AWS AI Blog

Introducing NNVM Compiler: A New Open End-to-End Compiler for AI Frameworks

by Mu Li | on 06 OCT 2017 | in Apache MXNet On AWS | Permalink | Comments | Share

You can choose among multiple artificial intelligence (AI) frameworks to develop AI algorithms. You also have a choice of a wide range of hardware to train and deploy AI models. The diversity of frameworks and hardware is crucial to maintaining the health of the AI ecosystem. This diversity, however, also introduces several challenges to AI developers. This post briefly addresses these challenges and introduces a compiler solution that can help solve them.

Let's review the challenges first, introduce you to the UW and AWS research teams, and then walk you through how the compiler works.

Three challenges

First, it is nontrivial to switch from one AI framework to another because of differences among the frontend interfaces and the backend implementations. In addition, algorithm developers might use more than one framework as part of the development and delivery pipeline. At AWS we have customers who want to deploy their Caffe model on MXNet to

Search the Amazon AI blog

Search

Posts by Product

Amazon Lex
Amazon Machine Learning
Amazon Polly
Amazon Rekognition

Posts by Framework

Apache MXNet
TensorFlow
Caffe

from smartphone chips to data center GPUs. Take MXNet as an example. It has a portable C++ implementation built from scratch. It also ships with target dependent backend support like cuDNN for Nvidia GPU and MKLML for Intel CPUs. Guaranteeing that these different backends deliver consistent numerical results to users is challenging.

Last, chip vendors need to support multiple AI frameworks for every new chip they build. The workloads in each framework are represented and executed in unique ways, so even a single operation such as Convolution might need to be defined in different ways. Supporting multiple frameworks requires enormous engineering efforts.

Introducing the research team from UW and AWS

Diverse AI frameworks and hardware bring huge benefits to users, but it is very challenging to AI developers to deliver consistent results to end users. Luckily, we are not the first to face this kind of problems. Computer science has a long history of running various programming languages on different hardware. One key technology to solve this problem is the compiler. Motivated by the compiler technology, a group of researchers including Tianqi Chen, Thierry Moreau, Haichen Shen, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy from Paul G. Allen School of Computer Science & Engineering, University of Washington, together with Ziheng Jiang from the AWS AI team, introduced the TVM stack to simplify this problem.

Today, AWS is excited to announce, together with the research team from UW, an end-to-end compiler based on the TVM stack that compiles workloads directly from various deep learning frontends into optimized machine codes. Let’s take a look at the architecture.

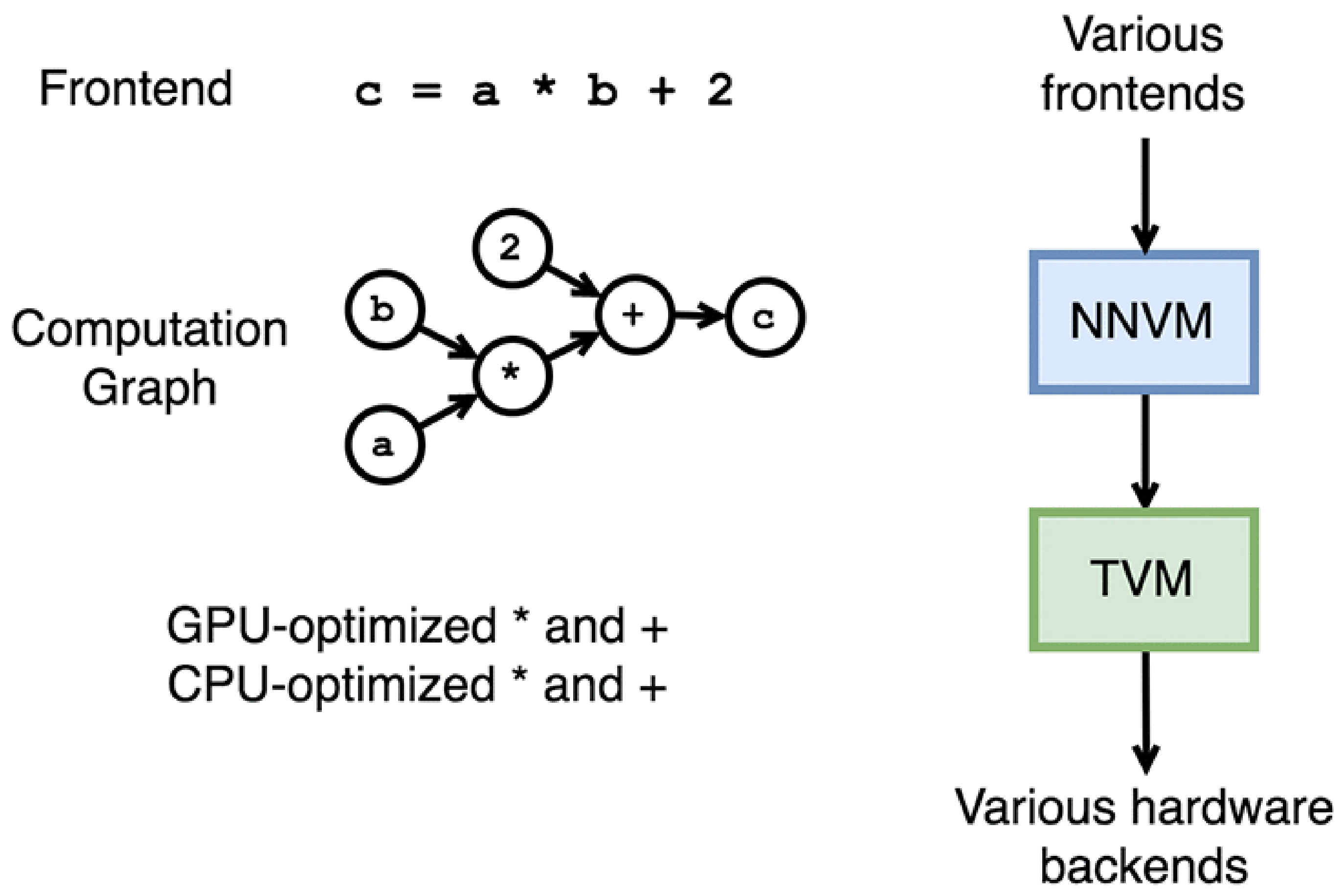
Architecture

Deep Learning Benchmark

Deep Learning CloudFormation Template

RSS Feed

Subscribe to this blog's feed



We observed that a typical AI framework can be roughly partitioned into three parts:

- The frontend exposed an easy-to-use interface to users.

for multiple hardware.

The new compiler, called the NNVM compiler, is based on two components in the TVM stack: NNVM for computation graphs and TVM for tensor operators.

NNVM – Computation graph intermediate representation (IR) stack

The goal of NNVM is to represent workloads from different frameworks into standardized computation graphs and then translate these high-level graphs into execution graphs. The computation graph, which is presented in a framework-agnostic format, is inspired from the layer definition in Keras and tensor operators from numpy.

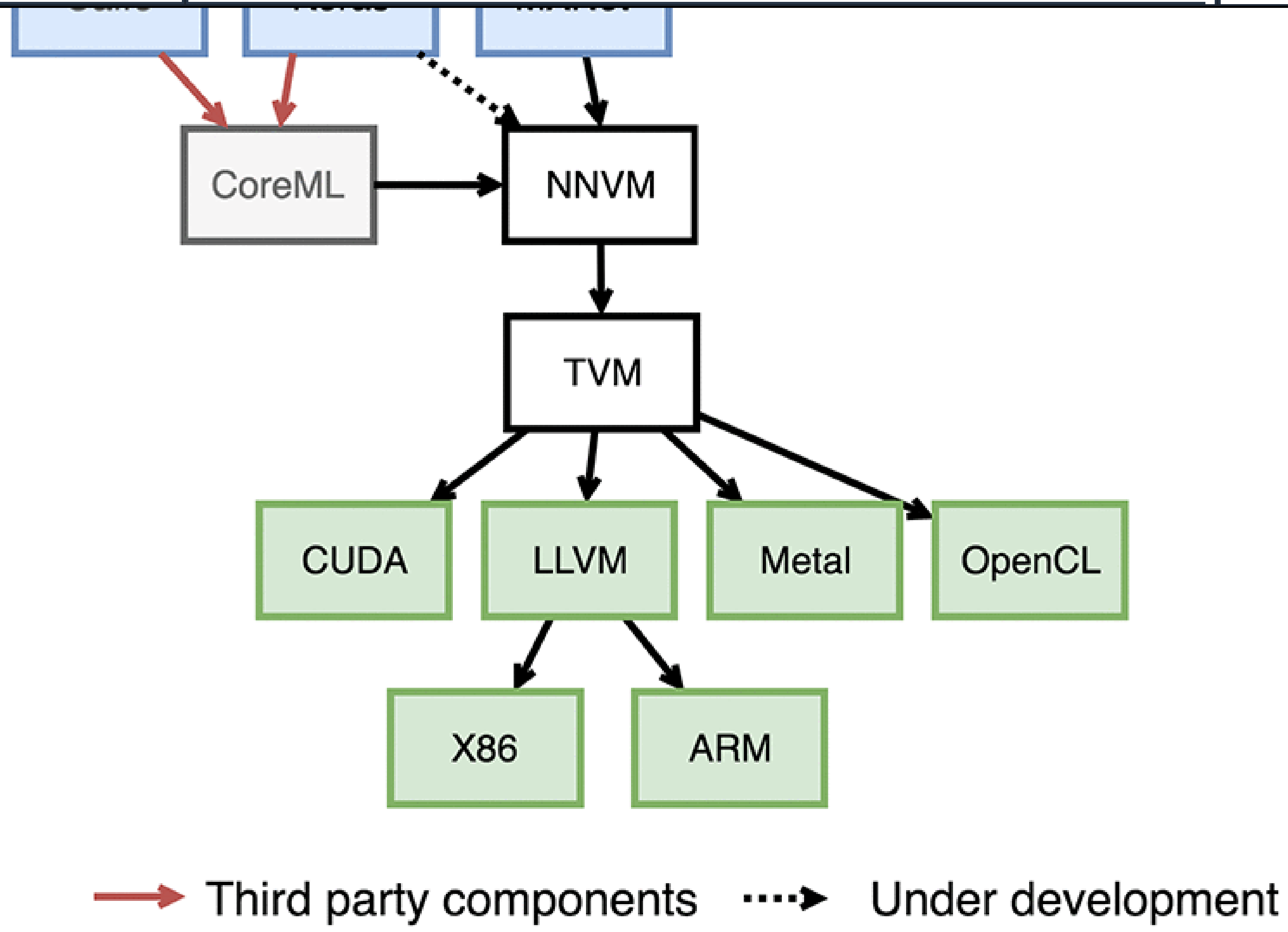
NNVM also ships with routines, called Pass by following the LLVM convention, to manipulate these graphs. These routines either add new attributes into the graph to execute them or modify graphs to improve efficiency.

TVM – Tensor IR stack

TVM, which originates from Halide, implements the operators used in computation graphs and optimizes them for target backend hardware. Unlike NNVM, it provides a hardware-independent, domain-specific language to simplify the operator implementation in the tensor index level. TVM also offers scheduling primitives, such as multi-threading, tiling, and caching, to optimize the computation to fully utilize the hardware resources. These schedules are hardware-dependent and can either be hand-coded or it is possible to search optimized schema automatically.

Supported frontend frameworks and backend hardware

The supported frontends and backends are illustrated in the following figure.



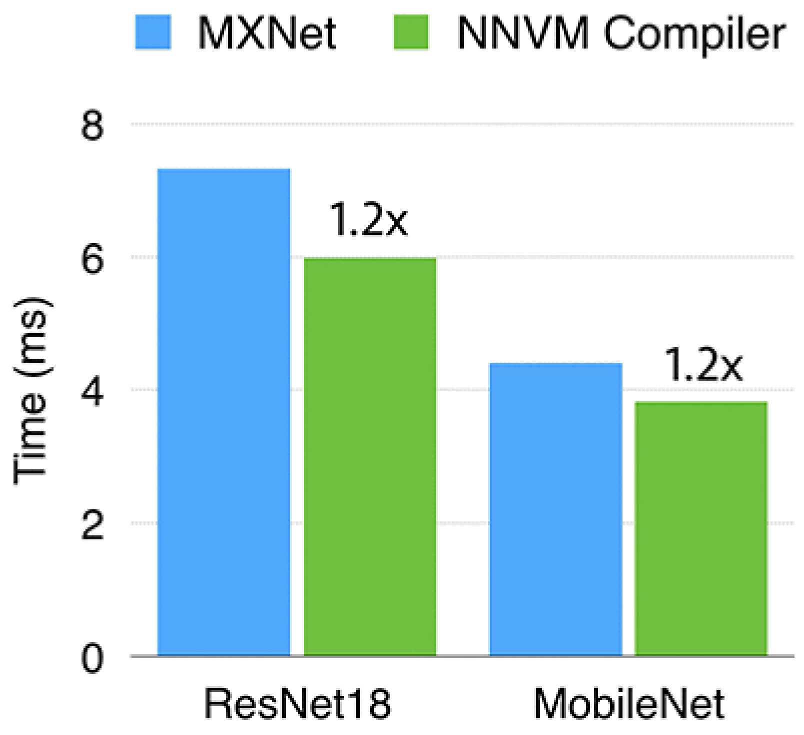
MXNet is supported by converting its computation graphs into NNVM graphs directly. Keras, still under development, is supported in a similar way. NNVM compiler can also take in model formats, such as CoreML. Therefore any framework that is able to use these formats can use this compiling stack as well.

defines both the computation graph and operator specification. To add new hardware, we can reuse the operator implementations with TVM and only need to specify how to schedule them efficiently.

Performance results

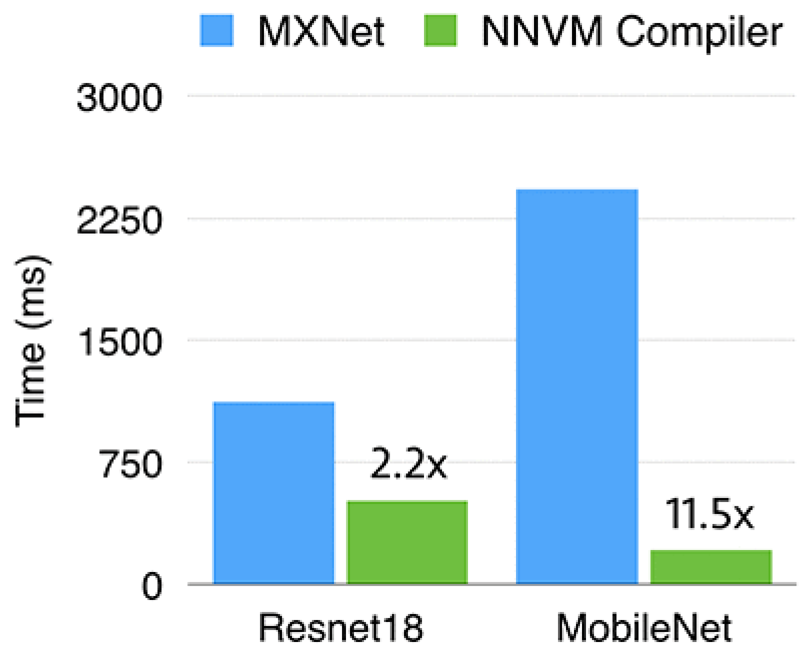
We show the NNVM compiler performance with MXNet as the frontend on two typical hardware configurations: ARM CPU on Raspberry PI and Nvidia GPU on AWS. Despite the radical architecture difference between these two chips, the only code difference is on the scheduling part.

Nvidia GPUs



kernels. As can be seen, NNVM compiler is slightly better (1.2x faster) than the cuDNN backend on both ResNet18 and MobileNet.

Raspberry PI 3b



The optimal schedules are picked through an auto tuner. In particular, we find the best schedule for each operator given the input shape by benchmarking its performance on a Raspberry Pi.

We compared NNVM compiler against MXNet. The MXNet is built with OpenBLAS and NNPACK enabled, and we manually turned on winograd convolution in NNPACK to get the best performance.

As can be seen, the NNVM compiler is 2.2x times faster on Resnet18. There is also an 11.5 times difference on MobileNet. This is mainly due to the fact that depthwise convolution is not optimized in MXNet (because of the lack of

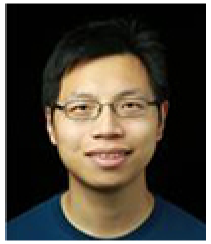
We introduce the NNVM compiler, which compiles a high-level computation graph into optimized machine codes. This compiler is based on two components in the TVM stack: NNVM provides a specification of the computation graph and operator with graph optimization routines, and operators are implemented and optimized for target hardware by using TVM. We demonstrated that with minimal effort this compiler can match and even outperform state-of-the-art performance on two radically different hardware: ARM CPU and Nvidia GPUs.

We hope the NNVM compiler can greatly simplify the design of new AI frontend frameworks and backend hardware, and help provide consistent results across various frontends and backends to users.

Additional Resources

- UW Announcement: Allen School and AWS team up on new NNVM compiler for deep learning frameworks
- NNVM on GitHub: https://github.com/dmlc/nnvm
- TVM on GitHub: https://github.com/dmlc/tvm/

About the Author



Mu Li is a Principal Scientist for AWS AI. He works with scientists and engineers to help our customers on deep learning projects, helping them shorten their time and cost when using AWS. In his spare time, he is a very busy new father.

TAGS: Apache MXNet

Menu Products Solutions Pricing Getting S My Account Create an AWS Account

Create a Free Account

Twitter Facebook Google+ Twitch AWS Blog What's New? RSS Subscribe to Updates

AWS & Cloud Computing

What is Cloud Computing?
What is Caching?
What is NoSQL?
What is DevOps?
Products & Services
Customer Success
Economics Center
Architecture Center
Security Center
What's New
Whitepapers
AWS Blog
Events
Sustainable Energy
Press Releases
AWS in the News
Analyst Reports
Legal

Solutions

Websites & Website Hosting

Business Applications

Backup & Recovery

Disaster Recovery

Data Archive

DevOps

Serverless Computing

Big Data

High Performance Computing

Mobile Services

Digital Marketing

Game Development

Digital Media

Government & Education

Health

Financial Services

Windows on AWS

Resources & Training

Developers

Java on AWS

JavaScript on AWS

Mobile on AWS

PHP on AWS

Python on AWS

Ruby on AWS

Windows & .NET on AWS

SDKs & Tools

AWS Marketplace

User Groups

Support Plans

Service Health Dashboard

Discussion Forums
FAQs
Documentation
Articles & Tutorials
Test Drives
AWS Business Builder

Manage Your Account

Management Console
Billing & Cost Management
Subscribe to Updates
Personal Information
Payment Method
AWS Identity & Access Management
Security Credentials
Request Service Limit Increases
Contact Us

Amazon Web Services is Hiring.

Amazon Web Services (AWS) is a dynamic, growing business unit within Amazon.com. We are currently hiring Software Development Engineers, Product Managers, Account Managers, Solutions Architects, Support Engineers, System Engineers, Designers and more. Visit our Careers page or our Developer-specific Careers page to learn more.

Amazon Web Services is an Equal Opportunity Employer.

Language Deutsch | English | Español | Français | Italiano | Português | Русский | 日本語 | 한국어 | 中文 (简体) | 中文 (繁體)

Site Terms | Privacy