Exploiting High-Bandwidth Memory for FPGA-Acceleration of Inference on Sum-Product Networks

Lukas Weber*, Johannes Wirth*, Lukas Sommer*, Andreas Koch*
*Embedded Systems and Applications Group, TU Darmstadt, Germany
*[surname]@esa.tu-darmstadt.de

***Abstract*—Due to the memory wall becoming increasingly problematic in high-performance computing, there is a steady push to improve memory architectures, mainly focusing on better bandwidth as well as latency. One of the results of this push is the development of High-Bandwidth Memory (HBM) which is an alternative to the regular DRAM typically used by accelerator-cards.**

This work adapts an existing accelerator architecture for inference on Sum-Product Networks (SPN) to exploit the HBM present on more recent high-performance FPGA-accelerator cards. The evaluation shows that the use of HBM enables almost linear scaling of the performance due to the embarrassingly parallel nature of batch-wise SPN inference. It is also shown that the only hindrance to this scaling is the limited bandwidth available for data-transfers between host and FPGA. Even with this bottleneck, the prior FPGA-based implementation is outperformed by up to 1.50x (geo.-mean 1.29x). Similarly, the CPU and GPU baselines are outperformed by up to 2.4x (geo.-mean 1.6x) and 8.4x (geo.-mean 6.9x) respectively.

Based on the evaluation, the scaling potential of HBM-based FPGA-accelerators is explored to give an outlook on what is to come with future generations of PCIe-based interfaces.

***Index Terms*—Sum-Product Network, Probabilistic Models, Machine Learning, High-Bandwidth Memory, FPGA**

I. Introduction

Artificial intelligence and machine learning (ML) have become pervasive in our every-day life, being deployed in applications such as voice-based smart assistants or in medical applications. Most of the progress made in recent years has not only been enabled by improvements to the ML models themselves, but also by the constant improvement of the execution hardware, which needs to provide sufficient computational power to train models with multiple billions of parameters, and compute inference quickly enough for real-time applications.

Much work on the acceleration of machine learning models has focused on (deep) neural networks (NN). Next to GPUs and dedicated ASIC-accelerators built for the single purpose of accelerating machine learning training and inference, such as Google's TPU or Graphcore's IPU, FPGAs have proven to be a compelling platform for deep neural network (DNN) acceleration [1].

However, despite their broad adoption, deep neural networks can still suffer from serious limitations in real-world usage scenarios. This has sparked an increased interest in *probabilistic* models, which are much better able to cope with real-world uncertainties. While inference for many probabilistic models is intractable in the general case, so-called *Sum-Product Networks* (SPN) [2] combine the strengths of probabilistic models with tractable inference for real-world applications. These properties make SPNs not only an interesting candidate model for ML applications, but also an attractive application for acceleration on different target-platforms, including GPUs [3]–[5], custom ASIC processors [3] and also FPGAs [4], [6], [7].

In our prior work, we developed an architecture for high-throughput inference in Sum-Product Networks, based on FPGAs available in Amazon's AWS cloud [8]. As a single instance of the pipelined accelerator for SPN inference did not fully exploit the available FPGA resources, we developed a multi-core architecture, with multiple identical accelerators conducting inference in parallel. However, even though we employed up to four parallel memory banks, the memory accesses during the computations quickly became a bottleneck, in particular due to the relatively low arithmetic intensity of SPN inference.

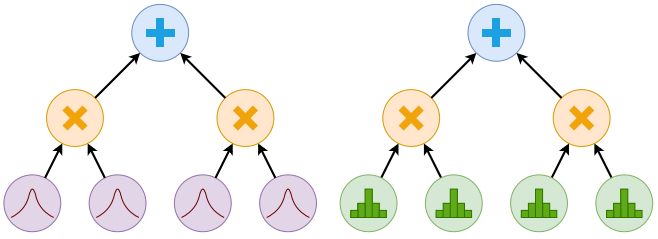
A promising alternative to overcome this limitation is the use of *High-Bandwidth Memory* (HBM), which FPGA vendors are now increasingly integrating into their products. Being composed of dozens of small, independent blocks of memory, HBM allows multiple memory accesses to be performed in parallel and thus increases the available memory bandwidth.

This work presents our contribution, which is an adapted accelerator architecture that exploits the highly parallel interface of HBM on FPGAs for high-throughput inference for Sum-Product Networks. The pipelined multi-core accelerator architecture is automatically generated from an SPN description, and is automatically integrated in a heterogeneous system. In addition, the hardware accelerator is combined with an efficient, multi-threaded software runtime interface on the host, to ensure a high-throughput supply of input data for the FPGA accelerator. Our contribution here includes the adapted overarching accelerator architecture, as well as the improved software runtime interface.

II. Background

A. Sum-Product Networks

In recent years, research interest in machine learning and artificial intelligence has been very high. Especially DNNs have been researched and improved to a great extent. A different, less explored model are Sum-Product Networks (SPN) [2]. Stemming from the class of probabilistic graphical models, they are able to capture joint probability distributions over many different random variables. Their two major



(a) Sum-Product Network (b) Mixed Sum-Product Network

Fig. 1. Different types of SPNs. (a) shows a typical SPN with Gaussian leaf nodes. (b) shows a Mixed SPN using histograms to approximate the gaussian distributions of (a).

advantages are their *tractability* and their ability to handle *uncertainty*. With regard to tractability, inference on SPNs can be performed in linear time w.r.t. the size of an SPN. With regards to uncertainty, SPNs are able to handle uncertainties like missing features or unclear classifications, due to the fact that they compute actual probability values. A very interesting example for this is discussed in the work by Peharz et al. [9] which uses randomly generated SPNs for classification tasks. Confronting an SPN trained for the image classification benchmark MNIST with out-of-domain images yields lower probabilities and thereby indicates that the SPN is *uncertain* about the resulting classification.

In general, SPNs are directed acyclical graphs, comprising three distinct node types: 1) The leaf nodes represent univariate probability distributions over single random variables. 2) Product nodes represent factorizations of independent variables. 3) Sum nodes represent mixtures of distributions.

Using these three node types, SPNs are capable of capturing complex joint probability distributions. To achieve this, the general process is as follows: For each dataset, an independence check is performed to determine if there are any independent variables. If so, this is represented in the SPN using a product node. If this is not the case, the dataset is divided by clustering. The resulting sub-datasets are then recursively traversed, until the data can be represented using a single univariate distribution. Due to this very simple structure, SPNs are a very simple and concise way of representing complex joint probability distributions. To perform inference on an SPN, a simple bottom-up evaluation has to be performed.

In this work, we rely on a specific flavor of SPNs called Mixed SPNs [10]. The main difference between pure SPNs and mixed ones is the fact that the leaf nodes are approximated using simple histograms (c.f. Fig. 1). These histograms can easily be mapped to hardware as shown in [4], [6], [11]. Specifically, we will build upon our prior work [8], which explored the application of SPNs in a heterogeneous reconfigurable cloud (Amazon AWS F1 Instances with FPGA accelerators).

B. High-Bandwidth Memory

In their current UltraScale+ series, Xilinx offers some FPGAs which include High-Bandwidth-Memory (HBM) in addition to conventional off-chip SDRAM. As the name implies, this new type of memory provides significantly more memory bandwidth compared to off-chip SDRAM: According to Xilinx the HBM used on their FPGAs can achieve up to 460GB/s. However, this number can only be achieved when issuing multiple memory requests in parallel, making it necessary to adapt existing architectures in order to actually exploit the additional bandwidth.

The HBM on these Xilinx FPGAs has a capacity ranging from 4GB to 16GB and is split into two stacks. Each stack features 16 **memory channels** with a width of 256 bit, each connected to its own memory region. By default, each of these channels can only access its associated memory region. Each of these memory channels is exposed to the user logic via one AXI3 interface, resulting in a total of 32 AXI3 interfaces for the HBM.

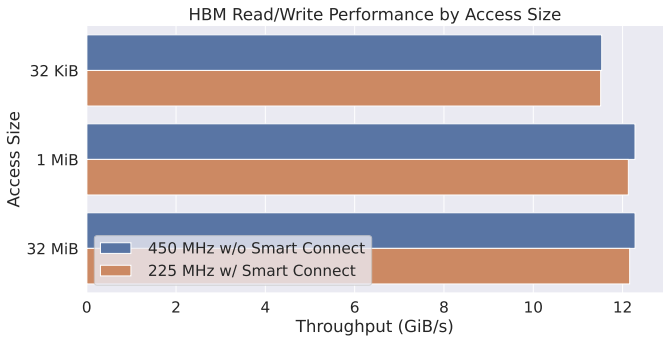


Fig. 2. Maximum throughput when issuing linear read and write accesses in parallel to one HBM memory channel for two different configurations and different request sizes. The first configuration runs the block generating the accesses with the 450 MHz clock used by the HBM and natively connects both. The second configuration runs the PE at *half* the clock frequency but the interface width is *doubled*. An AXI Smart Connect is used to perform clock- and data-width-conversion.

Xilinx offers an optional crossbar which, when enabled, hides the partitioning from the user and allows to access the *entire* memory space from each AXI interface. However, this comes at the cost of additional latency and decreased performance, where the actual impact is highly dependent on the concrete access pattern. For the rest of this work, we will *not* use the crossbar, since we aim to explore the maximally achievable performance.

Figure 2 shows the performance for *one* HBM memory channel. The performance data is generated using a special benchmark hardware block which generates linear memory reads and writes in parallel, as this is the access pattern used by our SPN accelerators. There are two major insights: First, the throughput caps at a request size of 1 MiB, as no further performance improvements are observed beyond this. And second, there is no significant performance benefit when running the benchmark block at 450 MHz with a connection to the HBM at its native interface width versus running the block at *half* the clock frequency and in turn *doubling* the interface width. This is a valuable insight, as it is often not possible to run user logic at 450 MHz. Because we do not use the crossbar, the different HBM memory channels are

completely independent and performance scales *linearly* w.r.t. to the number of channels/accelerators used.

III. Approach

In this section, we will introduce our approach for scaling up the number of SPN-accelerators using HBM. Additionally, we will discuss the motivation and reasoning for the upscaling.

A. Motivation

Considering the theoretical advantages of SPNs over other machine learning models, they have an obvious place in many real-world applications. The fast inference that can be achieved using FPGA-accelerated SPNs is an additional advantage. Since FPGAs are not as wide spread as GPUs, using the reconfigurable cloud is also a reasonable choice (as described in [8]). Unfortunately, looking at the architecture used by [8], there is an obvious problem: Due to the size of the SPN accelerators, as well as their memory-bandwidth requirements, it becomes increasingly hard to map them to Amazon AWS F1 instances. Looking at the NIPS80 benchmarks from [8], we can see that combining bigger accelerators with multiple memory controllers leads to a trade-off: Either we sacrifice memory controllers, which limits the overall throughput of the system by reducing the amount of data that can be accessed in parallel. Or, we reduce the number of accelerators, which means that fewer inferences can be handled concurrently. Specifically, the logic resources on the F1 are insufficient to hold the combination of four NIPS80 accelerators with four separate memory controllers. Thus, only two accelerators were used, which in turn slowed performance for that benchmark. Alternatively, it was possible to use a single memory controller in combination with three SPN accelerators, which also had a performance cost.

If we take into account the advantages of HBM memory (described in Section II-B), it seems very reasonable to replace the use of on-board DRAM with use of on-chip HBM. The HBM controllers are implemented as hard IP and thus do not consume FPGA resources. This, in turn, should allow the use of more accelerator-cores. Since soft memory controllers are also sensitive to clocking constraints, their removal should also improve the problem of globally deteriorating clock frequencies encountered in [8]. Specifically, the use of additional soft memory controllers had a larger impact on the achievable clock frequency than the addition of extra SPN accelerators. Last but not least, the independent HBM blocks can be exclusively assigned to individual SPN accelerators, avoiding interference between them. This should also be another advantage over the shared use of on-board DRAM.

B. SPN-Accelerator

Fundamentally, the basic SPN accelerator-cores have a simple architecture. The accelerator is connected to a memory via a AXI4 Full interface, which typically enables access to the on-board DRAM. The same interface can be used to access on-chip BRAM or HBM without requiring big changes to the accelerator. To ensure compatibility to all kinds of AXI-based

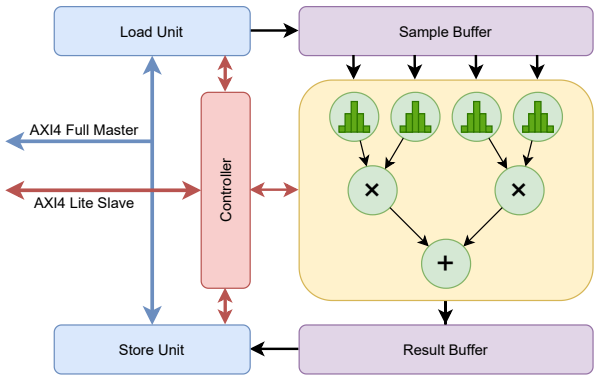


Fig. 3. Architecture of the SPN-Accelerator

memories, we made our interface generation more generic to also cover HBM memories. Accelerators are controlled by an AXI4 Lite Interface, which exposes a simple register file to the user. Due to the increased address-width of the HBM-data-channel, we had to adapt the control registers to 64 bit. Within the accelerator (also depicted in Fig. 3), there are multiple submodules which orchestrate the batch-wise inference: First, the **Load Unit** loads the data from the memory and pushes it into the **Sample Buffer**. This buffer collects incoming data until a complete vector of input values has been built. Then, the vector is pushed into the **SPN Datapath**. The result values of the SPN Datapath are collected in a **Result Buffer**. The result buffer collects 64 bit result values, until a 512 bit word is complete. This word is then pushed into the **Store Unit**, which will handle the AXI4 Write to store the results back to memory.

The most important part of the accelerator is the **SPN Datapath**, which can be generated automatically from a textual description of the SPN. The textual description is compatible with the SPFlow library [12], which enables a very simple and streamlined development toolflow. SPNs can be easily trained and evaluated using SPFlow, afterwards exporting them to the textual description for hardware-generation. In addition to the SPFlow-compatibility, the generator offers great flexibility with regards to the used internal number format. In prior work, the different number formats were discussed in more detail [4], but in general, the generator supports a Custom Floating Point (CFP) format as well as a Logarithmic Number System (LNS) format. Both formats can be configured at a very fine granularity. For CFP, the number of exponent- and mantissa-bits can be configured, as well as the used rounding scheme. For both formats, the generated digital arithmetic is optimized towards the use on FPGAs. The optimizations of LNS are further discussed in [11], while the CFP format is described in detail in [4]. For this work, we chose the suitable configurations determined in [4].

IV. Implementation

To enable the use of HBM, we have made two distinct changes to our prior work: First, we adapted the on-chip architecture to use HBM in a manner that enables the use

of many parallel SPN accelerators. In addition, we made some improvements to the software-interface to ensure that the parallelism provided by the many HBM channels is actually exploited.

A. On-Device Architecture

We use the open-source framework TaPaSCo [13] as a basis for our architecture. However, several modifications had to be made to accommodate the requirements for our use-case.

The biggest change is, of course, the use of HBM instead of off-chip DDR-SDRAM memory. We use a dedicated HBM block (and thus memory channel) per SPN accelerator. However, it is not possible to run the SPN accelerators at the 450 MHz of clock frequency used by the HBM. In order to achieve the same memory throughput, we run the accelerators at the more easily achievable half frequency (225 MHz), but double the interface width to 512 bit. As discussed in Section II-B, this indeed does not affect memory performance. We use an AXI SmartConnect between the accelerator and the HBM, which is responsible for data-width- and clock-conversion. It also performs protocol conversion, as the accelerators use AXI4, while the HBM only supports AXI3. Additionally, we employ register slices on these AXI connections where necessary, to achieve routability. This setup ensures that there are no unnecessary dependencies between the accelerators which might impact performance.

B. Parallel Runtime

To allow users to easily interface with the SPN inference accelerators on the FPGA, we have developed a software runtime, based on the TaPaSCo API. In contrast to prior work, where important parameters and information had to be supplied manually by the user, the new software runtime can now query the TaPaSCo system and the accelerator itself for these parameters, making it easier to interact with the accelerator. To this end, the accelerator was extended with a second execution mode to read out the configuration parameters specified at synthesis time.

In addition to providing an easy-to-use interface to the user, the second important task of the runtime is to orchestrate the execution of the accelerator instances on the FPGA.

As described in Section IV-A, accelerator instances on the FPGA are directly coupled to a dedicated HBM memory channel per instance, i.e., each accelerator instance only has access to a single HBM memory block. However, TaPaSCo currently does not support to split the device address space into distinct memory regions, so we cannot rely on TaPaSCo's memory management API to allocate and manage the HBM address space. Instead, our SPN runtime implements its own thread-safe device memory manager, which allows to manage the distinct HBM memory blocks separately. The device memory manager in our runtime supports allocation and freeing of memory blocks in a specific HBM block, making it possible to establish distinct address regions for each HBM block.

Prior work [8] also showed that overlapping the data-transfers between host and device with the execution on the accelerator can reduce overall execution time. To implement such a scheme, each compute job is broken down into multiple sub-jobs, according to an user-specified block-size. Each CPU thread then performs the same sequence of tasks: First, the data is transferred to the on-chip HBM. Then the SPN-accelerator is invoked and the CPU-thread waits for it to finish. As soon as the accelerator finishes the inference task, the CPU thread triggers the transfer of the results from HBM to the host.

By assigning multiple CPU threads to one accelerator instance on the FPGA, we can effectively overlap data transfers and computations, as one thread will be able to perform data transfers for block $n+1$, while another thread is waiting for the FPGA accelerator instance to complete computation of block n.

In the prior work, up to four threads per SPN accelerator were used to achieve maximum throughput. In our current implementation, measurements have shown that the DMA over PCIe bandwidth is already fully utilized with just two threads per SPN accelerator.

V. Evaluation

To evaluate how the use of HBM impacts SPN inference, we will first take a look at the hardware utilization and (potential) scaling capabilities of our approach. Afterwards we compare the results against our prior work focusing on AWS F1 [8], which mainly assesses the performance of SPN inference in a cloud computing setting. For all benchmarks we rely on datasets from the well-known NIPS corpus[1].

A. Resource Utilization

Compared to our prior work, there are three significant changes which impact the resource utilization: 1) Due to the exclusive use of HBM, it is not necessary to include soft DRAM controllers in our design. The HBM controllers are hardened IP, which means they do not require logic resources. Conversely, using a soft DRAM controller requires a significant amount of FPGA-resources. 2) While we are using the same SPNs as [8], we exploit additional prior work, which made the internal arithmetic format more flexible w.r.t. to the bitwidth, and also optimized the arithmetic for the SPNs [4], [11]. 3) Our evaluation was performed using a Bittware XUP-VVH accelerator card, which features a Xilinx UltraScale+ VU37P FPGA. In comparison, Amazon AWS F1 instances feature a similar FPGA, which does not have HBM capabilities. Both FPGAs are from the UltraScale+ series, but the AWS FPGA has slightly fewer logic resources. Additionally, all designs targeting the F1 instances have to include a shell for the host interface, which also incurs a resource overhead.

In addition to these differences, the results provided by [8] use varying numbers of SPN accelerators and memory controllers. For a valid comparison, we initially limit the scope to benchmarks with four accelerator-instances with a corresponding memory channel each (i.e., up to and including

[1] http://archive.ics.uci.edu/ml/datasets/bag+of+words

NIPS40). To contrast these with our new HBM-capable architecture, we built corresponding designs that feature four accelerator-instances, each connected to a dedicated HBM channel. The results are shown in Table I.

It is obvious that our new approach is more resource efficient in almost all resource types. Interestingly, the accelerators used in [8] generally require fewer LUTs used as Memory. The change to the custom floating point arithmetic here (originally developed in [4]) could explain this change. In terms of LUTs used as Logic, BRAM and DSPs, our work typically requires approx. 66% fewer resources. LUTs used as memory are slightly increased (except for NIPS40). The number of registers used here is roughly 50% less than in [8].

Overall, the resource requirements have vastly decreased in comparison to [8]. This opens up the potential to further replicate the accelerator to scale up. This allows us, e.g., to fit up to eight NIPS80 accelerators on the FPGA compared to only two in [8].

B. Performance Scaling

To describe the scaling of our architecture, we take a closer look at the very small NIPS10 benchmark. For each processed sample, the input consists of 10 single-byte values. The result is a single double-precision value. This means that each processed sample entails a total data transfer of 144 bits. Using a single SPN accelerator, the architecture is able to process 133,139,305 samples per second. Multiplying by the number of input and result bits per sample reveals that the accelerator requires 2.23 GiB/s of memory bandwidth. Given the HBM performance discussed earlier (c.f. Fig. 2), this shows that a single HBM channel should easily be able to provide the data required for a single accelerator. Hypothetically, linear scaling should be possible to at least 32 accelerators, due to the 32 HBM channels (and completely disregarding the limited logic resources).

To test this hypothesis, we ran multiple performance benchmarks for each of the benchmark SPNs and measured the end-to-end execution time required for computing inference over 100,000,000 samples using up to eight concurrent SPN cores, each controlled by up to two control-threads. From the results, we conclude that using more than one control-thread only improves performance for less than four accelerators. Thus, *all of the results* presented in this section are measured with only one dedicated control-thread per SPN accelerator. The corresponding benchmark results are visualized in Fig. 4.

If we look at the right side of Fig. 4, the scaling for NIPS10 is obviously slowing at five or more SPN accelerators. Adding additional accelerators after that point does not yield any significant performance improvements. Using five accelerators, we are able to process 614,654,595 samples per second, which in turn requires approx. 10.3 GiB/s of memory bandwidth. Due to the use of up to eight independent HBM channels with approx. 12 GiB/s each, the available memory bandwidth should not be an issue.

To further investigate this point we performed a separate set of benchmark runs. In the second run, we disregarded the host-to-device data-transfer times and only measured the on-device computation including the HBM accesses. The results are shown on the left in Fig. 4. It is clear that almost linear scaling is achieved for up to eight concurrent accelerators. This makes sense, since the batch-wise inference on SPNs is *embarrassingly parallel*. This trend is also likely to continue at least until all 32 HBM channels are used by at least one SPN accelerator. Unfortunately, scaling up that far is not possible due to the limited FPGA logic resources, as well as routing scarcity.

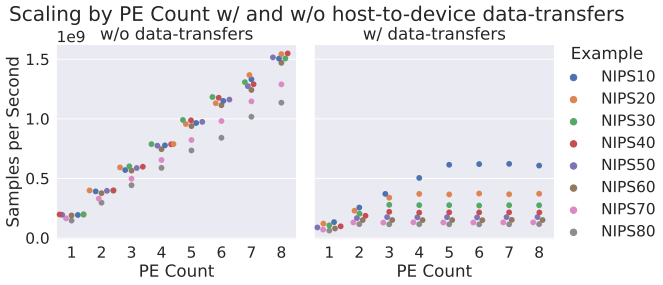


Fig. 4. Comparison of peak performance in samples per second. On the left, host-to-device data-transfers are excluded to disregard a PCIe-based bottleneck. On the right, actual end-to-end performance is measured. Note that the inclusion of the data-transfer time leads to severely skewed scaling.

After examining these results, we conclude that the issue is caused by the host-to-device data-transfers, which are performed using DMA-transfers via the PCIe 3.0 x16 interface of the accelerator card.

C. Scaling Limitations

The previous Section V-B shows that scaling is limited due to the host-to-device data-transfers as well as the available FPGA-resources. Disregarding these limitations, we want to give a perspective on the theoretical limitations of our approach focusing only on the HBM. According to the specification, the theoretical peak bandwidth of the HBM on the BittWare XUP-VVH platform is 460 GB/s (approx. 428 GiB/s). From the HBM benchmark shown in Fig. 2, we see that the practical performance is around 12 GiB/s per channel, assuming that that the blocks accessed are reasonably large.

Given the required data rates for the NIPS10 benchmark (144 bits per sample, 2.23 GiB/s), this means that a channel is easily able to accommodate at least four accelerators. Multiplying by the number of channels (32) would mean that overall, up to 128 NIPS10-accelerators could be used without any memory bandwidth limitations. The required memory bandwidth in that case would be $32 * 4 * 2.23$ GiB/s $= 285$ GiB/s, which is still well below the theoretical limit, as well as the practical limit ($32 * 12$ GiB/s $= 384$ GiB/s). Due to the independent nature of the HBM channels, it is relatively likely that this setup would actually allow linear scaling up to the practical limit of the HBM memory.

The HBM-scaling potential is further highlighted in Fig. 5. Using the data-sizes and samples per second for each benchmark, we calculated the required memory throughput of each

TABLE I
RESOURCE UTILIZATION OF THE COMPARABLE NIPS-BASED BENCHMARKS. "NEW" COLUMNS SHOW THE RESULT OF THIS WORK, WHILE [8] REFERS TO OUR PRIOR WORK.

	kLUTs as Logic		kLUT as Mem		kRegs		BRAM		DSP	
Example	New	[8]	New	[8]	New	[8]	New	[8]	New	[8]
NIPS10	169.8	376.0	66.9	45.4	275.1	530.2	122	360	200	612
NIPS20	180.5	467.0	69.6	54.4	320.7	650.6	126	388	448	1356
NIPS30	230.9	577.3	70.4	62.6	354.4	765.4	122	364	696	2100
NIPS40	241.2	664.1	72.9	75.1	401.6	907.1	132	380	976	2940
Available	1304.0	1182.0	601.0	592.0	2607.0	2364.0	2016	2160	9024	6840

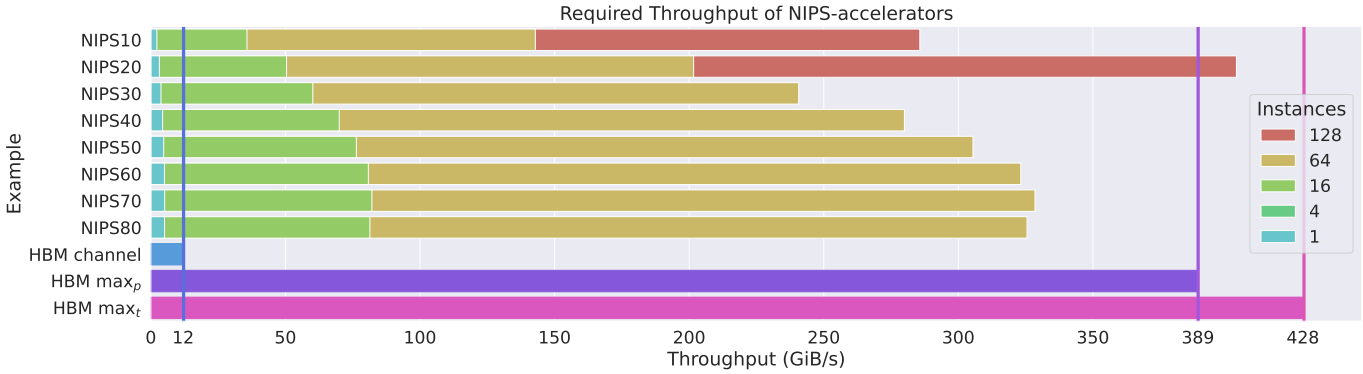


Fig. 5. Scaling potential of the presented architecture under the assumption that logic resources are sufficient and host-to-device bandwidth is available. For each benchmark, we depict the required memory throughput depending on the number of instantiated SPN-cores. The throughput is compared against the maximum throughputs of a single HBM channel as measured in Fig. 2. Additionally, we compare against the practical maximum throughput scaled from our single-channel benchmarks (HBM max$_p$), and the theoretical limit quoted by the vendor (HBM max$_t$).

of the benchmark SPNs. The resulting values are compared against the HBM throughput limitations. The comparison is drawn versus the single-channel result from our HBM benchmark shown in Fig. 2, the practical limitation imposed by 32 channels running at maximum channel throughput and the theoretical limit (as quoted by Xilinx). The vast memory bandwidth provided by HBM could theoretically allow the use of 64 accelerator-instances for all benchmarks, effectively boosting the current performance by up to 8x. For the smaller benchmarks NIPS10 and NIPS20, up to 128 instances could be served by the HBM, which in turn would double performance over 64 instances. While these values are currently out of reach, the improvements provided by the upcoming generations of PCIe will help improve the performance.

To put this into perspective, we look at NIPS80: Using 80 single-byte input values, we are able to process 116,565,604 samples per second. The input data alone requires a bandwidth of 8.7 GiB/s. When we consider the theoretical peak bandwidth of PCIe 3.0 x16, the one-directional theoretical limit is 15.754 GB/s (14.67 GiB/s), which is in practice never reached. For example, current PCIe-based DMA-engines like the Xilinx QDMA or Corundum [14] typically achieve speeds of 100 Gb/s which equates to 11.6415 GiB/s for single-direction transfers. The difference to our NIPS80 example can be explained by imperfect overlapping of the data transfers and the interference with the actual computation. Since the upcoming PCIe generations are specified to *double* the bandwidth with each generation, it is likely that corresponding DMA-engines will allow single-direction bandwidths of approx. 23 GiB/s, 46 GiB/s and 92 GiB/s for PCIe 4.0, 5.0 and 6.0 respectively. While this is still not comparable to the bandwidth of the on-chip HBM, it would definitively allow scaling much further.

In addition, this problem could also be circumvented by different approaches, where host-to-device data-transfers can be omitted due to shared memory, such as the Intel HARP prototype which unifies a high-performance FPGA with a server-grade CPU.

Last but not least, it is important to consider different approaches for delivering data to the SPN accelerators. In [7] for example, we used a streaming-based version of our accelerators to integrate them into a 100G network for in-network inference. The experimental results show that using a reasonable degree of replication, the SPN-accelerators are perfectly capable of performing inference at line rate. In light of the recent advancements in networking and shared memory systems, the potential of HBM becomes even more interesting as a reasonable option for buffering, especially when multiple 100G links are used to transport data in between multiple nodes.

D. End-to-End Performance

In the previous sections, we have discussed the performance results achieved in this work in the context of HBM performance and scaling properties. While the scaling potential is not fully exploited due to the bottleneck imposed by the host-to-device data-transfers, we still want to give a perspective on

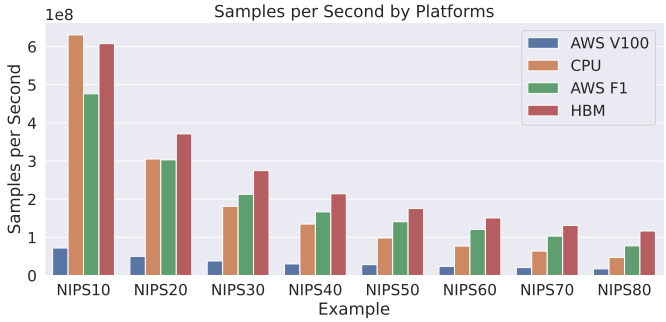


Fig. 6. Peak performance measurements for the different benchmark SPNs on different target platforms. The number of samples per second is calculated from the end-to-end execution time. For AWS F1, V100 and HBM, host-to-device data transfers are *included* in the runtime.

the overall end-to-end inference performance achieved with the HBM-based architecture.

To this end, we use the performance data **including** host-to-device data-transfers and compare it against the results reported by [8]. In this work, the AWS F1-based accelerator is evaluated against a Nvidia Tesla V100, as well as a 12-core Xeon E5-2680 v3 CPU. Both here and in [8], large inference runs are executed to measure the runtime. From the resulting total runtime and the known size of the datasets, the number of samples processed per second can be calculated. The resulting number of samples per second is shown in Fig. 6. The figure gives the best-case result for each target platform and each benchmark from this and the prior work [8].

From these results, it is clear that the Nvidia Tesla V100 is unsuitable for SPN inference due to its much lower overall performance. In contrast, the CPU baseline is able to outperform the AWS F1-based as well as the HBM-based implementation for the small NIPS10 benchmark. The reason for this is the lower compute intensity of smaller SPNs. Since NIPS10 just has a small number of nodes, the cost of the data transfers outweighs the increased compute performance, which results in the CPU outperforming GPUs and FPGAs. However, the CPU's advantage vanishes for increasing SPN sizes: For NIPS20, our HBM-based implementation is already able to outperform the CPU by a speedup of **1.21x**. From NIPS20 to NIPS80 the advantage of the FPGA becomes even greater, yielding a maximum speedup of **2.46x** for NIPS80. The geometrical mean of the speedups is **1.6x** for the CPU. Regarding the V100, the maximum speedup of the HBM-FPGA is **8.4x**, and the geometrical mean of the speedups is **6.9x**.

In comparison to the previous FPGA-implementation, the speedup of using HBM is similar for almost all examples, and close to the geo.-mean speedup of **1.29x**. This is somewhat disappointing, given that our implementation here uses at least twice as many accelerator instances. This less-than-expected speedup can be explained by the scaling limitations due to the limited PCIe bandwidth, as discussed in Section V-B. For the largest SPN (NIPS80), the prior work was not able to use more than two accelerator-cores (instead of four for the smaller benchmarks). In this case we achieve a speedup of **1.5x**.

In context of the adapted architecture from [7], we can get a perspective on the maximum performance of the NIPS80 accelerator: With the 99.078 Gbit/s peak throughput described there, and the 88 bytes of data per sample, we derive a theoretical peak performance of 140,748,580 samples per second. Comparing that to the measured peak performance of 116,565,604 samples per second we achieve in this paper, we see that the streaming-based architecture delivers about 17% increased performance. The reason for this is the much more streamlined architecture, which does not require any memory accesses. In addition, refresh cycles of the HBM also play a role at this level of performance. Taking these factors into account, the HBM-based architecture is very close to its theoretical peak performance, which is capped by the maximum PCIe throughput. Lastly, it is important to realize that the HBM-based architecture targets a different use-case than the streaming-based one: While the in-network streaming implementation makes sense on a very large scale (i.e. data-centers), the HBM-based architecture could be used in smaller high-performance setups. This would also remove the necessity of costly 100G networking infrastructure associated with the streaming approach.

VI. Related Work

While SPNs are still a lesser known machine learning model, they are gaining traction in the field of machine learning. As discussed in Section II-A, they have recently been used in the context of databases, specifically for cardinality estimation as well as approximate query processing [15]. To our knowledge, there is no prior work on accelerating SPN inference (independent of specific applications) apart from our own prior work. Our own prior work begins with [6], which introduced an automatic toolflow to map SPNs to the FPGA. In subsequent publications [4], [11], we looked into the impact of the arithmetic number formats. To this end, [11] introduced a custom logarithmic number system, which enables the computation of very small probability values, and decreases the number of hardware resources required. [4] introduces a customized floating point format, as well as posit number format based on PaCoGen [16]. In [4], the different number formats are optimized towards the SPN use-case and evaluated against each other. In [8], we adapted the original framework from [6] to the UltraScale+-based FPGAs in the Amazon AWS F1 instances. The evaluation showed that the reconfigurable cloud offers high inference performance without the need for expensive on-site FPGA-accelerator cards. Additionally, the F1 instances are able to outperform other state-of-the-art cloud-based hardware, such as a Xeon E5 CPU, and a Nvidia Tesla V100 GPU, for the more compute-intensive SPNs. In our most recent work [7], we adapted the SPN-accelerators to a streaming-based architecture, which in turn allowed us to integrate them into a 100G network for in-network processing of SPN inference. The corresponding work shows the potential of 100G networking for data-delivery to network-attached accelerators, as well as potential performance achievable using

our SPN accelerators. It also shows inference throughput of 99.089 Gbit/s, coming very close to the practical limitations of 100G networking.

While there is (to our knowledge) no other work on hardware-acceleration of SPNs, there is similar work targeting Arithmetic Circuits (AC). Similar to SPNs, ACs are probabilistic graphical models. Both models share similarities and ACs can be transformed into SPNs under certain conditions. Shah et al. [3] have presented a custom processor architecture based on ACs that is able to outperform the Nvidia Jetson TX2 embedded GPU by 12x.

Regarding the use of HBM, the available FPGA-specific research is still rather sparse. This is mainly due to the fact that HBM-enabled FPGA-accelerator cards are relatively new and typically come at a rather high cost. Despite this, there are two important works that explore the advantages and disadvantages of HBM. The work by Lu et al. [17] uses a number of micro-benchmarks to explore the different available memory technologies in recent FPGA-accelerator cards. Specifically, off-chip DRAM is compared against the on-chip HBM memory in a Xilinx Alveo U280 accelerator-card, which features a similar FPGA chip as the one used in our work. The paper goes into detail, how HBM and DRAM can best be used to achieve maximum bandwidth. The other work by Kara et al. [18] examines the use of HBM in the context of a columnar database. Using three database-specific workloads (selection, join, and stochastic gradient descent), the combination of FPGA and HBM is compared against a 14-core Xeon E5 and a dual-socket POWER9 system. In their work, they are able to outperform the server-grade CPU-based implementations by up to 12.9x for the join operation.

VII. Conclusion

In this work we presented an improved accelerator architecture that exploits the parallelism of multiple HBM channels to speed-up inference on SPNs. While the overall performance of our adaption yields speedups of up to 1.5x over the prior work, as well as speedups of 2.46x and 8.4x over a data-center CPU and GPU respectively, the results still fell short of our expectations. We explored this issue and discovered host-to-device DMA transfers via PCIe to be the hard bottleneck.

From our experiments, we conclude that without the host-to-device data-transfers, and disregarding logic and routing resource limitations, almost linear scaling is possible for at least eight accelerators. This trend can likely be continued for up to 64 or even 128 accelerators. While our expectations for the use of HBM have not been met, the current implementation still outperforms prior implementations using the same approach, with CPU-inference of the small NIPS10 benchmark being the only exception. It is important to note that the accelerator card used, was attached using PCIe 3.0 x16. While this is the current de-facto standard, the first PCIe 4.0 devices are already available for end-users. Next-generation PCIe 5.0 and 6.0 devices are also planned to ship within the next two years. Given the ongoing effort in improving PCIe in the future, it is only a matter of time until the full potential of on-chip HBM can be fully exploited for even faster SPN inference. If we take other advancements like 100G networking into account, it becomes clear that the data-delivery is a very important issue. Especially the combination of HBM and 100G networking could be very interesting for high-throughput data-processing in the context of machine learning and artificial intelligence.

References

[1] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "A survey of FPGA based neural network accelerator," *CoRR*, 2017. [Online]. Available: http://arxiv.org/abs/1712.08934

[2] H. Poon and P. Domingos, "Sum-Product Networks: a New Deep Architecture," *Proc. of UAI*, 2011.

[3] N. Shah, L. I. Galindez Olascoaga, W. Meert, and M. Verhelst, "Acceleration of probabilistic reasoning through custom processor architecture," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020.

[4] L. Sommer, L. Weber, M. Kumm, and A. Koch, "Comparison of arithmetic number formats for inference in sum-product networks on fpgas," in *Intl. Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2020.

[5] L. Sommer, C. Axenie, and A. Koch, "Spnc: An open-source mlir-based compiler for fast sum-product network inference on cpus and gpus," in *2022 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2022.

[6] L. Sommer, J. Oppermann, A. Molina, C. Binnig, K. Kersting, and A. Koch, "Automatic Mapping of the Sum-Product Network Inference Problem to FPGA-Based Accelerators," in *36th Intl. Conf. on Computer Design (ICCD)*, Oct 2018.

[7] M. Hartmann, L. Weber, J. Wirth, L. Sommer, and A. Koch, "Optimizing a hardware network stack to realize an in-network ml inference application," in *2021 IEEE/ACM International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC)*, 2021.

[8] M. Ober, J. Hofmann, L. Sommer, L. Weber, and A. Koch, "High-throughput multi-threaded sum-product network inference in the reconfigurable cloud," in *Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC)*, 2019.

[9] R. Peharz, A. Vergari, K. Stelzner, A. Molina, M. Trapp, K. Kersting, and Z. Ghahramani, "Probabilistic deep learning using random sum-product networks," 2018.

[10] A. Molina, A. Vergari, N. D. Mauro, F. Esposito, S. Natarajan, and K. Kersting, "Mixed Sum-Product Networks: A Deep Architecture for Hybrid Domains," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[11] L. Weber, L. Sommer, J. Oppermann, A. Molina, K. Kersting, and A. Koch, "Resource-Efficient Logarithmic Number Scale Arithmetic for SPN Inference on FPGAs," in *Intl. Conference on Field-Programmable Technology (FPT)*, 2019.

[12] A. Molina, A. Vergari, K. Stelzner, R. Peharz, P. Subramani, N. D. Mauro, P. Poupart, and K. Kersting, "Spflow: An easy and extensible library for deep probabilistic learning using sum-product networks," 2019.

[13] J. Korinth, J. Hofmann, C. Heinz, and A. Koch, "The TaPaSCo Open-Source Toolflow for the Automated Composition of Task-Based Parallel Reconfigurable Computing Systems," in *Applied Reconfig. Comp.*, 2019.

[14] A. Forencich, A. C. Snoeren, G. Porter, and G. Papen, "Corundum: An open-source 100-gbps nic," in *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2020.

[15] B. Hilprecht, A. Schmidt, M. Kulessa, A. Molina, K. Kersting, and C. Binnig, "Deepdb: learn from data, not from queries!" *arXiv preprint arXiv:1909.00607*, 2019.

[16] M. K. Jaiswal and H. K. H. So, "PACoGen: A Hardware Posit Arithmetic Core Generator," *IEEE Access*, vol. 7, pp. 74 586–74 601, 2019.

[17] A. Lu, Z. Fang, W. Liu, and L. Shannon, *Demystifying the Memory System of Modern Datacenter FPGAs for Software Programmers through Microbenchmarking*. Association for Computing Machinery, 2021.

[18] K. Kara, C. Hagleitner, D. Diamantopoulos, D. Syrivelis, and G. Alonso, "High bandwidth memory on fpgas: A data analytics perspective," in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, 2020.