Making a Case for Systolic Arrays for Graph CNN Inference

MOHAMMAD ALI JAUHAR, École Normale Supérieure Paris-Saclay, France

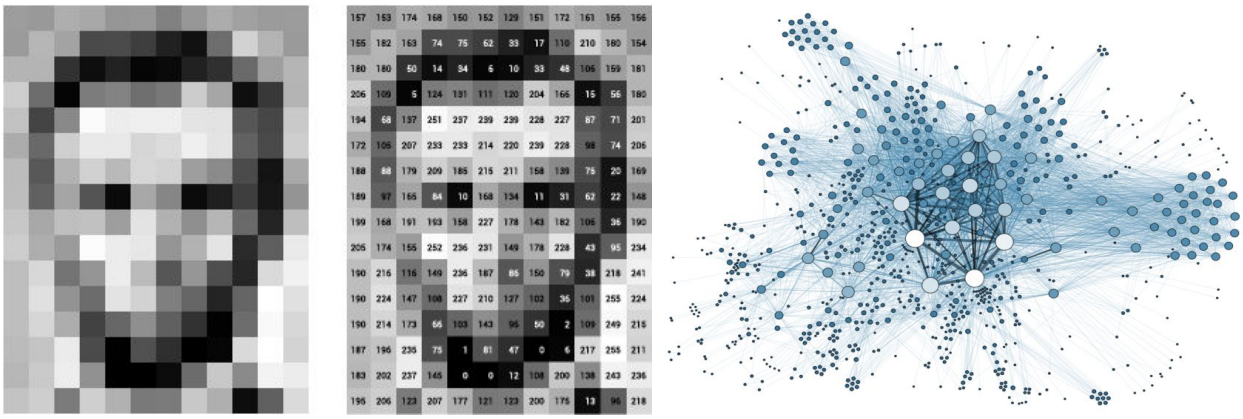

Fig. 1. Left, an image as a 2-d array which has very regular and right, a visualisation of a social network graph, which is highly irregular. Image by Martin Grandjean. [Public domain], via Wikimedia Commons. (https://commons.wikimedia.org/wiki/File:Social_Network_Analysis_Visualization.png)

Deep Learning has already out-performed humans on a number tasks involving images, speech, and natural language data. However, it still hasn't overcome many challenges. First, it is compute and memory intensive, in both training and inference. Therefore, there is a constant research and development towards developing specialised hardware and software tools to optimise learning and inference. Secondly, deep learning doesn't work out-of-box on data that do not have an underlying regular structure akin to images. Such data include social network graphs, 3D mesh, networking log data, etc. Therefore, techniques have emerged recently that try to use deep learning on unstructured non-Euclidean data. In this report, we will have a look at two works tackling each of the above mentioned challenges and we will try to reason if it is possible to marry both ideas to create a more computationally feasible solution.

Additional Key Words and Phrases: Systolic arrays, Graph neural networks, CNNs, Custom AI hardware, Accelerators

ACM Reference Format:
Mohammad Ali Jauhar. 2024. Making a Case for Systolic Arrays for Graph CNN Inference. 1, 1 (February 2024), 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Deep learning [7] is a technique for learning complex representations from data on multiple levels of abstractions by the virtue of having multiple layers of processing hierarchy. Convolutional neural networks (CNNs) [8] are one such realisation. They are efficient learners and have the ability to learn local structures and composing them to form a multi-layer structure has proven to be very successful across a variety of tasks such as machine translation, speech recognition, object detection, image segmentation, etc. [7] involving very big datasets of images, natural language, speech, etc. These types of data have an underlying Euclidean structure and their success with deep learning lies in modelling their invariances in the neural networks. CNNs, for example, model the statistical properties such as stationarity and compositionality from local statistics.

However, in many applications, the data that emerges doesn't have a Euclidean structure [2]. Social network graphs, word embeddings, 3D mesh, etc. are few examples. One common way of representing non-Euclidean data is using graphs, which are usually represented using a set of vertices, a set of edges, and an adjacency weight matrix. The resulting structure could be visualised as in Fig. 1.

While the current hardware and software offerings is highly tuned for undertaking deep learning on Euclidean data, they are not at all designed keeping non-Euclidean structures in mind. At the same time, we could see non-Euclidean structures as a generalisation of the grid-like Euclidean structures (images, speech, etc.) in higher dimensions. However, such generalisation comes at the cost of incorporating arbitrary irregularity in the structure which presents two significant challenges which are the focus of the current report.

- Convolutional (and associated) operations are not well defined for non-Euclidean structures.
- GPUs and deep learning accelerators are designed for regular grids and hence working with non-Euclidean data is very inefficient.

Author's address: Mohammad Ali Jauhar, mohammad.jauhar@ens-paris-saclay.fr, École Normale Supérieure Paris-Saclay, 4, avenue des Sciences, Gif-sur-Yvette, France, 91190.

In the next sections, we would have a brief overview of two papers, one incorporating convolutional neural networks on graphs using fast localised spectral filtering (Defferrard, et al.) [3], and another that proposes a scalable systolic array structure for deep neural network inference accelerator (Yüzügüler, et al.) [9].

2 SCALE-OUT SYSTOLIC ARRAYS

With deep learning methods' continued success in many applications, development of custom hardware for deep learning is an active area of research and we have many commercial offerings as well such as Google TPUs [5], Tesla FSD chips [1], Graphcore [4], etc. In hardware terms, they are application specific integrated circuits (ASICs). Designing such systems is not straightforward. Decisions have to be taken at level of design of the processing elements, data partitioning mechanism, scheduling scheme, and the interconnect network between processing elements, group of processing elements, and memory banks. For example, the IPU processors of Graphcore uses scratchpad memory to reuse data temporally and flexible interconnection. This however, translates to an increase in the number of memory access and data movement, decreasing both throughput and throughput/Watt i.e. power efficiency. Systolic arrays, on the other hand, adopts a much simpler architecture by doing away with scratchpad memory and implementing static interconnection, thereby increasing higher power efficiency and peak throughput.

2.1 What are Systolic Arrays?

Systolic arrays are an example of systolic architecture which aims to mimic the biological process of blood pumping to and from the heart. In such systems, data flows from computer memory in a rhythmic fashion, passing through many processing elements (PEs) before it returns to the memory [6]. As with most ASICs, the key to systolic arrays is having a simple, regular design which keep the number of unique parts small and regular. They are designed for specific applications and in our case, it is for doing deep learning. While GPUs have been the workhorse for undertaking deep learning workflows, they are still general purpose processors.

Such systems are different from the usual process execution mechanism in that they don't need to be kept a track of once the data has been sent to the processing elements, and thus translates to less overhead. Furthermore, as one of the most important design goals for a processing systems is to decrease latency and thereby increase utilisation and throughput. The data transfer from memory to cache is one major bottleneck and often times, it results into under-utilisation of the processing elements. Systolic systems, when implemented correctly, helps increase utilisation by not requiring return of partial sums back to memory and then reload them. And by having a series of processing elements that work on the output of previous processing elements, there is a considerable increase in throughput as well.

The most important and common arithmetic operation in deep learning is matrix multiplication. Therefore, having matrix multiplication units is at the heart of designing any deep learning accelerators. Even modern GPUs have dedicated matrix multiplication units (aka tensor processing units). Systolic arrays are no different when used in deep learning accelerators. Modern systolic systems are complex structures with many processing units, interconnect network between different groups of processing units, and memory buffers. This complexity is necessary to handle large data concurrently and to also make the accelerators available to multiple tenants, for example, in the case of Google Cloud TPUs.

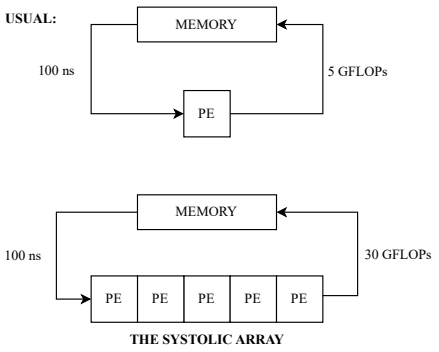

Fig. 2. Basic Principle of Systolic System

2.2 Why Scale-out Systolic Arrays?

Current systolic arrays achieve great peak throughout due to a higher level of silicon integration, power envelopes, and memory bandwidth. However, translating peak throughput to higher utilisation has proven challenging because of the mismatch between workload requirements and array size. Most well known implementation of systolic structure is found in Google TPUs and their TPUv1 and TPUv4 reported a utilisation of about 22% and 33%, respectively. Google has incorporated few coarse-grain systolic array pods in TPUv3 and TPUv4 to increase utilisation. While they do perform better than their monolithic predecessors, the challenge in variability in array size requirements remain a fundamental limitation. An array size too small results though maximises utilisation, they reduce power efficiency by over-provisioning on-chip memory. An array size too big results into under-utilisation and reduced power efficiency as many array elements in the PEs remain unused while at the same time consuming power.

Besides array size, interconnect also plays a key role in utilisation and effective throughput in multi-pod accelerators. As such, an ideal interconnect should provide high bisection among pods and on-chip memory to enable a continuous flow of operands and results. It should also provide high multiplexity to allow distribution of data and operand with minimum contention and delays. This means it should also have a low round-trip latency. Finally, as the total number of pods in an accelerator could grow to a large number, the interconnect should be highly scalable as well.

Even though we might optimise for array size very well, in practice, there is usually mismatches between array size and data size.

This requires partitioning data into tiles and scheduling their processing. Hence, tiling and scheduling are also important factor for achieving better utilisation and effective throughput.

Keeping in mind the above mentioned factores, scale-out systolic arrays (SOSA) proposes a multi-pod architecture that supports both single and multi-tenacy workloads. It incorporates optimaly sized systolic arrays and relies on a Butterfly network for interconnection between systolic arrays and memory banks for its scalability, high bisection bandwidth, and low latency. Finally, it implements a fixed-length tiling strategy that tries to maximise utilisation across a large number of pods along with a scheduler that maps the tiles onto systolic pods while handling interconnect constraints, tile dependencies, and memory bank conflicts.

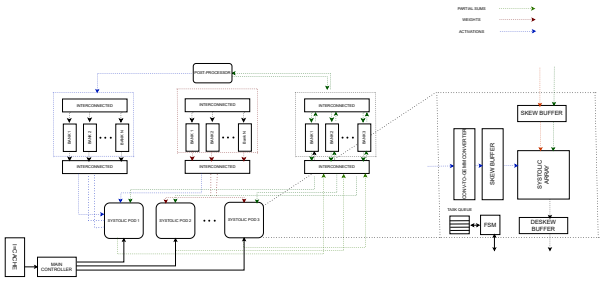

Fig. 3. SOSA Architecture

2.3 Scale-out Systolic Array Architecture

Deep neural networks contain a number of different types of layers such as convolutional, attention, and fully connected and we can express their computation as general matrix multiplication (GEMM) operations. The GEMM operations could be expressed in the form $XW + P_{in} = P_{out}$ where X and W are the neural network's activations and weights, respectively. P_{in} and P_{out} describe the input and output partial sums, respectively. Each systolic array is designed to do GEMM.

The main controller fetches instructions from a dedicated cache, assign them to corresponding pods, and synchronise their operations in locksteps. Activations, partial sums, and weights are stored in dedicated on-chip memory banks. A sinlge instruction multiple data (SIMD) post-processor performs element-wise on the partial sums to returns it back to activations or partial-sum banks as per the requirements.

Each systolic pod includes a systolic array along with few other components to reduce interconnect traffic. These are CONV-to-GEMM converters and skew/deskew buffers. The CONV-to-GEMM converter is a dedicated hardware that converts the activation data from four-dimensional convolution to two-dimensional matrix format to prevent redundant loading operations. Similarly, skew buffers apply a skew to activations and input partial sums, while deskew buffers remove the skew from the output partial sums. Fig. 4 shows the SOSA architecture.

Table 1. Performance of SOSA with various array sizes

Array Size	# Pods	Peak Power (Watts)	Util.	Eff. Throughput
512x512	1	113.2	10.3	191.3
256x256	8	245.0	14.0	183.0
128x128	32	283.1	13.8	205.0
64x64	128	362.2	17.4	200.9
32x32	256	260.2	39.4	317.4
16x16	512	210.6	40.0	198.9

2.4 Results

Array size of $(32x32)$ provides the highest effective throughput, while $(16x16)$ provides the highest (marginally more than $(32x32)$) utilisation. Furthermore, Butterfly network provide the best performance in connecting large number of systolic pods. Finally, a tiling mechanism where tiles are of fixed size, (rxr) performs the best, where r is the array size.

3 CNN ON GRAPHS WITH FAST SPECTRAL FILTERING

CNNs are efficient in extracting meaningful statistical patterns in large high-dimensional datasets. They do it due to the stationarity assumptions. However, such assumptions are not valid in non-Euclidean domains. Data emerging from a variety of applications are non-Euclidean and many among could be structured on graphs. Therefore, extending CNNs to graphs is desired. However, this is not straightforward as the building blocks of a CNN, i.e. convolution, pooling, etc. are not well defined for irregular graphs. This work takes the tools from graph signal processing (GSP) to develop spectral formulation of CNNs and design computationally feasible local spectral filters for graphs as something analogous to convolution filters and an analogous pooling operation.

3.1 Developing Filters for Graph Convolution

While there is no unique definition for convolution on graph in the spatial domain, there are well defined localisation operator for convolution using Kronecker delta (δ_i) in the spectral domain. For a graph $G = (V, E, R)$ with n vertices and a weighted adjacency matrix $W \in R^{nxn}$, we could define a signal x as:

$$x : V \rightarrow R, x \in R^n \tag{1}$$

Defining graph Laplacian as $L = D - W$, where $D \in R^{nxn}$ is diagonal matrix with $D_{ii} = \sum_j W_{ij}$. As L is a symmetric positive semi-definite matrix, its eigenvalues λ_i are non-negative and it has a complete set of orthonormal eigenvectors U. Hence, it could be decomposed into $U\Lambda U^T$.

The graph Fourier transform for a signal x could then be represented as $\hat{x} = U^T x$.

Filtering in graph signals is defined as:

$$y = g_\theta(L)x = Ug_\theta(\Lambda)U^T x \tag{2}$$

Where $g_\theta(\Lambda)$ is the filter. Since this filter is non-parameterised, it is not lcoalised in space. Furthermore, the learning complexity for such filter is $O(n)$, so we look for alternative options, which is polynomial parametrisation of the filters. For such filters, the

Table 2. Classification accuracies of graph CNN and a classical CNN on MNIST

Model	Architecture	Accuracy
Classical CNN	C32-P4-C64-P4-FC512	99.33
Graph CNN	GC32-P4-GC64-P4-FC512	99.14

Table 3. Time to Process a batch of 100 MNIST Images

Model	Architecture	CPU (ms)	GPU (ms)
Classical CNN	C32-P4-C64-P4-FC512	210	31
Graph CNN	GC32-P4-GC64-P4-FC512	1600	200

filter g_{theta}, centred at vertex i is given by $(g_\theta(L)\delta_i)_j = (g_\theta(L))_{i,j}$ $= \sum_k \theta_k (L^k)_{i,j}$. Since Kronecker delta functions are used, filters are K-localised.

To further improve filter complexity, it can be truncated as $g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\Lambda})$, where $\theta \in R^K$ is a vector of Chebyshev coefficients and $T_k(\tilde{\Lambda}) \in R^{nxn}$ is Chebyshev polynomial of order k. $\tilde{\Lambda}$ is Λ scaled to $[-1, 1]$.

3.2 Feature Map and Gradient Descent

The output feature map could be roughly given as:

$$y = \sum g_\theta(L)x \tag{3}$$

Where x is the input signal/feature map. For backpropagation, two gradients, $\frac{\delta E}{\delta_\theta}$, and $\frac{\delta E}{\delta x}$, where E is the loss over a batch of samples, is needed. The calculations involve K sparse matrix-vector multiplications and one dense matrix-vector calculation and hence should be able to be computed efficiently on GPUs using tensor operations.

3.3 Graph Pooling

Pooling operations require meaningful neighbourhood on graphs. This could be done by graph clustering. However, since clustering is a NP-hard problem, we have to resort to heuristics. This work used Graclus algorithm to clusterise the graph and coarse it. After clustering follows converting the graph into binary tree and then rearranging the vertices. Now the graph signal is a 1-D vector and pooling could be done efficiently on it.

3.4 Results

Graph CNNs with fast localised spectral filtering show performance of the same order as their classical counterparts, as seen in Table 1. However, they fare noticeably worse in their compute requirements. They are around x8 times slower on a CPU and around x7 times slower on a GPU.

4 OBSERVATIONS

- SOSA architecture shows promising results. However, it is based on empirical data and therefore would translate poorly to novel datasets and novel architectures.
- Inference in graph CNNs should be possible with systolic arrays. However, as SOSA has taken the dimensions of input activations as key design parameter, graph signals of arbitrary structure, as in equation (3), would be poorly optimised for inference on SOSA.

5 FUTURE WORK

As we move towards incorporating structures of arbitrary irregularity into deep learning, we increasingly see specialised hardwares being developed which in turn are heavily optimised for specific structure in input data and operations associated with it. This, in a way, presents us with diverging development. This could lead us into thinking that we may need more general purpose hardware for conducting deep learning on generalised non-Euclidean data. However, this couldn't be the solution since graph data are slower than Euclidean data on both GPUs and CPUs. Therefore, I believe that we may have to abandon the approach of finding graph analogue to classical deep learning building blocks such as convolution and instead look to develop alternative approaches altogether.

REFERENCES

[1] Pete Bannon, Ganesh Venkataramanan, Debjit Das Sarma, and Emil Talpes. 2019. Computer and Redundancy Solution for the Full Self-Driving Computer. In *2019 IEEE Hot Chips 31 Symposium (HCS)*. 1–22. https://doi.org/10.1109/HOTCHIPS.2019.8875645

[2] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric Deep Learning: Going beyond Euclidean data. *IEEE Signal Processing Magazine* 34, 4 (July 2017), 18–42. https://doi.org/10.1109/msp.2017.2693418

[3] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (Eds.), Vol. 29. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2016/file/04df4d434d481c5bb723be1b6df1ee65-Paper.pdf

[4] Zhe Jia, Blake Tillman, Marco Maggioni, and Daniele Paolo Scarpazza. 2019. Dissecting the Graphcore IPU Architecture via Microbenchmarking. arXiv:1912.03413 [cs.DC]

[5] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture* (Toronto, ON, Canada) *(ISCA '17)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3079856.3080246

[6] Kung. 1982. Why systolic architectures? *Computer* 15, 1 (1982), 37–46. https://doi.org/10.1109/MC.1982.1653825

[7] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436.

[8] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324. https://doi.org/10.1109/5.726791

[9] Ahmet Caner Yüzügüler, Canberk Sönmez, Mario Drumond, Yunho Oh, Babak Falsafi, and Pascal Frossard. 2023. Scale-out Systolic Arrays. *ACM Trans. Archit. Code Optim.* 20, 2, Article 27 (mar 2023), 25 pages. https://doi.org/10.1145/3572917