AIM: Accelerating Arbitrary-precision Integer Multiplication on Heterogeneous Reconfigurable Computing Platform Versal ACAP

ICCAD 2023

Zhuoping Yang∗, Jinming Zhuang∗, Jiaqi Yin†, Cunxi Yu†, Alex K. Jones∗ and Peipei Zhou∗

University of Pittsburgh[*]; University of Maryland, College Park†;

zhuoping.yang@pitt.edu
peipei.zhou@pitt.edu

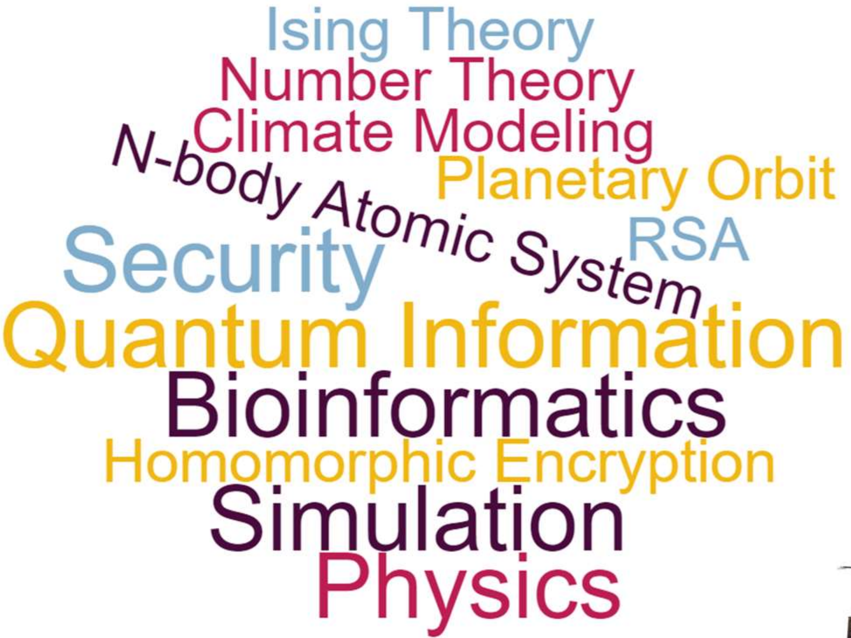
https://peipeizhou-eecs.github.io/

https://github.com/arc-research-lab/AIM

2023/11/29

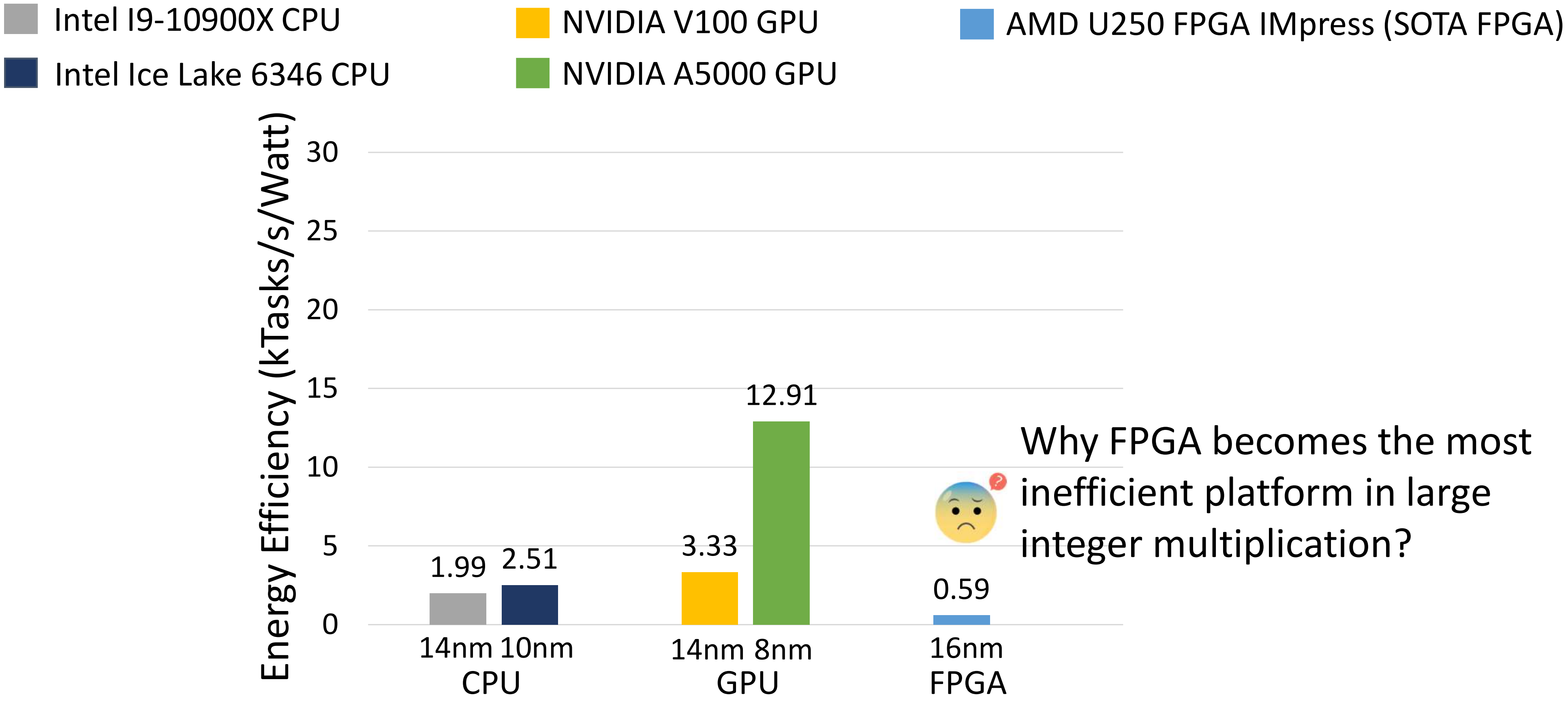
Motivation

- Arbitrary Precision Integer Multiplication



?

Motivation



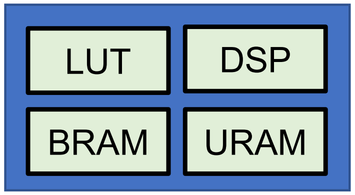
Why FPGA becomes the most inefficient platform in large integer multiplication?

Motivation

CPU

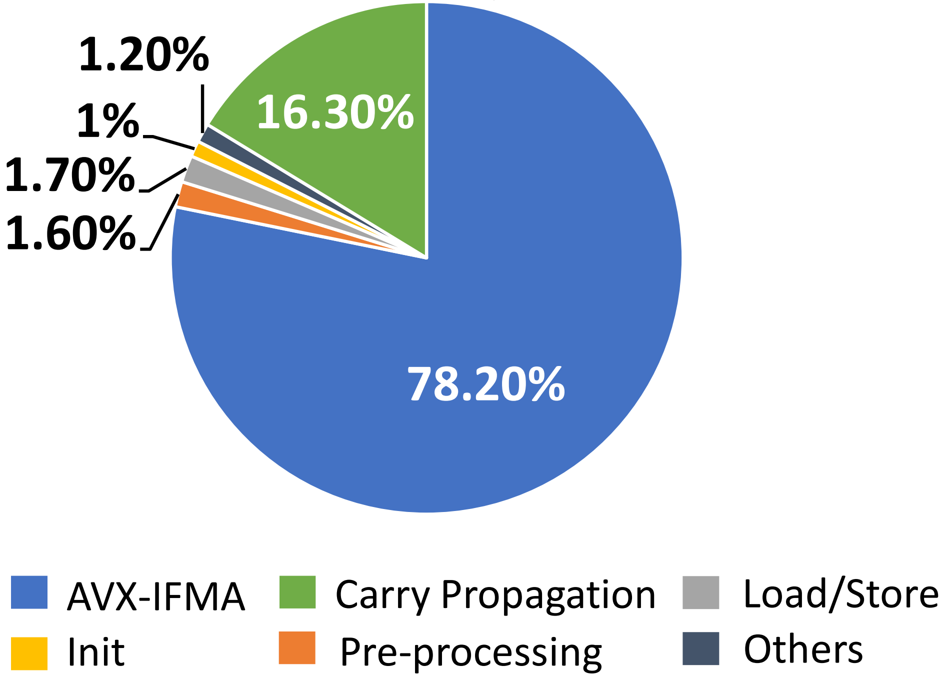
GPU

VS

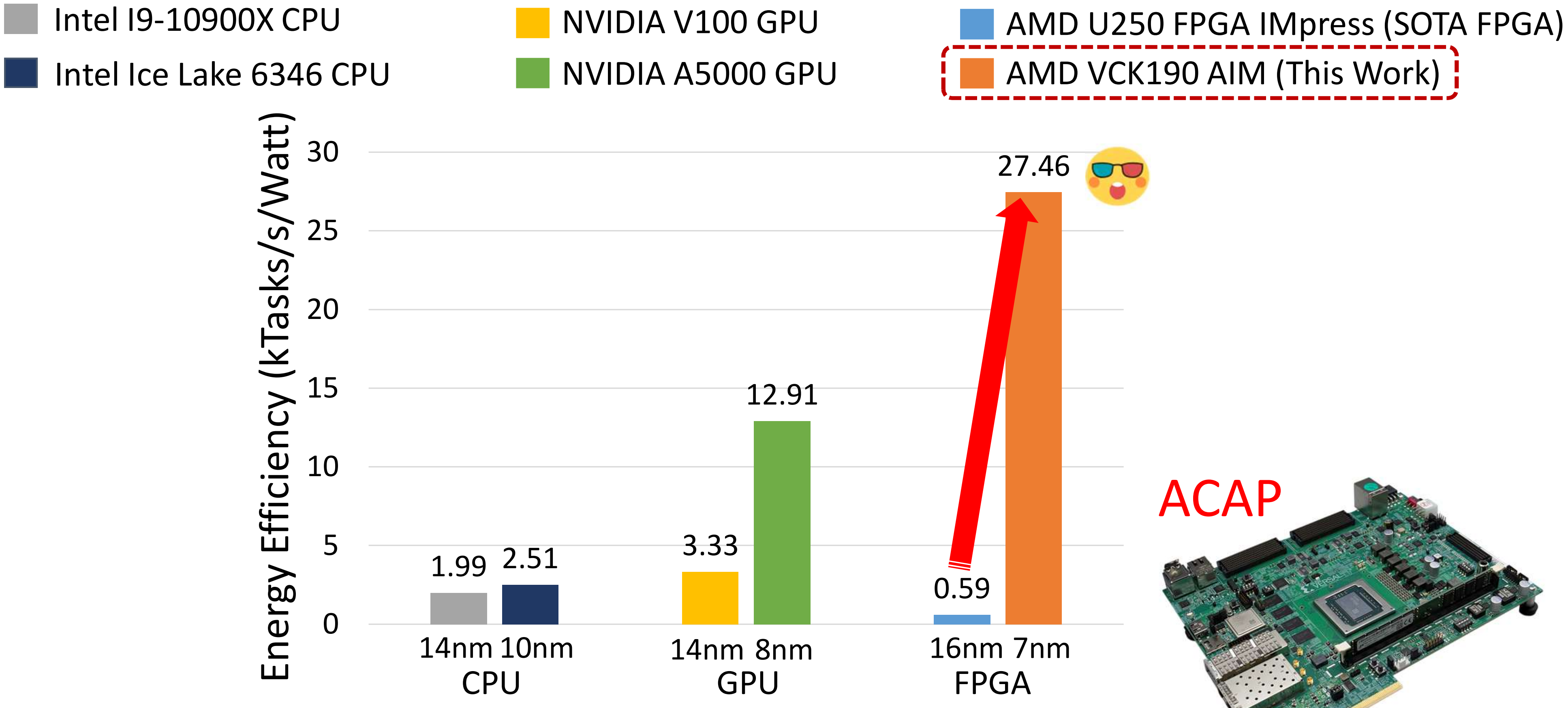


FPGA

CPU Instruction# Breakdown
(4096-bit Multiplication)



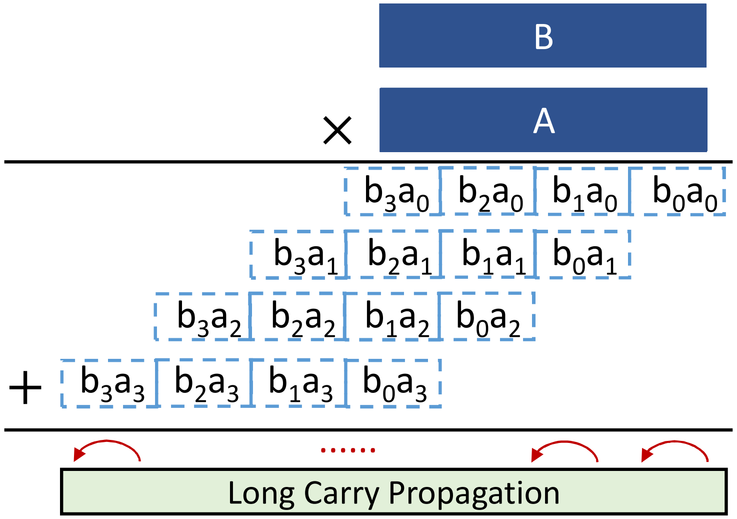
ACAP: FPGA + Vector Units



ACAP

Schoolbook Decomposition

Decomposition Method



B

× A

b_3a_0 b_2a_0 b_1a_0 b_0a_0

b_3a_1 b_2a_1 b_1a_1 b_0a_1

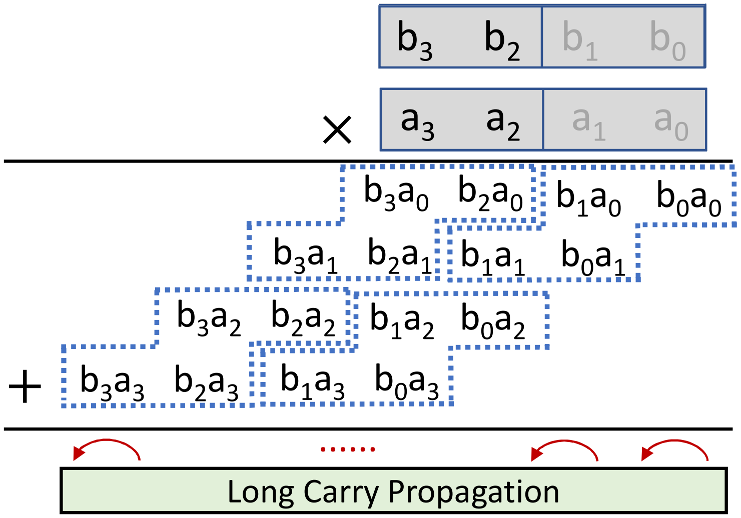
b_3a_2 b_2a_2 b_1a_2 b_0a_2

+ b_3a_3 b_2a_3 b_1a_3 b_0a_3

……

Long Carry Propagation

Background



b_3 b_2 b_1 b_0

× a_3 a_2 a_1 a_0

b_3a_0 b_2a_0 b_1a_0 b_0a_0

b_3a_1 b_2a_1 b_1a_1 b_0a_1

b_3a_2 b_2a_2 b_1a_2 b_0a_2

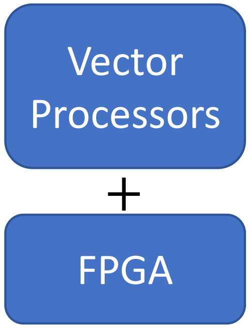
+ b_3a_3 b_2a_3 b_1a_3 b_0a_3

……

Long Carry Propagation

Motivation

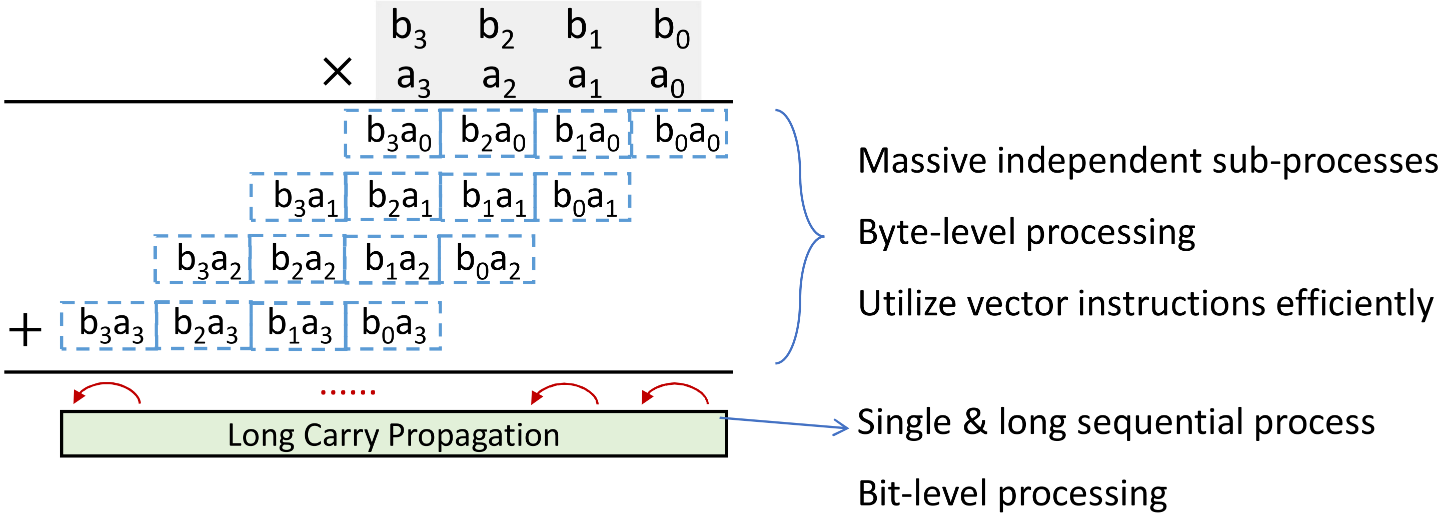
- Combination of massive parallelism and long sequential process



Vector Processors

+

FPGA



b_3 b_2 b_1 b_0
× a_3 a_2 a_1 a_0
b_3a_0 b_2a_0 b_1a_0 b_0a_0
b_3a_1 b_2a_1 b_1a_1 b_0a_1
b_3a_2 b_2a_2 b_1a_2 b_0a_2
+ b_3a_3 b_2a_3 b_1a_3 b_0a_3
……

Long Carry Propagation

Massive independent sub-processes

Byte-level processing

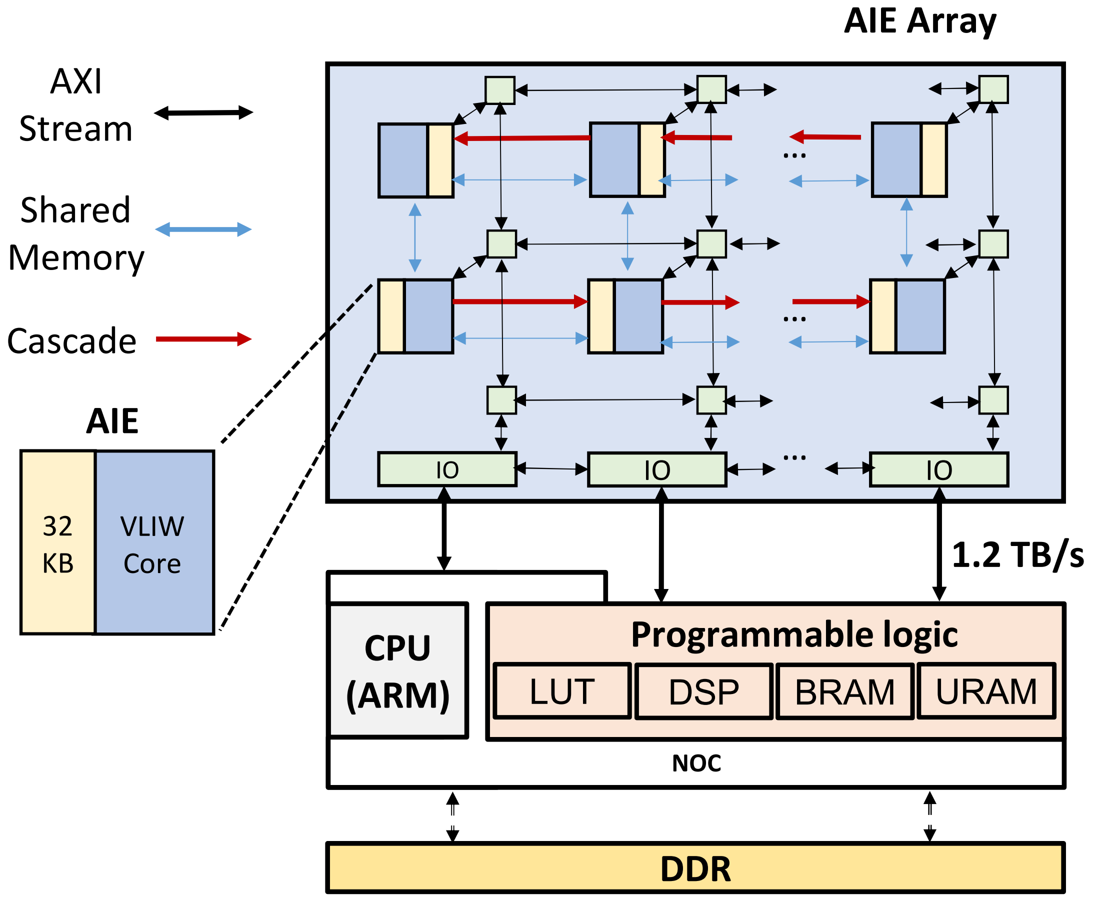
Utilize vector instructions efficiently

Single & long sequential process

Bit-level processing

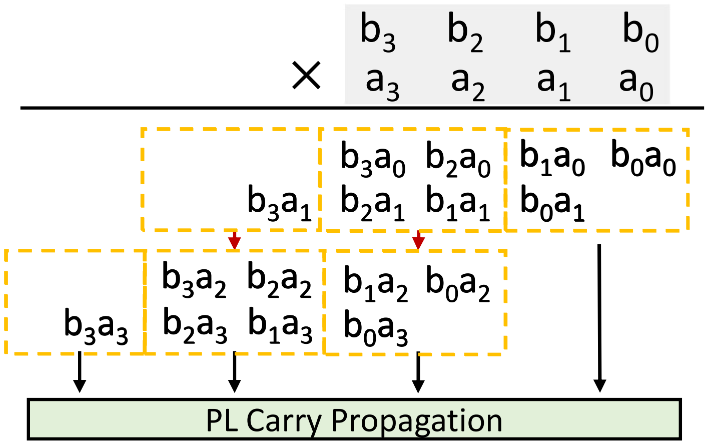
- Can FPGAs take advantage of vector processing?
- Will heterogeneity bring performance or energy efficiency gains?
- How to find the best mapping strategy? Programming Efforts? Etc.

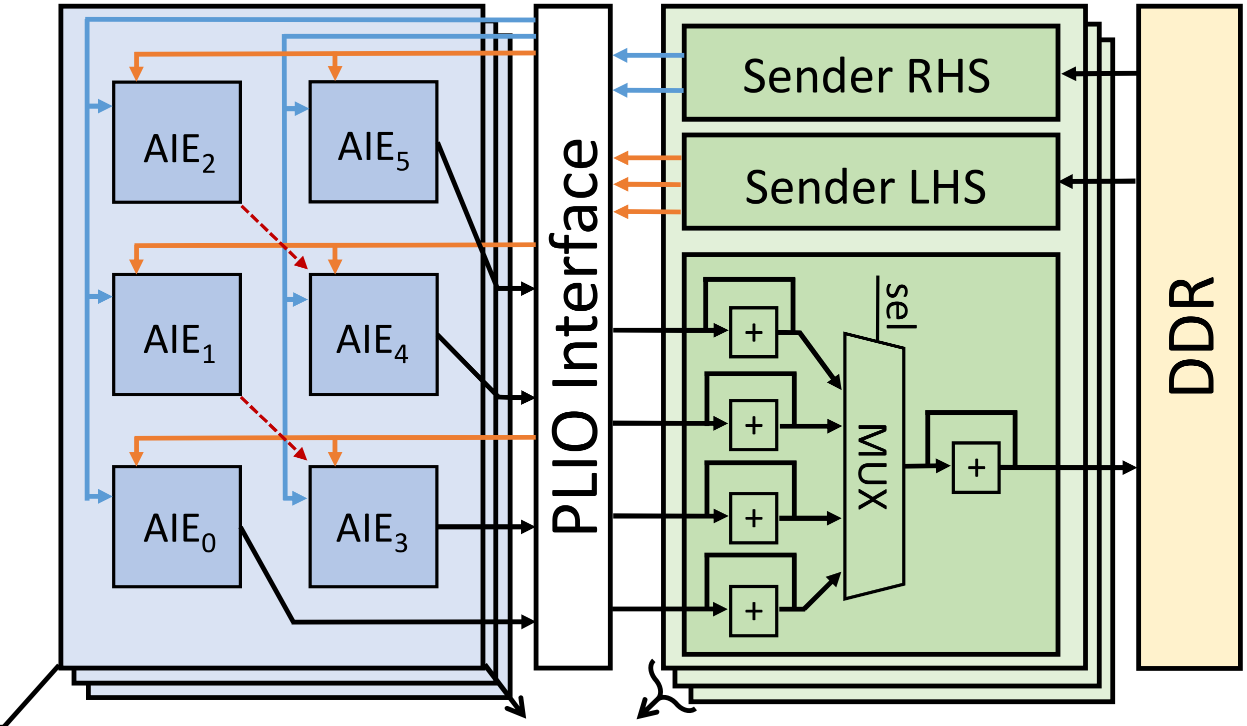
Versal ACAP Architecture



AIM Architecture

- AIM Mapping Strategy 1





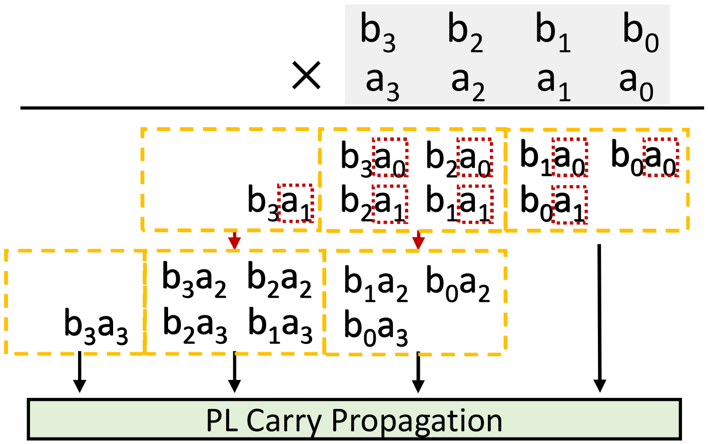
AIM explores Intra-Task Parallelism

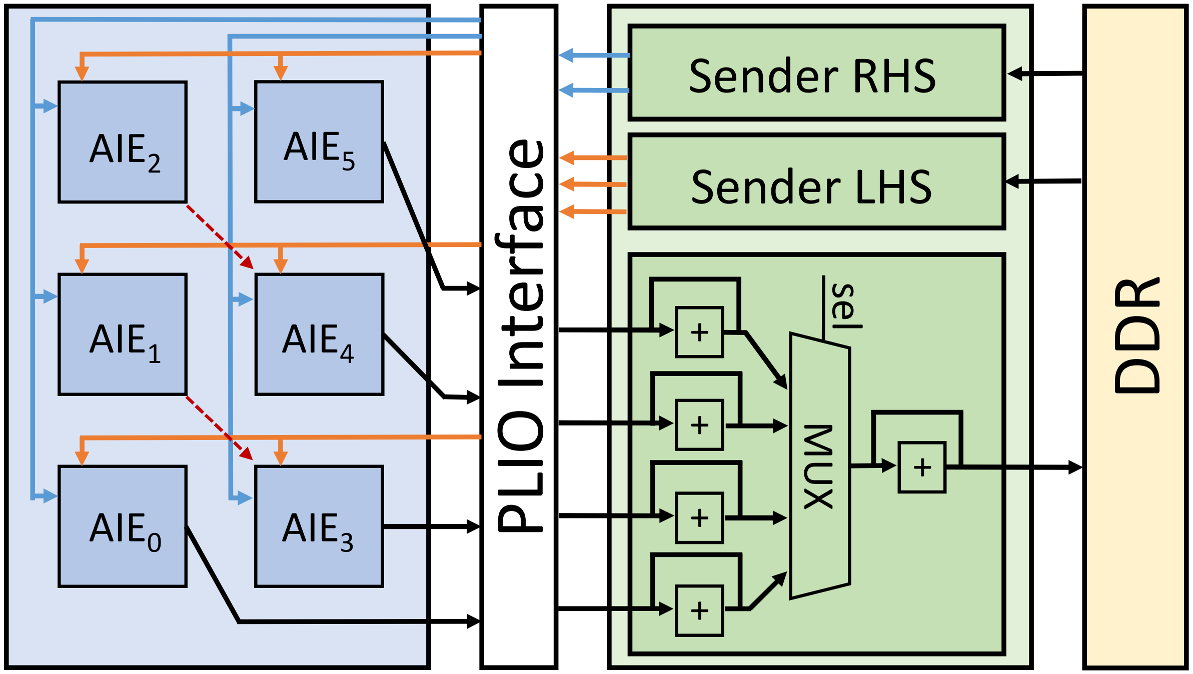
AIM explores Inter-Task Parallelism

Less hardware resources; Use more AIEs; Low AIE kernel efficiency;

AIM Architecture

- AIM Mapping Strategy 1

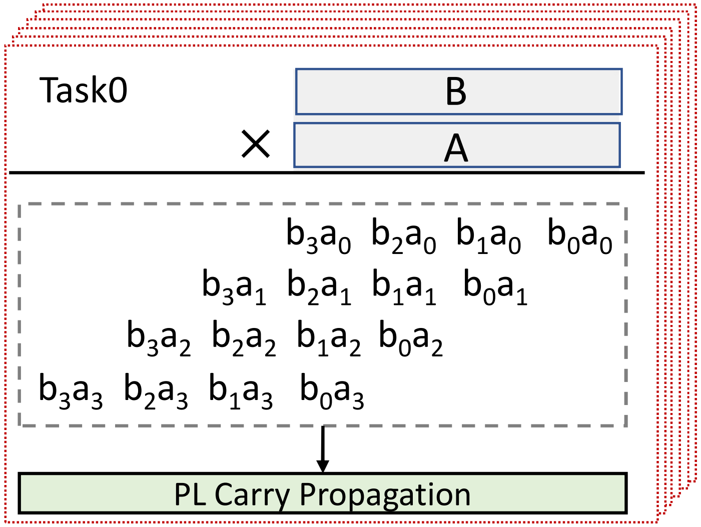

b_3 b_2 b_1 b_0
×
a_3 a_2 a_1 a_0
b_3a_0 b_2a_0 b_1a_0 b_0a_0
b_3a_1 b_2a_1 b_1a_1 b_0a_1
b_3a_2 b_2a_2 b_1a_2 b_0a_2
b_3a_3 b_2a_3 b_1a_3 b_0a_3
PL Carry Propagation

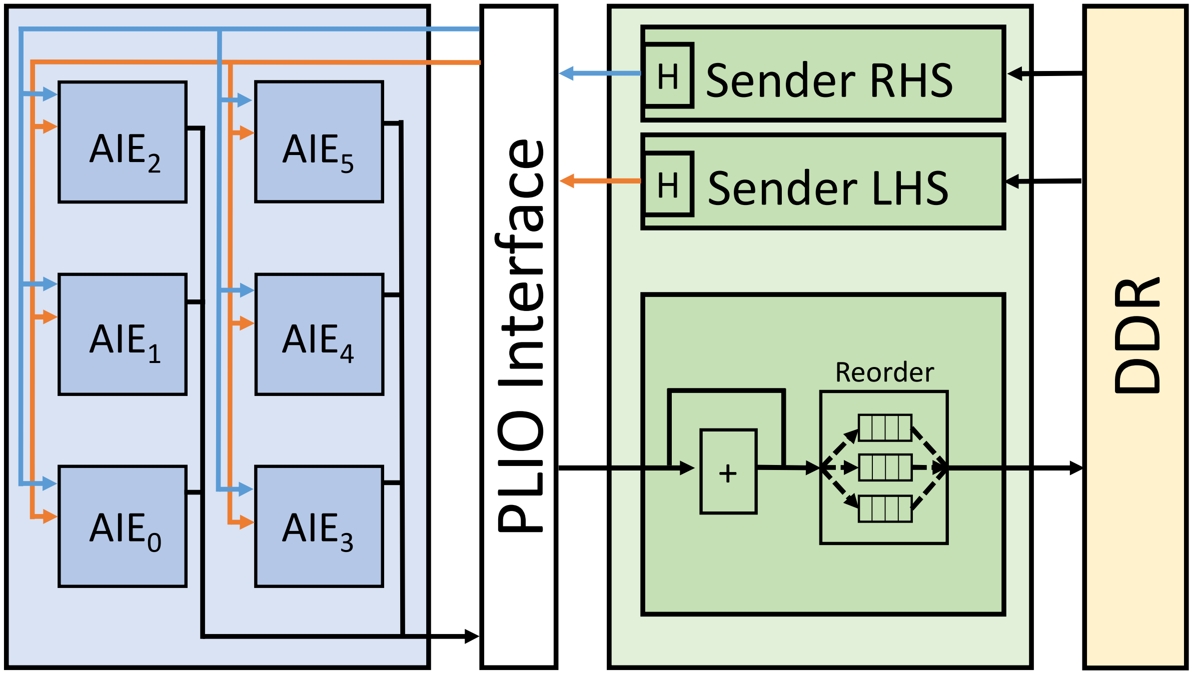


Less hardware resources;⬆ Use more AIEs;⬆ Low AIE kernel efficiency;⬇

AIM Architecture

- AIM Mapping Strategy 2


Task0
B
×
A
b_3a_0 b_2a_0 b_1a_0 b_0a_0
b_3a_1 b_2a_1 b_1a_1 b_0a_1
b_3a_2 b_2a_2 b_1a_2 b_0a_2
b_3a_3 b_2a_3 b_1a_3 b_0a_3
PL Carry Propagation

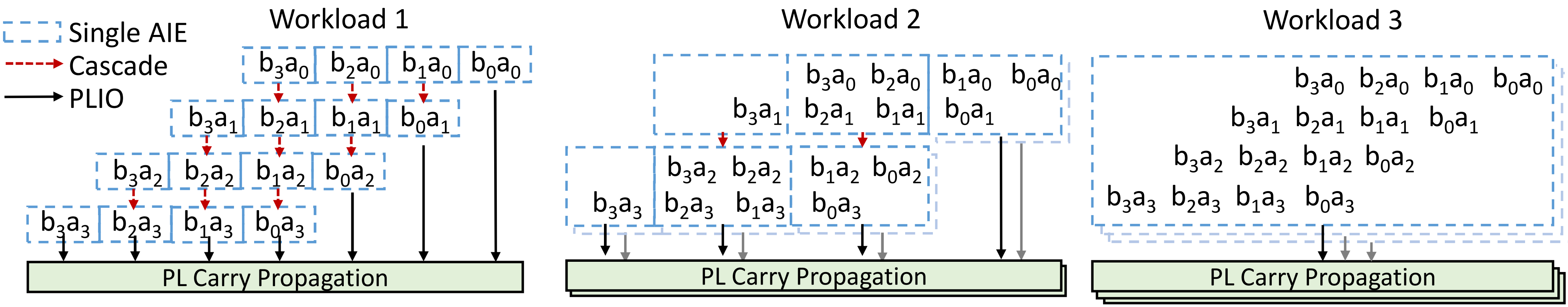


$C_2, C_4, C_1, C_5, C_0, C_3 \Rightarrow C_0, C_1, C_2, C_3, C_4, C_5$

More hardware resources;⬇ Use less AIEs;⬇ High AIE kernel efficiency;⬆

Workload Partition in AIM

- Maximum Intra-task Parallelism
- Maximum Inter-task Parallelism
- Hybrid Parallelism



Single AIE

Cascade

PLIO

Workload 1

b_3a_0 b_2a_0 b_1a_0 b_0a_0

b_3a_1 b_2a_1 b_1a_1 b_0a_1

b_3a_2 b_2a_2 b_1a_2 b_0a_2

b_3a_3 b_2a_3 b_1a_3 b_0a_3

PL Carry Propagation

Workload 2

b_3a_0 b_2a_0 b_1a_0 b_0a_0

b_3a_1 b_2a_1 b_1a_1 b_0a_1

b_3a_2 b_2a_2 b_1a_2 b_0a_2

b_3a_3 b_2a_3 b_1a_3 b_0a_3

PL Carry Propagation

Workload 3

b_3a_0 b_2a_0 b_1a_0 b_0a_0

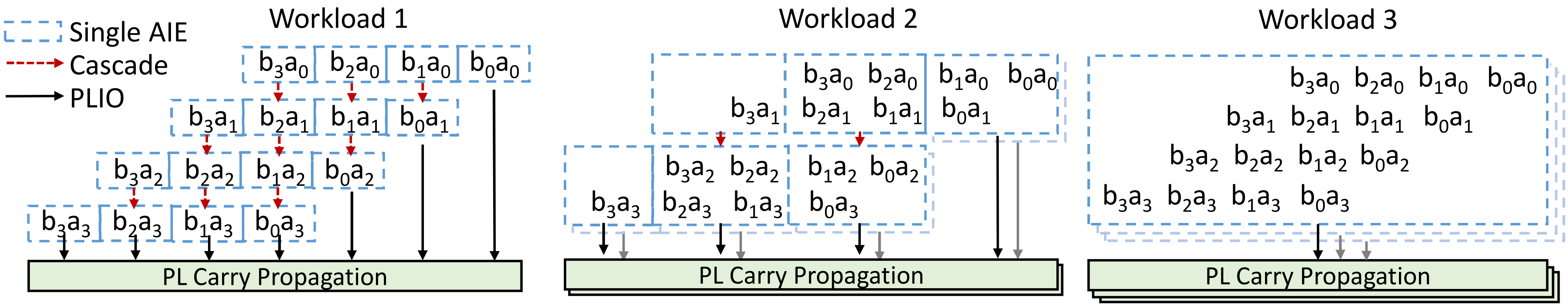
b_3a_1 b_2a_1 b_1a_1 b_0a_1

b_3a_2 b_2a_2 b_1a_2 b_0a_2

b_3a_3 b_2a_3 b_1a_3 b_0a_3

PL Carry Propagation

Workload Partition in AIM


Single AIE
Cascade
PLIO
Workload 1
b_3a_0 b_2a_0 b_1a_0 b_0a_0
b_3a_1 b_2a_1 b_1a_1 b_0a_1
b_3a_2 b_2a_2 b_1a_2 b_0a_2
b_3a_3 b_2a_3 b_1a_3 b_0a_3
PL Carry Propagation
Workload 2
b_3a_0 b_2a_0 b_1a_0 b_0a_0
b_3a_1 b_2a_1 b_1a_1 b_0a_1
b_3a_2 b_2a_2 b_1a_2 b_0a_2
b_3a_3 b_2a_3 b_1a_3 b_0a_3
PL Carry Propagation
Workload 3
b_3a_0 b_2a_0 b_1a_0 b_0a_0
b_3a_1 b_2a_1 b_1a_1 b_0a_1
b_3a_2 b_2a_2 b_1a_2 b_0a_2
b_3a_3 b_2a_3 b_1a_3 b_0a_3
PL Carry Propagation

System level performance of 8192-bit Multiplier

Case	P_{Intra}	P_{Inter}	#bits/AIE	PKT	LUT	BRAM	Tasks/s
1	306	1	496	1	43.6%	4.7%	1.6M
2	30	7	1736	1	78.1%	16.7%	9.6M
3	1	80	8192	4	60.5%	98.6%	5.7M

306 AIEs are used

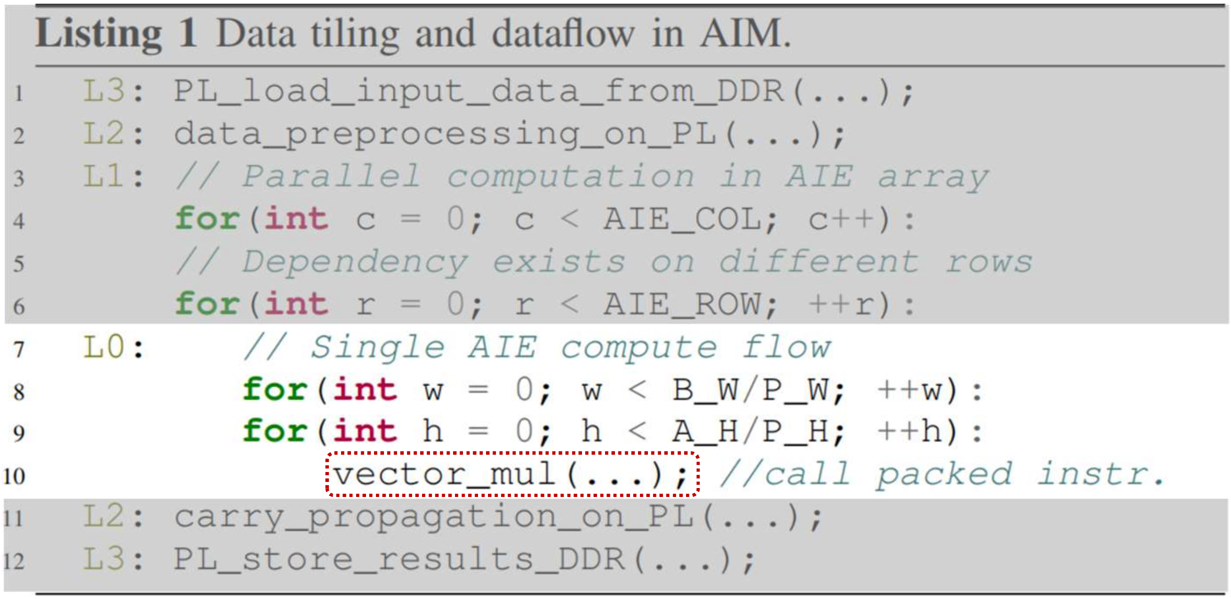
210 AIEs are used ★

Only 80 AIEs are used

Low kernel efficiency

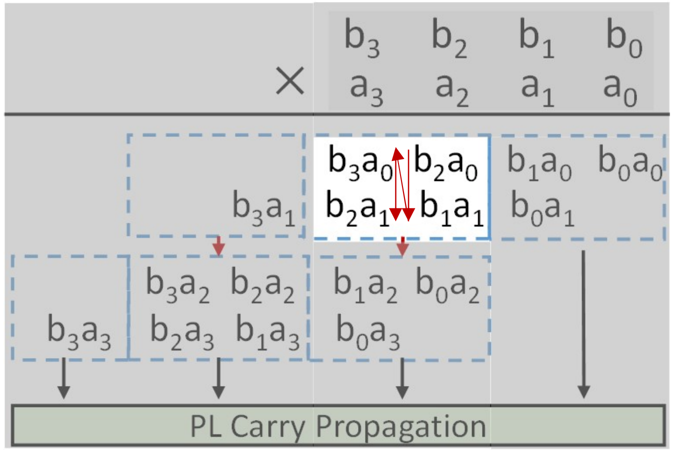
High kernel efficiency

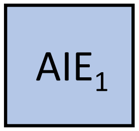
Single AIE Design



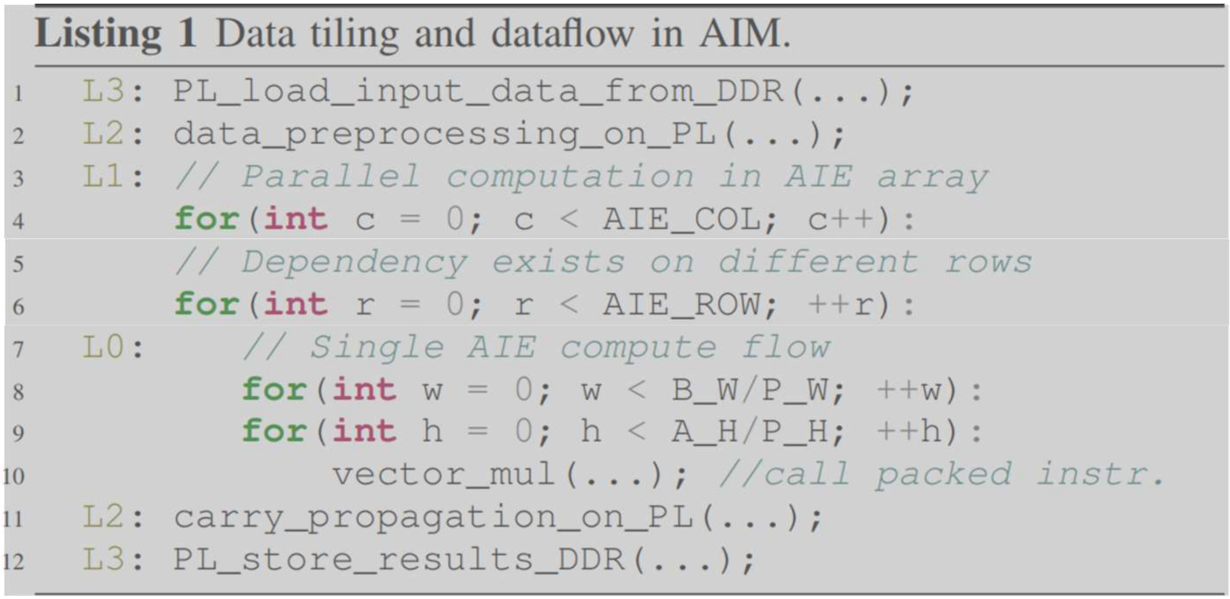
Listing 1 Data tiling and dataflow in AIM.

```
L3: PL_load_input_data_from_DDR(...);
L2: data_preprocessing_on_PL(...);
L1: // Parallel computation in AIE array
    for(int c = 0; c < AIE_COL; c++):
    // Dependency exists on different rows
    for(int r = 0; r < AIE_ROW; ++r):
L0:     // Single AIE compute flow
        for(int w = 0; w < B_W/P_W; ++w):
        for(int h = 0; h < A_H/P_H; ++h):
            vector_mul(...); //call packed instr.
L2: carry_propagation_on_PL(...);
L3: PL_store_results_DDR(...);
```


×
b_3 b_2 b_1 b_0
a_3 a_2 a_1 a_0
b_3a_0 b_2a_0 b_1a_0 b_0a_0
b_3a_1 b_2a_1 b_1a_1 b_0a_1
b_3a_2 b_2a_2 b_1a_2 b_0a_2
b_3a_3 b_2a_3 b_1a_3 b_0a_3
PL Carry Propagation

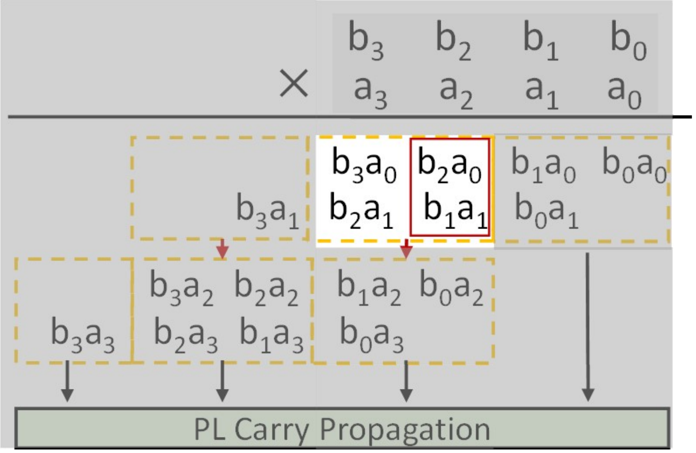


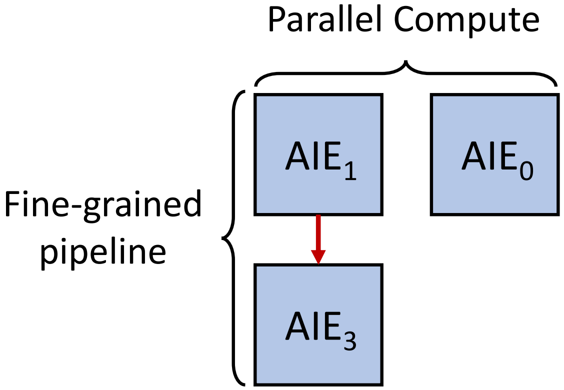
Scaling Out to AIE Array in AIM



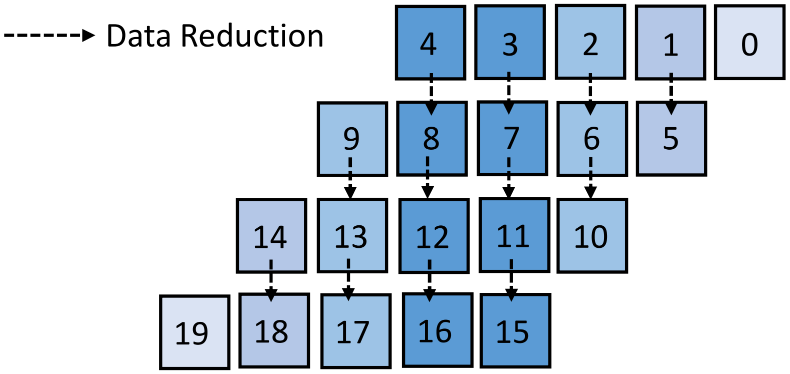
Listing 1 Data tiling and dataflow in AIM.

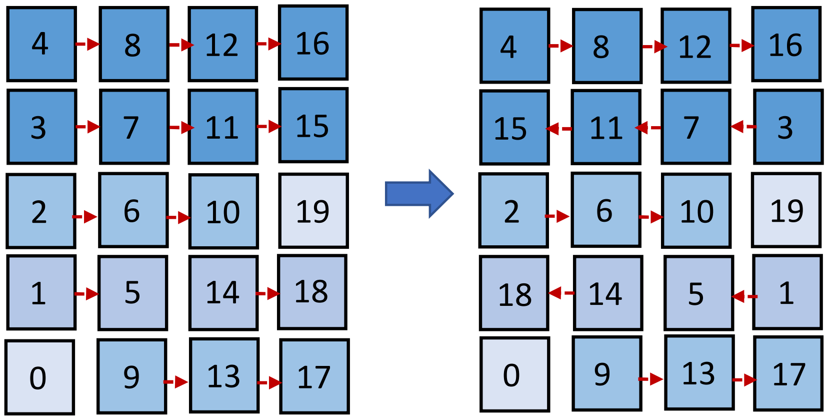
```
L3: PL_load_input_data_from_DDR(...);
L2: data_preprocessing_on_PL(...);
L1: // Parallel computation in AIE array
    for(int c = 0; c < AIE_COL; c++):
    // Dependency exists on different rows
    for(int r = 0; r < AIE_ROW; ++r):
L0:     // Single AIE compute flow
        for(int w = 0; w < B_W/P_W; ++w):
        for(int h = 0; h < A_H/P_H; ++h):
            vector_mul(...); //call packed instr.
L2: carry_propagation_on_PL(...);
L3: PL_store_results_DDR(...);
```

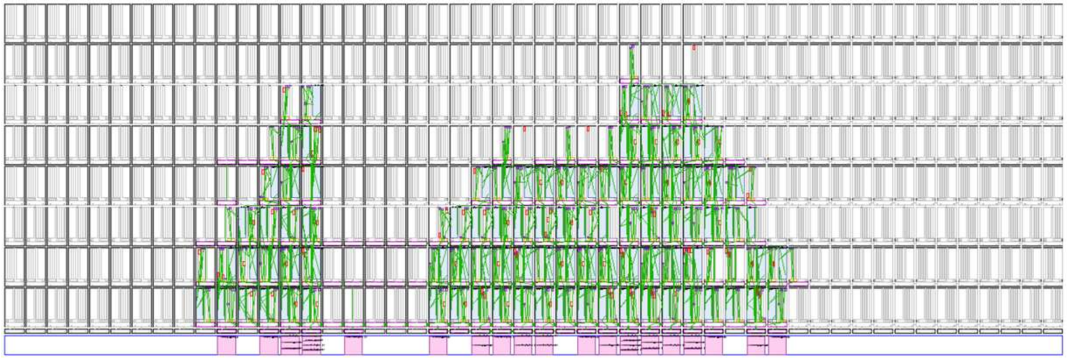




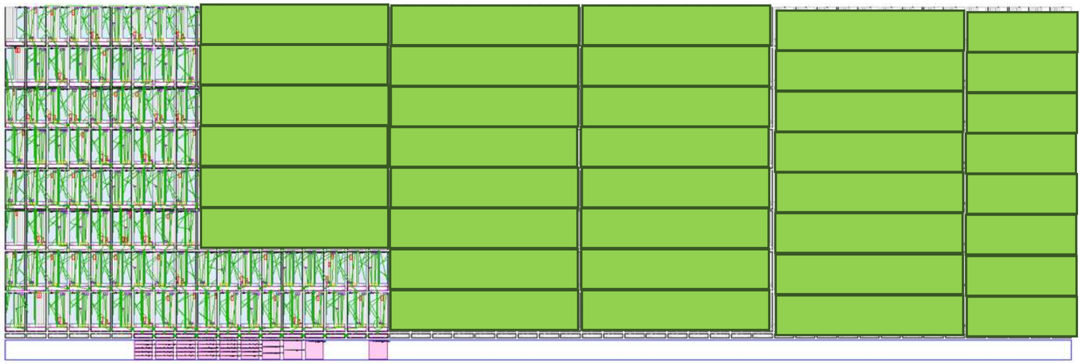
Scaling Out to AIE Array in AIM



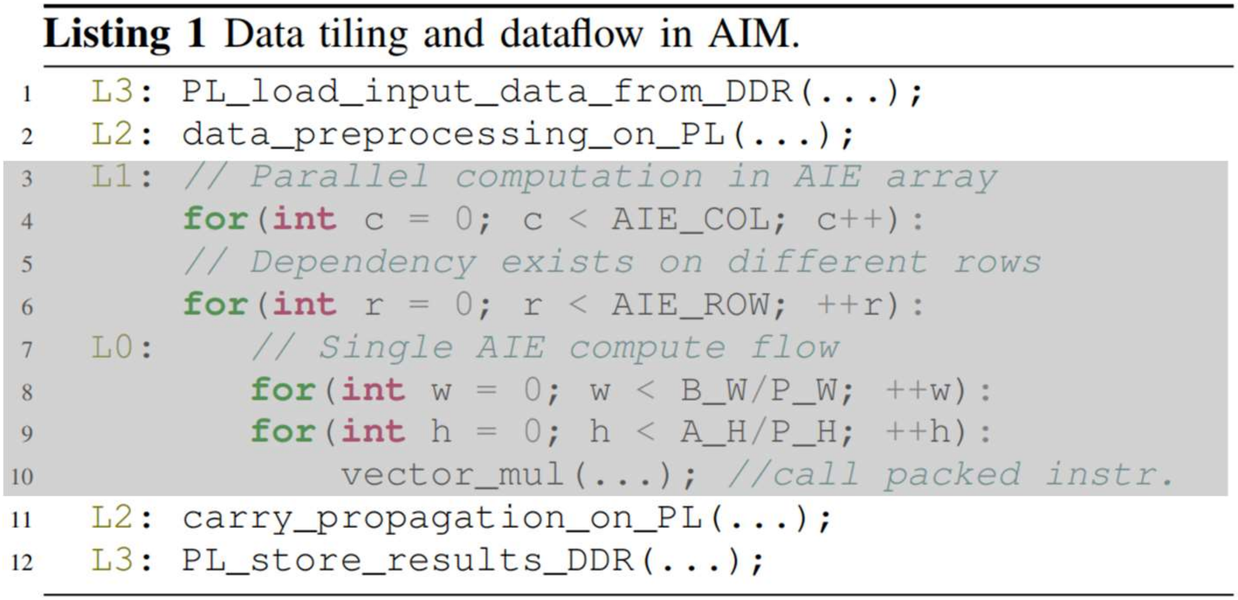




Up to 396 AIEs can be used

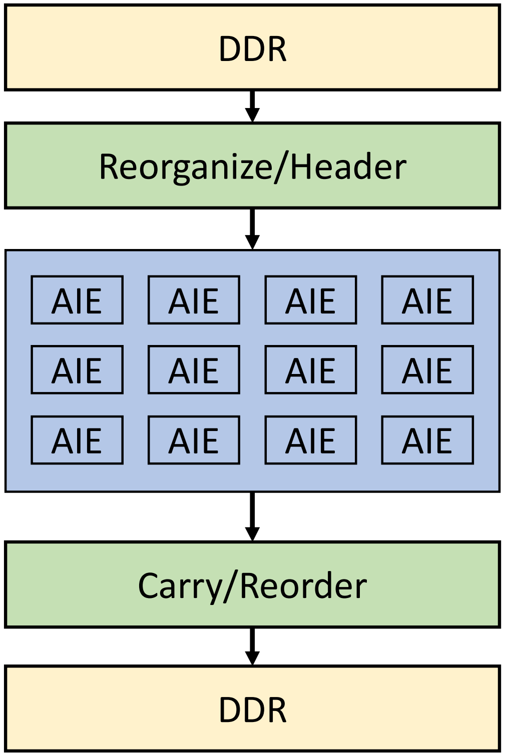


Fine-Grained Pipeline in AIM



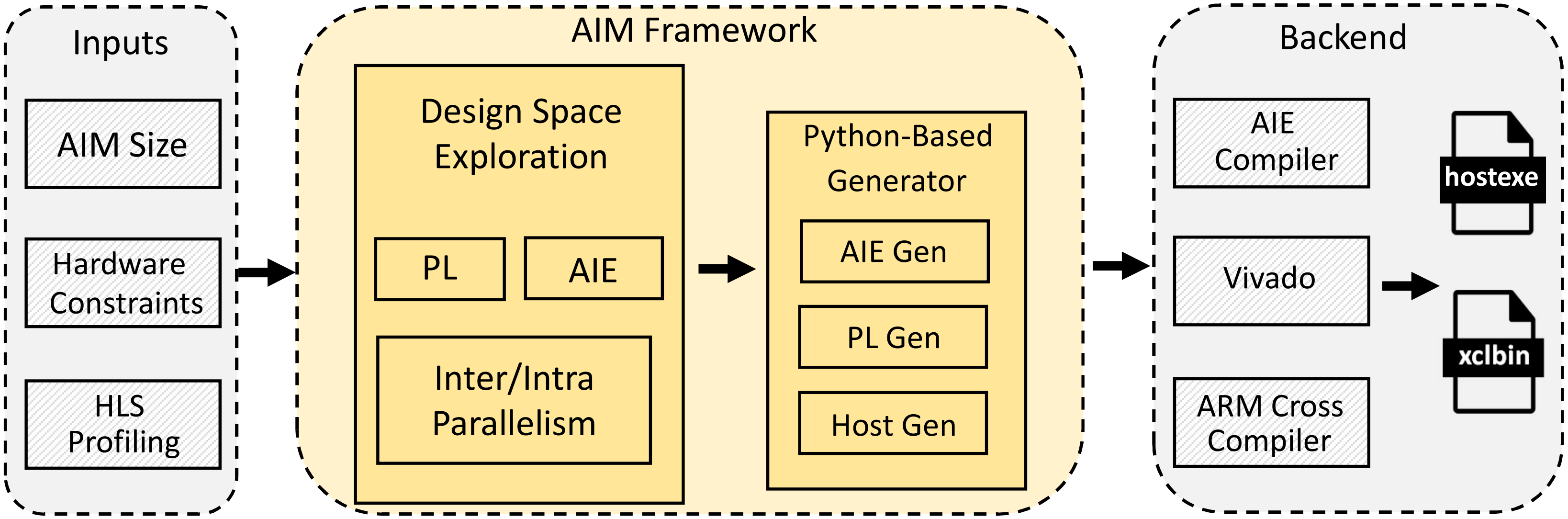
Listing 1 Data tiling and dataflow in AIM.

```
L3: PL_load_input_data_from_DDR(...);
L2: data_preprocessing_on_PL(...);
L1: // Parallel computation in AIE array
    for(int c = 0; c < AIE_COL; c++):
    // Dependency exists on different rows
    for(int r = 0; r < AIE_ROW; ++r):
L0:     // Single AIE compute flow
        for(int w = 0; w < B_W/P_W; ++w):
        for(int h = 0; h < A_H/P_H; ++h):
            vector_mul(...); //call packed instr.
L2: carry_propagation_on_PL(...);
L3: PL_store_results_DDR(...);
```



All modules are coordinated in a fine-grained pipeline

AIM Framework

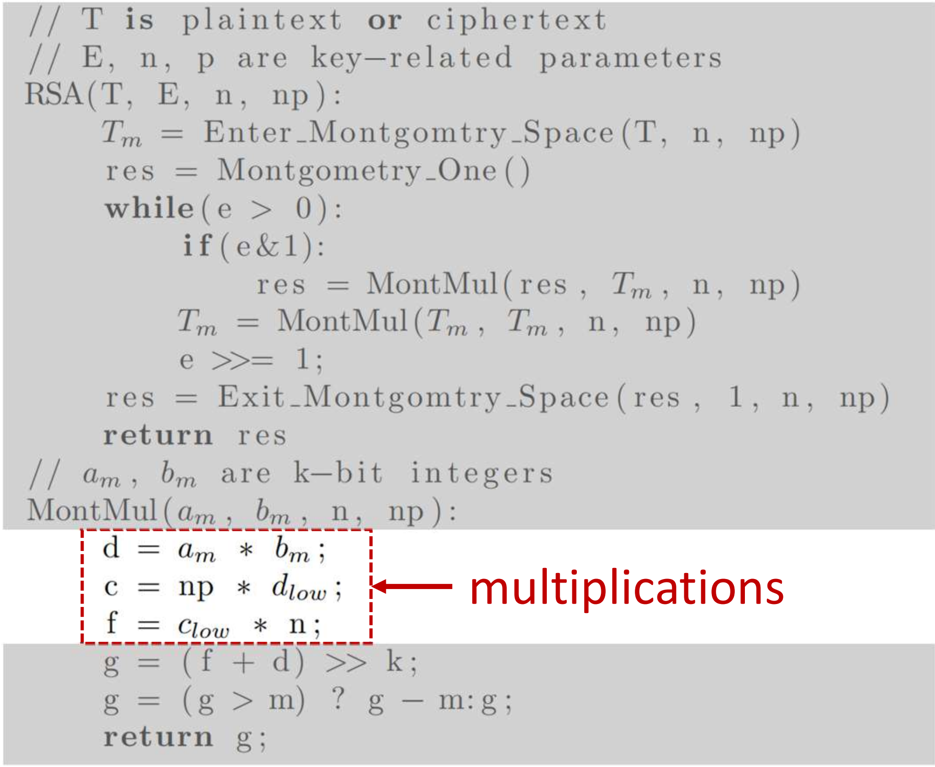


https://github.com/arc-research-lab/AIM

Applications - RSA

$ciphertext = plaintext^{E} \ \% \ N$

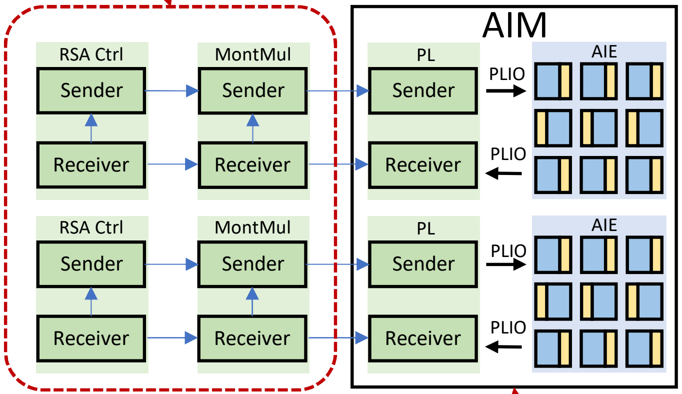
$plaintext = ciphertext^{D} \ \% \ N$



```
// T is plaintext or ciphertext
// E, n, p are key−related parameters
RSA(T, E, n, np):
    $T_m$ = Enter_Montgomtry_Space(T, n, np)
    res = Montgometry_One()
    while(e > 0):
        if(e&1):
            res = MontMul(res, $T_m$, n, np)
        $T_m$ = MontMul($T_m$, $T_m$, n, np)
        e >>= 1;
    res = Exit_Montgomtry_Space(res, 1, n, np)
    return res
// $a_m$, $b_m$ are k−bit integers
MontMul($a_m$, $b_m$, n, np):
    d = $a_m$ * $b_m$;
    c = np * $d_{low}$;
    f = $c_{low}$ * n;
    g = (f + d) >> k;
    g = (g > m) ? g − m:g;
    return g;
```

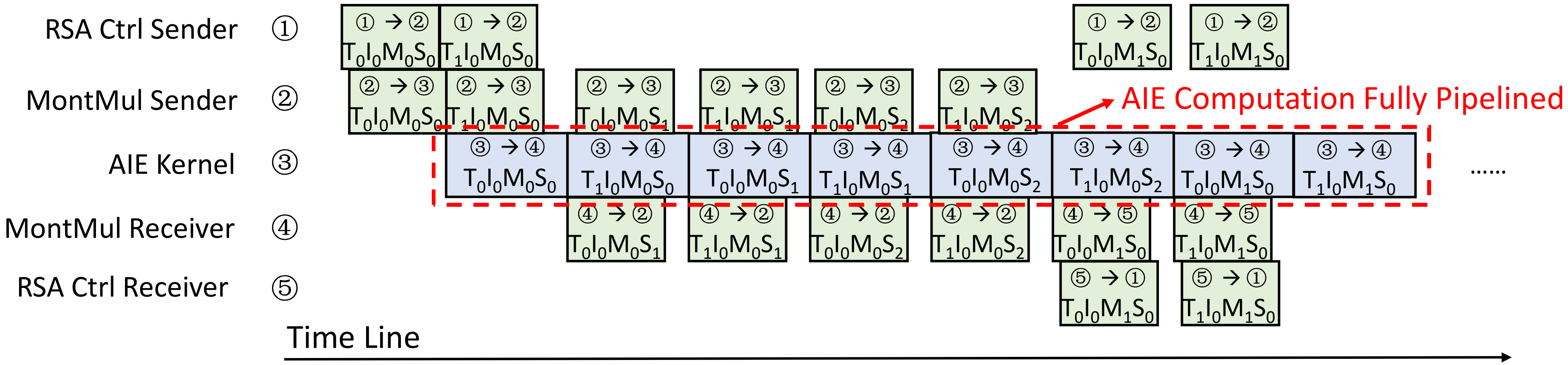
multiplications

Application specific control logic can be easily integrated into the pipeline



optimized multiplication kernels

Applications - RSA



RSA Ctrl Sender ①
① → ② $T_0I_0M_0S_0$
① → ② $T_1I_0M_0S_0$
① → ② $T_0I_0M_1S_0$
① → ② $T_1I_0M_1S_0$

MontMul Sender ②
② → ③ $T_0I_0M_0S_0$
② → ③ $T_1I_0M_0S_0$
② → ③ $T_0I_0M_0S_1$
② → ③ $T_1I_0M_0S_1$
② → ③ $T_0I_0M_0S_2$
② → ③ $T_1I_0M_0S_2$

AIE Kernel ③
③ → ④ $T_0I_0M_0S_0$
③ → ④ $T_1I_0M_0S_0$
③ → ④ $T_0I_0M_0S_1$
③ → ④ $T_1I_0M_0S_1$
③ → ④ $T_0I_0M_0S_2$
③ → ④ $T_1I_0M_0S_2$
③ → ④ $T_0I_0M_1S_0$
③ → ④ $T_1I_0M_1S_0$
……

MontMul Receiver ④
④ → ② $T_0I_0M_0S_1$
④ → ② $T_1I_0M_0S_1$
④ → ② $T_0I_0M_0S_2$
④ → ② $T_1I_0M_0S_2$
④ → ⑤ $T_0I_0M_1S_0$
④ → ⑤ $T_1I_0M_1S_0$

RSA Ctrl Receiver ⑤
⑤ → ① $T_0I_0M_1S_0$
⑤ → ① $T_1I_0M_1S_0$

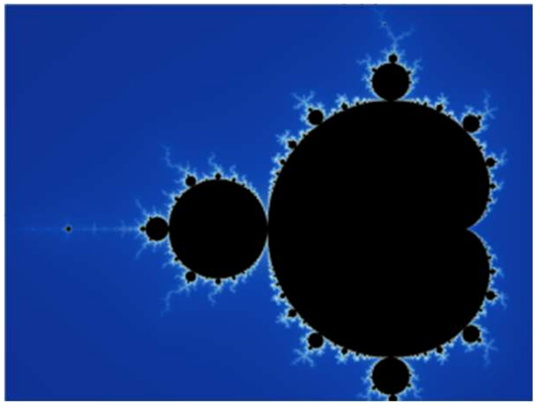
AIE Computation Fully Pipelined

Time Line

Applications - Mandelbrot Set

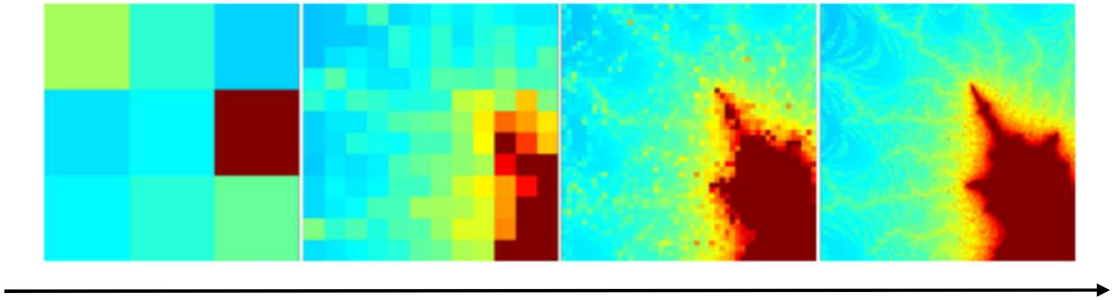
Mandelbrot set

- divergence tests of sampled points on the complex plane
- detailed structures at arbitrary precision



$$f_c(0), f_c(f_c(0)), f_c\left(f_c\left(f_c(f_c(0))\right)\right), ...$$

$$f_c(z) = Z^2 + C$$



Increase Precision

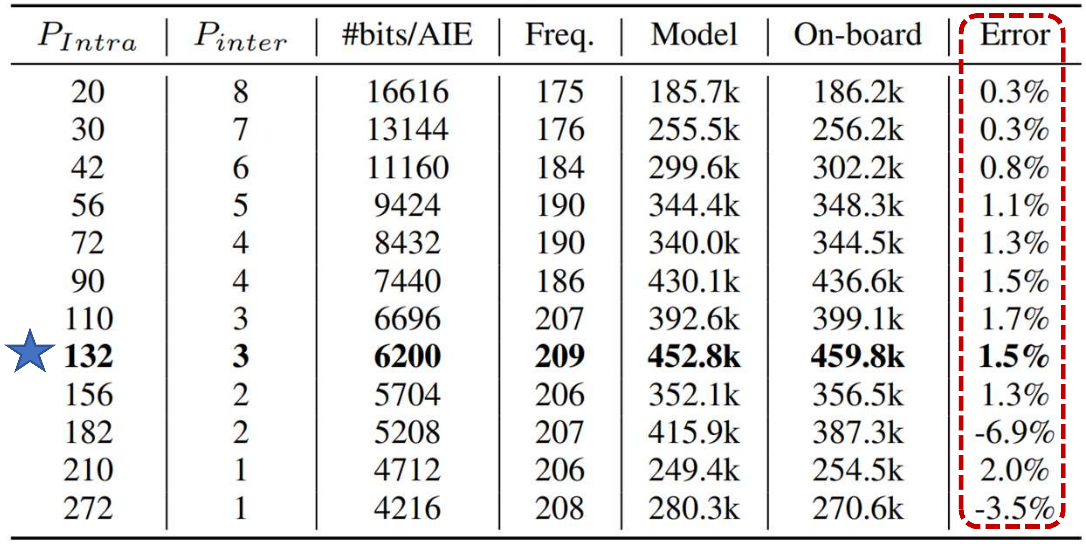
Experiment setup

- Implemented Platform: AMD Versal VCK190
- Frequency: AIE@1GHz, PL@160~220MHz
- Software Tools: Vitis 2021.1
- Applications: Large Integer Multiply, RSA, Mandelbrot Set
- Baseline
  - CPU: Intel Xeon Gold 6346, GMP 6.2.1
  - GPU: NVIDIA A5000, CGBN 2.0

Analytical Model Accuracy

- AIM predicts performance accurately for different configurations
- AIM can find optimal configuration quickly

Comparison between AIM modeling & on-board measurement

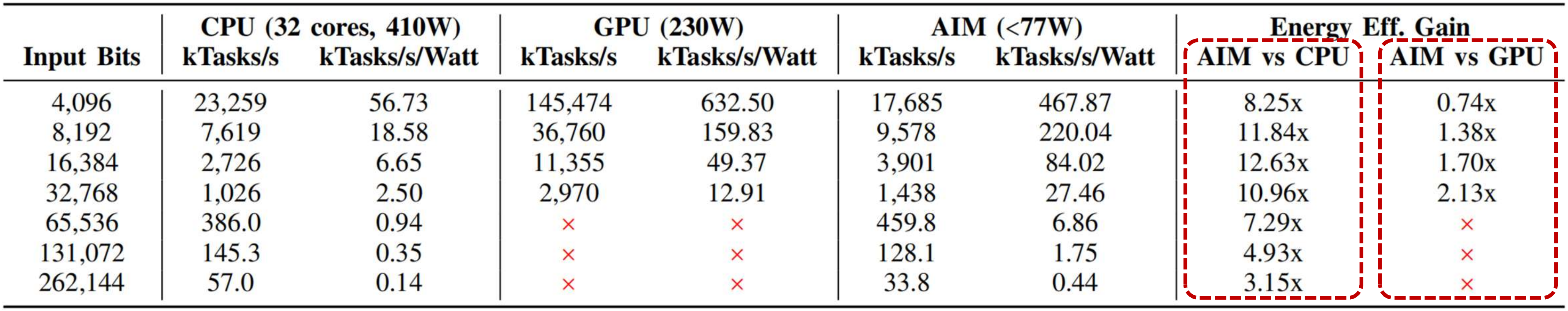


P_{Intra}	P_{inter}	#bits/AIE	Freq.	Model	On-board	Error
20	8	16616	175	185.7k	186.2k	0.3%
30	7	13144	176	255.5k	256.2k	0.3%
42	6	11160	184	299.6k	302.2k	0.8%
56	5	9424	190	344.4k	348.3k	1.1%
72	4	8432	190	340.0k	344.5k	1.3%
90	4	7440	186	430.1k	436.6k	1.5%
110	3	6696	207	392.6k	399.1k	1.7%
★ **132**	**3**	**6200**	**209**	**452.8k**	**459.8k**	**1.5%**
156	2	5704	206	352.1k	356.5k	1.3%
182	2	5208	207	415.9k	387.3k	-6.9%
210	1	4712	206	249.4k	254.5k	2.0%
272	1	4216	208	280.3k	270.6k	-3.5%

Performance & Energy Eff. Comparison

- AIM is more energy efficient than CPU and GPU
- AIM supports much larger multiplications than GPU

Comparison among optimal AIM implementation, Intel Xeon 6346 CPU, and Nvidia A5000 GPU for LIM with input sizes from 4,096 to 262,144 bit

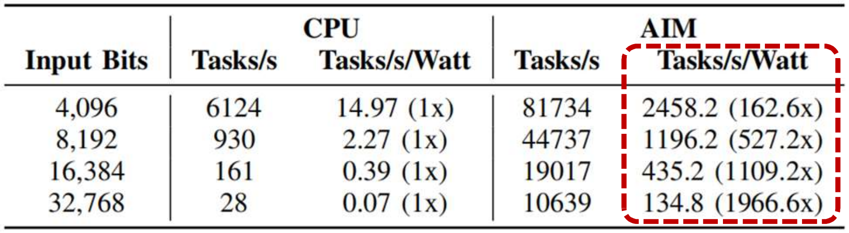


Input Bits	CPU (32 cores, 410W)		GPU (230W)		AIM (<77W)		Energy Eff. Gain	
	kTasks/s	kTasks/s/Watt	kTasks/s	kTasks/s/Watt	kTasks/s	kTasks/s/Watt	AIM vs CPU	AIM vs GPU
4,096	23,259	56.73	145,474	632.50	17,685	467.87	8.25x	0.74x
8,192	7,619	18.58	36,760	159.83	9,578	220.04	11.84x	1.38x
16,384	2,726	6.65	11,355	49.37	3,901	84.02	12.63x	1.70x
32,768	1,026	2.50	2,970	12.91	1,438	27.46	10.96x	2.13x
65,536	386.0	0.94	×	×	459.8	6.86	7.29x	×
131,072	145.3	0.35	×	×	128.1	1.75	4.93x	×
262,144	57.0	0.14	×	×	33.8	0.44	3.15x	×

Performance & Energy Eff. Comparison

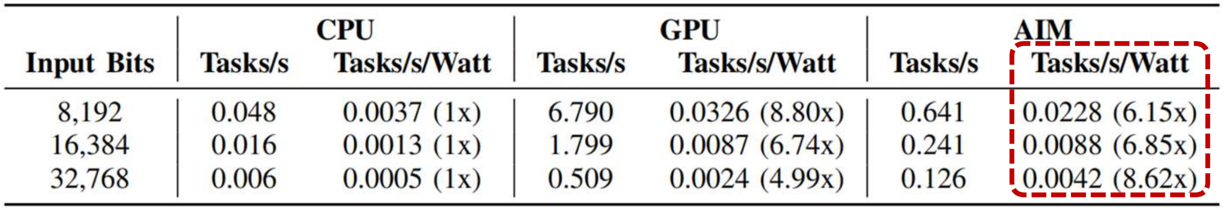
- AIM is efficient in end-to-end applications

Performance & energy efficiency comparison between CPU GMP and AIM for RSA



Input Bits	CPU		AIM	
	Tasks/s	Tasks/s/Watt	Tasks/s	Tasks/s/Watt
4,096	6124	14.97 (1x)	81734	2458.2 (162.6x)
8,192	930	2.27 (1x)	44737	1196.2 (527.2x)
16,384	161	0.39 (1x)	19017	435.2 (1109.2x)
32,768	28	0.07 (1x)	10639	134.8 (1966.6x)

Performance & energy efficiency comparison between CPU GMP, GPU CGBN and AIM for plotting Mandelbrot set



Input Bits	CPU		GPU		AIM	
	Tasks/s	Tasks/s/Watt	Tasks/s	Tasks/s/Watt	Tasks/s	Tasks/s/Watt
8,192	0.048	0.0037 (1x)	6.790	0.0326 (8.80x)	0.641	0.0228 (6.15x)
16,384	0.016	0.0013 (1x)	1.799	0.0087 (6.74x)	0.241	0.0088 (6.85x)
32,768	0.006	0.0005 (1x)	0.509	0.0024 (4.99x)	0.126	0.0042 (8.62x)

ARC-LAB

University of Pittsburgh

THANK YOU!

LPS LABORATORY FOR PHYSICAL SCIENCES

NSF National Science Foundation

AMD XILINX

Email: zhuoping.yang@pitt.edu
peipei.zhou@pitt.edu

Github Repo: https://github.com/arc-research-lab/AIM