Research update

Our approach to systematic co-processor (accelerator) construction for heterogeneous computing systems rests on two pillars:

1. Static (compile-time) analysis of resource space-time occupancy during computation.
2. Novel programming model with separate specification of Scheduling, Resources and Computation (SRC unbundling).

The main result, summarized in Table 1, confirms that on computations with irregular memory access patterns our approach delivers significant improvement in both latency and resource utilization, simultaneously.

The additional challenge for our current infrastructure is its scalability to support static analysis of immense computations that appear in standard neural networks.

The main opportunity arises from the observation that neural networks can be significantly pruned without loss of classification performance. The resulting network are sparse, hence they have irregular memory access patterns favored by our approach. Furthermore, pruning & compactification reduce the memory footprint and open the door for aggressive space-time aware schedulers to attempt optimization of latency and energy efficiency through careful planning of data movement. This will be the focus of our proposed work for the 2017 period.

Table 1: Latency and resources for linear system solver

<table>
<thead>
<tr>
<th></th>
<th>schedule cycle length</th>
<th>cycle frequency [MHz]</th>
<th>wall time [uS]</th>
<th>resources sqrt/div/mul/add</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM Cortex A9</td>
<td>N/A</td>
<td>667</td>
<td>15 (avg)</td>
<td>N/A</td>
</tr>
<tr>
<td>HLS @ FPGA</td>
<td>201</td>
<td>100</td>
<td>2 (always)</td>
<td>1/2/5/5</td>
</tr>
<tr>
<td>SRC Unbundling</td>
<td>193</td>
<td>100</td>
<td>2 (always)</td>
<td>1/1/2/2</td>
</tr>
</tbody>
</table>

Code contributions

We built a prototype compiler middleware to evaluate SRC unbundling approach. For productivity reasons, dynamic typing and access to Java eco-system, the prototype was developed in JVM hosted functional language, Clojure. The following libraries have been contributed to the BDD repository:
1. **bdd-konigsberg**
   Konigsberg is an extensible Directed Acyclic Graph (DAG) representation supporting arbitrary annotations on nodes/edges. This library is used as dataflow-like representation of numerical computation during static timing analysis and optimization. It is, currently, capable of efficiently manipulating graphs of up to 3 million nodes on standard JVM instances.

2. **bdd-morpheus**
   Morpheus is a library of computation kernels (e.g. Cholesky, LDLt, Matrix-Matri and Matrix-Vector multiply, etc.). The differentiating feature is that kernels matrix structures abstract over scalar implementations. This allows us to develop and test on floating point numbers, but later use fixed point or even graph representations without any change to kernels. Ability to extract computation after development & testing is the main problem with standard libraries like LAPACK, etc.

3. **bdd-edge**
   Edge implements Morpheus API on Konigsberg graphs, enabling extraction of dataflow graphs from Morpheus kernels.

4. **bdd-fsm**
   FSM is a simple helper library, similar in spirit to Caffe’s input specification language, used to improve performance of other libraries.

**Materials**

1. March 21st, 2016 meeting: [Agenda](#), [Video](#), [Presentation](#)
2. October 19th, 2016 meeting: [Agenda](#), [Video](#), [Presentation](#)

**Direction update and future work**

Through BDD engagements with industry sponsors we had an opportunity to discuss their interests and needs. We received a large number of inquires regarding our SRC unbundling programming model in the compiler middleware.

Two main learnings emerged from our engagements with the sponsors:

1. **Our approach to NN implementation**, based on SRC unbundling, is significantly different than what is currently available. This is perceived as a good thing as it provides an uncorrelated alternative to already ongoing efforts to build hand-coded or High Level Synthesis (HLS) based accelerators.

2. Interests and needs of industry sponsors are quite diverse with ongoing work that partially overlaps with our own. Hence, a number of sponsors expressed interest in specific details of our work, viewing the end-to-end NN-to-FPGA translation as a demonstration but not the end goal. Specifically, the possibility of bootstrapping compilation environment for in-house accelerators or obtaining network pruning & compactification framework from our toolchain received a lot of interest.

Further details on our direction update can be found in our continuation proposal.
BDD 2017: Continuing Research Proposal

Open-ended Programming Frontend for Fast Experimentation with Latency and Energy Footprint of Pruned Neural Networks

PI Prof. Vladimir Stojanovic / Co-PI Ranko Sredojevic

Based on the feedback we received in two research reviews and our understanding of interaction between our SRC unbundled programming model and DNN computation loads, we propose the following continuation of our research project:


   Training frameworks, e.g. Caffe, simplified research into DNN configurations by simplifying and speeding up network implementation and training. For the same reasons we expect a framework that enables easy and efficient experimentation with pruning and compactification strategies will improve our ability to find network configurations (in terms of hyper-parameters) and concrete implementations (trained coefficients) amenable to efficient hardware mapping through SRC unbundling or other approaches.

2. **(Long-term)** Continuing in the originally proposed direction by connecting the Open-ended Programming Frontend, proposed here, with the SRC unbundling middleware for static resource analysis we already developed.

   This will provide an end-to-end, NN specification, pruning and FPGA implementation environment. However, under this proposal the Open-ended nature of the frontend will allow others to add different backends that suit their research/product needs, using our FPGA backend as a reference implementation.

We suggest the same level of funding as in the original proposal, $100,000/year.

Both energy and latency cost of computation in modern computer architecture is dominated by data movement, Tables 2 and 3. Hence, the path to energy and latency efficient neural network implementations in embedded settings, including mobile processors and custom hardware, is through:

1. reduction of memory footprint of the network representation, and
2. careful planning of data movement during computation.

<table>
<thead>
<tr>
<th>Latency scenario</th>
<th>Wall time [ns]</th>
<th>Relative cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache fetch</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>Average instruction</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Branch misprediction</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>L2 cache fetch</td>
<td>7</td>
<td>14</td>
</tr>
<tr>
<td>DRAM fetch</td>
<td><strong>100</strong></td>
<td><strong>200</strong></td>
</tr>
</tbody>
</table>

Table 2: Computation latency cost in modern architectures

1 [http://norvig.com/21-days.html#answers](http://norvig.com/21-days.html#answers)
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>32 bit INT add</td>
<td>0.1</td>
<td>1</td>
</tr>
<tr>
<td>32 bit FP add</td>
<td>0.9</td>
<td>9</td>
</tr>
<tr>
<td>32 bit INT mul</td>
<td>3.1</td>
<td>31</td>
</tr>
<tr>
<td>32 bit FP mul</td>
<td>3.7</td>
<td>37</td>
</tr>
<tr>
<td>32 bit 32kB SRAM access</td>
<td>5</td>
<td>50</td>
</tr>
<tr>
<td>32 bit DRAM access</td>
<td>640</td>
<td>6400</td>
</tr>
</tbody>
</table>

Table 3: Computation energy cost in modern architectures

Recent publications demonstrate\(^3\) that significant pruning of network coefficients (setting them to 0) is possible with little or no loss, if more sophisticated pruning strategy is employed, in classification accuracy. Our own experiments, shown in Figure 1, on MNIST data set confirm this\(^4\). However, there seems to be no automated pruning framework and all published results seem to have done pruning by hand or in some ad-hoc manner.

![Figure 1: Sweeping the pruning threshold without re-training](image)

An open-ended pruning framework is a happy compromise between the needs of our SRC unbundled programming model research and the needs of sponsors working on their own hardware and in need of a domain specific compiler frontend, as well as those exploring memory bandwidth and energy footprint of accelerator implementations.

---


\(^4\) In collaboration with Shaoyi Cheng (UCB/BWRC) and Oscar Dorado (UCB)