UNIVERSITÄT BONN

Gas Det Bonn

Physikalisches Institut

GEFÖRDERT VOM

Bundesministerium für Bildung und Forschung

IMPLEMENTATION OF A PATTERN RECOGNITION NEURAL NETWORK FOR LIVE RECONSTRUCTION USING AI PROCESSORS

Michael Lupberger

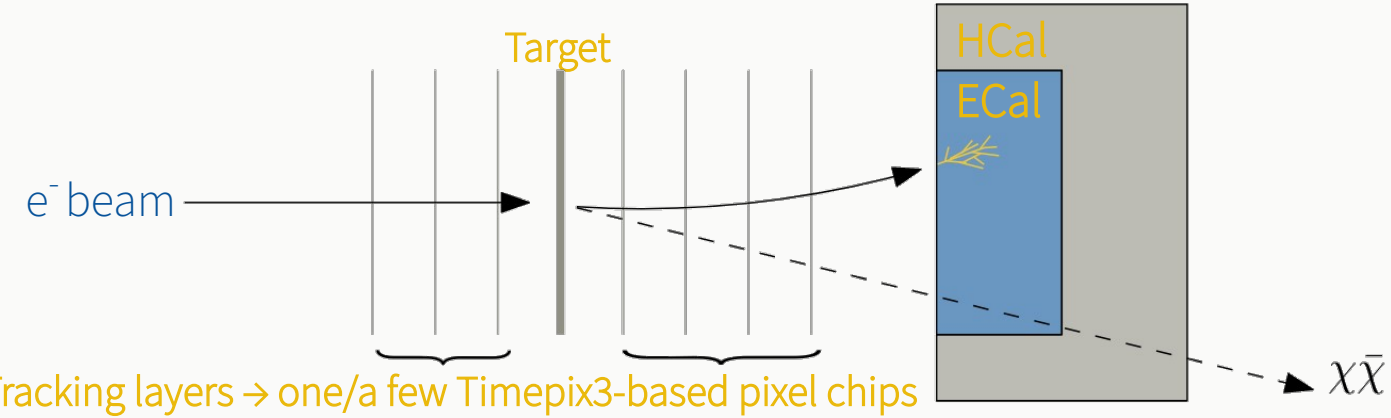
Patrick Schwäbig

10/03/2022

UNIVERSITÄT BONN

MOTIVATION

- Dark photon experiment, electron on target, based proposal by LDMX collaboration
- To be built at the ELSA accelerator at University of Bonn



- Low signal expected, want to measure 4×10^{14} electrons

  → Need sophisticated, fast trigger (~few microseconds)

  Use hardware accelerated algorithms for the trigger → live tracking(?)

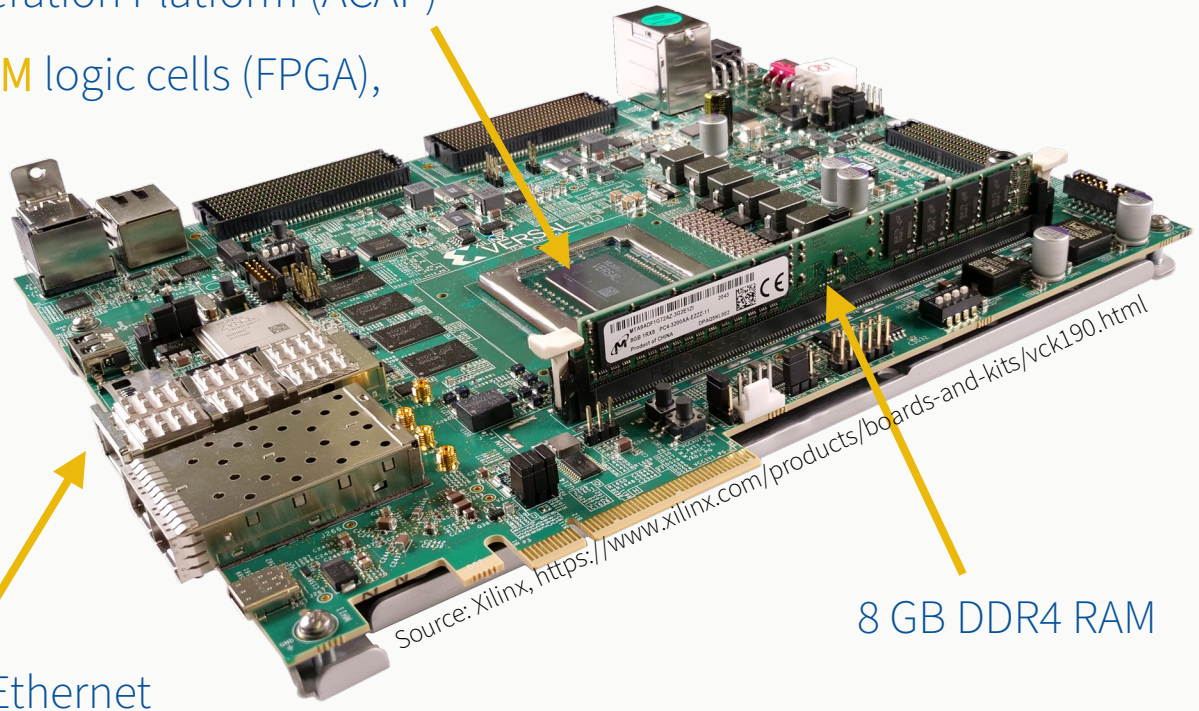
- This talk: implementation of pattern recognition using Versal AI Engines

UNIVERSITÄT BONN

HARDWARE: XILINX VERSAL / VCK190

- Evaluation board VCK190
- Versal VC1902 Adaptive Compute Acceleration Platform (ACAP)
- 400 AI processors (“AI engines”/AIEs), ~2M logic cells (FPGA), 2k DSPs, Arm CPU, Arm RPU

- AI Engines:
  - Programmed in C/C++
  - Running at 1 GHz

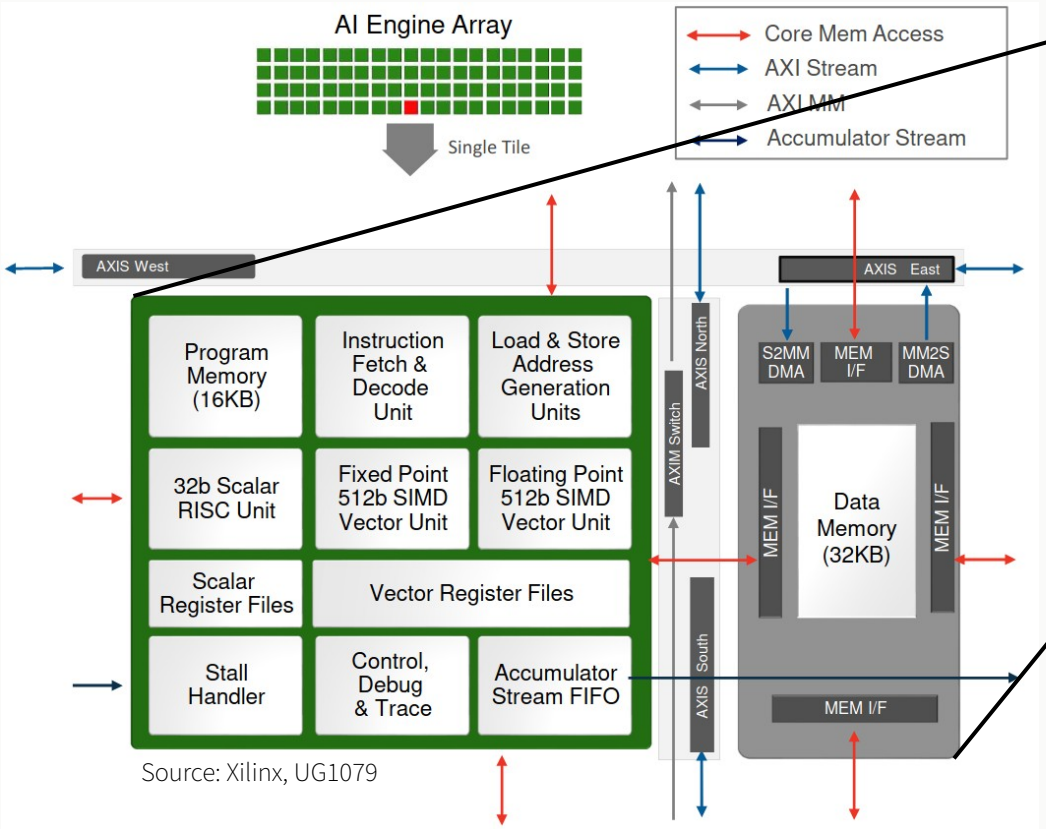


Source: Xilinx, https://www.xilinx.com/products/boards-and-kits/vck190.html

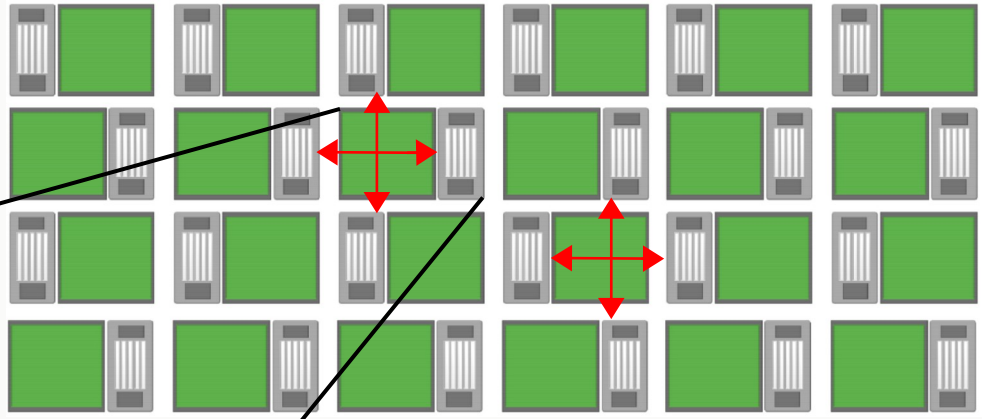
QSFP28 for 100 Gbit Ethernet

8 GB DDR4 RAM

AI ENGINES



Source: Xilinx, UG1079



Source: Xilinx

Access to “own” memory (32 KB)

+ North, South, East **or** West memory (3x 32KB)

→ 128 KB, seen as one contiguous memory

UNIVERSITÄT BONN

AI ENGINE UNITS

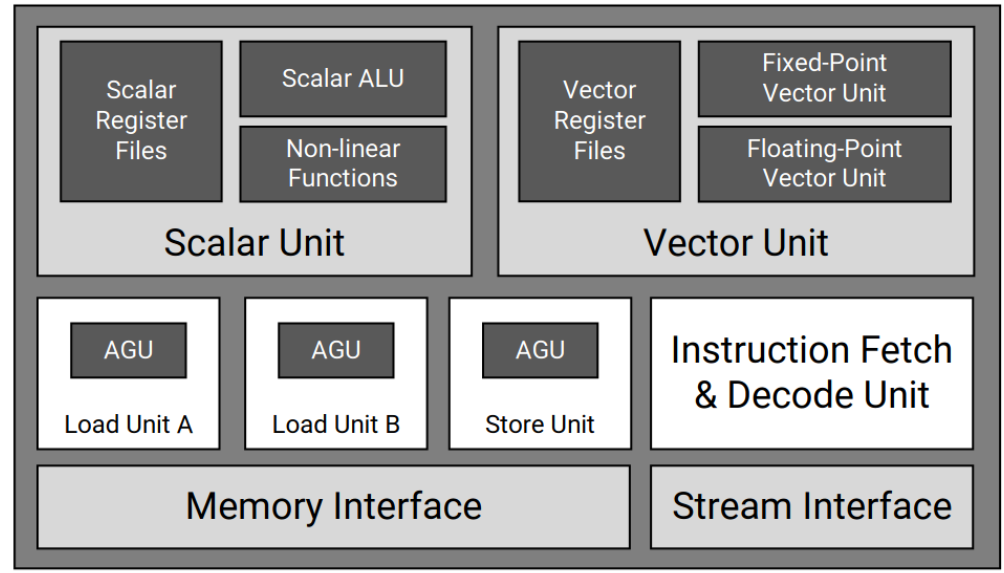
AI Engines:

- Running at 1 GHz (VCK190)
- Highly parallelized: 400 processors, VLIW, SIMD
- VLIW (Very long instruction word)

  → Simultaneous execution of:

  2x data load 1x data store

  2x data move 1x scalar operation



Source: Xilinx UG1079

1x vector operation → SIMD (Single instruction, multiple data) → ILP (Instruction level parallelism)

with multiple accumulators:

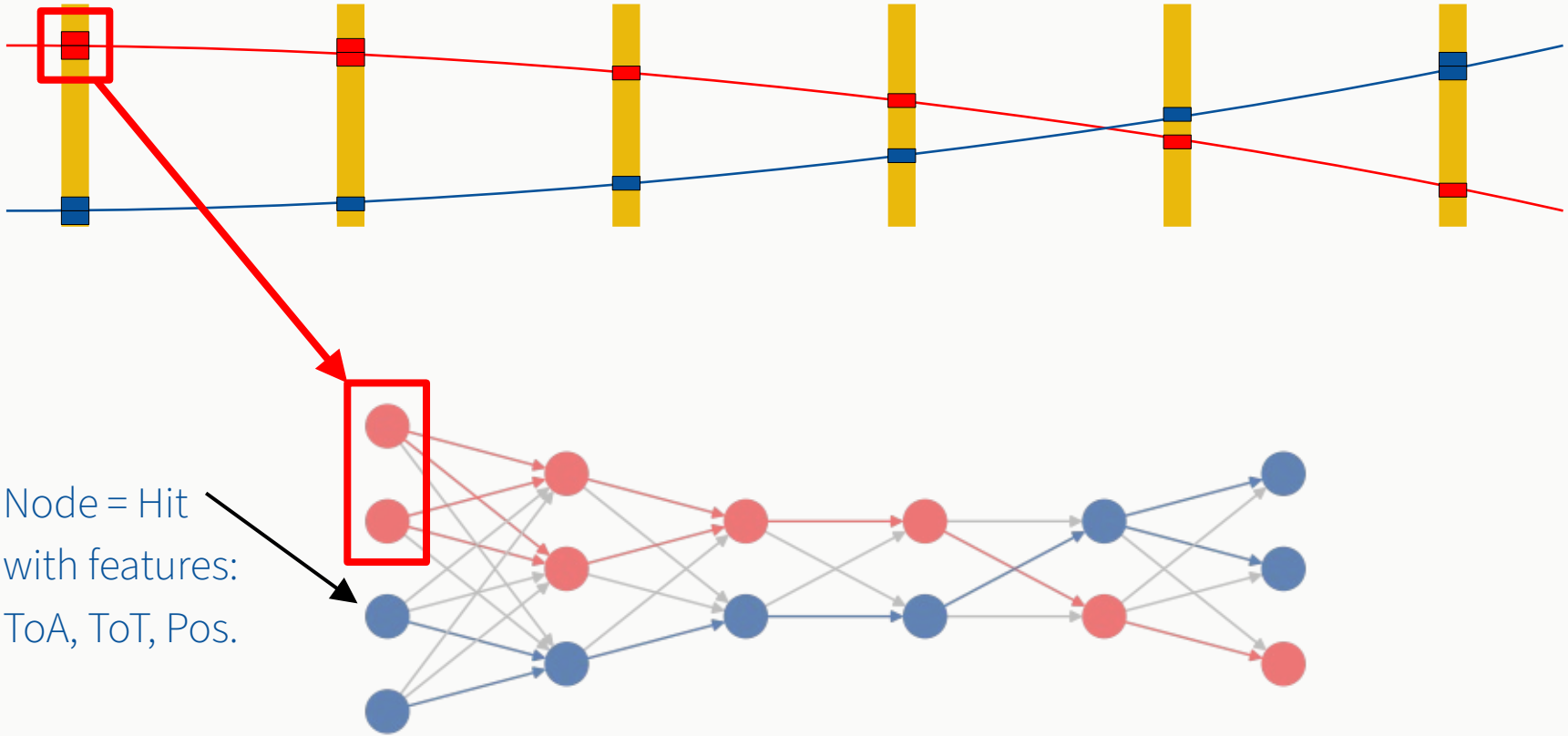
8 bit x 8 bit: 128 MAC/instruction

16 bit x 16 bit: 32 MAC/instruction

32 bit x 32 bit: 8 MAC/instruction

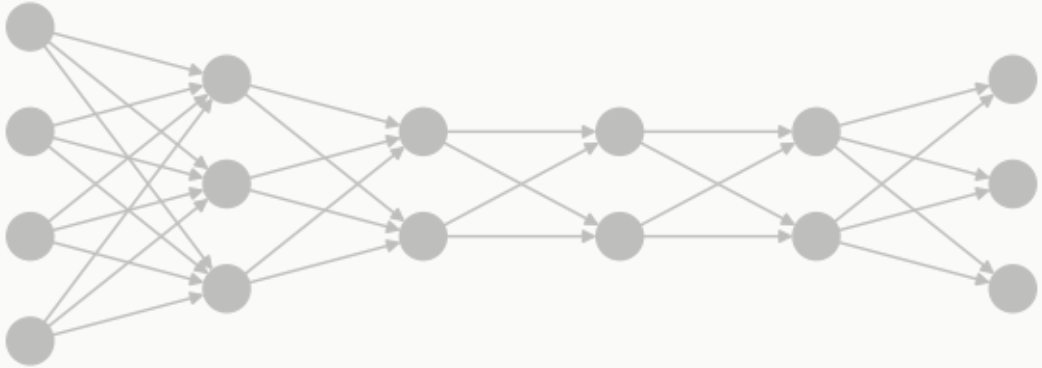
UNIVERSITÄT BONN

BUILDING THE GRAPH



UNIVERSITÄT BONN

GNN FOR PATTERN RECOGNITION



- Classify edges: true edge or false edge
- Use Interaction Network* which is a type of Graph Neural Network (GNN)
- Number of hidden units decreased to 16
- Uses previously built graph as input
- Training with GPU using simulated data

* described in DeZoort et al. (2021)

UNIVERSITÄT BONN

STRUCTURE OF THE GNN

Detector/Timepix3 → Readout & build graph (already on ACAP)

Classically:
Executed sequentially

1) Loop edges:
   - Run neural network “R1”
2) Loop nodes:
   - Rearrange output of “R1”
     → Run neural network “O”
3) Loop edges:
   - Rearrange output of “O”
     → Run neural network “R2”
     & Sigmoid/apply threshold

→ Classified edges

Each NN a small MLP: lin. Layer → ReLU →lin. Layer → ReLU → lin. Layer with 16 hidden units

UNIVERSITÄT BONN

IMPLEMENTATION FOR ACAP — PIPELINING

Detector/Timepix3 → Readout & build graph (already on ACAP)

1) Loop edges:
- Run neural network “R1” → AIE 1

2) Loop nodes:
- Rearrange output of “R1” → AIE 2

  → Run neural network “O” → AIE 3

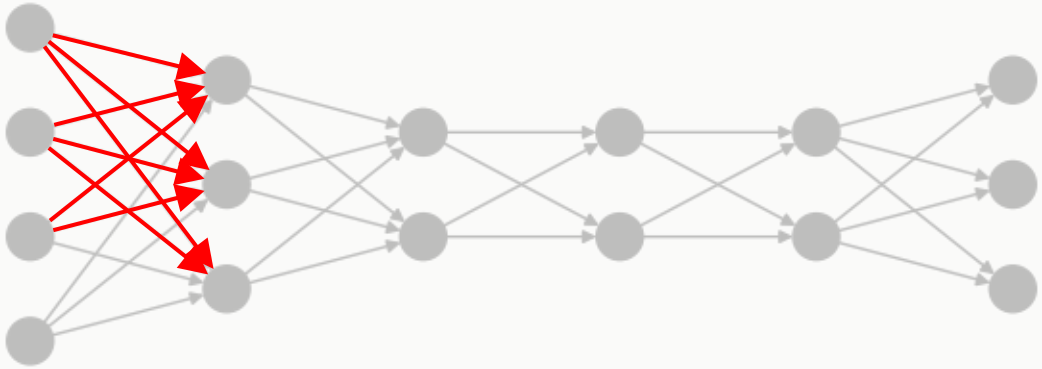
3) Loop edges:
- Rearrange output of “O” → AIE 4

  → Run neural network “R2” → AIE 5

  & Sigmoid/apply threshold → Classified edges

UNIVERSITÄT BONN

IMPLEMENTATION FOR ACAP — SIMD

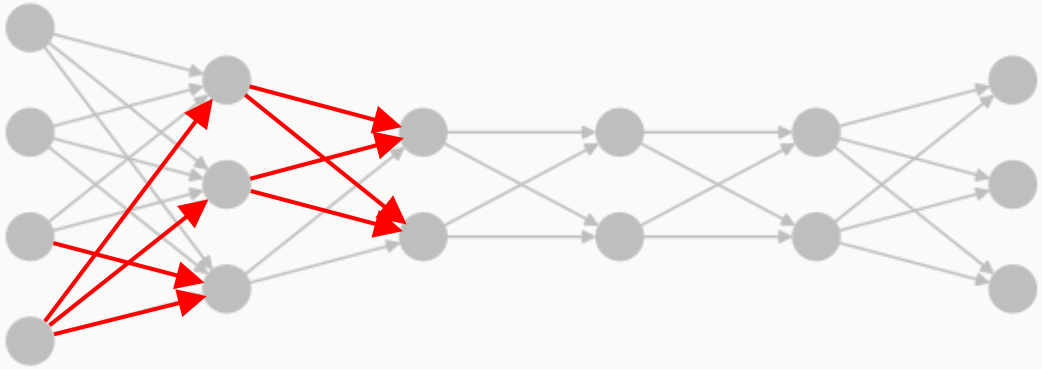


1) Loop edges:
   - Run neural network “R1” → AIE 1

...with 8 accumulators

UNIVERSITÄT BONN

IMPLEMENTATION FOR ACAP — SIMD



1) Loop edges:
   - Run neural network “R1” → AIE 1

...with 8 accumulators

UNIVERSITÄT BONN

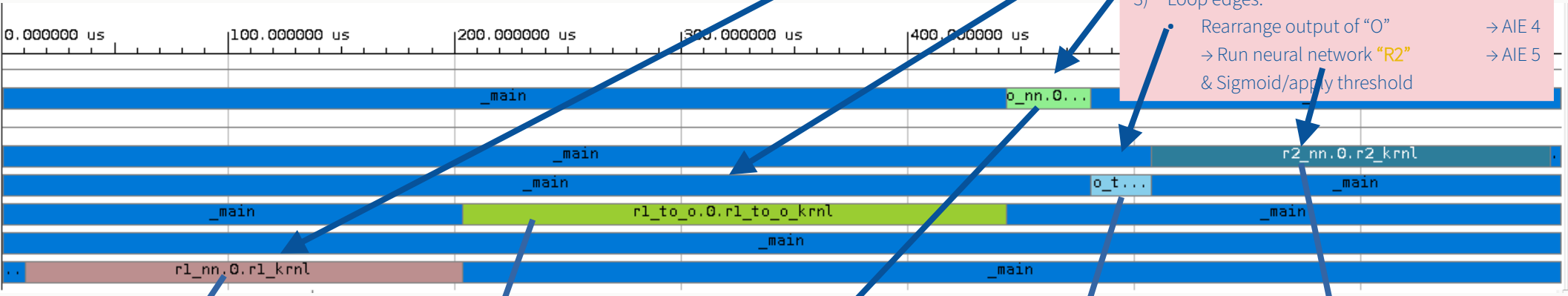
INFERENCE TIME

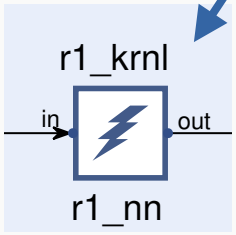
For example, inference time for graph with 384 edges and 82 nodes:

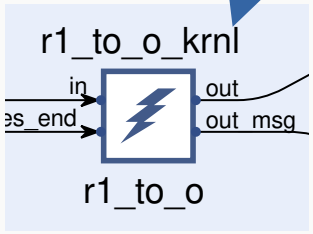
1) Loop edges:
   - Run neural network "R1" → AIE 1

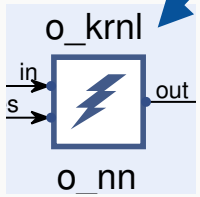
2) Loop nodes:
   - Rearrange output of "R1" → AIE 2
   - → Run neural network "O" → AIE 3

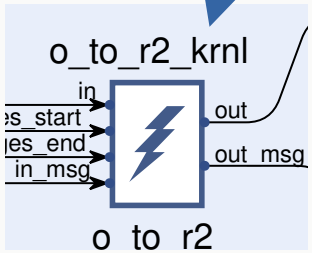
3) Loop edges:
   - Rearrange output of "O" → AIE 4
   - → Run neural network "R2" → AIE 5
   - & Sigmoid/apply threshold

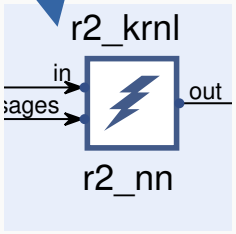








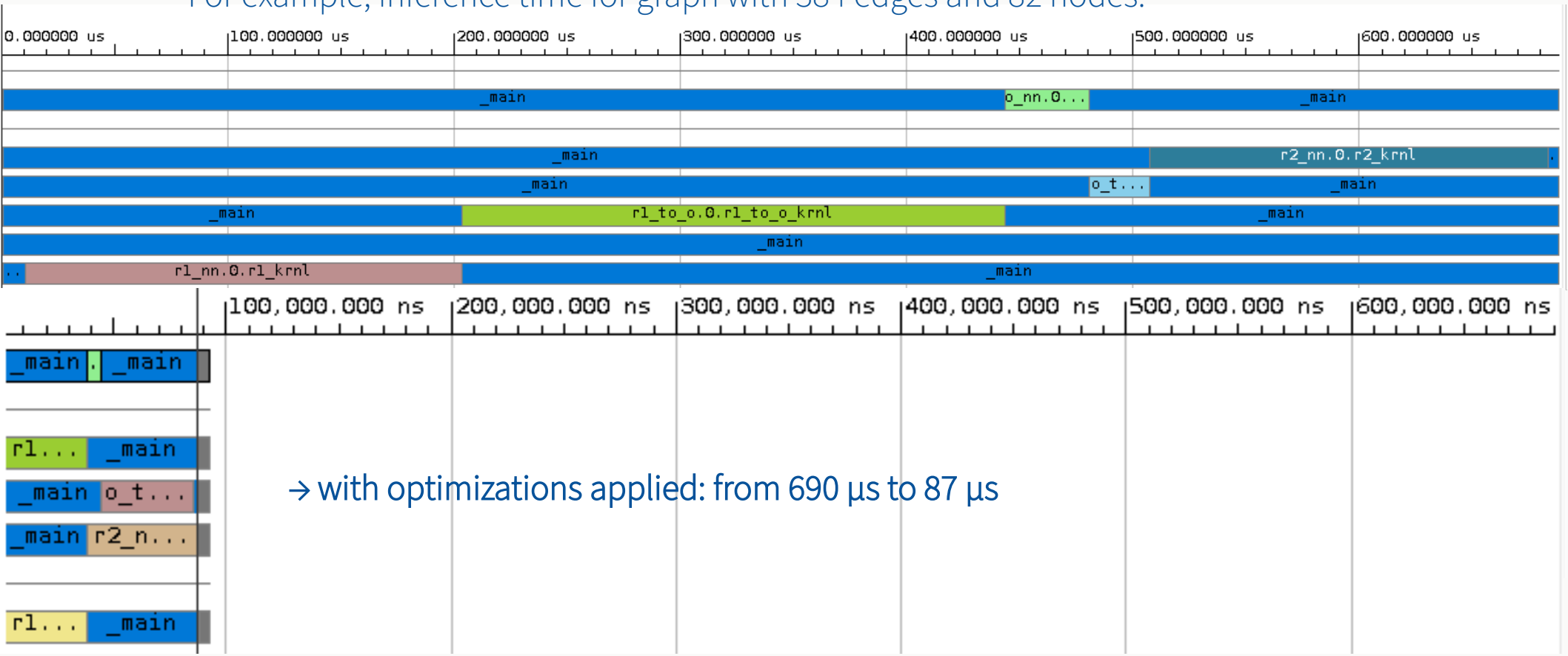




UNIVERSITÄT BONN

CODE OPTIMIZATIONS – VLIW USAGE

For example, inference time for graph with 384 edges and 82 nodes:



→ with optimizations applied: from 690 µs to 87 µs

UNIVERSITÄT BONN

QUANTIZED IMPLEMENTATION

We want to go faster: Use more AI Engines and quantize

1) Loop edges:
- Run neural network "R1" → AIE 1,2,3

2) Loop nodes:
- Rearrange output of "R1" → AIE 4
  → Run neural network "O" → AIE 5,6,7

3) Loop edges:
- Rearrange output of "O" → AIE 8
  → Run neural network "R2" → AIE 9,10,11
  & Sigmoid/apply threshold

QUANTIZED IMPLEMENTATION

Quantization:

- Limited by:
  - Supported integer sizes: 8b, 16b, 32b
  - Vector register sizes: 128b, 256b, 512b
- Can be mixed (8b x 16b vector multiplications, vectors in 128b and 256b registers)
- But can be challenging to implement
- 2 in-/output streams, can read/write 32 bit per instruction
- Additional Cascade stream between accumulators

→ currently still optimizing but speedup by factor of 10 observed

UNIVERSITÄT BONN

QUANTIZED IMPLEMENTATION

An example:

Input: 4 features, 8 bit each:

= 32 bit → too small for vector registers

→ pack data of four inputs into one vector register

… run neural network on four edges simultaneously …

Output: 8 nodes, 8 bit (of 4 edges) → 256 bit:

Split the output register to be written into two streams in parallel (4 clock cycles)

Simple example, can become arbitrarily complicated

UNIVERSITÄT BONN

SUMMARY & OUTLOOK

- Aim: Use Xilinx Versal ACAP for triggering in an electron on target experiment
- ACAP: Combination of CPU, RPU, FPGA and AI processors
- AIEs with highly parallelized architecture: Multi processor, SIMD, VLIW
- Currently in implementation stage: Interaction Network for pattern recognition
- Speedup by factor of ~8 with manual code optimizations
- Additional factor of 10 from quantization
- Fully working but room for improvement
  - More code/compute graph optimizations
  - Integrate FPGA-part for the final, full tracker
- Our vision: create building blocks optimized for low latency neural networks

UNIVERSITÄT BONN

Thanks for your attention!

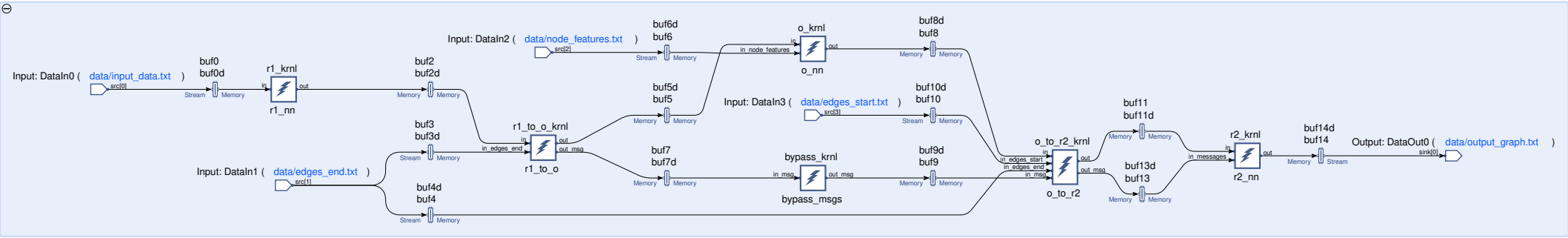
Questions?

UNIVERSITÄT BONN

BACKUP

UNIVERSITÄT BONN

PIPELINING



UNIVERSITÄT BONN

IMPLEMENTATION FOR ACAP

Con. 1:	ToA_{Start}	ToA_{End}	ToT_{Start}	ToT_{End}
Con. 2:	ToA_{Start}	ToA_{End}	ToT_{Start}	ToT_{End}
Con. 3:	ToA_{Start}	ToA_{End}	ToT_{Start}	ToT_{End}
Con. 4:	ToA_{Start}	ToA_{End}	ToT_{Start}	ToT_{End}
Con. 5:	ToA_{Start}	ToA_{End}	ToT_{Start}	ToT_{End}
Con. 6:	ToA_{Start}	ToA_{End}	ToT_{Start}	ToT_{End}
Con. 7:	ToA_{Start}	ToA_{End}	ToT_{Start}	ToT_{End}
Con. 8:	ToA_{Start}	ToA_{End}	ToT_{Start}	ToT_{End}

Repeat $N_{connections}/8$-times, iterate possible connections

IMPLEMENTATION FOR ACAP

Con. 1: ToA_{Start} ToA_{End} ToT_{Start} ToT_{End}
Con. 2: ToA_{Start} ToA_{End} ToT_{Start} ToT_{End}
Con. 3: ToA_{Start} ToA_{End} ToT_{Start} ToT_{End}
Con. 4: ToA_{Start} ToA_{End} ToT_{Start} ToT_{End}
Con. 5: ToA_{Start} ToA_{End} ToT_{Start} ToT_{End}
Con. 6: ToA_{Start} ToA_{End} ToT_{Start} ToT_{End}
Con. 7: ToA_{Start} ToA_{End} ToT_{Start} ToT_{End}
Con. 8: ToA_{Start} ToA_{End} ToT_{Start} ToT_{End}

Biases:  …

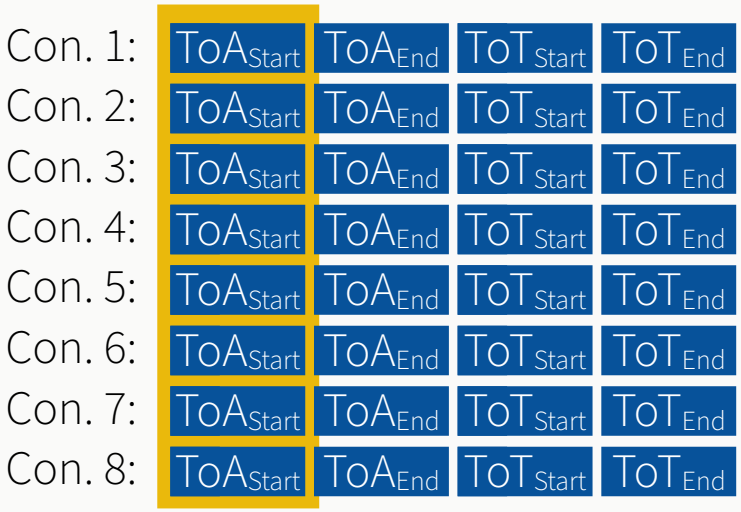
(From training on PC)

$Acc_1 = Bias_1$
$Acc_2 = Bias_1$
$Acc_3 = Bias_1$
$Acc_4 = Bias_1$
$Acc_5 = Bias_1$
$Acc_6 = Bias_1$
$Acc_7 = Bias_1$
$Acc_8 = Bias_1$

Repeat $N_{outputs}$-times, iterate biases

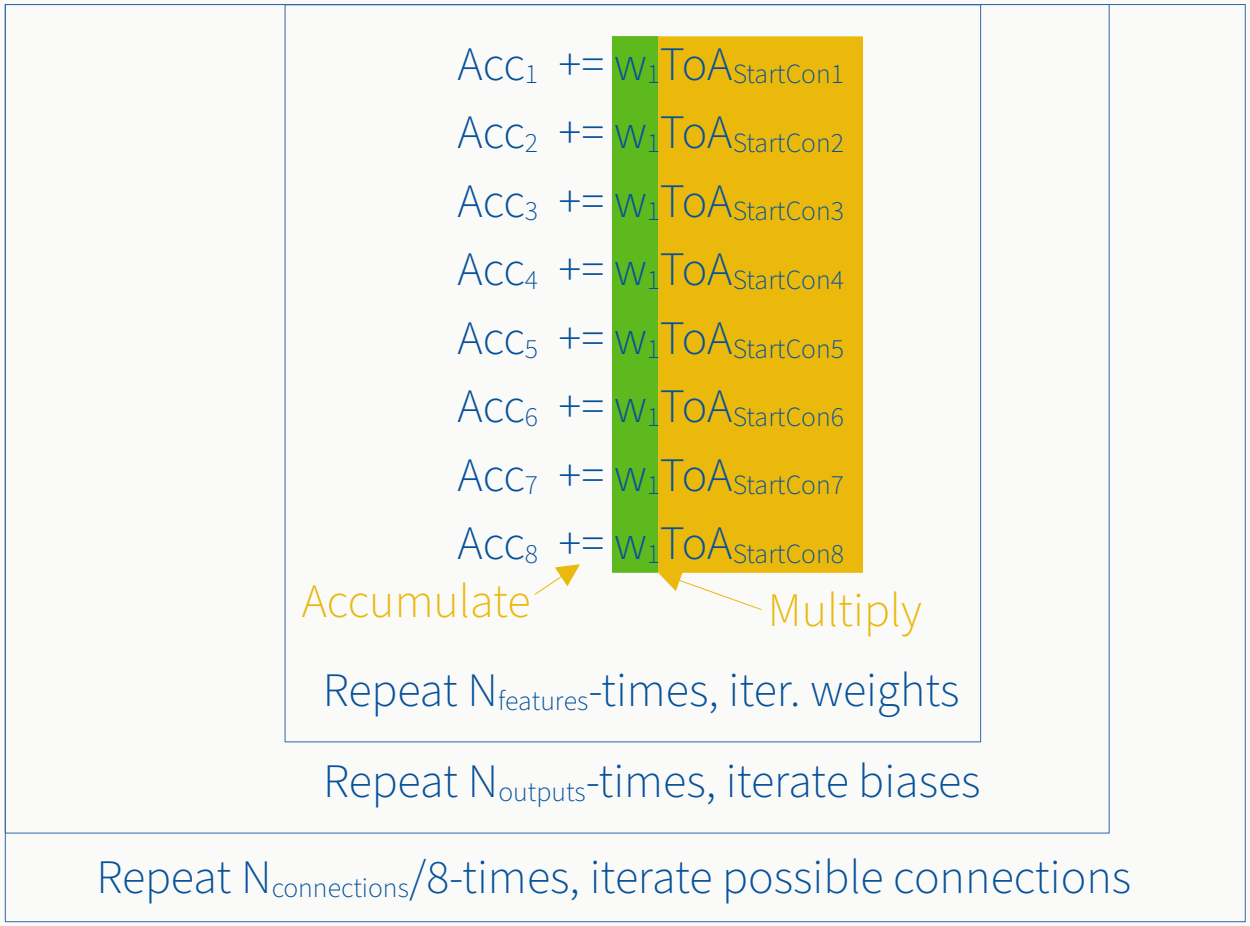
Repeat $N_{connections}/8$-times, iterate possible connections

IMPLEMENTATION FOR ACAP



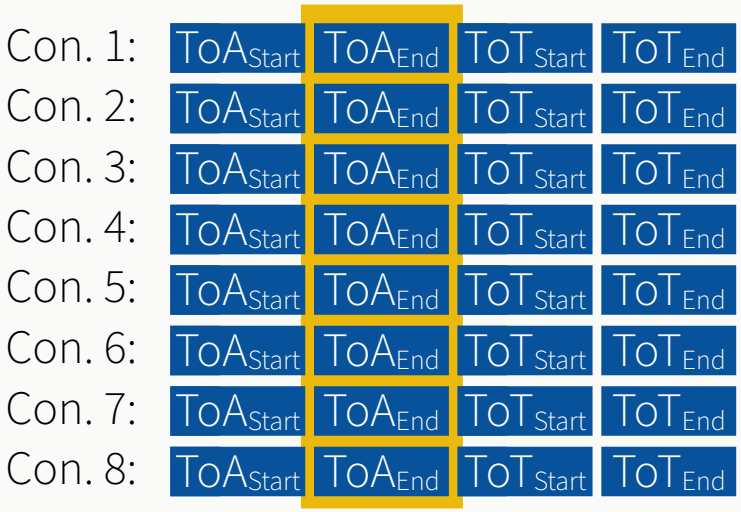
Weights: …

(From training on PC)



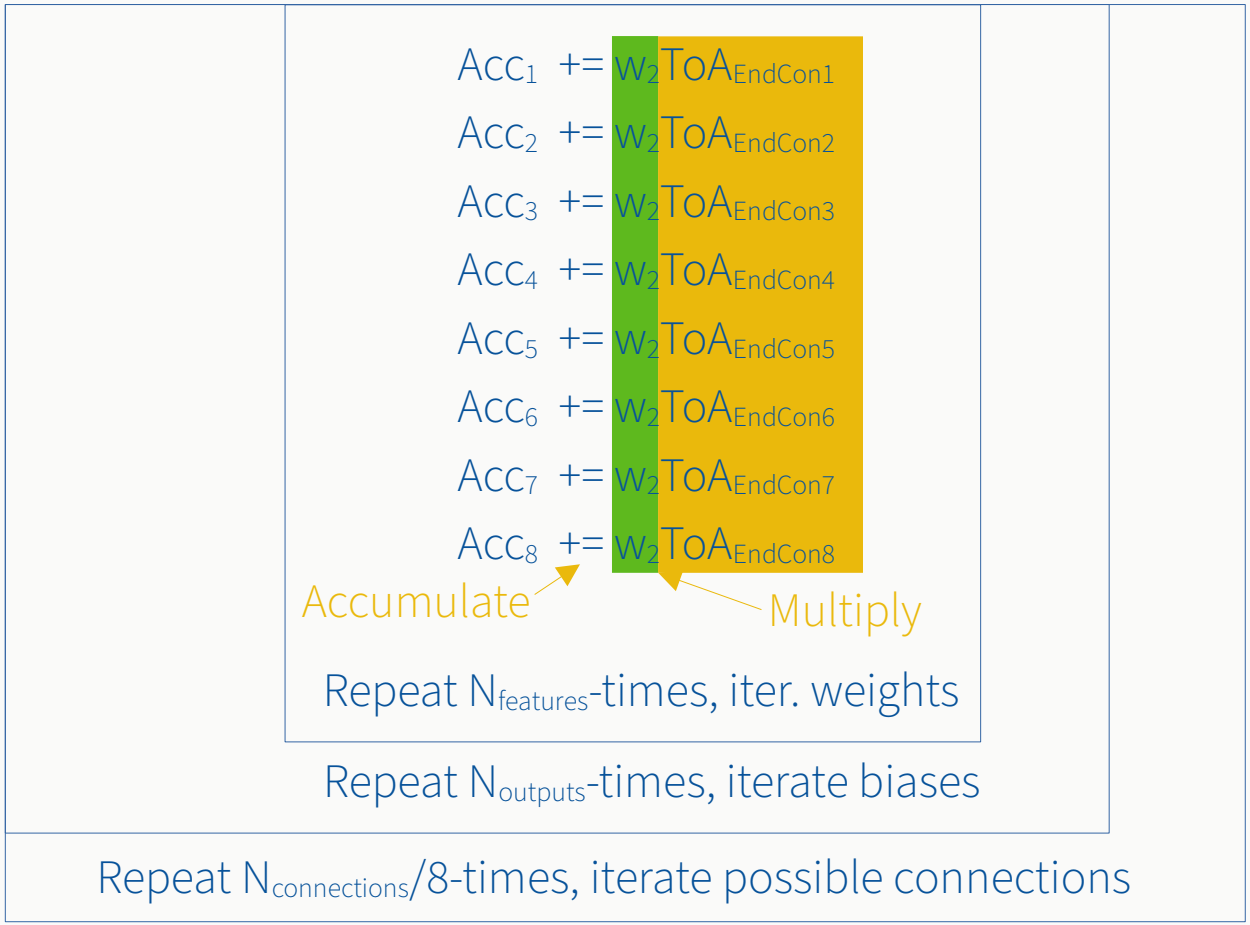
UNIVERSITÄT BONN

IMPLEMENTATION FOR ACAP



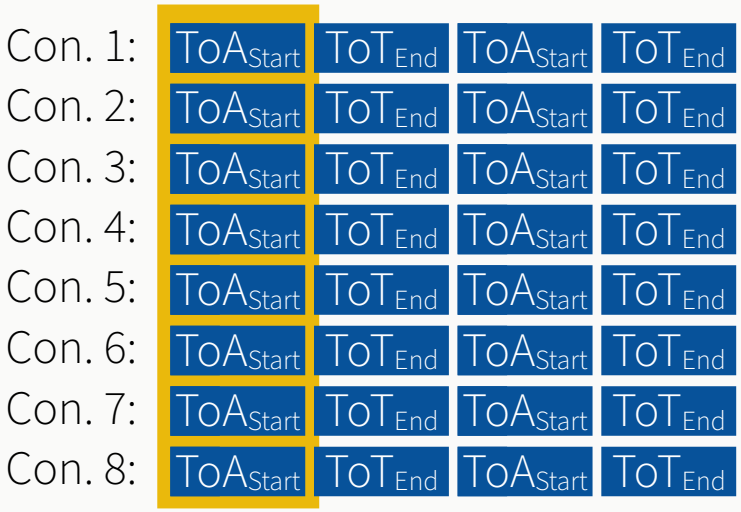
Weights: …

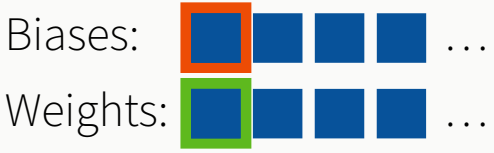
(From training on PC)



UNIVERSITÄT BONN

IMPLEMENTATION FOR VCK190





(From training on PC)

$Acc_1 = Bias_1$
$Acc_2 = Bias_1$
$Acc_3 = Bias_1$
$Acc_4 = Bias_1$
$Acc_5 = Bias_1$
$Acc_6 = Bias_1$
$Acc_7 = Bias_1$
$Acc_8 = Bias_1$

$Acc_1 \mathrel{+}= w_1 ToA_{StartCon1}$
$Acc_2 \mathrel{+}= w_1 ToA_{StartCon2}$
$Acc_3 \mathrel{+}= w_1 ToA_{StartCon3}$
$Acc_4 \mathrel{+}= w_1 ToA_{StartCon4}$
$Acc_5 \mathrel{+}= w_1 ToA_{StartCon5}$
$Acc_6 \mathrel{+}= w_1 ToA_{StartCon6}$
$Acc_7 \mathrel{+}= w_1 ToA_{StartCon7}$
$Acc_8 \mathrel{+}= w_1 ToA_{StartCon8}$

Accumulate

Multiply

Repeat $N_{features}$-times, iter. weights

Repeat $N_{outputs}$-times, iterate biases

Repeat $N_{edges}/8$-times, iterate possible connections

UNIVERSITÄT BONN

CODE OPTIMIZATIONS

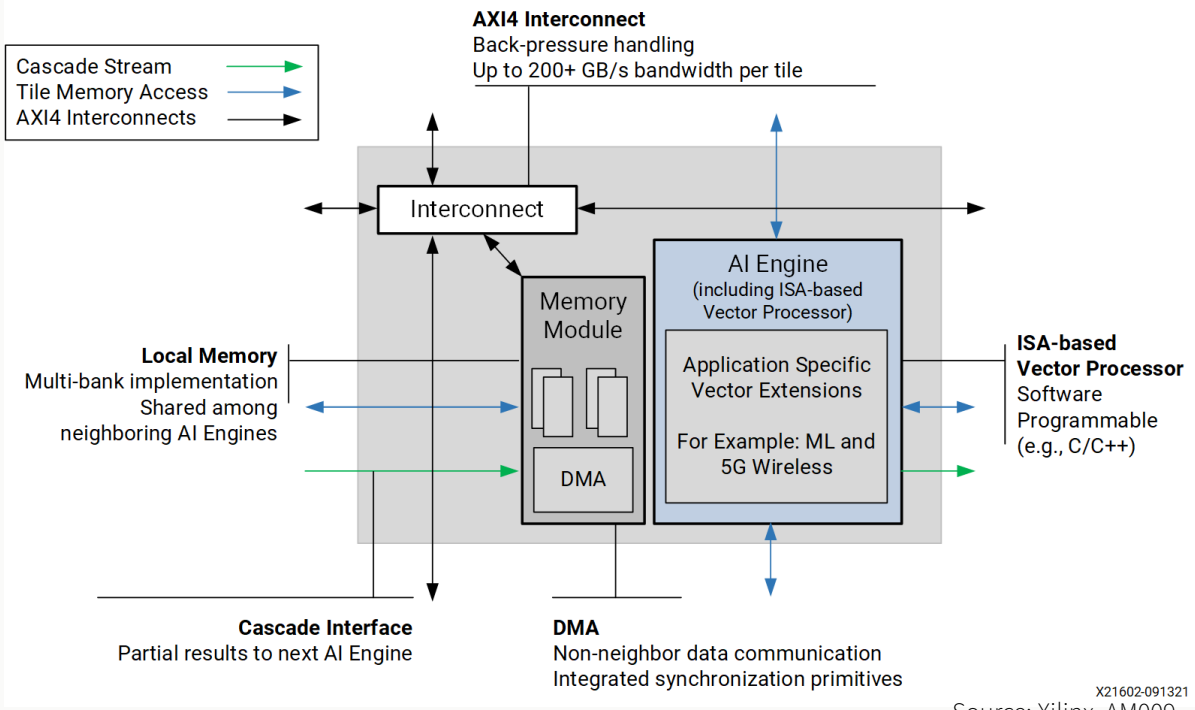
	Line/Label	Load A	Load B	Store	Scalar Op.	Move A	Move B	Vector Op.
168	2260	NOP	NOP	VST wc0, [sp, #-736	NOP	VMOV wr0, wr3	NOP	VSEL xc, ya.s32, r8, c5, r4, c5, c4, r8
169	2272	NOP	NOP	VST wc1, [sp, #-768	NOP	VMOV wr1, wr3	NOP	NOP
170	2280	NOP	NOP	VST.SPIL wd0, [sp,	NOP	NOP	NOP	NOP
171	2288	NOP	NOP	VST wd1, [sp, #-83	NOP	VMOV wd0, wc0	NOP	NOP
172	2296	VLDA wd0, [sp, #-5	NOP	NOP	NOP	NOP	NOP	VFPMAC wr3, r5, wr2, yd, r8, cl0, wc0, #0, cl5, #0, cl2
173	2312	NOP	NOP	NOP	NOP	NOP	NOP	VSEL xc, yd.s32, r8, c5, r4, c5, c4, r8
174	2316	VLDA.SPIL wr0, [sp	NOP	NOP	NOP	NOP	NOP	NOP
175	2324	NOP	NOP	NOP	NOP	NOP	NOP	VFPMAC wd1, r1, wr3, ya, r8, cl0, wc0, #0, cl0, #0, cl1
176	2332	VLDA.SPIL wc0, [sp	NOP	NOP	NOP	NOP	NOP	NOP
177	2340	VLDA.SPIL wc1, [sp	NOP	NOP	NOP	NOP	NOP	NOP
178	2348	VLDA wc0, [sp, #-6	NOP	NOP	NOP	VMOV wr3, wr1	NOP	NOP
179	2356	VLDA wr0, [sp, #-5	NOP	NOP	NOP	NOP	NOP	VSHL0.32 xa, r3
180	2364	VLDA wc1, [sp, #-6	NOP	NOP	NOP	VMOV wr1, wr3	NOP	VSEL xc, ya.s32, r8, c5, r4, c5, c4, r8
181	2376	VLDA.SPIL wc0, [sp	NOP	NOP	NOP	NOP	NOP	NOP
182	2384	VLDA.SPIL wr0, [sp	NOP	NOP	NOP	NOP	NOP	VFPMAC wr3, r1, wd1, ya, r8, cl0, wc0, #0, cl0, #0, cl1
183	2400	VLDA.SPIL wc1, [sp	NOP	NOP	NOP	NOP	NOP	NOP
184	2408	VLDA.SPIL wc0, [sp	NOP	NOP	NOP	NOP	NOP	VFPMAC wd1, r1, wr3, yd, r8, cl0, wc0, #0, cl0, #0, cl1
185	2424	VLDA.SPIL wc1, [sp	NOP	NOP	NOP	NOP	NOP	NOP
186	2432	VLDA.SPIL wr0, [sp	NOP	NOP	NOP	NOP	NOP	NOP
187	2440	VLDA.SPIL wr0, [sp	NOP	NOP	NOP	NOP	NOP	VFPMAC wd1, r1, wd1, ya, r8, cl0, wc0, #0, cl0, #0, cl1
188	2456	NOP	NOP	NOP	NOP	NOP	NOP	NOP
189	2458	NOP	NOP	NOP	NOP	NOP	NOP	NOP
190	2460	NOP	NOP	NOP	NOP	NOP	NOP	VFPMAC wd1, r1, wd1, ya, r8, cl0, wc0, #0, cl0, #0, cl1
191	2468	NOP	NOP	NOP	NOP	NOP	NOP	NOP
192	2470	NOP	NOP	NOP	NOP	NOP	NOP	NOP
193	2472	NOP	NOP	NOP	NOP	NOP	NOP	VFPMAC wd1, r1, wd1, ya, r8, cl0, wc0, #0, cl0, #0, cl1
194	2480	NOP	NOP	NOP	NOP	NOP	NOP	NOP
195	2482	NOP	NOP	NOP	NOP	NOP	NOP	NOP
196	2484	NOP	NOP	NOP	NOP	NOP	NOP	NOP
197	2486	NOP	NOP	NOP	NOP	NOP	NOP	NOP

CODE OPTIMIZATIONS

	Line/Label	Load A	Load B	Store	Scalar Op.	Move A	Move B	Vector Op.
138	2104	VLDA wc1, [p0], #32	NOP	VST.SPIL wd0, [sp,	NOP	NOP	NOP	NOP
139	2116	VLDA.SPIL wd0, [sp, #-672]	VLDB wd1, [p0], #3	NOP	NOP	NOP	NOP	NOP
140	2128	VLDA wc0, [p0], #32	NOP	VST.SPIL wc1, [sp,	NOP	NOP	NOP	NOP
141	2140	VLDA.SPIL wc0, [sp, #-96]	VLDB wr2, [p0], m	NOP	NOP	NOP	NOP	NOP
142	2152	NOP	NOP	VST.SPIL wd1, [sp,	NOP	NOP	NOP	NOP
143	2160	NOP	NOP	VST wc0, [sp, #-384	NOP	MOV ch5, p0	NOP	NOP
144	2168	NOP	NOP	VST.SPIL wd0, [sp,	NOP	NOP	NOP	NOP
145	2176	NOP	NOP	VST.SPIL wr2, [sp,	NOP	NOP	NOP	NOP
146	2184	VLDA.SPIL wc1, [sp, #-160]	NOP	NOP	NOP	NOP	NOP	NOP
147	2192	NOP	NOP	VST.SPIL wc0, [sp,	NOP	NOP	NOP	NOP
148	2200	NOP	NOP	VST.SPIL wc1, [sp,	NOP	NOP	NOP	VFPMUL wd0, r0, yd, r12, cl0, wc0, #0, cl4, #0, cl1
149	2216	VLDA.SPIL wr0, [sp, #-608]	NOP	NOP	NOP	NOP	NOP	NOP
150	2224	NOP	NOP	VST.SPIL wr3, [sp,	NOP	NOP	NOP	VFPMAC wd0, r0, wd0, ya, r12, cl0, wc0, #1, cl4, #0, cl1
151	2240	VLDA.SPIL wr0, [sp, #-576]	NOP	NOP	NOP	NOP	NOP	NOP
152	2248	NOP	NOP	VST.SPIL wd1, [sp,	NOP	NOP	NOP	NOP
153	2256	VLDA.SPIL wr0, [sp, #-544]	NOP	NOP	NOP	NOP	NOP	VFPMUL wd1, r0, yd, r12, cl0, wc1, #0, cl4, #0, cl1
154	2272	NOP	NOP	NOP	NOP	NOP	NOP	NOP
155	2276	VLDA.SPIL wr0, [sp, #-512]	NOP	NOP	NOP	NOP	NOP	VFPMAC wd1, r0, wd1, ya, r12, cl0, wc1, #1, cl4, #0, cl1
156	2292	NOP	NOP	NOP	NOP	NOP	NOP	VFPMAC wd0, r0, wd0, ya, r12, cl0, wc0, #2, cl4, #0, cl1
157	2300	VLDA.SPIL wr0, [sp, #-480]	NOP	NOP	NOP	NOP	NOP	VFPMAC wd1, r0, wd1, ya, r12, cl0, wc1, #2, cl4, #0, cl1
158	2316	NOP	NOP	NOP	NOP	NOP	NOP	VFPMAC wd0, r0, wd0, ya, r12, cl0, wc0, #3, cl4, #0, cl1
159	2324	VLDA.SPIL wr0, [sp, #-448]	NOP	NOP	NOP	NOP	NOP	VFPMAC wd1, r0, wd1, ya, r12, cl0, wc1, #3, cl4, #0, cl1
160	2340	VLDA wc0, [p5], #32	NOP	NOP	NOP	NOP	NOP	VFPMAC wd0, r0, wd0, ya, r12, cl0, wc0, #4, cl4, #0, cl1
161	2352	VLDA wc1, [p2], #32	NOP	NOP	NOP	NOP	NOP	VFPMAC wd1, r0, wd1, ya, r12, cl0, wc1, #4, cl4, #0, cl1
162	2364	NOP	NOP	NOP	NOP	NOP	NOP	VFPMAC wd0, r0, wd0, ya, r12, cl0, wc0, #5, cl4, #0, cl1
163	2372	VLDA.SPIL wr0, [sp, #-416]	NOP	NOP	NOP	NOP	NOP	VFPMAC wd1, r0, wd1, ya, r12, cl0, wc1, #5, cl4, #0, cl1
164	2388	NOP	NOP	NOP	NOP	NOP	NOP	VFPMAC wd0, r1, wd0, ya, r12, cl0, wc0, #6, cl4, #0, cl1
165	2396	VLDA.SPIL wr0, [sp, #-384]	NOP	NOP	NOP	NOP	NOP	VFPMAC wd1, r1, wd1, ya, r12, cl0, wc1, #6, cl4, #0, cl1
166	2412	NOP	NOP	NOP	NOP	NOP	NOP	VFPMAC wd0, r1, wd0, ya, r12, cl0, wc0, #7, cl4, #0, cl1
167	2420	VLDA wr0, [sp, #-352]	NOP	NOP	NOP	VMOV wr0, wr1	NOP	VFPMAC wd1, r1, wd1, ya, r12, cl0, wc1, #7, cl4, #0, cl1

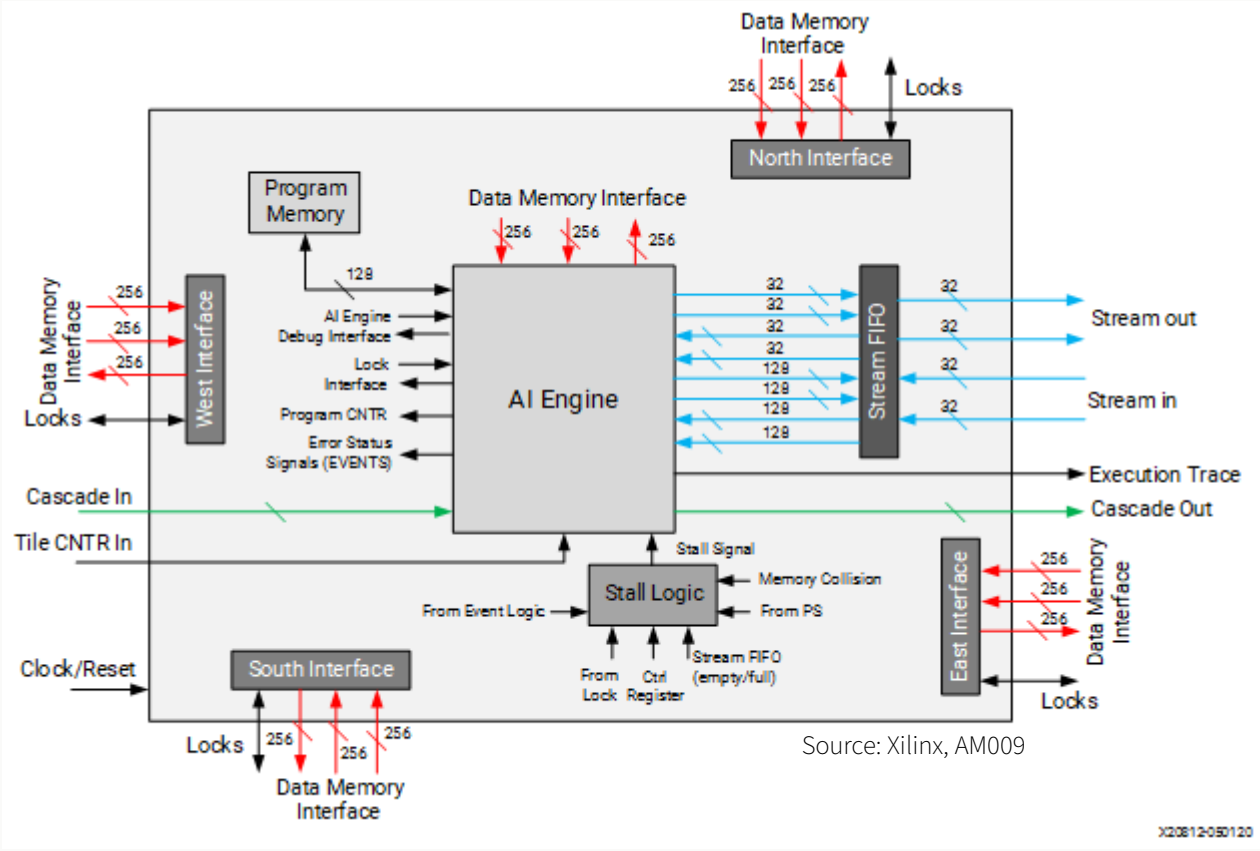
AIE TILE

32 KB
ECC / Parity
protected



Source: Xilinx, AM009

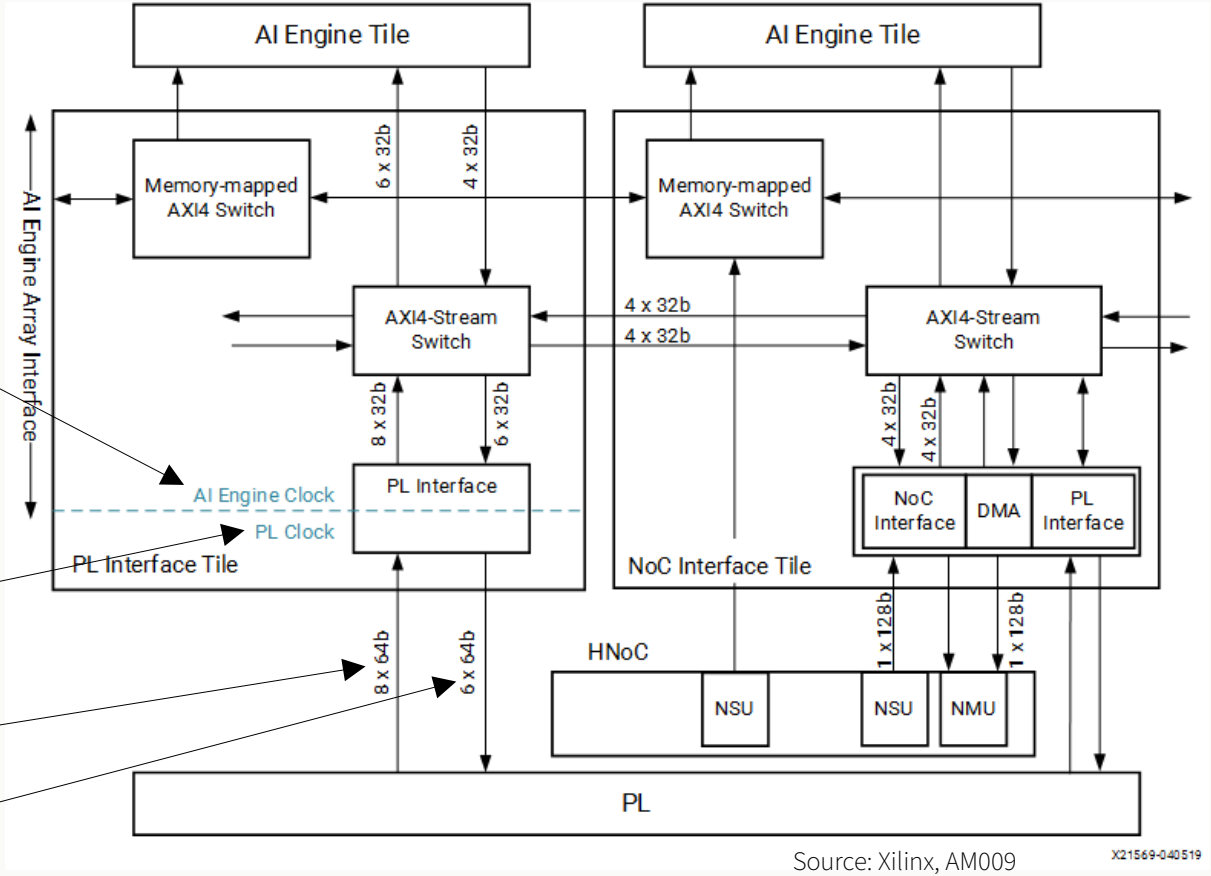
AIE INTERCONNECT



Source: Xilinx, AM009

UNIVERSITÄT BONN

INTERFACE TILE



Source: Xilinx, AM009

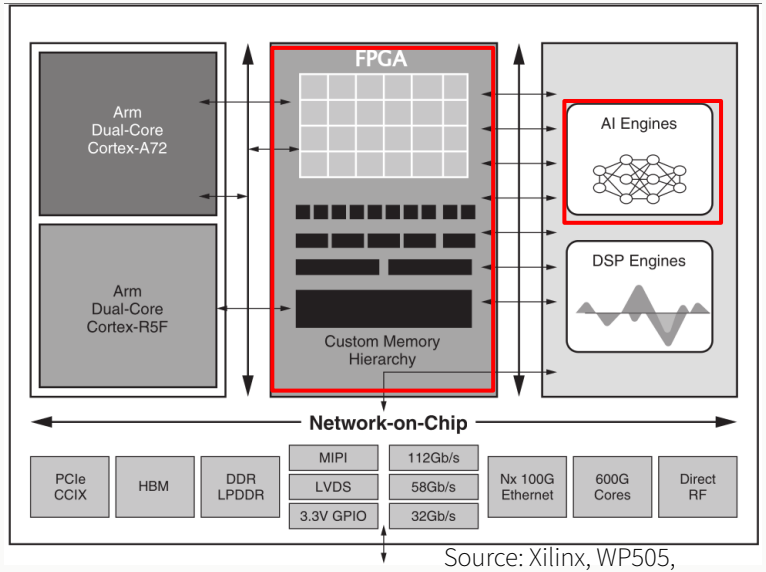
VC1902 chip

39 PL interface Tiles:

~ 1 TB /s

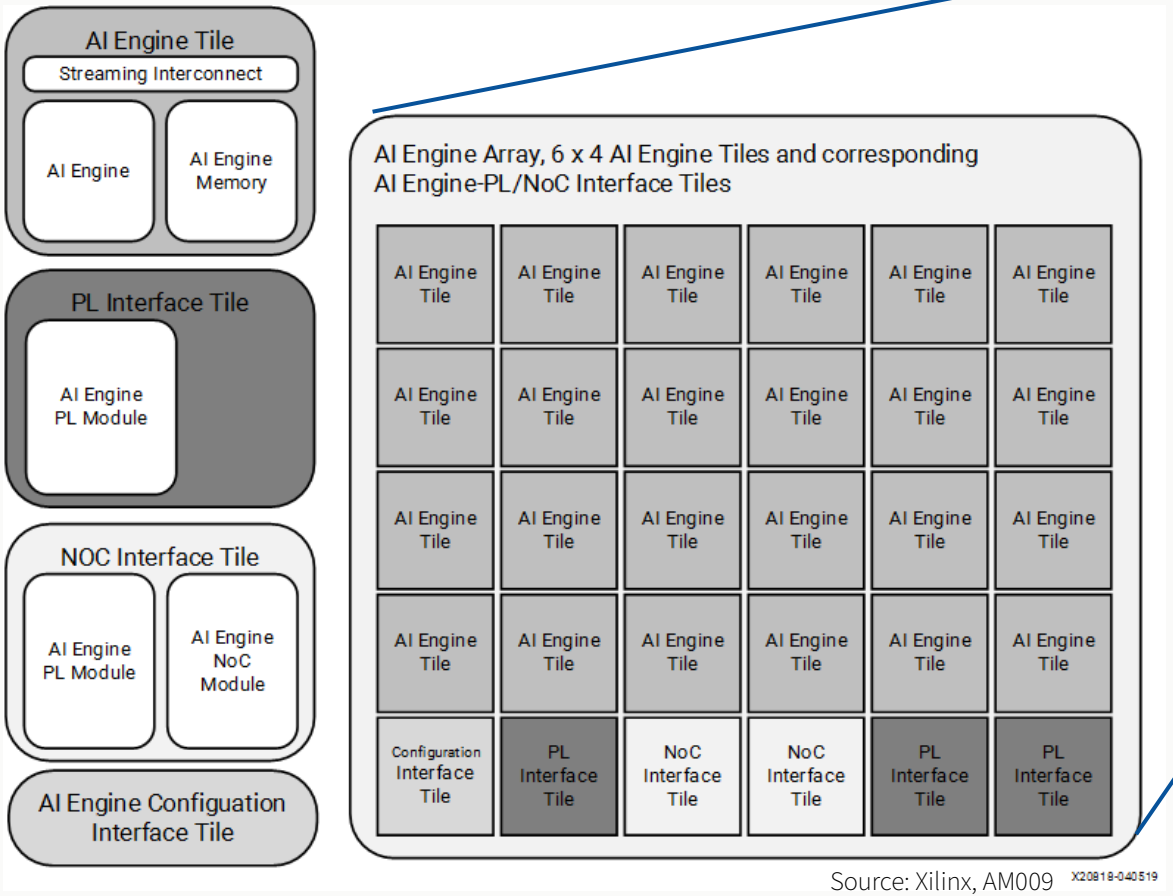
ADAPTIVE COMPUTE ACCELERATION PLATFORM (ACAP)

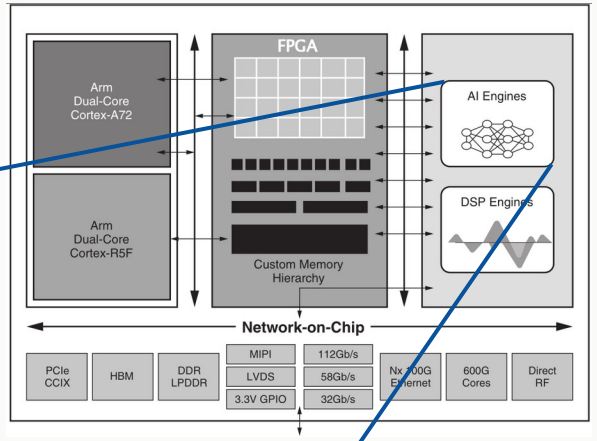
- Newly developed by Xilinx, heterogenous computation platform
- Data flow described using graph
- Processing done by kernels – “programs” running on AI Engines and FPGA
- Data can move back and forth between AI Engines and FPGA



Source: Xilinx, WP505, modified

AIE ARRAY AND TILES





Source: Xilinx, AM009

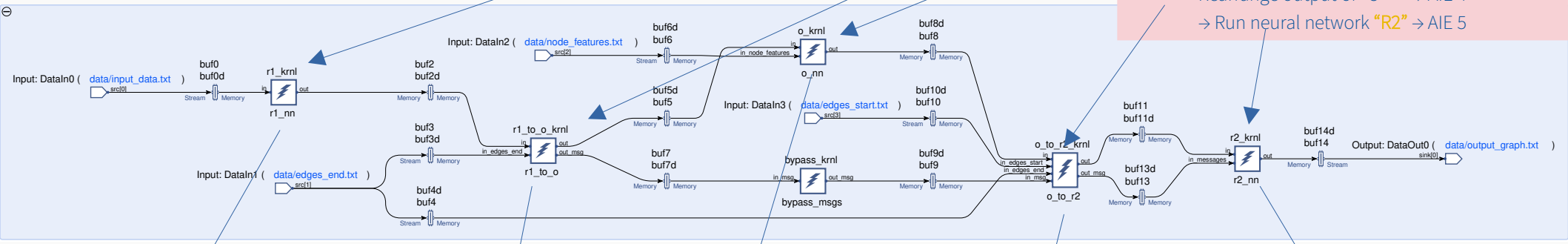
UNIVERSITÄT BONN

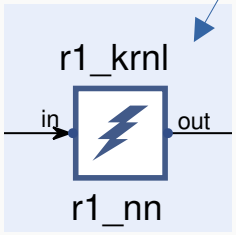
PIPELINING

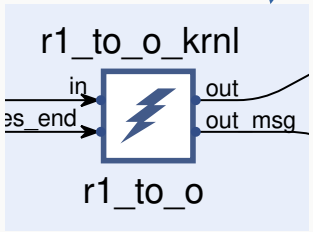
1) Loop edges:
   - Run neural network “R1” → AIE 1

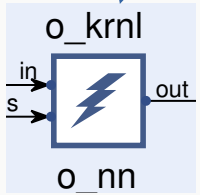
2) Loop nodes:
   - Rearrange output of “R1” → AIE 2
   → Run neural network “O” → AIE 3

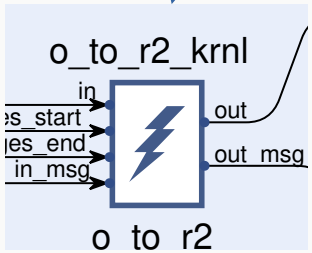
3) Loop edges:
   - Rearrange output of “O” → AIE 4
   → Run neural network “R2” → AIE 5

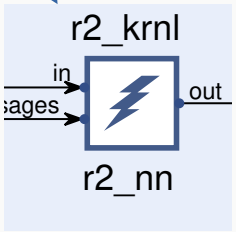












UNIVERSITÄT BONN

IMPLEMENTATION FOR ACAP — VLIW

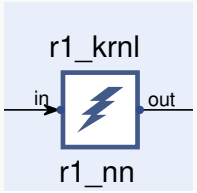
- Up to seven operations per VLIW
- Task: Merge operations from source code to single VLIW
- Done by compiler
- Changing algorithm structure (e.g. when to load and store data) allows for better “packing” into VLIWs
- Keeping in mind: available CPU registers, max. amount of data per load etc.
  → manual optimization improves performance

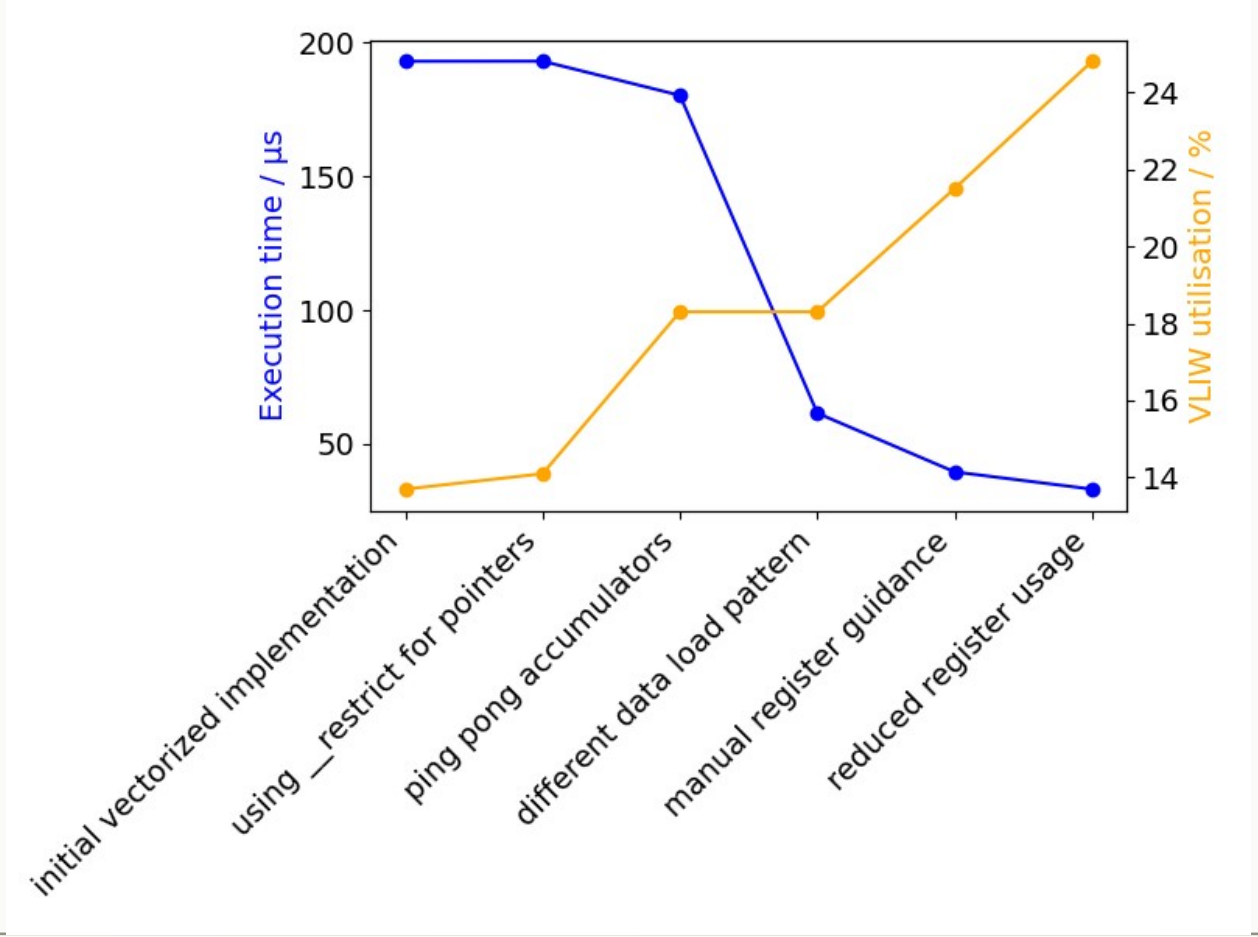
Example assembly:

Load A	Load B	Store	Scalar Op.	Move A	Move B	Vector Op.
NOP	NOP	VST wc0, [sp, #-736	NOP	VMOV wr0, wr3	NOP	VSEL xc, ya.s32, r8, c5, r4, c5, c4, r8
NOP	NOP	VST wc1, [sp, #-768	NOP	VMOV wr1, wr3	NOP	NOP
NOP	NOP	VST.SPIL wd0, [sp,	NOP	NOP	NOP	NOP
NOP	NOP	VST wd1, [sp, #-83	NOP	VMOV wd0, wc0	NOP	NOP
VLDA wd0, [sp, #-5	NOP	NOP	NOP	NOP	NOP	VFPMAC wr3, r5, wr2, yd, r8, cl0, wc0, #0, cl5, #0, cl2

→ one line = one VLIW → takes 1 ns to execute (but single units might take more time until finished)

CODE OPTIMIZATIONS – R1 KERNEL





More NOPs per line
→ lower VLIW utilisation

UNIVERSITÄT BONN

GNN → ACAP

- Programmed in C/C++ → compiled to machine code
- No VHDL, no Verilog, no HLS
  → AIEs are processors, not FPGAs
- Supported bit-widths: 8b, 16b, 32b (also complex)
- This implementation: 32b floating-point (no quantization yet)

  How to get the NN onto the ACAP?
  → Remember: a) 400 processors, b) VLIW, c) SIMD
- Compiler helps but:
  Each of the three has to be addressed during programming

	d B		Scalar Op.	Move A	Move B	Vector Op.
338		VST w	NOP	NOP	NOP	NOP
339		VST w	NOP	NOP	NOP	NOP
340		NOP	NOP	NOP	NOP	VFPMAC wd0, r0, wr2, yd, r8, cl0, wc1, #1, cl4,
341		NOP	NOP	NOP	NOP	NOP
342		NOP	NOP	VMOV wr0, wc0	NOP	VFPMAC wd0, r0, wd0, ya, r8, cl0, wc1, #2, cl4,
343		NOP	NOP	NOP	NOP	NOP
344		NOP	NOP	NOP	NOP	VFPMAC wd0, r0, wd0, ya, r8, cl0, wc1, #3, cl4,
345	#-28	NOP	NOP	VMOV wr2, wr0	NOP	VFPMAC wd1, r0, wd1, ya, r8, cl0, wc0, #6, cl4,
346		NOP	NOP	VMOV wr0, wr3	NOP	NOP
347		NOP	NOP	NOP	NOP	NOP
348	)2], m1, cyc0	NOP	NOP	NOP	NOP	VFPMAC wd0, r0, wd0, ya, r8, cl0, wc1, #4, cl4,
349		NOP	NOP	NOP	NOP	NOP
350		NOP	NOP	VMOV wr0, wr2	NOP	VFPMAC wd0, r0, wd0, ya, r8, cl0, wc1, #5, cl4,
351		NOP	NOP	NOP	NOP	NOP
352	m1, cyc0	NOP	NOP	NOP	NOP	VFPMAC wd0, r0, wd0, ya, r8, cl0, wc1, #6, cl4,
353		NOP	NOP	NOP	NOP	VFPMAC wd1, r0, wd1, ya, r8, cl0, wc0, #7, cl4,
354	m2, cyc0	NOP	NOP	NOP	NOP	VFPMAC wd0, r0, wd0, ya, r8, cl0, wc1, #7, cl4,
355		NOP	NOP	NOP	NOP	VFPMAC wd1, r0, wd1, ya, r8, cl0, wc0, #0, cl4,
356		NOP	NOP	NOP	NOP	VFPMAC wd0, r0, wd0, ya, r8, cl0, wc1, #0, cl4,
357		NOP	NOP	NOP	NOP	VFPMAC wd1, r0, wd1, ya, r8, cl0, wc0, #1, cl4,
358		NOP	NOP	NOP	NOP	VFPMAC wd0, r0, wd0, ya, r8, cl0, wc1, #1, cl4,
359		NOP	NOP	NOP	NOP	VFPMAC wd1, r0, wd1, ya, r8, cl4, wc0, #0, cl4,
360		NOP	NOP	NOP	NOP	NOP
361		NOP	NOP	NOP	NOP	VFPMAC wd0, r0, wd0, ya, r8, cl4, wc0, #0, cl4,
362		NOP	NOP	NOP	NOP	VFPMAC wd1, r0, wd1, ya, r8, cl4, wc0, #0, cl4,
363		NOP	NOP	NOP	NOP	VFPMAC wd0, r0, wd0, ya, r8, cl4, wc0, #0, cl4,

VLIW instructions: 682
From this pure NOP VLIWs: 224
Average VLIW utilisation: (15.65±14.48)%

#0:
#0: