RECODING THE CAFFE FRAMEWORK USING THE PERFORMANCE PORTABILITY PHILOSOPHY

Eduardo José Gómez-Hernández[1], Biagio Peccerillo[2], Sandro Bartolini[2], José Manuel García[1]

[1]Computer Engineering Department, University of Murcia, Spain
[2]Department of Information Engineering and Mathematical Sciences, University of Siena, Italy
{eduardojose.gomez,jmgarcia}@um.es, {peccerillo,bartolini}@dii.unisi.it

Motivation

We are entering a new era in computer architecture, the multi- and many-core era is not scaling as expected, and domain-specific architectures (DSAs) are achieving better efficiency. Each one of these DSAs has its own language (DSL). Programming each one of these architectures is hard, therefore an evolution in DSLs based in portability with low-performance losses is a priority. [Dean, Patterson and Young (2018)]

PHAST

- Single source programs for multiple devices is a development goal to be accomplished. Program once and execute anywhere. Easy portability across different parallel architectures is extremely important.
- This new approach recalls well-established, highly-expressive techniques.
- PHAST (Parallel Heterogeneous-Architecture STL-like Template library) is a modern C++ programming library based on the classic STL "containers", for the performance portability philosophy.

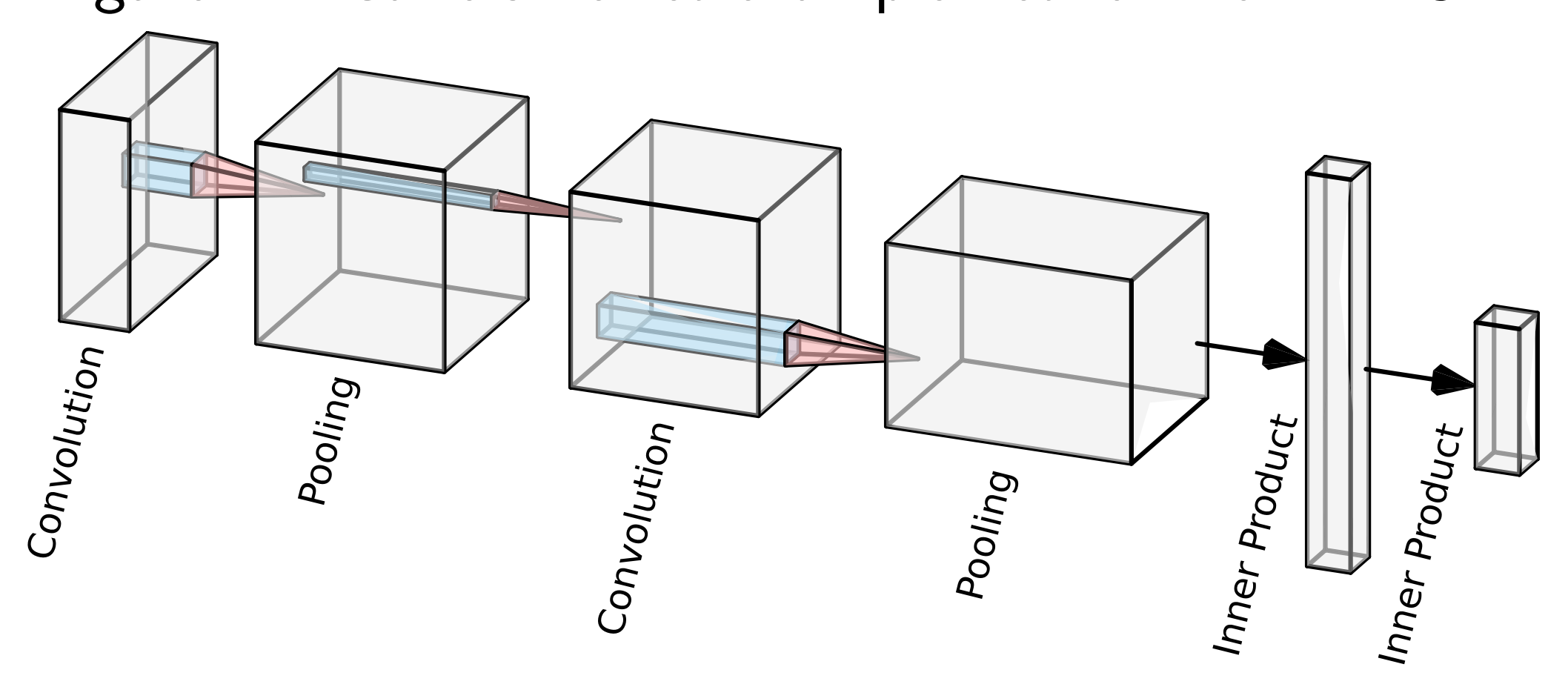
Figure 1 – PHAST Library Logo.

Source: Peccerillo and Bartolini (2019).

Deep Neural Networks & Caffe

- It seems that we are at the dawn of an era of computing systems where artificial intelligence is the key to everything.
- Deep learning has become more and more ubiquitous, specifically deep neural networks, being able to outperform human performance, in some tasks.
- Deep neural networks are complex to develop but also computer-intensive, requiring a lot of time to train. Therefore, frameworks like Tensorflow, MXNet, or Caffe provide an optimized environment for CPUs, GPUs, TPUs, FPGA, and so on.
- Caffe, developed by Berkeley, was the first one to arrive. Nowadays its development has finished but continues to be used, and it is an excellent tool for learning.

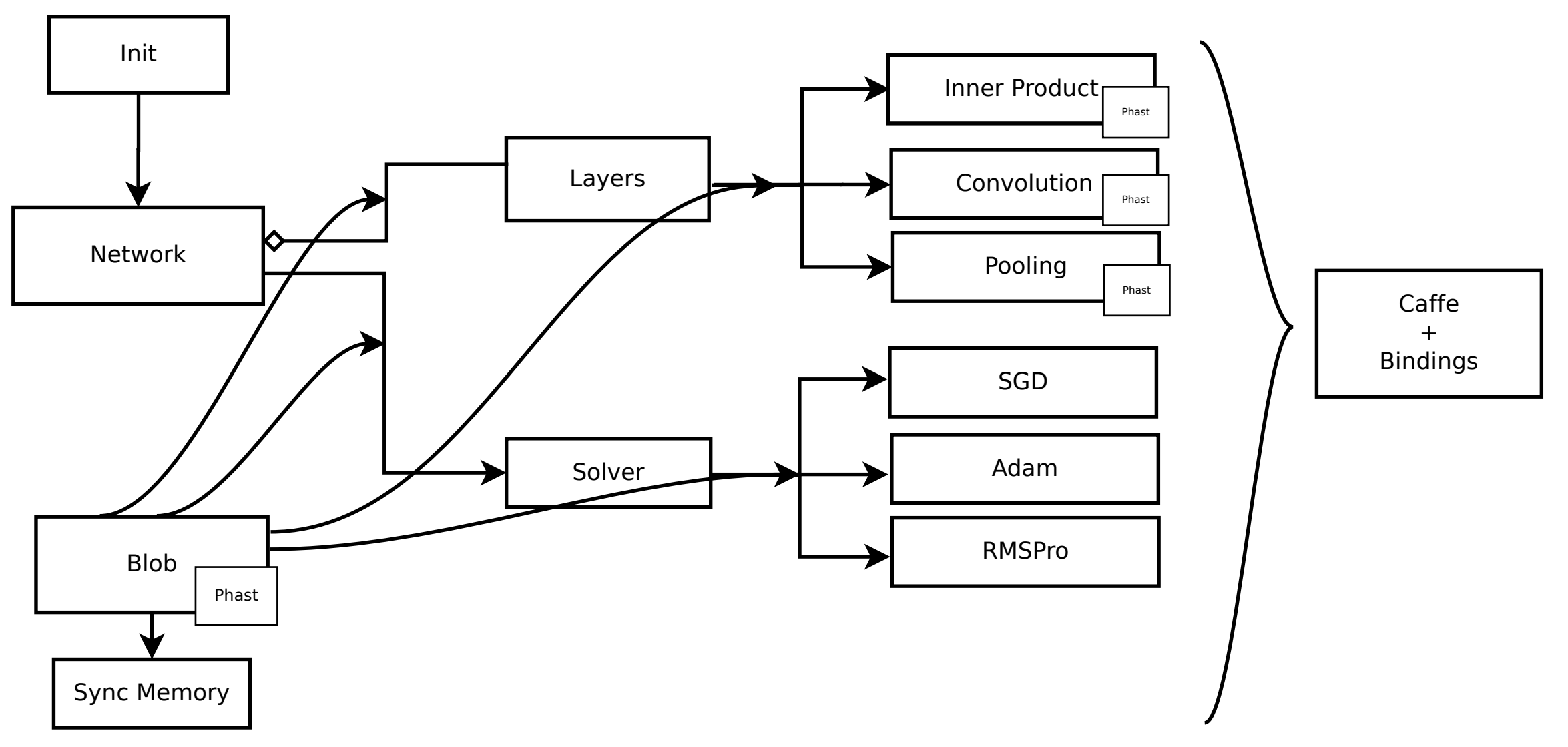
Figure 2 – Caffe's LeNet example network for MNIST.



Source: Jia and Shelhamer (2019).

Development

Figure 3 – Caffe framework schematic.



- Caffe is built from multiple modules.
- Blob is a container class and many others are the executors.
- To make the less number of changes as possible, we can add PHAST to the container and the executors.
- Until now, we have ported to PHAST the Blob and 7 layers with both feed- and back-forward.
- Each module that uses a Blob and is not modified to used PHAST, will force a copy to CPU memory, and then back to PHAST.
- We are looking to port other important layers, allowing us to execute more complex networks.
- Modify the solvers to avoid unnecessary copies between PHAST memory and Caffe memory.
- Each function that we need and is not implemented in PHAST, can be added using a functor, being reusable code to the whole program and easily portable to other applications.

Code Example

- Caffe original implementation of inner product for CPU and GPU as two separated source code files, each one has 8 and 12 lines of code respectively.
- The PHAST inner product version can be used for CPU and GPU, and it is only a file of about 10 lines of code.

```cpp
phast::matrix<float> matA = bottom[0]->getDataAsMatrix(M_, K_, false);
phast::matrix<float> matB = this->blobs_[0]->getDataAsMatrix(K_, N_,!transpose);
phast::matrix<float> matC = top[0]->getDataAsMatrix(M_, N_, false);
phast::dot_product(matA, matB, matC);
if (bias_term_) {
  matrixPlusVectorRows<float> matrixPlusVectorRows;
  matrixPlusVectorRows.vec.link(this->blobs_[1]->getDataAsVector(N_));
  phast::for_each(matC.begin_i(), matC.end_i(), matrixPlusVectorRows);
}
if(!transpose_) matB.transpose();
```

Listing 1: Phast InnerProduct Feed-Forward CPU/GPU/... implementation

```cpp
template <typename T, unsigned int policy = phast::get_default_policy()>
struct matrixPlusVectorRows : phast::functor::func_vec<T, policy> {
  _PHAST_METHOD matrixPlusVectorRows() {}
  _PHAST_METHOD void operator()(phast::functor::vector<T>& row) {
    for(auto r = row.begin(), i = vec.begin(); r != row.end(); ++r, ++i)
      *r += *i;
  }
  phast::functor::vector<T> vec;
};
```

Listing 2: matrixPlusVectorRows auxiliary function

Ultimate Goal

Reach the real modern code: one code for all platforms (optimized, portable and future-prof).

Acknowledgements

This work was partially supported by the Spanish MCIU and AEI, as well as European Commission FEDER funds, under grant RTI2018-098156-B-C53.

References

- DEAN, J.; PATTERSON, D.; YOUNG, C. A New Golden Age in Computer Architecture: Empowering the Machine-Learning Revolution. **IEEE Micro**, IEEE, v. 38, n. 2, p. 21–29, 2018.
- JIA, Y.; SHELHAMER, E. **Caffe | LeNet MNIST Tutorial**. [S.l.: s.n.]. Address: <https://caffe.berkeleyvision.org/gathered/examples/mnist.html>. Visited on: 22 May 2019.
- PECCERILLO, B.; BARTOLINI, S. **PHAST library**. [S.l.: s.n.]. Address: <https://www.phast-library.com/>. Visited on: 20 May 2019.