GPU TECHNOLOGY CONFERENCE

April 4-7, 2016 | Silicon Valley

UNLOCK PERFORMANCE LIMIT OF DNN BY CUDA® IN R

Patric Zhao, GPU Architect, NVIDIA

patricz@nvidia.com

PRESENTED BY

NVIDIA.

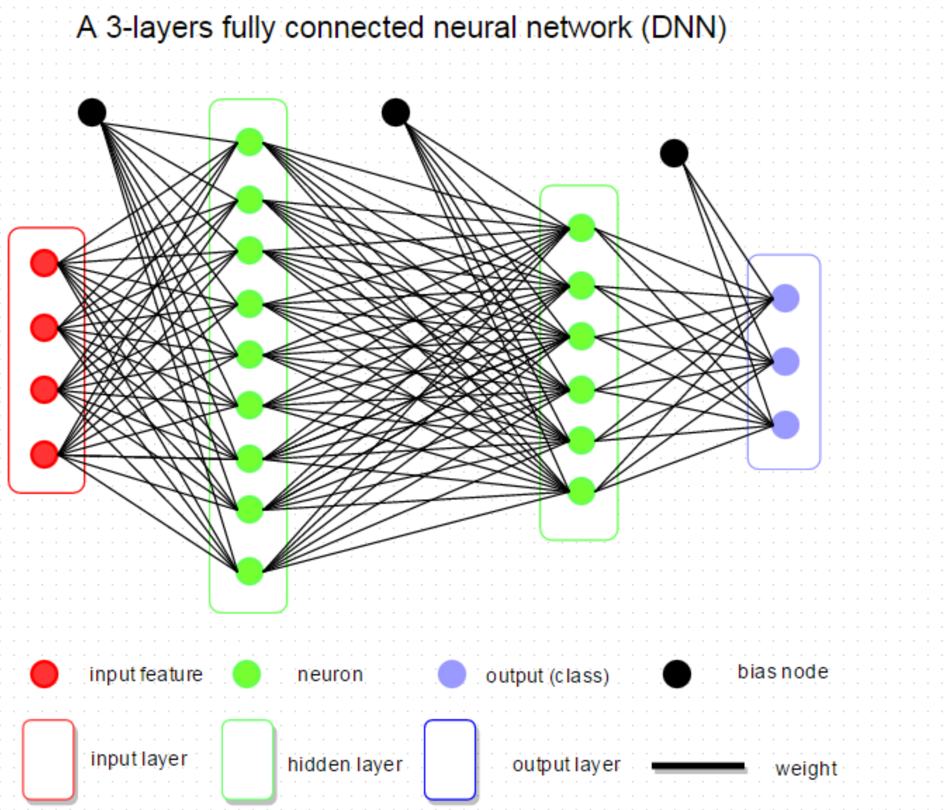
AGENDA

1. Background
2. Build DNN by R language
3. CUDA Accelerations and Optimizations
4. Scale out by Multi-GPUs
5. Summary

BACKGROUND

DNN: Deep Neural Network

- Great successful in CV, NLP, etc.
- Automatic Feature Extraction
- Computation intensive algorithm
- Still a rapid development field



GPU TECHNOLOGY CONFERENCE

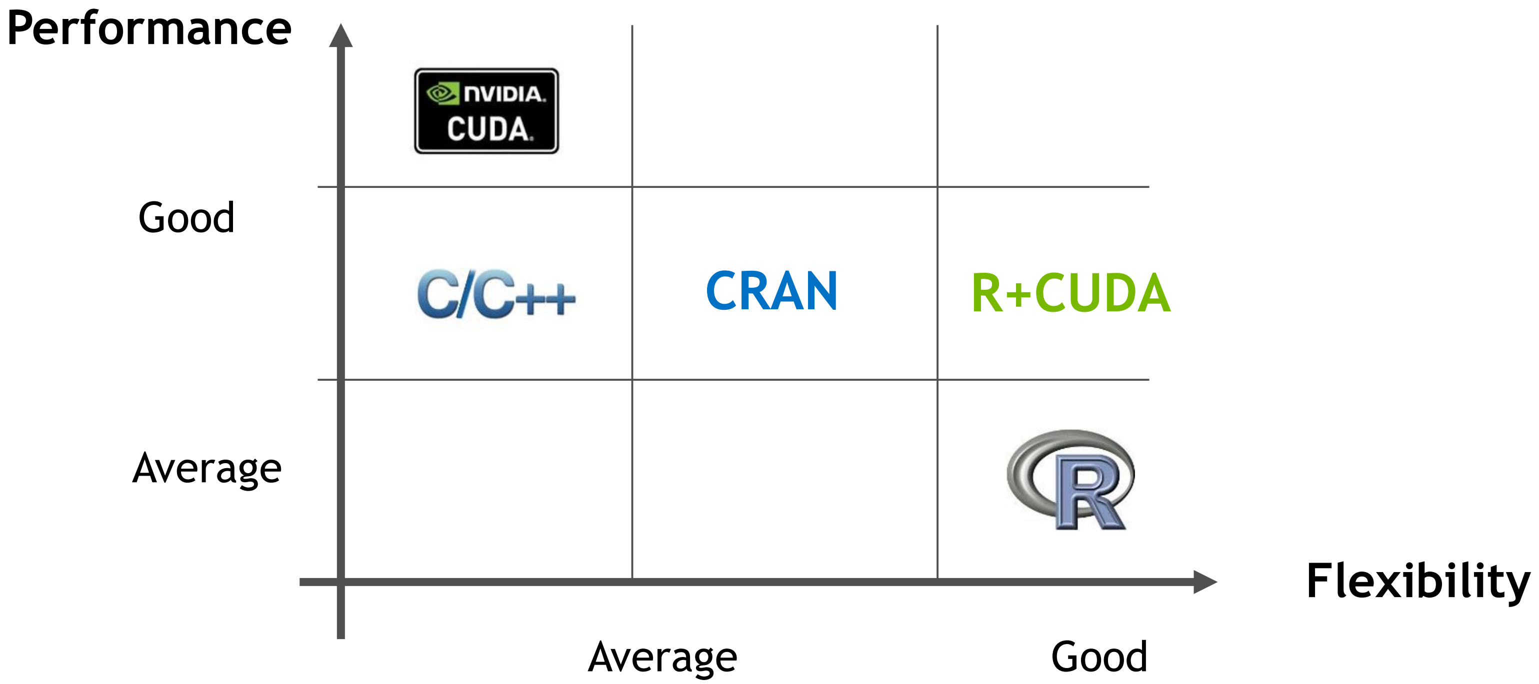
Mature Packages in R:

Packages	Backend	Compute Resources
nnet	C/C++	Single thread
nerualnet	C/C++	Single thread
DARCH	C/C++	Single thread
deepnet	C/C++	Single thread
H2O	JAVA	Multi-threads, multi-nodes
mxnet	C/C++/CUDA	Multi-threads, GPUs, multi-nodes

S6853 - MXNet: Flexible Deep Learning Framework from Distributed GPU Clusters to Embedded Systems
L6143 - Train and Deploy Deep Learning for Vision, Natural Language and Speech Using MXNet

In this talk, I am going to introduce how to:

- Build DNN network by native R
- Accelerate R code under CUDA ecosystem
- Make our solutions as simple as possible



BUILD DNN BY R LANGUAGE

GPU TECHNOLOGY CONFERENCE

- Code Frame: Stanford Open Course, CS231n, 2015

  Convolutional Neural Networks for Visual Recognition

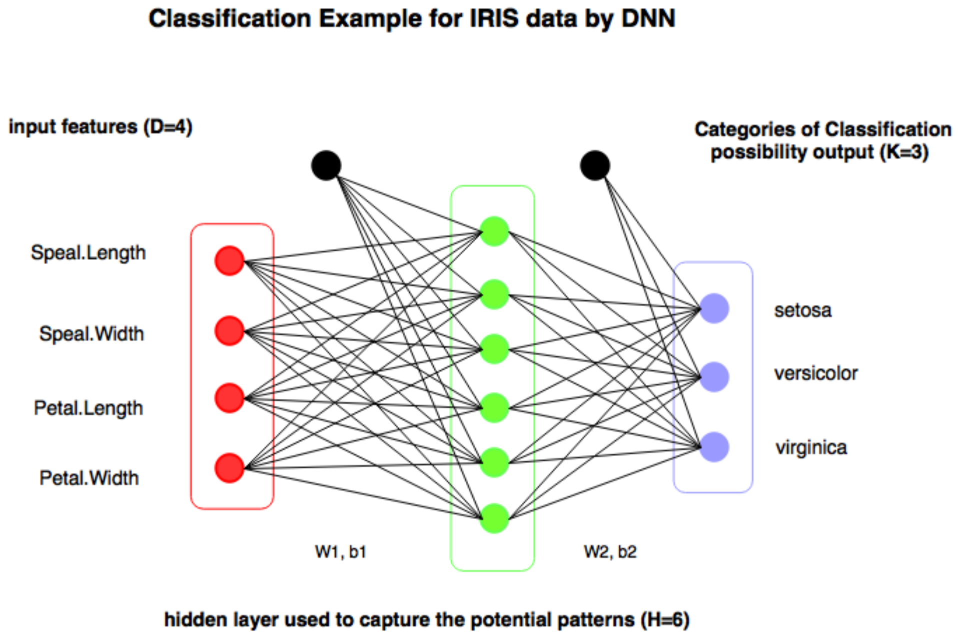
- Classification Network

  1 hidden layer w/ softmax, ReLu

- Vectorization Representation

- Fully connected network

- Python to R translations



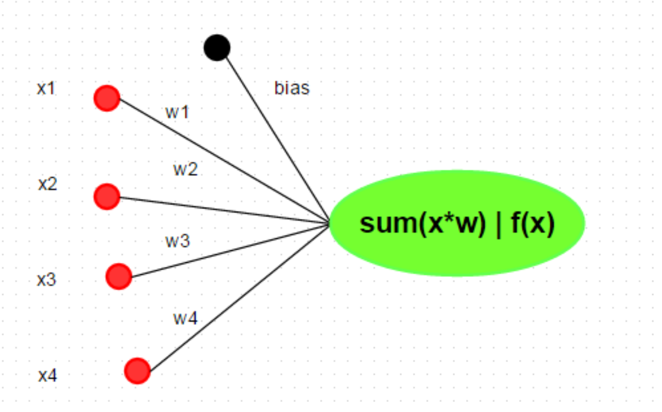
Core complements of DNN in R:

Weights and Bias : matrix representation

```
weight <- 0.01*matrix(rnorm(h*k), nrow=h, ncol=k)
bias    <- matrix(0, nrow=1, ncol=H)
```

Neuron : computation parts

```
neuron <- sweep(input %*% weights ,2, bias, '+')
neuron <- pmax(neuron, 0)   # ReLu
```



Cost function : Softmax

```
score.exp <- exp(score)
probs      <-sweep(score.exp, 1, rowSums(score.exp), '/')
```

Prediction: Feed Forward

```r
predict <- function(model, data = X.test) {
  new.data <- data.matrix(data)
  # Feed Forwad
  hidden.layer <- sweep(new.data %*% model$W1 ,2, model$b1, '+')
  # neurons : Rectified Linear
  hidden.layer <- pmax(hidden.layer, 0)
  score <- sweep(hidden.layer %*% model$W2, 2, model$b2, '+')
  # Loss Function: softmax
  score.exp <- exp(score)
  probs     <- sweep(score.exp, 1, rowSums(score.exp), '/')
  labels.predicted <- max.col(probs)
  return(labels.predicted)
}
```

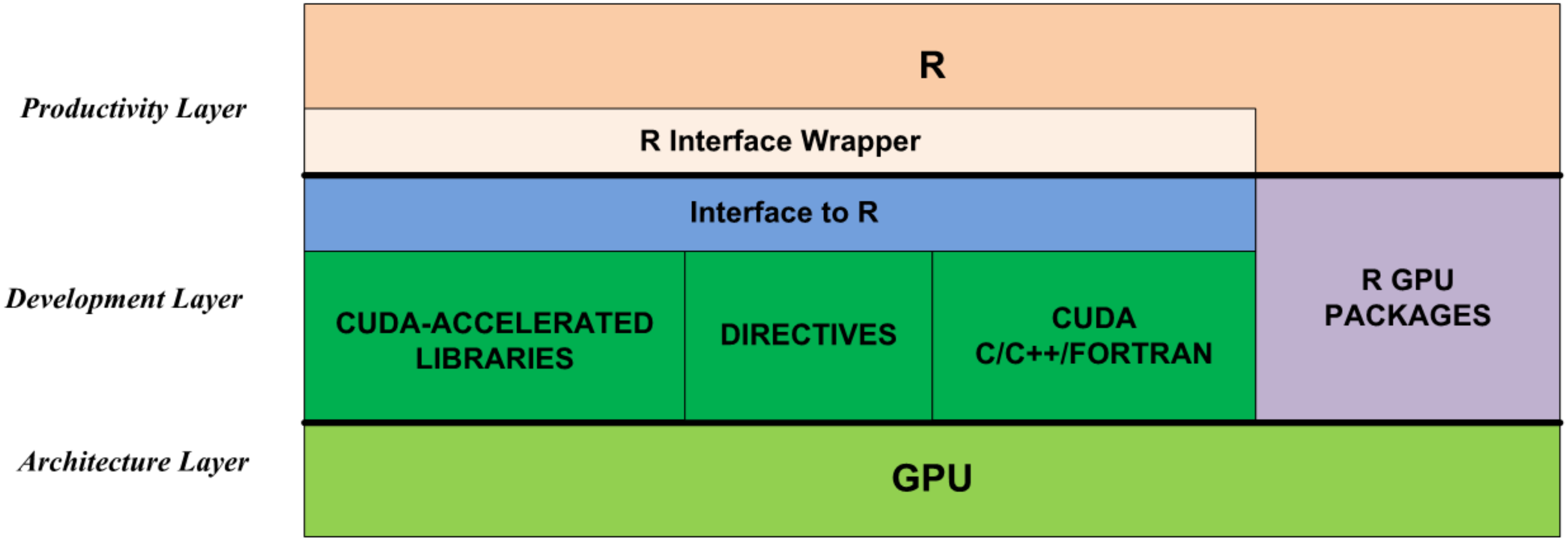
Training : Feed Forward + Back propagation

```
train <- function(x, y, model, traindata, hidden,…) {
      # 1. Feed Forward . . .
      # 2. Compute the loss . . .
      # 3. Backward
        dscores <- probs
        dscores[Y.index] <- dscores[Y.index] -1
        dscores <- dscores / batchsize
        dW2 <- t(hidden.layer) %*% dscores
        db2 <- colSums(dscores)
        dhidden <- dscores %*% t(W2)
        dhidden[hidden.layer <= 0] <- 0
        dW1 <- t(X) %*% dhidden
        db1 <- colSums(dhidden)
        # update ....
 }
```

CUDA ACCELERATIONS AND OPTIMIZATIONS

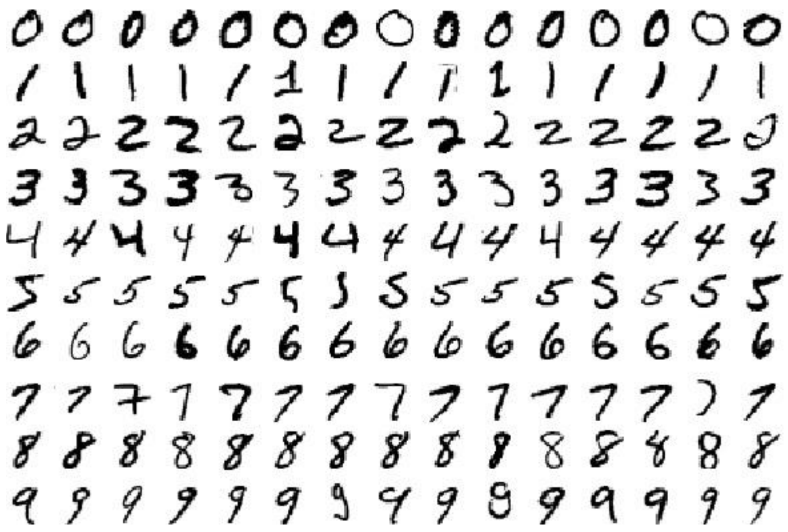
RECAP : How to accelerate R by CUDA ?

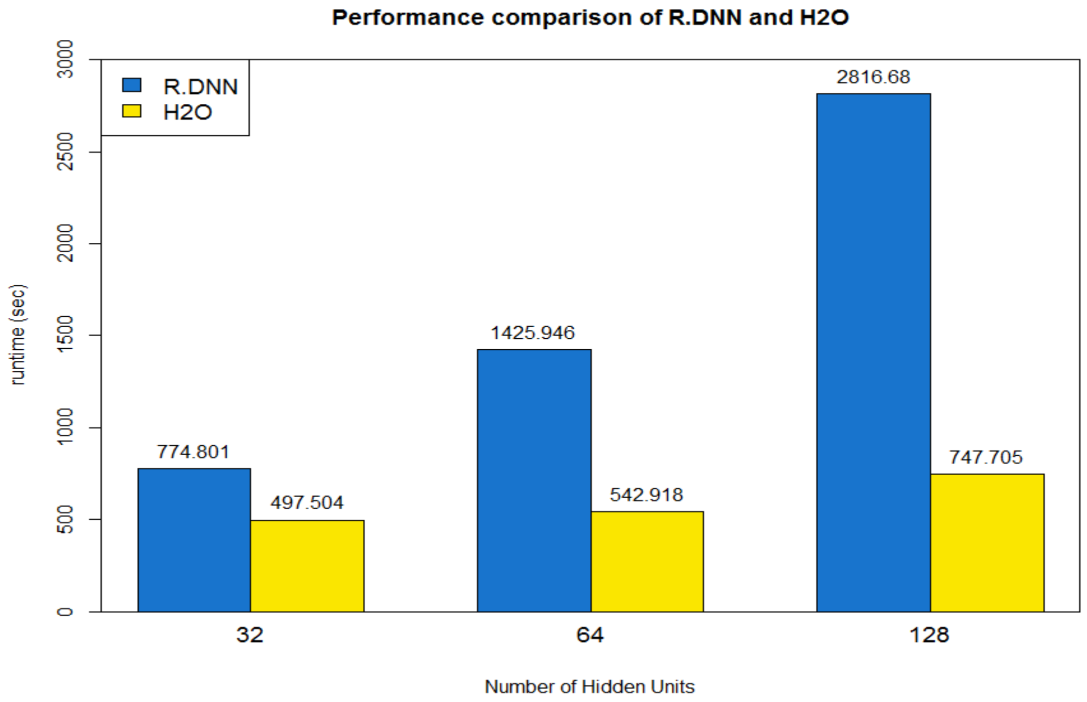
My GTC15 talk: Accelerate R Applications with CUDA



Benchmark : MNIST handwritten digit dataset

- Input features: 28X28 = 784, Output classes: 10 (0-9);
- Training Set 60,000, testing set: 1 0,000
- DNN Architecture: 2-layers fully connected neural network



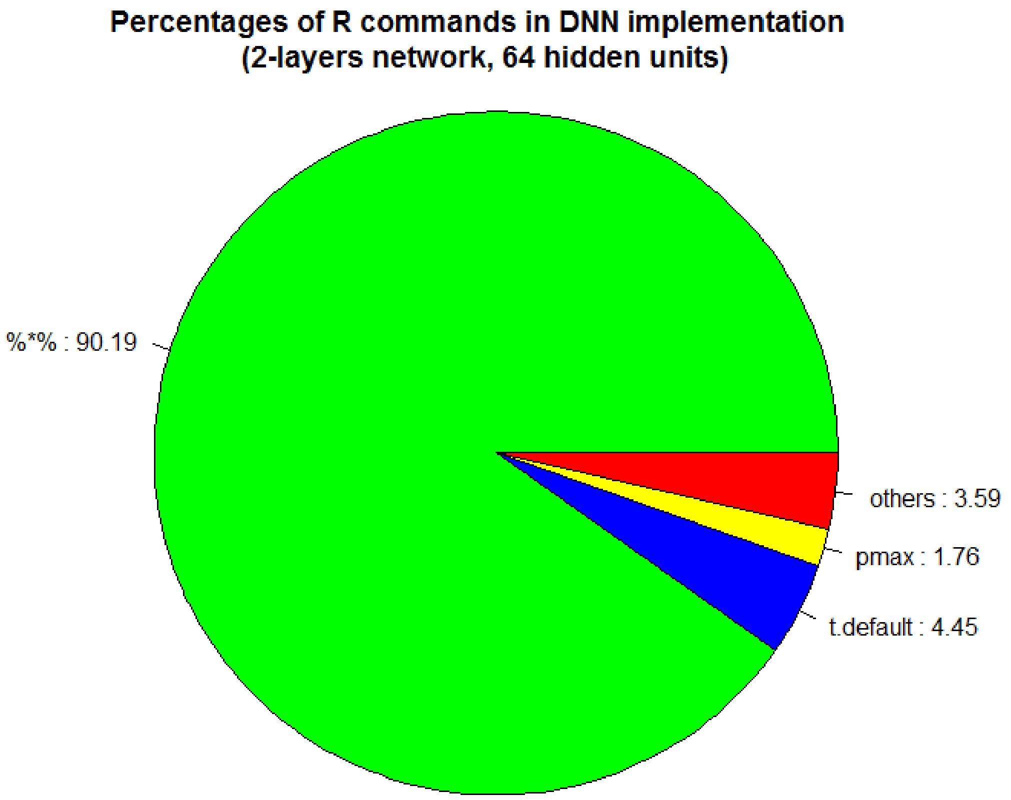


CPU: Ivy Bridge E5-2690 v2 @ 3.00GHz, dual socket 10-core, 128G RAM;

Profiling

Rprof(), summaryRprof()

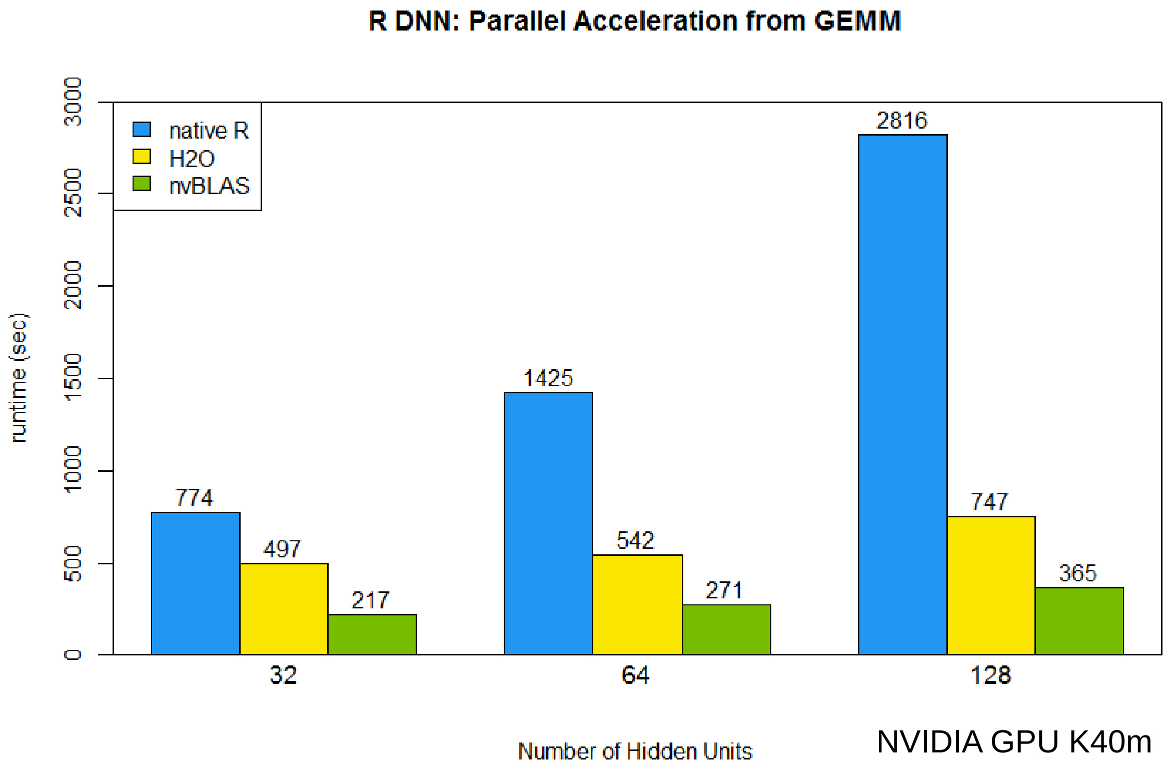
Break Donw R DNN Runtime				
	total.time	total.pct	self.time	self.pct
train.dnn	1386	100	9.74	0.7
%*%	1250.08	90.19	1250.08	90.19
sweep	676.32	48.8	1.58	0.11
t	61.64	4.45	0.02	0
t.default	61.62	4.45	61.62	4.45
pmax	28.42	2.05	24.4	1.76
aperm	21.96	1.58	0	0
aperm.default	11.6	0.84	11.6	0.84
array	10.36	0.75	10.36	0.75
<=	5.72	0.41	5.72	0.41
mostattributes<-	4.02	0.29	4.02	0.29
exp	3.6	0.26	3.6	0.26
unname	1.46	0.11	0.18	0.01
is.data.frame	1.28	0.09	1.28	0.09
data.matrix	1.28	0.09	0	0
colSums	0.86	0.06	0.86	0.06
/	0.52	0.04	0.52	0.04
rowSums	0.36	0.03	0.36	0.03
-	0.04	0	0.04	0
sum	0.02	0	0.02	0



DROP-IN ACCELERATION

By nvBLAS Library on Linux

> *env LD_PRELOAD=libnvblas.so   R CMD BATCH MNIST_DNN.R*



NVIDIA GPU K40m

OPTIMIZATIONS

- Profiling again after NVIDIA GPU acceleration

Break Donw R DNN Runtime (nvBLAS, HU=64)				
function	total.time	total.pct	self.time	self.pct
train.dnn	274	100	10.74	3.92
%*%	114.58	41.82	114.58	41.82
sweep	**87.28**	31.85	1.8	0.66
t.default	**73.42**	26.8	73.42	26.8
t	**73.42**	26.8	0	0
pmax	30.9	11.28	24.62	8.99
aperm	29.74	10.85	0.04	0.01
aperm.default	19.08	6.96	19.04	6.95
array	10.62	3.88	10.62	3.88
mostattributes<-	6.28	2.29	6.26	2.28

Opt.1 : replace *t(X) %*% matrix* and *matrix %*% t(X)* with R internal function

```
# original:  t() with matrix multiplication
    dW2     <- t(hidden.layer) %*% dscores
    dhidden <- dscores %*% t(W2)

# Opt1: use builtin function
   dW2     <- crossprod(hidden.layer, dscores)
   dhidden <- tcrossprod(dscores, W2)
```

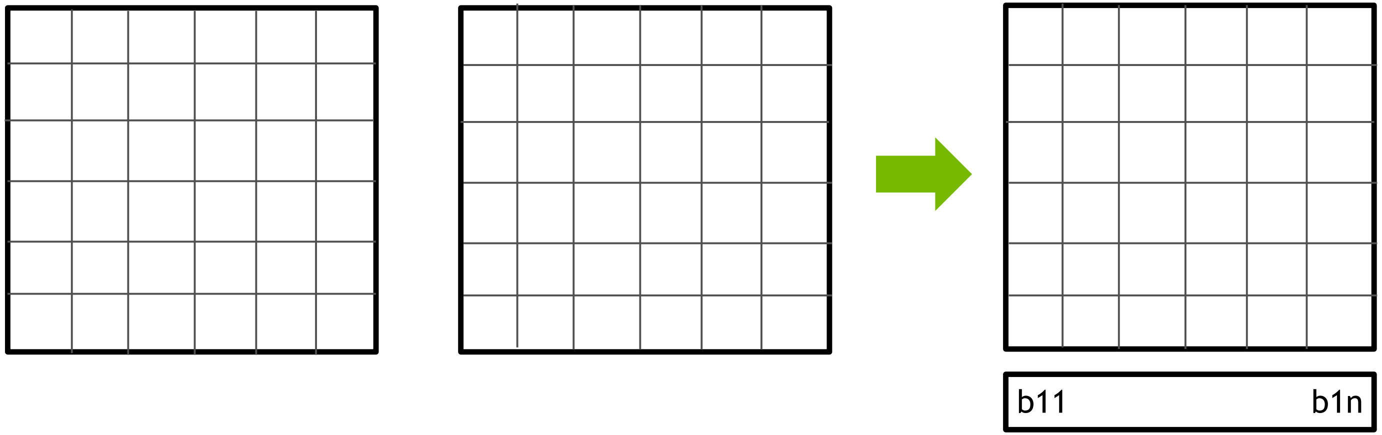
GPU TECHNOLOGY CONFERENCE

Opt.2 : replace *sweep()* by matrix multiplication

```
# Opt2: original code
 hidden.layer <- sweep(X %*% W1 ,2, b1, '+')
```

Matrix Multiplication

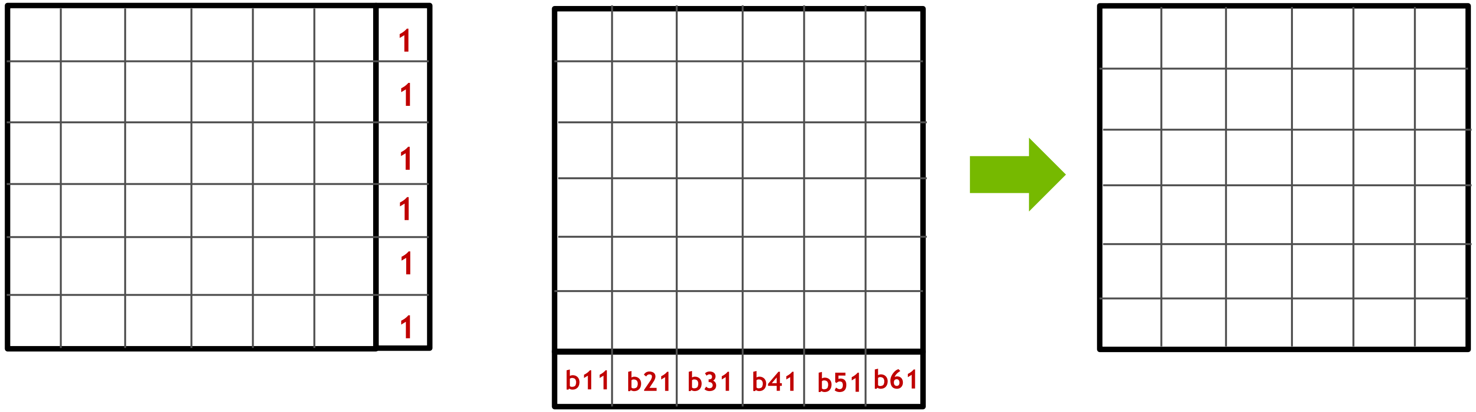
Sweep add bias



Opt.2 : replace *sweep()* by matrix multiplication

```
# Opt2: remove `sweep`
 hidden.layer <- X1 %*% W1b1
```

Matrix Multiplication



X1 <- cbind(X, rep(1, nrow(X)))

W1b1 <- rbind(W1, b1)

Optimization for R DNN (nvBLAS, HU=64)			
by.self	original	Opt1: replace t()	Opt2: remove sweep()
%*%	**112.02**	**53.28**	**53.72**
sweep	**90.46**	**86.7**	-
t	**73.98**	-	-
t.default	**73.96**	-	-
aperm	33.34	30.78	-
pmax	32.52	31.44	**31.58**
aperm.default	22.36	19.84	-
array	10.98	10.92	-
crossprod	-	**23.06**	**23.4**
tcrossprod	-	**2.52**	**2.54**
cbind	-	-	8.9
Total (sec)	266.28	166.76	144.71
Speedup	**1X**	**1.60X**	**1.84X**

GREEN: GPU accelerated parts
RED: Performance limiters

Opt.3 : implement *pmax()* by CUDA

- .*Call()* function in R with simple CUDA implementation of pmax()

(w/ .*C()* to call cuBLAS API and on Windows Platform)

```
# preload static object file
dyn.load("cudaR.so")

# GPU version of ReLU (pmax)
pmax.cuda <- function(A, threshold, devID=0)
{
  rst <- .Call("pmax_cuda", A, threshold, as.integer(devID))
  dim(rst) <- dim(A)
  return(rst)
}
```

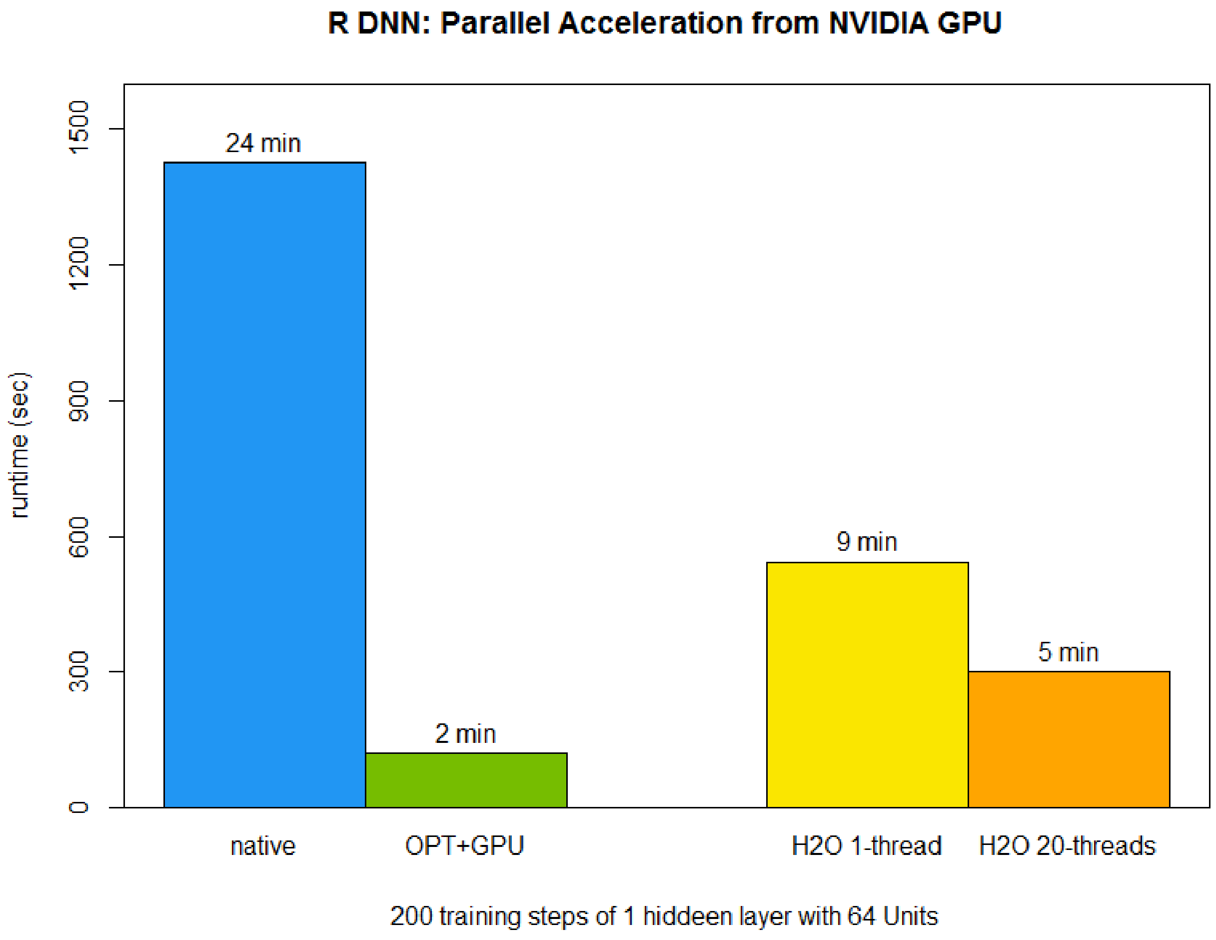
```
// CUDA: simple implementation of pmax
__global__ void pmax_kernel(double *A, const int M, const int N, const double threshold){
    int tid = blockIdx.x * blockDim.x + threadIdx.x;
    if(tid<M*N){ A[tid] = (A[tid] > threshold)?A[tid]:0; }
    return;
}
// Specified for DNN by .CAll format
SEXP pmax_cuda(SEXP A, SEXP threshold) {
    // Initialization including R to C data transfer, CUDA preparations
     . . .
    pmax_kernel<<<(mm*nn-1)/512+1, 512>>>(A_d, mm, nn, gw);
    cudaMemcpy(REAL(Rval), A_d, mm*nn*sizeof(double),  cudaMemcpyDeviceToHost);
    cudaDeviceSynchronize();
    // Free data, unprotect …
    return Rval;
}
```

GPU TECHNOLOGY CONFERENCE

Final Profiling:

Optimizations for R DNN (HU=64)					
by.total	native R	nvBLAS + R			CUDA
	base code	base code	Opt1: replace t()	Opt2: replace sweep()	Opt3: pmax.cuda
train.dnn	1436	278.7	167.7	144.66	119.86
%*%	1250.08	112.02	53.28	53.72	**53.72**
sweep	676.32	90.46	86.7	-	-
t	61.64	73.98	-	-	-
t.default	61.62	73.96	-	-	-
aperm	21.96	33.34	30.78	-	-
pmax	28.42	32.52	31.44	31.58	**6.76**
aperm.default	11.6	22.36	19.84	-	-
array	10.36	10.98	10.92	-	-
crossprod	-	-	23.06	23.4	**23.26**
tcrossprod	-	-	2.52	2.54	**2.62**
cbind	-	-	-	8.9	8.92
Total	**1425.946**	**266.28**	**166.76**	**144.71**	**119.40**
		1X	**1.60X**	**1.84X**	**2.24X**
Speedup	**1X**	**5.36X**	**8.55X**	**9.85X**	**11.94X**

Performance on Linux



SCALE OUT BY MULTI-GPUS

DATA PARALLEL BY HOGWILD!

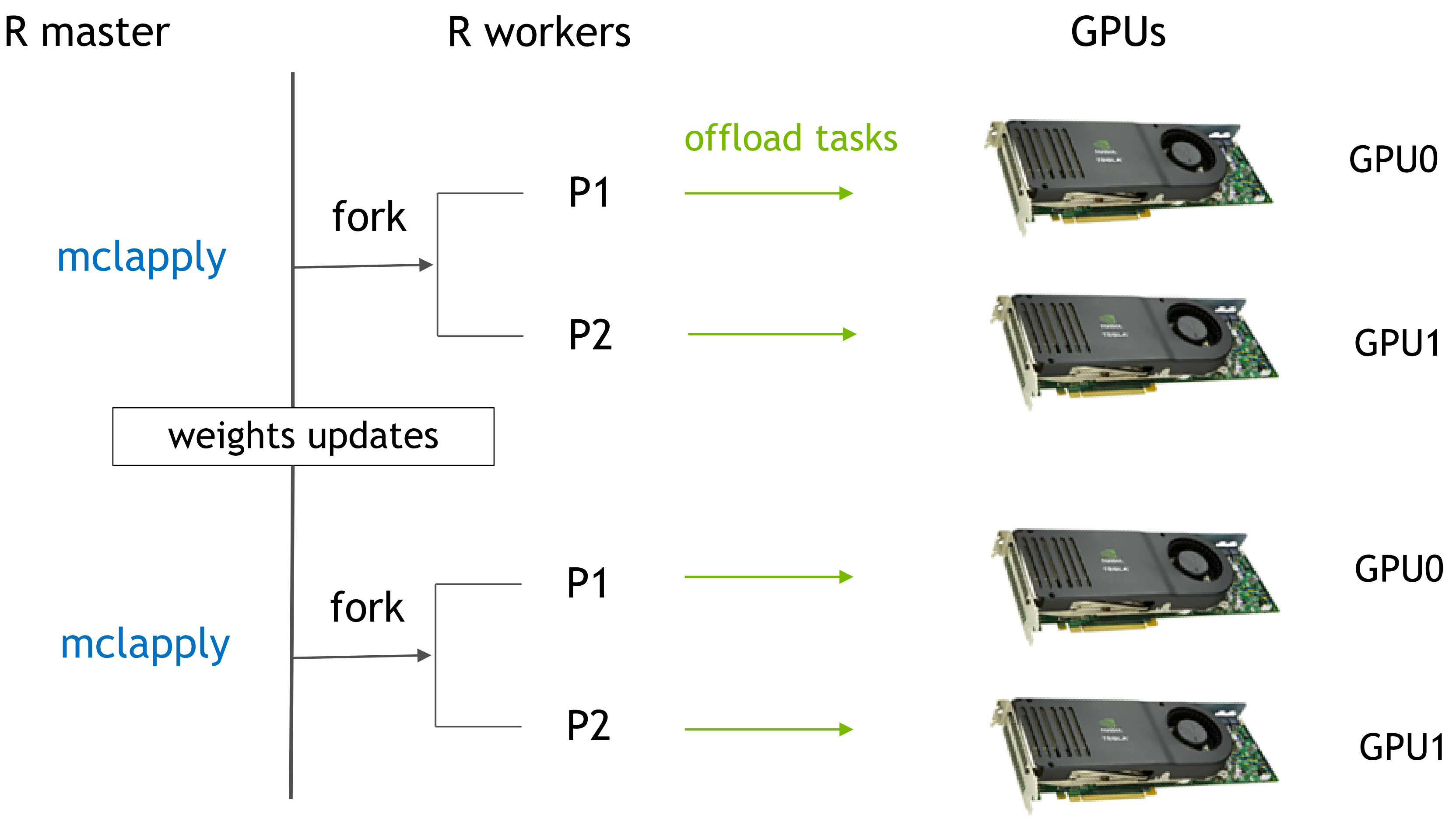
HOGWILD!

- A lock-free approach to parallelizing stochastic gradient descent
- MapReduce-like parallel-processing framework

DNN Training

- Launch several workers
- Each worker updates local weights/bias based on parts (1/N) of data
- Master collects and average all weights/bias from each worker
- Each worker update its weights/bias

Extend ‘multicores’ solution to multiGPUs



DATA DECOMPOSITION

- mclapply function to map data into each R processor

```
# Parallel Training
res <- mclapply(1:devNum, function(id) {
        train.dnn.cublas(x, y, omodel=para.model,
                         taindata=traindata[N.start[id]:N.end[id],],
                         devType="GPU", devID=(id-1), . . .) },
         mc.cores=devNum, mc.preschedule=TRUE)

# Model Updata
para.model <- list( W1= W1.sum/devNum, b1= b1.sum/devNum,
                    W2= W2/si,/devNum, b2= b2.sum/devNum)
```

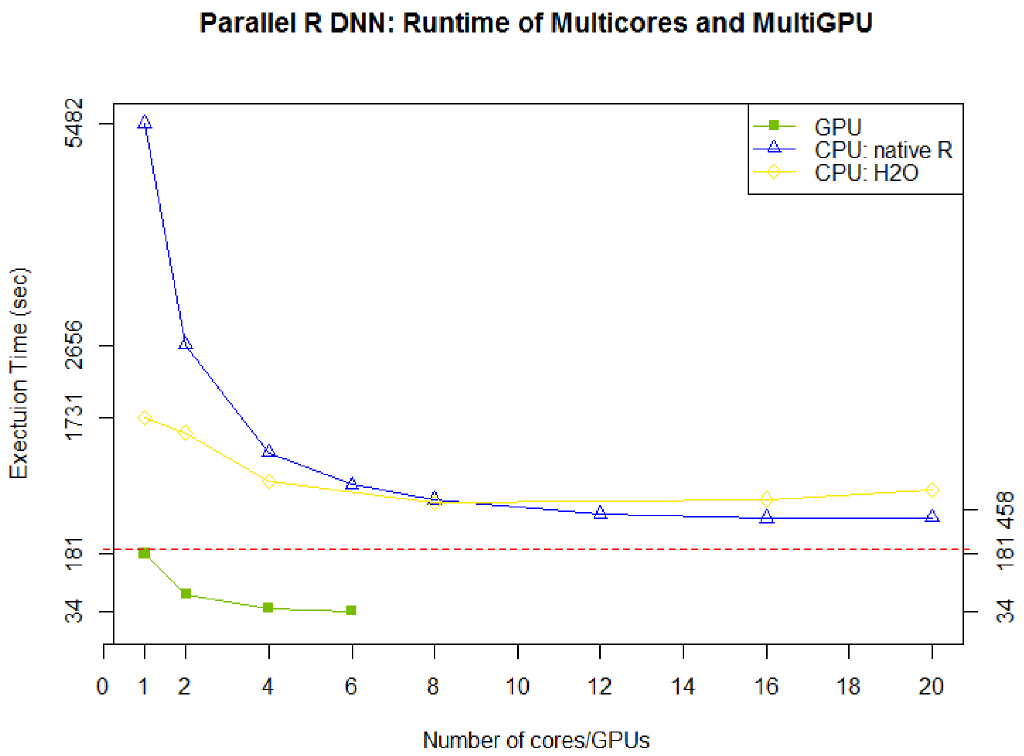
OFFLOAD TASKS TO GPUS

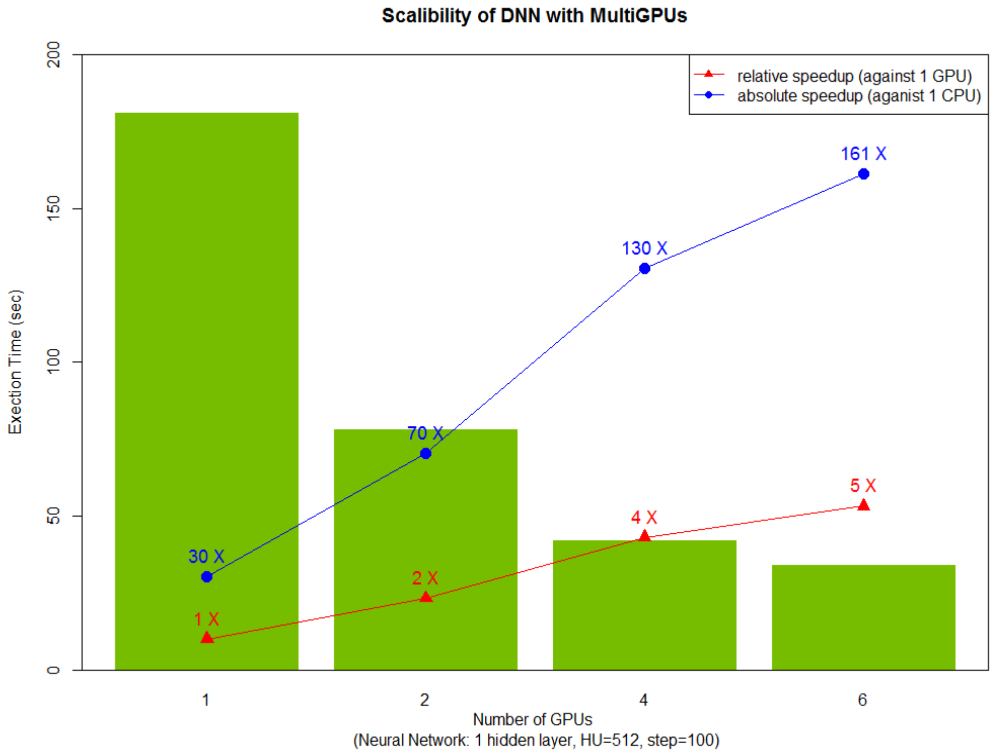
- Explicitly call cuBLAS API and pmax.cuda functions
- Set the GPU ID based on R's thread ID

```
# R level function call
res <- cuBLAS(hidden.layer, dscores, transA=T, devID=devID)

// GEMM cuda call by .CAll format and simplified for DNN
SEXP gemm_cuda(SEXP A, SEXP B, SEXP transA, SEXP transB, SEXP devID)
{   // init  . . .
    cudaSetDevice(gpuID);
   // cuBLAS: double precision matrix multiplication, DGEMM
   cublasDgemm(handle, cuTransA, cuTransB, mt, nt, kt, . . .);
   . . .
}
```

PERFORMANCE IMPROVEMENTS





CPU: Ivy Bridge E5-2690 v2 @ 3.00GHz, dual socket 10-core, 128G RAM; GPU: NVIDIA K40m, 12G RAM

SUMMARY

In this talk, we introduce solutions for HPA in R to

- Keep flexibility
- Achieve high speedup for native R code
- Extend multicore solution to multiGPUs
- Easy to apply these methods to multiple NN & other R algorithms

Further Works:

- Memory Optimizations
- Data Dependency Analysis
- Heterogeneous Computing both in CPU and GPU

Related Materials:

CODES:

All codes, scripts and window templates in this talk in here

TALKS:

- GTC15: Accelerate R by CUDA, slide
- GTC16: Data Science Applications of GPUs in the R Language

BLOG:

- Parallel FORALL, post
- ParallelR, R For Deep Learning:

(I) Build Fully Connected Neural Network From Scratch

(II) Achieve High-Performance DNN With Parallel Acceleration

(III) CUDA Acceleration And MultiGPUs Training

GPU TECHNOLOGY CONFERENCE

April 4-7, 2016 | Silicon Valley

THANK YOU

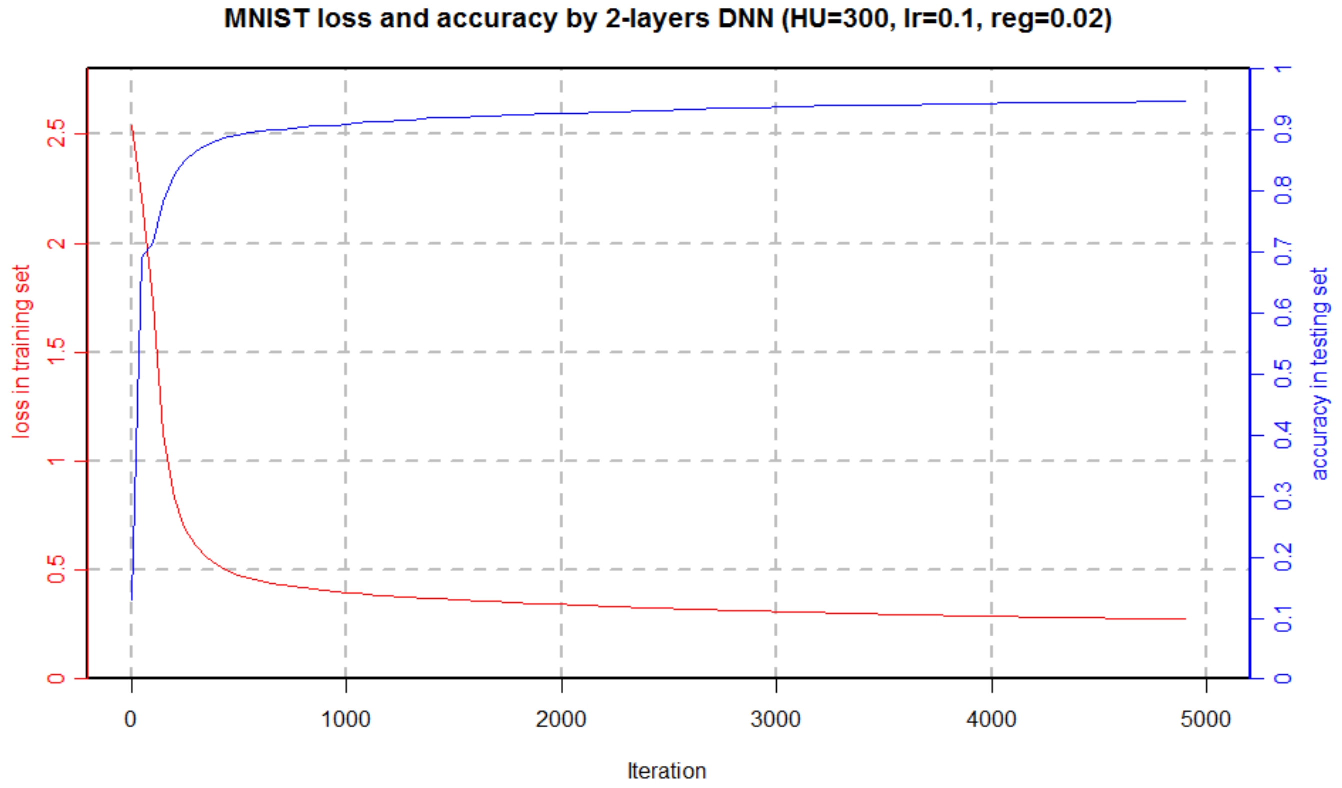
JOIN THE CONVERSATION

#GTC16

PRESENTED BY

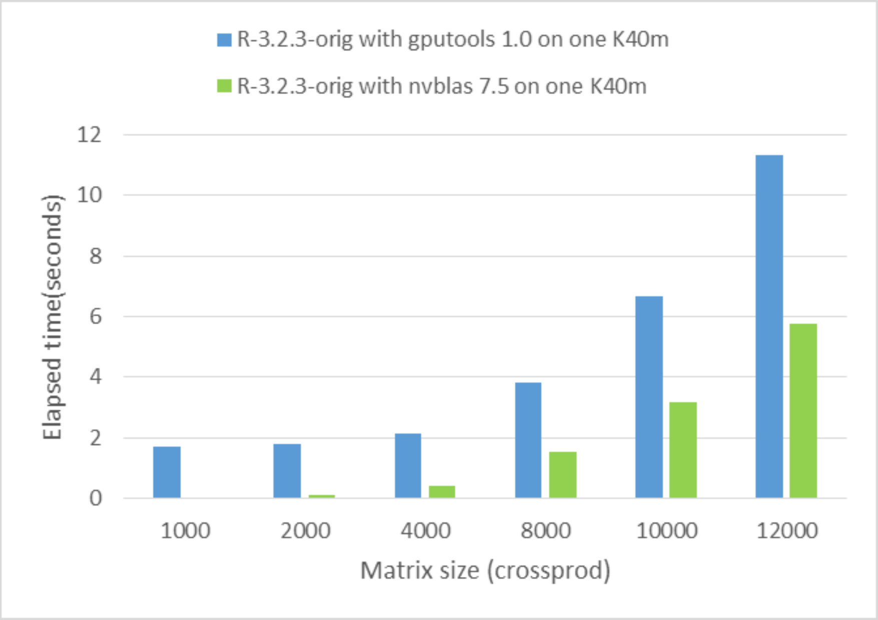
NVIDIA

TRAINING RESULTS



NOTE: Just to show the correctness of our codes and methods rather than achieve high accuracy of MNIST

gputools .vs. nvblas




```
Total seconds: time spent in function and callees.
Self seconds: time spent in function alone.

     %       total       %        self
 total     seconds     self    seconds      name
 100.0        4.64      0.0       0.00      "eval"
 100.0        4.64      0.0       0.00      "gpuCrossprod"
 100.0        4.64      0.0       0.00      "source"
 100.0        4.64      0.0       0.00      "system.time"
 100.0        4.64      0.0       0.00      "withVisible"
  84.5        3.92     84.5       3.92      ".Call"
  15.5        0.72     15.5       0.72      "t.default"
  15.5        0.72      0.0       0.00      "t"

     %        self       %       total
  self     seconds    total    seconds     name
  84.5        3.92     84.5       3.92      ".Call"
  15.5        0.72     15.5       0.72      "t.default"
```

```
Total seconds: time spent in function and callees.
Self seconds: time spent in function alone.

     %       total       %        self
 total     seconds     self    seconds      name
 100.0        2.18    100.0       2.18      "<Anonymous>"
 100.0        2.18      0.0       0.00      "crossprod"
 100.0        2.18      0.0       0.00      "eval"
 100.0        2.18      0.0       0.00      "source"
 100.0        2.18      0.0       0.00      "standardGeneric"
 100.0        2.18      0.0       0.00      "system.time"
 100.0        2.18      0.0       0.00      "withVisible"

     %        self       %       total
  self     seconds    total    seconds     name
 100.0        2.18    100.0       2.18      "<Anonymous>"
```