Dataflow Computing for General Purpose Vision

Clément Farabet

joint work with: Yann LeCun, Eugenio Culurciello, Berin Martini, Polina Akselrod, Selcuk Talay, Benoit Corda
General Purpose Vision?

the task of extracting information from an image to understand the underlying physical events/objects/scenes
as of today, the human brain is the best known hardware/software implementation*

* no published work really confirms that, but it’s widely acknowledged
The Visual Cortex

* neuroscientists have identified ~30 functional ‘modules’ in the cortex

* it represents 30% of the brain

[Felleman & Van Essen, 1991]
The Visual Cortex

and computer vision scientists have created/engineered thousands of models, to capture/reproduce the amazing features of the cortex

SIFT  ConvNets
HMAX  BoF
NN+SVM
Artificial Vision Systems

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

*unlike the visual cortex
a simple yet efficient model, currently extended in various ways:

✦ use of auto-encoders
✦ use of deeper architecture
✦ online learning methods
Convolutional Neural Nets
image processing algorithms can be expressed as flow-graphs with:

- **data nodes**: N-dim arrays
- **compute nodes**: arbitrary transforms $f()$: N-dim $\rightarrow$ P-dim
Dataflow Computing?

**Input Image**

1x500x500

**Normalized Image**

1x500x500

**Local Divisive Normalization**

**Convolutions w/ filter bank:**

20x7x7 kernels

**Pooling:**

20x4x4 kernels

**Convs:**

20x494x494

**S2:** 20x123x123

**C1:** 20x494x494

**Convs:**

100x7x7 kernels

**Pooling:**

20x4x4 kernels

**C3:** 20x117x117

**S4:** 20x29x29

**C5:** 20x23x23

**F6:** Nx23x23

**Convs:**

800x7x7 kernels

**Pooling:**

20x4x4 kernels

**Convs:**

100x7x7 kernels

**Convs:**

800x7x7 kernels

**Linear Classifier**

**Object Categories / Positions**

{ } at (x_i, y_i)

{ } at (x_j, y_j)

{ } at (x_k, y_k)

---

**total bandwidth requirement:**

✧ \#(data_node) \times \text{size(data_node[i])}

**total transform requirement (#OPs):**

✧ \#(compute_node) \times \#(compute_node[i].ops)
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
there is essentially no flow control, the information flows from a BPU to another as needed: no circuitry is wasted in fetching/caching/decoding/executing code ...
**Dataflow Computing?**

- 20PetaOP/s* for 20W
  ~
  1e6GOP/sec/W
  ~
  adult brain

- 1TeraOP/s for 200W
  ~
  5GOP/sec/W
  ~
  nVidia Tesla (S1070)

* 20 billion neurons, ~1000 connections per neuron
which is constrained by the cost of the silicon implementa-

tion. The number of MACs is only limited by the area of the device,

and can be optimized for passing one Tera-Op/s of performance, by using several grid

We designed the ASIC presented here with the goal of sur-

vesting a core size of 4.14 x 4.84 mm

output streams. 4 convolutions on a same input stream require

more than 4 filters on a given input: a sum of 4 convolutions

is typically not a problem, as common filter banks always apply

DMA was limited to 20, allowing the 12 convolvers to run

number of internal data lines connecting the PTs to the Smart

12 independent convolutions (24 streams). In this design, t

streamed in/out the PTs, which is enough to accommodate

memory. The bandwidth provided by the off-chip memory

Single data rate, resulting in a 256-bit bus for off-chip DDR

DMA is designed to handle a 512-bit internal memory bus of

specifications on the memory access unit. Hence the Smart

Encounter. The timing information provided from synthesis

verified by Synopsys VCS simulator through post-synthesis

the arrays were scaled up to 10 by 10 MACs, giving each

units operate at 500 MHz in the 65 nm process. In our system,

has been back annotated for the most accurate results.

As u m m a ry o f p e r f o r m a n c e o f t h i s d e s i g n i s g i v e n i n T a b l e I .

Maintaining the throughput for all PTs forces stringent

complex scheduling is required to dispatch multiple stream

DSP blocks (Xilinx’s multiply-and-accumulate units), mor

6, our system exploits 70% of the device, using about 400

FPGAs, using different sets of parameters: for a Virtex 4, a

inputs, each convolved with a separate kernel, and the resul

operator: a filter bank of 16

The ASIC implementation in an IBM 65 nm process re-

As u m m a ry o f p e r f o r m a n c e o f t h i s d e s i g n i s g i v e n i n T a b l e I .

Our architecture has been implemented on two different

numbers in Table II are not simple extrapolations.

by our custom compiler, and we used it to simulate results for

of data on the entire system. The scheduling is done statical

numbers in Table II are not simple extrapolations.

The ASIC was designed for an IBM 65 nanometer logic pro-

Best results are obtained when one MAC computes on a single data stream. A filter bank of 16

has been back annotated for the most accurate results.

The ASIC was designed for an IBM 65 nanometer logic pro-

Table II shows measured profiling results for a typical visio

the ASIC implementation of this architecture, therefore th

by our custom compiler, and we used it to simulate results for

scheduling is required to dispatch multiple stream

DSP blocks (Xilinx’s multiply-and-accumulate units), mor

6, our system exploits 70% of the device, using about 400

FPGAs, using different sets of parameters: for a Virtex 4, a

inputs, each convolved with a separate kernel, and the resul

operator: a filter bank of 16

The ASIC implementation in an IBM 65 nm process re-

As u m m a ry o f p e r f o r m a n c e o f t h i s d e s i g n i s g i v e n i n T a b l e I .

Best results are obtained when one MAC computes on a single data stream. A filter bank of 16
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

- Configurable Route
- Global Data Lines
- Runtime Config Bus

( indication on the width)

Off-chip Memory

Smart DMA

64bit CPU
Smart DMA
Configurable Route
Active Route
Active Data Lines
Off-chip Memory
Smart DMA
Configurable Route
Active Route
Active Data Lines
Off-chip Memory
Off-chip Memory

Configurable Route  Active Route  Active Data Lines
Smart DMA.

- Off-chip Memory
- Phy Mem Ctrl
- Mux / Arbiter
- Configurable Route
- Global Data Lines
- Runtime Config Bus

Rd / Wr.
Address Logic
User Space

Rd / Wr.
Address Logic
User Space

Rd / Wr.
Address Logic
User Space

...
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 home-grown compiler that converts flow descriptions of algorithms to sequences of grid reconfiguration.
luaFlow: A Dataflow Compiler

flow-graph model

\[\text{graph parsing} = \{\text{node reordering, node merging}\}\]

memory map & sequence of transforms

\[\text{device mapping} = \{\text{static scheduling, reconfiguration sequencing}\}\]

sequence of reconfigurations & memory transfers

\[\text{compilation} = \{\text{machine code generation}\}\]

binary code
luaFlow: A Dataflow Compiler

- **STEP**
  - Graph parsing = \{node reordering, node merging\}
  - Device mapping = \{static scheduling, reconfiguration sequencing\}
  - Compilation = \{machine code generation\}

- **MINIMIZING ?**
  - Overall bandwidth to/from off-chip memory
  - Number of grid reconfigurations
  - Binary size (for bandwidth)
luaFlow: The Dataflow API

- the user level is a simple 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

* 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))

e 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>neuFlow IBM 65nm</th>
<th>nVidia S1070</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Peak GOP/sec</strong></td>
<td>10?</td>
<td>40</td>
<td>160</td>
<td>182</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td><strong>Actual GOP/sec</strong></td>
<td>1.1</td>
<td>37</td>
<td>147</td>
<td>54</td>
<td>928</td>
<td>290</td>
</tr>
<tr>
<td><strong>FPS</strong></td>
<td>1.4</td>
<td>46</td>
<td>182</td>
<td>67</td>
<td>1152</td>
<td>360</td>
</tr>
<tr>
<td><strong>Power (W)</strong></td>
<td>30</td>
<td>10</td>
<td>15</td>
<td>30</td>
<td>2</td>
<td>220</td>
</tr>
<tr>
<td><strong>Embed? (GOP/s/W)</strong></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
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]
neuFlow fits Anywhere

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