Performance Analysis of DNN Accelerator using System Simulator

Mohamed Matar
Electrical and Computer Engineering
University of British Columbia
momran@ece.ubc.ca

Deval Shah
Electrical and Computer Engineering
University of British Columbia
devalshah@ece.ubc.ca

Abdul Rehman Anwer
Electrical and Computer Engineering
University of British Columbia
abanwer@ece.ubc.ca

William Yang
Electrical and Computer Engineering
University of British Columbia
dingqingy@ece.ubc.ca

Abstract

In this report, we show a case study of integrating a Deep Neural Network (DNN) simulator with a system simulator (gem5) to enable the study of DNN accelerator based systems and their effect on system performance, as well as estimate their overhead on the system. We implement a software layer using NNAPIs to communicate with the hardware simulator (DNNSIM) integrated within gem5 [1]. We use system calls to emulate the interface between software and hardware which is a feature provided in gem5.

1 Introduction

Hardware acceleration has become an essential part of SoC design due to the increased performance requirements of applications need. Current hot applications such as video decoding, image processing, cryptography, and machine learning require hardware accelerators to perform the intensive computation to meet the performance requirements. These accelerators can have a varied effect on the performance of the overall system. This need made the community, of SoC architecture and hardware design, think of software simulators to model systems that include different kinds of accelerators working on different tasks and assessing their performance as part of a complete system. In this work, we focus on a specific machine learning models called neural networks. Neural networks have proven to be successful and gained popularity for solving problems related to computer vision, such as object detection and recognition. They are also quite successful in voice recognition which is widely deployed in a lot of services such as Google Assistant and Amazon's Alexa. Neural networks are very large and data-intensive that require dedicated hardware to perform its task efficiently. Neural Network accelerators have gained immense popularity as different industries have deployed them as part of their computing infrastructures such as Google's TPU[5] for their data center workloads, as well as mobile platforms developed by Huawei, Qualcomm, and Mediatek[4]. In this work we try to integrate a Neural Network Simulator (DNN-SIM) with a commonly used industrial/research system simulator Gem5[3], we also provide a software layer interface using google NNAPIs[1] to allow users to write their models and analyze the system and accelerator performance on Gem5-DNNSIM simulator.

[1]The source code is available here: `https://bitbucket.org/deval281shah/eece571t_new/src/master/`

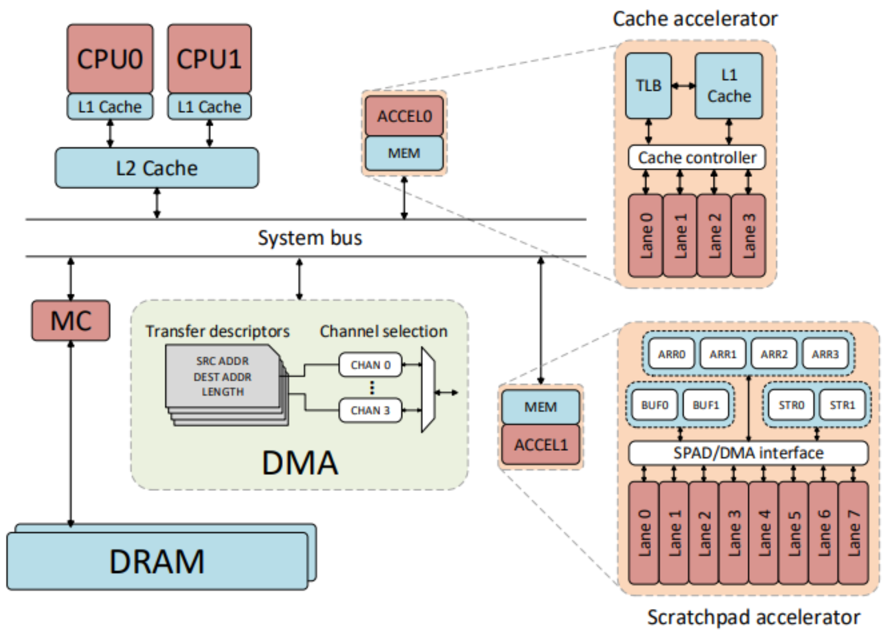

Figure 1: An example SoC that can be modeled using gem5-Aladdin[8]

2 Background

2.1 Gem5-Aladdin

Gem5-aladdin [8] is a system simulator intended to study co-design of accelerators in SoCs, it models the dynamic interactions between accelerators and the SoC platform. Accelerators are often designed as standalone hardware blocks that communicates with the rest of the system using Direct Memory Access (DMA) interface. However, the tasks of moving data around and launching kernels to execute on these accelerators are handled by software drivers, which makes it hard to quantify the overhead of these tasks compared to the accelerator design. For such reason, gem5-aladdin is a good tool to analyze and study the trade-offs involved in the accelerator design and its effect on the entire system. Gem5-aladdin allows users to integrate any accelerator to be connected to the system bus and uses a memory mapped interface to allow the processor to control this IP.
As seen in figure 1, different accelerators can be connected to the system bus in gem5 and managed by the processor or the Direct Memory Access (DMA) to copy data from or to the memory.

2.2 DNNsim

DNNsim is a cycle-level simulator that implements a collection of existing DNN accelerators including Stripes[6], SCNN [7], and Laconic [9] and etc. The simulated accelerators exploit different aspects of value property of DNN models including value sparsity, bit sparsity, and reduced precision requirement. In order to support exploiting value property, DNNsim uses run-time value trace as input. This enables accurate simulation of overhead introduced by the aforementioned value properties such as number of bank conflicts in SCNN accumulator arrays. However, DNNsim lacks support for modeling data movement in the memory hierarchy. So, to better understand the impact of memory hierarchy and more broadly system-level impact, we choose to integrate DNNsim with Gem5.

2.3 NNAPI

Android Neural Networks API [1] is an API interface designed to aid developers in integrating neural nets in their application. NNAPI provides the capability to construct neural net models and using them for inference during execution of an android application. NNAPI is implemented in C and

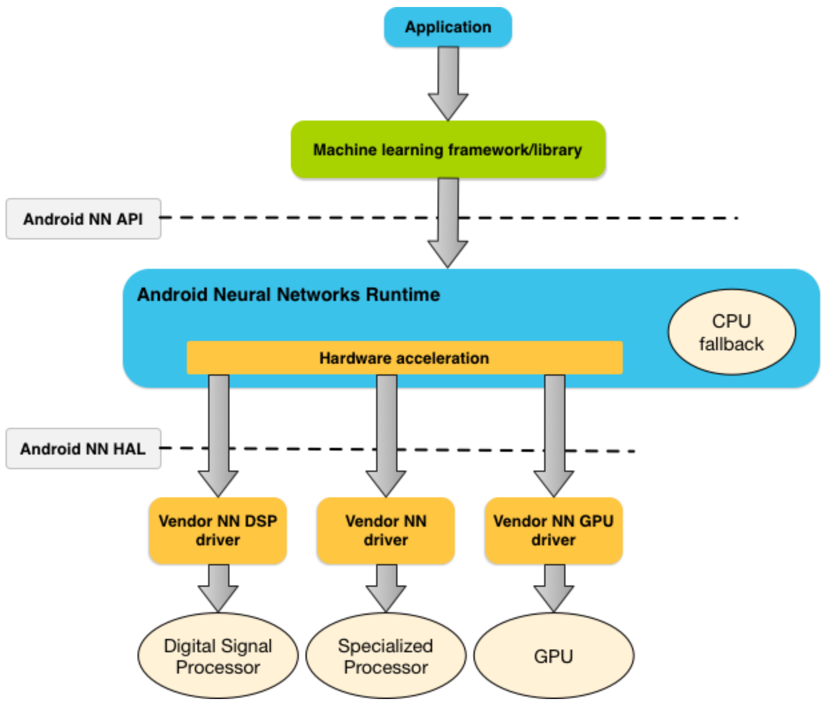

Figure 2: Architecture of NNAPI [1].

provides a low level functionality, and is intended to be is used for constructing high level frameworks, which in-turn can be used by android application developers directly in their application. NNAPI thus acts as a conduit between the Android Neural Network Run-time and machine learning frameworks. Figure 2 shows the architecture of NNAPI system. Commonly, the machine learning frameworks use java native interface to access the NNAPI functionality written in C, in their java based environment. NNAPI is dependent on android Neural Network run-time, which communicates with the application and different on-board processors. It communicates the capabilities of hardware to the application, and distributes computations in the application model to different processors that are best suited for each operation.

There are three main components of NNAPI:

- Model
- Compilation
- Execution

Figure 3 shows the basic programming workflow of NNAPI. A *Model* constructed by an application, is composed of operands and operations that are applied on them, which can be added to the model according its architecture. The operands can be scalar as well as multi-dimensional. Weights and biases from trained model can be used to set the value of operands in the model. When the construction of model is complete it is compiled to create a *Compilation*, which represents a configuration about the compilation of the model into lower level code for different processors chosen for various operations. Same model can have different compilations due to different performance and energy requirements chosen when the model is constructed. After the compilation of model it can be applied, both synchronously and asynchronously to a set inputs to get the outputs, by creating an *Execution* for those inputs. Another important abstraction used by NNAPI is *Memory*, which is represented by memory buffers. The buffers are used for storing the value of operands needed for a model, reading files for getting training data for model and storing the inputs and outputs of different executions.

3 Methodology

Gem5 has three different mode of operations:

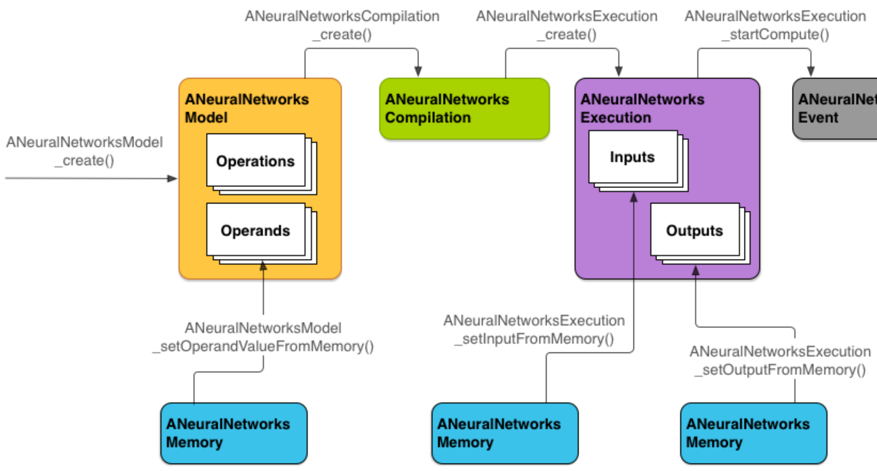
ANeuralNetworksCompilation _create()
ANeuralNetworksExecution _create()
ANeuralNetworksExecution _startCompute()
ANeuralNetworksModel _create()
ANeuralNetworks Model
Operations
Operands
ANeuralNetworks Compilation
ANeuralNetworks Execution
Inputs
Outputs
ANeuralNetworks Event
ANeuralNetworksModel _setOperandValueFromMemory()
ANeuralNetworksExecution _setInputFromMemory()
ANeuralNetworksExecution _setOutputFromMemory()
ANeuralNetworks Memory
ANeuralNetworks Memory
ANeuralNetworks Memory

Figure 3: Flow of NNAPI [1].

1. Full-system: in this mode gem5 models a full system including its operating system, allowing users to emulate a full-system with software and hardware layers.
2. standalone mode: in this mode gem5 models a master DMA connecting to other slaves on a bus interface. Input is fed to modules through trace files (acting as I/Os in hardware).
3. sysemul mode: in this model gem5 models system calls that allows users to write programs to execute on hardware without modeling operating system effects.

Gem5-alaadin uses sysemul mode since its interest to analyze the effect of running a program launching on the accelerator and only modeling the context switching between program execution on the processor and the accelerator.
In our system, users can write their models using NNAPIs which gets translated to native system calls, these system calls gets translated to commands to copy data to the accelerator and start its operation and return back once done.

4 Implementation

4.1 NNAPI

The source code of NNAPI is available on **. However, this source code is a part of higher level system and has many dependencies for successful build. As our main focus is to analyze machine learning workload using NNAPI, we implemented custom NNAPI source code, the original source code and the documentation is used to understand the functionality of each API call.

As mentioned in section 2.3, there are Five main components of NNAPI which are represented as classes. An overview about the implementation of each component is given below.

- ANeuralNetworksModel:
  This class contains the information about the mathematical operations which defines the model. Once a ANeuralNetworksModel is created for a network, network architecture is defined by adding operands (e.g. weights and activation) and operations such as conv, relu, pooling and so on. Once the model is defined, the constant network parameters such as weights and bias which will not be modified across different inference instances, can be set. Input and Output operands are specified for ANeuralNetworksModel.
- ANeuralNetworksCompilation:
  The class ANeuralNetworksCompilation creates one instance of compilation. When a model is passed along with the compilation instance, it performs the task of work distribution. If there are more then one devices connected such as NN accelerator and GPU and no preference for work partition is given, compilation step will try to partition the work

amongst multiple devices according to the performance attributes set by the user and the capabilities of the devices.

- ANeuralNetworksMemory:
  The class ANeuralNetworksMemory is used to represent shared memory, memory mapped files or special hardware buffers. We have used shared memory for our purpose. It can be easily extended to support Memory mapped files and special hardware buffers.

- ANeuralNetworksExecution
  The class ANeuralNetworksExecution represents the Execution instance of given Model. DMA transfer of operands (weight parameters) is initiated on the creation of this class. Also, the Input Operands are set.
- ANeuralNetworksEvent:
  The class ANeuralNetworksEvent represent one instance of inference. Start compute commands initiated DMA transfer pf Input operands to the accelerator and initiates an ioctl call to the accelerator.

In order to communicate the the network architecture with the accelerator, a device driver can be added. Our main focus was on integration of single accelerator and hence no division of the work. The division of the work can be added in the compilation step.

Similarly, once the the execution plan is communicated with the accelerator, DMA access requests should be generated by the accelerator once execution starts.

4.2 Modifications to DNN accelerator

We choose the SCNN simulator to integrate as a case study. We wrap core components in DNNsim that related to executing a workload on SCNN into gem5. We register this as an event and start to launch this event after DMA transfer has completed. We provide the offline value trace to DNNsim to evaluate cycles required for executing this workload, then we accumulate the global cycle counts and initiate results offloading back to system main memory. We can repeat the same process until we finish processing the entire network.

4.3 Modifications to Gem5-Alaadin

We use gem5-alaadin as our base for our simulator. Similarly, we use system call `ioctl` to invoke the accelerator whenever it is called from the user program. We modified gem5-aladdin to invoke our DNN accelerator instead of Alaadin accelerator, we use the same memory mapped interface of Alaadin, however, we have less commands (memory registers) to program the accelerator.
When we invoke DNN-sim, data is copied to the accelerator through `DmaLoad` command, and configuration parameters of DNN-sim are set.
Gem5-aladdin profiles the dynamic execution of a program and constructs a dynamic data dependence graph (DDDG) as a dataflow representation of an accelerator, which is generic to any kind of accelerator, however, our case define a specefic accelerator with fixed computations. Hence, our design is simpler and doesn't include DDDG.

5 Experimental Results

In this section, we show proof of concept results for invocation of DNNSim by gem5 through NNAPI (software layer). We use a system with the following configurations:

Architecture	x86
number of CPUs	1
System clock	1Ghz
Memory controller	DDR3
Memory size	8GB
Cache line size	64

We run gem5 in system emulation mode which executes a pre-compiled program binary, this program has NNAPI calls to construct a neural network and invoke **ioctl()** call which instructs the CPU to execute this compute kernel on the accelerator. We fix the accelerator architecture to SCNN[7] with different configurations for its MAC units. In this experiment, we run different hardware configurations for the accelerator micro-architecture and report the total system cycles it take to run different network architectures as seen in the following table:

Network	**I**	**F**	**Ht**	**Wt**	**Accel cycles**	**total cycles**
Alexnet	4	4	4	4	40886942	41026565000
Alexnet	8	8	4	4	17679144	17818765000
GoogleNet	8	8	4	4	3047620	3187238000
VGGNet	8	8	4	4	31198300	31337905000

The table above shows different configurations for SCNN accelerator that runs on gem5 reporting the number of cycles taken by DNNSim to compute the workload of different networks, we also report the total number of cycles which is the time taken for the system to run the program, invoke the accelerator and schedule the computations. The accelerator parameters are row multipliers per PE(F), column multipliers per PE (F), number of PE columns (Wt) and number of PE rows (Ht).

6 Discussion/Future work

The integration of DNN-sim with Gem5 is not complete and there are many limitations of the current implementation.

6.1 Future work for system changes

Currently, gem5 supports one mode of DNNSim with a certain architecture. Due to the difference in interface between gem5 and DNNSim, we implemented a simple network interface with fixed parameters, however, the system should have supported features for different network architectures and structures that can be configured through the software layer. Current version of DNNSim does not model DRAM memory interaction or memory ports and hence does not model memory transfer cycles from system memory to accelerator memory. Scratchpad memory ports need to be added so that DMA transfer timing model can be added. This requires changes in DNNSim.

6.2 Future work for NNAPIs

We have implemented necessary NNAPIs to build, compile and execute a model and can be easily expanded. We have added necessary comments in the implemented code for these extensions.

NNAPIs currently doesn't flexibly model all kinds of networks since we hard-coded one network that we use for prototyping, future versions should extend this model to support other kinds of networks. This needs conversion of NNAPI model to DNNSim compatible format. This can also be achieved by usage of a framework compatible with NNAPI and DNNSim both. For example, DNNSim supportes a Caffe model, and there are open-source implementations availabale to translate the Caffe model to NNAPI [2].

The implementation assumes only one accelerator in the system and runs the entire model on the accelerator. A more sophisticated approach is needed to run parts of the model on different accelerator and is left to future work.

References

[1] https://developer.android.com/ndk/guides/neuralnetworks.

[2] https://github.com/microsoft/mmdnn.

[3] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011.

[4] Andrey Ignatov, Radu Timofte, William Chou, Ke Wang, Max Wu, Tim Hartley, and Luc Van Gool. AI benchmark: Running deep neural networks on android smartphones. *CoRR*, abs/1810.01109, 2018.

[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, 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, and Jonathan Ross. In-datacenter performance analysis of a tensor processing unit. 2017.

[6] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor M Aamodt, and Andreas Moshovos. Stripes: Bit-serial deep neural network computing. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12. IEEE, 2016.

[7] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W. Keckler, and William J. Dally. Scnn: An accelerator for compressed-sparse convolutional neural networks. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, pages 27–40, New York, NY, USA, 2017. ACM.

[8] Y. S. Shao, S. L. Xi, V. Srinivasan, G. Wei, and D. Brooks. Co-designing accelerators and soc interfaces using gem5-aladdin. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12, Oct 2016.

[9] Sayeh Sharify, Alberto Delmas Lascorz, Mostafa Mahmoud, Milos Nikolic, Kevin Siu, Dylan Malone Stuart, Zissis Poulos, and Andreas Moshovos. Laconic deep learning inference acceleration. In *Proceedings of the 46th International Symposium on Computer Architecture*, pages 304–317. ACM, 2019.