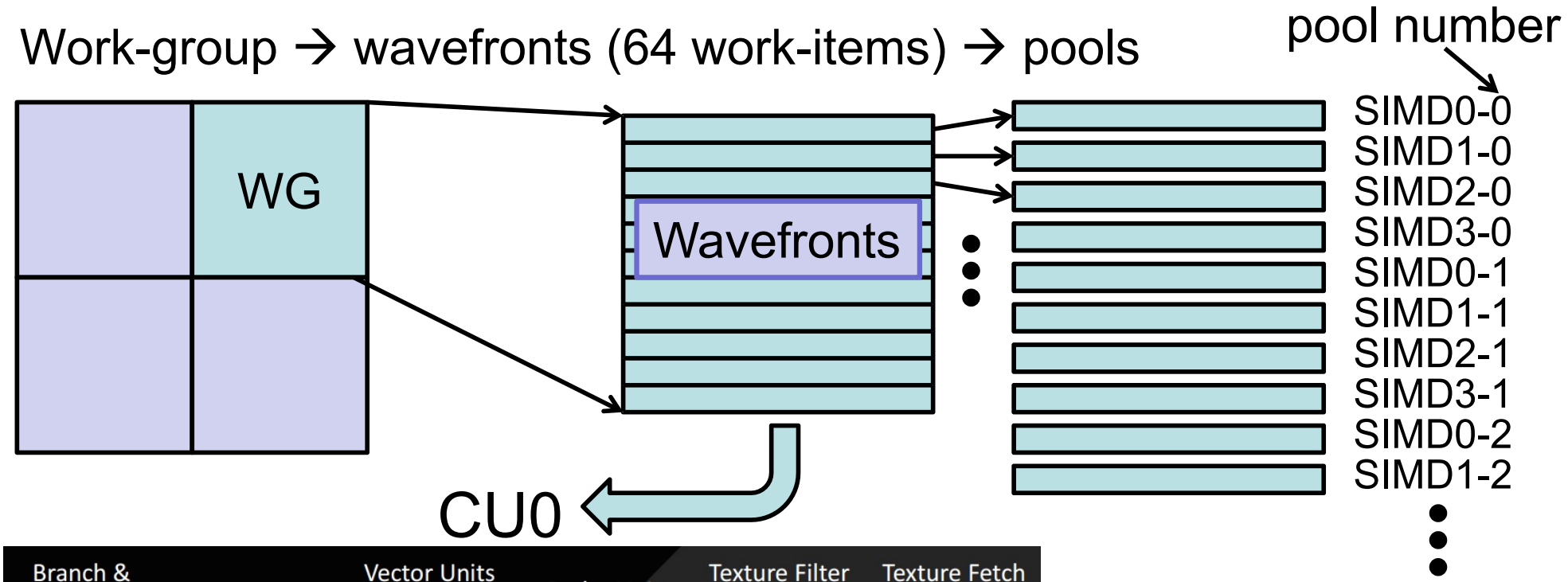
- In Kaveri,
  - 8 Compute Units (CU)
  - Each CU: **4 x 16-wide SIMD**
  - Total: 512 FPUs
  - 866 MHz

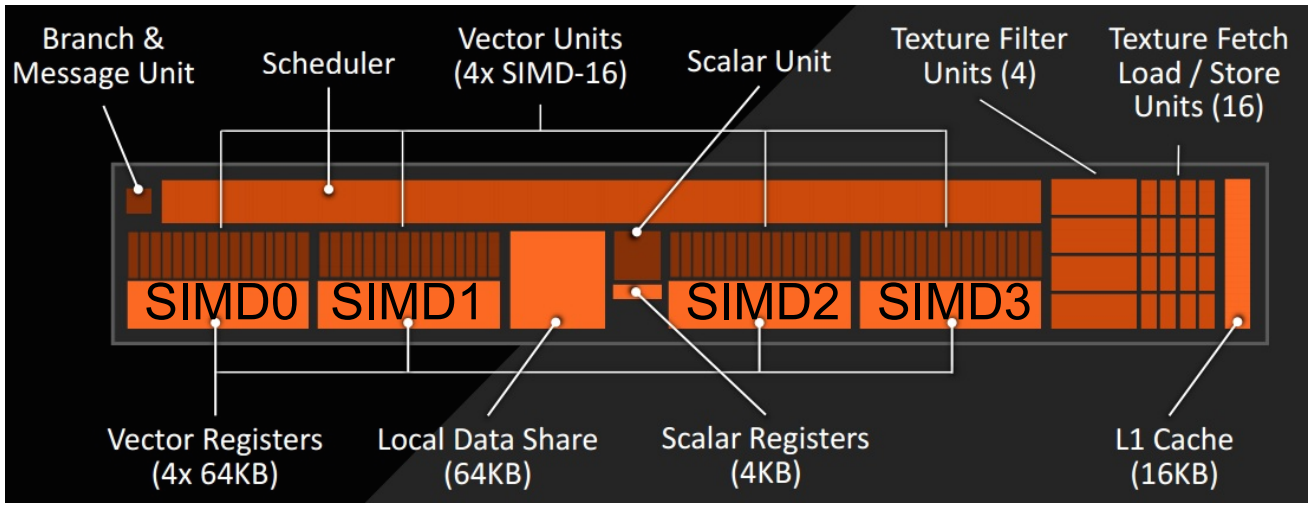
- Max GFLOPS=
  - 0.86 GHz x
  - 512 FPUs x
  - 2 fmad =
  - **880 GFLOPS**



OpenCL execution on GCN

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

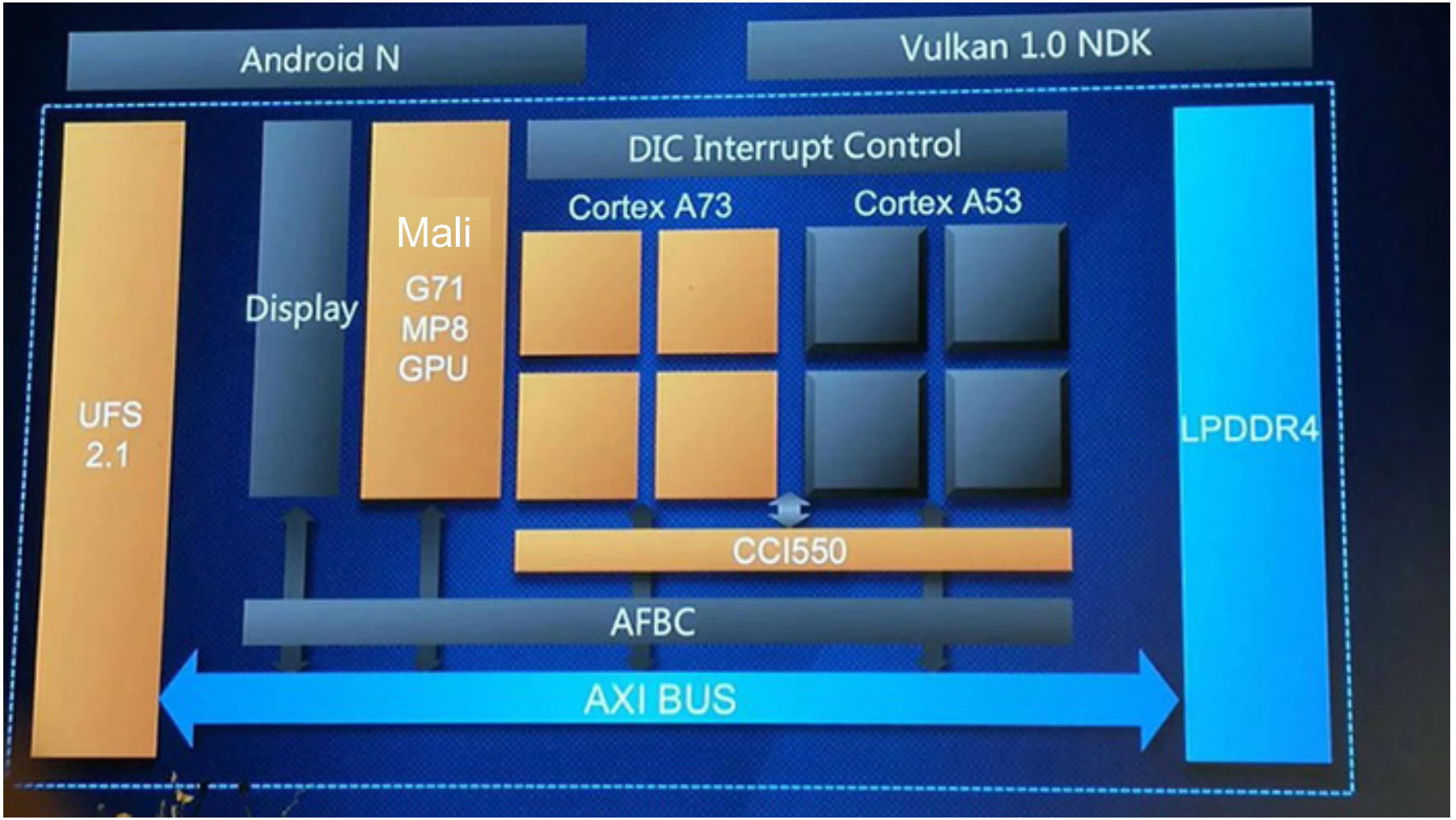




4 pools:
4 wavefronts in flight per SIMD

4 ck to execute each wavefront

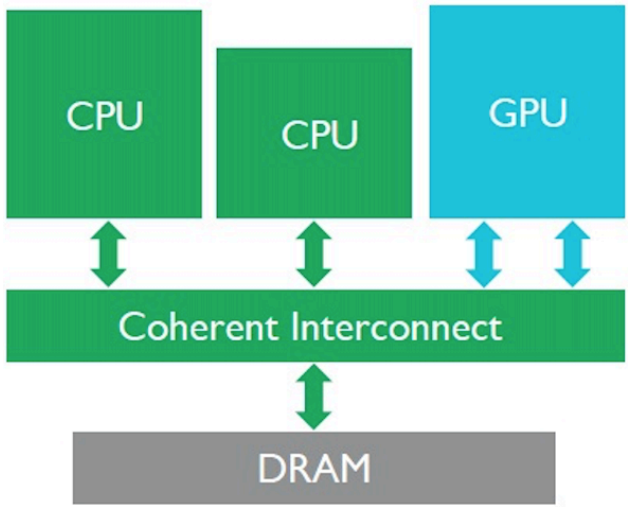
HiSilicon Kirin 960

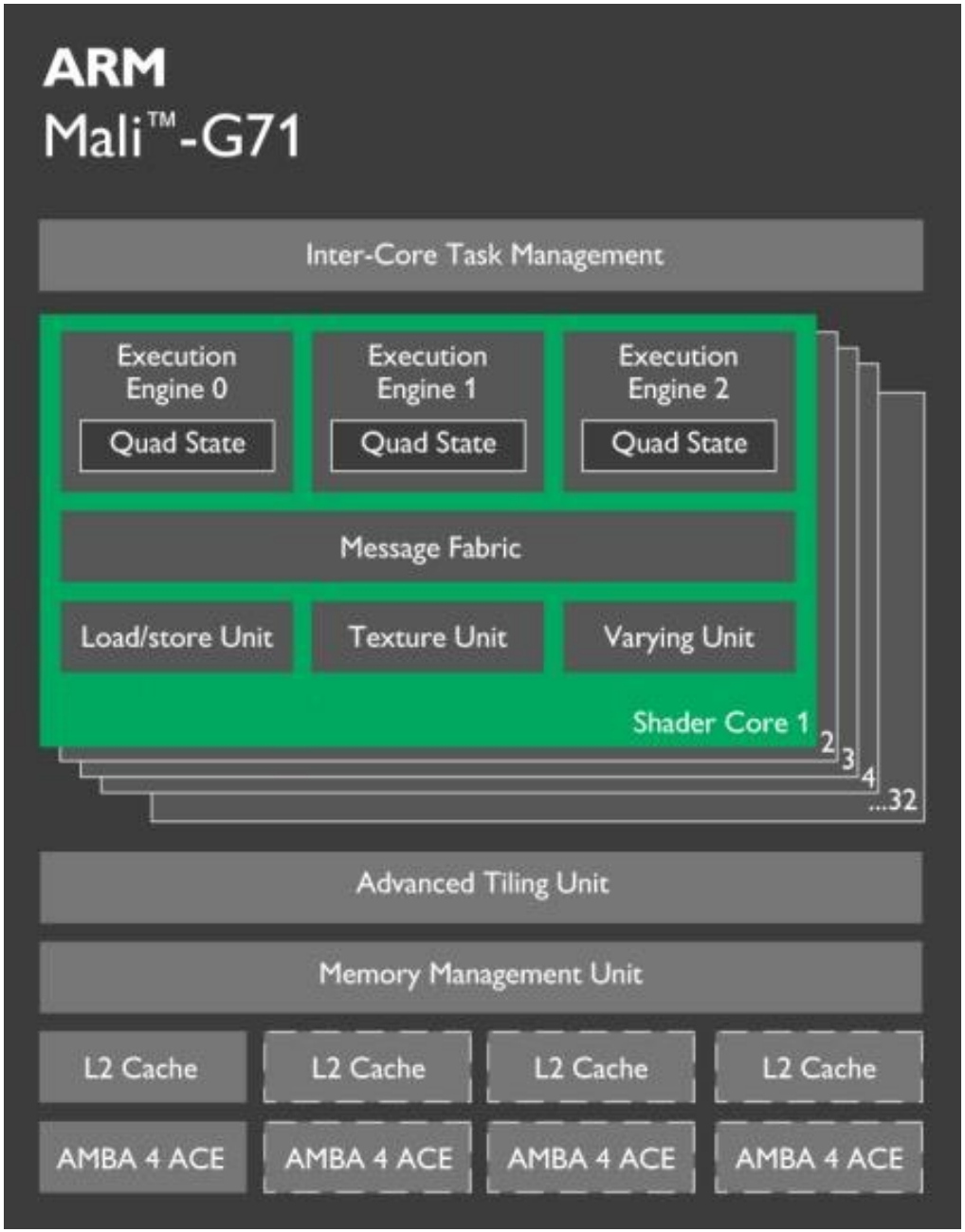


- ARM64 big.LITTLE (Cortex A73+Cortex A53) + Mali G71

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**



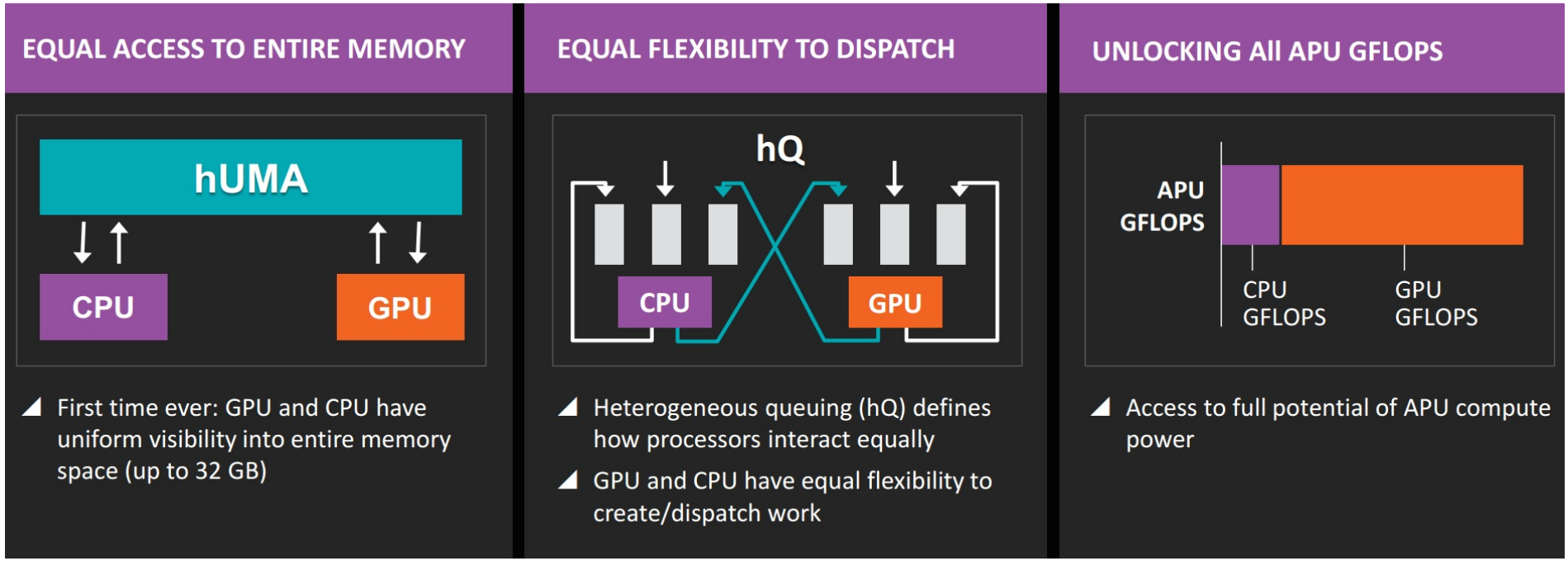


HSA (Heterogeneous System Architecture)

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

AMD ARM Imagination MEDIATEK QUALCOMM SAMSUNG

HSA FOUNDATION

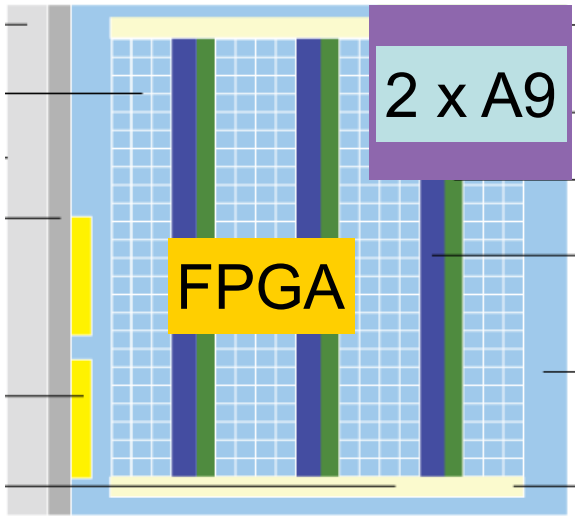


Advantages of iGPUs

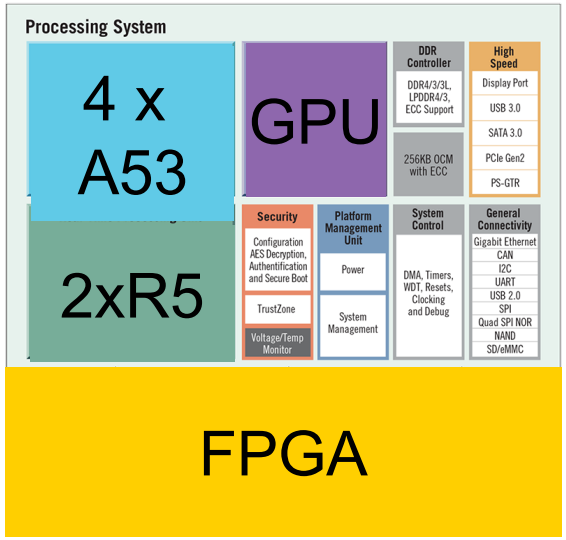
- Discrete and integrated GPUs: different goals
  - NVIDIA Pascal: 3584 CUDA cores, 250W, 10 TFLOPS
  - Intel Iris Pro Graphics 580: 72EU x 8 SIMD, ~15W, 1.15 TFLOPS
  - Mali-G71: 32 EU x 3 x 4 SIMD, ~ 2W, 0.652 TFLOPS
- CPU and GPU are both **first-class citizens**.
  - **SVM** and **platform atomics** 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!

Hardware: iFPGAs

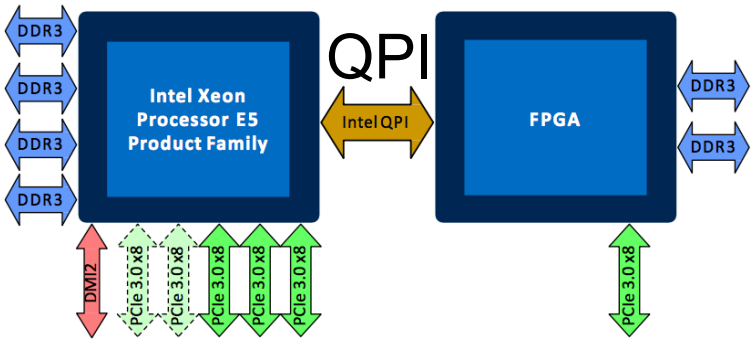
Altera Cyclone V



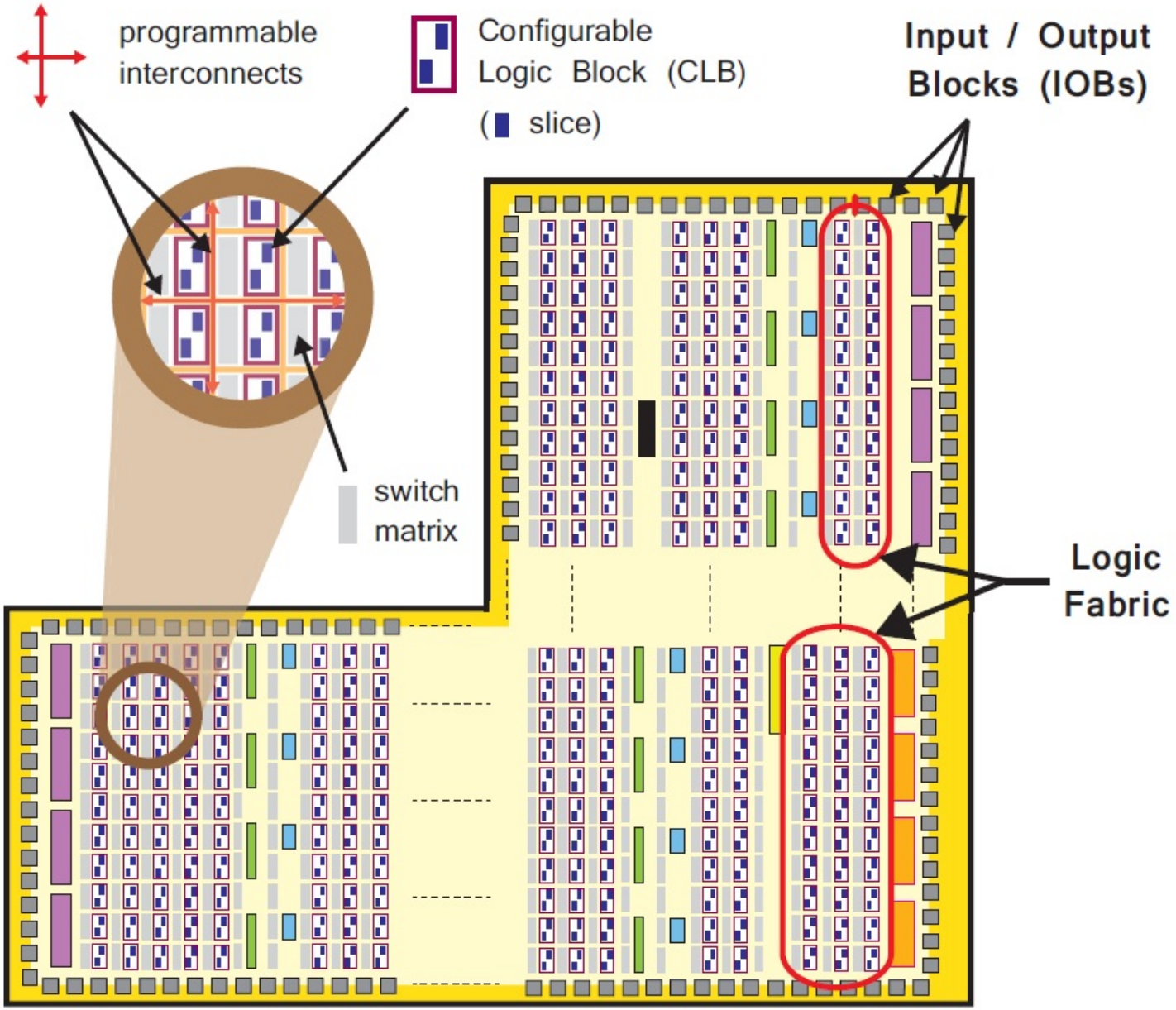
Xilinx Zynq UltraScale+



Intel Xeon+Arria10

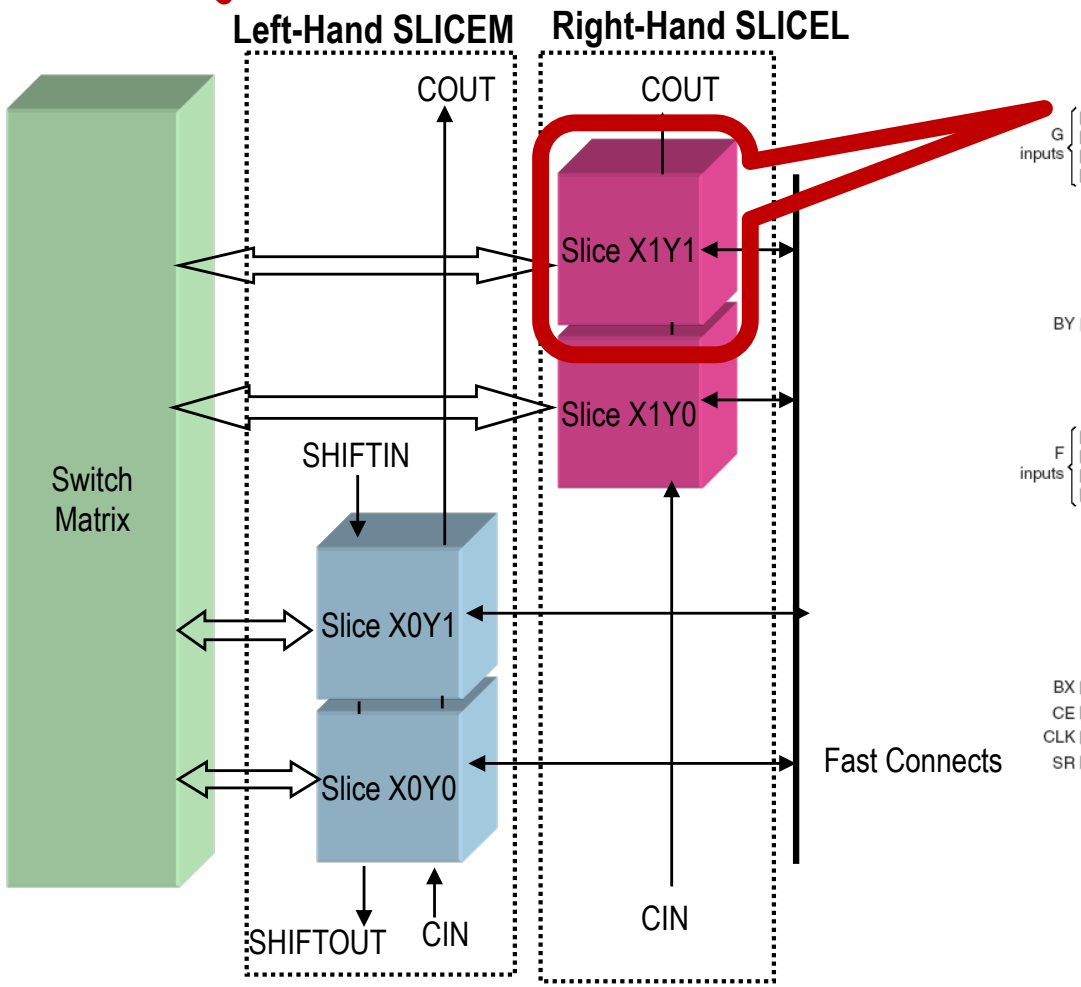


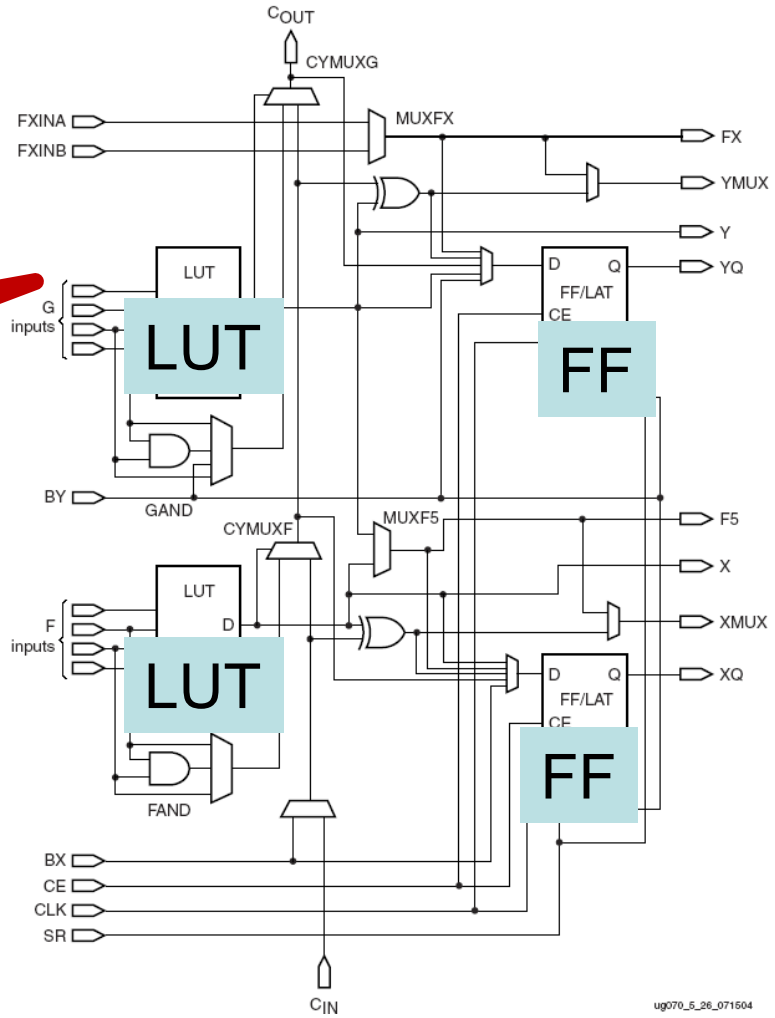
FPGA Architecture



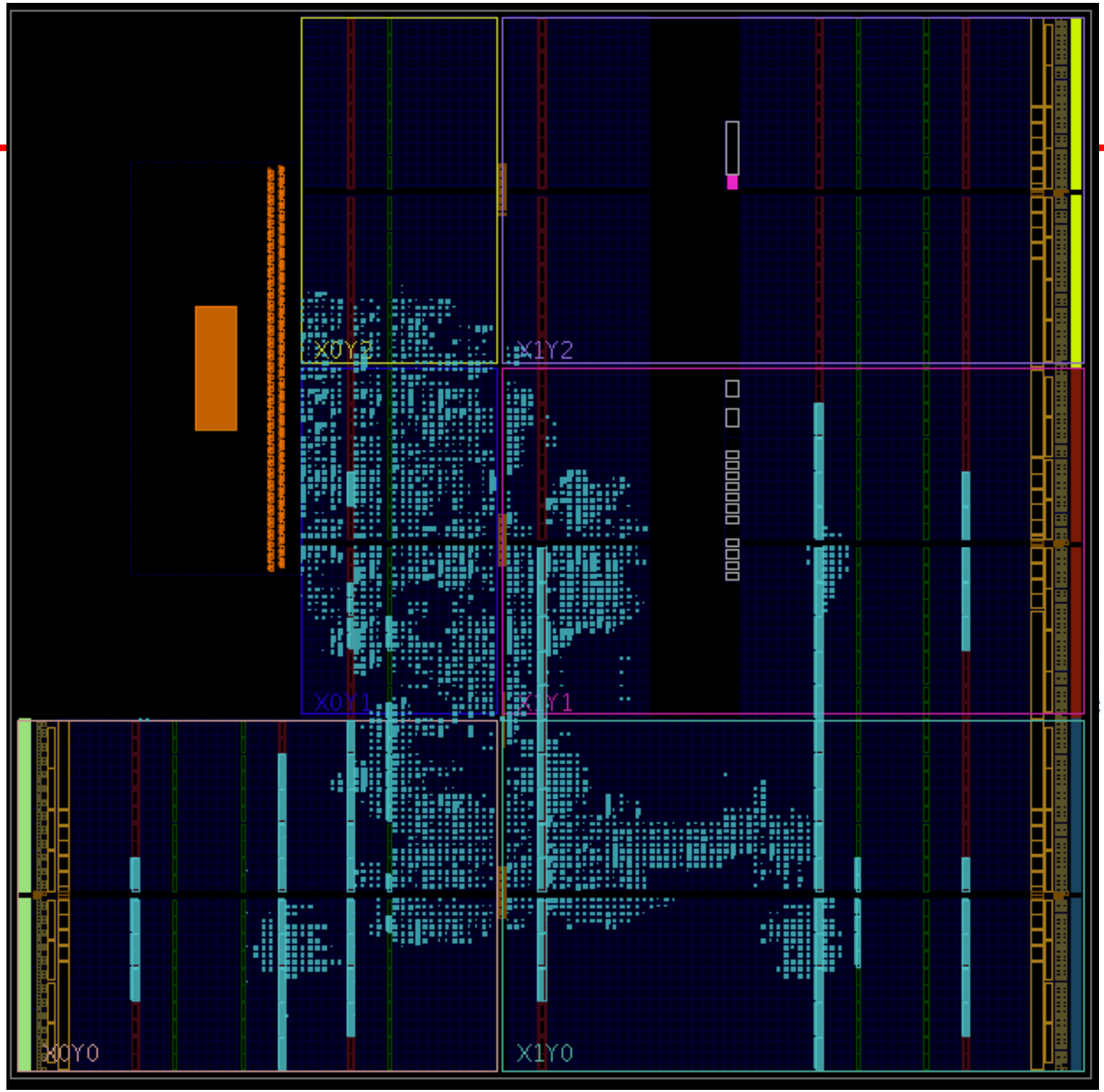
Courtesy: Javier Navaridas, U. Manchester.

CLB: Configurable Logic Block





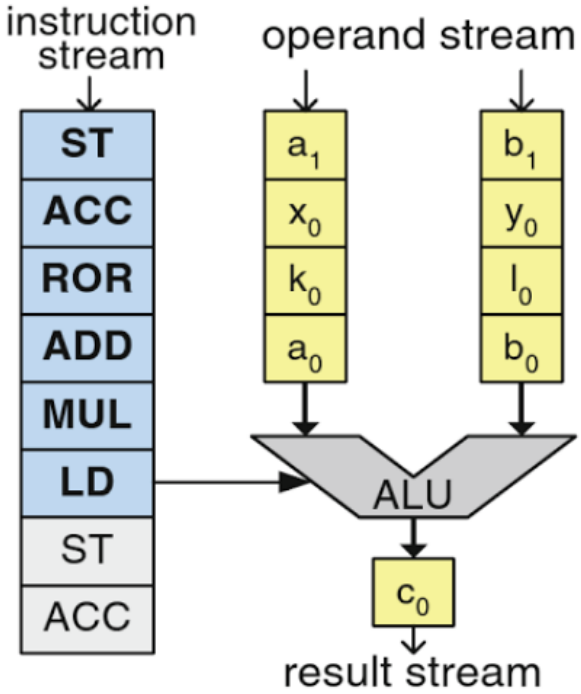
Courtesy: Jose Luis Nunez-Yanez, U. Bristol.


X0Y2
X1Y2
X0Y1
X1Y1
X0Y0
X1Y0

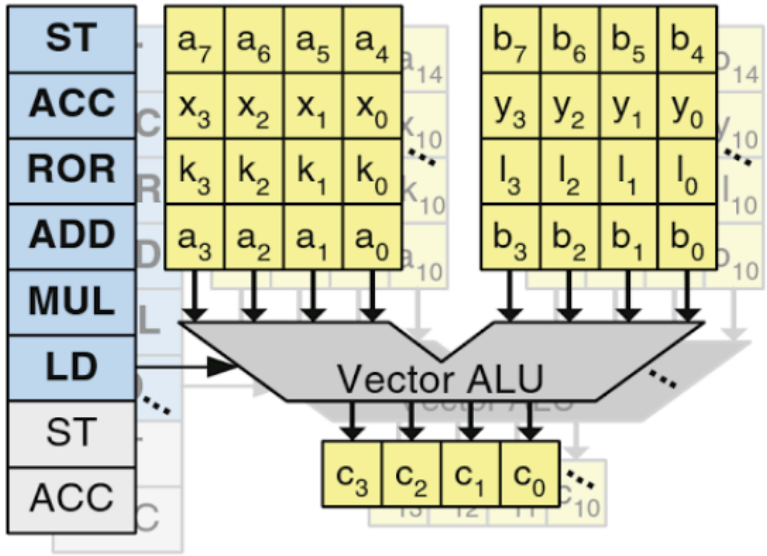
How do we “mold this liquid silicon”?

- 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:



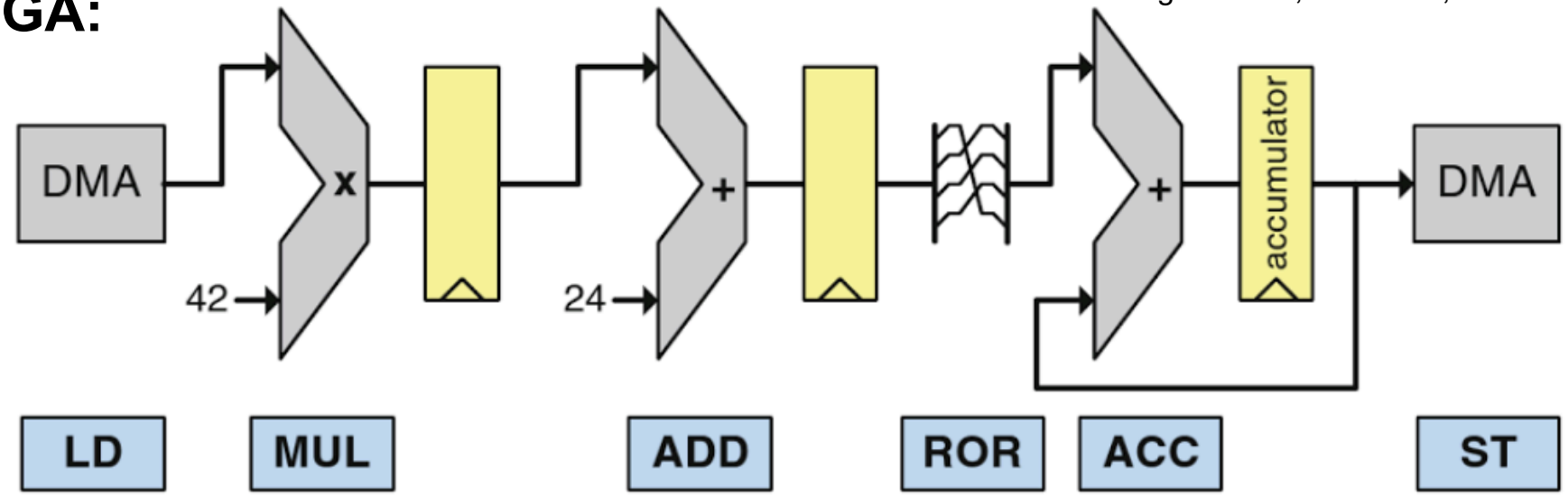
GPU:



How do we “mold this liquid silicon”?

FPGA:

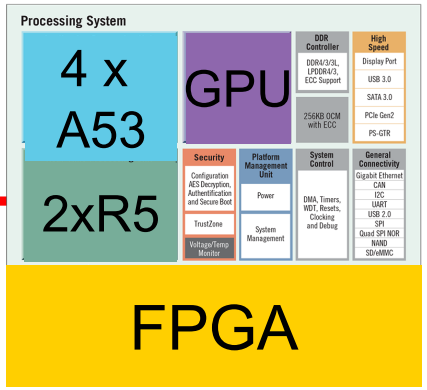
FPGAs for Software Programmers, Dirk Kock, Daniel Ziener



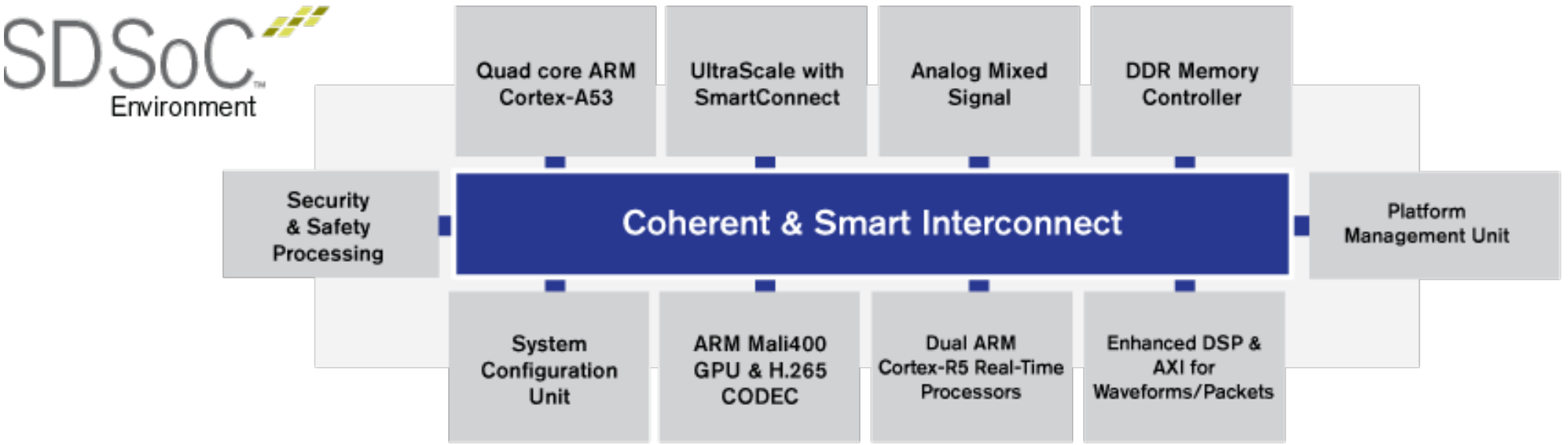
- ✓ No FETCH nor DECODE of instructions → already hardwired
- ✓ Data movement (Exec. Units ⇔ MEM) reduction → Power save
- ✓ Not constrained by a fixed ISA:
  - ✓ Bit-level parallelism; shift, bit mask, ...; variable precision arith.
- 👎 Less frequency (hundreds of MHz)
- 👎 In order execution (proposal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

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

Xilinx Zynq UltraScale+ MPSoC

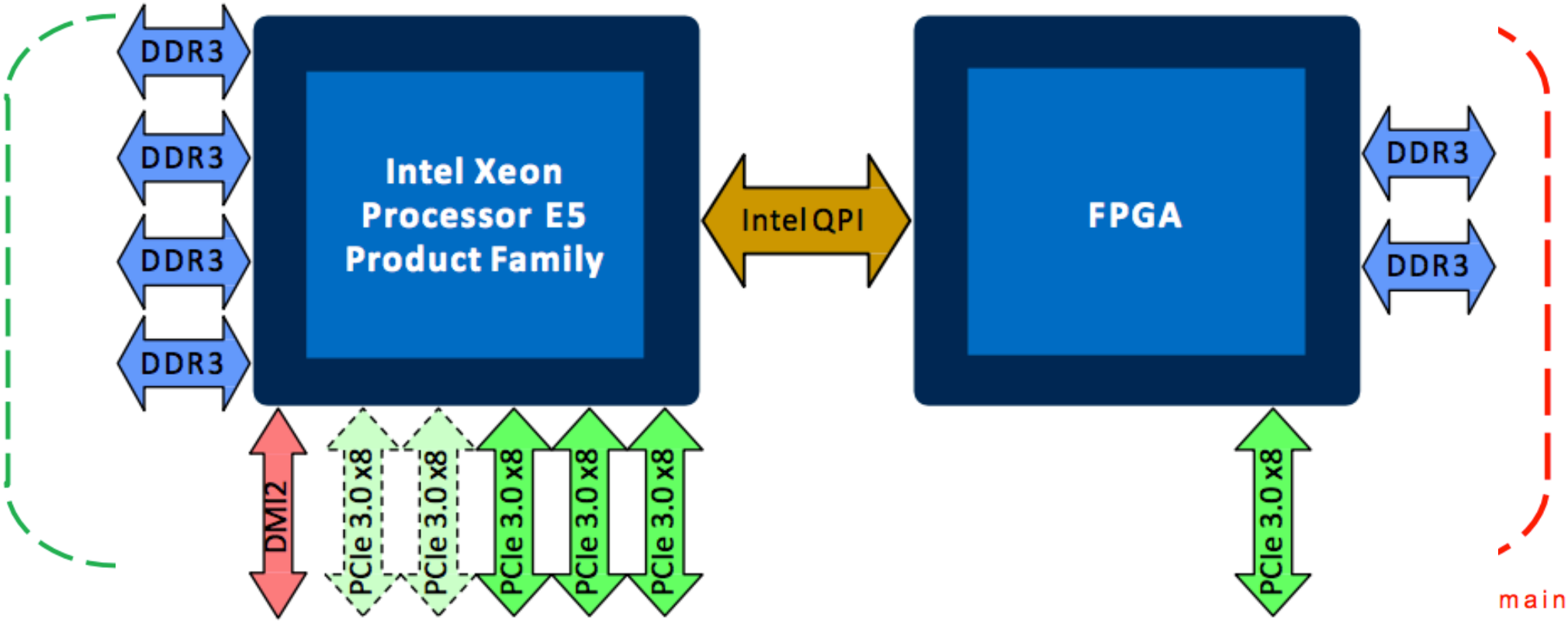


- Available:
  - SW coherence (cache flush) → Slow
  - HW coherence (MESI coherence protocol) → Fast. Ports:
    - Accelerator Coherency Port, ACP
    - Cache-coherent interconnect (CCI) ACE-Lite ports
      - Part of ARM® AMBA® AXI and ACE protocol specification



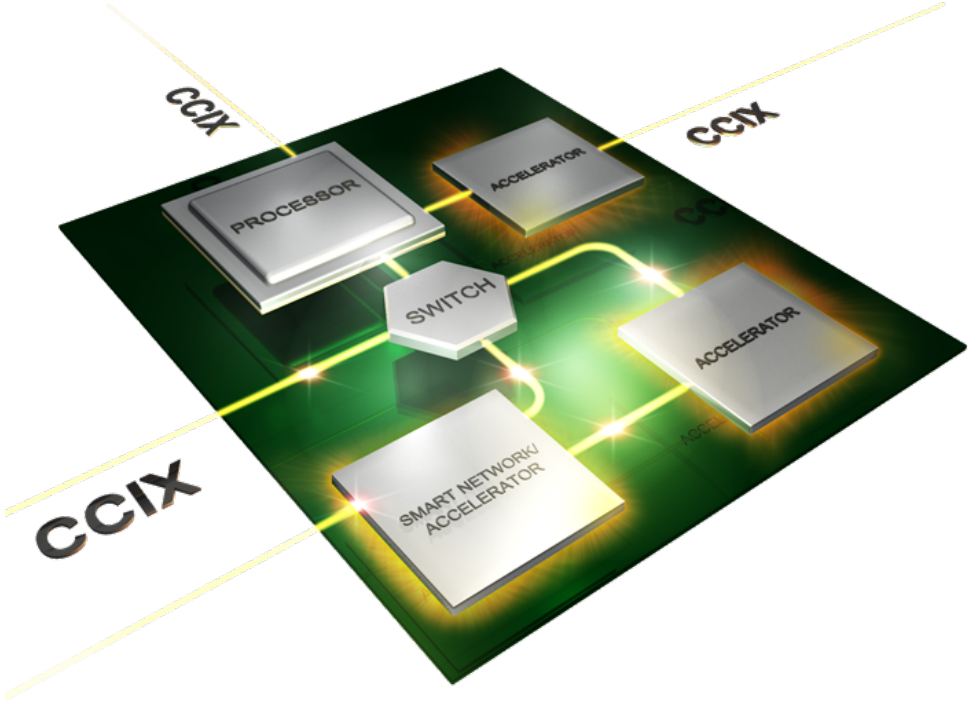
Intel Broadwell+Arria10

- 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
  - Protocol built for FPGAs, GPUs, network/storage adapters, ...
  - Already in Xilinx Virtex UltraScale+

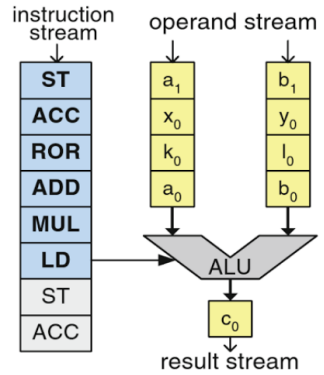


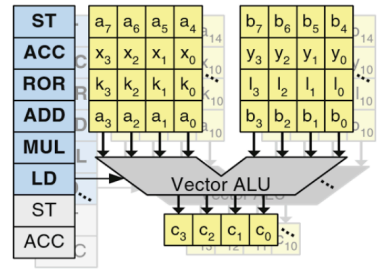
AMD · Amphenol · ARM · Arteris · avery design systems · BROADCOM · Bull atos technologies · cadence · CAVIUM · HUAWEI · IBM · IDT · KEYSIGHT TECHNOLOGIES · Mellanox TECHNOLOGIES · Micron · NETSPEED SYSTEMS · QUALCOMM · redhat · SYNOPSYS · tsmc · TELEDYNE LECROY Everywhereyoulook · XILINX ALL PROGRAMMABLE

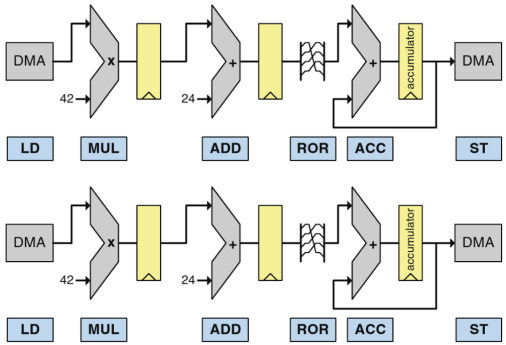
GEN Z · OpenCAPI™

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







Agenda

- Motivation
- Hardware: iGPUs and iFPGAs
- **Software**
  - Programming models for chips with iGPUs and iFPGAs
- Our heterogeneous templates based on TBB
  - parallel_for
  - parallel pipeline

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

HOW STANDARDS PROLIFERATE

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

Programming models for heterogeneous: GPUs

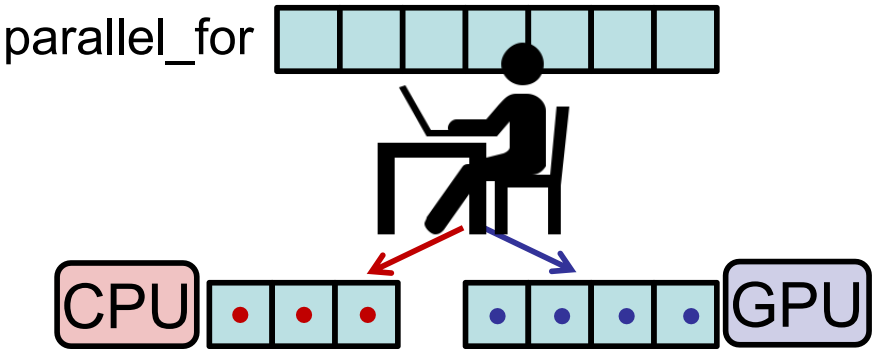
- Targeted at exploiting **one device** at a time
  - CUDA (NVIDIA)
  - **OpenCL** (Khronos Group Standard)
  - OpenACC (C, C++ or Fortran + Directives → OpenMP 4.0)
  - C++AMP (Windows' extension of C++. Recently HSA announced own ver.)
  - Sycl (Khronos Gruop's specification for "Single-source C++ 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

OpenMP4.0

Matlab

ROCm

R

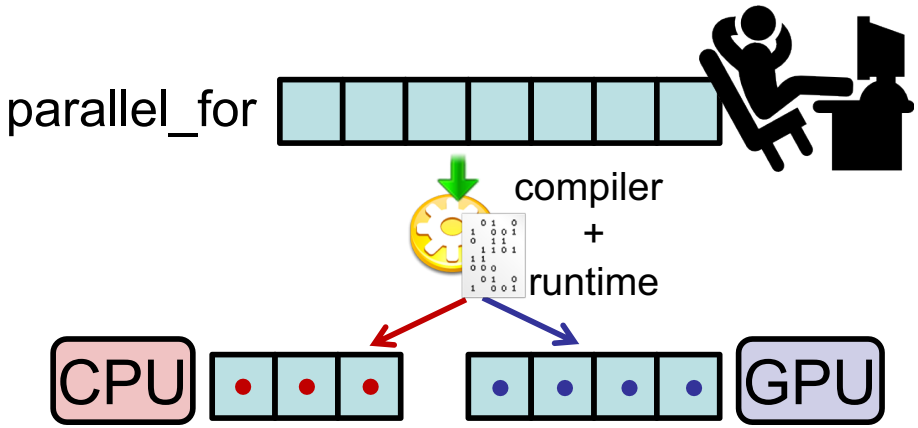
JavaScript

Numba Python

RenderScript

ParallDroid

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



OmpSs

Qilin

StarPU

XKaapi

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]`)

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%)

Intel Concord

- C++ heterogeneous programming framework

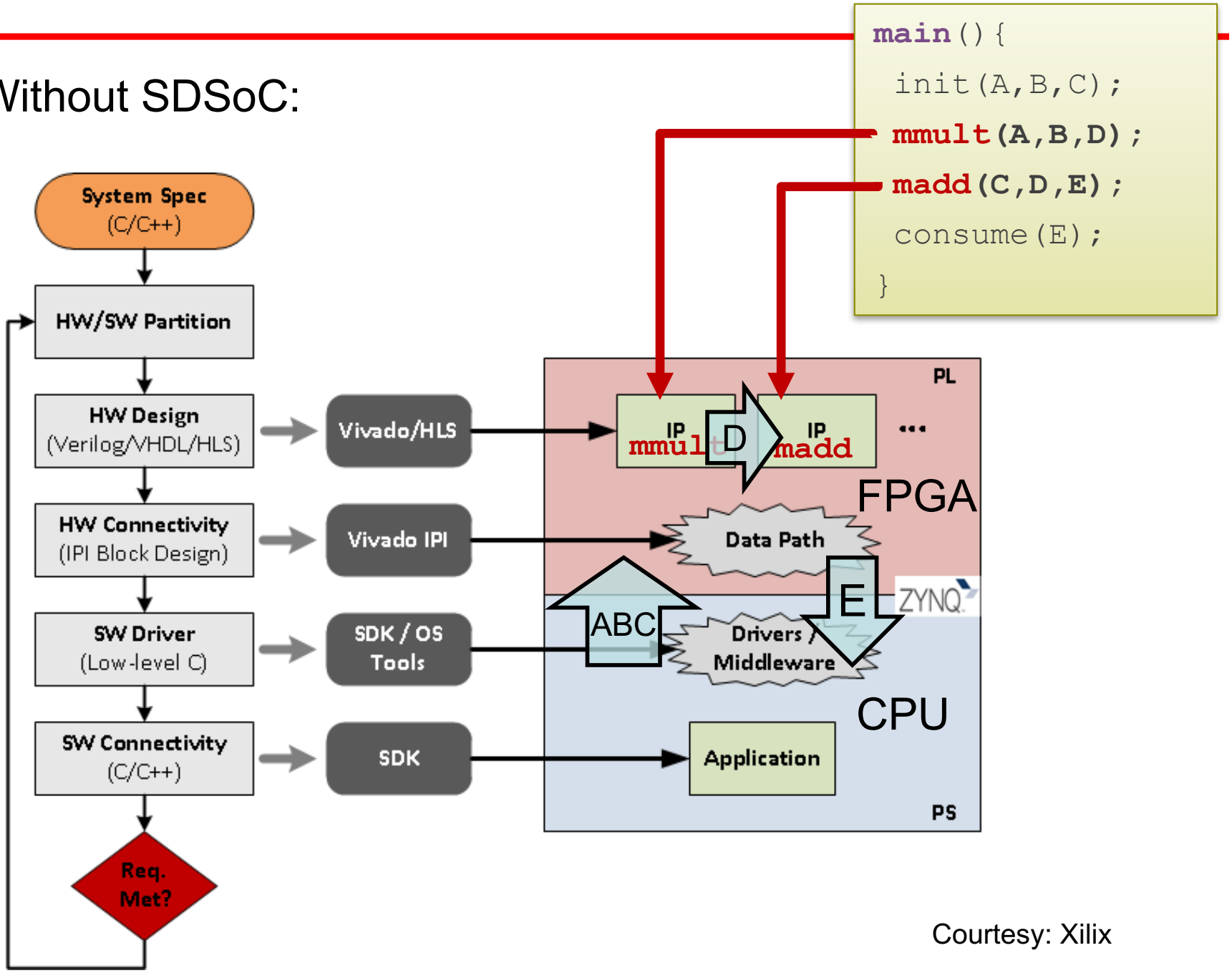
```
class Foo {
   int *a, *b, *c;
   void operator(int i) { c[i] = a[i]+b[i]; }

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

Xilinx SDSoC

- Without SDSoC:



Courtesy: Xilix

Xilinx SDSoC

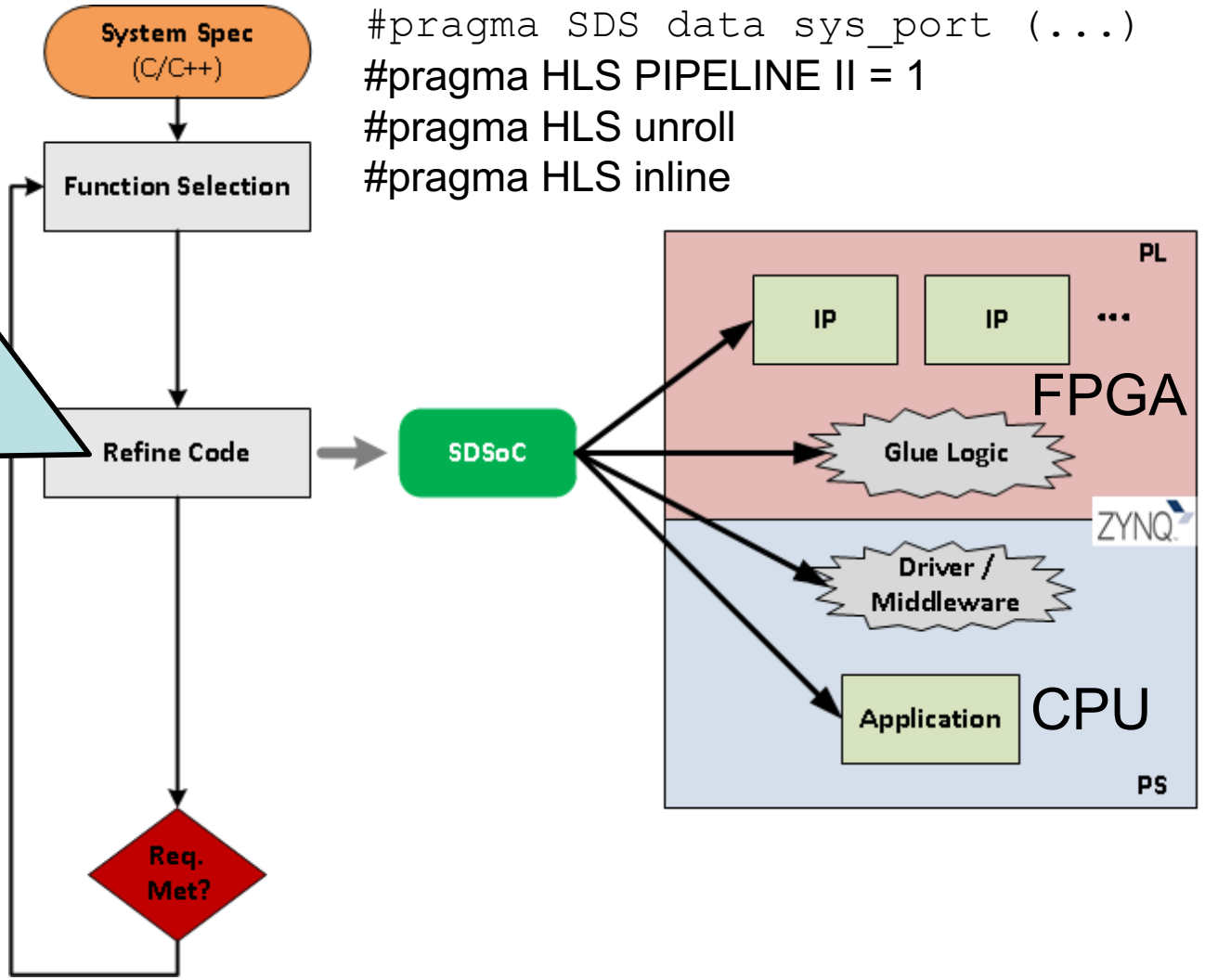
- With SDSoC:
  - Eclipse-based SDK

```
#pragma SDS data copy(Array)
#pragma SDS zero_copy(Array)
#pragma SDS data access_pattern
#pragma SDS data sys_port (...)
```

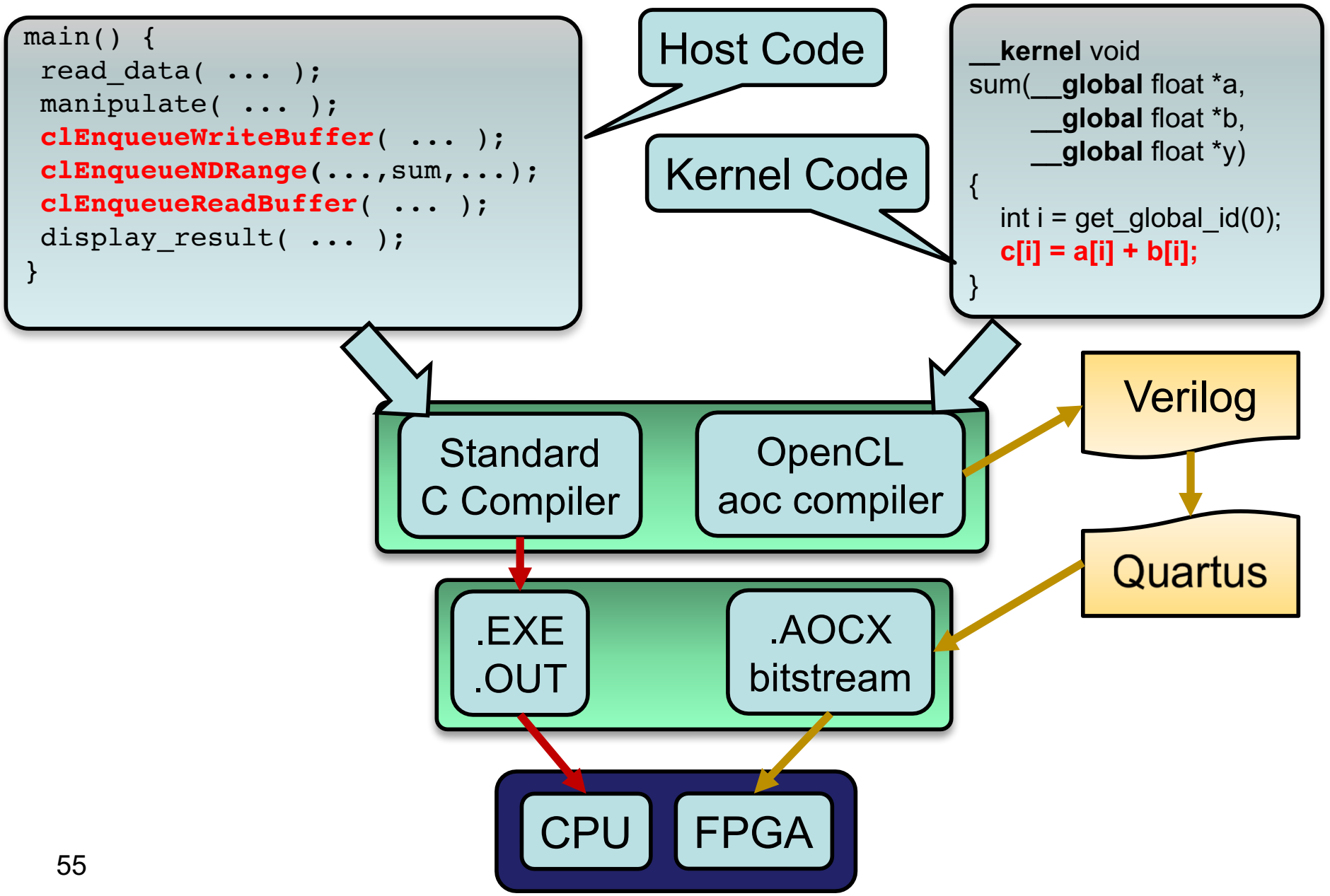
#pragma HLS PIPELINE II = 1
#pragma HLS unroll
#pragma HLS inline

Refine Code:

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

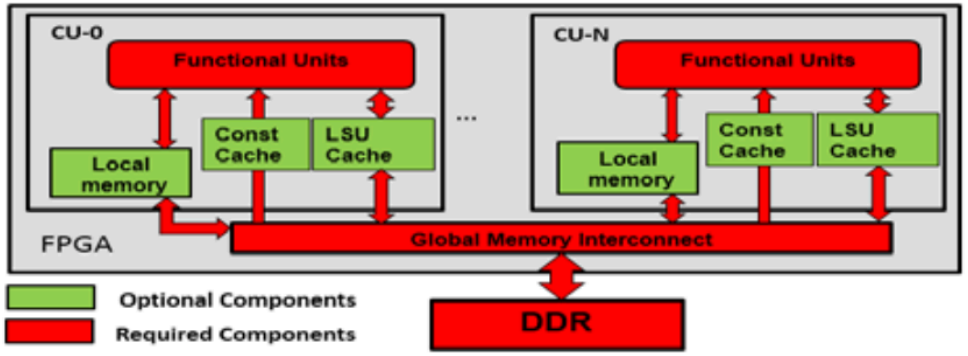


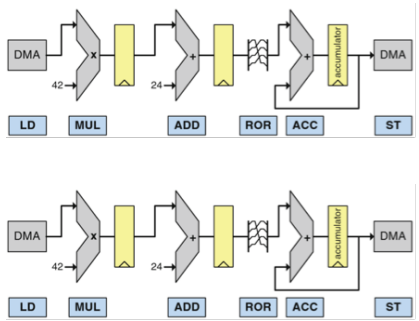
Altera OpenCL



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 ⬇

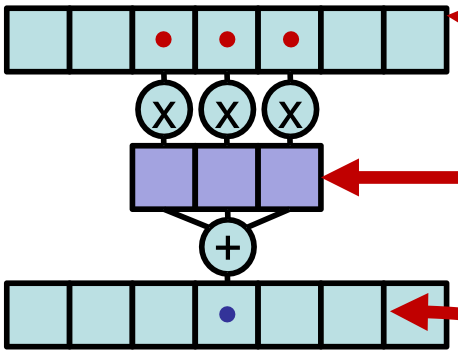




Shift Register Pattern

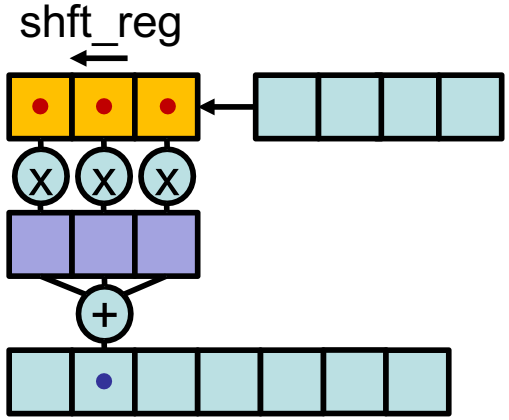
- Example: 1D stencil computation

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 !!

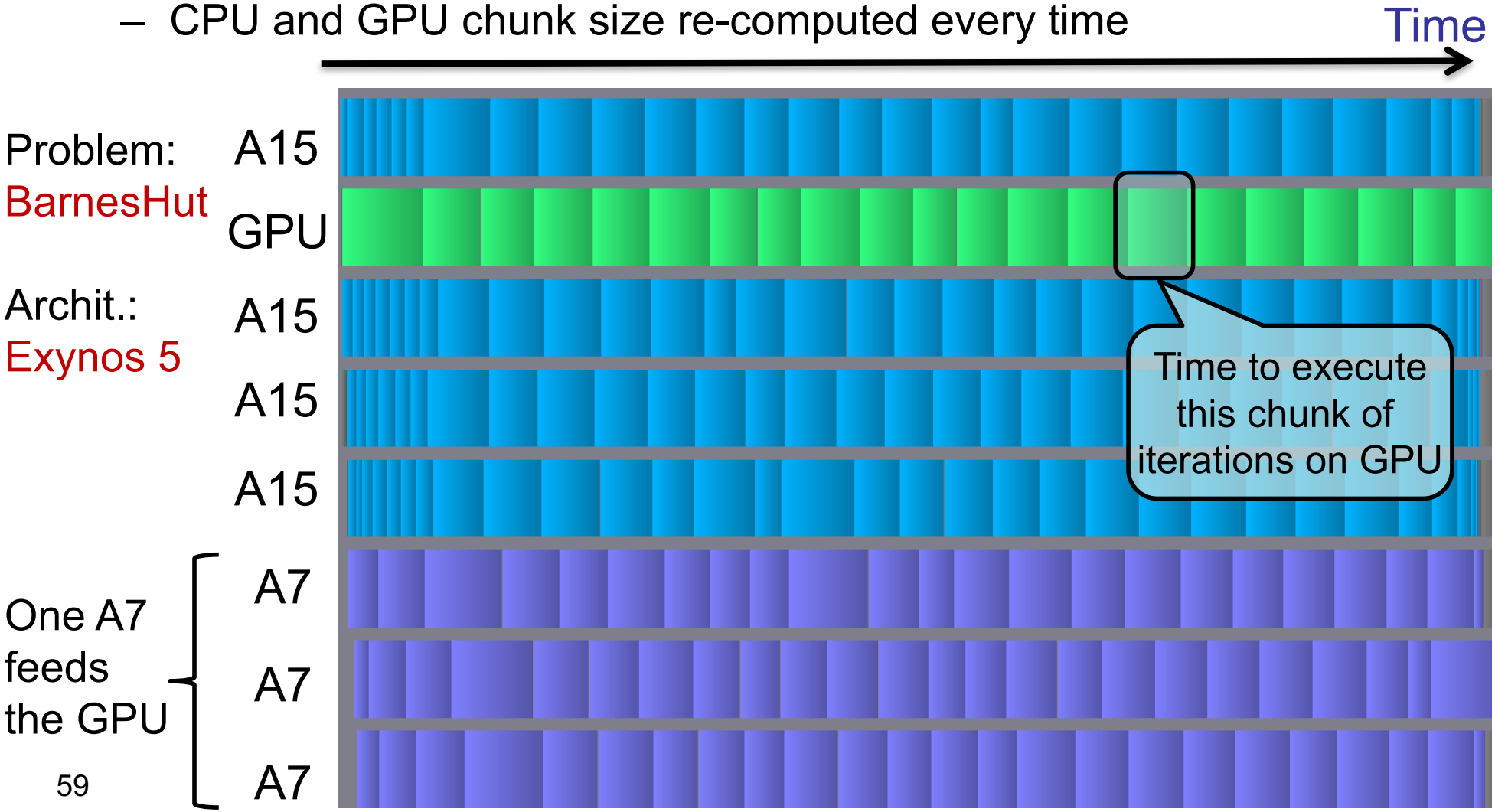
64x faster than Naive (17 elem. filter)
“Tuning Stencil Codes in OpenCL for FPGAs”, ICCD’16

Agenda

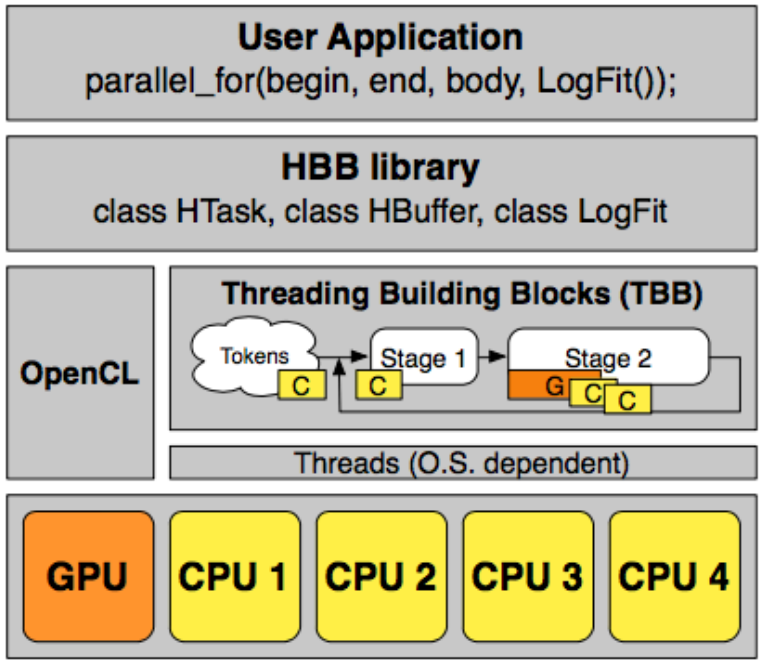
- Motivation
- Hardware: iGPUs and iFPGAs
- Software
  - Programming models for chips with iGPUs and iFPGAs
- **Our heterogeneous templates based on TBB**
  - parallel_for
  - parallel pipeline

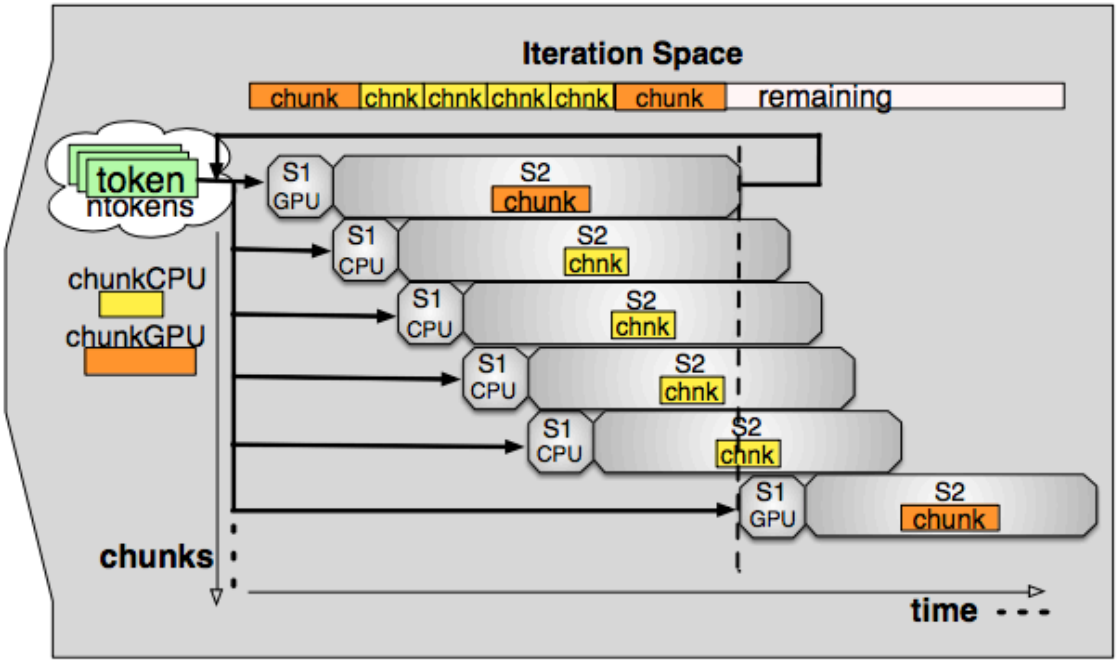
Our work: parallel_for

- Heterogeneous parallel for based on TBB
  - Dynamic scheduling / Adaptive partitioning for irregular codes
  - CPU and GPU chunk size re-computed every time



New scheduler and API





New API

```
1  #include ''parallel_for.h''
2  using namespace hbb;
3  int main(int argc, char* argv[]){
4    // Start task scheduler
5    HInit HInit(numcpus, true);
6    ...
7    KernelInfo k(kernelFile, kernelName);
8    HBuffer<int> * a = new HBuffer<int>(N,USE_ZCB);
9    Body body(k, a);
10   ...
11   parallel_for(begin, end, body, new LogFit());
12   ...
13 }
```

Use the GPU

Exploit Zero-Copy-Buffer

Scheduler class

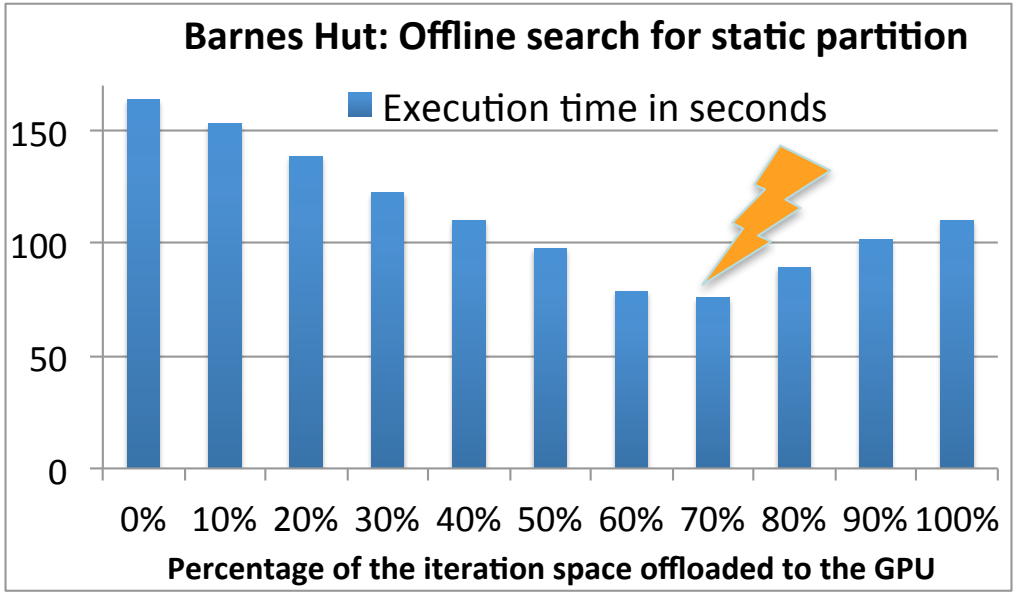
```
1  #include ''HTask.h''
2  #include ''HBuffer.h''
3  using namespace hbb;
4
5  class Body : public HTask{
6     HBuffer<int> * buf_a;
7  public:
8    Body(KernelInfo k, HBuffer * buf_a) : HTask(k){...}
9    void operatorCPU(int begin, int end) {
10      int *a = buf_a->getHostPtr(BUF_RW);
11      for(i=begin; i!=end; i++) a[i] = a[i] + 1;
12   }
13   void operatorGPU() (int begin, int end){
14      //Setting kernel arguments
15      setKernelArg(0, sizeof(int), &begin);
16      setkernelArg(1, sizeof(int), &end);
17      setKernelBuf(2, sizeof(BUF), buf_a->getDevicePtr(BUF_RW));
18      //Launching kernel
19      launchKernel(begin, end);
20   }
21 };
```

Related work: Qilin

- Static Approach: **Bulk-Oracle**

  Offline search (11 executions)

  – Work partitioning between CPU and GPU

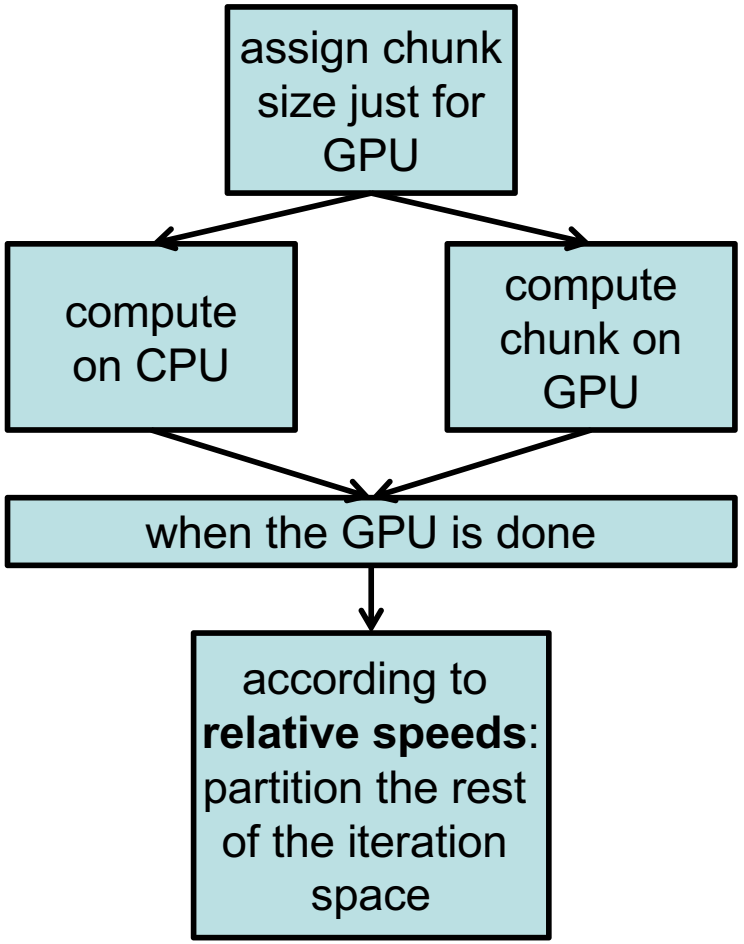


  – One big chunk with 70% of the iterations: on the GPU

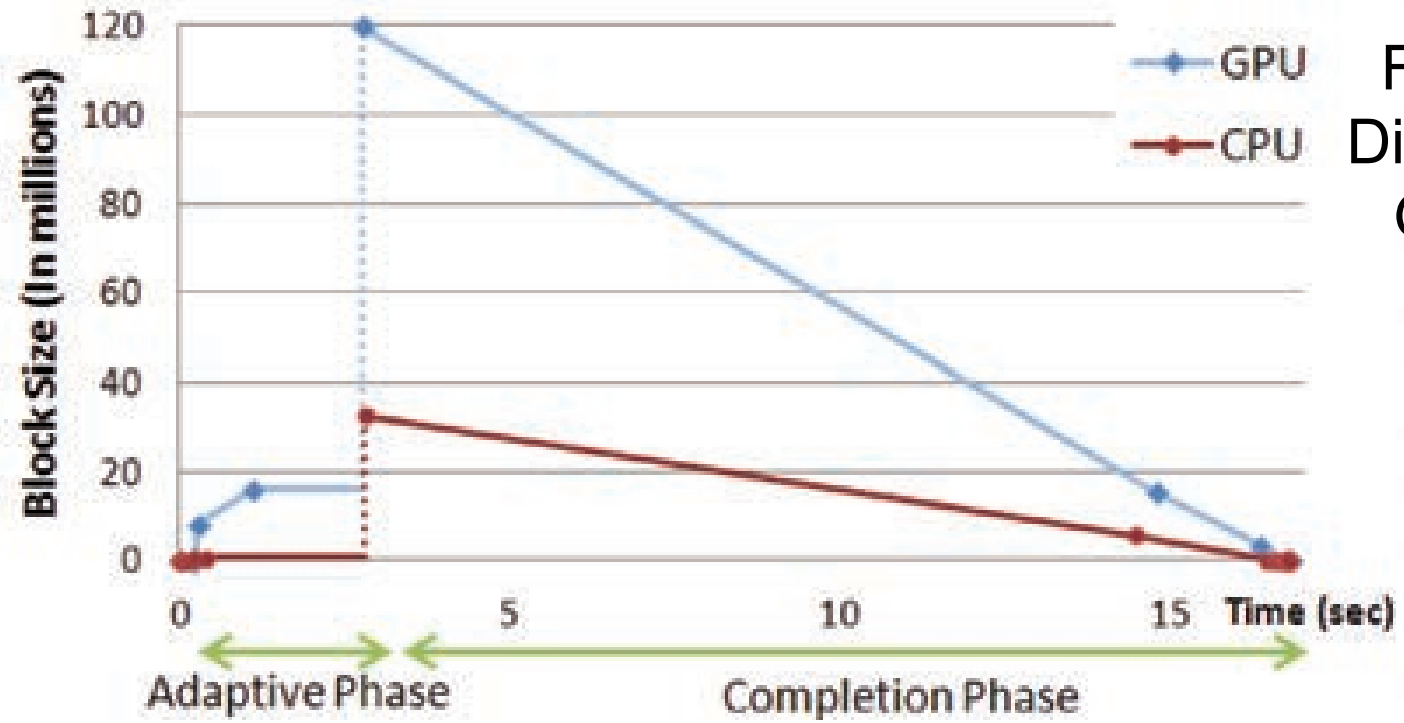
  – The remaining 30% processed on the cores

Related work: Intel Concord

- Papers:
  - Rashid Kaleem et al. Adaptive heterogeneous scheduling on integrated GPUs. **PACT 2014**.
  - Naila Farooqui et al. Affinity-aware work-stealing for integrated CPU-GPU processors. **PPoPP '16** (poster).



Related Work: HDSS

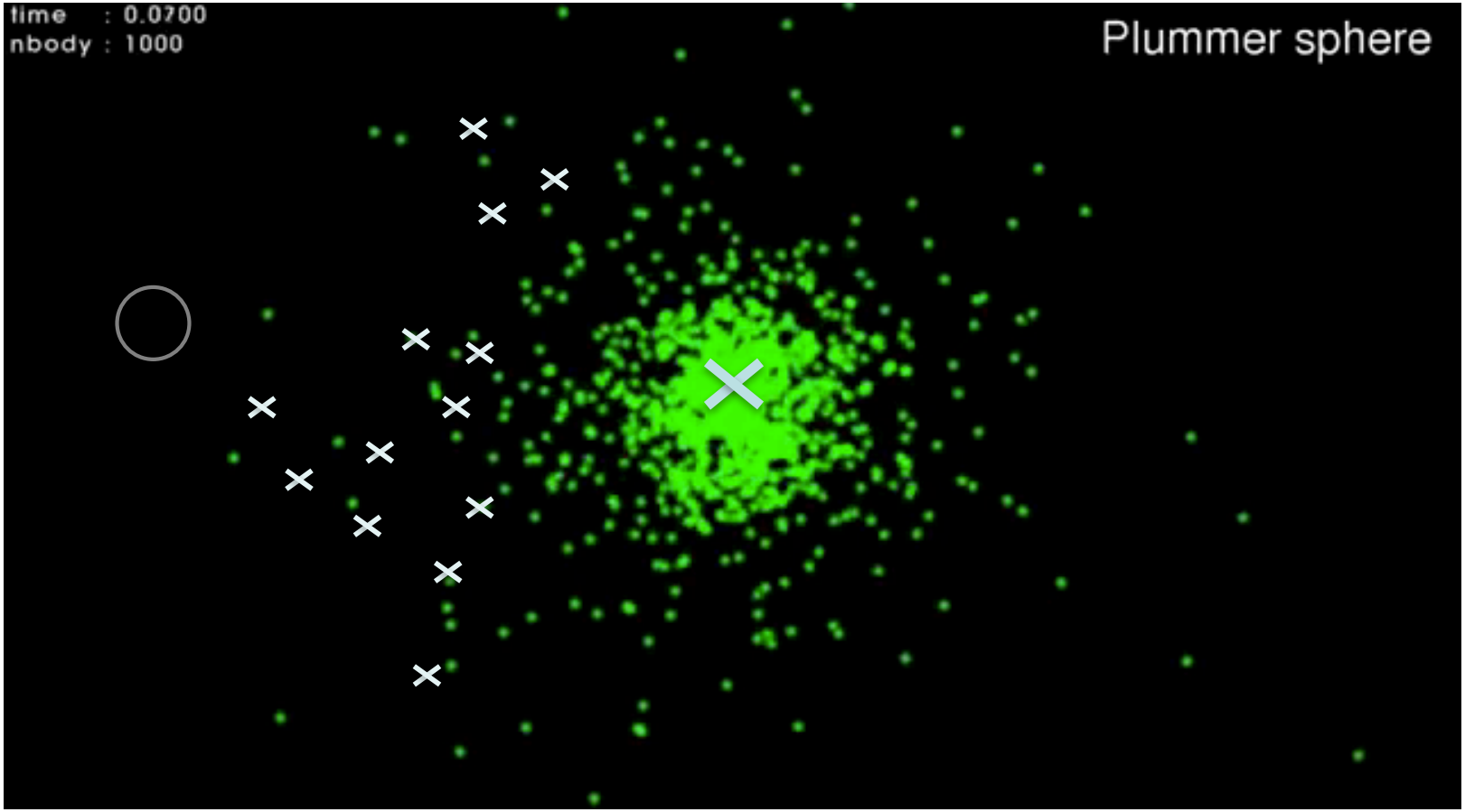


Fermi Discrete GPU

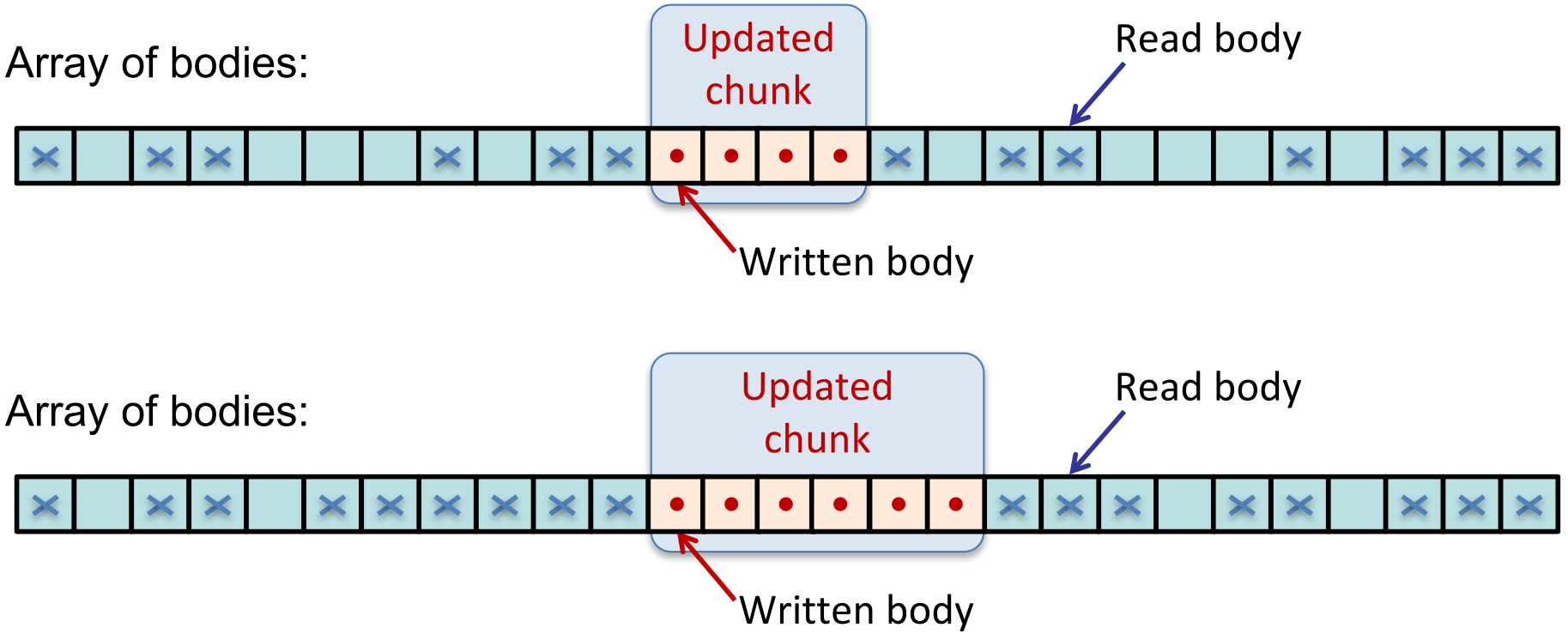
- Belviranli et al, A Dynamic Self-Scheduling Scheme for Heterogeneous Multiprocessor Architectures, **TACO 2013**

Debunking big chunks

- Example: irregular Barnes-Hut benchmark



Debunkign big chunks

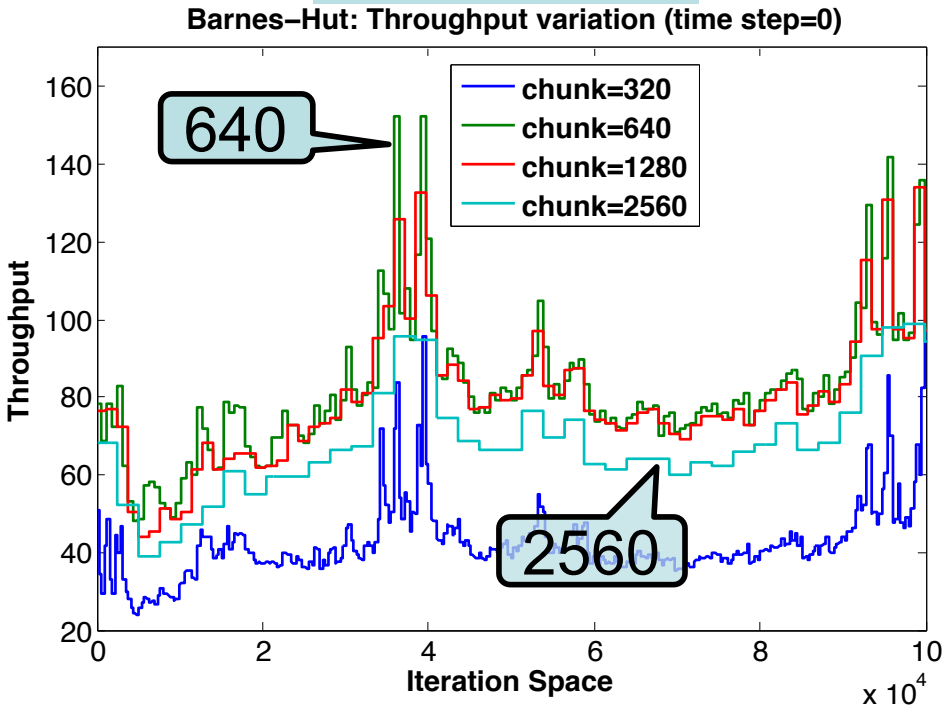


A larger chunk causes more memory traffic

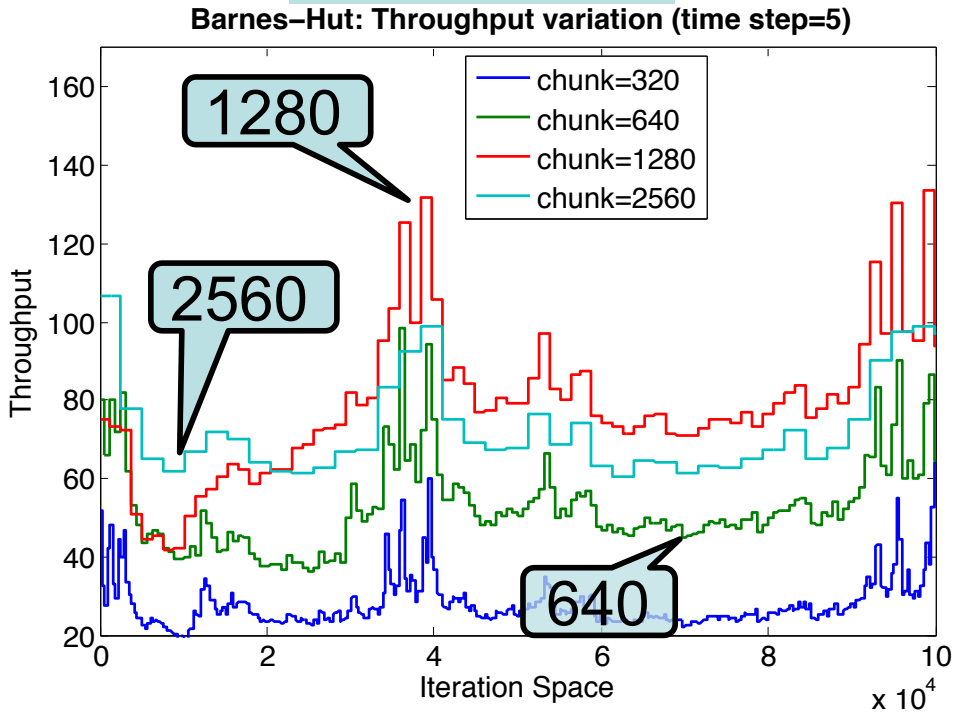
Debunking big chunks

- BarnesHut: GPU throughput along the iteration space
  - For two different time-steps
  - Different GPU chunk-sizes

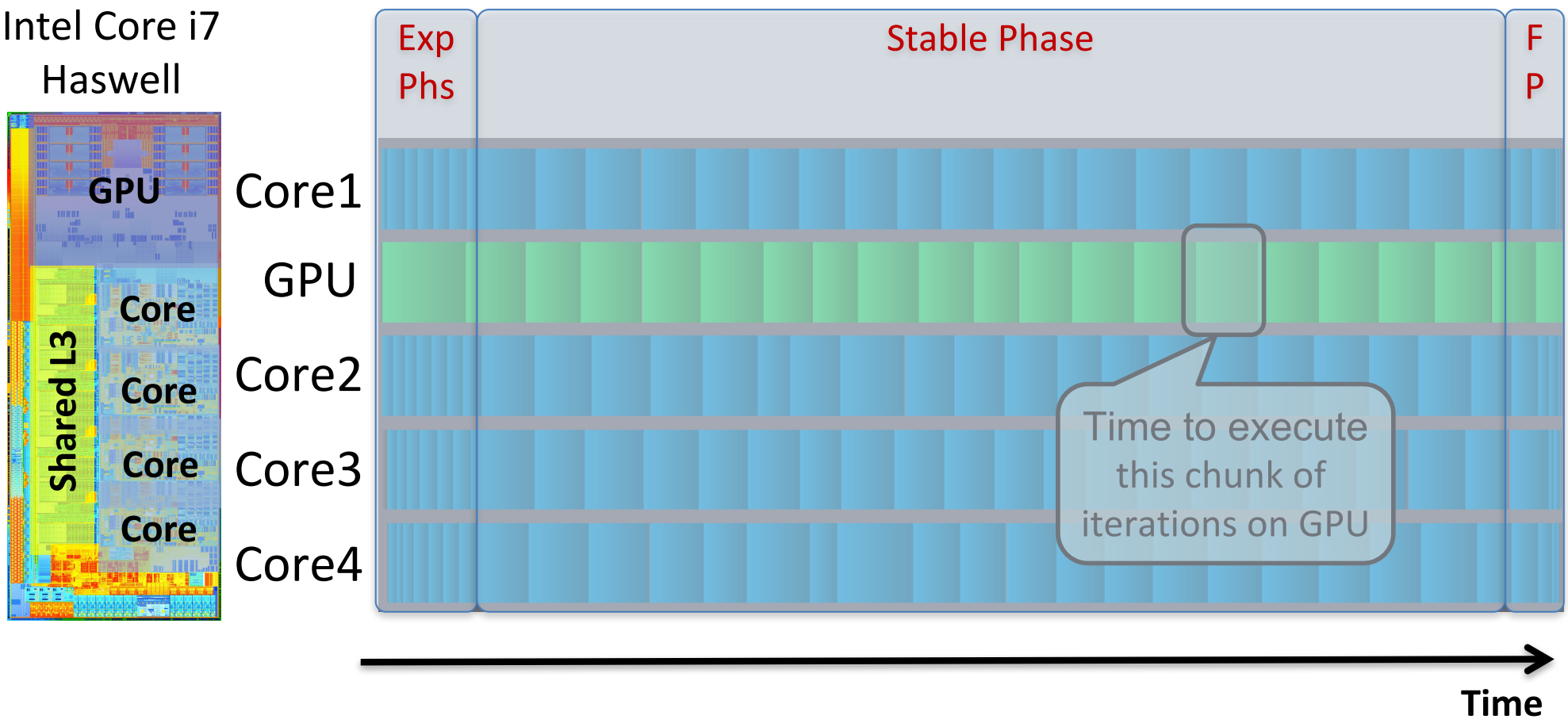
Time step=0



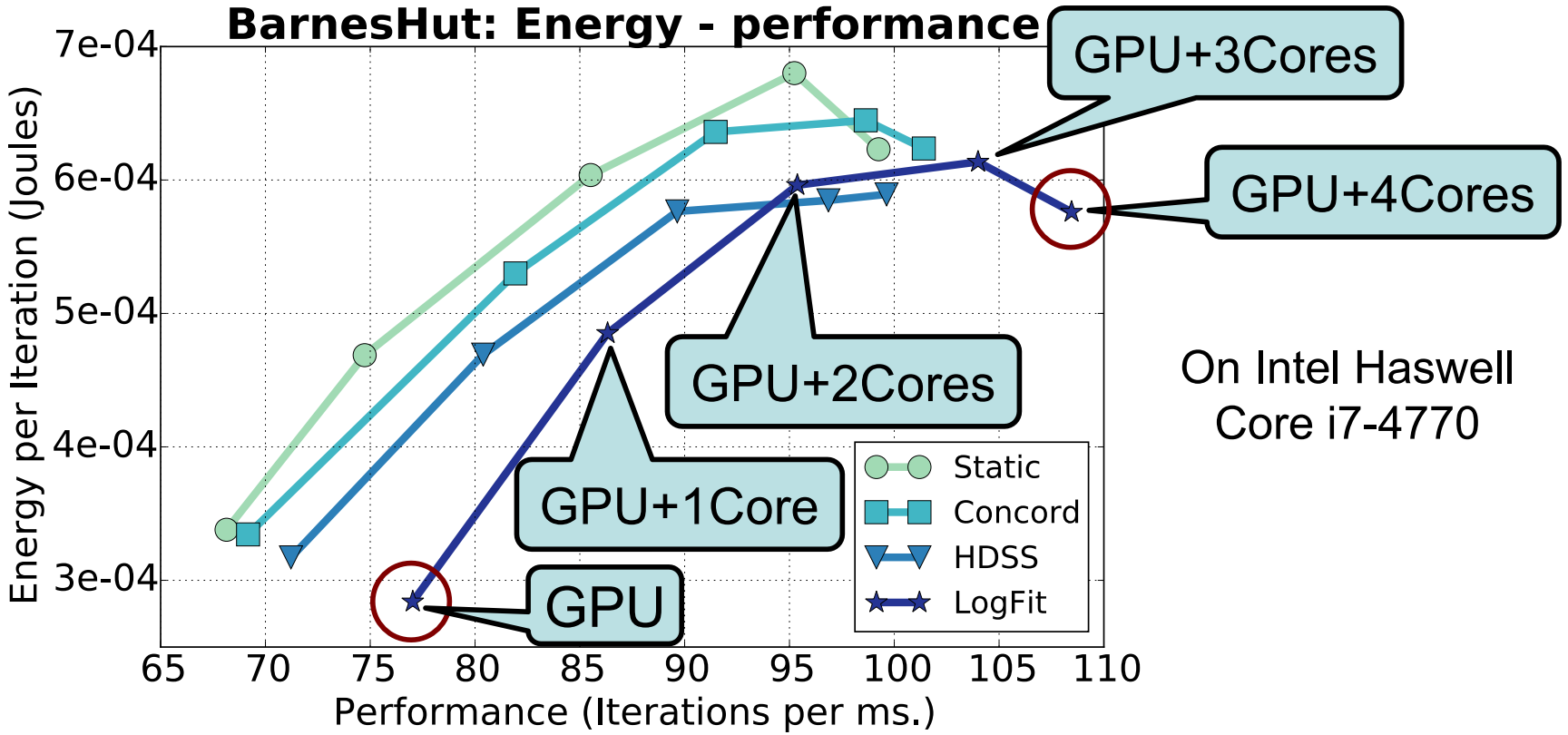
Time step=5



LogFit main idea



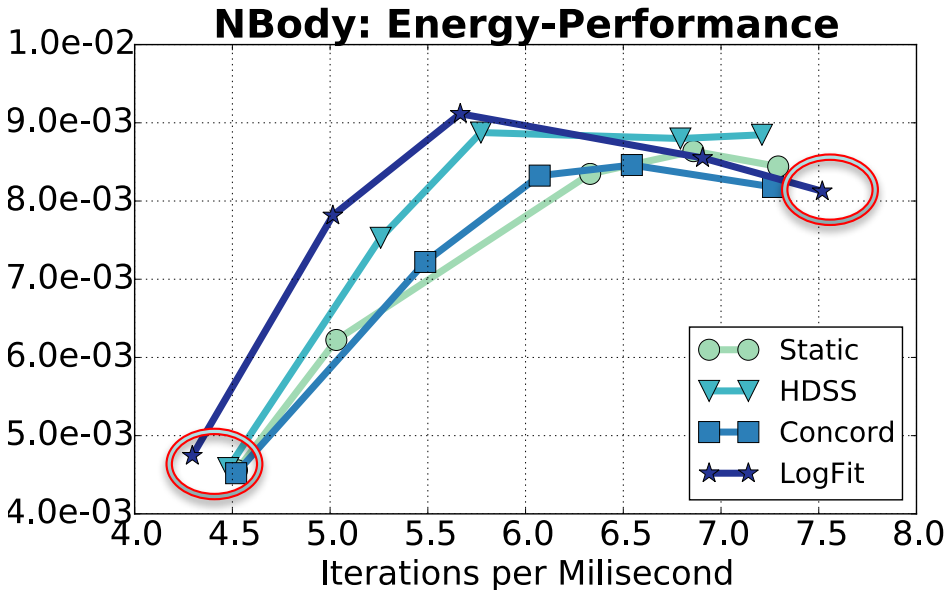
Parallel_for: Performance per joule

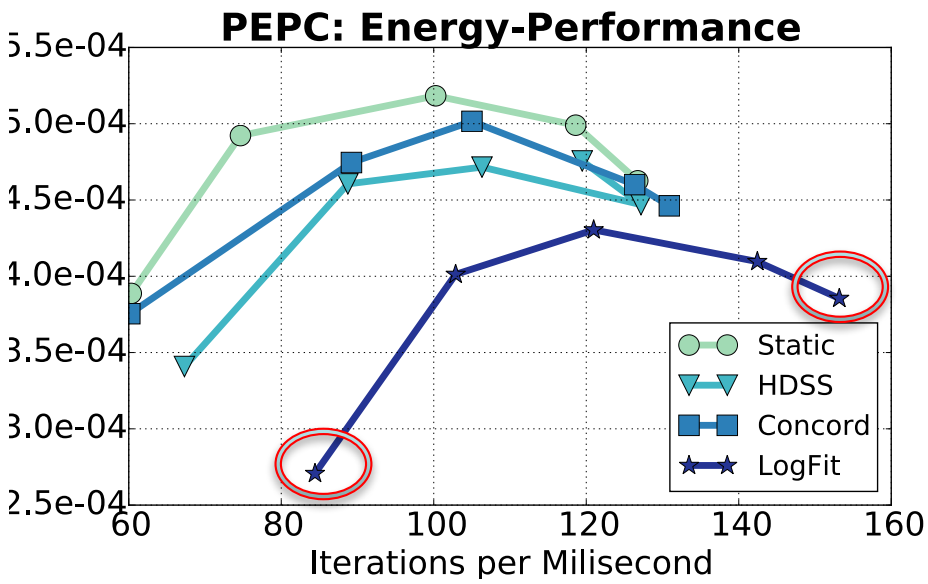


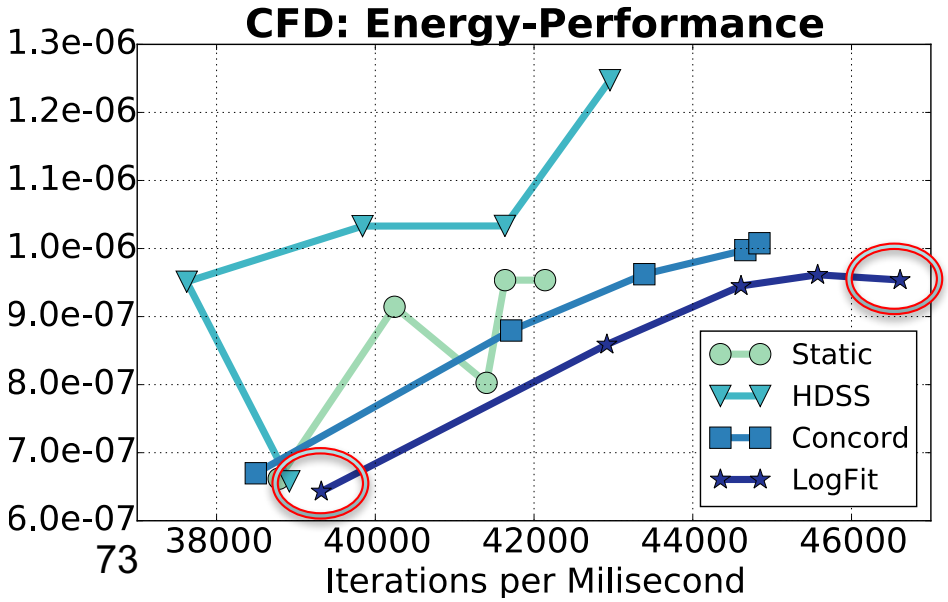
- **Static**: Oracle-Like static partition of the work based on profiling
- **Concord**: Intel approach: GPU size provided by user once
- **HDSS**: Belviranli et al. approach: GPU size computed once
- **LogFit**: our work

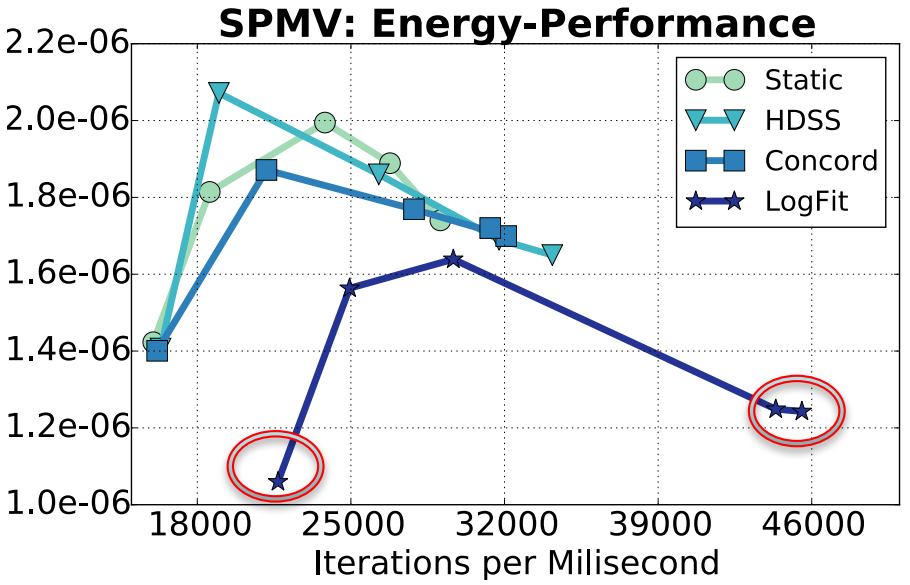
Antonio Vilches et al. *Adaptive Partitioning for Irregular Applications on Heterogeneous CPU-GPU Chips*, **Procedia Computer Science**, 2015.

LogFit: better for irregular benchmarks



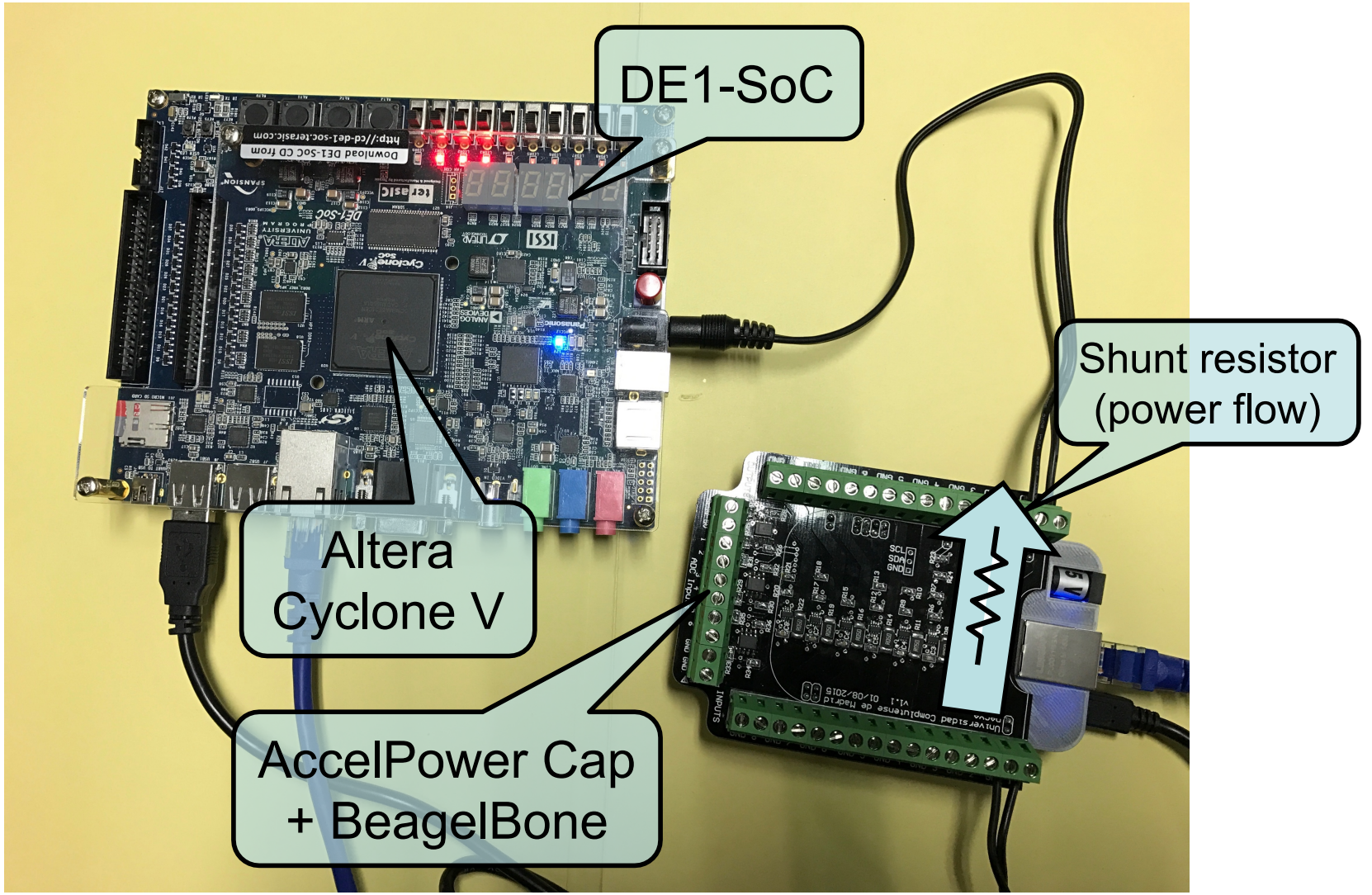




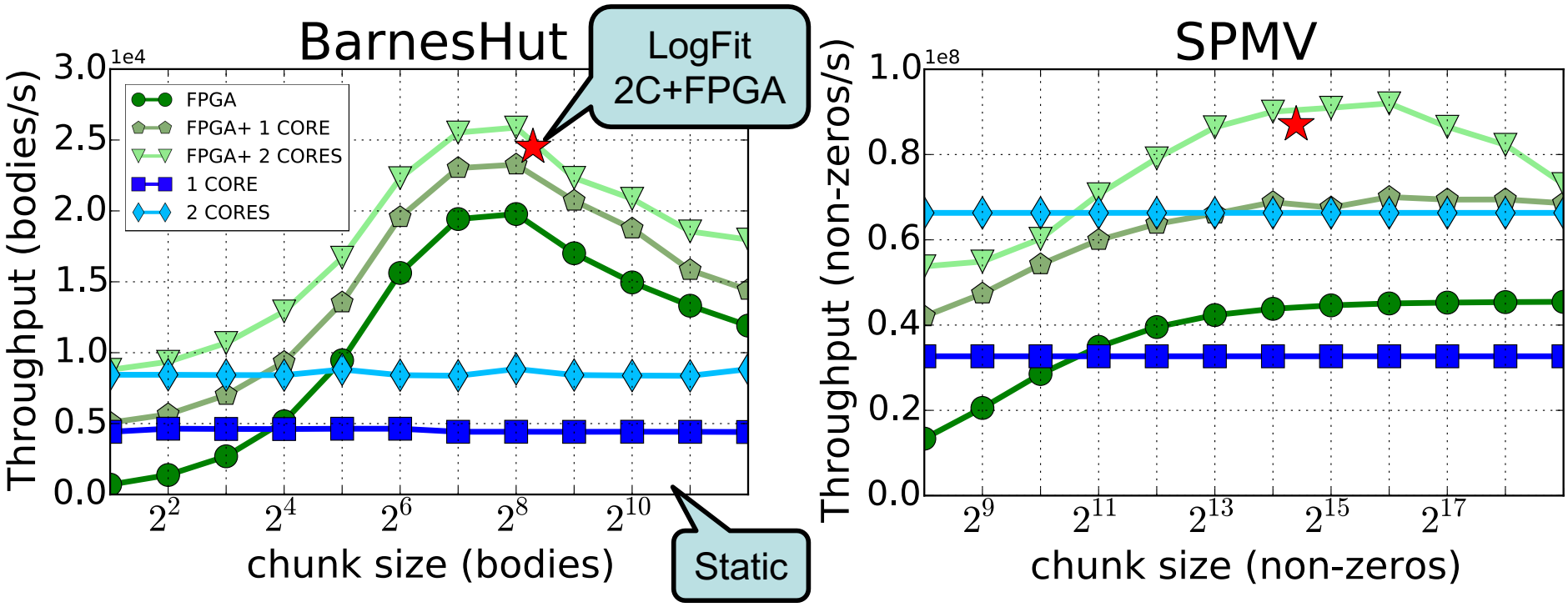


LogFit working on FPGA

- Energy probe: AccelPower Cap (Luis Piñuel, U. Complut.)



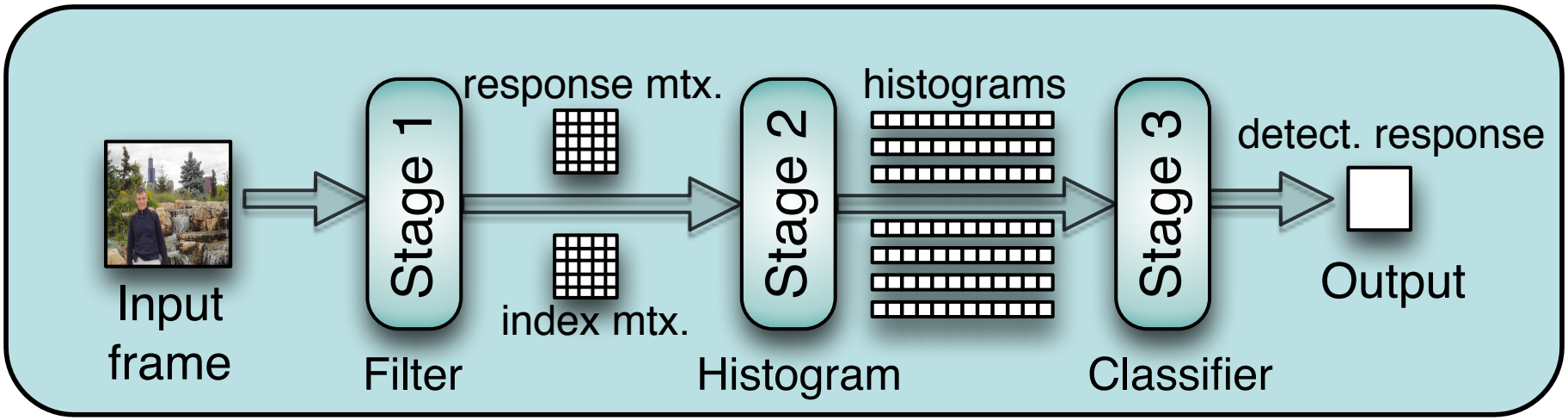
LogFit working on FPGA



- Terasic DE1: Altera Cyclone V (FPGA+2 Cortex A9)
- Energy probe: AccelPower Cap (Luis Piñuel, U. Complut.)

Our work: pipeline

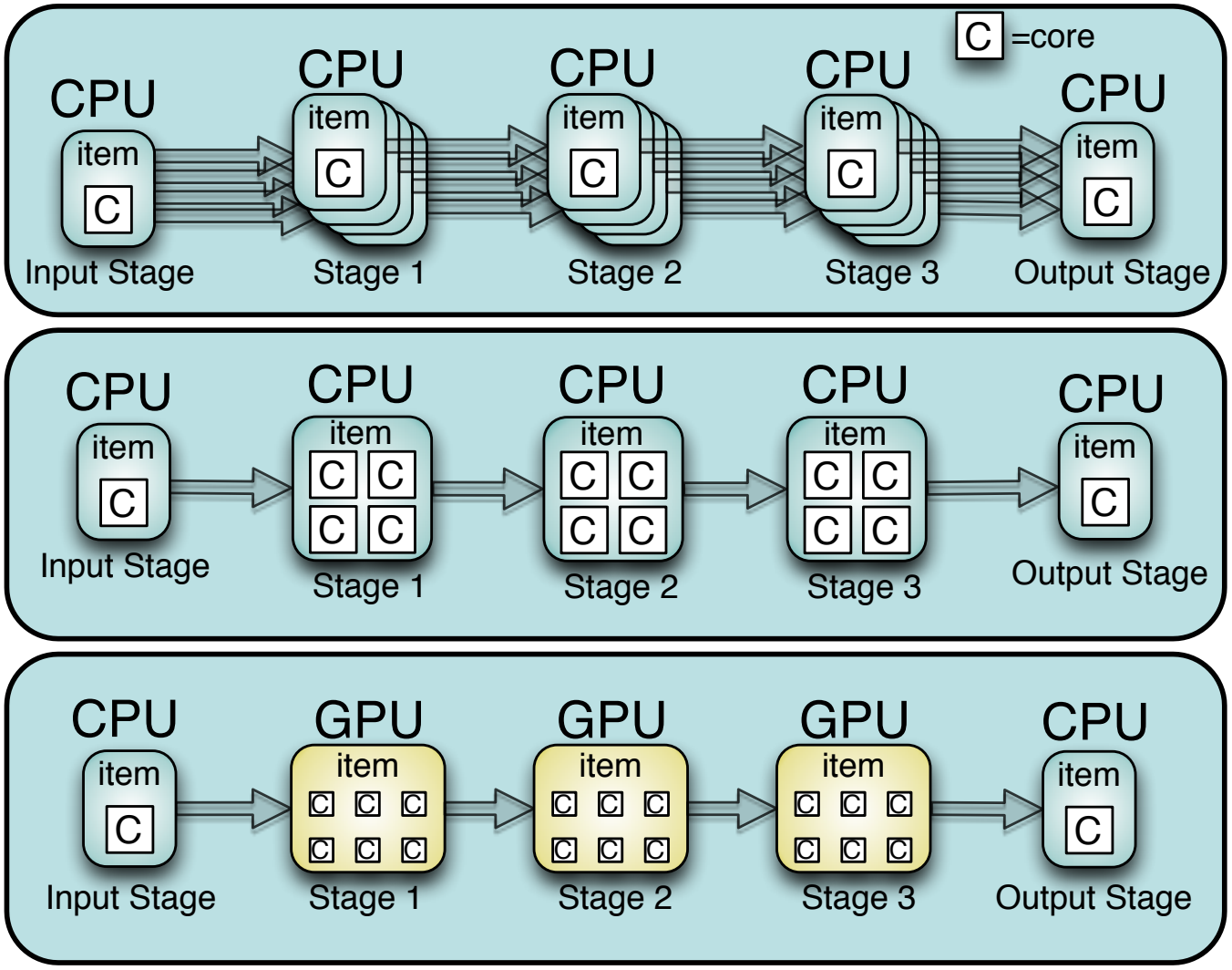
- ViVid, an object detection application
- Contains three main kernels that form a pipeline



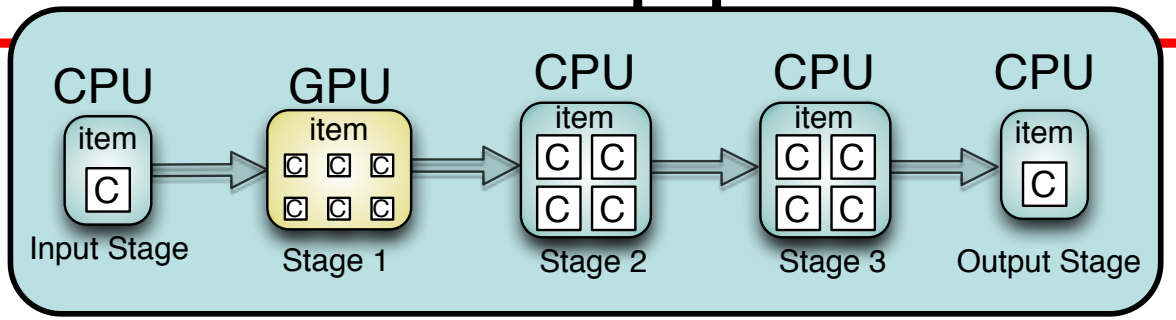
- Would like to answer the following questions:
  - **Granularity**: coarse or fine grained parallelism?
  - **Mapping of stages**: where do we run them (CPU/GPU)?
  - **Number of cores**: how many of them when running on CPU?
  - **Optimum**: what metric do we optimize (time, energy, both)?

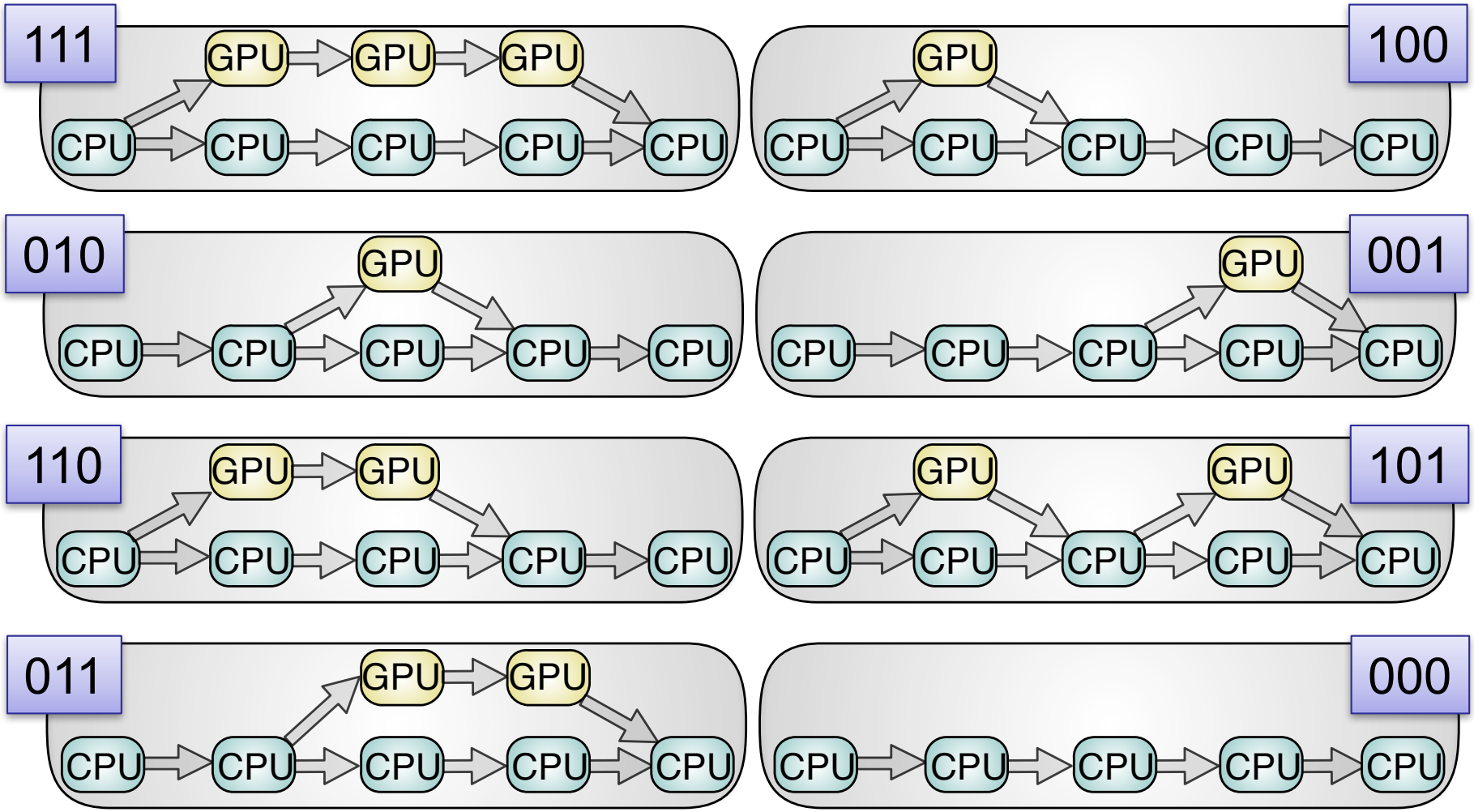
Granularity

- Coarse grain:
  - CG
- Medium grain:
  - MG
- Fine grain:
  - Fine grain also in the CPU via AVX intrinsics



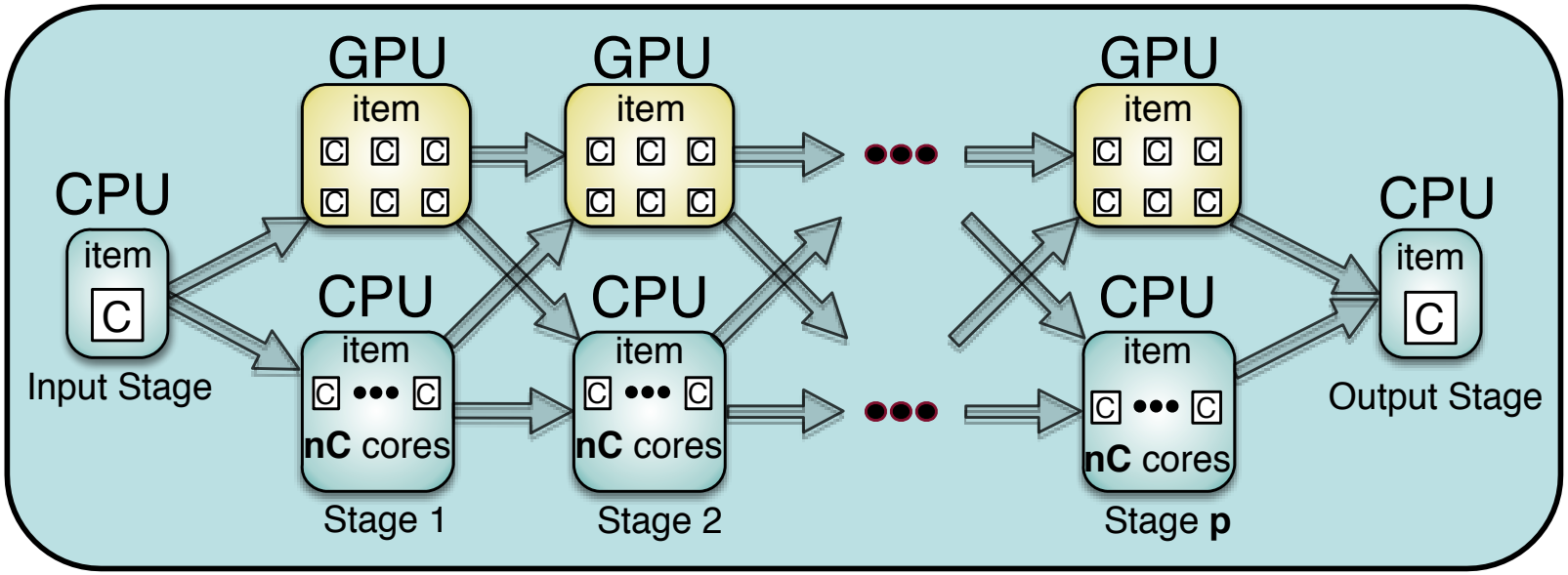
Our work: pipeline





Accounting for all alternatives

- In general: **nC** CPU cores, **1** GPU and **p** pipeline stages



$$\#\ \text{alternatives} = 2^{p} \times (nC + 2)$$

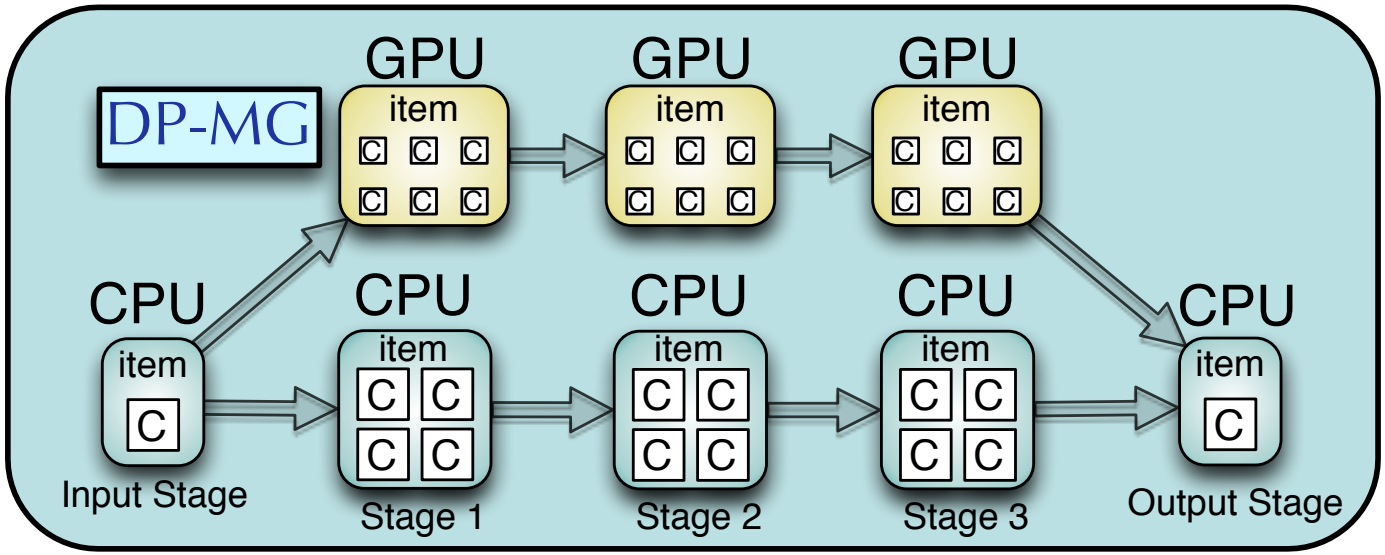
- For Rodinia's SRAD benchmark (p=6,nC=4) → **384** alternatives

Mapping streaming applications on commodity multi-CPU and GPU on-chip processors, **IEEE Tran. Parallel and Distributed Systems**, 2016.

Framework and Model

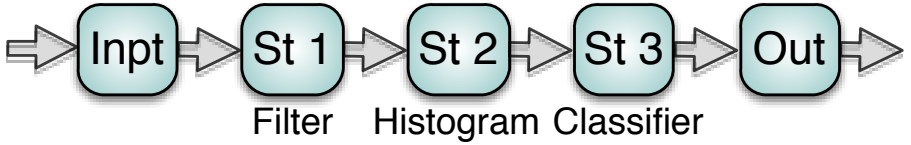
- Key idea:
  1. Run only on GPU
  2. Run only on CPU

  } collect λ and E (homogeneous values)
  3. Analytically extrapolate for heterogeneous execution
  4. Find out the best configuration → RUN

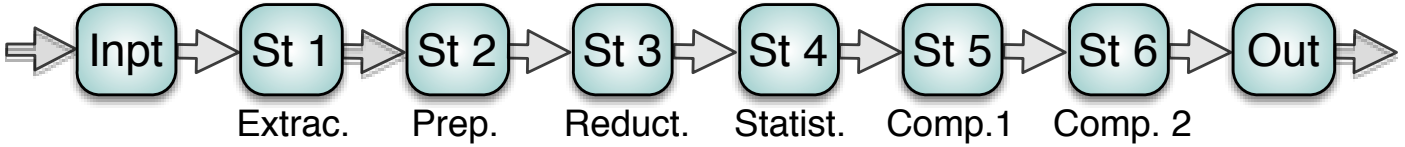


Environmental Setup: Benchmarks

- Four Benchmarks
  - ViVid (Low and High Definition inputs)

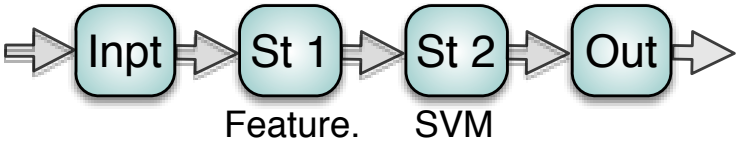


  - SRAD

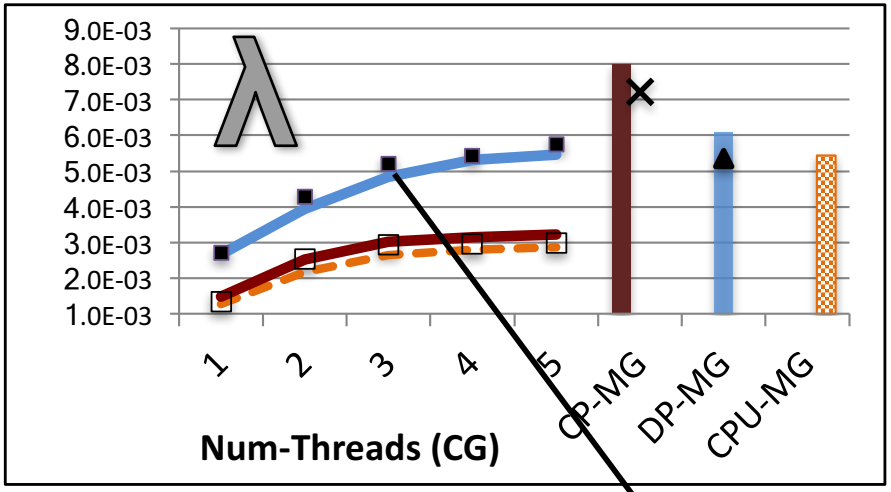


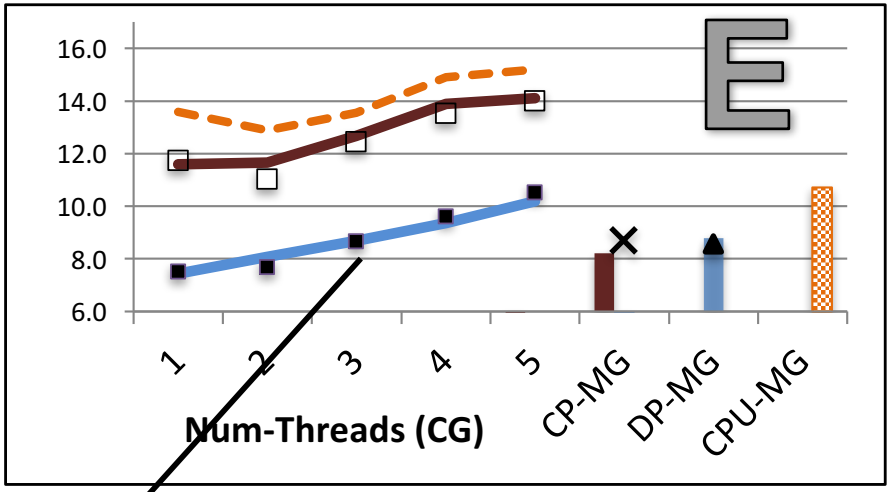
  - Tracking

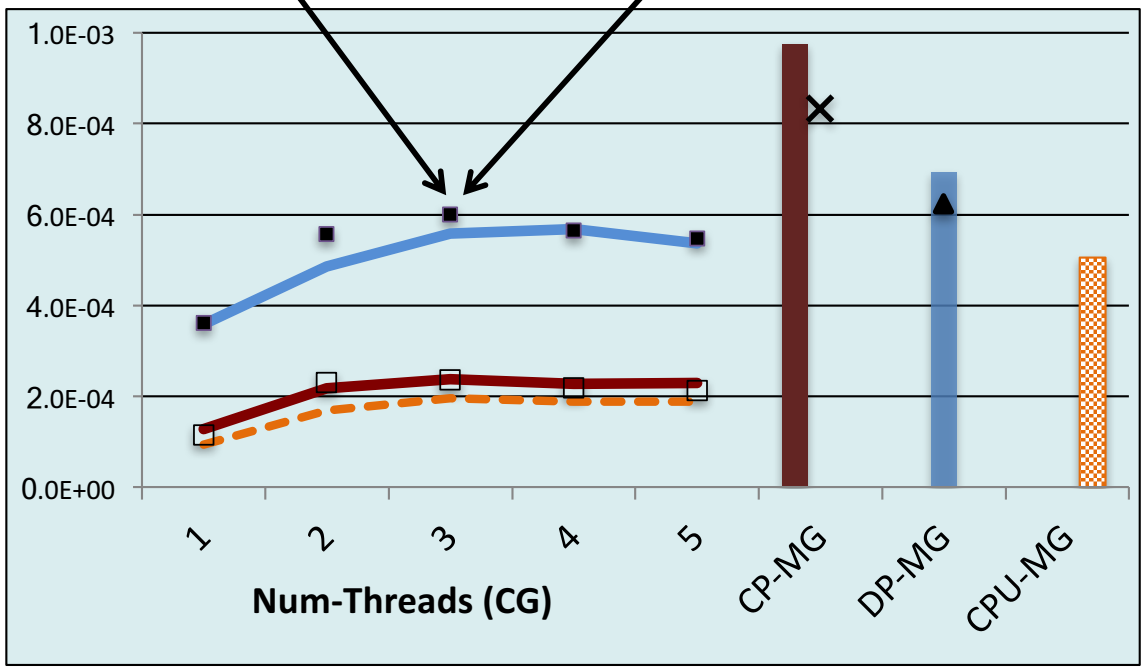
  - Scene Recognition



Does Higher Throughput imply Lower Energy?



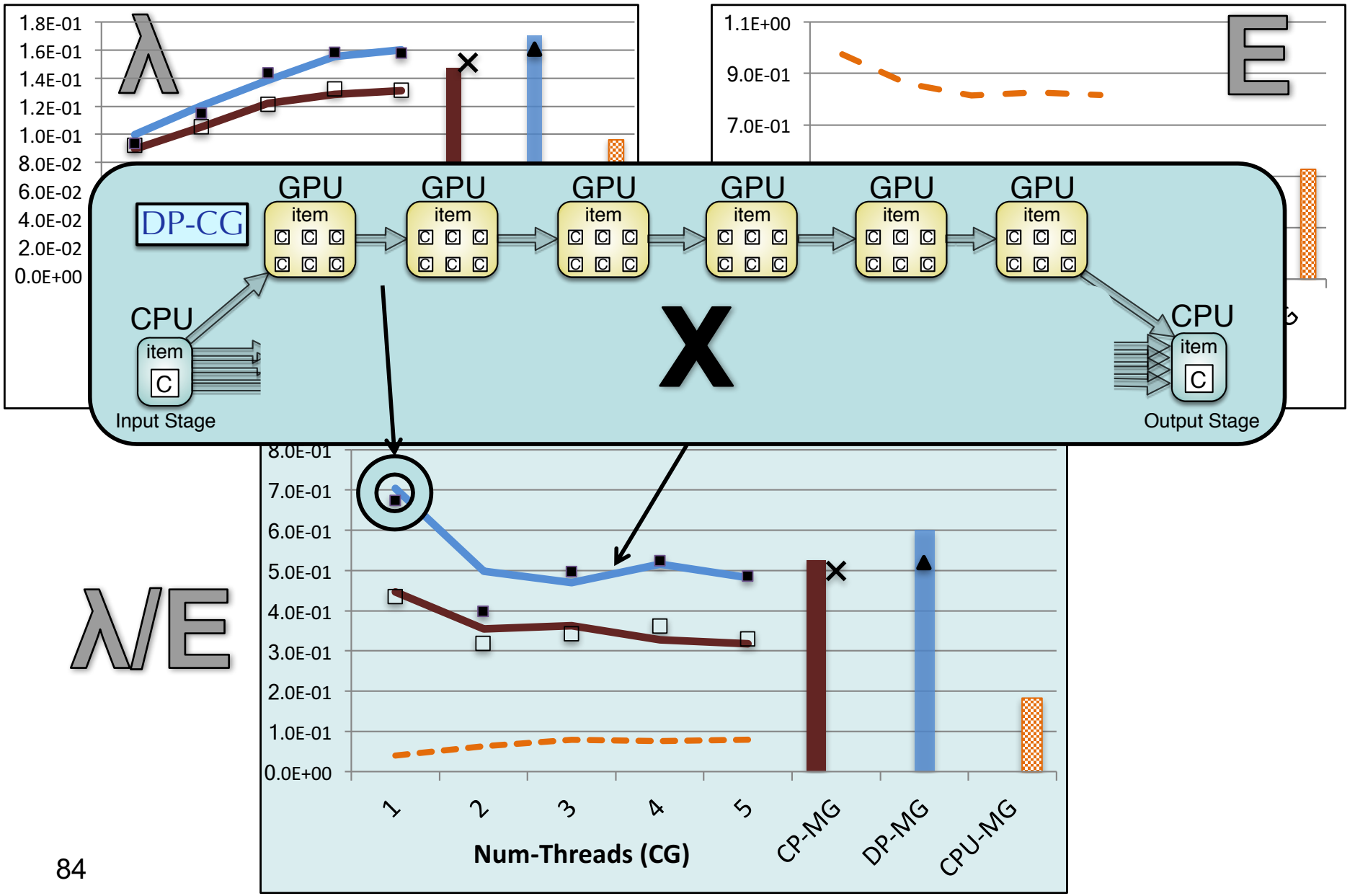




λ/E

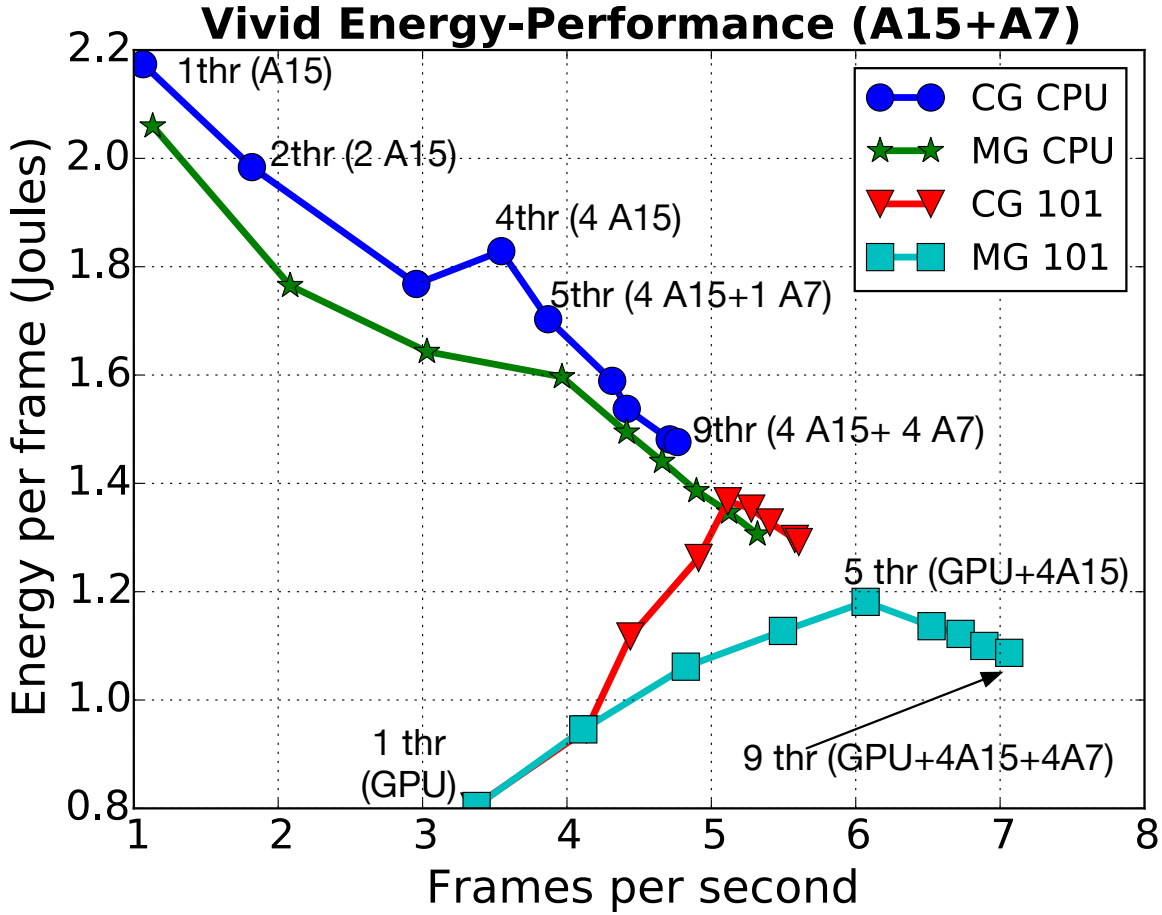
Haswell
HD

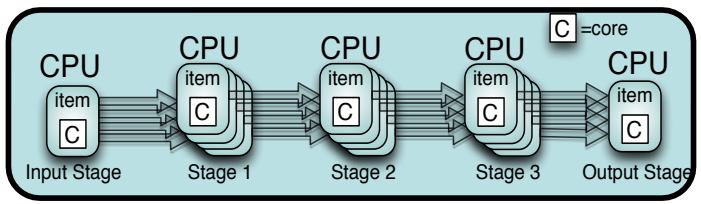
SRAD on Haswell

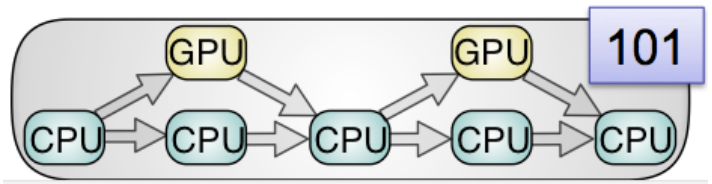


Results on big.LITTLE architecture

- On Odroid XU3: 4 Cortex A15 + 4 Cortex A7 + GPU Mali







Pipeline on CPU-FPGAs

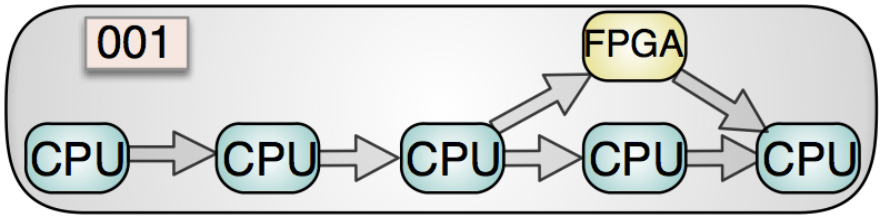
- Change the GPU by a FPGA
  - Core i7 4770k 3.5 GHz Haswell quad-core CPU
  - Altera DE5-Net FPGA from Terasic
  - Intel HD6000 embedded GPU
  - Nvidia Quadro K600 discrete GPU (192 cuda cores)

- Homogeneous results:

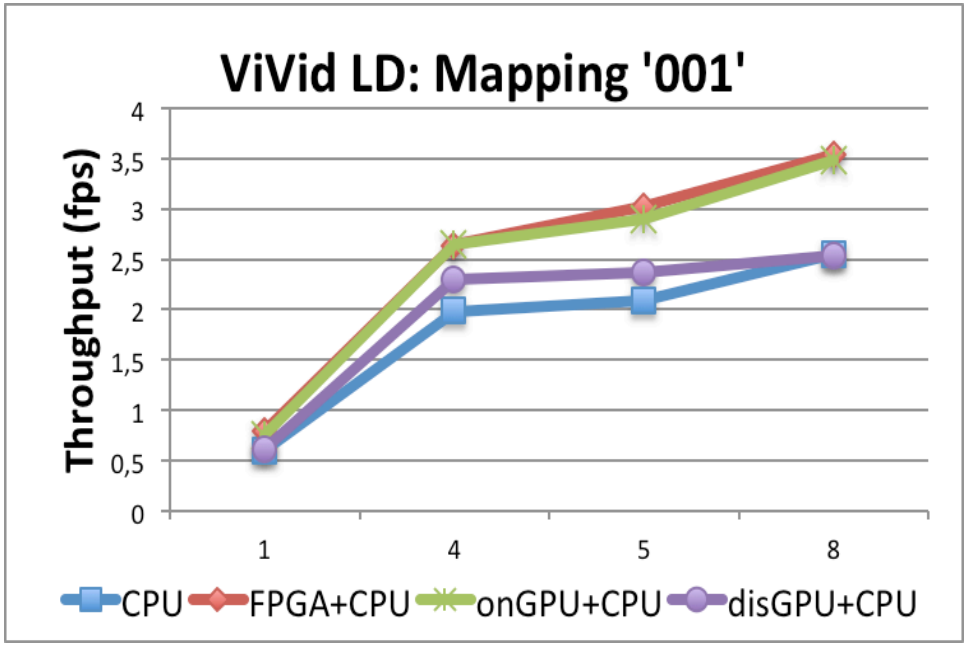
Device	Time (sec.)	Throughput (fps)	Power (watts)	Energy (Jules)
CPU (1 core)	167.610	0.596	37	6201
CPU (4 cores)	50.310	1.987	92	4628
FPGA	13.480	7.418	**5**	**67**
On-chip GPU	**7.855**	**12.730**	16	125
Discrete GPU	13.941	7.173	47	655

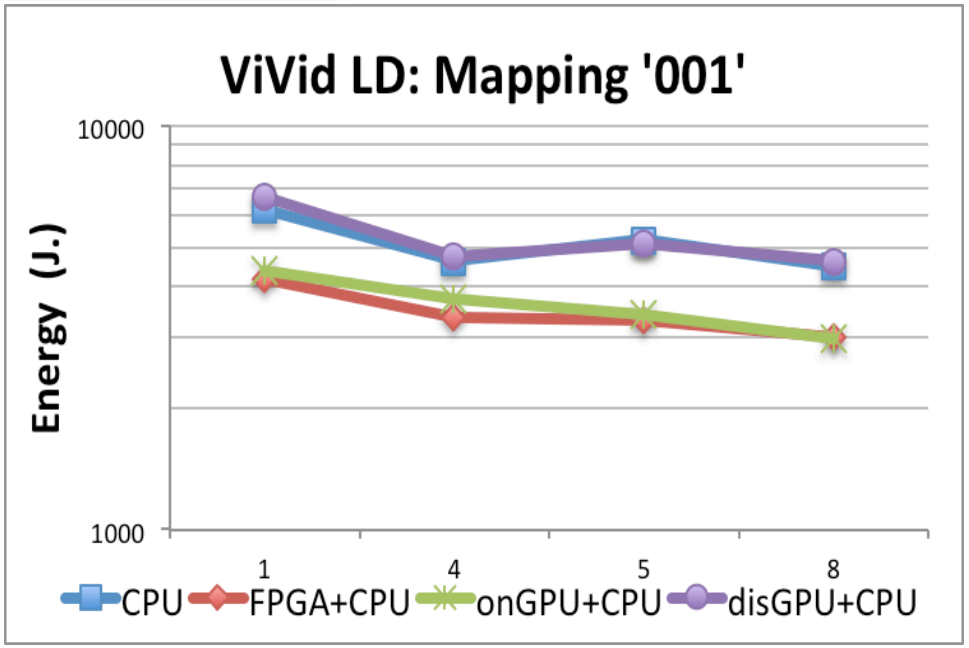
Pipeline on CPU-FPGA

- Heterogeneous results based on our TBB scheduler:
  - Accelerator + CPU
  - Mapping 001



onGPU+CPU ~ FPGA+CPU





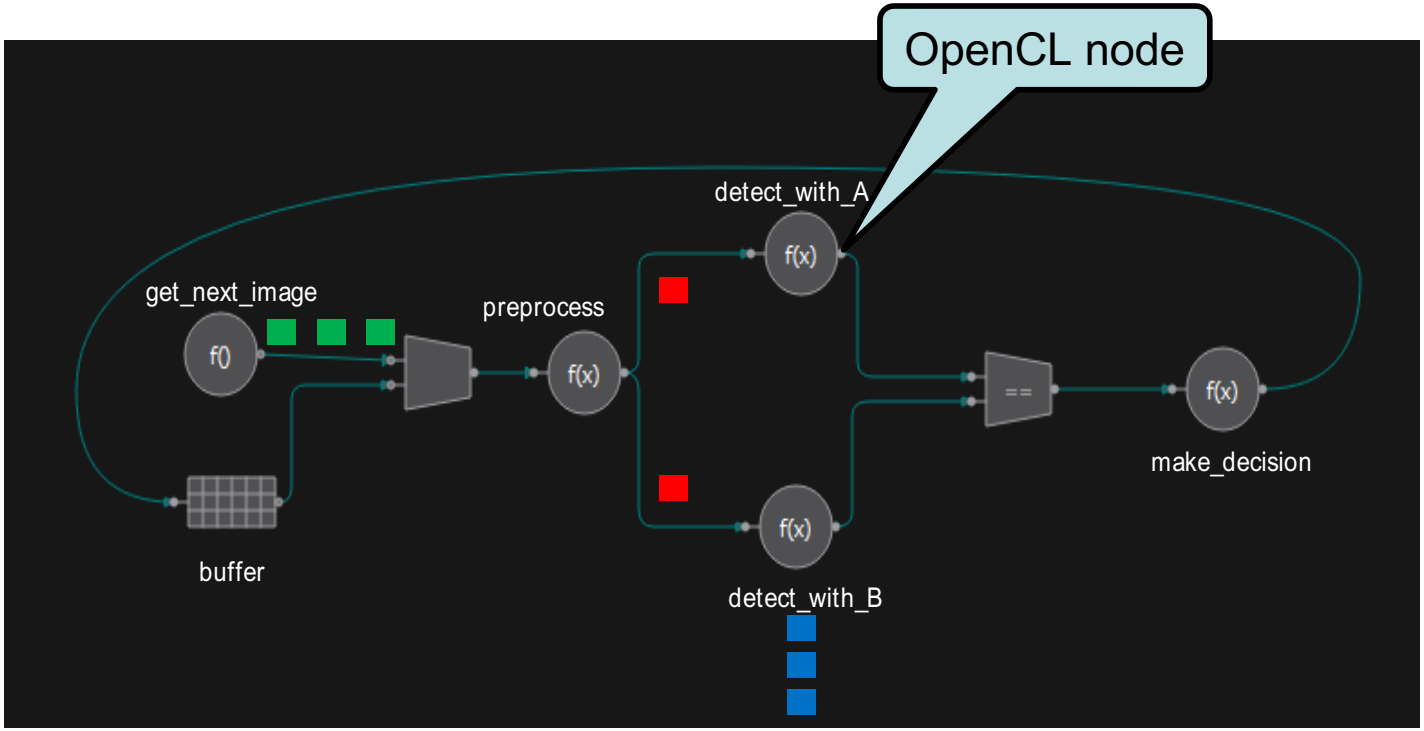
Conclusions

- Plenty of heterogeneous on-chip architectures out there
- Why not use all devices?
  - Need to find the best mapping/distribution/scheduling out of the many possible alternatives.
- Programming models and runtimes aimed at this goal are in their infancy: striving to solve this.
- Challenges:
  - Hide HW complexity providing a productive programming model
  - Consider energy in the partition/scheduling decisions
  - Minimize overhead of adaptation policies

Future Work

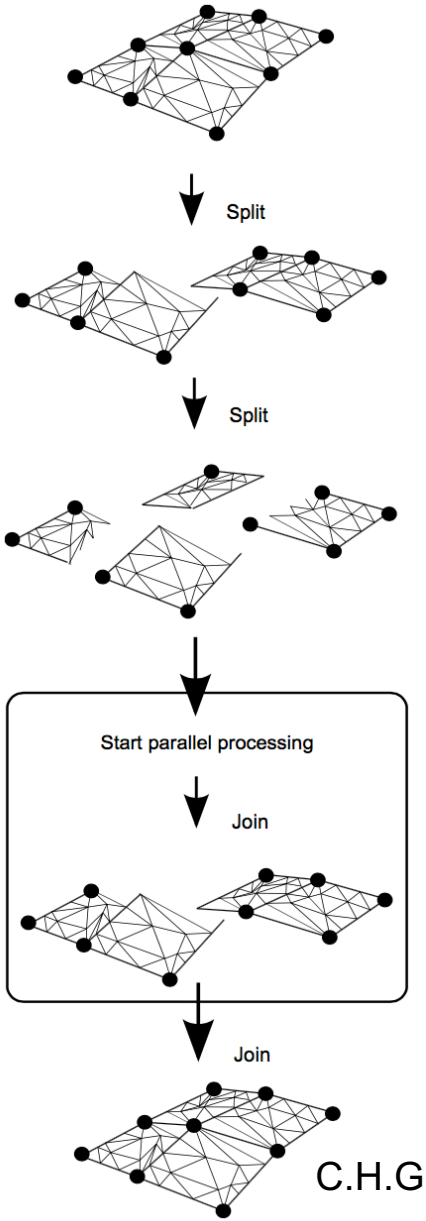
- Exploit **CPU-GPU-FPGA chips** and **coherency**
- **Predict energy** consumption on heterogeneous CPU-GPU-FPGA chips
- **Consider energy** consumption in the partitioning heuristic
- Rebuild the scheduler engine on top of the **new TBB library (OpenCL node)**
- Tackle **other irregular codes**

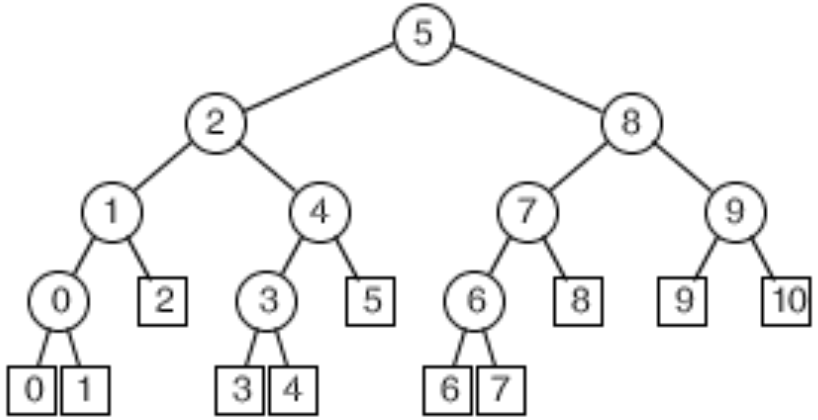
Future Work: TBB OpenCL node

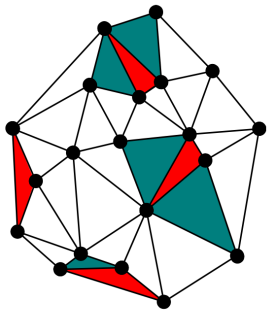


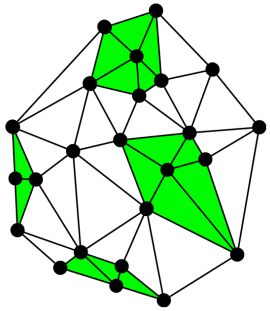
Can express pipelining, task parallelism and data parallelism

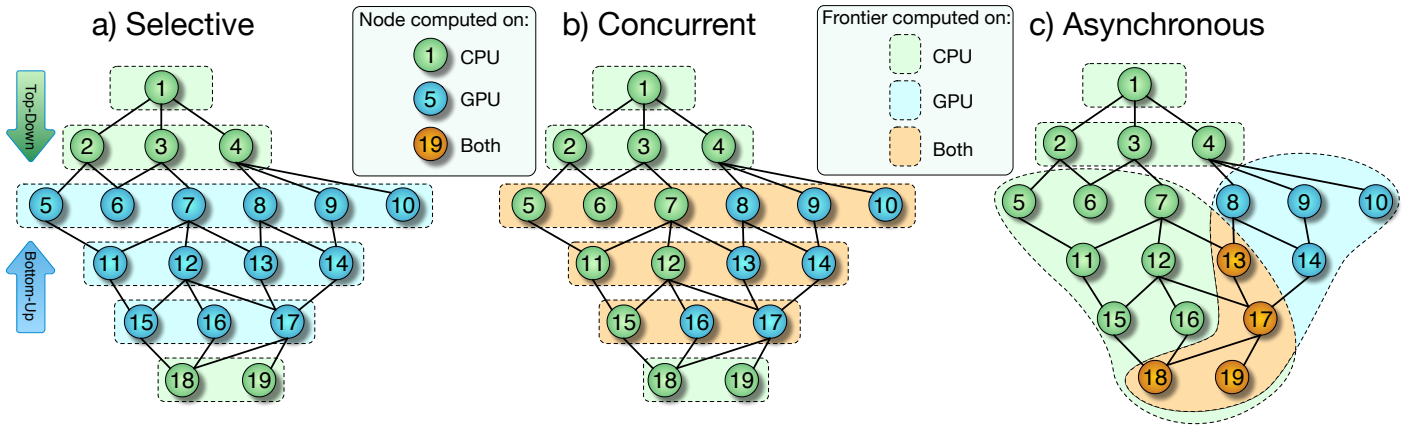
Future Work: other irregular codes











C.H.G Vázquez, PhD, Library based solutions for algorithms with complex patterns, 2015

Collaborators

- Mª Ángeles González Navarro (UMA, Spain)
- Andrés Rodríguez (UMA, Spain)
- Francisco Corbera (UMA, Spain)
- Jose Luis Nunez-Yanez (University of Bristol)
- Antonio Vilches (UMA, Spain)
- Alejandro Villegas (UMA, Spain)
- Juan Gómez Luna (U. Cordoba, Spain)
- Rubén Gran (U. Zaragoza, Spain)
- Darío Suárez (U. Zaragoza, Spain)
- Maria Jesús Garzarán (Intel and UIUC, USA)
- Mert Dikmen (UIUC, USA)
- Kurt Fellows (UIUC, USA)
- Ehsan Totoni (UIUC, USA)
- Luis Remis (Intel, USA)

Questions

asenjo@uma.es