Sparsity Enables 100x Performance Acceleration in Deep Learning Networks

A Technology Demonstration

Numenta Whitepaper

VERSION 2.0, May 20, 2021

https://numenta.com/

EXECUTIVE SUMMARY

Deep learning networks today have accomplished a great deal but are hitting bottlenecks as they scale to more complex tasks and bigger models. Researchers attempt to break through the bottleneck by adding more compute power and training data. These enormous models consume vast amounts of power, limiting scalability and creating environmental damage. We need a new algorithmic approach to achieve breakthroughs in performance and scalability.

Although deep learning techniques use neuroscience-like terminology, in fact they operate very differently than the human brain. Unlike deep learning networks, the brain is highly efficient, requiring a mere 20 Watts to operate, less power than a lightbulb. At Numenta, we believe that by studying the brain and understanding what makes it so efficient, we can create new algorithms that approach the efficiency of the brain.

How is the brain so efficient? There are many reasons, but at its foundation is the notion of sparsity. The brain stores and processes information as sparse representations. At any given time, only a small percentage of neurons in the brain are active. This sparsity may vary from less than one percent to a few percent of neurons being active, but it is always sparse. In addition, unlike deep learning networks, the connectivity between neurons in the brain is also highly sparse. In this whitepaper, we demonstrate the application of Numenta's brain-inspired, sparse algorithms to machine learning. Using these algorithms on Xilinx Field Programmable Gate Array (FPGA)s and the Google Speech Commands (GSC) dataset, we show the substantial benefits of leveraging sparsity in order to scale deep learning models.

Sparse networks perform inference 100 times faster than dense networks

This dramatic speed improvement delivers great benefits, enabling:

- Implementation of far larger networks using the same resource
- Implementation of more copies of networks on the same resource
- Implementation of more sophisticated sparse networks on edge platforms with limited resources where the corresponding dense networks do not fit
- Massive energy savings and lower costs due to scaling efficiencies

This technology demonstration is the beginning of a robust roadmap based on our deep neuroscience research. Not only can we achieve speed-ups on the GSC dataset by adding more sparse networks on chip, we also can apply these sparse techniques to other FPGA and other hardware platforms and more complex datasets like image recognition and natural language processing. Further, we can apply sparse networks to training tasks, which could lead to reduced training time and smaller training sets. Moreover, we plan to implement continual learning, which offers the promise of substantial benefits over batch training. Beyond sparsity, as we add more elements of our neocortical model, we expect additional benefits in unsupervised learning, robustness and sensorimotor behavior.

Table of Contents

PERFORMANCE PROBLEMS IN DEEP LEARNING

Over the last decade, deep learning networks have accomplished a great deal but are hitting bottlenecks as they scale to more complex applications. Researchers attempt to break through the bottleneck by creating ever larger models, adding more and more compute power, and more and more training data.[1,2] Additionally, these enormous models consume vast amounts of power, limiting scalability and creating environmental damage.[3] We believe that a new algorithmic approach is required to achieve breakthroughs in performance and scalability.

In contrast to today's deep learning models the brain is amazingly efficient, and provides a roadmap as to how to break through these scaling barriers. By studying the brain and understanding what makes it so efficient, we can create new algorithms based on neuroscience principles.

At Numenta we have done exactly that for over 15 years. Our focus is the neocortex, which is the largest brain region, and the area primarily responsible for our intelligence. The foundation of neocortical efficiency is that the brain stores and processes information as sparse representations. In our past work we have described some of the benefits of sparsity to areas such as robustness and continuous learning. In this whitepaper we show that by applying the principles of sparsity to deep learning, we can lay the groundwork for breakthrough performance acceleration. By implementing Numenta's sparse algorithms on Xilinx FPGAs we demonstrate these principles on inference tasks using the Google Speech Commands (GSC) dataset.

Our results show a speed-up of over 100x[1]

This technology demonstration is the beginning of a robust roadmap based on our deep neuroscience research. Not only can we achieve performance improvements on inference, the principles of sparsity can also lead to dramatically improved training times. Going beyond sparsity, as we incorporate more elements of our cortical model, we can shrink the size of training sets and reduce the need for large, manually labeled datasets. Moreover, we can enable continual learning similar to humans, which will eliminate the need to constantly retrain the model on the entire training set (batch training). Taken together, these techniques will eventually provide several orders of magnitude improvements in scaling. We also expect to see additional benefits in generalization, robustness, and sensorimotor behavior.

[1] A previous version of this paper, V1.0, claimed a 50x improvement. V1.0 featured a sparse-dense network, while V2.0 features a sparse-sparse network, yielding even greater performance improvements.

NEUROSCIENCE SOLUTIONS

The Efficient Brain

It is easy to intuit that the human brain solves problems much more efficiently than a deep learning network. Brains are estimated to require a mere 20 watts of power to perform a wide range of tasks, from reasoning to language, processing visual and auditory inputs and executing complex behaviors.[4] In contrast, today's deep learning networks are energy hogs and often require large amounts of training running on many servers for many days. For example, a recent study from University of Massachusetts, Amherst, showed that a single large Transformer model (a natural language processing model) consumed 656,000 kWh at a cost of $1M- $3M just to train the network.[5]

How is the brain so intelligent with such amazing efficiency? One reason is that most of the neocortex is sparse. It stores and processes information in the context of extremely sparse neural activity and sparse connectivity. Sparsity is foundational to the comprehensive theory of cortical function we have developed called the Thousand Brains Theory of Intelligence[6]. It is beyond the scope of this paper to describe the theory in detail, but it is extensively documented in peer-reviewed papers[7]. It's also described at a higher level in the book *A Thousand Brains*[8]. We discuss applying some additional aspects of the theory in the Future Work section.

Sparse Representations

One of the most remarkable observations about the neocortex is that no matter where you look, the activity of neurons is sparse; only a small percentage of neurons are sending signals at any point in time. The activity might vary from less than one percent to several percent, but it is always extremely sparse. In addition, unlike deep learning networks, the connectivity between neurons in the brain is also sparse. We have shown through mathematical analysis and simulation that sparsity enables efficient use of resources, generalization and robustness. For more details on the nature of sparsity, see Chapter 3 of our digital book Biological and Machine Intelligence (BAMI)[9] and our paper, "How Can We Be So Dense? The Benefits of Using Highly Sparse Representations."[10]

Deep learning has traditionally used dense representations in which the neurons are both highly interconnected and highly active. As can be readily imagined, working with these dense implementations is computationally intensive. To compute the output for each and every neuron, the contribution of each connected neuron much be taken into consideration. This computation is usually formulated as a matrix multiplication, in which each row vector must be multiplied by each column vector.

However, we can create sparse versions of these networks by borrowing several aspects of brain sparsity. Limiting the number of neurons that are active simultaneously is referred to as activation sparsity, while limiting the interconnectedness of the neurons is referred to as weight sparsity. In this sparse network, as a result of both the limited connectivity and limited activations, the matrix multiplications that are required to compute neuron outputs are performed on matrices for which the majority of matrix values are zero. When these sparse rows

and columns are multiplied together, a large fraction of the products can be eliminated. If an implementation is able to 'skip' the computation of the zero products, significant efficiency benefits can be derived from the network sparsity. Weight sparsity and activation sparsity can be leveraged independently or concurrently. The benefits of exploiting activation sparsity in conjunction with sparsified weights is multiplicative, enabling large efficiency improvements. For example, if both activation sparsity and weight sparsity approach 90%, on average only 1% of the products will have two non-zero values. The non-zero multiplications required could therefore be reduced by 100 fold! **The challenge is to train networks to have high levels of sparsity without sacrificing accuracy, and then to pair them with hardware implementations that can efficiently focus computation on the non-zero products.**

Sparsity in Neural Networks

Given the potential of sparse networks, it's not surprising that the sparsification of neural nets has become increasingly discussed over the last few years. Today, a variety of open-source software libraries provide not only tools for sparsifying model weights, but also model runtimes that begin to leverage this sparsity to deliver improved inference performance. Unfortunately, fully exploiting these efficiency improvements can be surprisingly difficult with today's hardware. Most systems perform best on dense computations, where the predictability of memory access patterns allows data to be prefetched in a timely manner, and the dense data packing enables a processor's vector units to be leveraged to full effect.

Furthermore, there is significant variation in the sophistication of techniques via which sparsity is introduced into a model. When removing, or 'pruning' weights, there are a variety of important considerations, including timing, whether to remove weights gradually or all at once, whether to control weight sparsity per layer or globally, and the criteria for selecting which weights to prune.

These choices are extremely consequential and dictate the level of sparsity that can be achieved before model accuracy degrades irreparably. And, as can be readily appreciated, small differences in the final sparsity level can have significant impact on the potential for inference performance improvements. Numenta has developed techniques that allow the creation of extremely sparse models, while maintaining model accuracy. These techniques allowed us to remove 95% of the weights in our GSC CNN model, while delivering an accuracy equivalent to the original dense model. This sparsity level is close to the levels in the brain, and significantly higher than in machine learning models. It provides significant opportunity for performance improvements.

While weight sparsity has become more commonplace, attempts to both promote and exploit sparsity in activations is extremely uncommon. Again, Numenta has developed techniques that allow for extreme activation sparsity while maintaining model accuracy.

In this paper we make two main contributions. Firstly, we illustrate that it is possible to construct highly sparse networks that deliver equivalent accuracy to their dense counterparts. Secondly, we highlight that this sparsity can be fully leveraged by today's hardware, achieving performance improvements over the dense baseline that are an order-of-magnitude larger than typically discussed in contemporary literature.

TECHNOLOGY DESCRIPTION

To validate the efficiency of sparse networks, we compared inference performance between dense and sparse deep learning networks. In order to do so, we went through the following steps:

1. Choose a dataset for the comparison.
2. Create a sparse neural network for the dataset.
3. Choose a hardware platform to run the comparison tests.
4. Implement both the sparse and dense networks on the chosen hardware platform.
5. Run performance tests on both networks.
6. Compare the results.

Choosing the dataset

We chose the Google Speech Commands (GSC) dataset, which consists of 65,000 one-second long utterances of keywords spoken by thousands of individuals. The task is to recognize the word being spoken from the audio signal. This task is representative of modern embedded smart home applications that respond to speech commands. Competitive results on this dataset are in the range of 96-97% accuracy.

Creating the sparse network

We created the sparse network with highly sparse weights and activations, like in the neocortex. Unlike many contemporary approaches to sparse networks we introduce sparsity at the beginning of the training process, rather than simply pruning weights after initial training is completed. This approach allows us to create extremely sparse networks that retain their accuracy. To achieve this result, we made two modifications to the standard deep learning layer (see also Figure 1):

1. We created sparsity in the weights by initializing the weights using a sparse mask, that only permits a fraction of the weights to contain non-zero values. This sparse mask defines the placement of the non-zero weights, and the remainder of the weights are clamped to zero throughout the life of the network.
2. We created sparse activations by retaining only the top-k active units of each layer; the rest are set to zero. This k-winner step is non-linear and can be thought of as a substitute for the ReLU function.

The above formulation is an extension of our previous work on the HTM Spatial Pooler[11], adapted for neural networks trained with back-propagation.

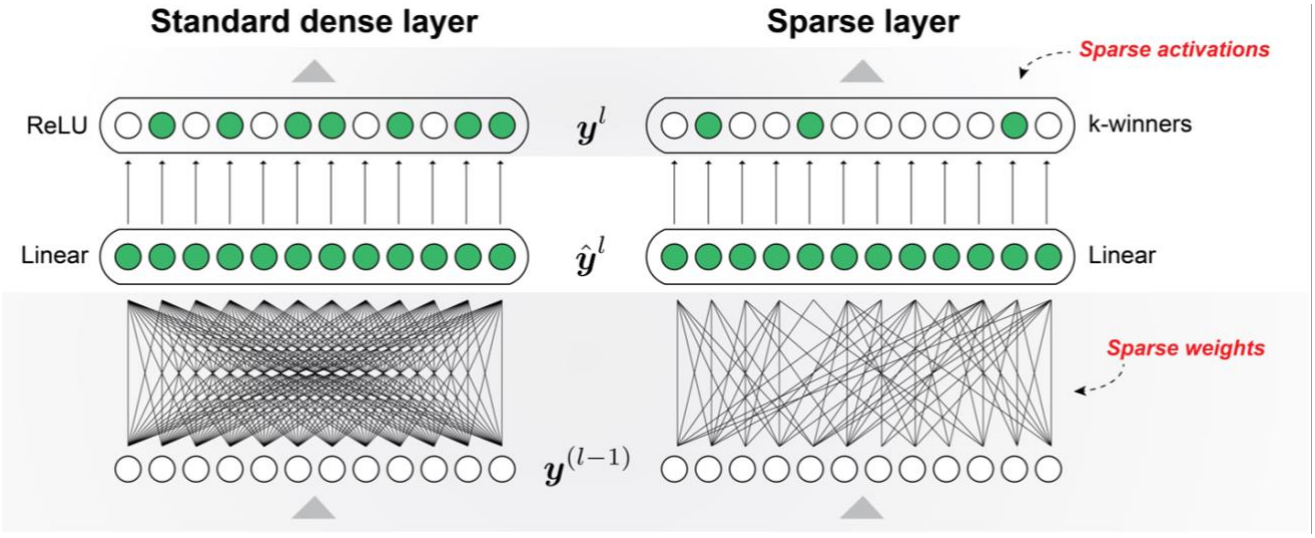

Figure 1: Standard dense layer vs our sparse layer. Our sparse layer contains both sparse weights and sparse activations.

Our dense GSC network is a standard convolutional network with two convolutional layers, a linear hidden layer plus an output layer, as shown in Table 1. As is standard practice in speech processing, the raw audio signals are converted to 32-band Mel spectrograms before being fed to the network. Our sparse GSC network is identical to the dense network except it contains sparse weights and the k-winner take all function as described above. The accuracies of our sparse and dense networks are in the range of 96.4% to 96.9%. The sparse network contains 127,696 non-zero weights compared to 2,522,128 weights in the dense network, or about 95% sparse. The activations in the sparse network range from 88% to 90% sparsity (i.e. 10-12% of the neurons are 'winners'), depending on the layer.

Layer	Filter/neuron count	Filter size	Filter stride	Output Shape
Input	-	-	-	32x32x1
CONV1	64	5x5x1	1	28x28x64
MaxPool1	-	2x2x1	2	14x14x64
CONV2	64	5x5x64	1	10x10x64
MaxPool2	-	2x2x1	2	5x5x64
Flatten	-	-	-	1600x1
Fully Connected	1500	1600x1	-	1500x1
Fully Connected	12	1500x1	-	12x1

Table 1: Architecture of our CNN network trained on the GSC data

The sparsity levels in our networks are much higher than what is commonly seen in the deep learning literature. In such high sparsity regimes, it is possible for a small subset of the neurons to dominate and become active for a large percentage of the patterns. In this situation the network is limited to a small fraction of the possible patterns. To address the issue, we employ a boosting function during training, which favors units that are inactive.

Details of our sparse network, other implementation issues, and our use of the GSC and other datasets are discussed in Numenta's research paper, "How Can We Be So Dense? The Benefits of Using Highly Sparse Representations."[10]

Like any traditional network hyperparameters, the degree of sparsity influences the accuracy of the model. Accuracy equivalent to the dense counterpart can be achieved even with high levels of sparsity, and the sparse network will typically also demonstrate improved generality and improved noise robustness compared to the dense[9]. See Box 1 for more details on how sparsity can yield comparable accuracy. Sparsity levels can be increased even further, with a slight impact to accuracy, effectively trading increased performance for a slight reduction in accuracy. In this paper, we only consider sparse networks with equivalent accuracy to the original dense model, but it should be recognized that even greater speedups could be achieved if accuracy requirements are relaxed.

For our hardware implementation, we apply block sparsity and other structured sparsity spatial constraints to the weights of our sparse network. For example, to create a block-sparse matrix, the weight matrix is structured in a way where a large matrix is divided into smaller matrices with most blocks containing only zero values and a few blocks containing only non-zero values. This structure aids in compression and efficiently using the on-chip processing logic, and we found that, correctly architected, this constraint on sparsity could be introduced without negatively impacting the levels of sparsity achievable or the accuracy of the sparse network. Similarly, light-weight constraints are applied to the locations of the activations. The weights for both sparse and dense networks are quantized to 8 bits.

Box 1: Sparsity, accuracy, and dimensionality

It may not be intuitive how introducing sparsity can yield comparable accuracy. One would expect that if you take information away, you will take a hit on accuracy, so how is it possible to sparsify and retain accuracy? One answer has to do with dimensionality.

Even though most values in a sparse network are zero, you can still convey a lot of information with enough dimensions. To illustrate how, imagine the following scenario.

Suppose a doctor wants to describe a patient with a series of yes/no questions, where a "yes" can be thought of as a non-zero. Here's an example with 10 questions:

1. Is the patient female?
2. Is the patient male?
3. Is the patient above 60?
4. Is the patient between 40 and 60?
5. Is the patient between 30 and 40?
6. Is the patient between 20 and 30?
7. Is the patient younger than 20?
8. Is the patient taller than 6 feet?
9. Is the patient between five and 6 feet?
10. Is the patient shorter than 5 feet?

A typical description will have three yes's and 7 no's (three non-zeros). Those answers will give the doctor limited information about the patient. Now imagine a second description with many more questions, but you are only allowed two yes's. You can now create questions that are more nuanced, such as:

1. Is the patient female and taller than 6 feet?
2. Is the patient female and between five and 6 feet?
3. Is the patient female and shorter than 5 feet?
4. Is the patient male and taller than 6 feet?
5. Is the patient male and between five and 6 feet?
6. Is the patient male and shorter than 5 feet?
7. Is the patient female and above 60?
8. Is the patient female and between 40 and 60?
9. Is the patient female and between 30 and 40?
10. Is the patient female and between 20 and 30?
11. Is the patient female and younger than 20?
12. Etc.

Now the yes's convey even more information than in the first example. Here you can achieve the same description with only two yes's instead of three. If you were allowed hundreds or thousands of questions, even one yes could provide a lot of information about the patient.

In the example, the number of questions is the dimensionality. In summary, the higher the dimensionality, the more informative the non-zero's are, giving you more information about the entire set. If you're interested in exploring the topic of sparsity and dimensionality further, see this research meeting.

Choosing a hardware platform

In sparse systems a majority of the computation results are zero (since a majority of the inputs are also zero). If the machine knows in advance the location of the zeros, it can skip many useless operations. In addition only the non-zero weights need to be stored, which results in a much smaller memory footprint. In principle this idea is simple but in practice it is challenging to find hardware architectures that can exploit both these properties of sparse systems.

We chose an FPGA (Field Programmable Gate Array) as the hardware platform to run the performance tests because of the flexibility it provides in handling sparse data efficiently. FPGAs can do as many arbitrary functions in parallel as it has logical elements (thousands to several million). When processing sparse data, an FPGA can be programmed to ignore zeros and only compute non-zero values, in addition to computing functions such as k-winner. This allows the FPGA implementation to efficiently take full advantage of the sparsity. In contrast, it is currently not possible to parallelize these efficiently in a GPU or CPU. In addition, random access to memory is far more granular and efficient on an FPGA, enabling FPGA implementations to efficiently handle the unstructured access patterns in sparse networks. This ability to program the FPGA in a flexible manner allows it to process sparse data orders of magnitude faster and much more energy-efficiently than a CPU or GPU.

Overall there are two main reasons why sparse networks are more efficient than dense networks on an FPGA platform:

- Fewer computations because the logic on chip can skip zeros, enabling computations with non-zero elements to be performed efficiently
- Smaller memory footprint because only non-zero elements are stored, enabling the chip to run more networks simultaneously

Note that these two reasons have a multiplicative effect when considering overall system throughput. For example, if a sparse network is twice as fast as the dense network, and you can fit three times as many networks on the chip, the sparse system will process six times as many inputs per second as the dense version. Depending on the achievable sparsity of the network, these two factors contribute to a net performance improvement that can be several orders of magnitude higher.

For our technology demonstration, we chose two off-the-shelf Xilinx FPGAs and Platforms: the Alveo™ U250 and the Zynq™ UltraScale+ ZU3EG. The Alveo U250 is a powerful platform designed for datacenters, while the Zynq class of FPGAs is much smaller and designed for embedded applications. Table 2 shows the relative capabilities of the FPGA platforms. As we show later, our sparse network is able to run efficiently on even the smallest of these platforms (unlike the dense network).

FPGA platform	System logic cells	Internal Memory	DSP slices	System power
Alveo U250	1,728,000	54MB	12,288	225W
Zynq™ UltraScale+ ZU3EG	154,000	0.95MB	360	24W

Table 2: This table lists the relative capabilities of the two Xilinx FPGA platforms.

Note that although we feel FPGAs are an ideal platform for this approach, we also believe that current generation CPUs and GPUs would achieve benefits from sparsity, just not as dramatic. In the future, we plan to propose exciting architecture enhancements to CPUs and GPUs that would enable greater use of sparsity for substantial performance gains.

Implementing the networks

We ran the dense and sparse networks on the above Xilinx FPGA platforms. We also used FPGA design tools for programming, block diagram, functional testing, regression and overall integration.

We implemented the dense GSC network using the Xilinx software "Vitis AI," which is a highly optimized solution for deploying deep learning networks on the Xilinx chips. After specifying the parameters and weights, the software generates a complete FPGA design and the required software "drivers" at the OS level.

We implemented the sparse network using a tool called Proximus (see Appendix for details). The sparse GSC network implementation is made up of sparse convolutional layers, sparse linear layers, k-winner-take-all modules, plus input/output (host interface) modules.

Two different sparse networks were implemented:

1. A 'Sparse-Dense' implementation: in this version the hardware implementation took full advantage of the sparse weights, but did not take advantage of the sparsity in the activations (i.e. they were considered dense). This is the simplest form of sparse network to implement, as the sparsity pattern in the weights is pre-determined for a specific network and the hardware implementation can be explicitly optimized for this pattern. In contrast, while only k neurons will be selected as active at each layer, the location of these winners is input dependent and is constantly changing.
2. A 'Sparse-Sparse' implementation : in this version the hardware implementation took full advantage of the sparsity in both the weights and the activations.

We compare the performance of these two sparse networks against the dense network baseline, and each other, to highlight the benefits of fully exploiting the sparsity in the model. Performance details of the Sparse-Dense implementation have been previously discussed by Numenta[12].

Since each sparse network instance is small compared to a dense network instance, multiple sparse GSC network instances can fit in one FPGA. In the FPGA implementation described below, using an Alveo U250 board, up to five sparse-sparse GSC networks fit in one "Super Logic Region" (SLR). There are four SLRs on an Alveo U250, which means there are 20 sparse-sparse network instances on the full Alveo board, compared to four total dense network instances (one per SLR).

For more information on implementation details, see the Appendix.

Running the performance tests

We ran the performance tests on each of the FPGA platforms, installed in a server. The dense network and sparse networks tests ran on the same card, sequentially, by downloading the selected network into the card and then feeding in input data. For the purposes of this test, the input data is a repeating sequence of 50,000 pre-processed audio samples (audio sample processing is not part of these tests).

FPGA platform	Network type	Throughput words/sec	Speedup over dense
Alveo U250	Dense	3,049	-
Alveo U250	Sparse-Dense	35,714	11.71
Alveo U250	Sparse-Sparse	**102,564**	**33.63**
ZU3EG	Dense	0	-
ZU3EG	Sparse-Dense	21,053	Infinite
ZU3EG	Sparse-Sparse	45,455	Infinite

Table 3: Shows throughput, measured as the number of speech words processed per second, for a **single** dense, sparse-dense and sparse-sparse network on two different platforms. The throughput of the sparse-sparse network on the Alveo U250 is 33.6X times faster than the fastest dense configuration.

DETAILED RESULTS

In this section we describe the measured performance of dense vs. sparse networks on the various platforms. For sparse networks we show a couple of different configurations where we vary the number of network copies that are placed on the chip. We compare performance using three different metrics: throughput, power usage, and resource utilization.

FPGA platform	Network type	Number of networks on chip	Full chip throughput (words/sec)	Full chip speedup
Alveo U250	Dense	4	12,195	-
Alveo U250	Sparse-Dense	24	689,655	56.5
Alveo U250	Sparse-Sparse	20	**1,369,863**	**112.3**
ZU3EG	Dense	0	0	-
ZU3EG	Sparse-Dense	1	21,053	Infinite
ZU3EG	Sparse-Sparse	1	45,455	Infinite

Table 4: Shows throughput, measured as the number of speech words processed per second, for dense, sparse-dense and sparse-sparse networks on two different platforms for full-chip configurations. The throughput of the 20 network sparse-sparse configuration on the Alveo U250 is 112.3X times faster than the fastest dense configuration.

Throughput

Our throughput metric measures the total number of inputs processed per second. Tables 3 and 4 show the throughput on each platform for both dense and sparse networks. Table 3 presents the results for a single network on the chip. As can be seen in the right-hand columns of Table 3, the sparse networks outperform the dense network baselines by a considerable margin. On the Alveo U250 a single sparse-dense network is more than 10 times faster than a single dense network, while a single sparse-sparse network is over 30 times faster. These speedups are clearly highlighted in the left-most set of columns in Figure 2.

Since the large Alveo U250 has sufficient resources to accommodate multiple networks on chip, Table 4 shows full-chip throughput numbers, where we pack as many copies of each network as possible onto the FPGA. The U250 can accommodate 4 copies of the dense network, 24 copies of the sparse-dense network and 20 copies of the sparse-sparse network (The additional logic required to route the sparse activations to the appropriate sparse weights increases the size of the sparse-sparse implementation, slightly reducing the full-chip network count in comparison to the sparse-dense version. However, the performance benefits of leveraging activation sparsity more than compensate for the decreased network density). With 24 network copies on an Alveo U250, the sparse-dense networks can process data at 689,000 words/second, **more than 56 times faster** than the dense implementation on that platform, while the sparse-sparse networks

can process data at 1,369,000 words/second, **more than 112 times faster** than the dense implementation. These speedups are clearly illustrated in the right-most set of columns in Figure 2. It is also interesting to note that a single sparse-dense or sparse-sparse network, a small fraction of what can be accommodated on a U250, nevertheless outperforms the full-chip dense result by 2.9X and 8.4X respectively!

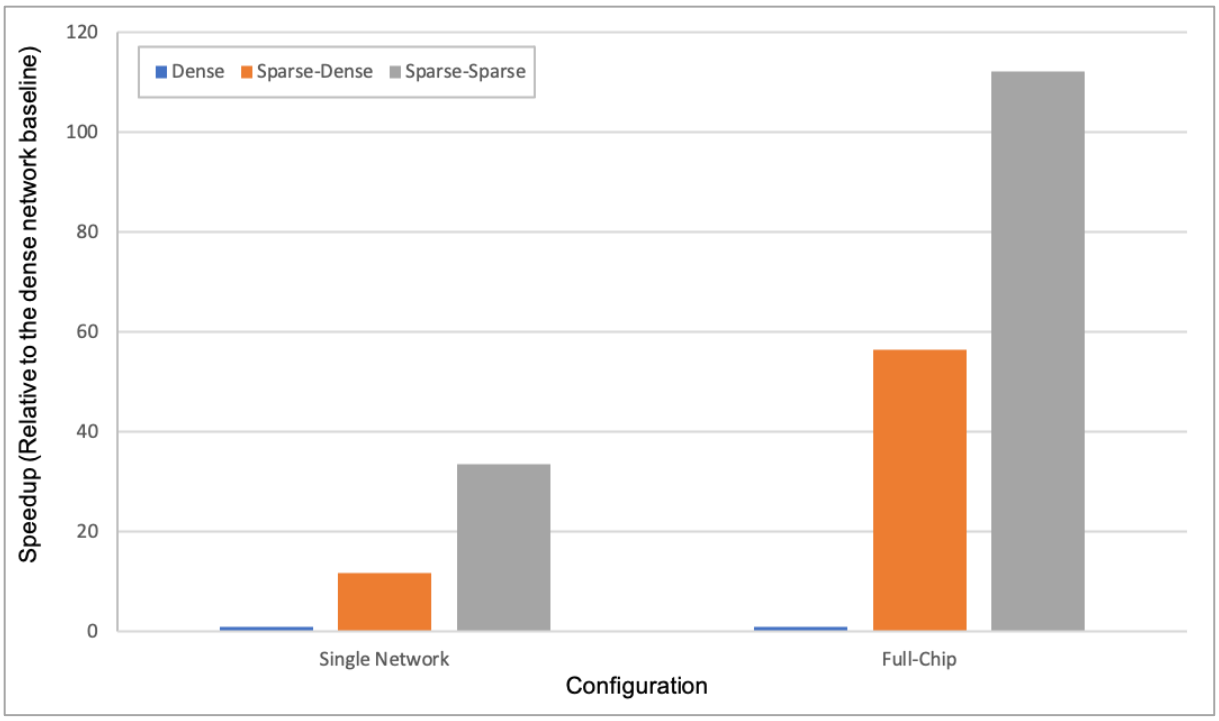

Figure 2: An illustration of the performance benefits of sparse networks, for both single-network and a full-chip (U250) configurations

Note that the per network speed drops as we pack more networks on chip. This effect is likely due to communication bottlenecks, since the amount of data that has to be transferred per second grows with the number of networks running in parallel, coupled with the slight reductions in clock speed that can be required for full-chip configurations. Still, the gain in overall throughput is far more than the drop in each network's speed.

The small ZU3EG FPGA results shown in Tables 3 and 4 are also extremely interesting. The dense GSC network cannot fit on that system. The sparse networks are significantly smaller and thus we can fit a single sparse-dense or sparse-sparse network on that platform (see the Resource utilization section below). Interestingly, the throughput of that single sparse-sparse network on the small chip is 3.7X times faster than the total throughput of four dense networks running on the powerful Alveo U250 (45,455 words/sec vs 12,195 words/sec). This result opens up new product categories where ultra-small, energy efficient, embedded platforms can run deep learning based applications without compromise, helping make "AI at the Edge" a reality.

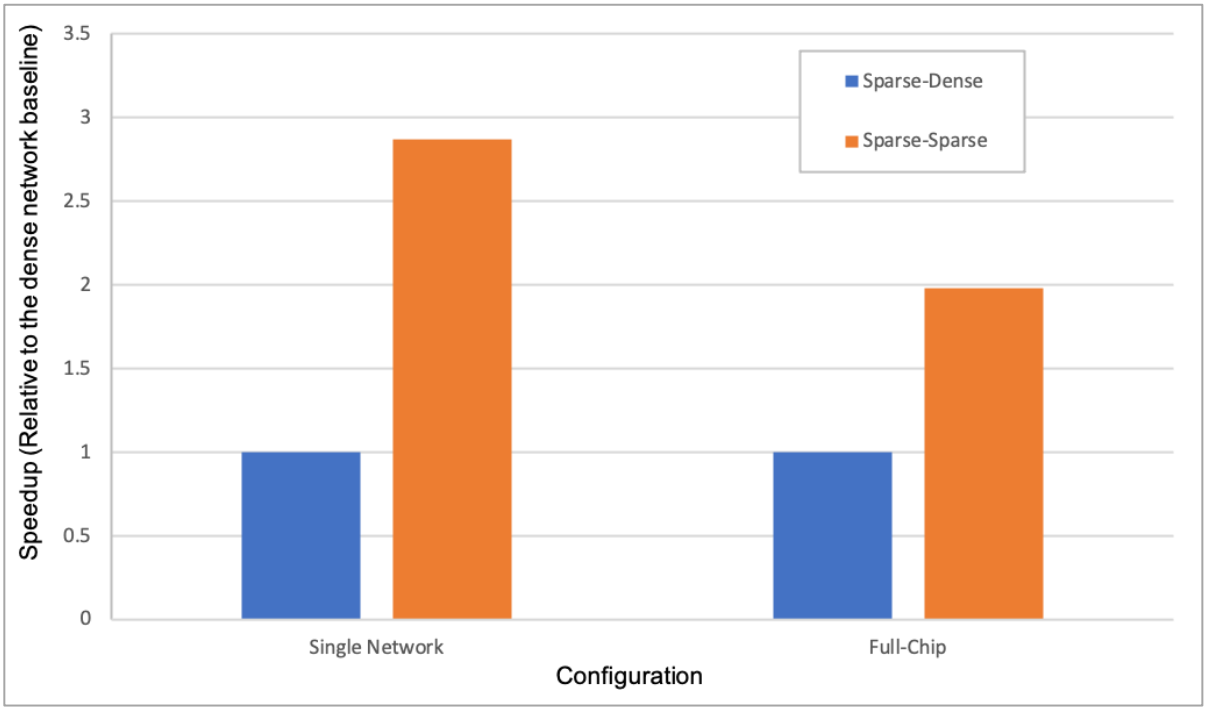

Figure 3: An illustration of the performance benefits of sparse-sparse networks, compared with a sparse-dense baseline, for both single-network and full-chip (U250) configurations

In addition to comparing the sparse networks to the dense baseline network, we can also compare the sparse networks to each other. Remember, in the sparse-dense implementation only sparsity in the weights is exploited for computational efficiency, while in the sparse-sparse version sparsity in both the weights and the activations is exploited. By exploiting both elements of sparsity we double the performance of the sparse network (as illustrated in Figure 3), allowing the sparse-sparse implementation to increase throughput by two orders of magnitude compared with the dense baseline!

Power usage

Power utilization is rapidly becoming an important criterion in measuring the efficiency of deep learning systems. We use the metric words/second/watt to evaluate power usage. Table 5 shows the numbers for dense vs. sparse networks on the two platforms.

The relative efficiency column on the right measures the power improvement of each implementation relative to the dense network baseline on the Alveo U250. As can be seen, the 20 network sparse configuration on the Alveo U250 is over 50X more power efficient than the dense configuration, while the sparse-sparse configuration is over 100X more power efficient. Similarly, while the absolute performance delivered by the ZU3EG is lower than the U250, the power efficiency is significantly higher, providing a compelling AI at the edge solution when combined with the sparse networks.

Note that measuring exact power usage is tricky. In Table 5 we use the max wattage measurement of the development board as our power consumption. A specific product using a custom board should get significantly better absolute power usage across the board, perhaps by as much as a factor of 4. Nevertheless, we expect the general trends and the relative power efficiency of sparse networks to largely reflect the results shown in Table 5. It is indisputable that sparse networks are far more efficient than dense networks.

FPGA platform	System power	Network type	Number of networks	Words/sec/ watt	Relative efficiency
Alveo U250	225	Dense	4	54	100%
Alveo U250	225	Sparse-Dense	1	158	292%
Alveo U250	225	Sparse-Dense	24	**3065**	**5675%**
Alveo U250	225	Sparse-Sparse	1	455	842%
Alveo U250	225	Sparse-Sparse	20	**6088**	**11274%**
ZU3EG	24	Dense	0	0	0
ZU3EG	24	Sparse-Dense	1	877	1624%
ZU3EG	24	Sparse-Sparse	1	1893	3505%

Table 5: Overall power usage, measured in words processed per second per watt, for each configuration. The relative efficiency column, measured against the dense U250 implementation, shows that the sparse networks are far more efficient than the dense

Resource utilization

FPGA platforms have a diverse set of compute and memory components, each with different but overlapping capabilities. Optimizing any implementation often involves balancing between these various resources. Table 6 shows the percentage utilization of these resources for various sparse network configurations (we did not have access to the dense network utilization numbers).

FPGA platform	Network type	Network copies	LUT usage	BRAM usage	URAM usage	DSP
Alveo U250	Sparse-Dense	1	1.64%	1.72%	2.66%	3.56%
Alveo U250	Sparse-Dense	24	38.26%	50.29%	63.75%	85.46%
Alveo U250	Sparse-Sparse	1	2.97%	4.50%	4.06%	3.07%
Alveo U250	Sparse-Sparse	20	63.12%	86.93%	96.56%	61.44%
ZU3EG	Sparse-Dense	1	49%	80%	NA	94%
ZU3EG	Sparse-Sparse	1	79.6%	95.2%	NA	88.3%

Table 6: FPGA resource utilization for different sparse configurations.

As can be seen from the first and third rows of the table, a single sparse-dense or sparse-sparse network takes up a tiny percentage of the overall resources on an Alveo U250. This result means that you can have several networks running in parallel, while still leaving significant room for the rest of the application. Sparse networks offer much more flexibility than dense networks in achieving high throughput while still allowing room for other complex application code.

Finally, while the sparse-sparse implementations do consume more resources than their sparse-dense counterparts, the significant throughput increases achieved by sparse-sparse implementation more than offsets the increase in resource utilization.

Comparison with GPUs

Our primary goal in this whitepaper is to highlight the performance advantages of sparse representations vs. dense representations. To do so, we held the platform constant, implementing both sparse and dense networks on the same FPGA platforms. We did not implement an optimized sparse network on GPUs. In this section we provide some approximate performance numbers of the dense network on two GPU systems to get a rough sense of the relative speeds.

We used PyTorch to run the dense network on two popular NVIDIA platforms: the Tesla™ K80 and the Tesla™ V100. Table 7 shows the throughput of the dense network on these platforms for various batch sizes (GPU performance is optimized for high batch sizes). Overall, the dense network has a consistently higher throughput on GPUs than does the dense network on the Alveo. However, our sparse networks are significantly faster than any of the dense implementations, FPGA or GPU. Although it is difficult to compare across widely different architectures, there is no doubt that an FPGA running a sparse network as described here will have a substantial price performance advantage over a GPU running a dense network.

Note that these numbers should only be used to get a very rough sense of comparative performance. There are numerous factors that come into play, such as transistor counts, price points, chip size, and manufacturing density. In addition to the differences between chips, the software implementations are very different. The PyTorch implementation uses 32 bit floating point numbers, whereas the Alveo implementation uses 8 bit integer numbers. It is likely that the GPU throughput of the dense network could be increased with a more optimized implementation. Nevertheless, the large gap between the sparse and dense network throughputs shows the clear advantages of our optimized sparse implementations.

Platform	Network type	Batch size	Overall throughput
Alveo U250	Dense	500	12,195
Alveo U250	Sparse-Dense	N/A (streaming)	689,655
Alveo U250	Sparse-Sparse	N/A (streaming)	1,369,863
Tesla K80	Dense	256	16,024
Tesla K80	Dense	1024	17,710
Tesla K80	Dense	8192	20,118
Tesla V100	Dense	256	45,450
Tesla V100	Dense	1024	61,638
Tesla V100	Dense	8192	54,301

Table 7: Throughput for dense networks on two GPU platforms for different batch sizes. Our sparse networks significantly outperform all dense implementations.

Recently NVIDIA has started to invest more heavily in sparsity, particularly in their Ampere architecture[13]. Our networks are much sparser than the ones they showed and our networks incorporate both activation sparsity as well as weight sparsity. Given the promise of such highly sparse networks, it is possible that additional improvements to the underlying GPU architecture could eventually lead to much larger benefits.

Summary of results

As can be seen throughout the above discussion, sparse networks offer significant performance benefits over dense networks. An individual sparse network is faster than a comparable dense network. Since sparse networks are much smaller than dense networks, more copies can be implemented on the same chip, improving throughput even further. Sparse networks are far more energy efficient, and our optimized sparse implementation is significantly faster than dense networks running on more powerful chips (both FPGA and GPU).

FUTURE WORK

This technology demonstration validates that sparsity will be a key factor in scaling deep learning networks. We are working with strategic partners to commercialize this technology.

Future work will proceed in a couple of directions. First, these impressive performance results are not specific to the GSC dataset or the particular CNN network presented in Table 1. Rather, Numenta has developed a generalized approach for creating highly sparse networks for which the potential efficiency gains associated with the network can be realized on current hardware architectures, such as FPGAs, We are in the process of applying these techniques to more complex networks (such as ResNet and Transformer networks), more challenging datasets, and on additional hardware platforms, with the goal of clearly demonstrating the broad applicability of these techniques to deep neural architectures. Second, this whitepaper has focused on inference tasks, but the same principles apply to training. We plan to create a technology demonstration to validate that sparsity can significantly improve the efficiency of training deep learning networks.

Sparsity is foundational to the Thousand Brains Theory, but it is only the beginning. Deep learning models have had significant challenges beyond performance that can be addressed by implementing more of the neocortical theory. To begin with, a major challenge faced by these networks is an inability to learn continuously. As new data arrives, a large model needs to be retrained in batch mode in order to update it, using huge additional resources. Our brains adapt continuously with each new data point. Moreover, today's machine learning models require supervised learning with labeled data while your brain is able to rapidly classify similar objects without labeling. The Thousand Brains Theory neuron model describes how a brain is continuously updated and how it learns without supervision. In the future we can apply these techniques to machine learning in order to enable continuous learning and unsupervised learning.

Another major challenge is that deep learning models are notoriously brittle[14]. Small changes to the input or missing information can completely throw off inference. The Thousand Brains Theory describes how our brains constantly make predictions, and constantly improves by learning from mistakes in these predictions. The ability to make accurate structured predictions explains why our brains are so robust. It explains how we are able to effortlessly fill in missing pieces of a scene, such as the lower half of a person blocked by a car door. The same principle can be applied to improve the robustness of deep learning models.

Finally, deep learning models have yet to be successfully applied to sensorimotor behavior in advanced robotics. If we want to create more powerful machine intelligence, we need to be able to make decisions and implement actions. The Thousand Brains Theory, which Jeff Hawkins writes about in his recent book, *A Thousand Brains*[8]*,* explains how reference frames provide a framework to extend the success of machine intelligence in static tasks to sensorimotor tasks in robotics.

CONCLUSION

Since the beginning of the field of AI over fifty years ago, scientists have speculated that the brain, as the only demonstration of intelligence in the universe, can show the path towards implementing machine intelligence. Yet, over these many years, and in spite of spectacular growth in the field of neuroscience, little has been made of this possibility. Instead, the field of AI has focused on inefficient techniques enabled by vast amounts of compute power and data, quite different from the neocortex. As these techniques reach their inevitable limitations, turning to the brain for insights has become not just an alternative, but a necessity, to advance the field. Today, with a far more complete understanding of the human brain, we now see a clear roadmap on how to apply these concepts to building efficient, intelligent machines. We propose a starting point of using sparsity to dramatically improve the performance of deep learning networks. As we continue to implement more and more of the Thousand Brains Theory in algorithms, we are confident that we are finally on the path to machine intelligence.

REFERENCES

1. Thompson, N., Greenewald K., Lee, K., Manso, G. (2020). *The Computational Limits of Deep Learning.* MIT Computer Science and A.I. Lab. https://arxiv.org/pdf/2007.05558.pdf
2. "AI and Compute: Addendum." Open AI.com Blog. November 7, 2019. https://openai.com/blog/ai-and-compute/#addendum
3. Hao, K. (2019). "Training a single AI model can emit as much carbon as five cars in their lifetimes" *MIT Technology Review*, June 6, 2019. https://www.technologyreview.com/2019/06/06/239031/training-a-single-ai-model-can-emit-as-much-carbon-as-five-cars-in-their-lifetimes/
4. Elert, G. (2001). *Power of the Human Brain.* Available at https://hypertextbook.com/facts/2001/JacquelineLing.shtml
5. Strubell, E., Ganesh, A., McCallum, A. (2019). *Energy and Policy Considerations for Deep Learning in NLP.* 57th Annual Meeting of the Association for Computational Linguistics (ACL). Florence, Italy. July 2019. https://arxiv.org/abs/1906.02243
6. Hawkins, J., Lewis, M., Purdy, S., Klukas, M., Ahmad, S. (2019). *A Framework for Intelligence and Cortical Function Based on Grid Cells in the Neocortex*. Frontiers in Neural Circuits 12, 121. https://doi.org/10.3389/fncir.2018.00121
7. Hawkins, et al. Numenta research papers. https://numenta.com/papers
8. Hawkins, J. A Thousand Brains: A New Theory of Intelligence, Basic Books, March 2021.
9. Hawkins, J. et al. 2016-2020. Biological and Machine Intelligence. Release 0.4. Accessed at https://numenta.com/resources/biological-and-machine-intelligence/
10. Ahmad, S., Scheinkman, L. (2019). *How Can We Be So Dense? The Benefits of Using Highly Sparse Representations.* https://arxiv.org/abs/1903.11257
11. Cui, Y., Ahmad, S., Hawkins, J. (2017). *The HTM Spatial Pooler—A Neocortical Algorithm for Online Sparse Distributed Coding.* Frontiers in Neuroscience, 11. https://doi.org/10.3389/fncom.2017.00111
12. "Numenta Demonstrates 50X Speed Improvements On Deep Learning Networks Using Brain-Derived Algorithms", Numenta.com Blog. November 9, 2020. https://numenta.com/press/2020/11/10/Numenta-Demonstrates-50x-Performance-Acceleration-Deep-Learning-Networks
13. "How Sparsity Adds Umph to AI Inference." NVIDIA.com Blog. May 14, 2020. https://blogs.nvidia.com/blog/2020/05/14/sparsity-ai-inference/
14. Su, J., Vargas, D., Kouichi, S. (2019). *One pixel attack for fooling deep neural networks*. IEEE Transactions on Evolutionary Computation, Vol.23 , Issue.5 , pp. 828--841. https://arxiv.org/abs/1710.08864

REVISION HISTORY

The table notes changes between versions. Minor changes such as small clarifications or formatting changes are not noted.

Version	Date	Changes
1.0	Oct 30, 2020	Initial release
2.0	May 20, 2021	Initial release showed 50x performance improvements using sparse-dense implementation. V2.0 shows 100x performance using sparse-sparse implementation

APPENDIX

Reproducibility

The performance tests can be reproduced by a third party on their own Alveo U250 board. Contact sparse@numenta.com if you're interested in learning more.

Glossary

- High Level Synthesis (HLS): an automatic generation of electronic circuitry from a high-level algorithmic description (for example in C++).
- Register Transfer Level (RTL): a design abstraction which models a synchronous digital circuit in terms of the transfer of data between hardware registers (memory or flip-flops), and the logical operations performed on that data.
- Super Logical Region (SLR): Most of Xilinx FPGA devices in the "Alveo" line consist of multi-chip-modules comprised of several physical chips. Each of the chips are referred to as an SLR. It is an important aspect in the design of a system because there is a slight timing impact in the transition between the SLRs and the number of connections is limited.
- Proximus: A proprietary FPGA integrated development environment that allows the design of systems at the block level by expressing the overall function as communicating parallel processes.
- Vitis: Part of the Xilinx FPGA design tool offering, this is the high-level platform which deals with software components, hardware drivers, high level design entry and HLS. Vitis translates high level designs (C++) into RTL.
- Vitis AI: Xilinx library-based AI offering for FPGA design which allows the designer to parameterize a wide range of different AI networks and map them to a subset of available chips and cards.

- Vivado: Xilinx physical design integrated development environment (IDE) in the Vitis platform, used to do the mapping of an electronic design to the FPGA chip, using RTL-synthesis.

Design Flow

1. Using Proximus, a block diagram is developed which consists of communicating functional parts, in this case the different layers of a neural network (CNN1, CNN2 etc.). Each block can contain a purely functional description (for example in C++) or contain more levels of blocks which eventually contain simpler functional descriptions (for example a multiplication). Functionality is verified at this level and given some assumptions (e.g. # of cycles for operations), performance can be estimated and trade-offs can be made at this point in the design.
2. The design is then exported into the Xilinx tool set, where Vitis generates the hardware drivers and does High Level Synthesis. Circuit simulation can be done at the HLS stage.
3. Assuming there are no timing problems found, Vivado then does RTL and physical synthesis, place and route, timing verification and bitsteam generation. Circuit simulation can be done at the RTL level during this stage.
4. The design is then transferred to the physical FPGA. Proximus then connects to the FPGA hardware and performs the overall system execution.

Sparse network implementation details

The next several figures walk through the sparse network implementation details.

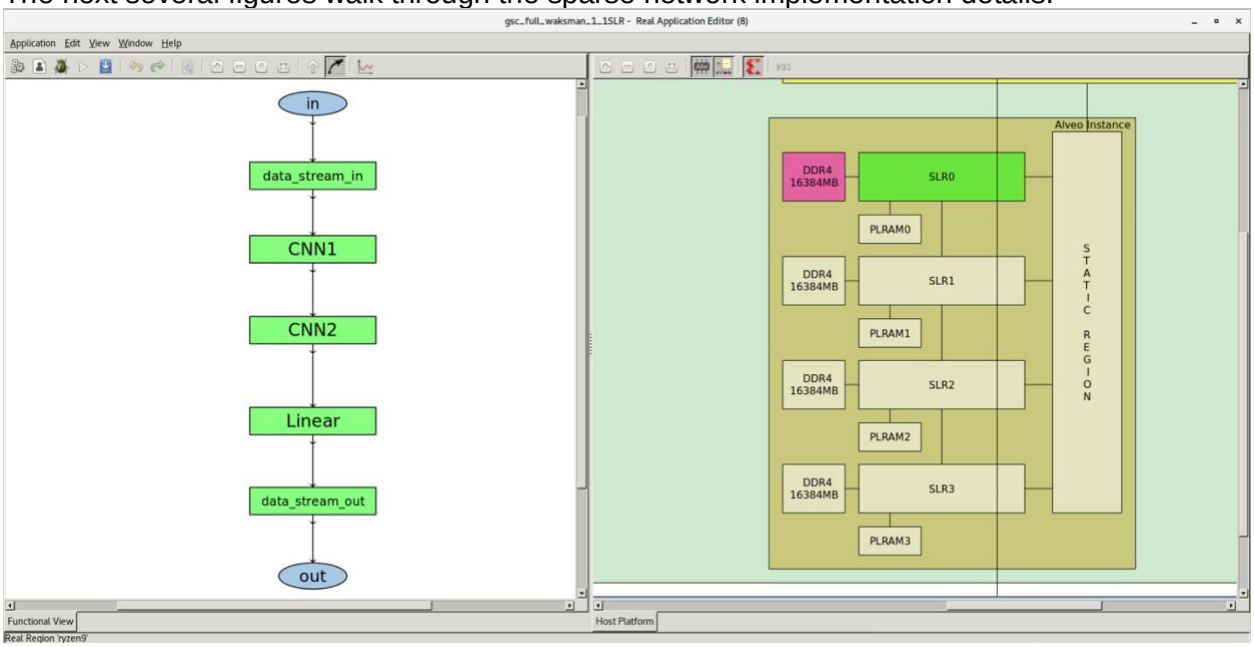

Figure A. Single sparse-sparse GSC network in one SLR, shown in Proximus. The right side of the figure is a high-level block diagram of the Alveo U250, showing all four SLRs. SLR0 is highlighted, and the left side of the figure shows the block diagram of a single copy of the GSC sparse-sparse network instance which is implemented in this SLR.

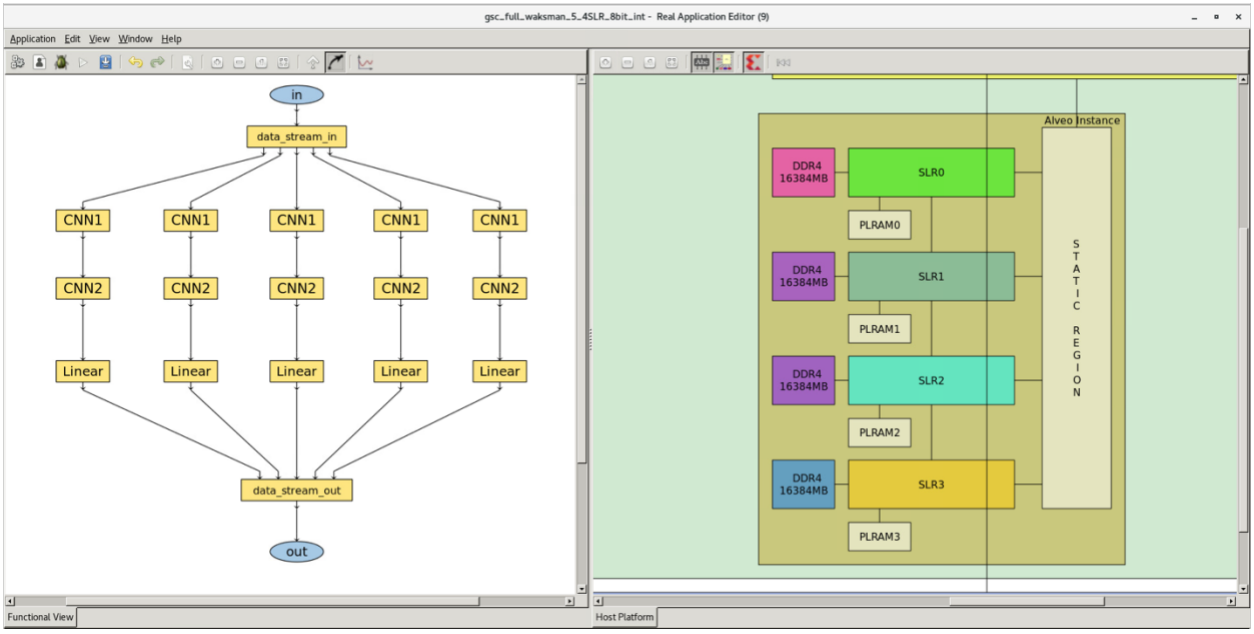

Figure B. Multiple sparse-sparse GSC networks in one SLR, shown in Proximus. The right side of the figure is a high-level block diagram of the Alveo U250, showing all four SLRs. The left side of the figure shows five instances of the sparse-sparse GSC network which are implemented in one SLR. The inputs are distributed into these five networks in round robin fashion using a map reduce algorithm. Each box in the left side of the diagram contains C++ code implementing the function of the network layer.

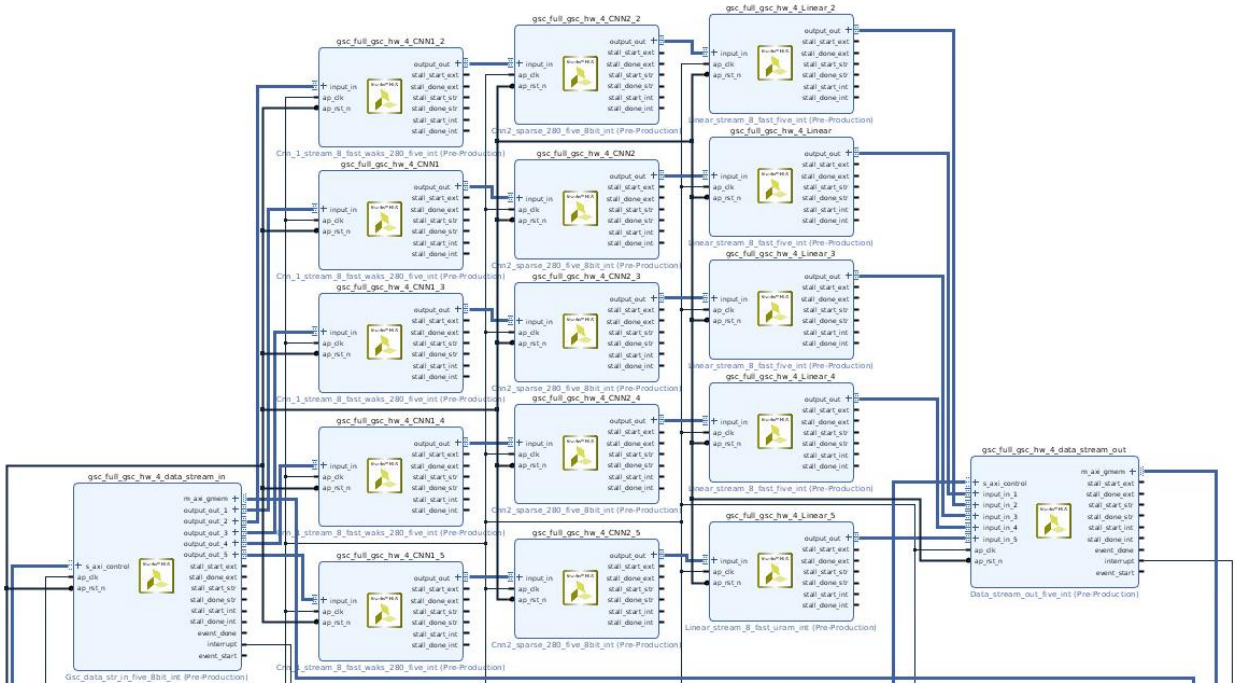

Figure C. Multiple sparse-sparse GSC networks in one SLR, shown in Vivado, after the design is exported from Proximus into Vitis, and high-level synthesis has run. Figure C is equivalent to Figure B, but shown in a different tool view. The dataflow is shown top-down in Proximus (Figure B) and left-right in Vivado (Figure C). Each box in Figure C contains synthesized RTL (Register Transfer Level).

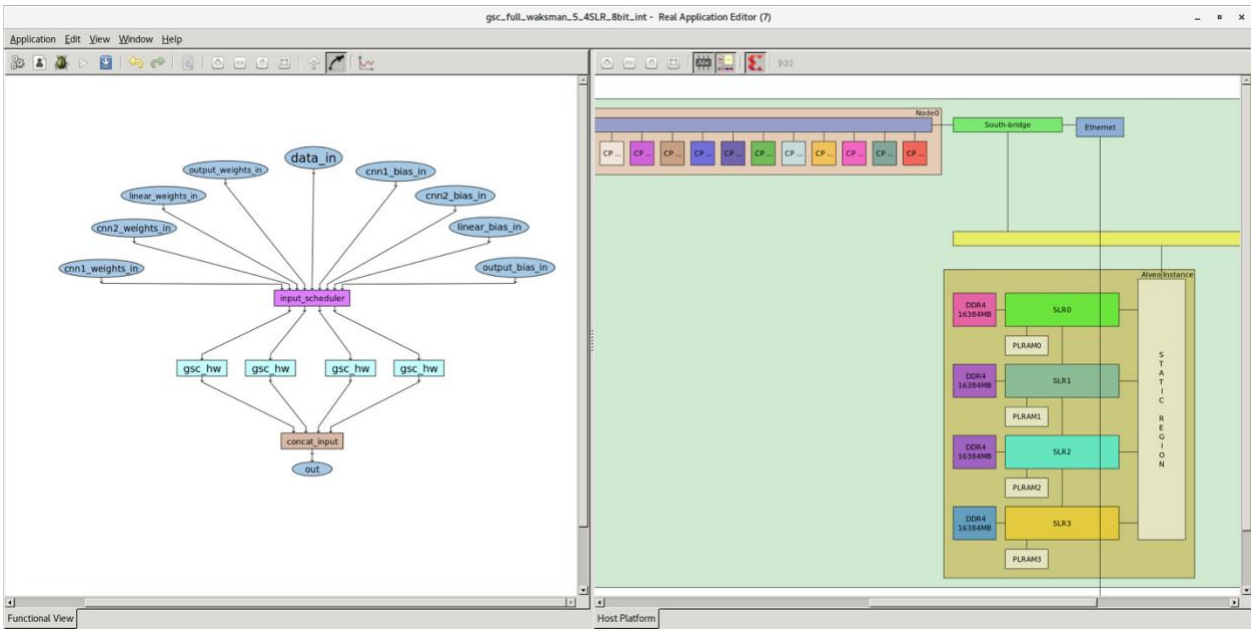

Figure D. Multiple GSC networks distributed across 4 SLRs, shown in Proximus. The right side of the figure is a high-level block diagram of the Alveo U250, showing all four SLRs. The left side of the figure shows the full chip design, with each “gsc_hw” block equivalent to the block diagram shown in Figure B, representing one SLR for a total of four SLRs. Each “gsc_hw” block contains 5 copies of the sparse-sparse GSC network, with a total of 20 networks implemented on the chip.

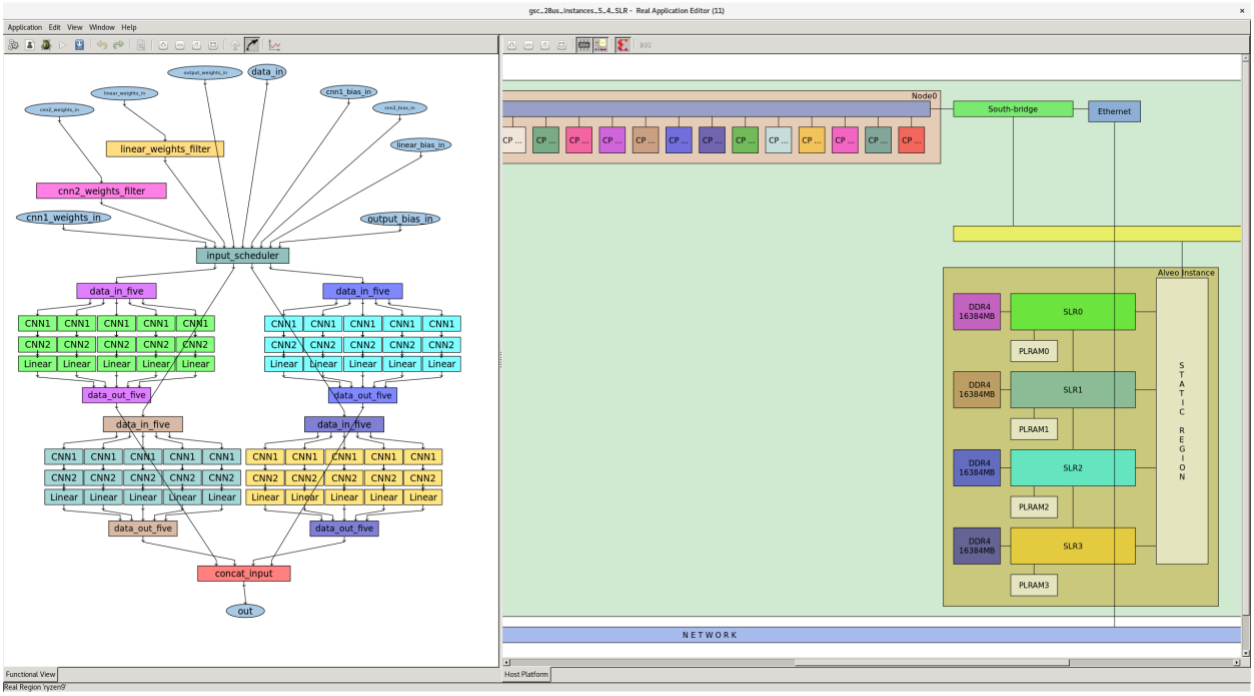

Figure E: The same, full design of 5 networks in each of 4 SLRs on Alveo U250, in flattened (hierarchy removed) view, shown in Proximus.

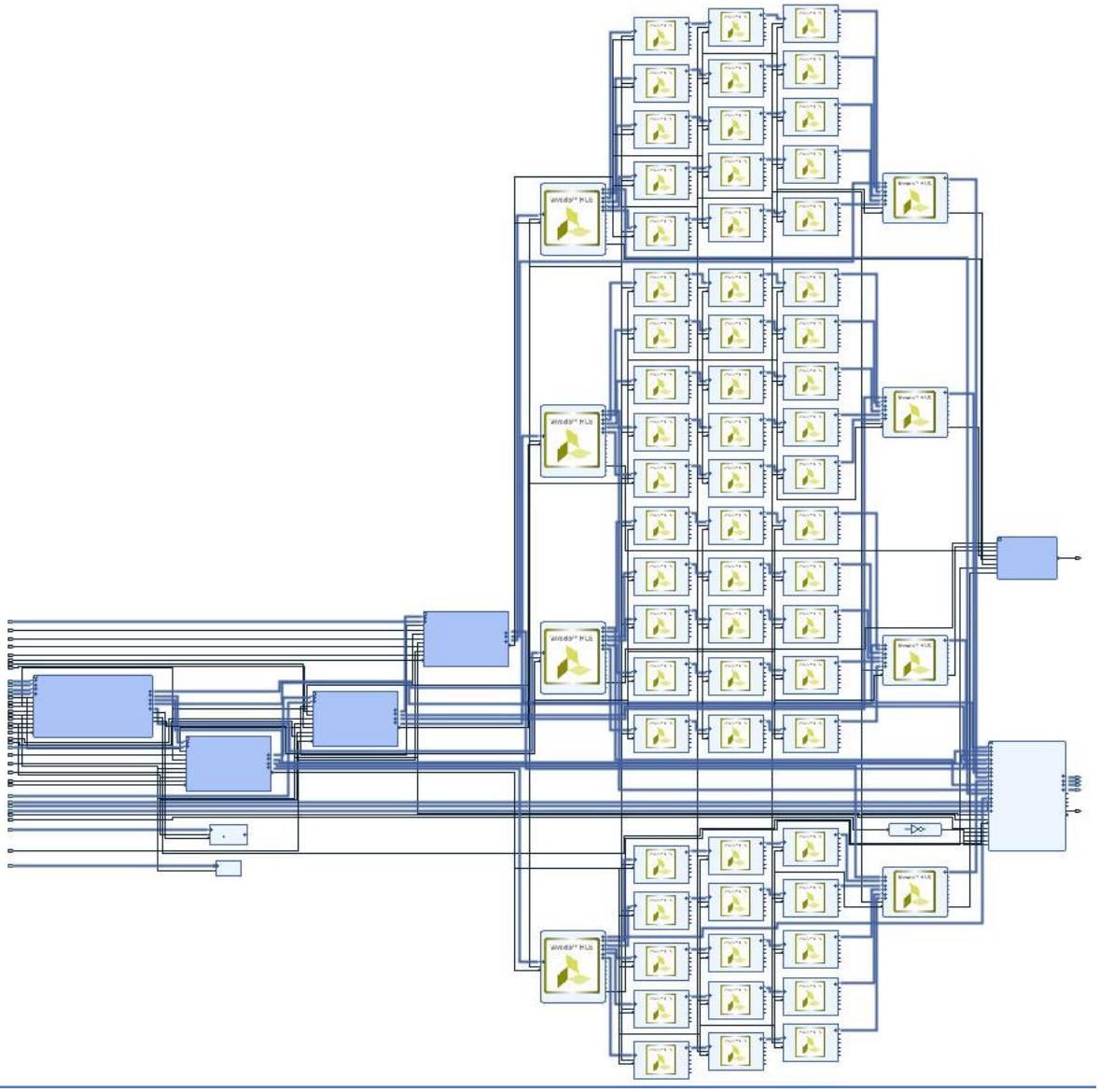

Figure F. This is the entire design (20 network copies distributed over 4 SLRs) shown in Vivado. The block level diagram shows the 5x4 logical designs with their two-level distribution and map-reduce logic. In addition, to reduce the dependency on slight timing-differences between modules, each one has a FIFO module on its input as well as output.

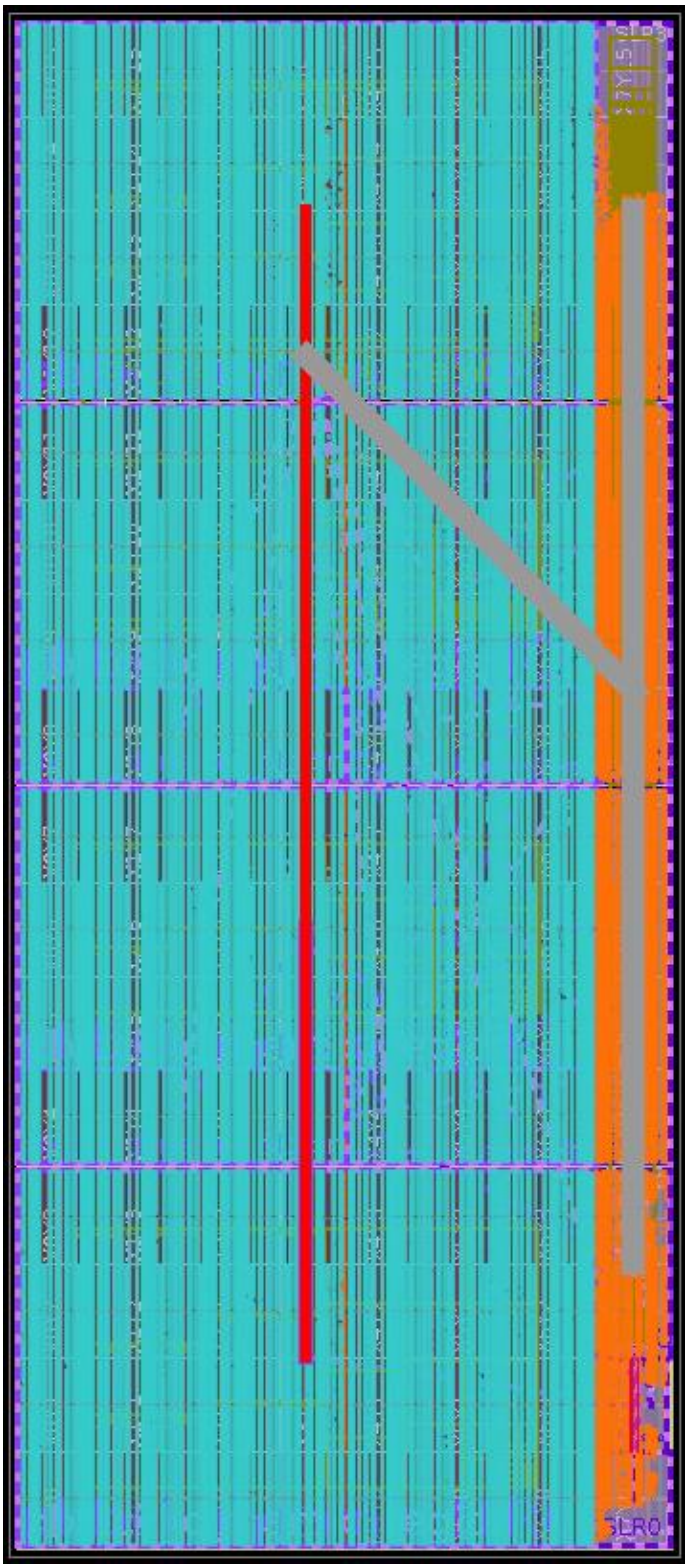

Figure G. Pictured here is the physical layout of 5 sparse-sparse instances per SLR, 20 in total, on the Alveo U250.

The parts in the design are built automatically by Vitis-HLS, which translates C++ from Proximus into RTL. Then Vivado synthesizes RTL into FPGA gate level and places and routes the design on the FPGA.

In this Vivado physical view (which is the result of the “place and route” process) the 4 SLRs can be seen clearly stacked vertically. You can also see the “static region” which is the space reserved for the host interface (PCI-e 3.0x16) as well as the DDR4 interfaces to the 4 parallel memory DIMM available on the board. The logic is smeared as a large number of tiny pieces (LUT, DSP, BRAM, URAM and routing) and depicted in light blue.