Topic 3: Energy-efficient implementation of probabilistic circuits

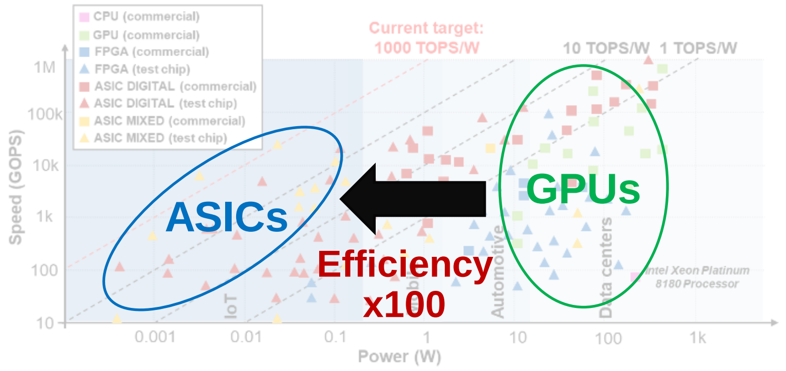
ELEC-L352002 - Postgraduate Course in Electronic Circuit Design II V D

Jelin Leslin (jelin.leslin@aalto.fi)

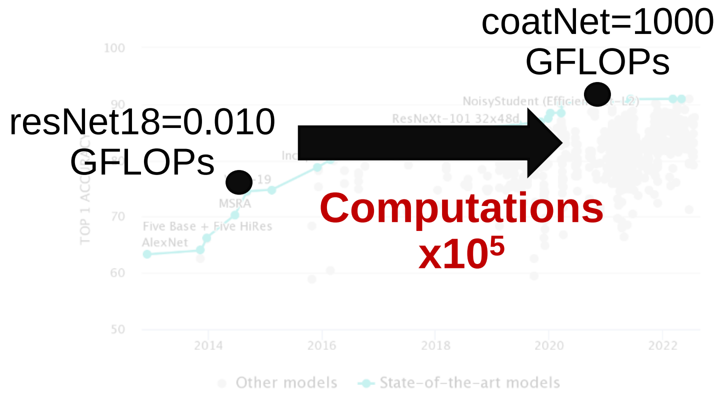
17/05/2023

Aalto-yliopisto
Aalto-universitetet
Aalto University

Motivation: efficient edge AI?



Hardware “accelerators” [1]



Top 1 accuracy on ImageNet [2]

- Although we develop more efficient accelerators, the size of NN models explode ➔ We need to explore **alternatives**
- We need models that are generative (use them for various applications), and more compact (to be embedded online)
- We look at probabilistic models and **probabilistic circuits (PCs)**

Aalto-yliopisto
Aalto-universitetet
Aalto University

[1] https://nicsefc.ee.tsinghua.edu.cn/projects/neural-network-accelerator

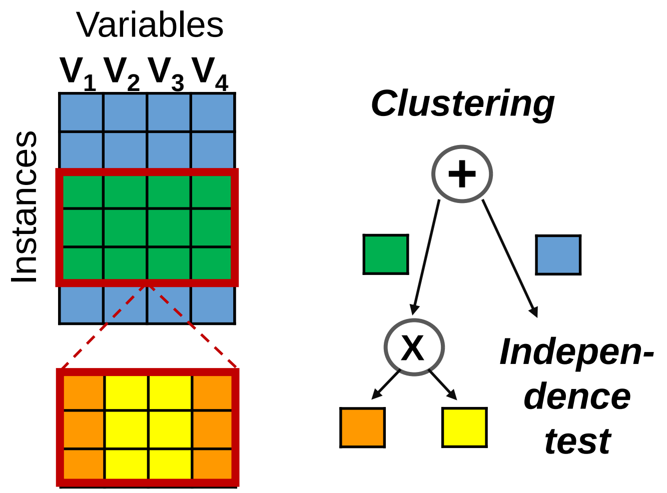
[2] https://paperswithcode.com/sota/image-classification-on-imagenet

Probabilistic circuits: training

Training involves both **structure** and **parameter** learning [3]

Structure learning:

- Active research topic
- Example with LearnSPN
    - Iterative clustering (+) and independence tests (x)



Parameter learning:

- Main algorithm is expectation-maximization [4]
- Idea: Find the best likelihood under missing data

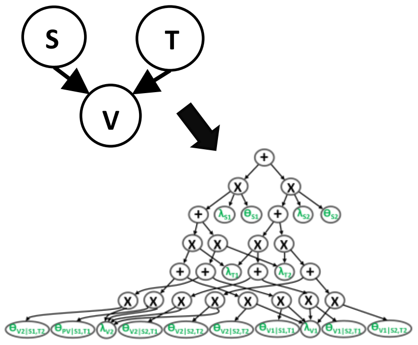
[3] Y. Choi, A. Vergari, and G. Van den Broeck, "Probabilistic circuits: A unifying framework for tractable probabilistic models," oct 2020.

A variety of PC implementations

Arithmetic Circuits (ACs)

ACE [5]

Compiled from Bayesian networks

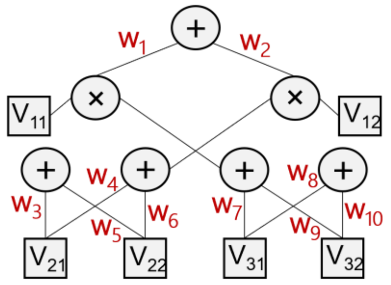


The **compilation** gives a fixed structure

Sum-Product Networks (SPNs)

LearnSPN [6], LibSPN [7]

Learned from data



Example: **recursive learning** through independence tests and clustering

Structure Decomposable PCs

SDPCs [8]

Impose extra structural constraints for more tractability

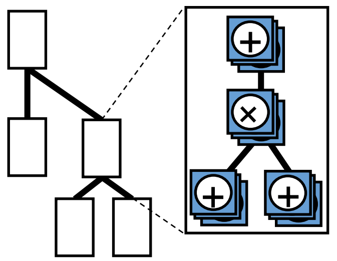
The PC is **normalized to a vtree** (binary tree)

The extra constraint **can modify the structure**

Vectorized PCs

RAT-SPNs [9], Einets [10]

Sum and products concatenated as vectors



Hyperparameters control the structure (vector size, depth..)

Aalto-yliopisto
Aalto-universitetet
Aalto University

[5] UCLA ACE compiler
[6] Poon & Domingos, 2011
[7] Pronobis *et al.*, 2007
[8] Dang *et al.*, 2020
[9] Peharz *et al.*, 2019
[10] Peharz *et al.*, 2020

Basic structure of a PC

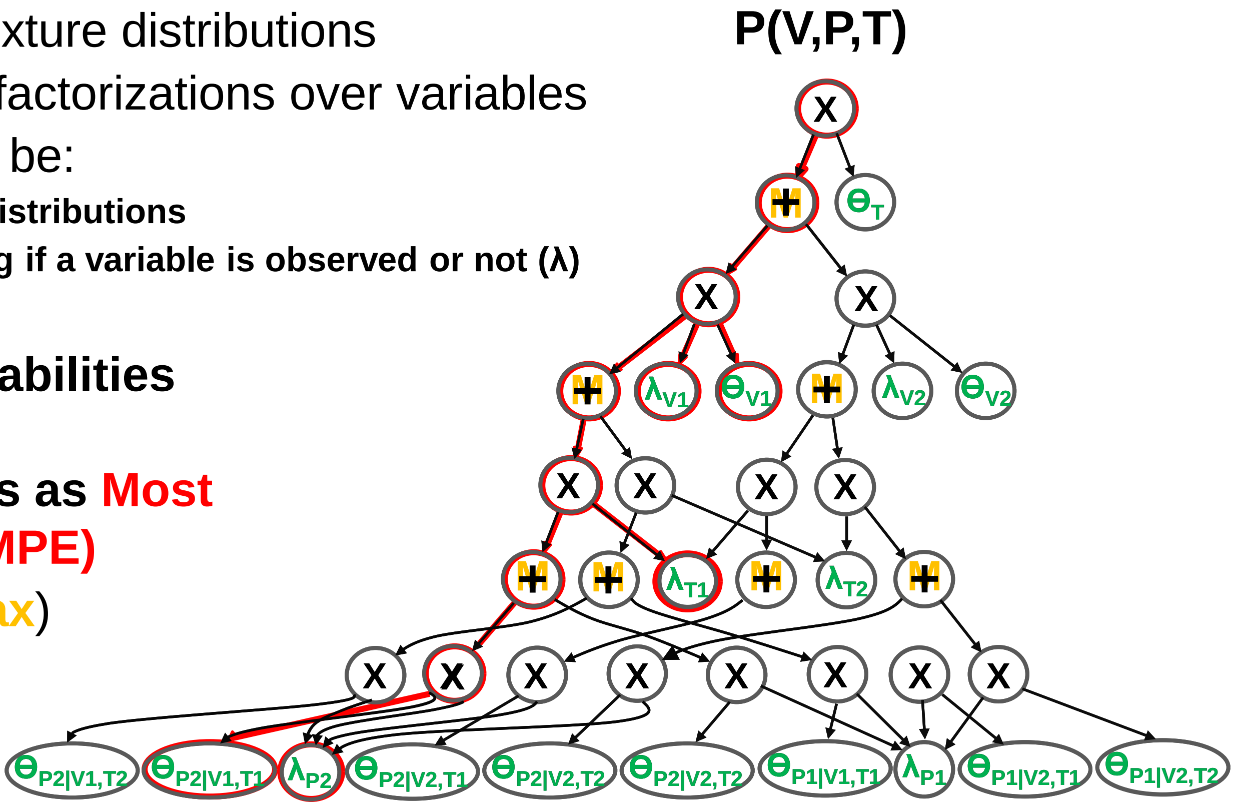
Composed (at least) of three types of nodes:

- Sums represents mixture distributions
- Products represent factorizations over variables
- Leaf nodes that can be:
  - **Univariate probability distributions**
  - **Binary indicators saying if a variable is observed or not (λ)**

Parameters (ϴ) are probabilities

We can compute queries as Most Probable Explanation (MPE)

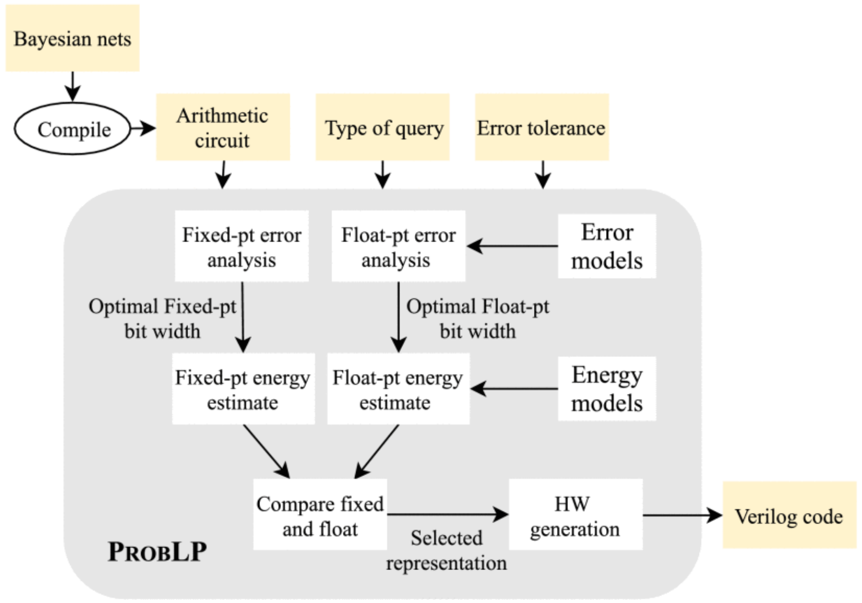
- (+) replaced by (max)
- Result in one pass



ProbLP [11]

Main Idea: A holistic framework to automate the design of low-precision custom hardware for ACs.

- Get a model , evaluate error bounds and determine optimal representations for a given requirements set.
- Increments the number of fraction bits for fixed-point representation until the error requirement is met.
- Increments the number of float bits for mantissa in floating-point representation until the error requirement is met.
- Estimates the minimum number of integer and exponent bits using min and max analysis.
- The analysis involves considering the extreme values and boundaries of the range, taking into account both the precision requirements and the constraints of the system or problem at hand.
- Selects between fixed-point and floating-point representations based on energy consumption, estimated using operator-level energy models in TSMC 65nm technology.

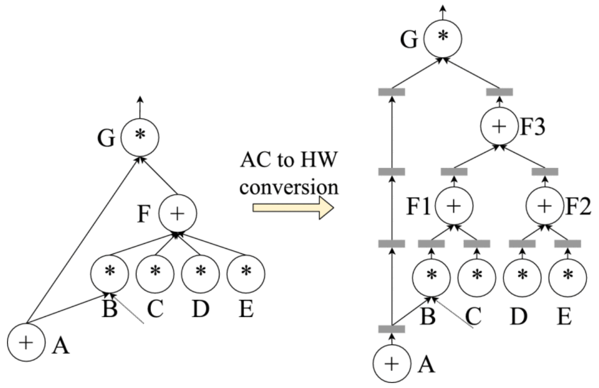


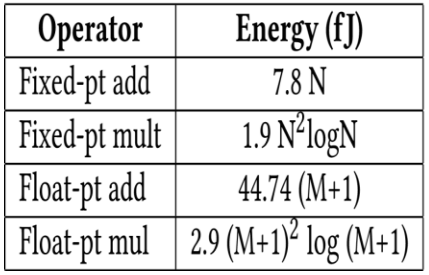
[11] shah *et al.,* 2019

ProbLP

Hardware generation after deciding a selected representation for the given model.

- The hardware generation process consists of two main stages: operator decomposition and pipeline register insertion.
- In the first stage, AC operators with more than two inputs are decomposed into a tree structure of 2-input operators. This decomposition helps in optimizing the hardware implementation.
- In the second stage, the generator inserts pipeline registers after every operator. This step ensures proper timing and can involve inserting multiple registers when there is a timing mismatch in a specific path.
- The analytically simulated results are verified with the generated RTL using modelsim.
- The default error tolerance is set at 0.01, and increasing energy efficiency can be accomplished by allowing for more relaxed error tolerances.





Operator	Energy (fJ)
Fixed-pt add	7.8 N
Fixed-pt mult	$1.9\ N^2 \log N$
Float-pt add	44.74 (M+1)
Float-pt mul	$2.9\ (M+1)^2 \log (M+1)$

Aalto-yliopisto
Aalto-universitetet
Aalto University

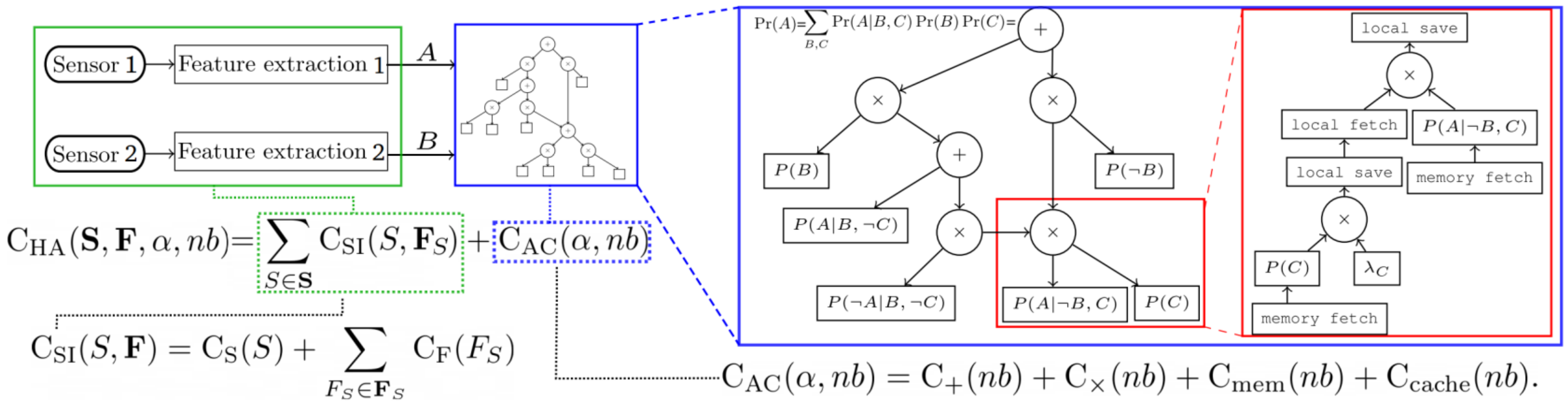
HW Aware PC [12]

Main Idea: To propose a resource-aware cost metric that considers hardware constraints and specifications to determine efficient deployment of probabilistic models in edge computing and enabling optimal device settings for meeting user requirements.

• Framework utilizes tractable probabilistic models, which enable efficient inference and possess desirable traits such as robustness to missing data, joint prediction capabilities, explainability, and minimal data requirements.

• Four key system properties determine the hardware-aware cost versus task performance trade-off in the system: inference model complexity (α), the type and number of sensors and features (F, S), and the number of bits used for computations (nb).

• The search process occurs in multiple consecutive stages based on the specific task (classification or density estimation), the type of input to the model learning strategy (data or probabilistic model), and the hardware capabilities (support for low precision arithmetic, for example).



$$C_{HA}(\mathbf{S},\mathbf{F},\alpha,nb)=\sum_{S\in\mathbf{S}}C_{SI}(S,\mathbf{F}_S)+C_{AC}(\alpha,nb)$$

$$C_{SI}(S,\mathbf{F}) = C_S(S) + \sum_{F_S\in\mathbf{F}_S} C_F(F_S)$$

$$C_{AC}(\alpha,nb) = C_+(nb) + C_\times(nb) + C_{mem}(nb) + C_{cache}(nb).$$

[12] Olascoaga *et al.*, 2020

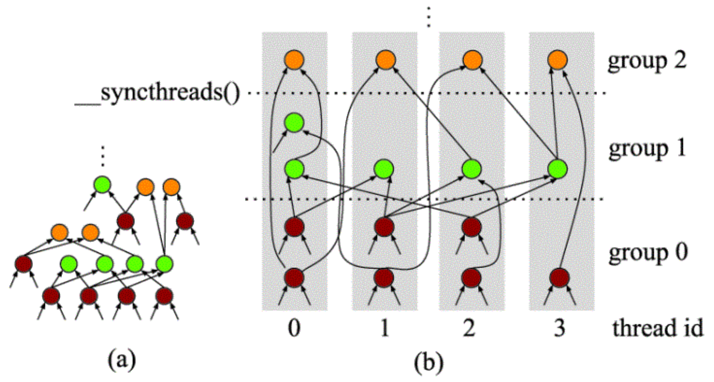
GPU becomes a bad choice !

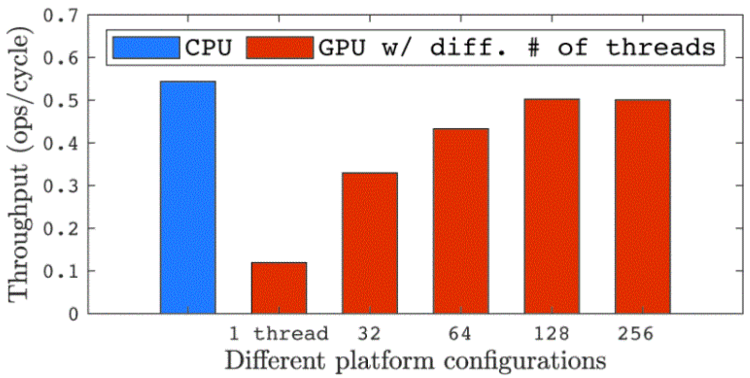
A GPU (Graphics Processing Unit) consists of multiple processing units called threads, which work together in groups called warps. These warps can access shared memory, a space all threads within a GPU can use.

If multiple threads within a warp want to access the same part of the shared memory simultaneously, it can create a problem known as a bank conflict and this slows down the overall performance of the GPU. A solution is to employ coloring-based bank allocation algorithm.

Use of 256 threads, however, only increases the throughput by a factor 4.1x, a sublinear scaling due to the following reasons:

- Overhead of thread synchronization.
- Secondly, the shared memory in the Jetson TX2 GPU, which has 32 banks distributed among 128 CUDA cores, has limited bandwidth. Since all the threads need to read from and write to the shared memory, its bandwidth becomes a bottleneck. This limitation in bandwidth can restrict the speed at which data can be transferred, thereby reducing the potential performance gains.
- Thread divergence due to selection between sum and product operations leads to inactive threads in a warp.





Aalto-yliopisto
Aalto-universitetet
Aalto University

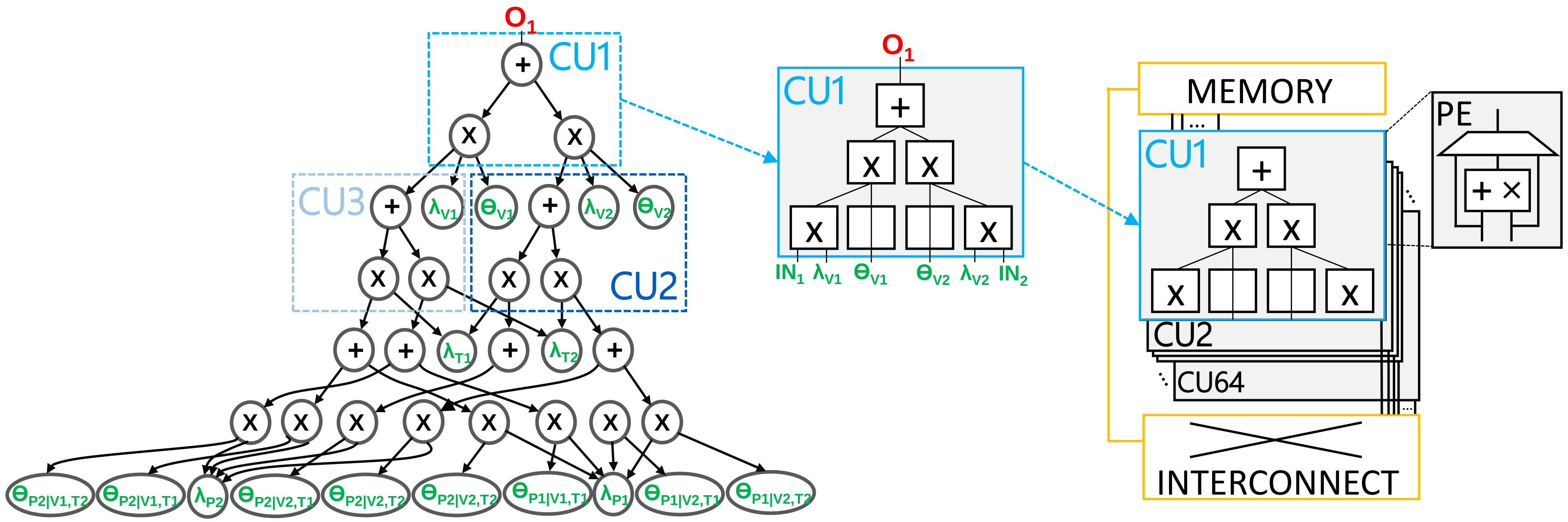
Accelerate PCs?

Specificities of PCs

- **Irregular graph** = less parallelism
- High **computation resolution** (20-50 bits in float)= more energy

Possible accelerator:

- **Trees** of processing elements (PEs)[13]
- An **optimizer** decomposes the graph across computation units (**CUs**) [14]



[13] Shah *et al.*, *DATE* 2020

[14] Shah *et al.*,GraphOPT,2021

Probabilistic Inference Unit (PIU) [15]

The PIU solution :

a) Stream-based compute with a co-optimized memory hierarchy:

•It utilizes a stream-based computation approach, where data flows through the processor in a streaming manner, improving efficiency.

•This stream-based approach allows for aggressive data prefetching, reducing the impact of memory latency.

•The PIU's memory hierarchy, including scratchpads and register banks, is co-optimized to store and access data during computations efficiently/parallely.

•By carefully managing the flow of data through the memory hierarchy, the PIU minimizes stalls and maximizes the utilization of compute resources.

b) Precision-scalable posit arithmetic:

•The PIU employs a posit arithmetic system, specifically the unum III representation, a recently proposed format.

•Posit arithmetic allows for dynamic adjustment of the length of the regime and fraction fields based on the value being encoded.

•This dynamic adjustment enables a trade-off between fraction bits (accuracy) and regime bits (range) during runtime.

•The PIU further customizes the standard posit representation by using more exponent bits, prioritizing increased range over accuracy around the value 1.

•This customization is particularly beneficial for tasks like probabilistic inference, where encountered probabilities can be very small.

•The precision-scalable posit unit in the PIU performs operations in different precisions (e.g., 1×32b, 2×16b, or 4×8b), enabling batch inference for lower precisions without sacrificing accuracy.

Probabilistic Inference Unit (PIU)

c) Aligned compiler optimizations to accelerate exact inference workloads:

•The PIU incorporates compiler optimizations specifically designed to accelerate exact inference workloads in probabilistic models.

•The compiler analyzes the structure of the Sum-Product Networks (SPNs) and decomposes them into layers of disjoint subgraphs.

•By decomposing the SPNs in this way, the compiler ensures no edges between subgraphs within a layer, allowing for parallel execution on different compute units (CUs).

•Synchronization points, including special global barrier instructions, are inserted by the compiler to enable data sharing between CUs after each layer of subgraphs.

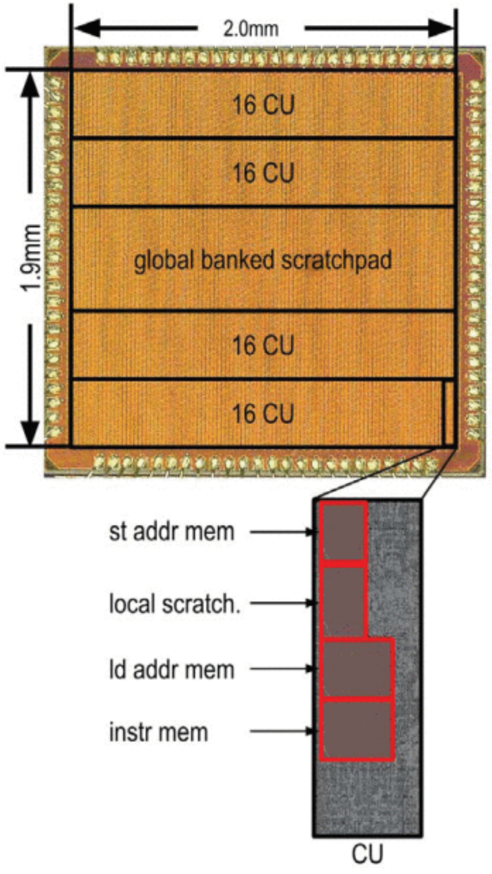
•Load-compute and compute-store dependencies are handled using FIFO-based data transfers, while load-after-store dependencies are managed through compiler-inserted local barrier instructions.

•The compiler optimizations enable efficient data flow, minimize dependencies, and exploit the parallelism of the SPN structure, resulting in accelerated exact inference workloads.

Peak energy-efficiency: The PIU demonstrates a peak energy-efficiency of 248GOPS/W (Giga Operations Per Second per Watt) at a voltage of 0.6V, frequency of 113MHz, and precision of 8b. This showcases the energy-efficient nature of the PIU, allowing for efficient execution of probabilistic computations while minimizing power consumption.

Outperforms simulated ASIC, FPGA, desktop and embedded GPUs, and CPUs in all familiar PC benchmarks.

Probabilistic Inference Unit (PIU)



Technology	28nm		
Active area	2.0 x 1.9 mm^2		
Supply Voltage	0.9 V (min. 0.6V)		
Precision	8/16/32b custom posit		
On-chip SRAM	864 kB		
	8b	**16b**	**32b**
Fmax (MHz)	288	281	278
Power (mW) @Fmax	228	227	228
Peak Through. (GOPS)*	33.7	16.3	8.1
Peak Through. per area (GOPS/mm^2)*	8.9	4.3	2.1
Peak Energy eff. (GOPS/W)*	248 @0.6V	121 @0.6V	57.8 @0.6V

*averaged over UCLA model zoo

Aalto-yliopisto
Aalto-universitetet
Aalto University

Assignment questions

1. Briefly explain how an arithmetic circuit/ Probabilistic circuit is converted to a pipelined hardware circuit – (ref : section 3.4 in Problp [11]).
2. What are the three solutions/ideas discussed in the Probabilistic Inference Unit (PIU).

References

1. https://nicsefc.ee.tsinghua.edu.cn/projects/neural-network-accelerator.
2. https://paperswithcode.com/sota/image-classification-on-imagenet.
3. Y. Choi, A. Vergari, and G. Van den Broeck, “Probabilistic circuits: A unifying framework for tractable probabilistic models,” oct 2020.
4. Moon, Todd K. "The expectation-maximization algorithm." *IEEE Signal processing magazine* 13.6 (1996): 47-60.
5. UCLA –ACE compiler : http://reasoning.cs.ucla.edu/ace/
6. Poon, Hoifung, and Pedro Domingos. "Sum-product networks: A new deep architecture." *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*. IEEE, 2011.
7. Libspn framework : https://www.libspn.org/
8. Dang, Meihua, Antonio Vergari, and Guy Broeck. "Strudel: Learning structured-decomposable probabilistic circuits." *International Conference on Probabilistic Graphical Models*. PMLR, 2020.
9. Peharz, Robert, et al. "Random sum-product networks: A simple and effective approach to probabilistic deep learning." *Uncertainty in Artificial Intelligence*. PMLR, 2020.
10. Peharz, Robert, et al. "Einsum networks: Fast and scalable learning of tractable probabilistic circuits." *International Conference on Machine Learning*. PMLR, 2020.
11. Shah, Nimish, et al. "Problp: A framework for low-precision probabilistic inference." *Proceedings of the 56th Annual Design Automation Conference 2019*. 2019.
12. Galindez Olascoaga, Laura Isabel, Wannes Meert, and Marian Verhelst. "Hardware-Aware Probabilistic Circuits." *Hardware-Aware Probabilistic Machine Learning Models: Learning, Inference and Use Cases*. Cham: Springer International Publishing, 2021. 81-110.
13. Shah, Nimish, et al. "Acceleration of probabilistic reasoning through custom processor architecture." *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020.
14. Shah, Nimish, Wannes Meert, and Marian Verhelst. "GRAPHOPT: constrained-optimization-based parallelization of irregular graphs." *IEEE Transactions on Parallel and Distributed Systems* 33.12 (2022): 3321-3332.
15. Shah, Nimish, et al. "9.4 piu: A 248gops/w stream-based processor for irregular probabilistic inference networks using precision-scalable posit arithmetic in 28nm." *2021 IEEE International Solid-State Circuits Conference (ISSCC)*. Vol. 64. IEEE, 2021.

Thank you for your attention.

Aalto-yliopisto
Aalto-universitetet
Aalto University

PC accelerators – Benchmarks

CPU,GPU

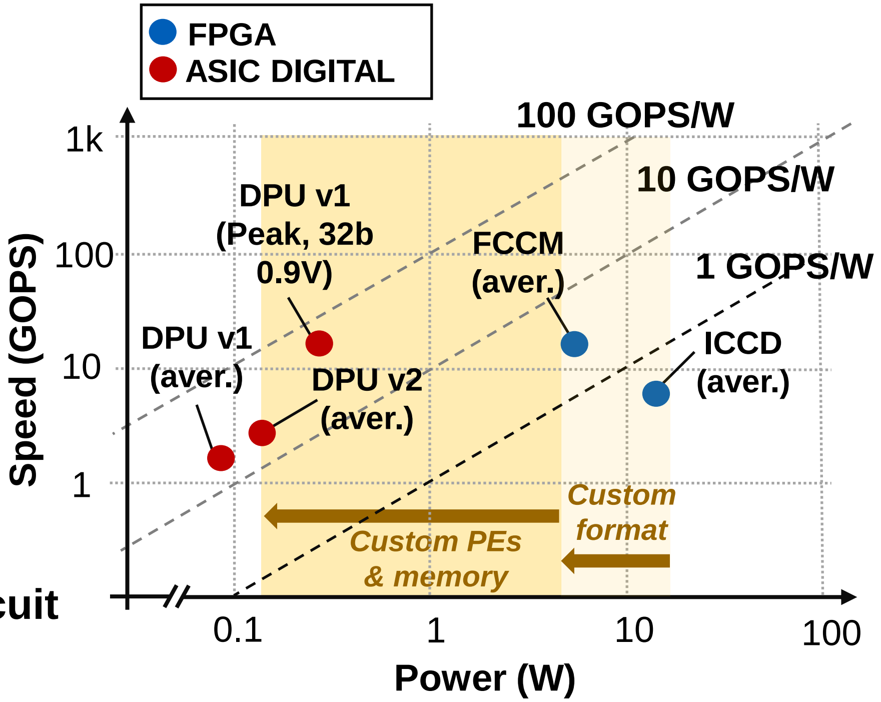
- SPNC [13]: customized compilation flow based on MLIR (CPU and GPU) ➔ up to 800x acceleration compared to regular CPU

FPGA

- **Sommer ICCD [14]:** automatic HW generation with pipeline float operators (double)
- **Sommer FCCM [15]:** Adding customized arithmetic formats (Float, Posit, Log) ➔ 10x Eff.
- **Kruppe [16]:** Adding more advanced scheduling for processing the PC
- **Choi [17]:** processor architecture with PEs to handle larger PCs (1-3 GOPS/query with HLS)

Custom Application-Specific Integrated Circuit

- DPU [18]: tree-shaped PEs, custom memory and scheduler ➔ 10x Eff. compared to best FPGA
- DPUv2 [19]: Advanced scheduler for processing



Aalto-yliopisto
Aalto-universitetet
Aalto University