Neural Networks as Function Primitives:
Software/Hardware Support with X-FILES/DANA

Schuyler Eldridge[1] (schuye@bu.edu) Tommy Unger[2] Marcia Sahaya Louis[1]
Amos Waterland[3] Margo Seltzer[3] Jonathan Appavoo[2] Ajay Joshi[1]

[1]Boston University Department of Electrical and Computer Engineering
[2]Boston University Department of Computer Science
[3]Harvard University School of Engineering and Applied Sciences

Neural Networks are "Hot Hot Hot!!!"

- Neural networks are layered structures consisting of neurons useful for:
  - Classification
  - Regression
  - Problems with difficult to discern solutions
- They have broad use in high tech and big data, e.g., Google [1]
- They also enable alternative computing methodologies
  - Automatic parallelization, e.g., ASC [2]
  - Approximate computing, e.g., NPU [3]

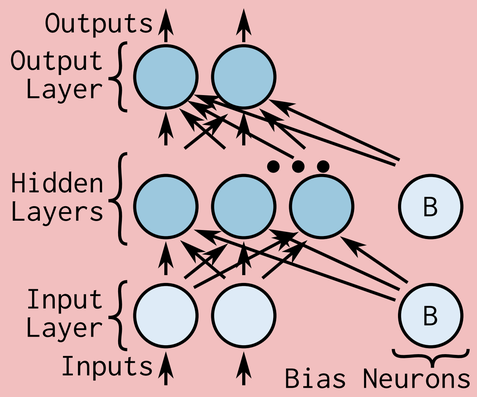

Figure 1: An example neural network

[1] Google TensorFlow, https://github.com/tensorflow/tensorflow
[2] A. Waterland, E. Angelino *et al.*, "Asc: Automatically scalable computation," in *Proc. ASPLOS*, 2014.
[3] H. Esmaeilzadeh, A. Sampson *et al.*, "Neural acceleration for general-purpose approximate programs," in *Proc. MICRO*, 2012.

Our Vision of General-Purpose Neural Network Computing

1. Treat neural networks as a *functional primitive* backed by hardware acceleration [1]
2. Follow a transaction model for neural network computation
3. Exploit anticipated sharing of neural networks across applications

- We think about neural acceleration like floating point acceleration
- We break this into two distinct contributions:
  - X-FILES – Software/Hardware extensions for transaction management
  - DANA – One possible backend accelerator that interfaces with the X-FILES

[1] S. Eldridge, A. Waterland *et al.*, "Towards general-purpose neural network computing," in *Proc. PACT*, 2015.

Safe Multi-Transaction Management using ASIDs

- Processes grouped into address spaces, identified with ASIDs (Figure 2 left)
- An ASID defines a set of processes that share neural network configurations, identified with NNIDs
- An OS-managed ASID–NNID Table enables NNID dereferencing (Figure 2 right)
- All transparently handled by the OS

X-FILES: Software/Hardware Extensions

- A defined user and supervisor interface for neural networks (Table 1)
- Uses ASIDs and NNIDs to allow access to shared neural networks
- Relies on a Hardware arbiter to maintain architectural state (Figure 3)
- Currently supports a subset of FANN [1]

[1] S. Nissen, "Implementation of a fast artificial neural network library (fann)," DIKU, Tech. Rep., 2003, http://fann.sf.net.

DANA: A Multi-Transaction Accelerator

- A backend accelerator aligning with our multi transaction vision (Figure 3)
- Understands a FANN derivative binary description of a neural network
- Implemented in Chisel [1]
- Supporting fixed point feedforward and gradient descent learning computations

[1] J. Bachrach, H. Vo *et al.*, "Chisel: Constructing hardware in a scala embedded language," in *Proc. DAC*.

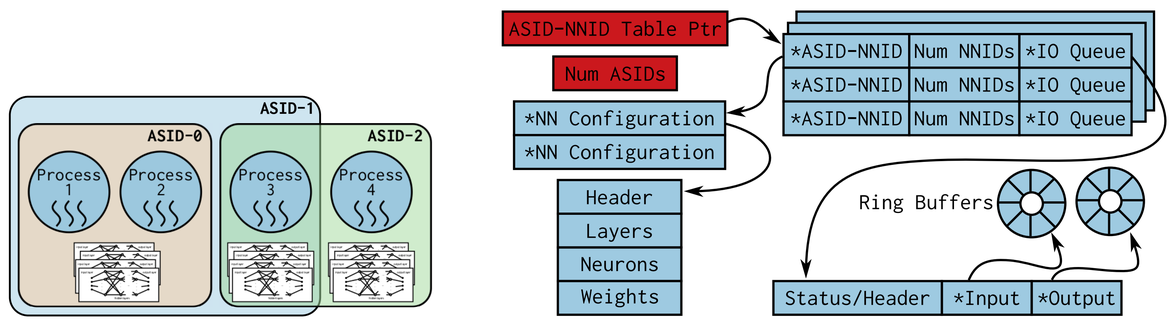

Figure 2: *Left*: Grouping of processes into address spaces. *Right*: An ASID–NNID Table for dereferencing neural network configurations from an ASID and an NNID.

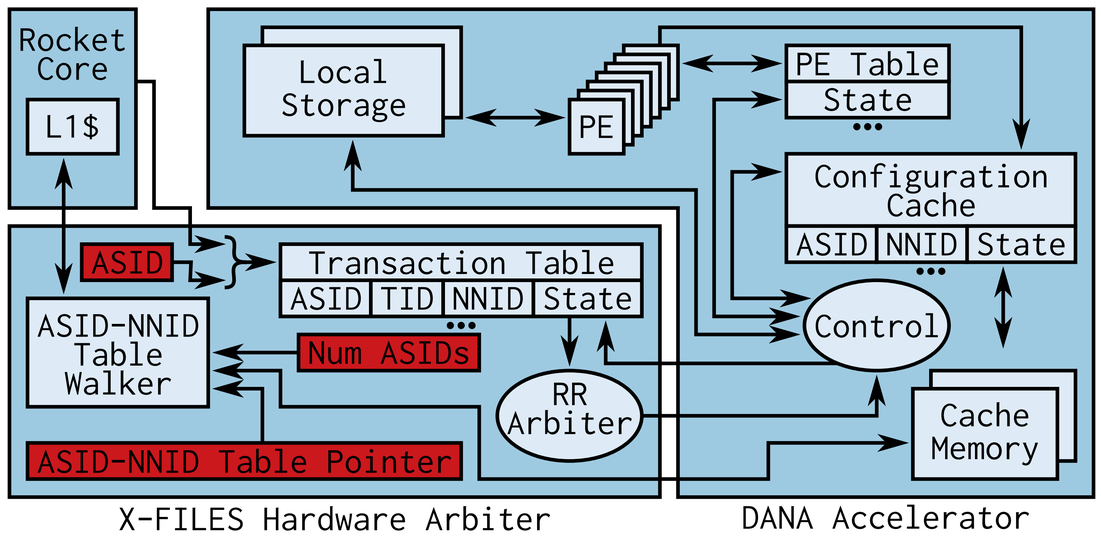

Figure 3: The X-FILES/DANA hardware architecture. Neural network transactions are managed by a hardware arbiter and executed on a backend accelerator.

Usage with a RISC-V Microprocessor

1. Grab a Rocket Chip RISC-V Microprocessor [1]
2. Build a RISC-V toolchain
3. Grab a copy of our X-FILES/DANA accelerator [2]
4. Build an FPGA configuration for Rocket + X-FILES/DANA
5. User processes can safely throw *transactions* at X-FILES
   - With support for feedforward and learning computation
   - Figure 4 shows three separate FPGAs all running Rocket + X-FILES/DANA

[1] Rocket Chip git repository, UC Berkeley, Online: github.com/ucb-bar/rocket-chip
[2] X-FILES/DANA git repository, Boston University, Online (*soon!*): github.com/bu-icsg/xfiles-dana

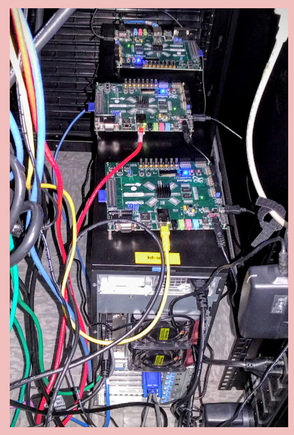

Figure 4: Three Rocket + X-FILES/DANA FPGAs attached to a server

Open Source Plans and Acknoledgments

- Remaining Items
  - Linux kernel integration
  - Support for asynchronous data transfer
- Open Source Availability
  - Should be ready by the end of February
  - On GitHub (*soon!*): github.com/bu-icsg/xfiles-dana
- This work was supported by the following:
  - A NASA Space Technology Research Fellowship
  - An NSF Graduate Research Fellowship
  - NSF CAREER awards
  - A Google Faculty Research Award

Table 1: X-FILES software library functions for communication with the X-FILES arbiter and managing the ASID–NNID Table

Function	User/Supervisor	Description
`tid = newWriteRequest(nnid, learningType, numOutputs)`	user	Initiate a new transaction returning a TID
`writeData(tid, *inputs, num_input)`	user	For register mode, write input data and, optionally, training data
`readDataSpinlock(tid, *output, num_output)`	user	For register mode, try to read output data until successful
`tidKill(tid)`	user	Kills an executing transaction
`oldTid = setAsid(asid)`	supervisor	Change the ASID returning the old TID to the OS for storage
`setAntp(*table)`	supervisor	Set the ASID–NNID Table Pointer and the number of ASIDs
`asidNnidTableCreate(**table, numAsid, numNnid)`	supervisor	ASID–NNID Table constructor
`asidNnidTableDestroy(**table)`	supervisor	ASID–NNID Table destructor
`nnid = addNnid(**table, asid, *nnConfiguration)`	supervisor	Adds an NN Configuration to an existing ASID–NNID Table
`removeNnid(**table, nnid)`	supervisor	Remove a specific NNID from an exsiting ASID–NNID Table