BuQuaTi

Blogs by BuQuaTi

A Practical Benchmark: AMD GPU with OpenCL and CPU on Google TensorFlow

> *Why*

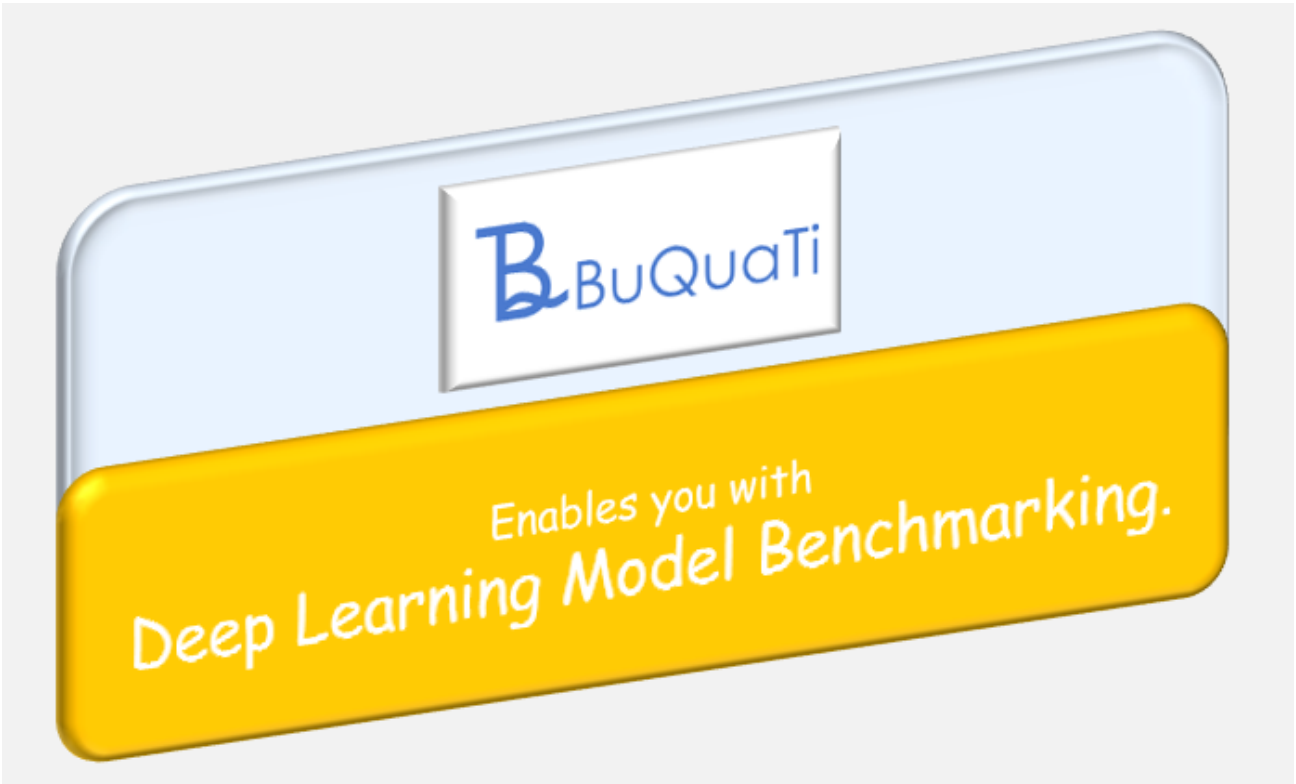
`Google TensorFlow` is an open-source, popular, and capable deep learning library.
It actually has 2 versions: `CPU` and `GPU`.
GPU version is much faster primarily due to GPU technology's large bandwidth, and parallel computation capability.
AND this is provided by `NVIDIA's CUDA` technology on `NVIDIA GPU`.
We should admire NVIDIA's vision here to have invested on this area since 2000s.
Today there's no official support for other GPU vendors.

There are a few initiatives to make `TensorFlow` work on `AMD GPU` with `OpenCL` technology.
RocM is the official one but is very restricted in terms of GPU compliance.
`Coriander` is an open-source initiative, and ComputeCC is another one.
Thanks to those people who make it happen, and freely share with others without hassle.

Not all users can afford high-end GPU.
In this blog I'll share with you the results of a simple, practical benchmark analysis of an ordinary `AMD GPU with OpenCL` using `Coriander` distribution of `Google TensorFlow`.

Here we go.

		AMD performance on TensorFlow		
		VM + CPU	CPU bare	GPU OpenCL
Basic	Linear Regression	real 0m12.106s user 0m8.420s sys 0m1.968s	real 0m9.494s user 0m14.052s sys 0m1.337s	real 4m12.193s user 5m30.658s sys 0m46.841s
Basic	Logistic Regression	real 0m38.686s user 0m42.245s sys 0m26.526s	real 0m52.047s user 1m9.149s sys 0m7.472s	real 11m5.592s user 15m56.675s sys 2m2.841s
Basic	Nearest Neighbor	real 0m4.186s user 0m5.508s sys 0m0.392s	real 0m9.565s user 0m10.424s sys 0m4.062s	real 0m6.805s user 0m7.442s sys 0m0.943s
Deep	Auto Encoder	real 0m22.263s user 0m33.744s sys 0m8.097s	real 0m22.582s user 0m52.931s sys 0m2.819s	real 1m27.800s user 0m47.659s sys 0m7.809s
Deep	Multilayer Perceptron	real 0m15.449s user 0m18.792s sys 0m9.667s	real 0m17.559s user 0m33.273s sys 0m1.554s	real 1m11.078s user 1m21.659s sys 0m11.796s
Deep	Recurrent Network	real 0m16.895s user 0m22.339s sys 0m4.998s	real 0m21.165s user 0m40.075s sys 0m2.175s	real 1m43.175s user 1m10.587s sys 0m16.273s
Deep	Dynamic RNN	real 0m25.781s user 0m32.050s sys 0m9.856s	real 0m21.104s user 0m34.279s sys 0m2.617s	real 3m4.922s user 2m35.687s sys 0m32.535s
Deep	Bidirectional RNN	real 0m18.724s user 0m24.626s sys 0m5.644s	real 0m21.842s user 0m40.931s sys 0m2.427s	real 1m49.824s user 1m14.711s sys 0m17.253s
Deep	Convolutional Network	real 0m34.015s user 0m54.220s sys 0m10.573s	real 0m38.090s user 1m54.241s sys 0m9.654s	real 9m37.408s user 35m21.498s sys 0m8.433s



Scenarios

(1) CPU and Virtual Machine
I5-6300U
RAM: 8GB
TensorFlow standard CPU (no tweak) with pip install
VM Host: Windows10
VM Guest: Linux SUSE

(2) AMD CPU A10-7850K, bare metal
RAM: 8Gb
TensorFlow standard CPU (no tweak) with pip install
Ubuntu 16.04TLS

(3) GPU AMD Radeon R9 380, bare metal
AMDGPU-PRO for OpenCL
RAM: 8Gb
GPU: AMD Radeon R9 380 2Gb
TensorFlow_cl by Coriander standard GPU (no tweak) with pip install .whl
Ubuntu 16.04TLS

Remarks

Installation is rather simple.
Sufficient to follow the instructions for
`AMD GPU PRO`,
`TensorFlow Coriander`,
and `Google TensorFlow`.

TensorFlow version of the Coriander is outdated. I had to upgrade the examples to run the same under current TensorFlow environment. Consider that TensorFlow has improved its performance too. So this is not only the difference between CPU and GPU with OpenCL, but also between old, and new TensorFlow.

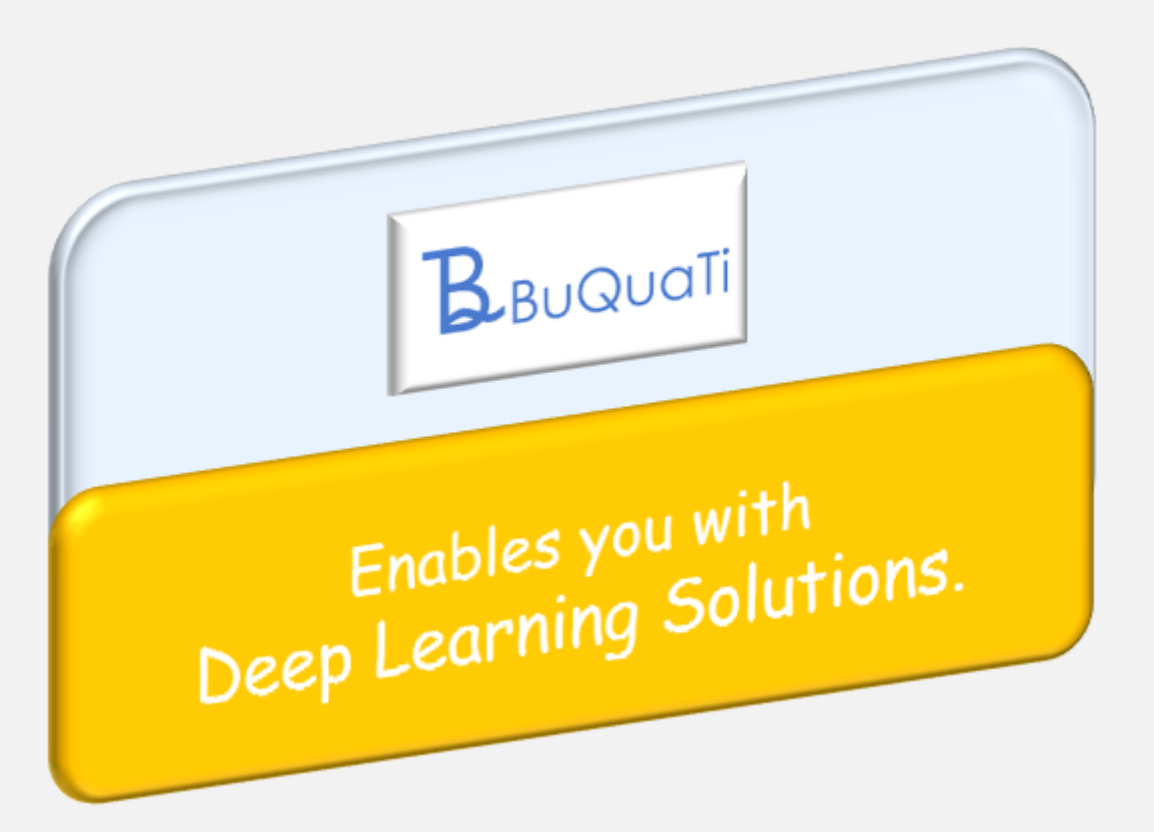
I couldn't make `Coriander` run `Google Object Detection API` (which is more recent), because of version incompatibility of libraries.

I couldn't make it run with multiple GPU devices neither.

Conclusion

How to say... really... you understood what I mean.
It's a pity for the entire ecosystem that we miss desperately competition or alternatives.

And we have no particular interest in any of the brands, companies or people mentioned in this article.



Appendix:Files

For the more curious of you, here are the plain text outcome of the tests.
Scenario(1):out, Scenario(1):times

Scenario(2):out, Scenario(2):times

Scenario(3):out, Scenario(3):times

Hakan Yerlikaya / April 4, 2018 / Artificial Intelligence, Back-End Development, EN, Google, Language, Solution Development, TensorFlow