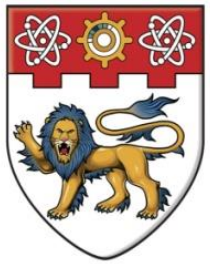


NANYANG TECHNOLOGICAL UNIVERSITY

Resource-Aware Just-in-Time OpenCL Compiler for Coarse-Grained FPGA Overlays

Abhishek Kumar Jain, **Douglas L. Maskell**
School of Computer Science and Engineering
Nanyang Technological University (NTU), Singapore

Suhaib A. Fahmy
School of Engineering
University of Warwick, UK

3rd International Workshop on Overlay Architectures for FPGAs (OLAF)
22nd Feb 2017, Monterey, CA, USA

Hardware Accelerators

- Many different platforms for hardware acceleration of compute intensive applications
  - These include GPUs, FPGAs, etc.
- GPUs are more widely used due to
  - The ease of use and better design productivity
  - Better support for the OpenCL programming model
  - Faster design cycles

Application in high-level language
< 1 min application compile time

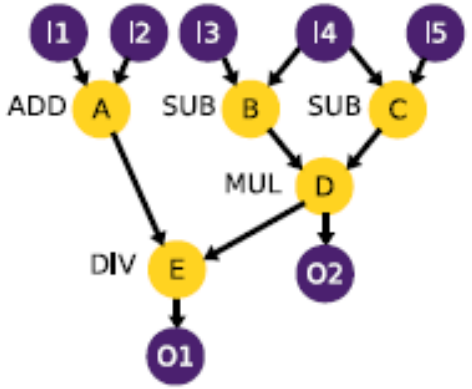
Application in high-level language
10 hour compile time

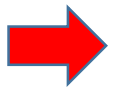
Hardware Accelerators

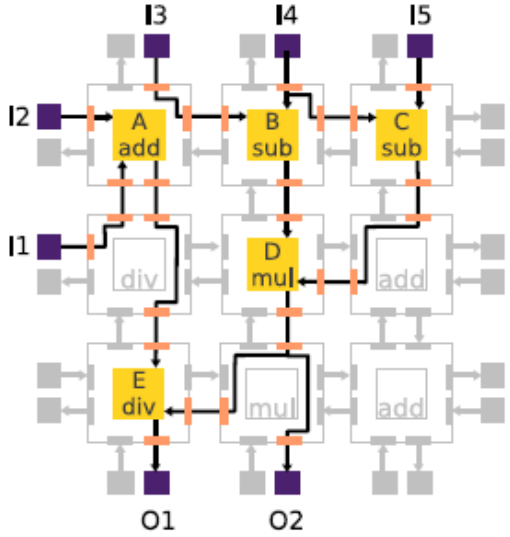
- Many different platforms for hardware acceleration of compute intensive applications
  - These include GPUs, FPGAs, etc.
- GPUs are more widely used due to
  - The ease of use and better design productivity
  - Better support for the OpenCL programming model
  - Faster design cycles
- Issues with FPGA based accelerators *(past and current)*
  1. Low level of programming abstraction (RTL) – HLS tools, Eg: Vivado-HLS
  2. Runtime management (interfaces and drivers) – Newer tools, Eg: SDSoC, SDAccel, AOCL
  3. Long compile times and slow switching between application kernels – Overlays
  4. A lack of application portability and performance scalability – Overlays + OpenCL

So what is an Overlay?

- A coarse-grained circuit abstraction which sits on top of the FPGA fabric
- Many similarities to CGRAs
- Because it is coarse-grained, it provides easier application mapping, faster compilation and faster application kernel configuration



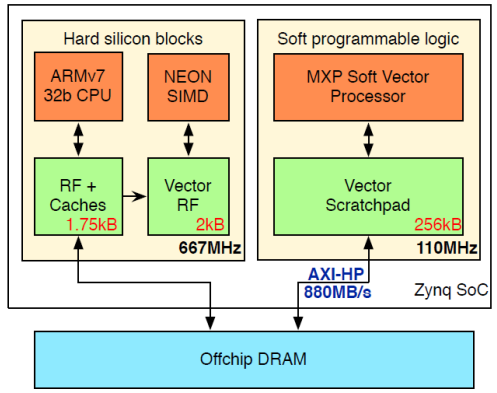




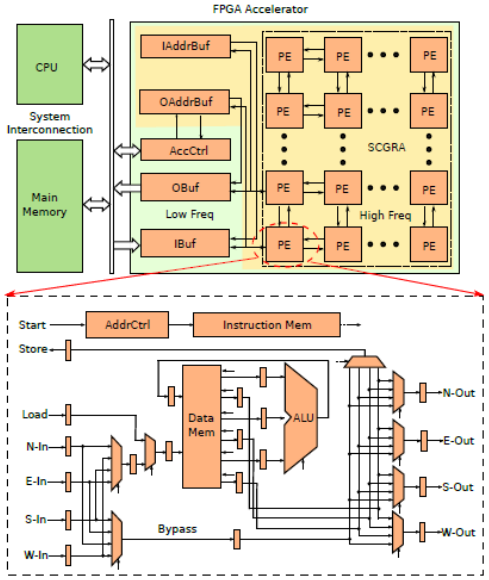
Images from University of Toronto [3]

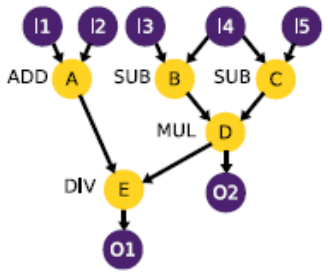
So what is an Overlay?

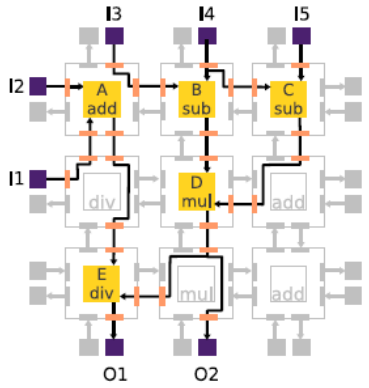
MXP Overlay (VectorBlox) [1]



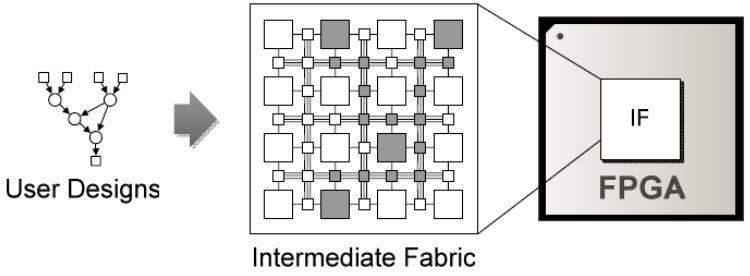
SCGRA Overlay (HKUST, Hong Kong) [2]



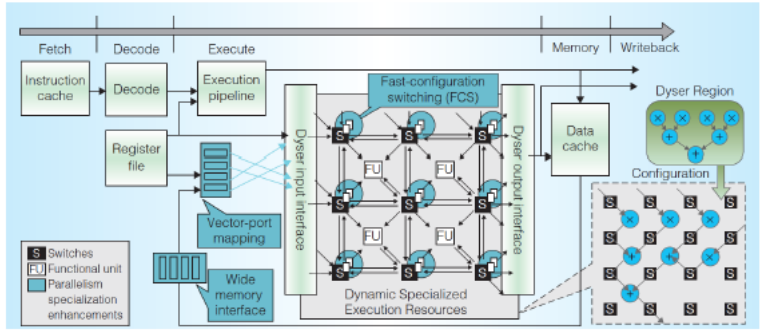




Mesh-of-FU Overlay (University of Toronto) [3]



Intermediate Fabric (University of Florida) [4]

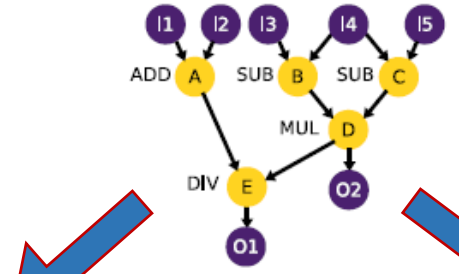


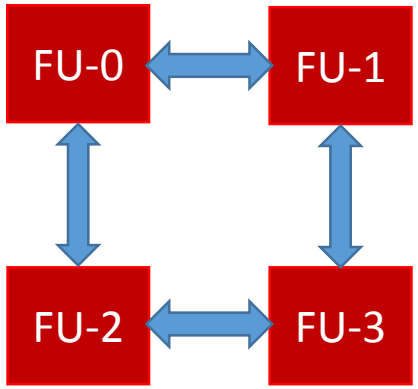
DySER (UW Madison) [5]

1. *Severance, Aaron, and Guy GF Lemieux. "Embedded supercomputing in FPGAs with the VectorBlox MXP matrix processor." CODES+ ISSS, 2013.*
2. *Liu, Cheng, Ho-Cheung Ng, and Hayden Kwok-Hay So. "QuickDough: a rapid fpga loop accelerator design framework using soft CGRA overlay." FPT 2015.*
3. Capalija, Davor, and Tarek S. Abdelrahman. "A high-performance overlay architecture for pipelined execution of data flow graphs." *FPL* 2013.
4. *G. Stitt and J. Coole, "Intermediate fabrics: Virtual architectures for near-instant FPGA compilation," IEEE ESL, vol. 3(3), 2011.*
5. *J. Benson et al., "Design, integration and implementation of the DySER hardware accelerator into OpenSPARC," HPCA 2012.*

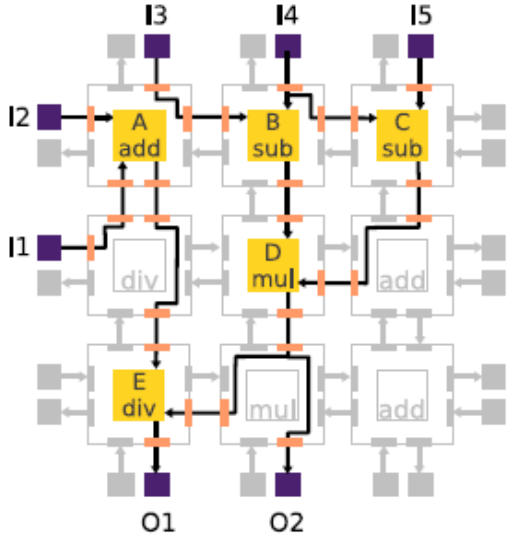
Classification based on Architecture[6]

- Time multiplexed
- Spatially configured
- Packet switched, and
- Circuit switched





Overlay (Time-multiplexed)
Similar to conventional CGRAs



Overlay (Spatially-configured)

Images from: University of Toronto [3]

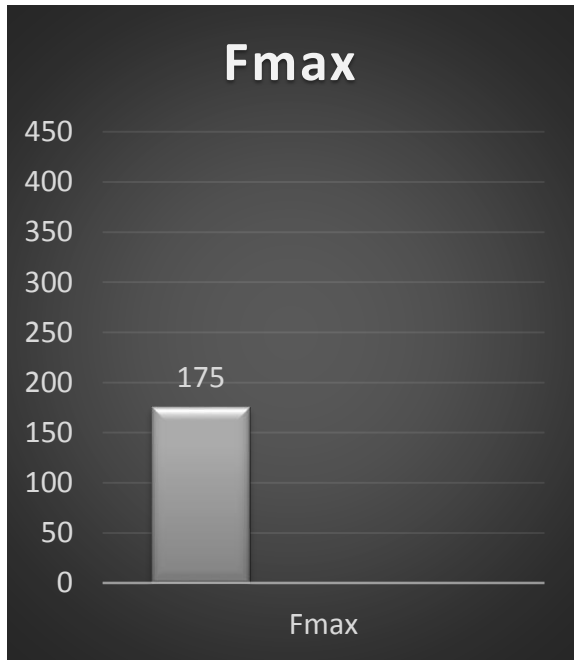
6. *Kapre, Nachiket, et al. "Packet switched vs. time multiplexed FPGA overlay networks." FCCM 2006.*

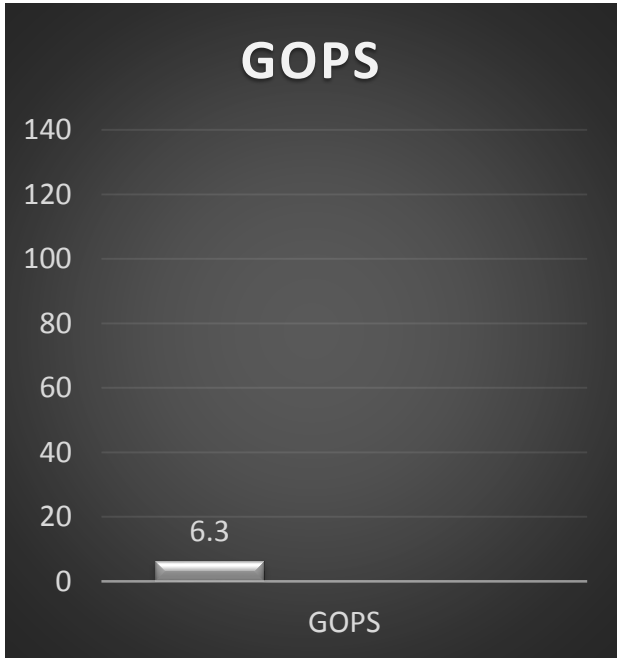
Started looking at SC overlays in 2013

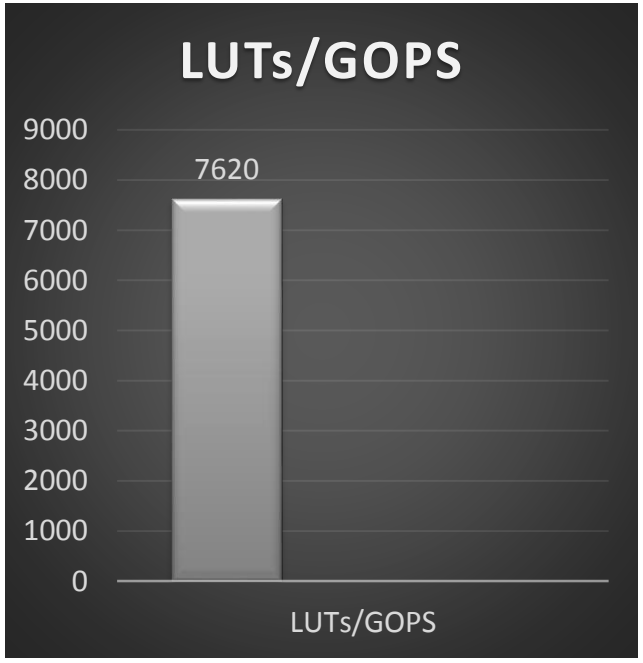
DSP Block DySER

Showed that properly exploiting DSP Block programmability and pipelining improves overlay FU area by 25% and frequency by 2.5×

Jain, Maskell, Fahmy, HEART 2015





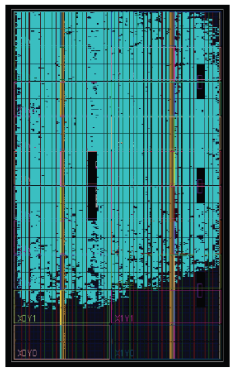


Started looking at SC overlays in 2013

DSP Block DySER

Showed that properly exploiting DSP Block programmability and pipelining improves overlay FU area by 25% and frequency by 2.5×

Jain, Maskell, Fahmy, HEART 2015



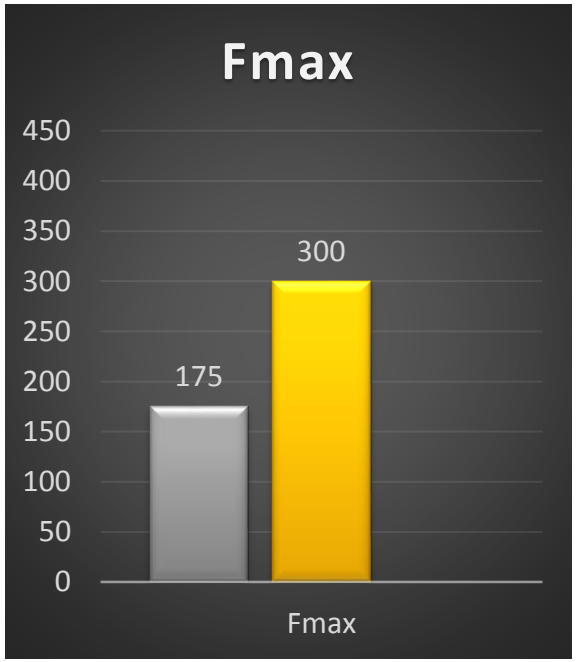
Large DSP Block Overlay

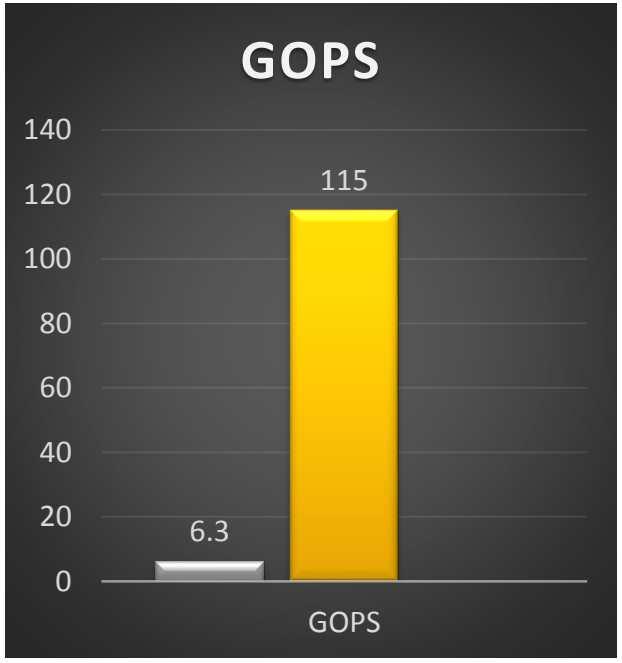
Showed that architecture aware design of an overlay can enable implementation of a large overlay on Zynq/V7

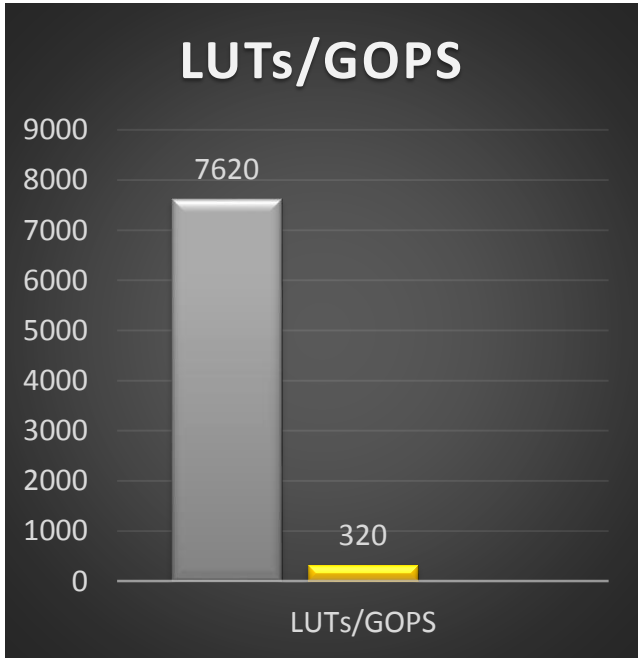
Zynq: 128 DSP blocks at 300 MHz

V7: 800 DSP blocks at 380 MHz

Jain, Fahmy, Maskell





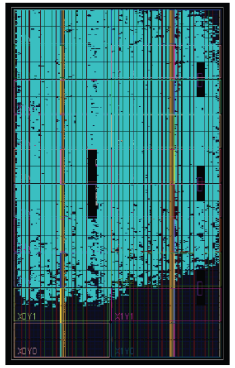


Started looking at SC overlays in 2013

DSP Block DySER

Showed that properly exploiting DSP Block programmability and pipelining improves overlay FU area by 25% and frequency by 2.5×

Jain, Maskell, Fahmy, HEART 2015



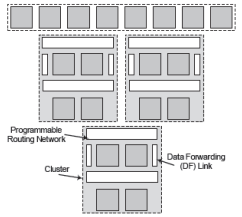
Large DSP Block Overlay

Showed that architecture aware design of an overlay can enable implementation of a large overlay on Zynq/V7

Zynq: 128 DSP blocks at 300 MHz

V7: 800 DSP blocks at 380 MHz

Jain, Fahmy, Maskell

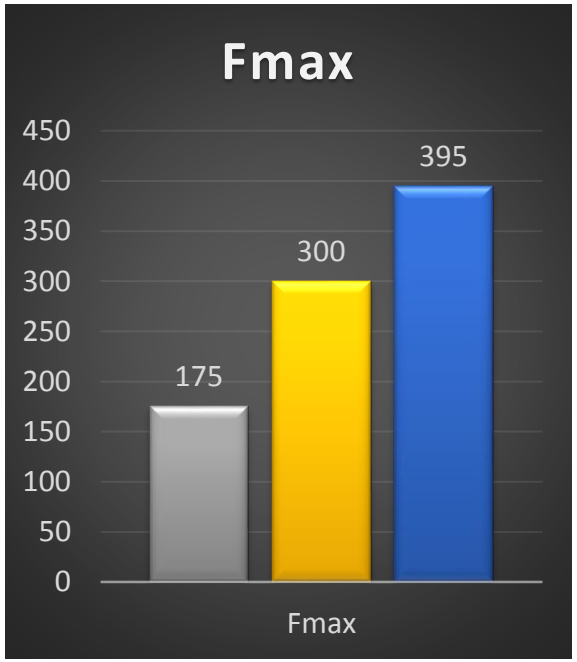


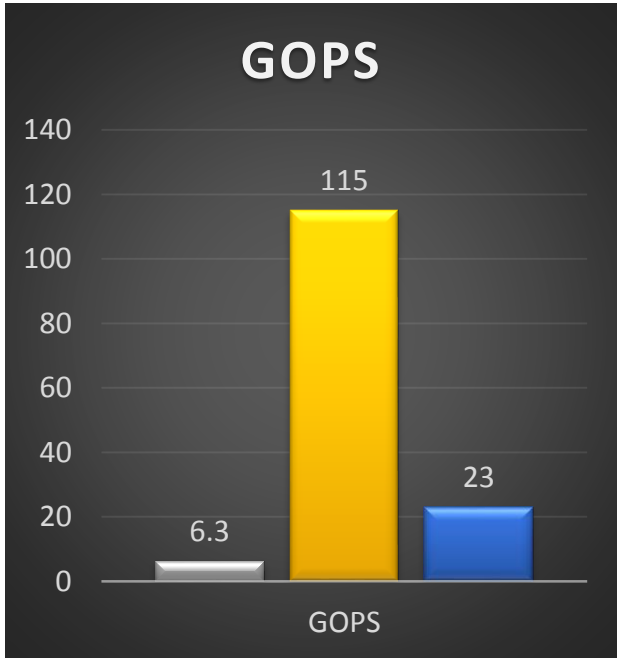
DeCO Datapath Overlay

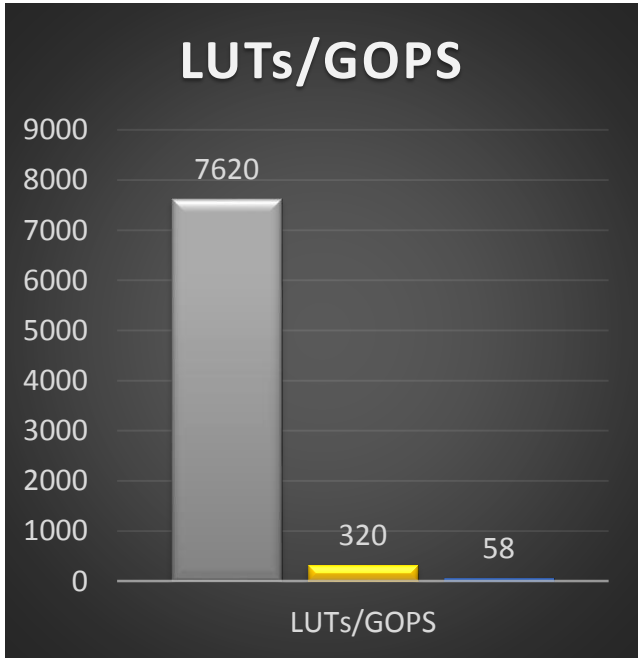
For many applications, completely general routing is not needed so built a feed-forward overlay with reduced area and more balanced LUT/DSP Block usage

Configuration in 2 us.

Jain, Maskell, Fahmy, FCCM 2016





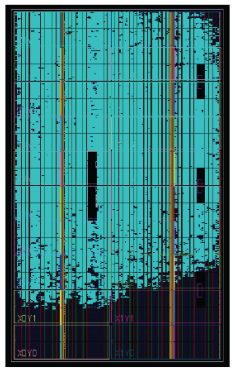


Coarse-grained Overlays

DSP Block DySER

Showed that properly exploiting DSP Block programmability and pipelining improves overlay FU area by 25% and frequency by 2.5×

Jain, Maskell, Fahmy, HEART 2015



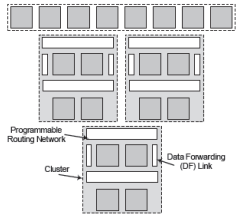
Large DSP Block Overlay

Showed that architecture aware design of an overlay can enable implementation of a large overlay on Zynq/V7

Zynq: 128 DSP blocks at 300 MHz

V7: 800 DSP blocks at 380 MHz

Jain, Fahmy, Maskell



DeCO Datapath Overlay

For many applications, completely general routing is not needed so built a feed-forward overlay with reduced area and more balanced LUT/DSP Block usage

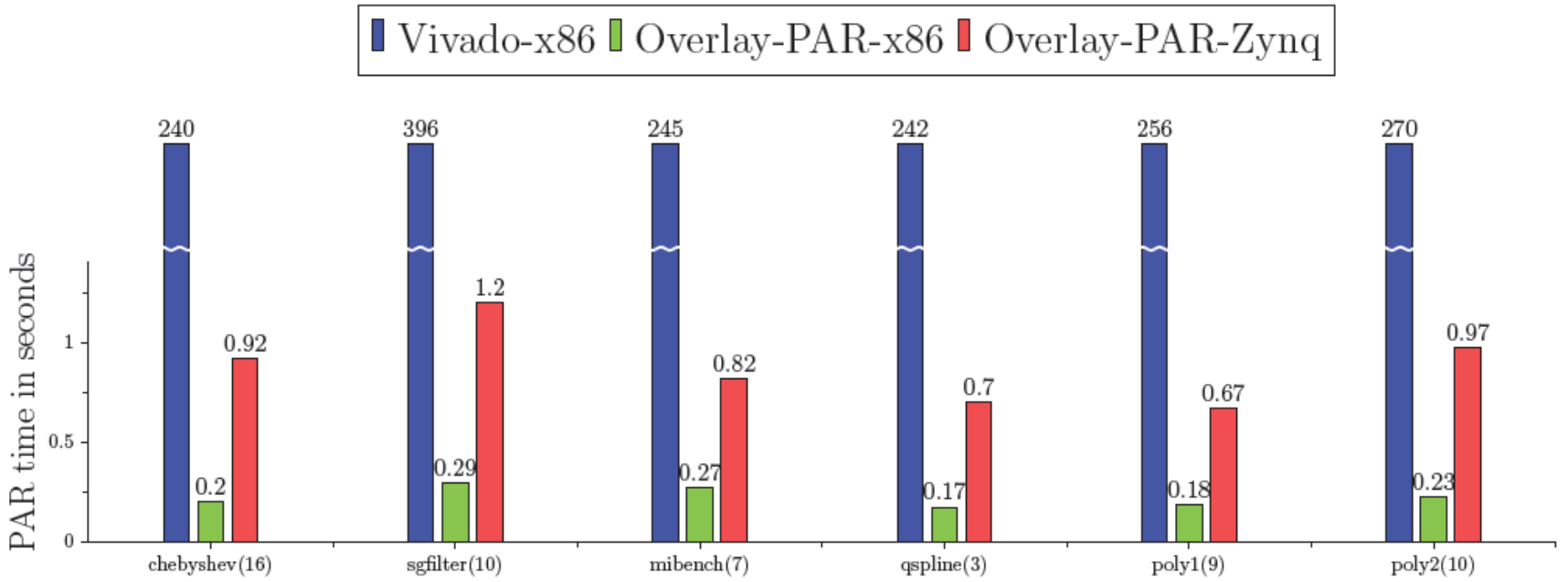
Configuration in 2 us.

Jain, Maskell, Fahmy, FCCM 2016

1000x faster place and route

Place and route within a second on embedded ARM in Zynq

Get a very fast configuration time. (µs rather than ms)



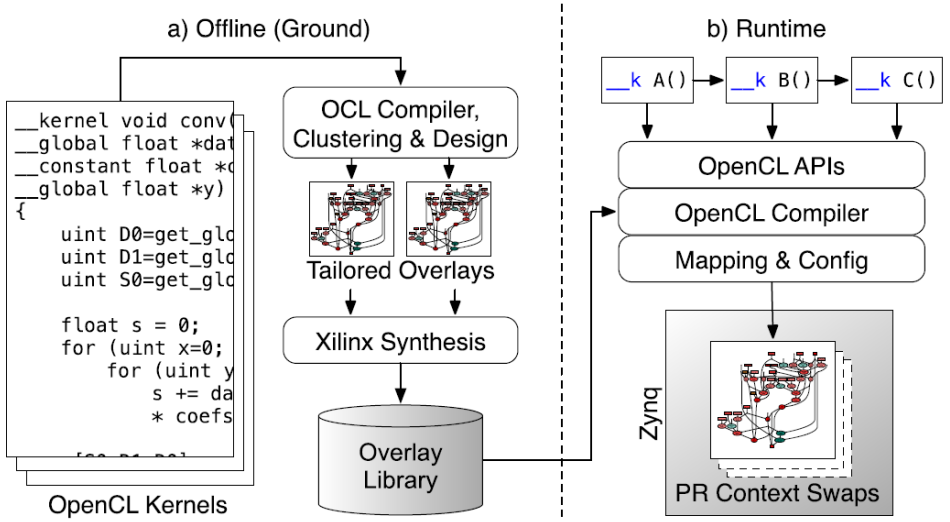
Coarse-grained Overlays

So can we use these overlays to provide a More GPU like experience?

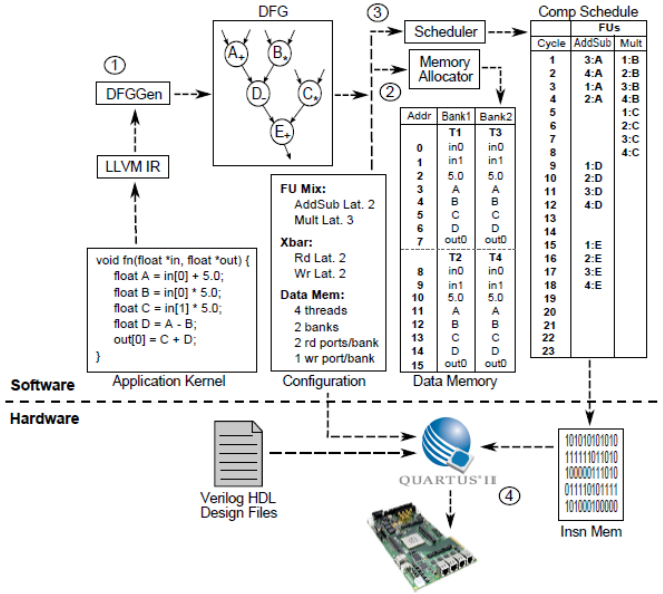
- OpenCL on GPU allows:
  - Fast compilation and configuration
  - Application portability across other accelerators (LLVM IR as abstraction layer)
  - Performance scaling by exploiting just-in-time (JIT) compilation

Coarse-grained Overlays + OpenCL

- Recent work focused on exposing overlay as an OpenCL device
  - Provides a more GPU like experience by exploiting fast compilation and configuration
  - Does not exploit kernel replication feature of OpenCL like the one used by GPUs



Intermediate Fabric Overlay (University of Florida) [7]

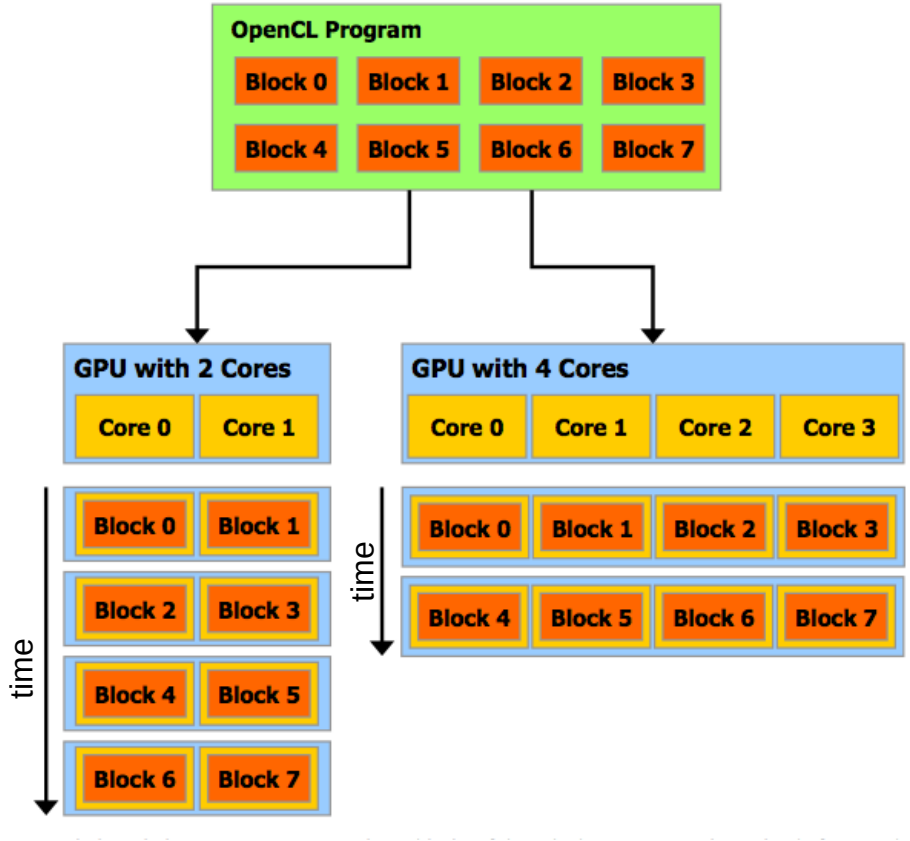


TILT Overlay (University of Toronto) [8]

1. *Coole, James, and Greg Stitt. "Fast, flexible high-level synthesis from OpenCL using reconfiguration contexts." IEEE Micro, 2014*
2. *Rashid, Rafat, J. Gregory Steffan, and Vaughn Betz. "Comparing performance, productivity and scalability of the TILT overlay processor to OpenCL HLS." FPT 2014.*

Performance Scaling on GPUs

- Can automatically (at runtime) scale performance if additional hardware resource is avaliable
- Idea:
  - Compiling application kernel at runtime from kernel source (JIT)
  - unroll/replicate the kernel based on the availability of hardware resources
- Direct runtime compilation for FPGAs is infeasible
- Overlays can allow runtime compilation and performance scaling!

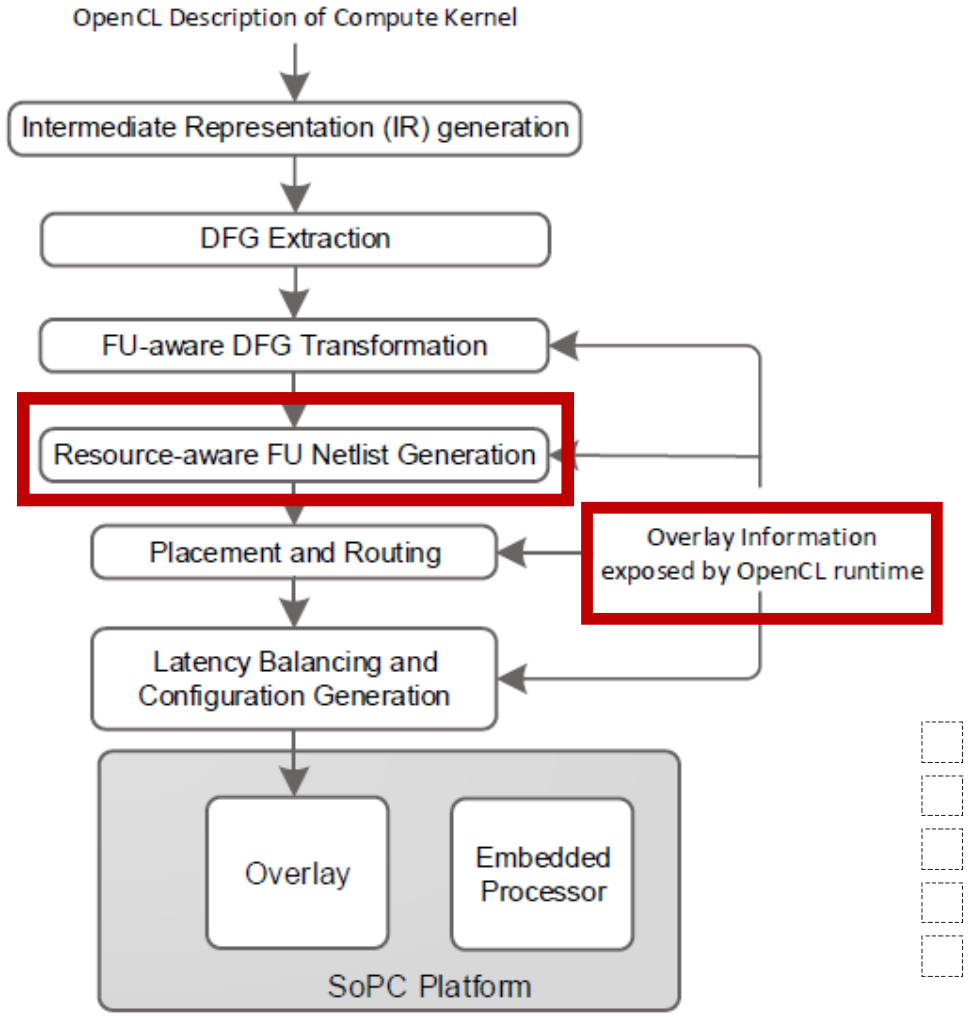


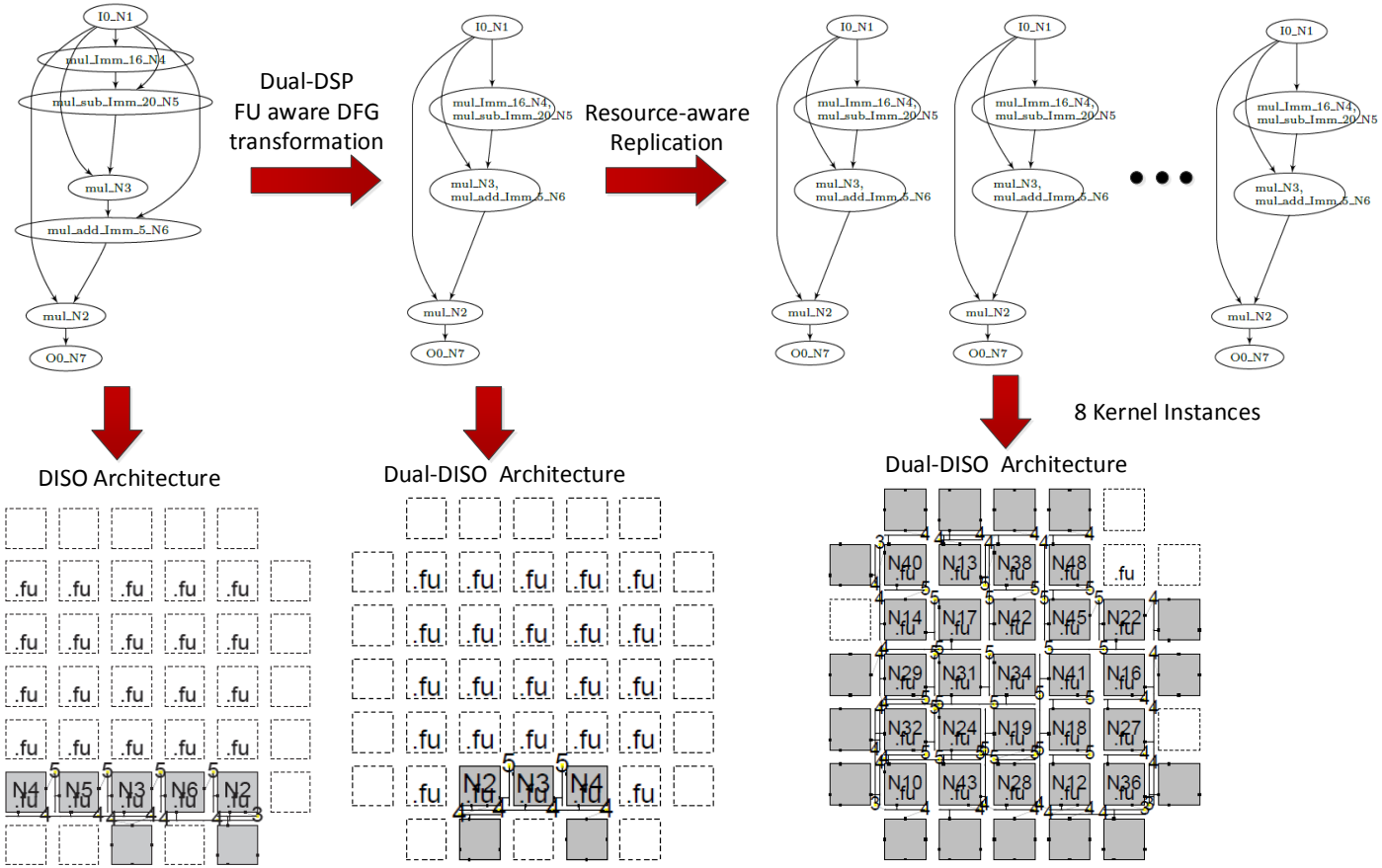
RESOURCE, BUILD TIME AND POWER OF OPENCL AES KERNEL

attr.	Logic	Reg.	Memory	DSP	Time	Power
Original	44%	12%	30%	0%	70 min	29.6 W
SIMD 2	62%	13%	29%	0%	162 min	29.9 W
SIMD 4	98%	14%	31%	0%	523 min	30.1 W
COMP 2	68%	15%	35%	0%	82 min	30.6 W
COMP 3	89%	17%	41%	0%	94 min	31.0 W

1. S. Gao and J. Chritz. "Characterization of OpenCL on a Scalable FPGA Architecture" *ReConFig 2014*

Coarse-grained Overlays + OpenCL

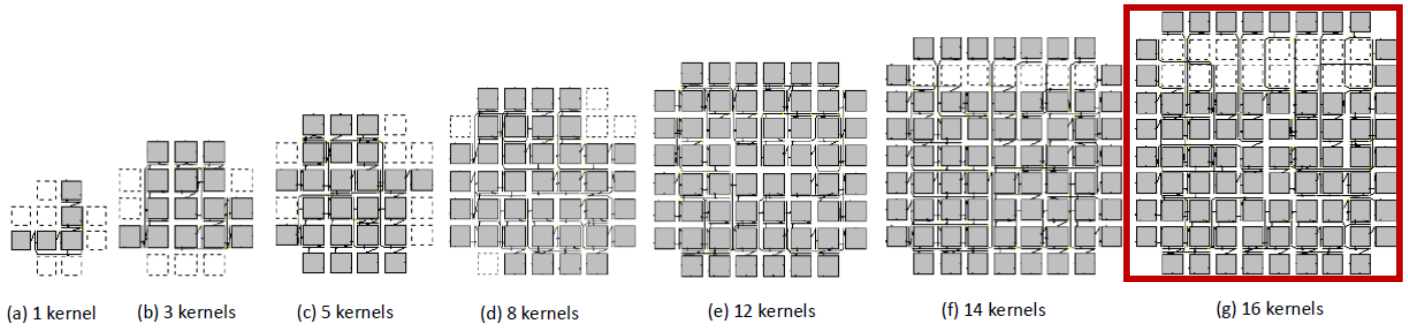


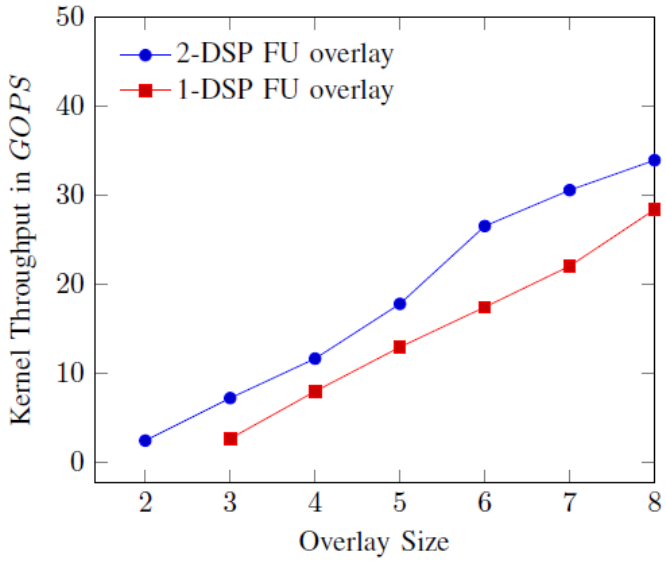


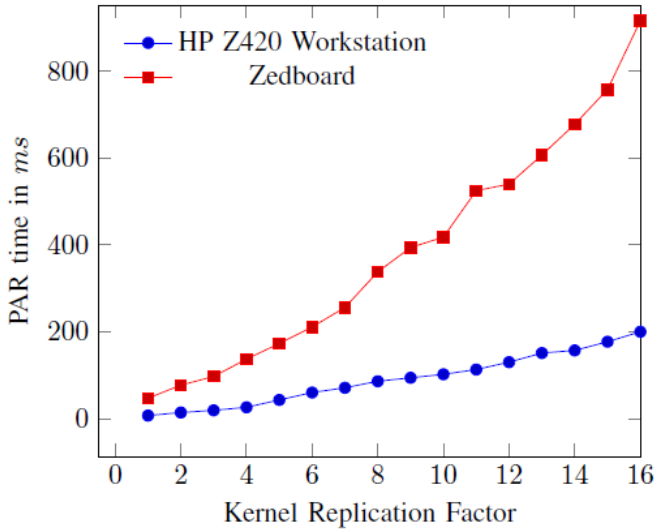
Resource-aware Performance Scaling

- Proposed approach can help in performance scaling on providing more hardware resources

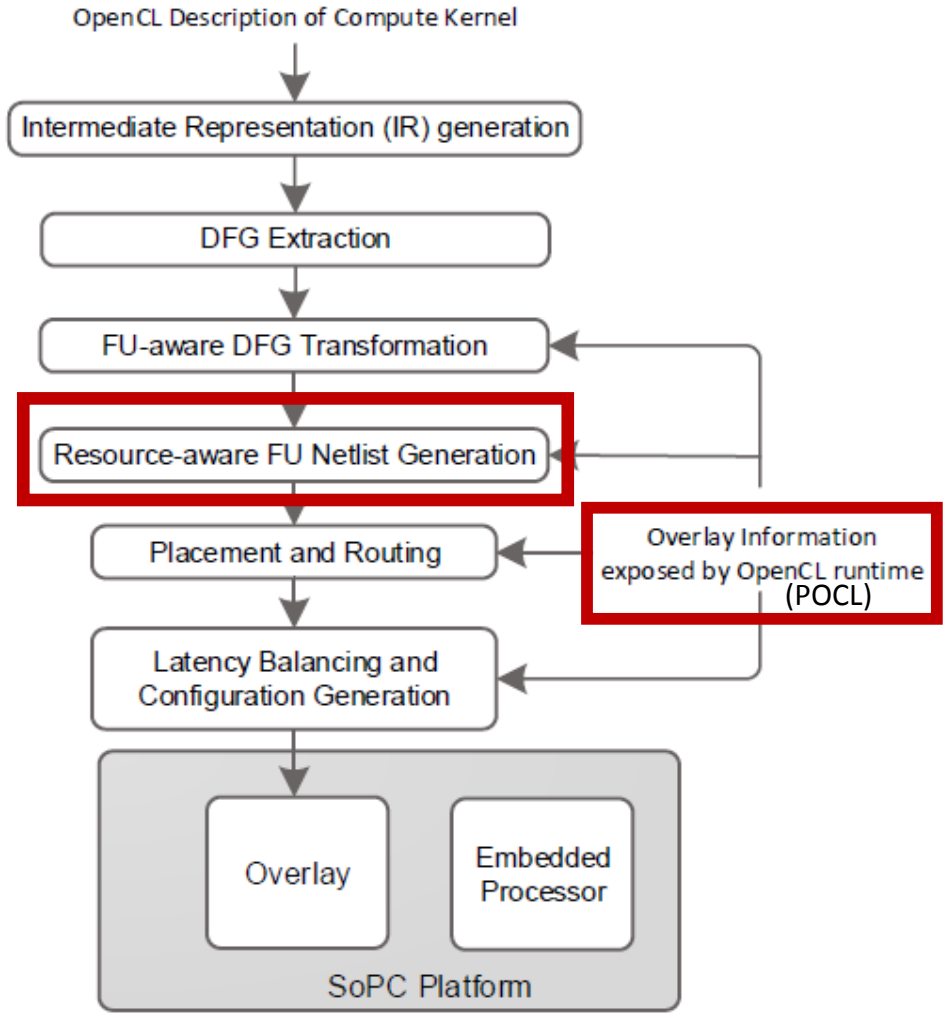
- Instead of mapping single copy of kernel on 8x8 array, compiler can replicate the kernel 16 times

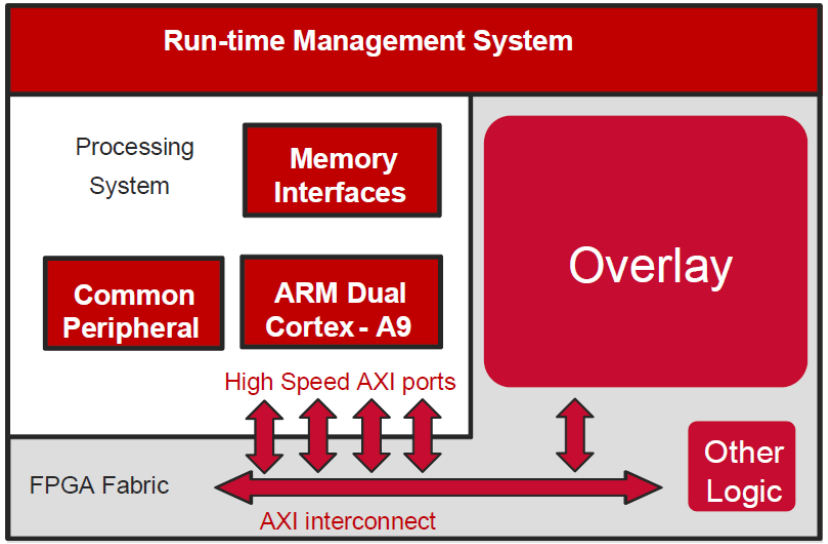






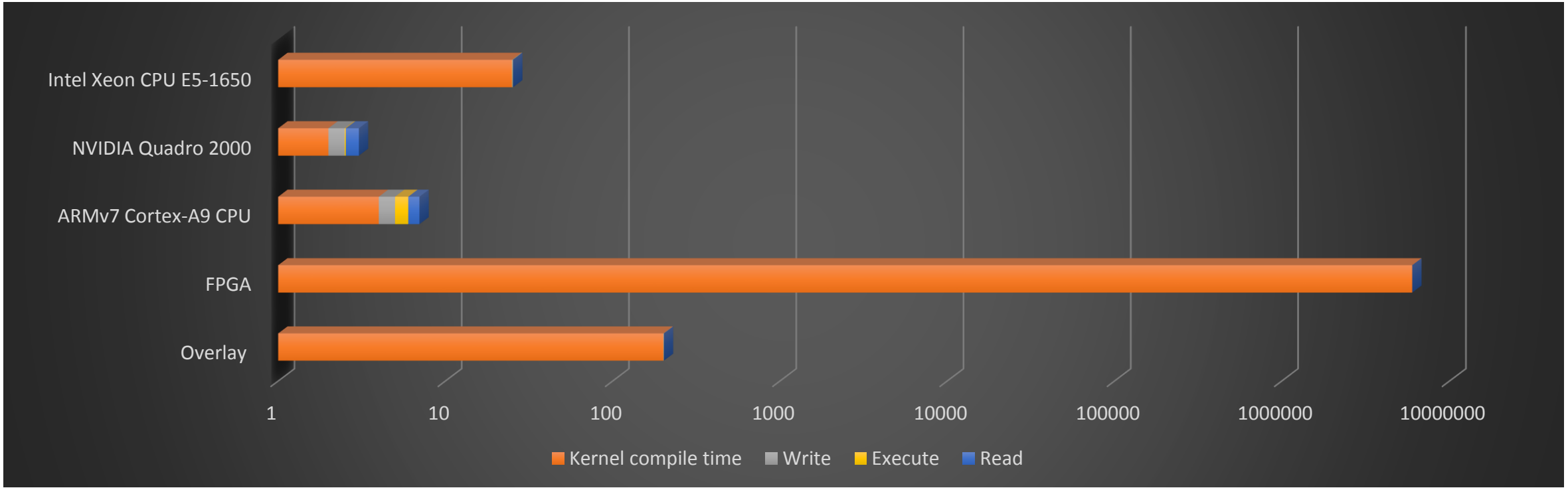
Coarse-grained Overlays + OpenCL





OpenCL Kernel Execution Profile

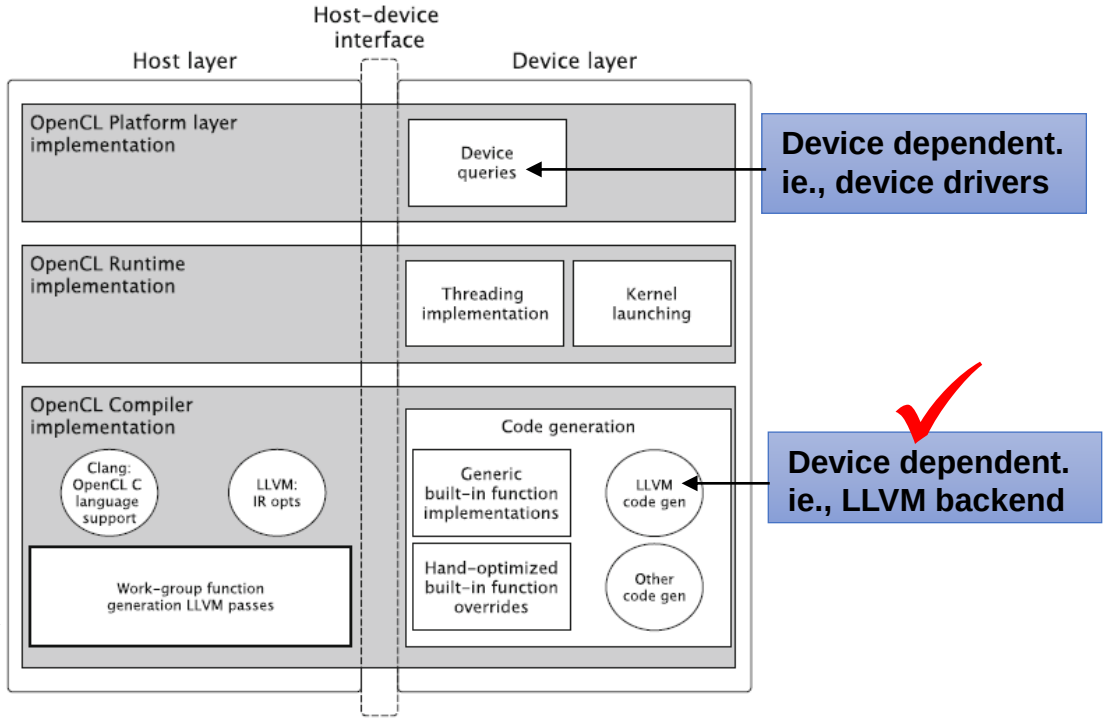
- Application: Apply negate kernel (total inversion, *255 – pixel value*) on 400x225 grayscale image
- X-axis shows time in milliseconds



Summary

- Proposed a resource-aware Just-in-Time OpenCL compiler for FPGA overlays
- Embedded ARM processor on Zynq device can compile kernels within a second

- Future Work:
  - Integration within POCL framework on Zynq-v2
  - Execution of OpenCL benchmarks
  - Comparison with GPU
  - Continue the DSP-based overlay research
    - Integration of multiple DeCO in the accelerator framework
    - Efficient time multiplexed overlays
    - Compilation of TensorFlow Graphs onto Overlays



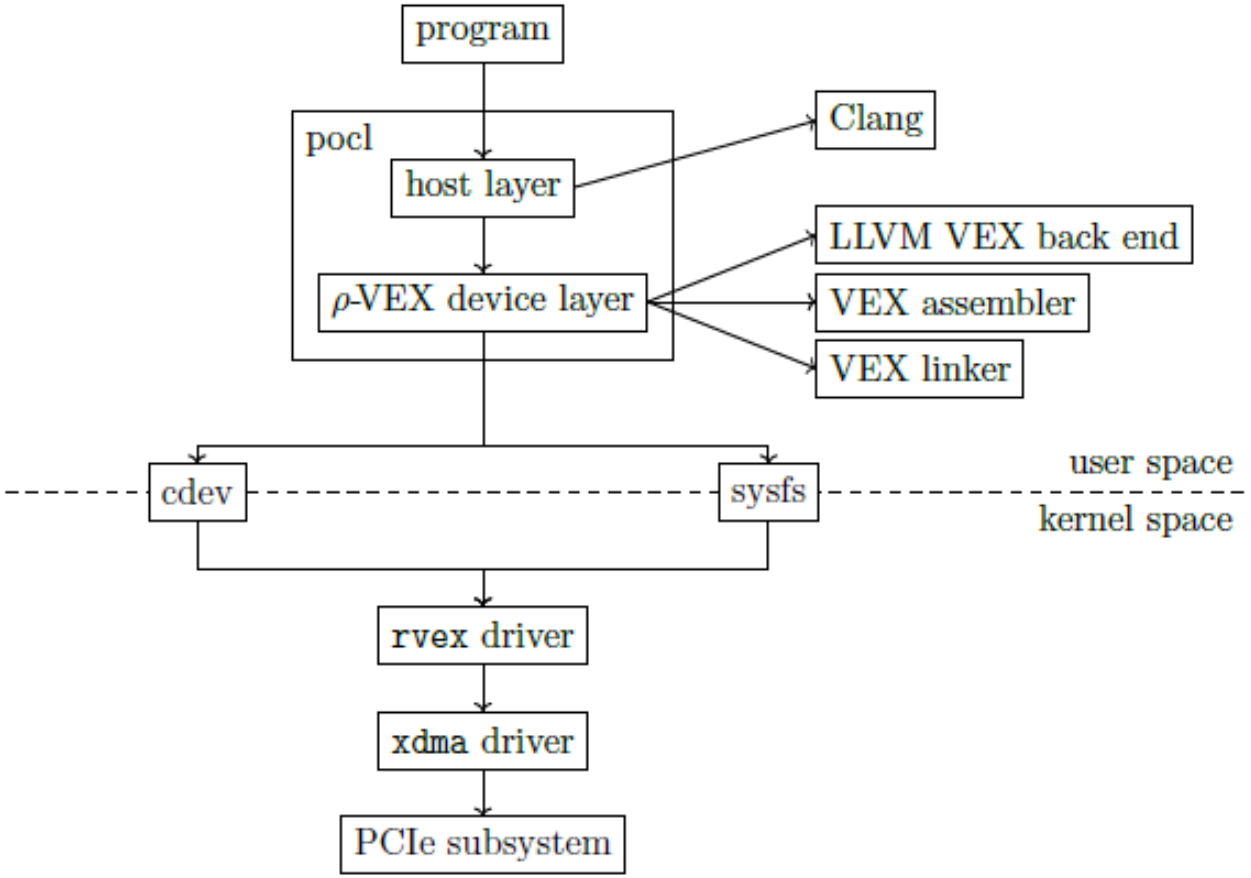
Thank you

Results

Benchmark name	Overlay implementations			Direct FPGA implementations			Resource Penalty (DSP — Slices)	F_{max} Improvement	PAR speedup
	PAR time (seconds)	F_{max} (MHz)	Resource (DSP — Slices)	PAR time (seconds)	F_{max} (MHz)	Resource (DSP — Slices)			
chebyshev(16)	0.2	300	128 — 12617	240	225	48 — 251	2.6× — 50×	1.3×	1200×
sgfilter(10)	0.29	300	128 — 12617	396	185	100— 797	1.2× — 15×	1.6×	1365×
mibench(7)	0.27	300	128 — 12617	245	230	21 — 403	6.0× — 31×	1.3×	907×
qspline(3)	0.17	300	128 — 12617	242	165	36 — 307	3.5× — 41×	1.8×	1423×
poly1(9)	0.18	300	128 — 12617	256	175	36 — 425	3.5× — 29×	1.7×	1422×
poly2(10)	0.23	300	128 — 12617	270	172	40 — 453	3.2× — 27×	1.7×	1173×

Summary

- Similar work: Supporting *ρ-VEX* vector processor as an OpenCL device using POCL



Summary

- Similar work: Supporting *ρ-VEX* vector processor as an OpenCL device using POCL
- Bad performance for the evaluated benchmark
- Edge-detection algorithm applied on 640x480 image using 3x3 mask size

Processed images	Time (ms)	
	rvex	basic
1	1520	17.84
2	3040	32.60
3	4560	47.73
4	6083	64.42
5	7604	79.39
6	9125	95.28
7	10646	110.44
8	12166	126.62

ρ-VEX				
compile	work-group function	instr. gen.	link	total
1746.4 ms	2129.3 ms	754.8 ms	28.8 ms	4659.4 ms

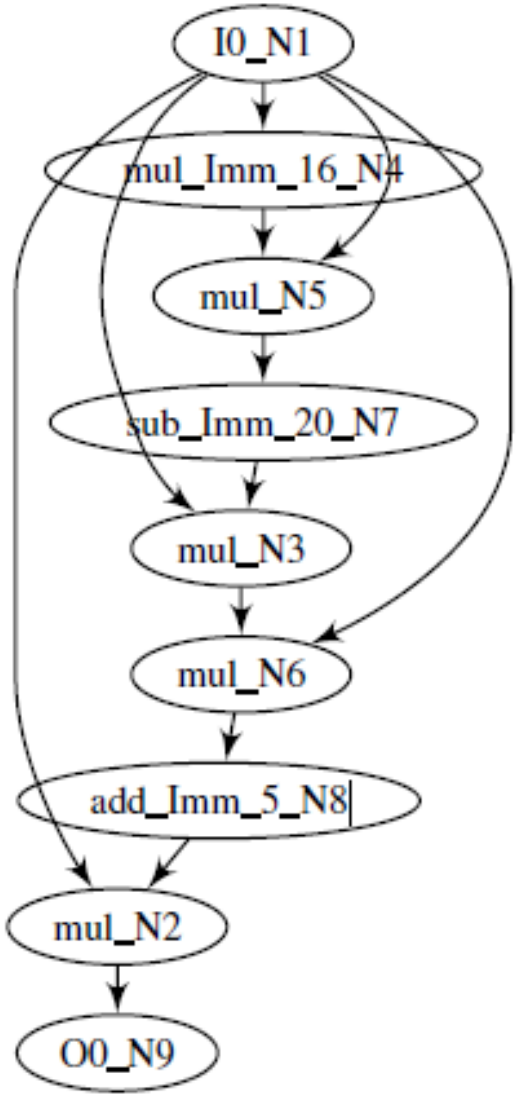
basic			
compile	work-group function	instr. gen.	total
2820.6 ms	2357.2 ms	302.4 ms	5480.2 ms

Additional Slides for tool-flow

Additional Slides for tool-flow

(a) OpenCL Description of the Kernel

```
__kernel void example_kernel(__global int *A, __global int *B)
{
        int idx = get_global_id(0);
        int x = A[idx];
        B[idx] = (x*(x*(16*x*x-20)*x+5)) ;
}
```



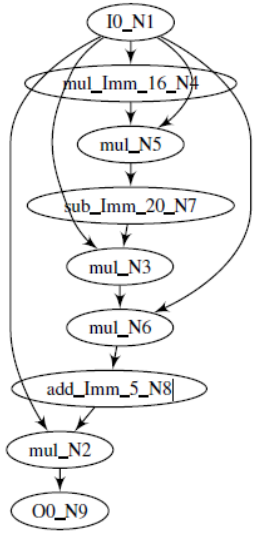
Compilation

(b) Intermediate Representation (IR) of the Kernel

```
%0:
%1 = alloca i32*, align 4
%2 = alloca i32*, align 4
%idx = alloca i32, align 4
%x = alloca i32, align 4
store i32* %A, i32** %1, align 4
store i32* %B, i32** %2, align 4
%3 = call i32 bitcast (i32 (...)* @get_global_id to i32 (i32)*)(i32 0)
store i32 %3, i32* %idx, align 4
%4 = load i32* %idx, align 4
%5 = load i32** %1, align 4
%6 = getelementptr inbounds i32* %5, i32 %4
%7 = load i32* %6
store i32 %7, i32* %x, align 4
%8 = load i32* %x, align 4
%9 = load i32* %x, align 4
%10 = load i32* %x, align 4
%11 = mul nsw i32 16, %10
%12 = load i32* %x, align 4
%13 = mul nsw i32 %11, %12
%14 = sub nsw i32 %13, 20
%15 = mul nsw i32 %9, %14
%16 = load i32* %x, align 4
%17 = mul nsw i32 %15, %16
%18 = add nsw i32 %17, 5
%19 = mul nsw i32 %8, %18
%20 = load i32* %idx, align 4
%21 = load i32** %2, align 4
%22 = getelementptr inbounds i32* %21, i32 %20
store i32 %19, i32* %22
ret void
```

(c) Optimized IR of the Kernel

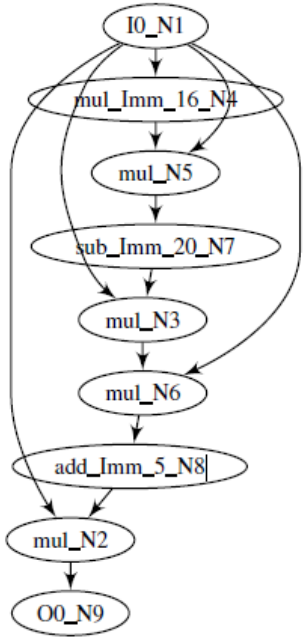
```
%0:
%1 = call i32 bitcast (i32 (...)* @get_global_id to i32 (i32)*)(i32 0)
%2 = getelementptr inbounds i32* %A, i32 %1
%3 = load i32* %2
%4 = mul nsw i32 16, %3
%5 = mul nsw i32 %4, %3
%6 = sub nsw i32 %5, 20
%7 = mul nsw i32 %3, %6
%8 = mul nsw i32 %7, %3
%9 = add nsw i32 %8, 5
%10 = mul nsw i32 %3, %9
%11 = getelementptr inbounds i32* %B, i32 %1
store i32 %10, i32* %11
ret void
```



Compilation

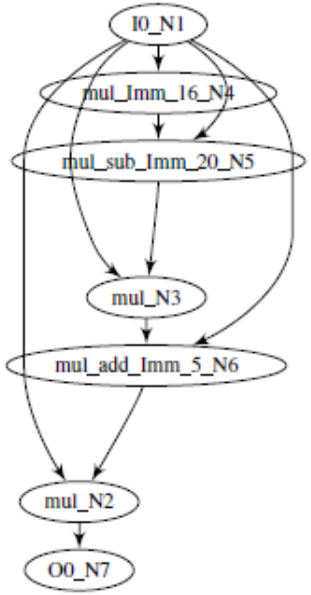
(a) DFG Description of the Kernel

```
digraph example_kernel {
N8 [ntype="operation", label="add_Imm_5_N8"];
N9 [ntype="outvar", label="O0_N9"];
N1 [ntype="invar", label="I0_N1"];
N2 [ntype="operation", label="mul_N2"];
N3 [ntype="operation", label="mul_N3"];
N4 [ntype="operation", label="mul_Imm_16_N4"];
N5 [ntype="operation", label="mul_N5"];
N6 [ntype="operation", label="mul_N6"];
N7 [ntype="operation", label="sub_Imm_20_N7"];
N8 -> N2;
N1 -> N5;
N1 -> N6;
N1 -> N2;
N1 -> N3;
N1 -> N4;
N2 -> N9;
N3 -> N6;
N4 -> N5;
N5 -> N7;
N6 -> N8;
N7 -> N3;
}
```

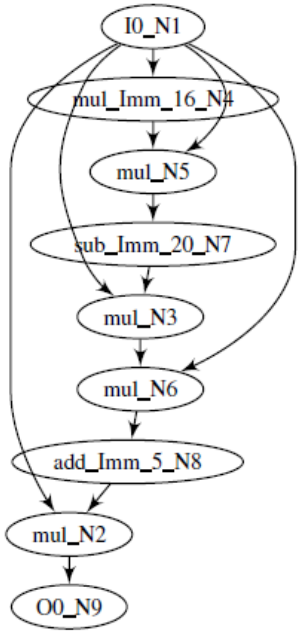


(b) FU-aware DFG Description of the Kernel

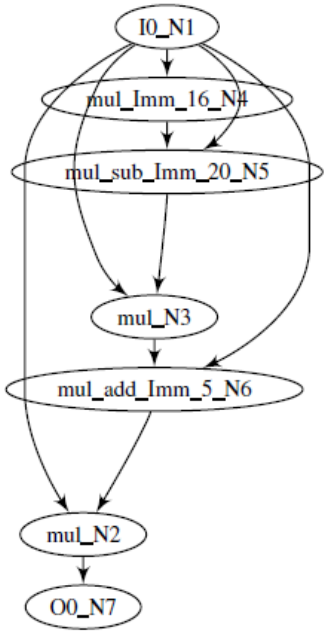
```
digraph example_kernel {
N7 [ntype="outvar", label="O0_N7"];
N1 [ntype="invar", label="I0_N1"];
N2 [ntype="operation", label="mul_N2"];
N3 [ntype="operation", label="mul_N3"];
N4 [ntype="operation", label="mul_Imm_16_N4"];
N5 [ntype="operation", label="mul_sub_Imm_20_N5"];
N6 [ntype="operation", label="mul_add_Imm_5_N6"];
N1 -> N5;
N1 -> N6;
N1 -> N2;
N1 -> N3;
N1 -> N4;
N2 -> N7;
N3 -> N6;
N4 -> N5;
N5 -> N3;
N6 -> N2;
}
```



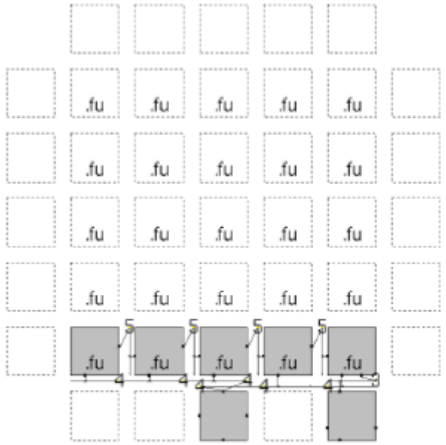
Compilation



(a) DFG extracted from Kernel

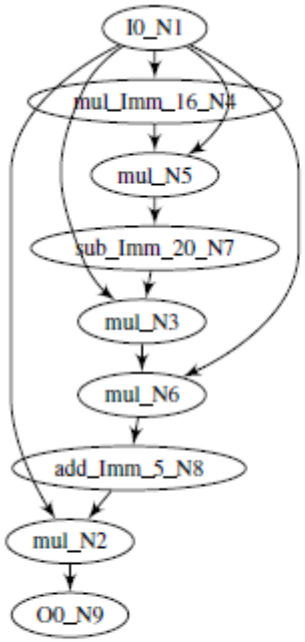


(b) FU-aware DFG where FU consists of one DSP block

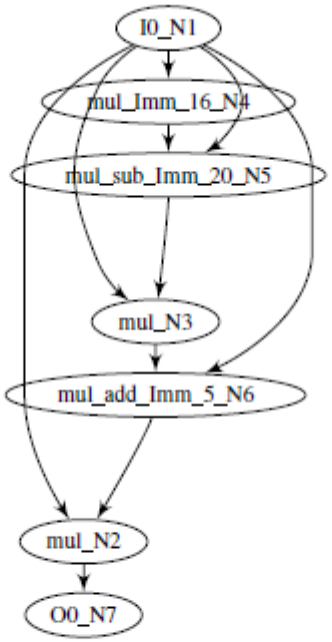


(c) FU-aware DFG placed and routed on 5x5 overlay

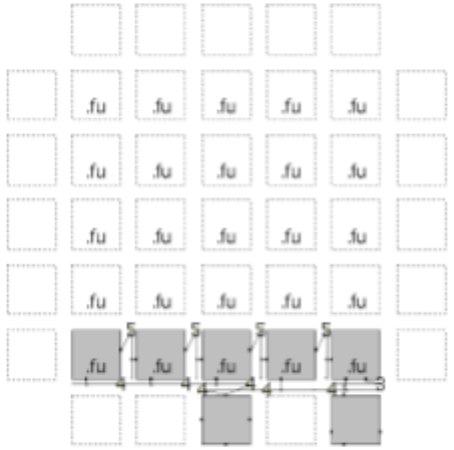
Compilation



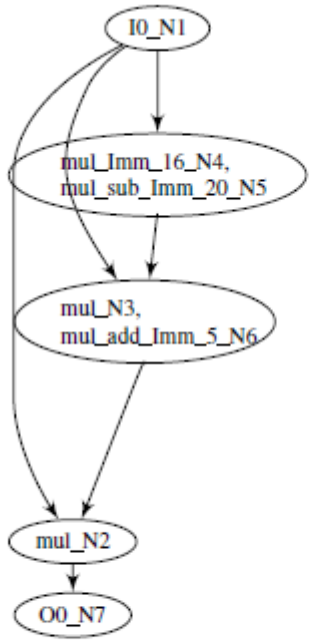
(a) DFG extracted from Kernel



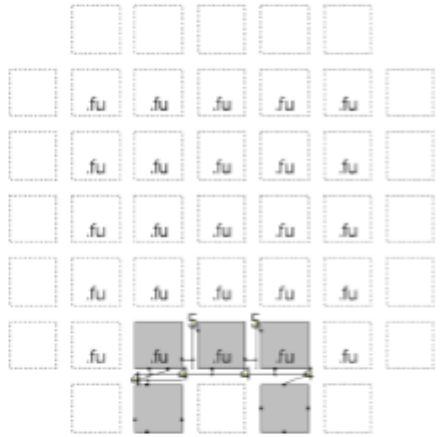
(b) FU-aware DFG where FU consists of one DSP block



(c) FU-aware DFG placed and routed on 5x5 overlay

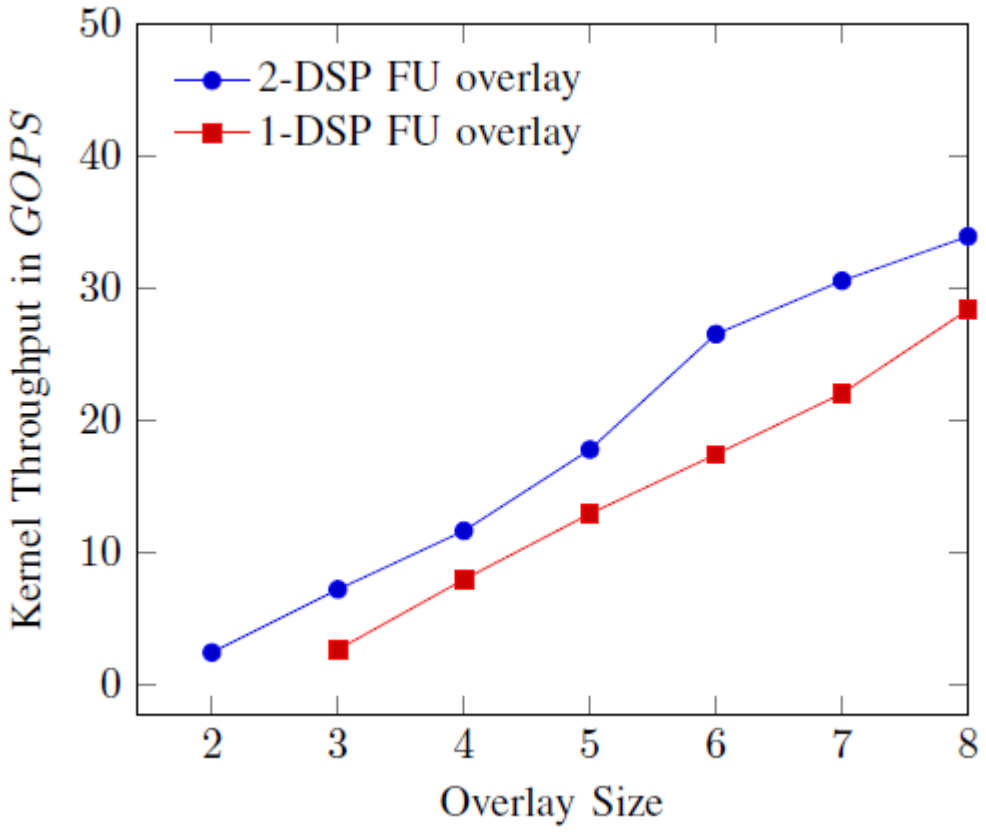


(d) FU-aware DFG where FU consists of two DSP blocks



(e) FU-aware DFG placed and routed on 5x5 overlay

Resource-aware Performance Scaling

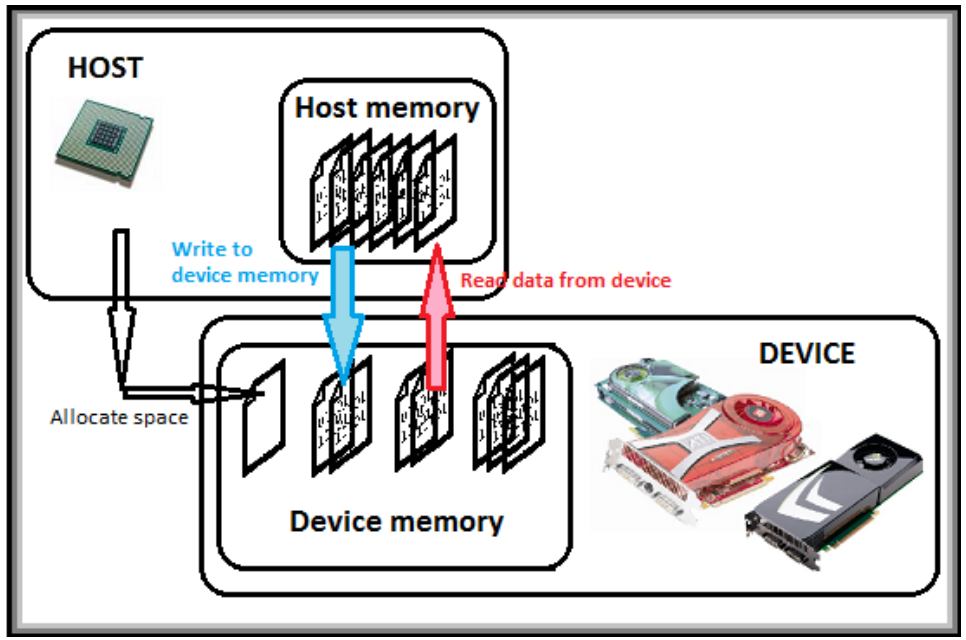


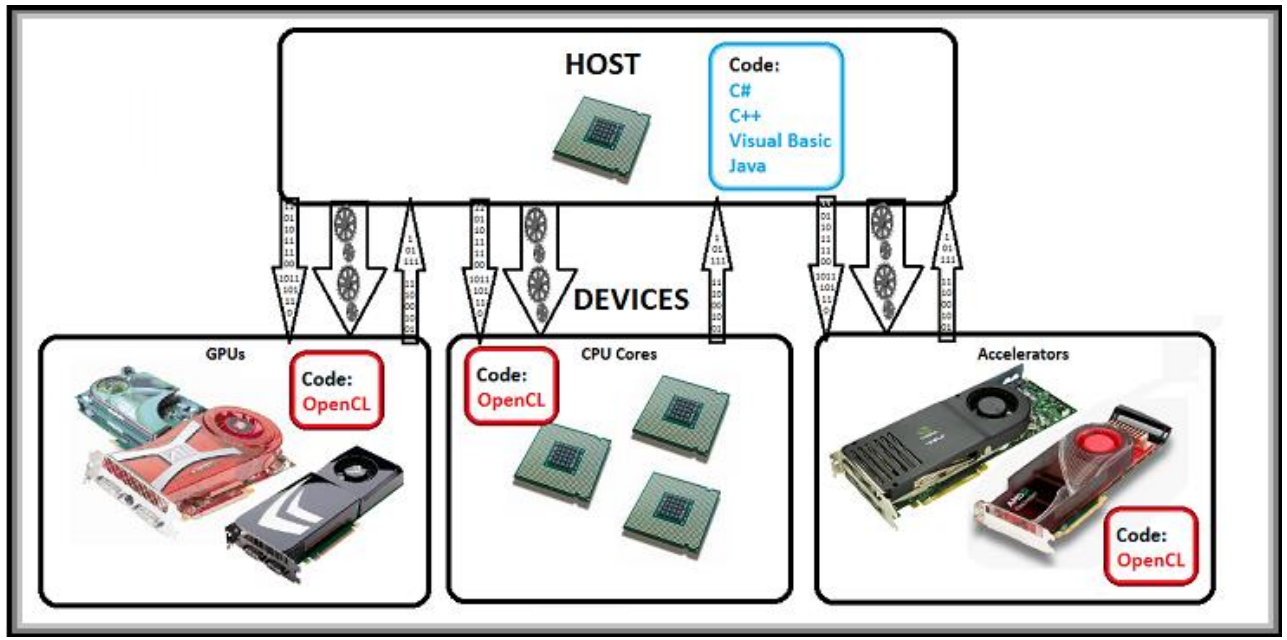
Kernels

1.	chebyshev	$out = (x * (x * (16 * x^2 - 20) * x + 5))$
2.	sgfilter	$out = (x * (x * (7 * x - 76 * y + 7) + y * (92 * y - 39) + 7) - y * (y * (984 * y + 46) + 46) - 75)$
3.	mibench	$out = (x * (x + 2 * y + 6 * z + 43) + y * (y + 6 * z + 43) + z * (9 * z + 1))$
4.	qspline	$out = (z * u^4 + 4 * a * u^3 * v + 6 * b * u^2 * v^2 + 4 * w * v^3 * u + q * v^4)$
5.	poly1	$out = (x^2 * (x + y - 1) - y^2 * (x - y + 1))$
6.	poly2	$out = (x^2 * (2 * x^2 - 3 * y) + (y - 1)^2 * y^2)$

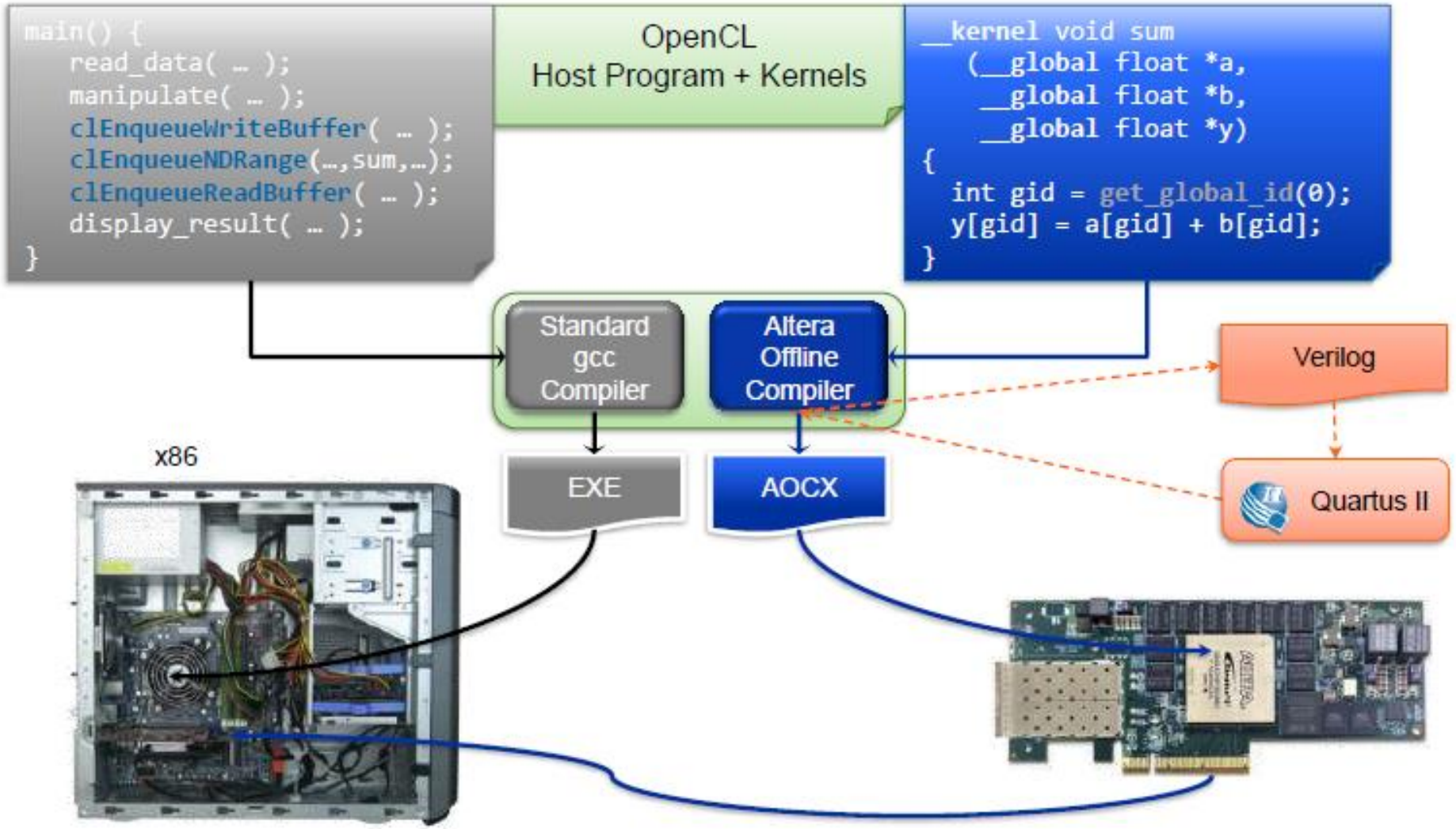
Runtime Management of Accelerators using OpenCL

configure → Data Write → Trigger for processing → Data Read

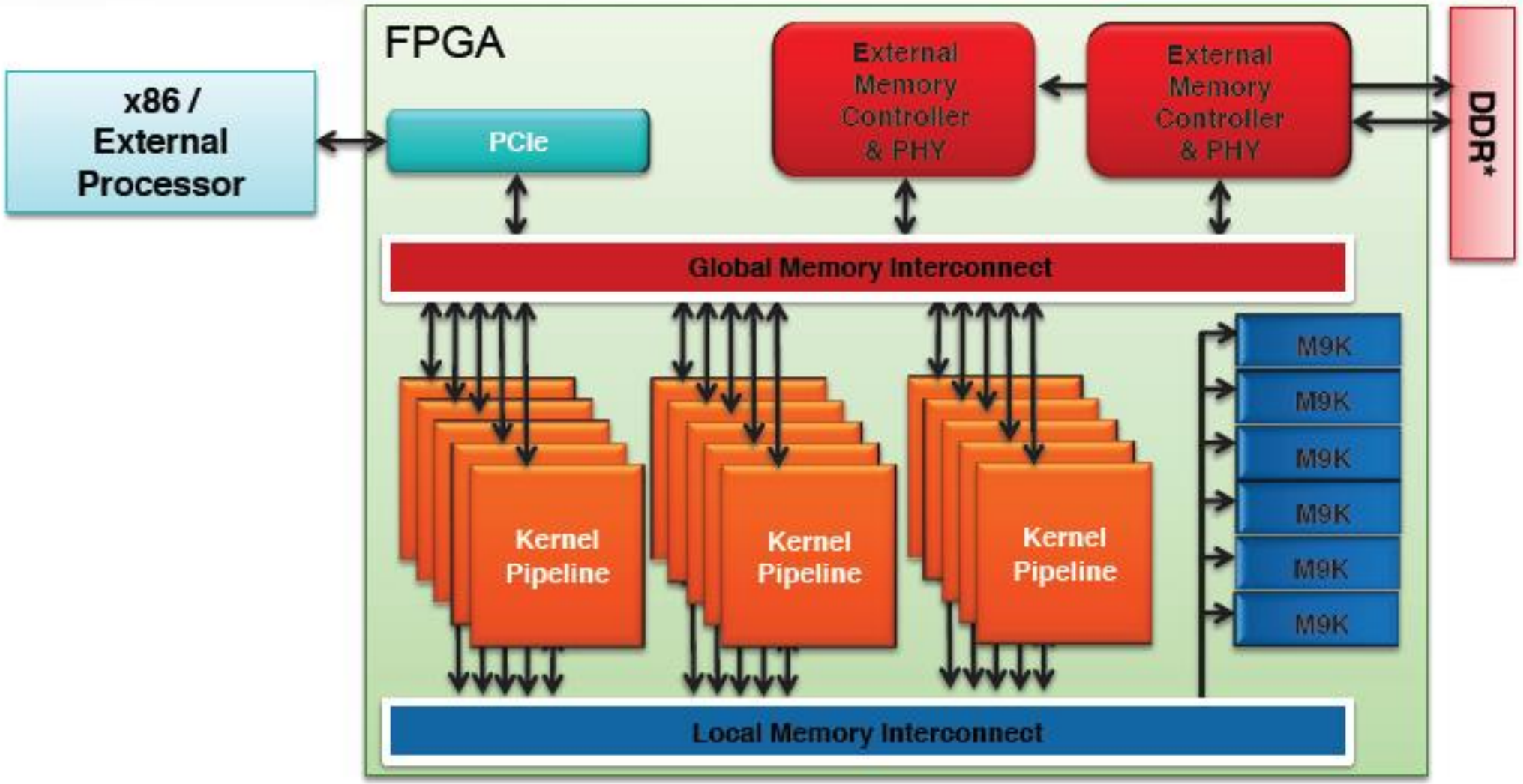




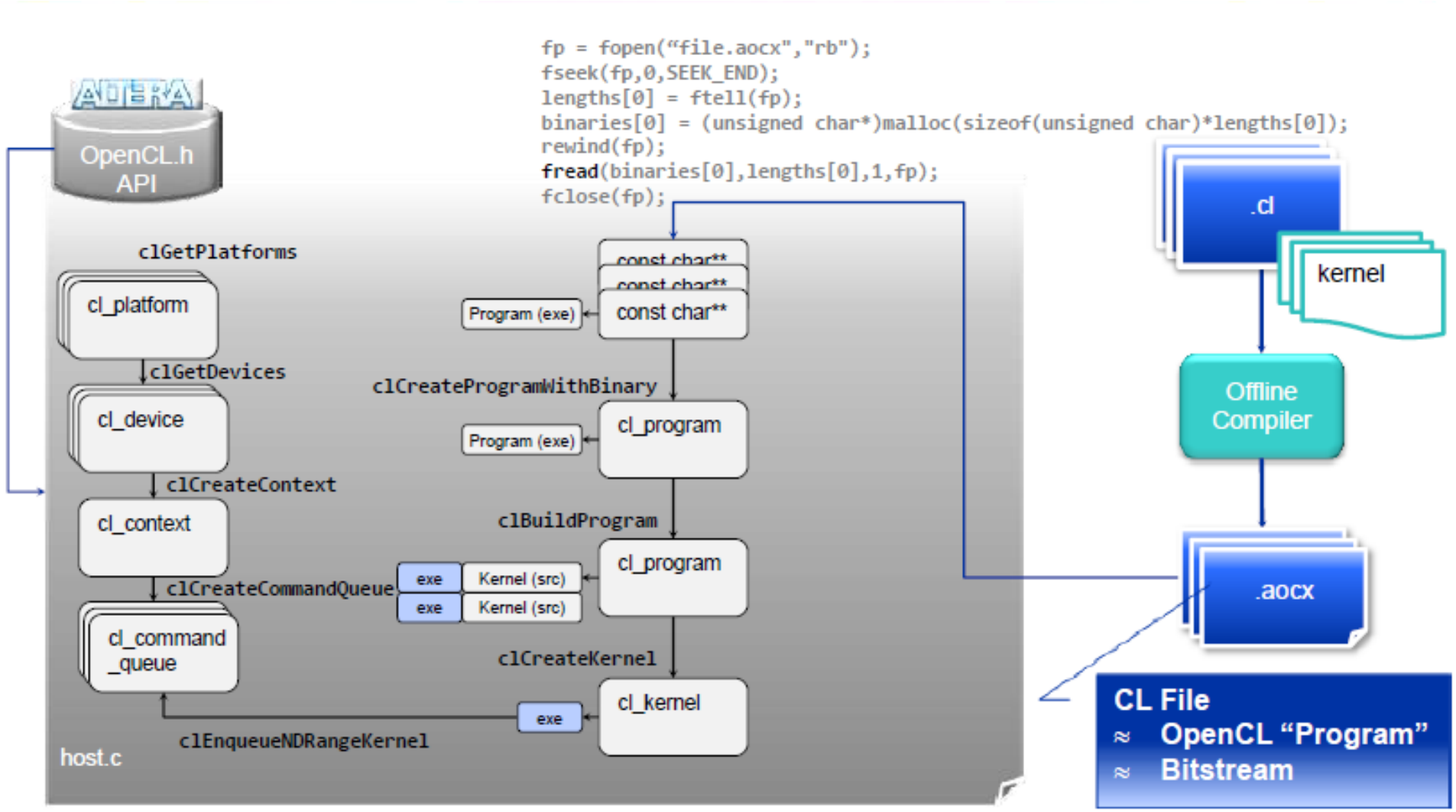
OpenCL as a programming model



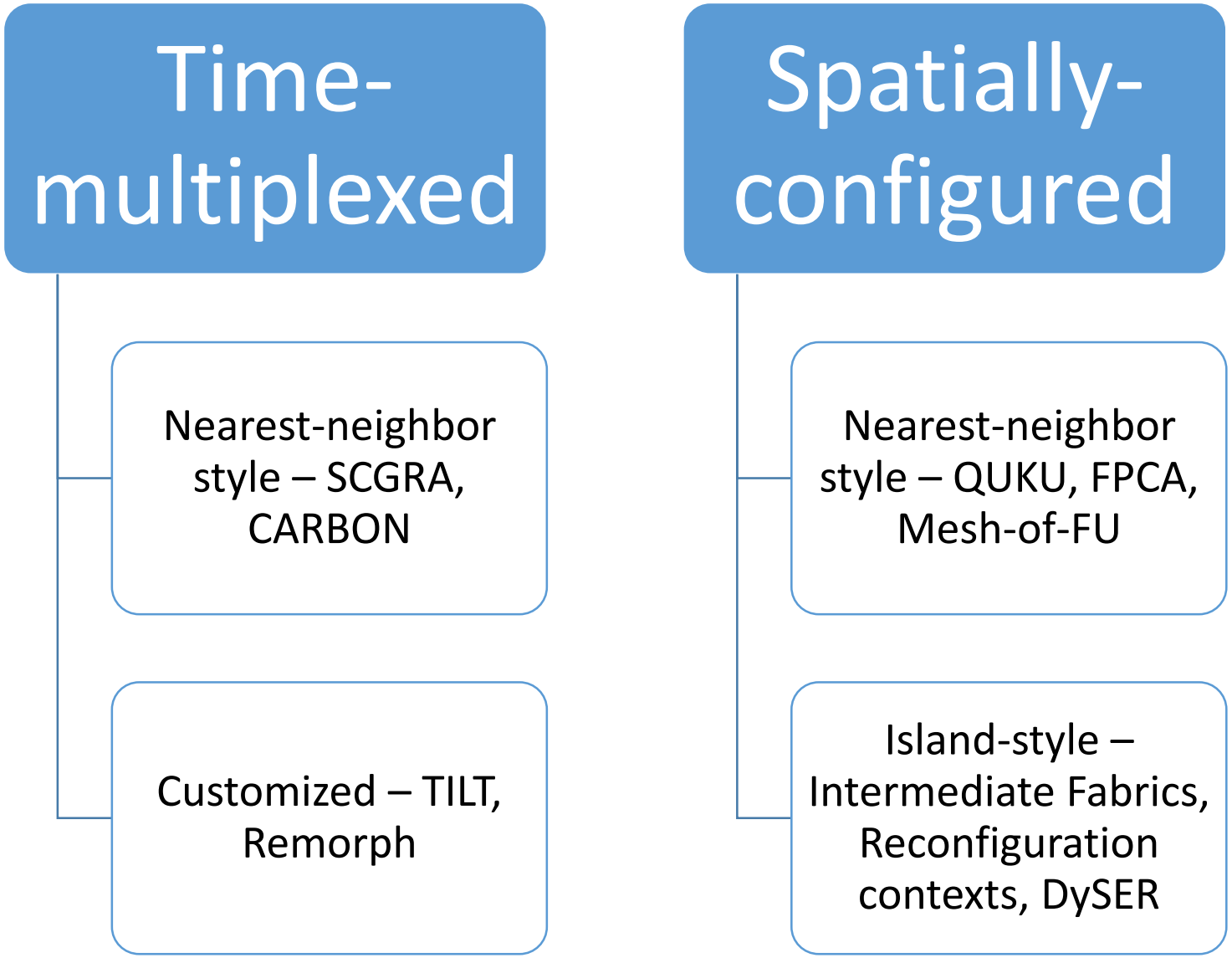
OpenCL as a programming model



OpenCL as a programming model



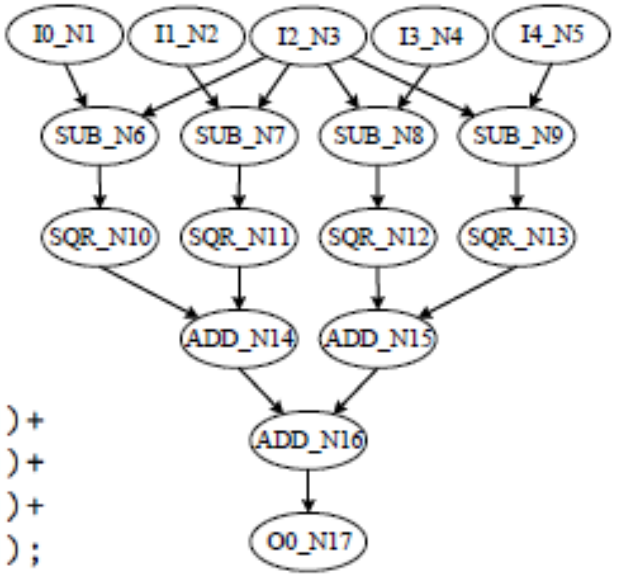
Overlays


Time-multiplexed
Nearest-neighbor style – SCGRA, CARBON
Customized – TILT, Remorph
Spatially-configured
Nearest-neighbor style – QUKU, FPCA, Mesh-of-FU
Island-style – Intermediate Fabrics, Reconfiguration contexts, DySER

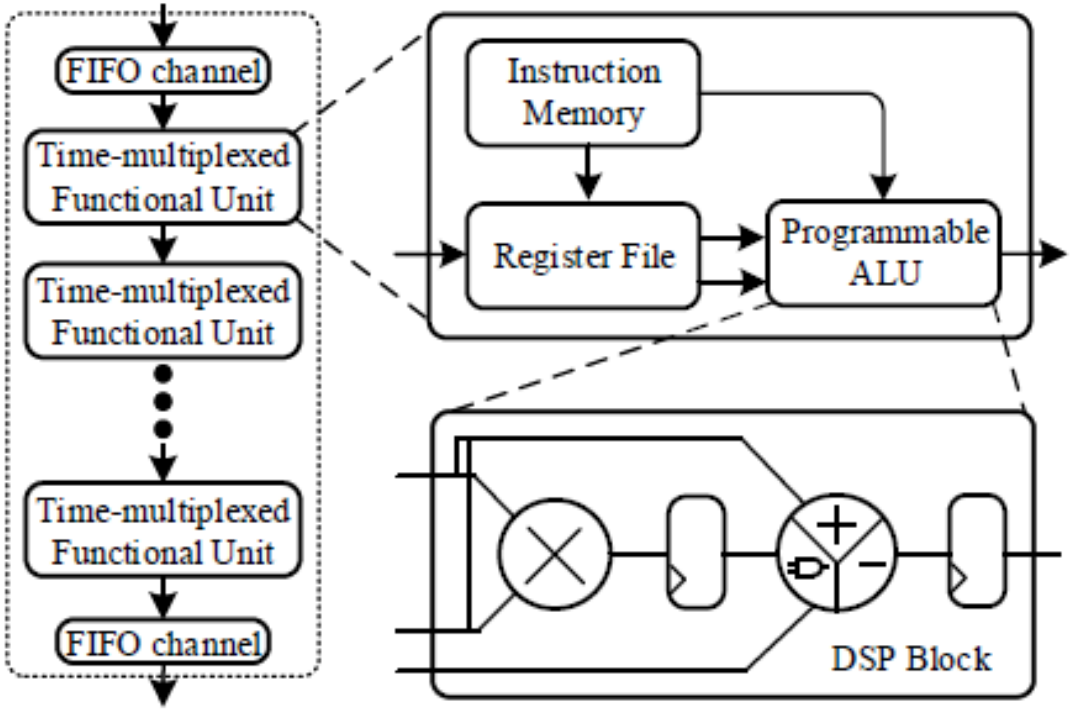
Concept of Time-multiplexing

```
#define SQR(x) ((x)*(x))
for(int i=0;i<64;i++)
  for(int j=1;j<63;j++)
    for(int k=1;k<63;k++)
      b[i][j][k] =
      SQR(a[i][j][k]-a[i][j][k-1])+
      SQR(a[i][j][k]-a[i][j][k+1])+
      SQR(a[i][j][k]-a[i][j-1][k])+
      SQR(a[i][j][k]-a[i][j+1][k]);
```

(a) C Source Code



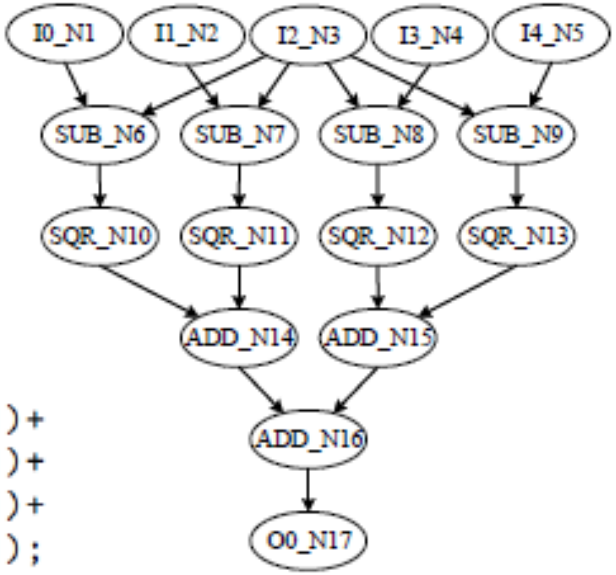
(b) Data Flow Graph



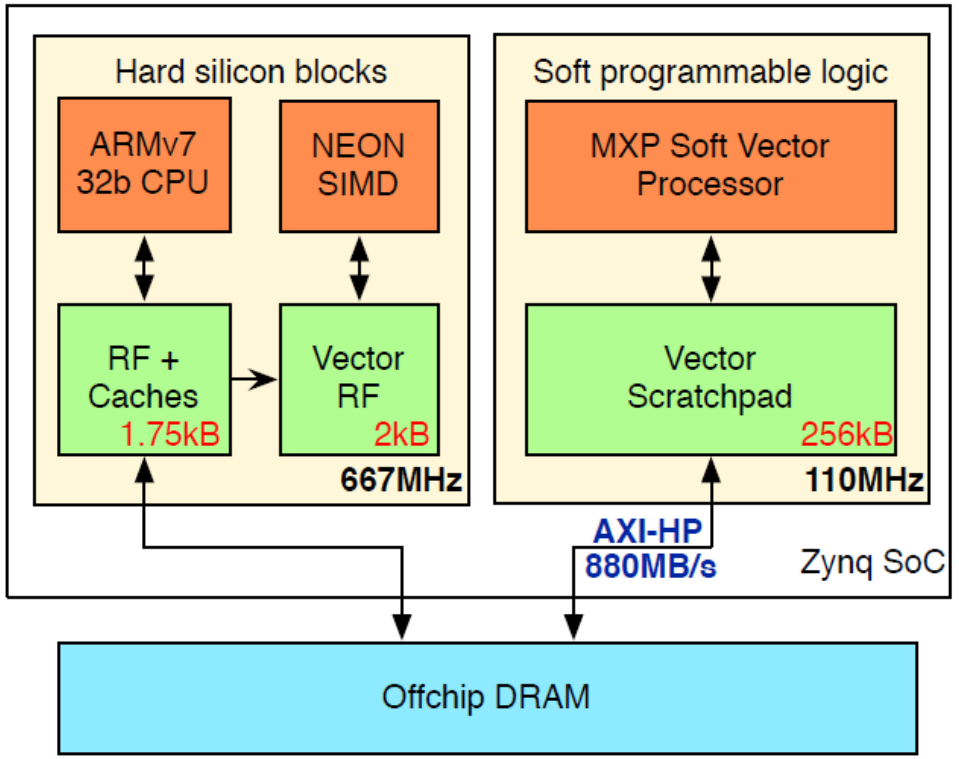
Time-multiplexed Overlays: VectorBlox MXP

```
#define SQR(x) ((x)*(x))
for(int i=0;i<64;i++)
  for(int j=1;j<63;j++)
    for(int k=1;k<63;k++)
      b[i][j][k] =
      SQR(a[i][j][k]-a[i][j][k-1])+
      SQR(a[i][j][k]-a[i][j][k+1])+
      SQR(a[i][j][k]-a[i][j-1][k])+
      SQR(a[i][j][k]-a[i][j+1][k]);
```

(a) C Source Code



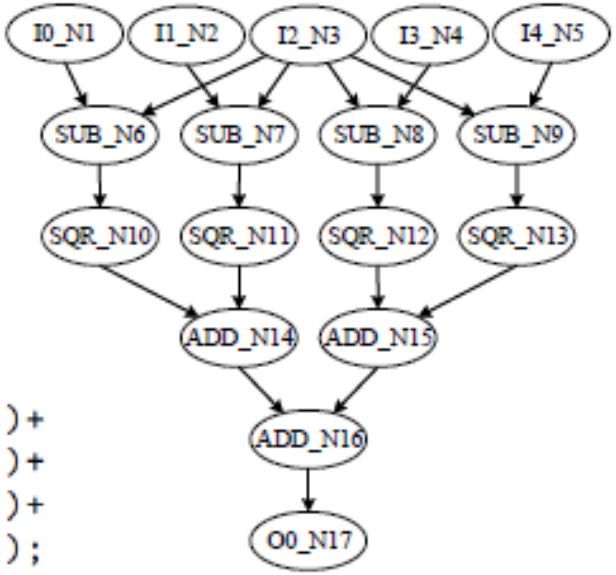
(b) Data Flow Graph



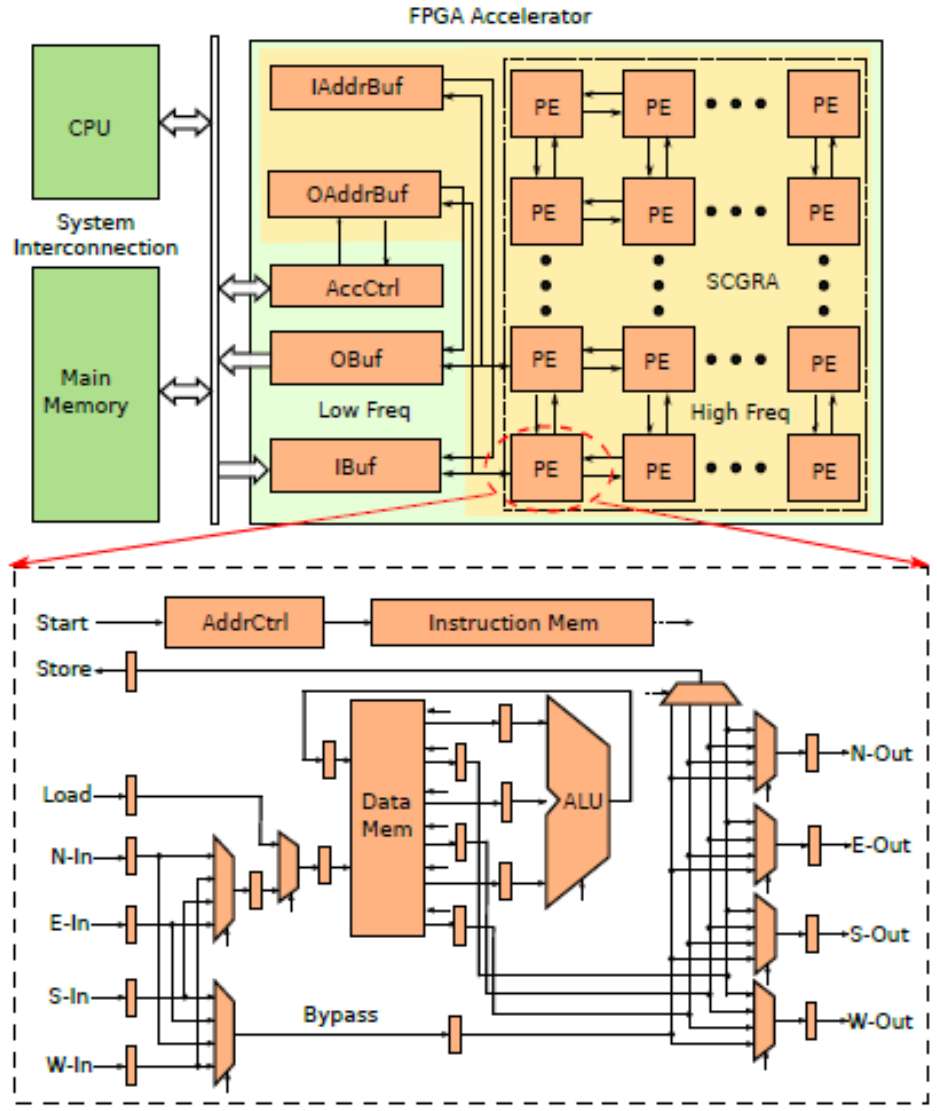
Time-multiplexed Overlays: SCGRA

```
#define SQR(x) ((x)*(x))
for(int i=0;i<64;i++)
  for(int j=1;j<63;j++)
    for(int k=1;k<63;k++)
      b[i][j][k] =
      SQR(a[i][j][k]-a[i][j][k-1])+
      SQR(a[i][j][k]-a[i][j][k+1])+
      SQR(a[i][j][k]-a[i][j-1][k])+
      SQR(a[i][j][k]-a[i][j+1][k]);
```

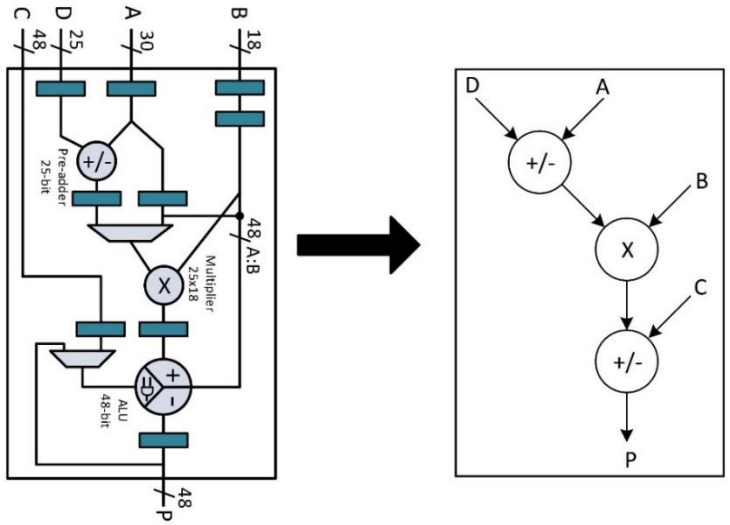
(a) C Source Code

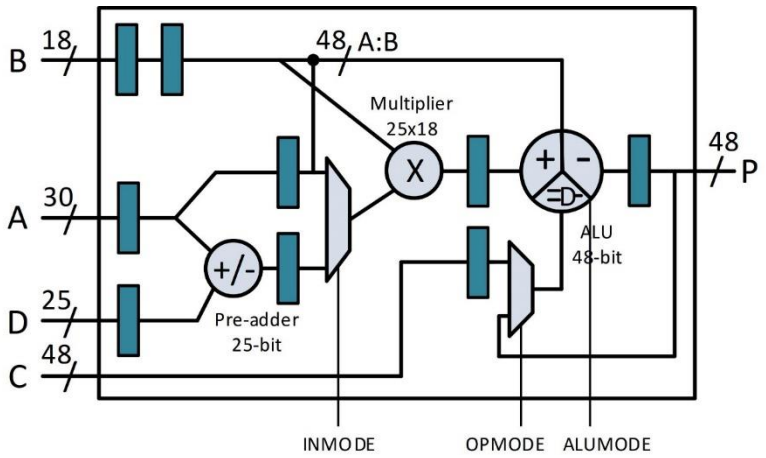


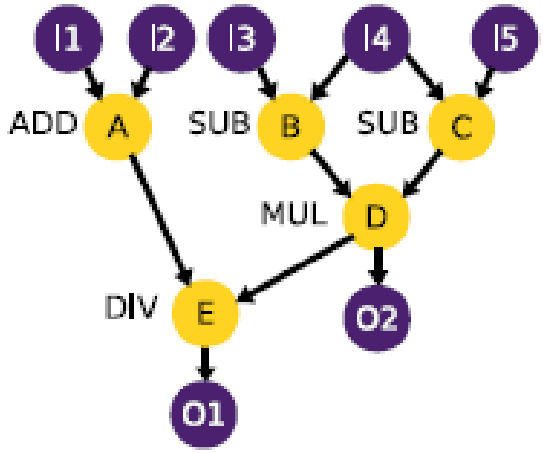
(b) Data Flow Graph

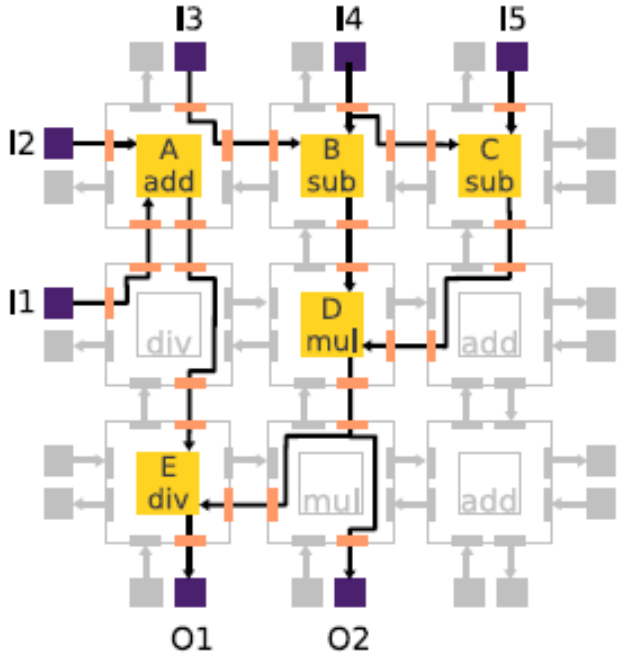


Spatially-Configured Overlays



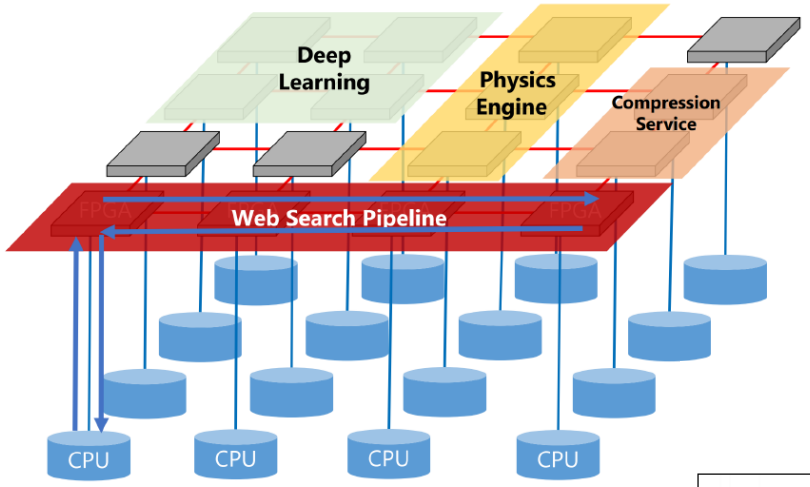




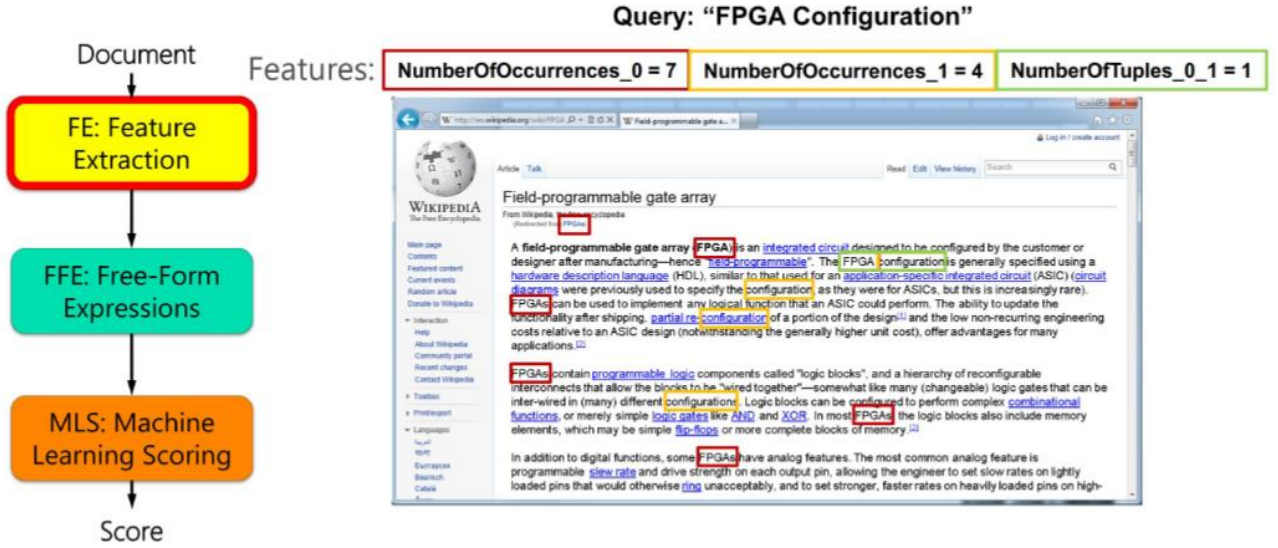


Success Story

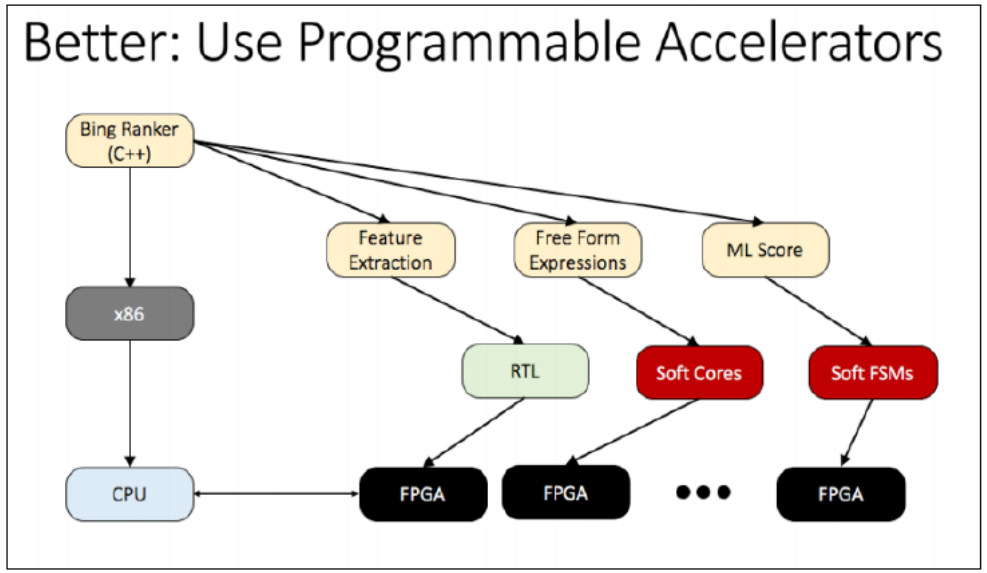
Catapult: An Elastic Reconfigurable Fabric for Datacenters



Catapult Bing search query ranking

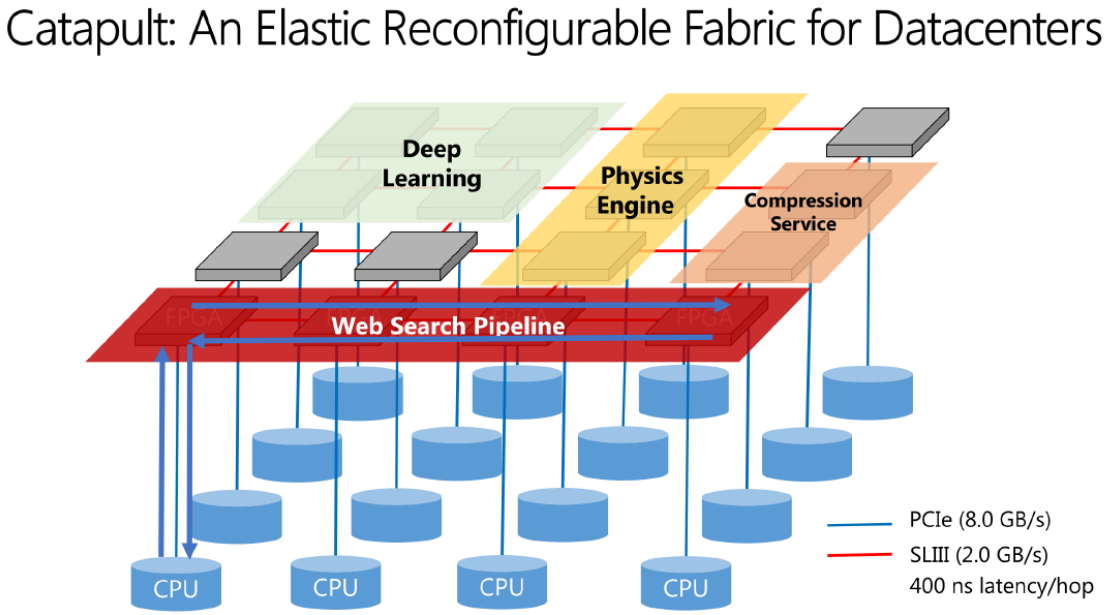


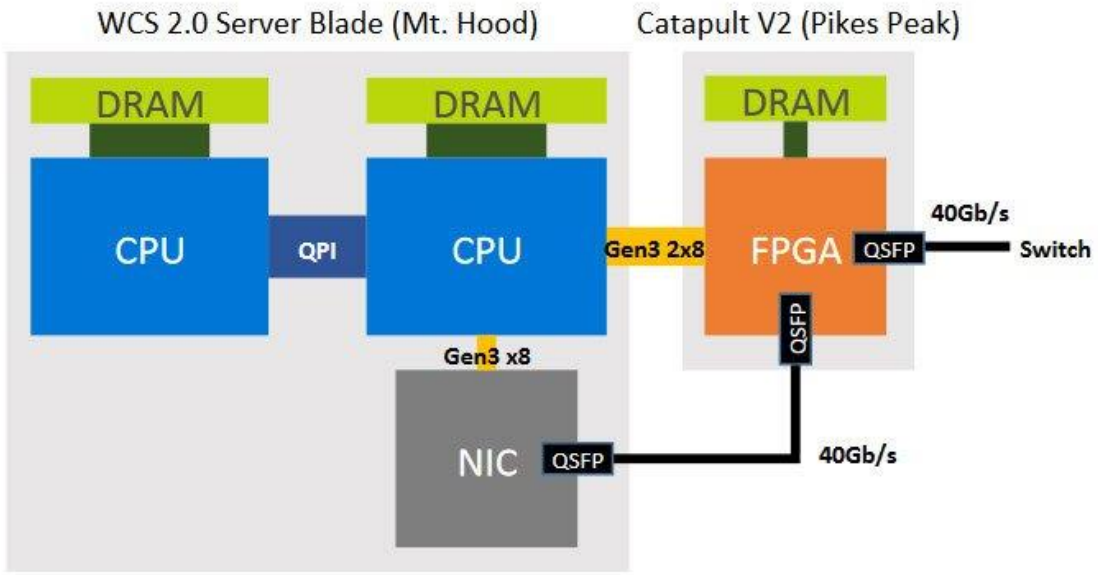
Catapult Experience



FPGAs in Cloud

- 1/3rd of the cloud service provider nodes to use FPGAs by 2020
- Microsoft: Doubling the throughput of Bing search engine using FPGAs
- Microsoft Azure Cloud services and Amazon EC2 (FPGA-backed F1 instances)

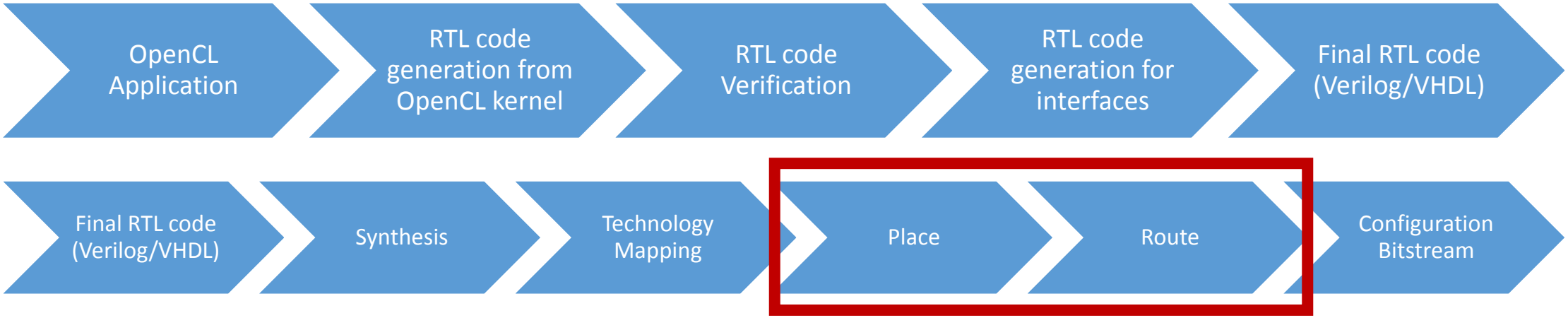




- *Nicole Hemsoth, " The FPGA Accelerated Cloud Push Just Got Stronger," 30 November 2016, THENEXTPLATFORM*

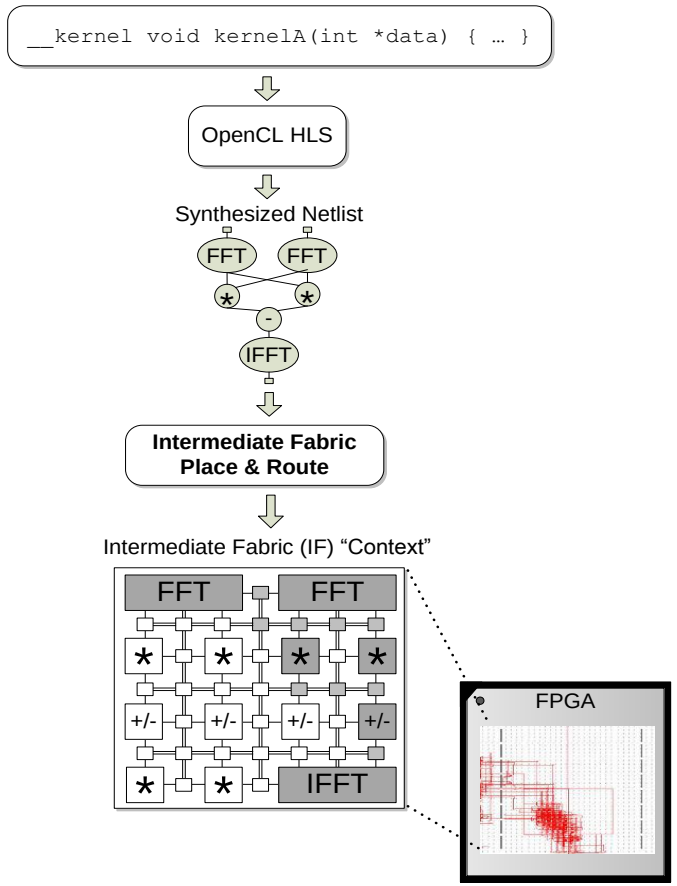
Main issue?

- Long compilation times (specifically place and route times)

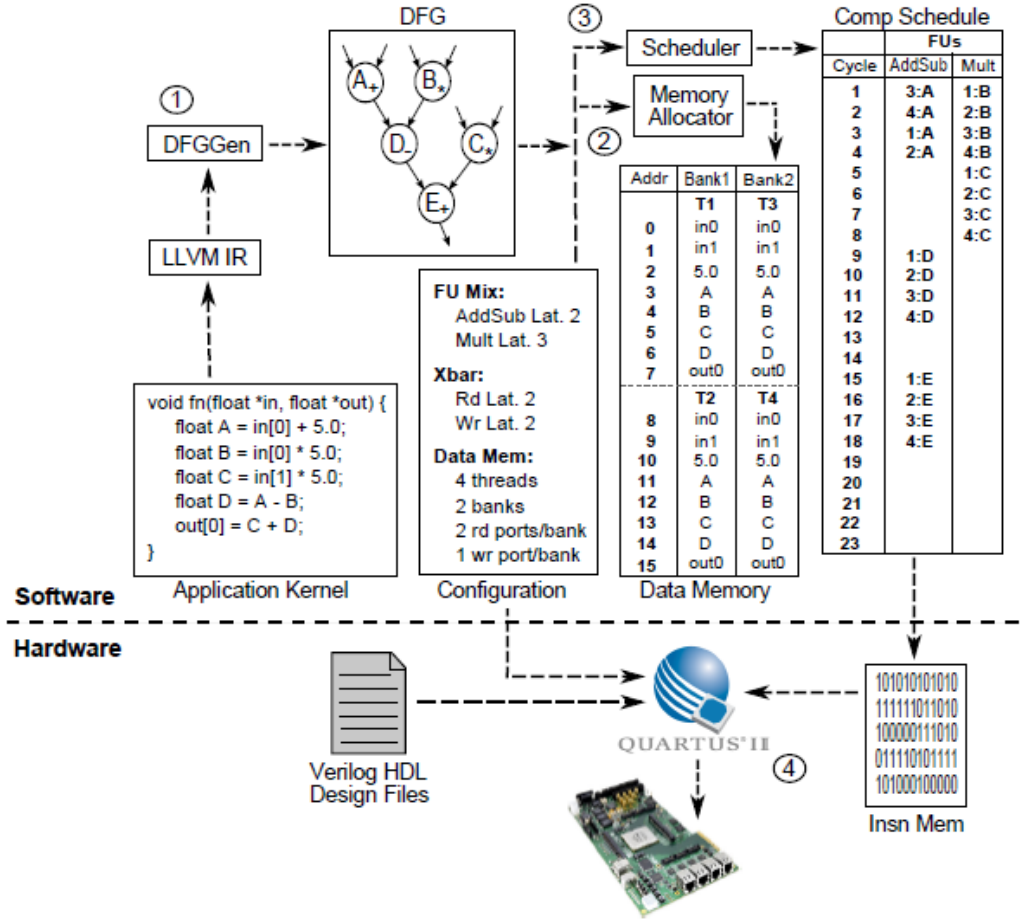


- Need for not only software-like abstractions but also fast development cycles
  - Similar to application development for GPUs
  - Resource aware Just-in-Time compilation can enable performance scaling

OpenCL compiler for Coarse-grained Overlays

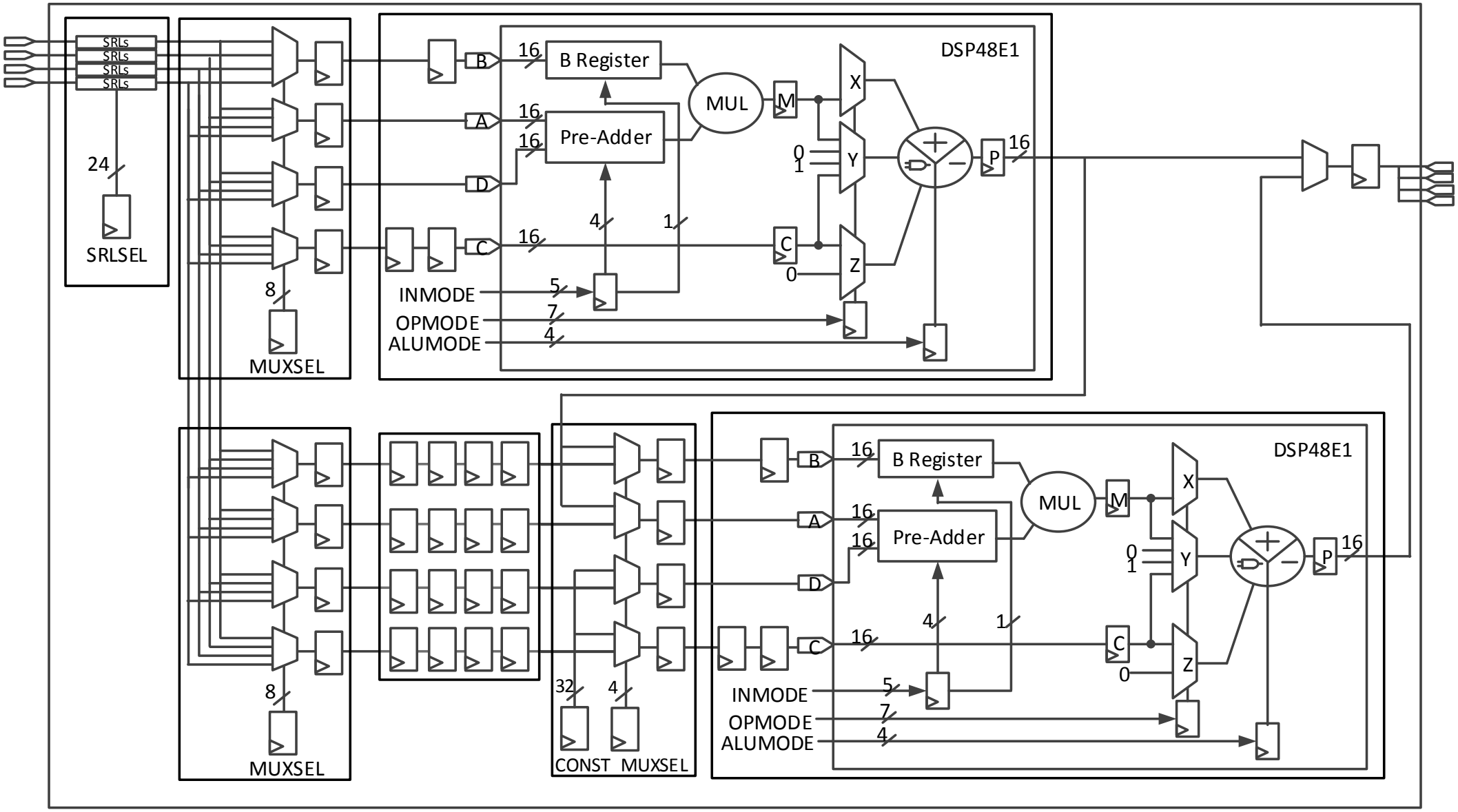


Intermediate Fabric Overlay (University of Florida)

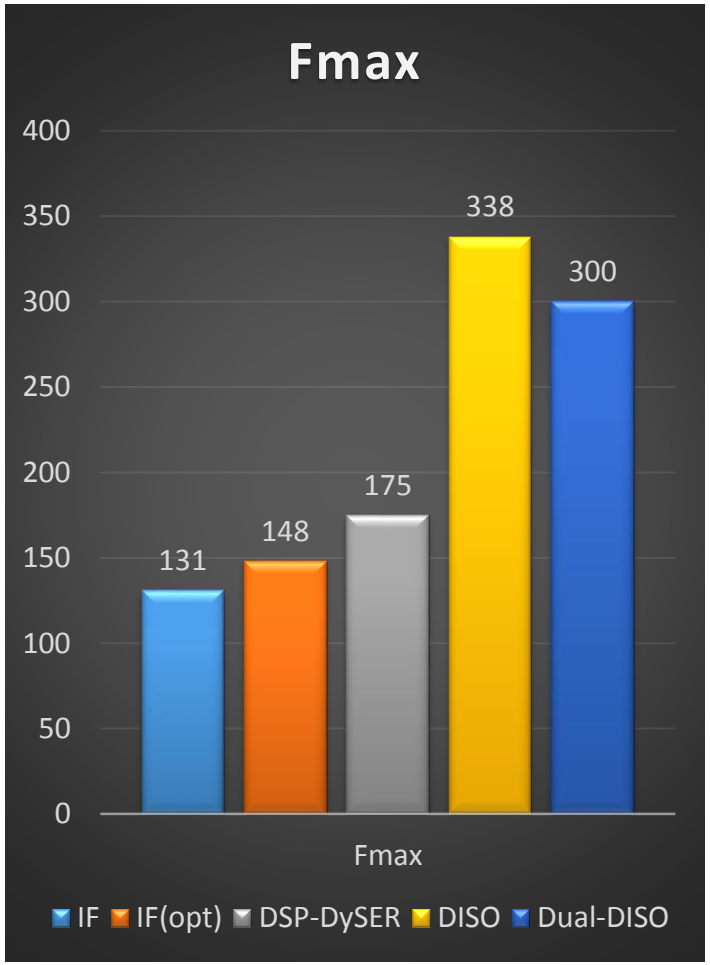


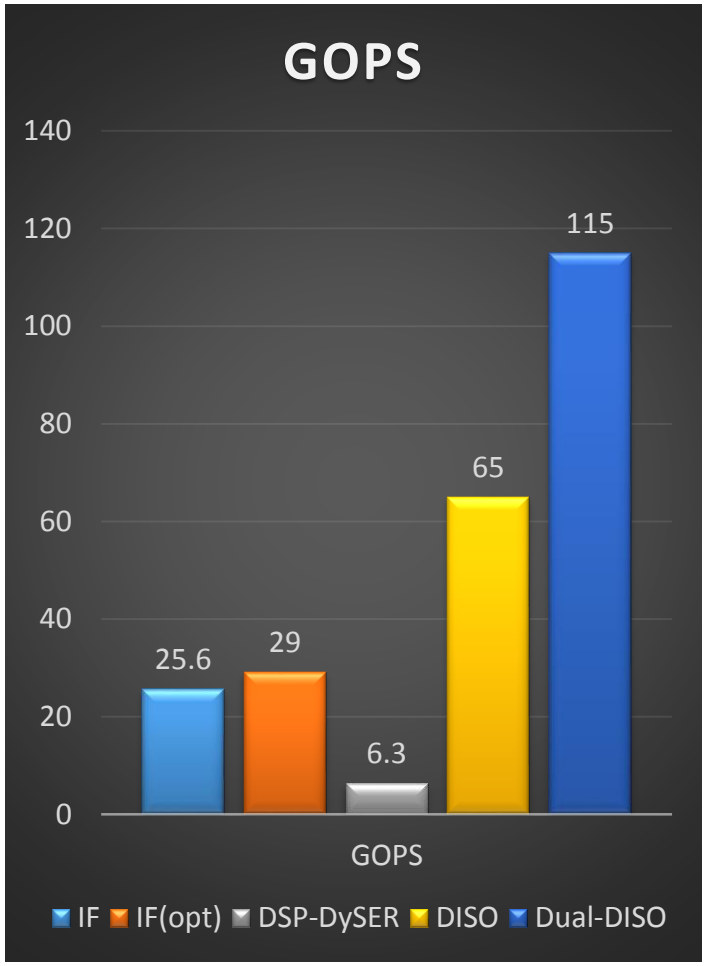
TILT Overlay (University of Toronto)

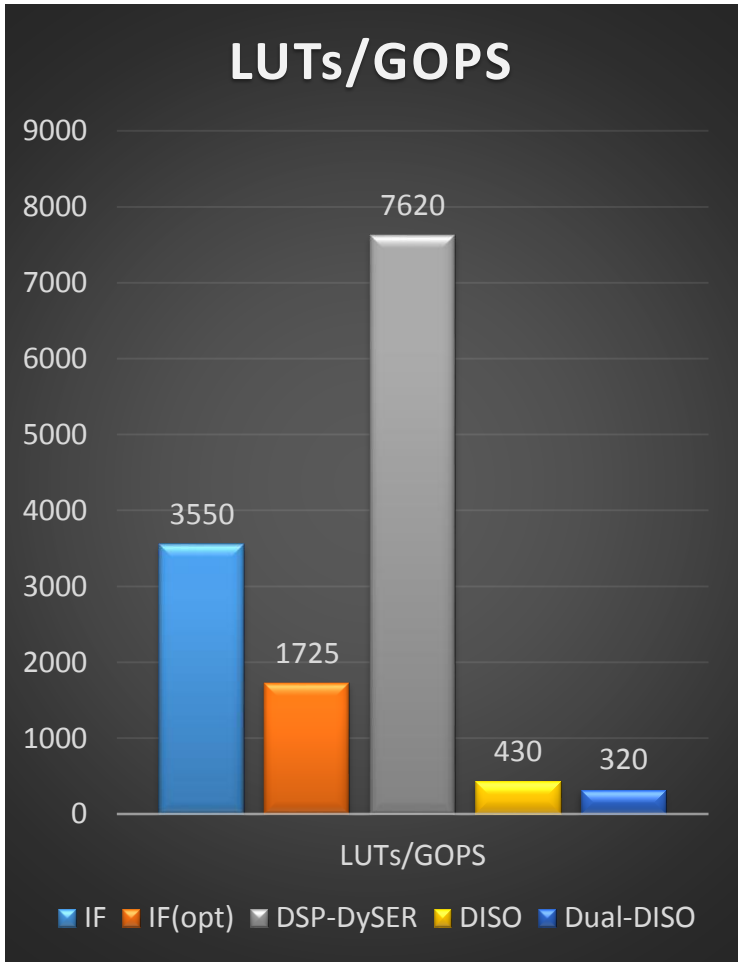
Dual-DISO Functional Unit



Summary







- *Landy, Aaron, and Greg Stitt. "A low-overhead interconnect architecture for virtual reconfigurable fabrics." ACM CASES 2012.*