Bio-Inspired Vision Processor for Ultra-Fast Object Categorization

Clément Farabet

joint work with: Yann LeCun, Eugenio Culurciello, Berin Martini, Polina Akselrod, Selcuk Talay, Benoit Corda
NeuFlow
Synthetic Vision System

e-Lab
Clement Farabet
Berin Martini
Polina Akselrod
Selcuk Talay

Computational & Biological Learning Laboratory
NEW YORK UNIVERSITY

Yann LeCun [NYU] + Eugenio Culurciello [Yale]
neuroscientists have identified ~30 functional ‘modules’ in the cortex

[Felleman & Van Essen, 1991]
Convolutional Neural Nets

- they are hierarchical
- essentially feedforward
- each level is a transform that extracts or combines some features
Dataflow Computing?

the software written for the brain executes on several billion BPUs*, each connected to several thousands other BPUs

* Brain Processing Unit, more commonly known as neuron
the entire brain fabric is doing useful computation: if a synapse is assumed to perform a MAC* operation, then the brain computes ~20 petaOP/sec**

* Multiply and Accumulate = 2 OPs  ** 20 billion neurons, ~1000 connections per neuron
Dataflow Computing?

Brain (God)
- >20POP/s 20W
- 1e6GOP/sec/W

Tesla GPU (nVidia)
- 1TOP/s 200W
- 5GOP/sec/W
Dataflow Computing!

\(~1\text{e}^3\text{GOP/sec/W}\)
NeuFlow: 
A Dataflow Computer

- Processing units are kept simple/small with maximum throughput.
- These units are replicated over a large grid, locally connected.
- Flow control exists only at a very coarse level, and is not distributed over the grid.
- Data drives computations, ‘for’ loops don’t exist.
A Runtime Reconfigurable Dataflow Architecture

Control & Config

Off-chip Memory

Smart DMA

Configurable Route

Global Data Lines

Runtime Config Bus

(indication on the width)
Input Stream X

Internal storage ($\Delta = 1$ register)

Hard-wired logic in operator

Data Stream from/to global data lines

Local cache to preload up to 16 kernels

Convolution Operator

Convolved data + Y = Z

Pooling Subsampling

Input Stream Y

Output Stream Zs
Instructions Set

- compiling this Lua code:
  ```lua
  for i=1,100 do
    a[i] = b[i]
  end
  ```

- for a classical flow-control machine:
  ```
  SETI    REGD  12340
  SETI    REGE  3455
  READ    REGD  REGB
  WRITE   REGE  REGB
  INCR    REGD
  INCR    REGE
  COMPI   REGD  12440  REGC
  JUMPREL -5    REGC
  ```

- for our flow-based CPU:
  ```
  SETI    REGD  12340
  SETI    REGE  3455
  STREAM  REGD  REGE  100
  ```
luaFlow: A Dataflow Compiler

a specialized compiler that converts flow descriptions of algorithms to sequences of grid reconfiguration
luaFlow: A Dataflow Compiler/API

- The compiler is an API written in Lua, which allows full development in this easy-to-learn, fast prototyping language.

- It relies on Torch (an efficient N-dim array library*) to allow algorithms to be described as functional transforms, or flow-graphs of computations.

- The API is similar to CUDA in some ways, the developer first has to elaborate the code, e.g. build the code that will run on the platform, and then write the code for the host computer.

- The compiler iterates over the input flow and tries to merge as many nodes as possible, to minimize grid reconfigurations.

* Similar to NumPy
initializing neuFlow:
neuFlow = NeuFlow{mode='runtime'}

describing a neural net:
input_host = torch.Tensor(100,100)
net = nn.Sequential()
net:add(nn.SpatialConvolution(1,16,9,9))
net:add(nn.Tanh())
net:add(nn.SpatialLinear(16,4))

elaborating the code for neuFlow:
neuFlow:beginLoop('main')
  input_nf = neuFlow:copyFromHost(input_host)
  output_nf = neuFlow:compile(net, input_nf)
  output_host = neuFlow:copyToHost(output_nf)
neuFlow:endLoop('main')
loading the bytecode on neuFlow:
neuFlow:loadBytecode()
-- at this point, neuFlow executes its new code

now simply describe the host code:
while true do
    input_host = camera:getFrame()
    neuFlow:copyToDev(input_host)
    neuFlow:copyFromDev(output_host)
    result = soft_classifier:forward(output_host)
end

at this point the code is running in a loop, neuFlow is computing the neural net, while the host computes a simple linear classifier on the results
# Profiling*

<table>
<thead>
<tr>
<th></th>
<th>Intel 2Core</th>
<th>neuFlow Virtex4</th>
<th>neuFlow Virtex 6</th>
<th>nVidia GT335m</th>
<th>neuFlow65 nm</th>
<th>nVidia S1070</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peak GOP/sec</td>
<td>10?</td>
<td>40</td>
<td>160</td>
<td>182</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>Actual GOP/sec</td>
<td>1.1</td>
<td>37</td>
<td>147</td>
<td>54</td>
<td>928</td>
<td>290</td>
</tr>
<tr>
<td>FPS</td>
<td>1.4</td>
<td>46</td>
<td>182</td>
<td>67</td>
<td>1152</td>
<td>360</td>
</tr>
<tr>
<td>Power (W)</td>
<td>30</td>
<td>10</td>
<td>15</td>
<td>30</td>
<td>2</td>
<td>220</td>
</tr>
<tr>
<td>Embed? (GOP/s/W)</td>
<td>0.03667</td>
<td>3.7</td>
<td>9.8</td>
<td>1.8</td>
<td>464</td>
<td>1.31818</td>
</tr>
</tbody>
</table>

* computing a 16x10x10 filter bank over a 4x500x500 input image
neuFlow fits Anywhere

a home-made PCB that includes a Virtex4 and some quite large bandwidth to/from QDR memories
Thank You.