PRNGine: Massively Parallel Pseudo–Random Number Generation and Probability Distribution Approximations on AMD AI Engines

Mohamed Bouaziz, Suhaib A. Fahmy

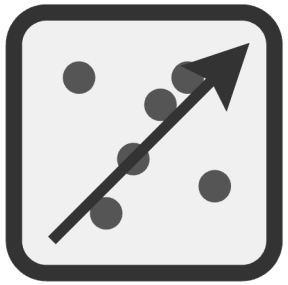
Accelerated Connected Computing Lab (**ACCL**),

King Abdullah University of Science and Technology (**KAUST**), Saudi Arabia

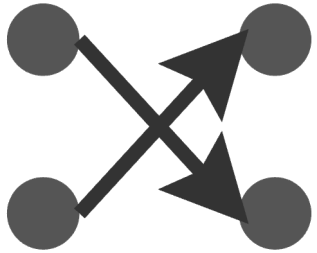
Pseudo–Random Number Generation/ors

- Random numbers are crucial for running many applications
- Some applications require a massive generation of random numbers in parallel

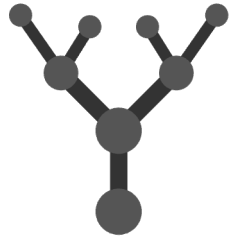
Cryptography



Monte Carlo simulation



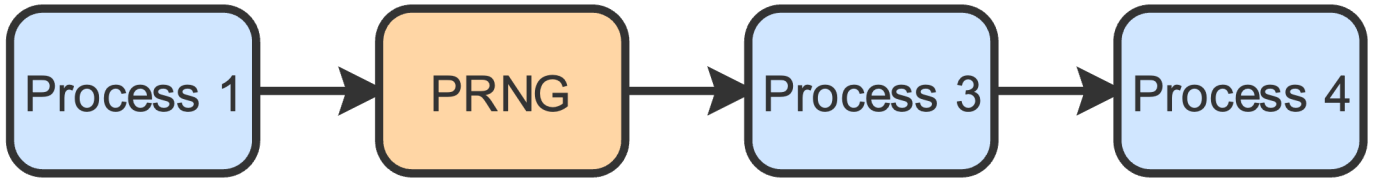
Data shuffling



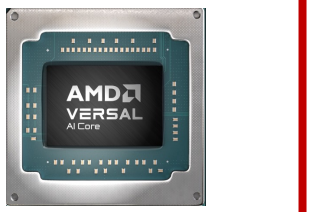
Procedural content generation

PRNGs in Processing Pipelines

- CGRAs/RDAs are well-suited for running dataflows in pipelines.



- There is an increasing demand for such dataflow accelerators.



AMD AI Engines[1]

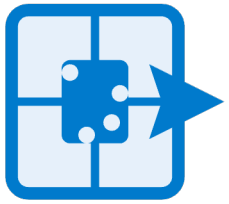
Groq[2]

SambaNova[3]

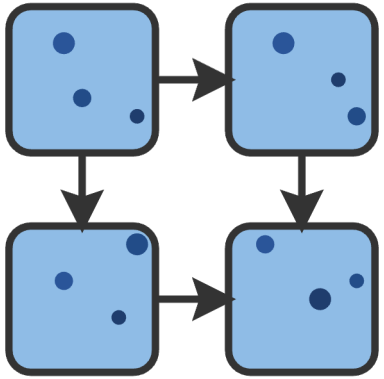
1: https://www.amd.com/en/products/adaptive-socs-and-fpgas/technologies/ai-engine.html
2: https://groq.com/
3: https://sambanova.ai/

Existing work

- LUT-optimised generators on FPGAs ➡️ fine-grained bitwise manipulations.

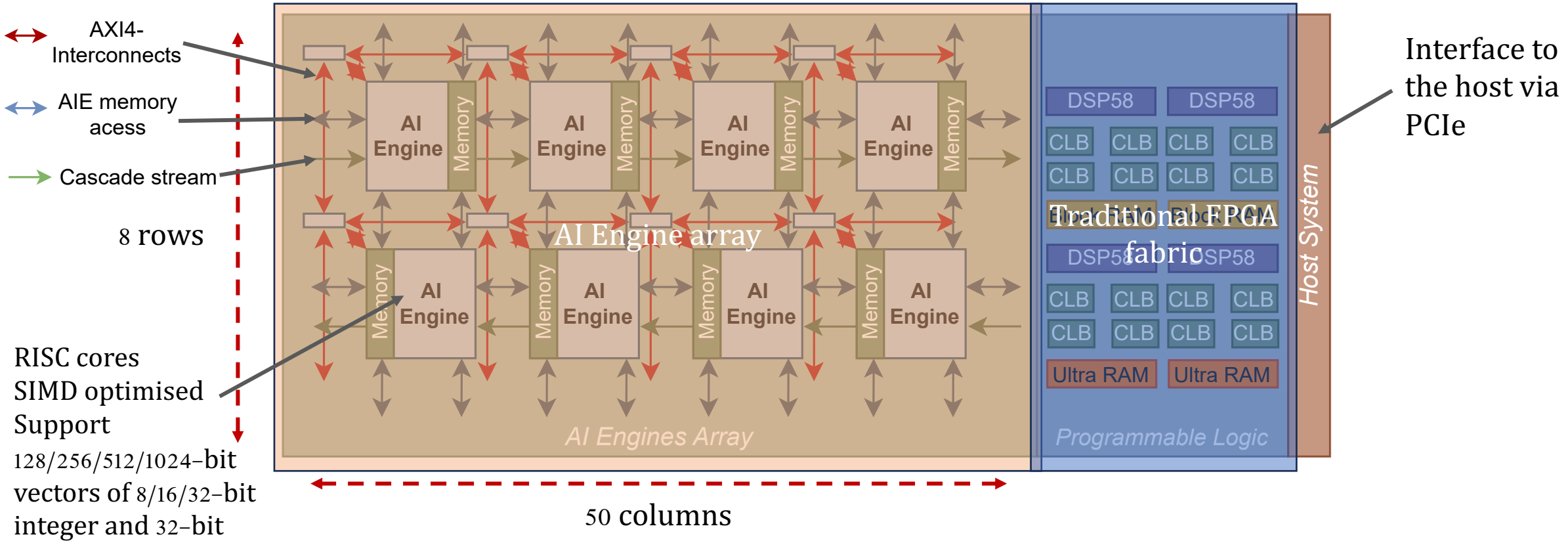


- PRNG implementation on Ambric AM2045 ➡️ An example mapping PRNG on a coarse-grained RISC-based architecture



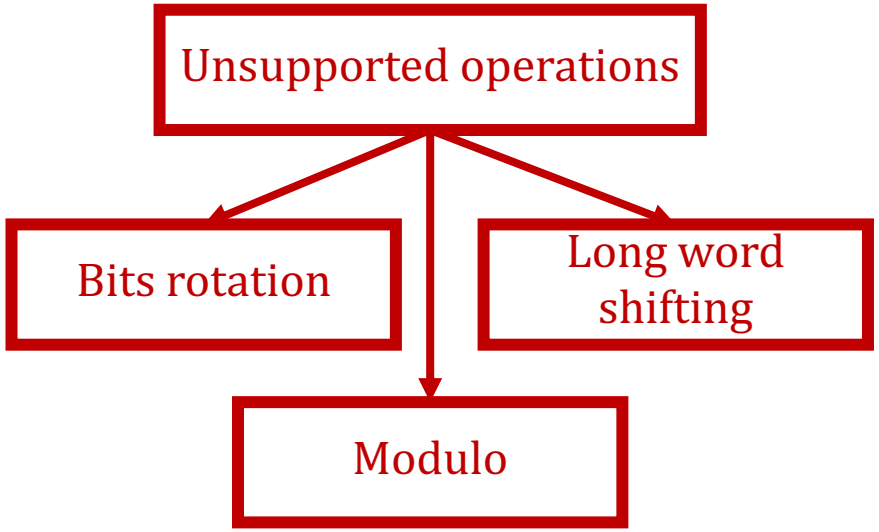
AMD AI Engines architecture is also a RISC-based coarse-grained architecture with more capabilities!

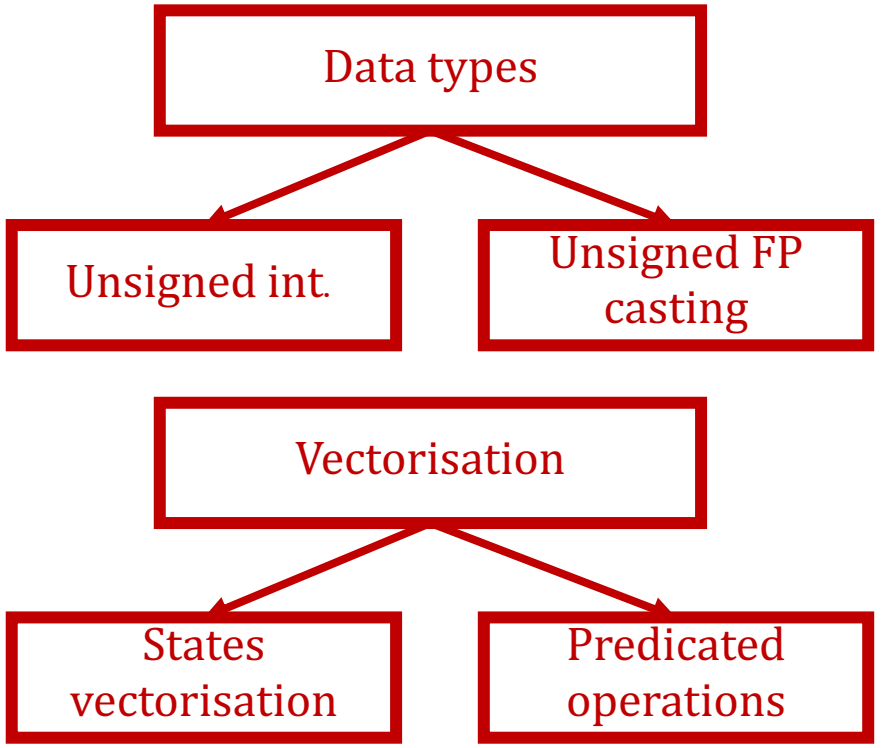
AMD Versal SoC architecture



- RISC cores
- SIMD optimised
- Support 128/256/512/1024-bit vectors of 8/16/32-bit integer and 32-bit FP elements.

Challenges in mapping PRNGs and Probability distribution approximations on AI Engines





Main Contributions

- Propose an efficient implementation of the missed features.
- Propose the dataflow overlays of two execution models:
  - Co–processor model
  - Standalone accelerator model

PRNG Execution Requirements

		Data type	Required operations	Vectorisation
CUDA's by-default PRNG	XORWOW	32-bit unsigned int.		
	XOROSHIRO	64-bit unsigned int.	64-bit operations	
Vectorised version of MT	SFMT	128-bit unsigned int.	128-bit operations Modulo operation	128-bit mask

PRNG Execution Requirements, contd...

unsigned integer => Bitwise operations are insensitive to the sign-bit

Algorithm 1 XORWOW Random Number Generation

Input: $s_0, \cdots, s_4$, $\#RNs$
Output: RNs
for $i \leftarrow 0$ to $\#RNs$ **do**
  $cnt \leftarrow cnt + 362437$
  $value \leftarrow s_4$
  $value \leftarrow value \oplus (value >> 2)$
  $value \leftarrow value \oplus (value << 1)$
  $value \leftarrow value \oplus (s_0 \oplus (s_0 << 4))$
  $value \leftarrow value + cnt$ ▷ Weyl sequence
  $RNs[i] \leftarrow value$
  $s_4 \leftarrow s_3, s_3 \leftarrow s_2, \cdots, s_1 \leftarrow s_0$
  $s_0 \leftarrow value$
end for

Algorithm 2 XOROSHIRO Random Number Generation

Input: s_0, s_1, $\#RNs$
Output: RNs
for $i \leftarrow 0$ to $\#RNs$ **do**
  $RNs[i] \leftarrow \text{rotate\_left}(s_0 + s_1, 17) + s_0$
  $s_1 \leftarrow s_0 \oplus s_1$
  $s_0 \leftarrow \text{rotate\_left}(s_0, 49) \oplus s_1 \oplus (s_1 << 21)$
  $s_1 \leftarrow \text{rotate\_left}(s_1, 37)$
end for

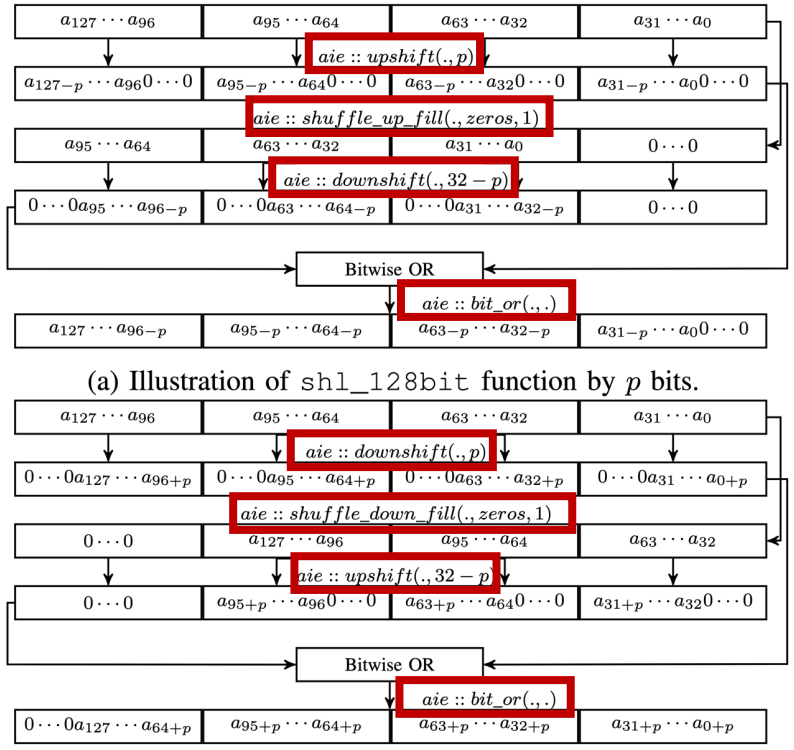
- *Rotation op.*
- *64-bit ops.*

Algorithm 3 SFMT Random Number Generation

Macros: For MT19937 based algorithms, $N = 156$ and $M = 122$
Input: $s_0, s_1, \cdots, s_N$, $\#RNs$
Output: RNs
$r_1 \leftarrow s_{N-2}, r_2 \leftarrow s_{N-1}$
for $i \leftarrow 0$ to $\#RNs$ **do**
  $v_0 \leftarrow s_i, v_M \leftarrow s_{(i+M) \bmod N}$
  $A \leftarrow (v0 \overset{128}{\underset{32}{<<}} 8) \oplus v_0$
  $B \leftarrow (vM >> 11)\ \&\ vMASK$
  $C \leftarrow r_1 \overset{128}{\underset{32}{>>}} 8$
  $D \leftarrow r_2 << 18$
  $r_1 \leftarrow r_2$
  $r2 \leftarrow A \oplus B \oplus C \oplus D$
  $RNs[i] \leftarrow r_2$
  $s_{N-1} \leftarrow r_2$
end for

- *Long word shifting*
- *Modulo*
- *128-bit ops.*

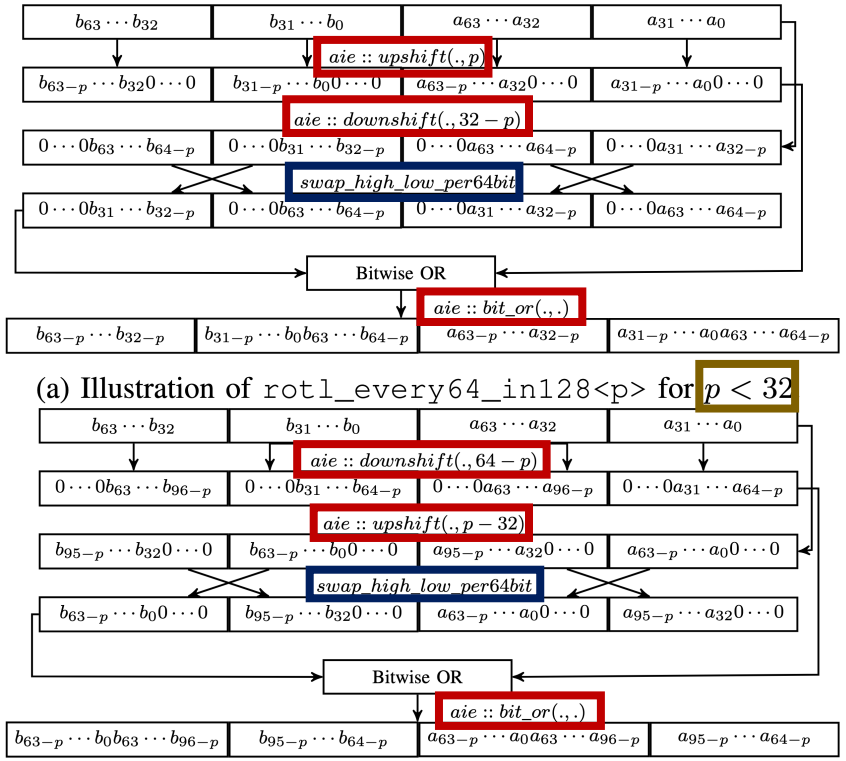
PRNG Execution Requirements, contd...



(a) Illustration of `shl_128bit` function by p bits.

(b) Illustration of `shr_128bit` function by p bits.

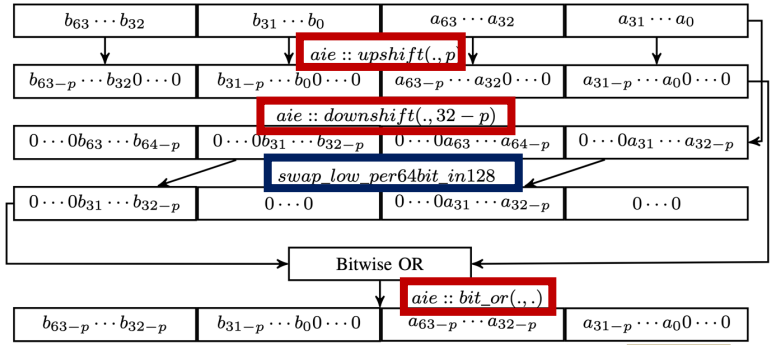
Fig. 2: 128-bit shift operations.



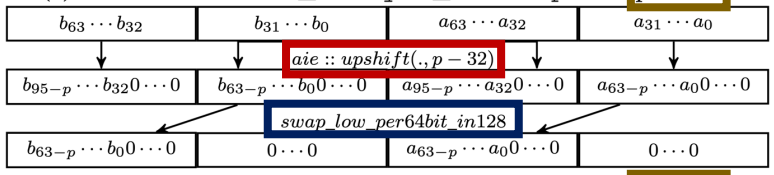
(a) Illustration of `rotl_every64_in128<p>` for $p < 32$

(b) Illustration of `rotl_every64_in128<p>` for $p \geq 32$

Fig. 3: $2 \times$ 64-bit rotate left operation.



(a) Illustration of `shl_every64_in128<p>` for $p < 32$.



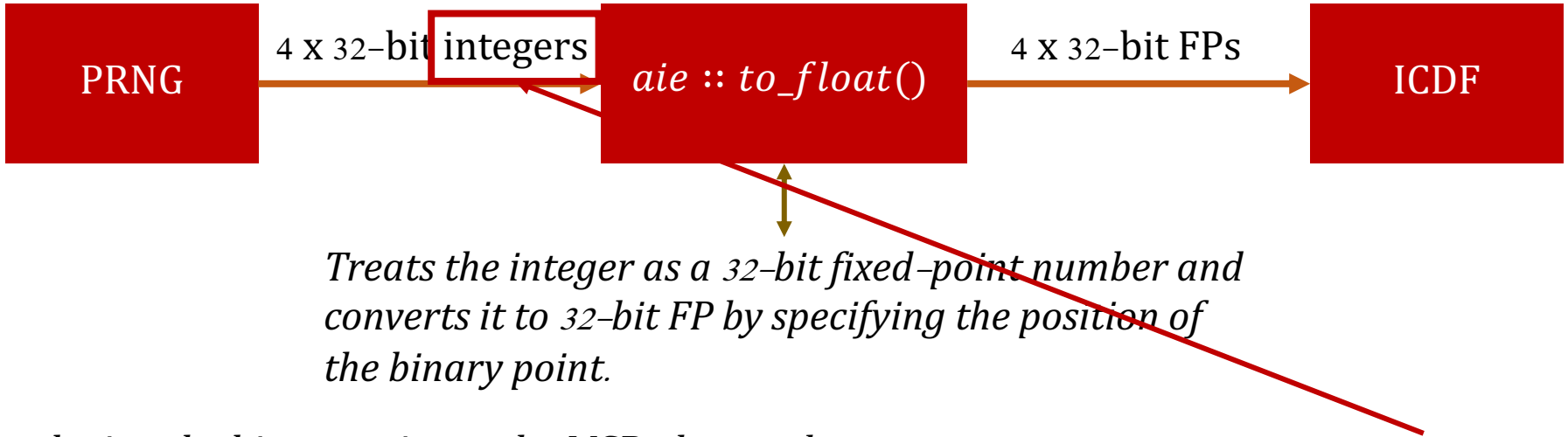
(b) Illustration of `shl_every64_in128<p>` for $p \geq 32$.

Fig. 4: $2 \times$ 64-bit left shift operation.

The same follows for the right shift

Probability Distribution Approximations Requirements

Approximation of probability distribution



→ *By placing the binary point at the MSB, the numbers will be in* [0,1)

BUT!! The data types are considered signed!!

→ Integers with MSB = 1 are misread as negative which results in negative FP.

Probability Distribution Approximations Requirements

Single–precision (32–bit) floating point representation

$$F = (-1)^{MSB} \times M \times 2^{E-128} \qquad 1 \leq M < 2$$

We want to extract the MSB as well

Probability Distribution Approximations Requirements

$$P(R(X)) = \frac{\left(\left(\left(\left((A_1R^2 + A_2)R^2 + A_3\right)R^2 + A_4\right)R^2 + A_5\right)R^2 + A_6\right)R}{\left(\left(\left((B_1R^2 + B_2)R^2 + B_3\right)R^2 + B_4\right)R^2 + B_5\right)R^2 + 1}$$

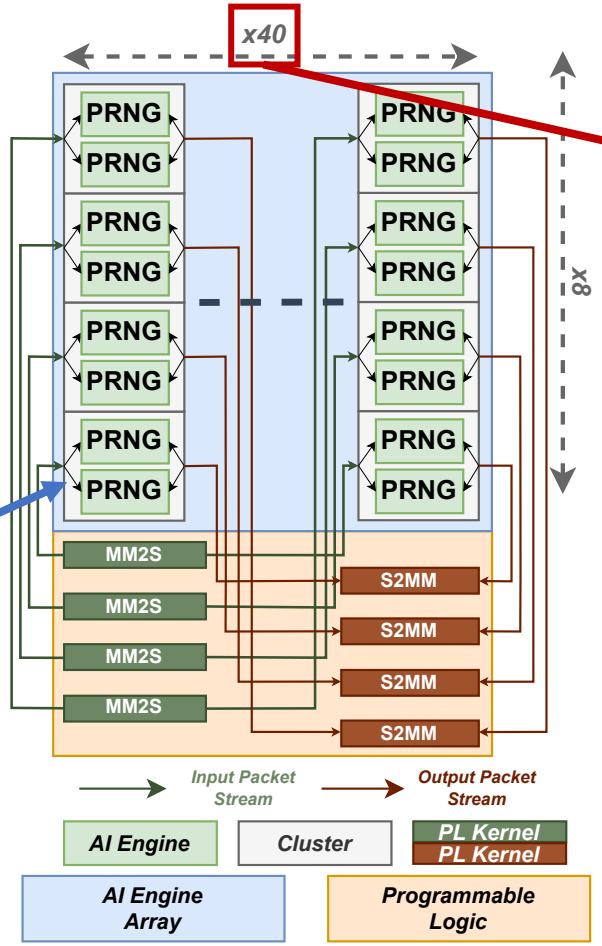
where $R(X) = X - 0.5$ *(coefficients in the paper)*

- *Inner part of Acklam's approximation*
- *SIMD compliant*

Experimental Setup

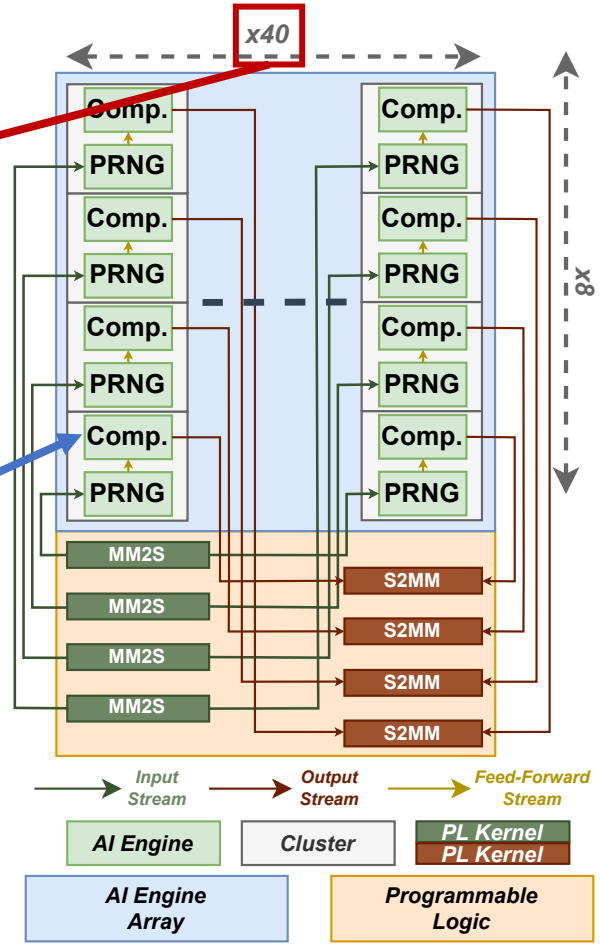
M1
Co-processor model

Two PRNGs are packet – streamed



Only 40/50 columns

Kernel that uses the random numbers (e. g. ICDF)



M2
Standalone accelerator

Experimental Results

PRNG	Model	160K RNs	1.6M (10× ←) RNs	16M (10× ←) RNs	160M (10× ←) RNs	M2 to M1
XORWOW	M1: Co-processor	2.858	17.477 (6.11×)	164.762 (9.42×)	1634.63 (9.92×)	3.15×
	M2: Dataflow	6.315	59.749 (9.46×)	593.921 (9.94×)	5934.01 (9.99×)	
SFMT	M1: Co-processor	4.831	30.494 (6.31×)	285.603 (9.36×)	2835.71 (9.92×)	1.86×
	M2: Dataflow	6.722	60.242 (8.96×)	594.355 (9.86×)	5934.5 (9.98×)	
XOROSHIRO	M1: Co-processor	7.637	70.596 (9.24×)	697.669 (9.88×)	6967.69 (9.98×)	0.96×
	M2: Dataflow	6.961	68.520 (9.84×)	683.628 (9.97×)	6833.75 (9.99×)	

- *Scaling is sub–linear to linear*
- *Overheads such as packet switching are well amortised (especially with XORWOW and SFMT at 1.6M)*
- *XORWOW > SFMT > XOROSHIRO in speed in M1*
- *XOROSHIRO ~ SFMT & XORWOW in speed in M2*

→ *XORWOW and SFMT are suitable for M1 while XOROSHIRO is suitable for M2*

Conclusion

- PRNGs and probability distribution approximations can be part of dataflow processing on modern CGRAs/RDAs.
- AMD AI Engines is an example of such adaptive dataflow architectures.
- PRNGs and probability distribution approximations do not natively map to this architecture.
- Building the lacking features can be done using the available operations.
- Performance can be sub–linear to linear using these manipulations.
- Future work will focus on applying similar transformations on arithmetic operations to overcome the limitations. of signed arithmetic with the Multiple–With–Carry PRNG family.

Thanks!

PRNGine: Massively Parallel Pseudo–Random Number Generation and Probability Distribution Approximations on AMD AI Engines

Github Repository

Reach out via email

Connect on LinkedIn