Evaluacija biblioteka za matrične operacije na grafičkim procesorima

Tehnička dokumentacija

Verzija <1.0>

Studentski tim: Katarina Blažić
Marin Bralić
Nikola Bunjevac
Lucija Ivković
Krešimir Kralj
Stjepan Močilac
Pero Skoko
Borna Skukan

Nastavnik: Siniša Šegvić

Sadržaj

Tehnička dokumentacija

1. Uvod

U sklopu projekta evaluirali smo performanse izvođenja nekoliko okvira za duboko učenje i matrične operacije na pločici (System on chip) TX1. Taj proizvod, razvijen od strane NVidie, je reklamiran kao "prvo superračunalo na modulu", sa Cortex-A57 CPU i 256-jezgrenim Maxwell GPU. CPU baziran na ARM-ovoj arhitekturi i Maxwell Nvidijna tehnologija upućuju da je cilj TX1 modula bio proizvesti SoC niske potrošnje (po dokumentaciji i službenim forumima se vrte mjerenja od 1W dok je modul u praznom hodu, 8-10 W prilikom normalnog opterećenja, te 15W prilikom overclockanja sustava), pritom čuvajući solidne performanse. U sklopu projekta provjerili smo vrijede li te tvrdnje.

Sam modul, osim navedenih specifikacija, ima i radnu memoriju od 4 Gb DDR4, koja se pokazala i više nego dovoljnom, te interni spremnik od 16 Gb, dodatno povećan SD karticom veličine 64 Gb, wireless access point, kao i port za pristupanje preko Ethernet kabela. Što se tiče softverske podrške, na modulu je flashan Ubuntu Linux sustav, verzije 16.04, sa Linux for Tegra paketom drivera. Takav odabir operacijskog sustava, zajedno sa pojedinim detaljima arhitekture modula, su otežali pokretanje i testiranje nekih modula, kao i samo početno postavljanje uređaja, no o tome više u za to predviđenom dijelu.

Same usporedbe performansi su između TX1 GPU (1 TFLOP/s 256-core with NVIDIA Maxwell Architecture), TX1 CPU (64-bit ARM® A57 CPUs), GPU GTX 970 (raspoloživ na računalu *Nazgul*) te CPU i7 ili i5.

Prilikom flashanja TX1, obavezno host računalo mora biti Ubuntu verzije 14, inače flashanje neće biti uspješno provedeno. Također, flash mora biti proveden preko grafičkog sučelja, jer je NVIDIa jedino omogućila takav način flashanja.

2. Mjerenja performansi

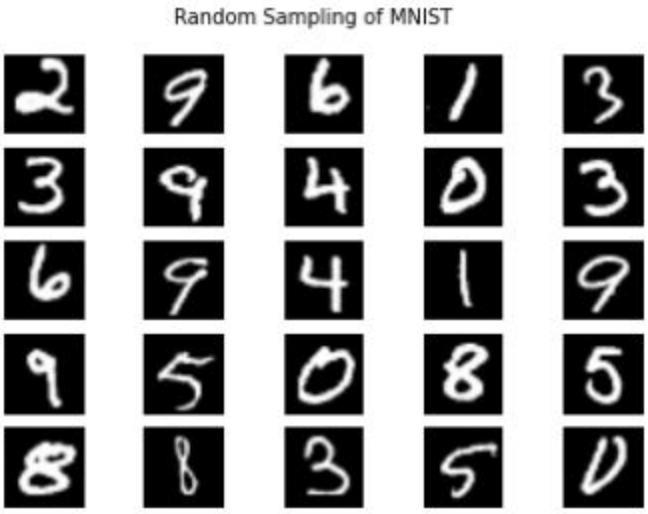
2.1 Tensorflow

Prva biblioteka koju smo testirali je Tensorflow. TensorFlow je biblioteka otvorenog koda za strojno učenje i matematičke izračune pomoću računskih grafova. Čvorovi računskog grafa predstavljaju matematičke operacije a bridovi grafa predstavljaju multidimenzionalne nizove podataka, nazvane tensori, koji se prenose između operacija. Fleksibilna arhitektura TensorFlowa omogućava obavljanje komputacija na jednom ili više CPU-ova ili GPU-ova. TensorFlow se trenutno koristi u razvoju i proizvodnji desetak Googleovih proizvoda, poput Gmail-a i Google photos.

Testiranje TensorFlowa

Kao projektni zadatak uspoređene su brzine izvođenja programa deep-net.py na NVIDIA Jetson TX-1 ugradbenom sustavu, GTX 970 stolnom GPU-u i dvama stolnim računalima.

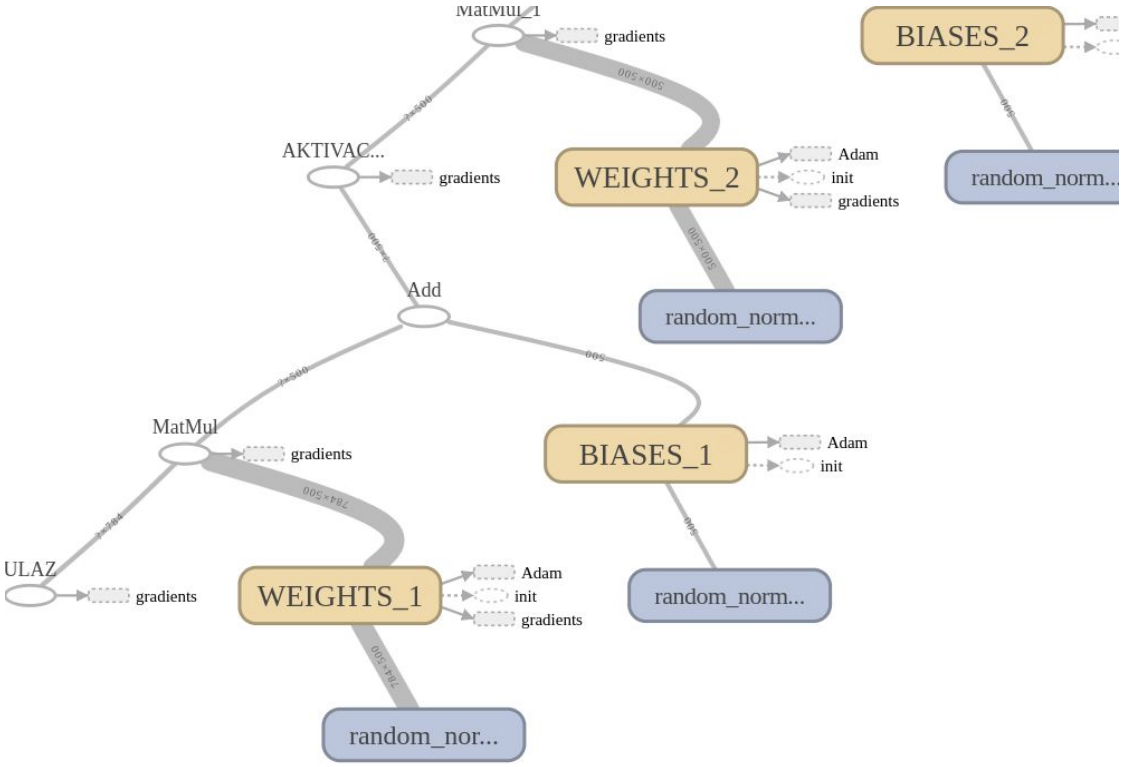
Kao izvor podataka korišen je MNIST (Mixed National Institute of Standards and Technology) skup podataka koji se sastoji od slika ručno pisanih znamenki: 60 000 slika koje se koriste u fazi treniranja modela i 10 000 slika koje se koriste u fazi testiranja, sve veličine 28x28 piksela.



Slika 2.1. Primjer MNIST znamenki

U primjeru je modelirana neuronska mreža sa 3 skrivena sloja, od kojih svaki ima 500 čvorova. Ulaz u mrežu je slika veličine 784 (28x28) piksela predstavljena kao tensor ili matrica veličine [784, 1]. Izlaz iz mreže je tensor veličine [broj klasa, 1]. Broj klasa je 10, a svaka klasa predstavlja jednu znamenku. Na temelju izlaza očitamo klasu slike reprezentiranu najvećom vrijednosti u tensoru. Primjerice ako element na poziciji redak = 0 ima najveću vrijednost zaključujemo da se na slici nalazi znamenka 0.

Jednadžba modela slojeva mreže: $\mathbf{izlaz_{sloj} = (ulaz_{sloj} * weights_{sloj}) + biases_{sloj}}$, gdje su weights i biases TensoFlow varijable(tensori). Ulaz u prvi skriveni sloj je slika a ulaz u svaki sljedeći je izlaz iz prethodnog sloja. Na slici 2.2 prikazan je dio računskog grafa koji prikazuje kako podaci putuju između slojeva, i koje se operacije obavljaju.
Nakon izračunavanje izlaza iz mreže računa se križna entropija izlaza i očekivanog rezultata. Potom se entropija nastoji optimizirati(smanjiti). Za optimizaciju se koristi Adam optimizator.



graf

Slika 2.2. Računski

Što se događa u pozadini?
TensorFlow pretpostavlja vrijednosti za weights i biases za svaki od 3 skrivena sloja i za izlazni sloj. S tim pretpostavljenim vrijednostima izračunava izlaz iz mreže i uspoređuje ga sa očekivanim izlazom. Potom se prilikom obavljanja optimizacije vraća unatrag po mreži i mijenja vrijednosti weights i biases varijabli kako bi rezultat bio što sličniji željenom rezultatu. Jedan takav prolaz kroz mrežu i unatrag po mreži naziva se epoha. U primjeru je provedeno testiranje na 10 epoha.

Za samu instalaciju Tensorflowa bilo je potrebno učiniti slijedeće korake, uključujući izmjene konfiguracijskih datoteka:

```
#postavljanje Jave
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java8-installer (get java)



#postavljanje potrebnih paketa
sudo apt-get install autotools-dev
sudo apt-get install autoconf
sudo apt-get install git zip unzip autoconf automake libtool curl zlib1g-dev maven swig

#instalacija Googleovih protobufa
git clone https://github.com/google/protobuf.git
cd protobuf
# autogen.sh downloads broken gmock.zip in d5fb408d
git checkout master
./autogen.sh
git checkout d5fb408d
autoreconf -vif
./configure --prefix=/usr (--prefix=/usr/lib/erlang/lib;)
make -j 4
sudo make install
cd java
sudo mvn package

#instalacija Bazela
git clone https://github.com/bazelbuild/bazel.git (get bazel)
cd bazel; git checkout 0.2.1;
```

```
cp /usr/bin/protoc third_party/protobuf/protoc-linux-arm32.exe
cp ~/protobuf/java/target/protobuf-java-3.0.0-beta-2.jar
third_party/protobuf/protobuf-java-3.0.0-beta-1.jar
// In file src/main/java/com/google/devtools/build/lib/util/CPU.java line 25
// - ARM("arm", ImmutableSet.of("arm", "armv7l"))
// + ARM("arm", ImmutableSet.of("arm", "armv7l", "aarch64"))
sudo ./compile.sh
sudo cp output/bazel /usr/local/bin
```

#instalacija NumPy-a
sudo apt-get install python-numpy

#promjene potrebne za instalaciju Tensorflowa, ispis git log-a
FILE 1: tensorflow/core/kernels/BUILD

```
-985,7 +985,7 @@ tf_kernel_libraries(
      "reduction_ops",
      "segment_reduction_ops",
      "sequence_ops",
-          "sparse_matmul_op",
+          #DC "sparse_matmul_op",
    ],
    deps = [
      ":bounds_check",
```

- FILE 2: tensorflow/python/BUILD

```
-1110,7 +1110,7 @@ medium_kernel_test_list = glob([
   "kernel_tests/seq2seq_test.py",
   "kernel_tests/slice_op_test.py",
   "kernel_tests/sparse_ops_test.py",
-  "kernel_tests/sparse_matmul_op_test.py",
+  #DC "kernel_tests/sparse_matmul_op_test.py",
   "kernel_tests/sparse_tensor_dense_matmul_op_test.py",
 ])
```

- FILE 3: tensorflow/core/kernels/cwise_op_gpu_select.cu.cc

```
@@ -43,8 +43,14 @@ struct BatchSelectFunctor<GPUDevice, T> {
   const int all_but_batch = then_flat_outer_dims.dimension(1);

   #if !defined(EIGEN_HAS_INDEX_LIST)
-  Eigen::array<int, 2> broadcast_dims{{ 1, all_but_batch }};
-  Eigen::Tensor<int, 2>::Dimensions reshape_dims{{ batch, 1 }};
+  //DC Eigen::array<int, 2> broadcast_dims{{ 1, all_but_batch }};
```

```
+   Eigen::array<int, 2> broadcast_dims;
+   broadcast_dims[0] = 1;
+   broadcast_dims[1] = all_but_batch;
+   //DC Eigen::Tensor<int, 2>::Dimensions reshape_dims{{ batch, 1 }};
+   Eigen::Tensor<int, 2>::Dimensions reshape_dims;
+   reshape_dims[0] = batch;
+   reshape_dims[1] = 1;
     #else
           Eigen::IndexList<Eigen::type2index<1>, int> broadcast_dims;
```

- FILE 4: tensorflow/core/kernels/sparse_tensor_dense_matmul_op_gpu.cu.cc

```
    @@ -104,9 +104,17 @@ struct SparseTensorDenseMatMulFunctor<GPUDevice, T, ADJ_A,
ADJ_B> {
       int n = (ADJ_B) ? b.dimension(0) : b.dimension(1);

     #if !defined(EIGEN_HAS_INDEX_LIST)
-    Eigen::Tensor<int, 2>::Dimensions matrix_1_by_nnz{{ 1, nnz }};
-    Eigen::array<int, 2> n_by_1{{ n, 1 }};
-    Eigen::array<int, 1> reduce_on_rows{{ 0 }};
+    //DC Eigen::Tensor<int, 2>::Dimensions matrix_1_by_nnz{{ 1, nnz }};
+    Eigen::Tensor<int, 2>::Dimensions matrix_1_by_nnz;
+    matrix_1_by_nnz[0] = 1;
+    matrix_1_by_nnz[1] = nnz;
+    //DC Eigen::array<int, 2> n_by_1{{ n, 1 }};
+    Eigen::array<int, 2> n_by_1;
+    n_by_1[0] = n;
+    n_by_1[1] = 1;
+    //DC Eigen::array<int, 1> reduce_on_rows{{ 0 }};
+    Eigen::array<int, 1> reduce_on_rows;
+    reduce_on_rows[0] = 0;
     #else
       Eigen::IndexList<Eigen::type2index<1>, int> matrix_1_by_nnz;
```

- FILE 5: tensorflow/stream_executor/cuda/cuda_blas.cc

```
    @@ -25,6 +25,12 @@ limitations under the License.
     #define EIGEN_HAS_CUDA_FP16
     #endif

    +#if CUDA_VERSION >= 8000
    +#define SE_CUDA_DATA_HALF CUDA_R_16F
    +#else
    +#define SE_CUDA_DATA_HALF CUBLAS_DATA_HALF
    +#endif
    +
```

```
 #include "tensorflow/stream_executor/cuda/cuda_blas.h"

 #include <dlfcn.h>
@@ -1680,10 +1686,10 @@ bool CUDABlas::DoBlasGemm(
    return DoBlasInternal(
    dynload::cublasSgemmEx, stream, true /* = pointer_mode_host */,
    CUDABlasTranspose(transa), CUDABlasTranspose(transb), m, n, k, &alpha,
-   CUDAMemory(a), CUBLAS_DATA_HALF, lda,
-   CUDAMemory(b), CUBLAS_DATA_HALF, ldb,
+   CUDAMemory(a), SE_CUDA_DATA_HALF, lda,
+   CUDAMemory(b), SE_CUDA_DATA_HALF, ldb,
    &beta,
-   CUDAMemoryMutable(c), CUBLAS_DATA_HALF, ldc);
+   CUDAMemoryMutable(c), SE_CUDA_DATA_HALF, ldc);
 #else
    LOG(ERROR) << "fp16 sgemm is not implemented in this cuBLAS version "
           << "(need at least CUDA 7.5)";
```

- FILE 6: tensorflow/stream_executor/cuda/cuda_gpu_executor.cc

```
@@ -888,6 +888,9 @@ CudaContext* CUDAExecutor::cuda_context() { return context_; }
 // For anything more complicated/prod-focused than this, you'll likely want to
 // turn to gsys' topology modeling.
 static int TryToReadNumaNode(const string &pci_bus_id, int device_ordinal) {
+  // DC - make this clever later. ARM has no NUMA node, just hardoce to return 0
+  LOG(INFO) << "ARM has no NUMA node, hardcoding to return zero";
+  return 0;
 #if defined(__APPLE__)
    LOG(INFO) << "OS X does not support NUMA - returning NUMA node zero";
    return 0;
```

I, na kraju, instalacija samog Tensorflowa:

```
sudo ./configure

sudo bazel build --jobs 1 -c opt --config=cuda
//tensorflow/tools/pip_package:build_pip_package --verbose_failures
sudo bazel-bin/tensorflow/tools/pip_package/build_pip_package
/media/sdcard/tensorflow/tensorflow_pkg

sudo pip install tensorflow_pkg/tensorflow-0.9.0-py2-none-any.whl
```

Performanse Tensorflowa na TX1 i na drugim uređajima:

Epoha	*Entropija*	*Vrijeme treniranja [s]*	*Vrijeme testiranja [s]*	*Točnost*
TX1 GPU+CPU				
1	1732116,71	6,36	0,23	89,42 %
2	406147,16	5,99	0,24	92,62%
3	222690,67	19,28	0,22	92,63%
4	131807,76	6,21	0,24	93,38%
5	81027,86	6,19	0,23	93,18%
6	53087,35	5,97	0,25	94,08%
7	33479,39	6,29	0,24	94,59%
8	27676,36	6,12	0,24	95,02%
9	24140,80	6,10	0,26	95,29%
10	20701,32	6,23	0,23	95,36%
				Vrijeme izvođenja: 79,90 s
TX1 CPU				
1	1865658,20	22,74	0,89	90,00%
10	18678,99	23,17	0,90	94,46%
				Vrijeme izvođenja: 250,74 s

Intel(R) Core(TM) i7-4510U CPU @ 2.00GHz (CPU)				
1	1890135,89	10,30	0,48	89,07%
10	17472,56	12,09	0,50	95,13%
			Vrijeme izvođenja: 114,54 s	
Intel(R) Core(TM) i5-4200U CPU @ 1.60GHz (CPU)				
1	2016731,73	13,55	0,60	89,79%
10	21447,96	13,97	0,60	94,89%
			Vrijeme izvođenja: 144,58 s	
GTX 970				
1	1859434,93	1,28	0,05	89,93%
10	17010,85	1,00	0,03	95,36%
			Vrijeme izvođenja: 11,59 s	

Tablica 2.1.1 Usporedbe vremena izvođenja

2.2 cuBLAS - CUDA Basic Linear Algebra Subroutines

CuBLAS je implementacija BLAS (Basic Linear Algebra Subroutines), biblioteke s osnovnim matričnim operacijama, na CUDA-i

CuBLAS dolazi kao dio CUDA paketa, kojega smo već instalirali za potrebe Tensorflowa i Caffe-a, pa nije bilo potrebe za većim promjenama na sustavu.

CuBLAS je benchmarkan množenjem matrica raznih veličina. Kako bi se dodatno benchmarkale i performanse CPU-a i radne memorije modula TX1, korišteno je i "naivno" množenje matrica, bez paralelizacije.

CuBLAS kodovi jednostavno se kompajliraju korištenjem nvcc-a

```
nvcc -o program program.c -lcublas
```

Rezultati:

Kvadratne matrice, vrijeme u sekundama
Tip: float (10 iteracija za svaku veličinu)

Veličina matrice	TX1	nazgul (GTX 970)
2	3.18e-05	6.8e-06
4	2.55e-05	5.9e-06
8	2.41e-05	6.8e-06
16	2.36e-05	4.2e-06
32	2.91e-05	4.9e-06
64	2.87e-05	4.8e-06
128	3.36e-05	5.4e-06
256	0.0005768	8.8e-06
512	0.0005776	7.1e-06
1024	0.000579	9.6e-06

Tip: double (10 iteracija za svaku veličinu)

Veličina matrice	TX1	nazgul (GTX 970)
2	2.99e-05	4.5e-06
4	2.58e-05	4e-06
8	2.46e-05	4.9e-06
16	2.34e-05	4.9e-06
32	2.92e-05	5.6e-06
64	2.71e-05	5.6e-06
128	3.26e-05	5.7e-06
256	0.0005769	1.39e-05
512	0.0005805	1.37e-05
1024	0.000584	1e-05

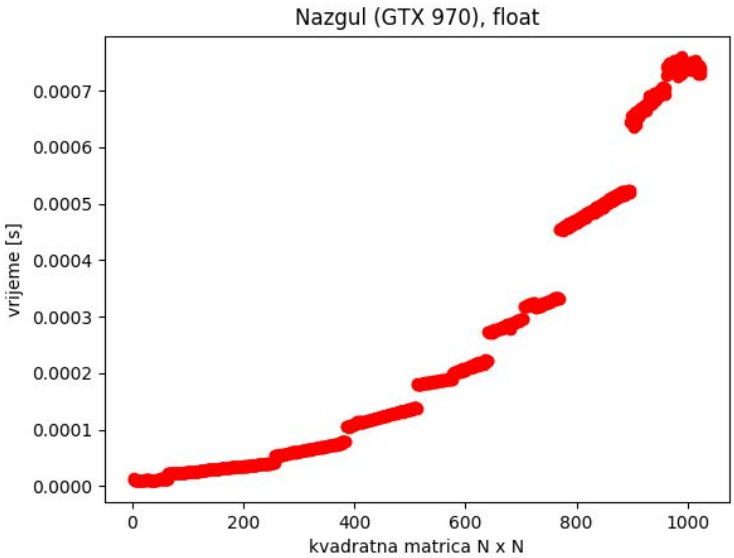
Bonus:

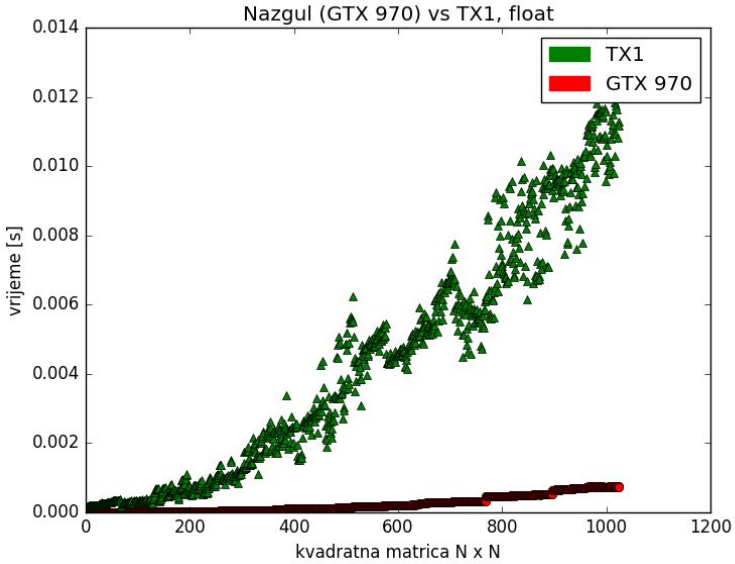
CPU gcc s -O3, na nazgulu kompajlirano bez zastavice -msse4

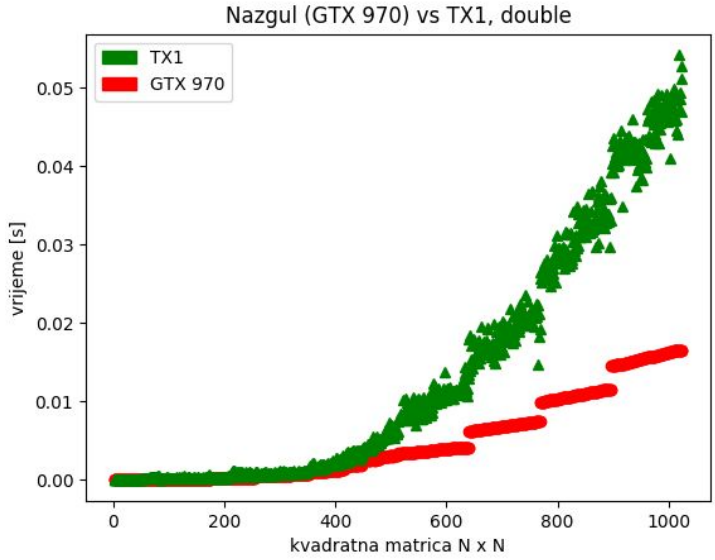
Tip: float (10 iteracija za svaku veličinu)

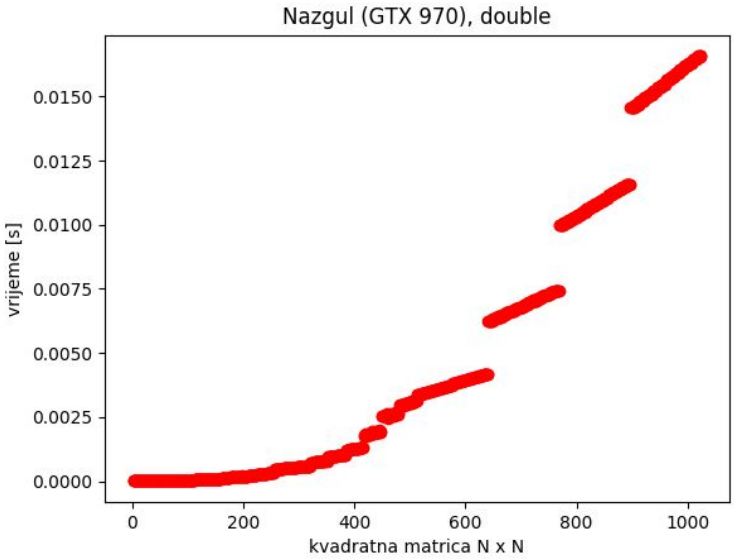
Veličina matrice	TX1(A57)	nazgul (i7)
2	3e-07	1e-07
4	6e-07	1e-07
8	2.1e-06	1.3e-06
16	1.17e-05	3.3e-06
32	7.53e-05	1.64e-05
64	0.0005889	0.0001071
128	0.0112163	0.0012035
256	0.0870655	0.0159207
512	0.715215	0.169786
1024	36.7075	1.4348

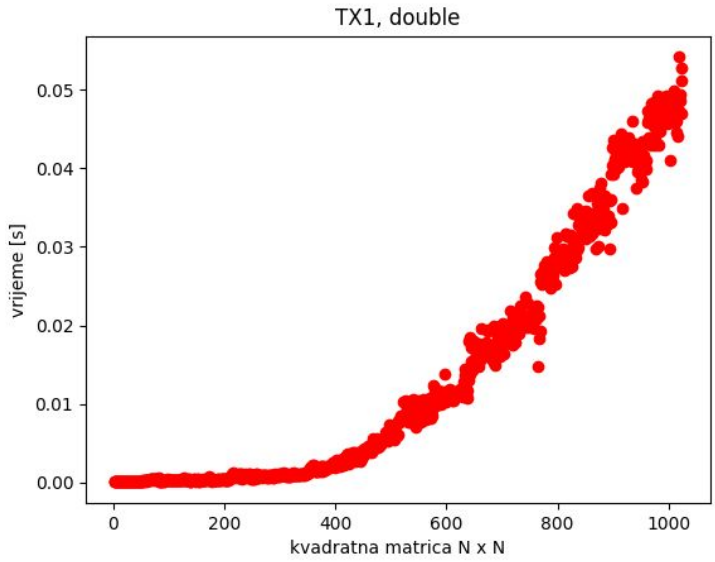
Ovisnosti veličine matrice o vremenu izračuna grafički su prikazani u nastavku:




Nazgul (GTX 970) vs TX1, float
TX1
GTX 970
vrijeme [s]
kvadratna matrica N x N
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
0
200
400
600
800
1000
1200


Nazgul (GTX 970) vs TX1, double
TX1
GTX 970
vrijeme [s]
kvadratna matrica N x N
0.05
0.04
0.03
0.02
0.01
0.00
0
200
400
600
800
1000





Iz navedenog očito je da GPU TX1 veoma zaostaje za stolnim računalima po performansama, no njegov CPU je još nižih performansi u odnosu na "stolne" ekvivalente. No, s obzirom na dimenzije i potrošnju energije TX1, ovi rezultati su i dalje veoma zanimljivi. Također, pri množenju matrica dimenzija 8128x8128 dolazi do zapunjavanja radne memorije grafičkog procesora TX1 i pucanja programa.

2.3 Caffe

Caffe je okvir za duboko učenje (*deep learning*), razvijen na Berkeleyu. Umjesto kao kod, modeli mu se predaju kao sheme u običnom tekstu (*plaintext*).

Prilikom instalacije TX1 bilo je nekoliko problema; kako se za flashanje TX1 prema službenim uputama zahtjeva Ubuntu verzije 14.04., koji se i instalira na TX1. No, kada se instaliraju potrebni paketi za Caffe, čini se da se usput povuku i potrebni paketi za upgrade na 16.04. To ponašanje mi je bilo neočekivano, jer sam u dosadašnjem korištenju distribucija izvedenih iz Debiana uvijek nadogradnju verzije Linuxa morao inicirati sa
`apt-get dist-upgrade.`

Kako smo originalno imao na TX1 imali stariju verziju Ubuntua, koristio sam i upute za verzije manje od 16.04, koje nisu bile prikladne nakon nadogradnje sustava, uz poneke detalje koji nisu navedeni u dotičnim uputama. Nakon mnogo pokušaja i isprobavanja, ustanovljeni su svi koraci potrebni za instalaciju Caffea na TX1:

```
#Upute za instalaciju:
#prvo slijedi instalacija dependency-a
sudo add-apt-repository universe
sudo apt-get update -y

sudo apt-get install cmake -y

sudo apt-get install libprotobuf-dev libleveldb-dev libsnappy-dev libhdf5-serial-dev
protobuf-compiler -y
sudo apt-get install --no-install-recommends libboost-all-dev -y
```

```
sudo apt-get install libatlas-base-dev -y

sudo apt-get install libgflags-dev libgoogle-glog-dev liblmdb-dev -y
sudo apt-get install python-dev python-numpy -y

sudo usermod -a -G video $USER
#direktorij se mijenja u HOME direktorij USER-a, u slučaju da se caffe želi instalirati na
#custom lokaciji, ovdje je to moguće definirati
cd $HOME

#kloniranje repozitorija s GitHuba
git clone https://github.com/BVLC/caffe.git
cd caffe
# Makefile.config je konfiguracijska datoteka koju koristi Makefile
cp Makefile.config.example Makefile.config
mkdir build
cd build
sudo cmake -DCUDA_USE_STATIC_CUDA_RUNTIME=OFF ..

echo "INCLUDE_DIRS += /usr/include/hdf5/serial/" >> ../Makefile.config

sudo make -j4 all

sudo make -j4 runtest
```

Već zadnja linija *builda* (runtest) pruža benchmark kompletnog sustava. U slučaju da se žele filtrirati testovi, može se koristiti

$(CAFFE_ROOT)/build/tests/python/gtest --filter GPU

Što se tiče performansi, sam runtest (koji vrti testove i na GPU, i na CPU), sa oko 2000 testova, traje oko 45 minuta dok je TX1 relativno neopterećen (ne vrte se drugi veći procesi), dok na sustavu s i7 procesorom i NVidia 950m GPU traje oko 7 minuta. Valja napomenuti da većina vremena ode na testove memorije i CPU.

U slučaju da se želi testirati treniranje čestih mreža, postoje gotovi primjeri u examples direktoriju. Na primjer, kao primjer je dana blvc_alexnet mreža, čije treniranje izvođenjem

```
build/tools/caffe time --model=models/blvc_alexnet/deploy.prototxt --gpu=0
```

traje ukupno 11.34 ms. Ova mreža u potpunosti je trenirana na GPU. Ako se ista mreža utrenira bez korištenja GPU supporta pomoću

```
build/tools/caffe time --model=models/blvc_alexnet/deploy.prototxt
```

Kao drugi primjer uzet je MNIST, kao i u djelu s Tensorflowom. Ovdje ukupno treniranje s 10000 iteracija traje ukupno malo vise od 3 minute. (Iteracija je ovdje isto što i epoha u primjeru s Tensorflowom). Samo testiranje mreže traje ukupno 493 ms (prosječno 9.8 ms po iteraciji) na GPU, odnosno 15.469 sekundi na CPU

Zbog problema oko nekompaktibilnosti novijih verzija Cude s Caffe-om, iste mreže nismo mogli isprobati i na *Nazgul-u*.

2.4 Torch

Torch je okvir za znanstveno računanje koji podržava niz algoritama strojnog učenja bazira se na upotrebi GPU-a. Cilj nam je bio pomoću njega ispitati efikasnost CUDNN-a na TX1.

#Preuzimanje i instalacija
git clone https://github.com/torch/distro.git torch --recursive
cd torch
bash install-deps
./install.sh
source ~/.profile

#instalacija dodatnih paketa
luarocks install cunn
luarocks install cudnn

S adrese https://github.com/jcjohnson/cnn-benchmarks smo preuzeli skripte za ispitivanje performansi, a s https://drive.google.com/open?id=0Byvt-AfX75o1STUxZTFpMU10djA smo preuzeli modele pomoću kojih smo ispitivali performanse.

Ispitivanje performansi smo pokrenuli sa:
python2 run_cnn_benchmarks.py

Svi testovi se su se trebali pokretati u 10 iteracija, no već prilikom prve iteracije za svaki test bi nam se ispisalo *Killed*.
Analizom potrošnje radne memorije, utvrdili smo da se pri pokretanju svakog testa zauzme skoro cijela radna memorije prije nego što se proces uništi.

3. Zaključak

Kao rezultat projekta dobili smo informaciju da se okviri Tensorflow, Caffe i CuBLAS mogu pokrenuti na TX1, i imaju vrlo dobre performanse ako se u obzir uzme njegova dimenzija i potrošnja energije. Službene upute za instalaciju istih su se pokazale manjkave, te su potpune upute uključene u dokument. Sam modul se pokazao relativno nestabilnim, s povremenim rušenjima, uključujući jedan potpun kvar koji je zahtijevao zamjenu TX1.

4. Literatura

http://www.nvidia.com/object/jetson-tx1-module.html

https://github.com/tensorflow/tensorflow

https://github.com/BVLC/caffe

http://stackoverflow.com/

https://forums.geforce.com/

http://yann.lecun.com/exdb/mnist/