S3443 - Clicking GPUs into a Portable, Persistent and Scalable Massive Data Framework

Rob Farber

Chief Scientist, BlackDog Endeavors, LLC
Author, “CUDA Application Design and Development”
Research consultant: ICHEC, Fortune 100 companies, and others

Scientist: Los Alamos National Laboratory, Berkeley Lab, Pacific Northwest National Laboratory, Santa Fe Institute

Dr. Dobb’s Journal CUDA & OpenACC tutorials
- OpenCL “The Code Project” tutorials
- Columnist Scientific Computing, and other venues
int main() {
    cout << "Hello World" << endl;
    // load data and initialize parameters
    init();
    #pragma acc data \
        copyin(param[0:N_PARAM-1]) \
        pcopyin(example[0:nExamples*EXAMPLE_SIZE-1])
    {
        optimize( objFunc ); // the optimizer calls the objective function
    }
    return 0;
}

double objFunc( ... )
{
    double err=0.;
    #pragma acc parallel loop reduction(+:err)
    #pragma omp parallel for reduction(+ : err)
    {
        err = 0.;
        for(int i=0; i<nExamples; i++) {
            // transform
            float d=myFunc(i, param, example, nExamples, NULL);
            //reduce
            err += d*d;
        }
    }
    return sqrt(err);
}
A general mapping: \( \text{energy} = \text{objFunc}(p_1, p_2, \ldots p_n) \)

**Optimization Method**

(Powell, Conjugate Gradient, Other)

**Step 1**
Broadcast parameters

**Step 2**
Calculate partials

**Step 3**
Sum partials to get energy

- **GPU 1**
  - \( p_1, p_2, \ldots p_n \)
  - Examples 0, N-1

- **GPU 2**
  - \( p_1, p_2, \ldots p_n \)
  - Examples N, 2N-1

- **GPU 3**
  - \( p_1, p_2, \ldots p_n \)
  - Examples 2N, 3N-1

- **GPU 4**
  - \( p_1, p_2, \ldots p_n \)
  - Examples 3N, 4N-1

**Host**
Optimize an “objective function”
Exascale Capable!

Applicable to a general class of optimization problems

- Locally Weighted Linear Regression (LWLR)
- Neural Networks
- Naive Bayes (NB)
- Gaussian Discriminative Analysis (GDA)
- k-means
- Logistic Regression (LR)
- Independent Component Analysis (ICA)
- Expectation Maximization (EM)
- Support Vector Machine (SVM)
- Others: (MDS, Ordinal MDS, etcetera)
See a path to exascale
(MPI can map to hundreds of GPUs)

- Over 350TF/s of performance on Longhorn (including communications!)
- Dominant runtime of code that scales to 500 GPUs
- 600+ GF/s per K20
- Expect many petaflop/s average performance from Titan

Always report “Honest Flops”

$$\text{EffectiveRate} = \frac{\text{TotalOpCount}}{T_{\text{broadcast}} + T_{\text{objectfunc}} + T_{\text{reduce}}}$$
Lessons learned from CUDA

CUDA was first introduced in February 2007

- People like it. Taught worldwide: Stanford, Oxford, Beijing, ...
- A strong scaling execution model is essential
  - Programming with 10,000 or even a million threads is no big deal
- Use them to build the worlds fastest supercomputers
  - Titan, Tehane-1, Nebulae, ...
  - Manufacturers sell teraflop capable supercomputers to anyone for a few hundred dollars.

What is GPU computing good for?

Is that enough?
NLPCA (Nonlinear PCA)

$G() = G()$

2x10x10x10x2 autoencoder

Yes, performance did increase slightly
Love those SFUs! *(Special Function Units)*

- Fast transcendental functions
  - The world is nonlinear ... so are many computational models
Data handling can take as much time as the computational problem!

• Longhorn GPU capabilities
  – 2,048 GB of GPU memory in 512 Quadro FX 5800 GPUs

• ORNL Titan
  – 112,128 GB of GPU memory in 18,688 K20x GPUs

Expect 600+ GF/s per device

*big number* here

Average sustained performance
Fast scalable data loads via parallel file systems

Each MPI client on each node:
1. Opens file
2. Seeks to location
3. Reads data
4. Close
Data generation and pre-processing via a “click together” hybrid framework

Run on a workstation, across a LAN, via the WAN, or within the cloud

1. Fast and Scalable
2. Heterogeneous
3. Flexible and Robust

The subject of my Doctor Dobb’s Journal article, “Using NVCC for Big-Data Heterogeneous Plug-in Based Applications and Workflows”

• Covers lessons learned from using this framework at US National Laboratories, commercially, and in my drug discovery startup.
Framework must be language agnostic

- Use the highest level interface first
  - Delve down into lower level programming when
    - You need higher performance
    - The high level API does not do what you want

- Use a single source tree
OpenACC portability

C

/* matrix-acc.c */
int main()
{
    ...

    // Compute matrix multiplication.
    #pragma acc kernels copyin(a, b) copy(c)
    for (i = 0; i < SIZE; ++i) {
        for (j = 0; j < SIZE; ++j) {
            for (k = 0; k < SIZE; ++k) {
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }
    return 0;
}

Fortran

! matrix-acc.f
! program example1
! ...
!$acc data copyin(a, b) copy(c)
!$acc kernels loop
! Compute matrix multiplication.
! do i=1, n_size
!     do j=1, n_size
!         do k = 1, n_size
!             c(i,j) = c(i,j) + a(i,k) * b(k,j)
!             enddo
!         enddo
!     enddo
!$acc end data
! end program example1

C++

int main()
{
    cout << "Hello World" << endl;

    // load data and initialize parameters
    init();

    #pragma acc data \
        copyin(param[0:N_PARAM-1]) \ 
        pcopyin(example[0:nExamples*EXAMPLE_SIZE-1])
    {
        optimize( objFunc ); // the optimizer calls the objective function
    }
    return 0;
}

Coprocessor and GPU demos shown at SC12 by PGI and CAPS
Fast and scalable heterogeneous workflows

Full source code in my DDJ tutorial
http://www.drdobbs.com/parallel/232601605
Header fundamentals

- A stream of packets using a constant sized header
  - Seekable streams and mmap() can avoid bandwidth limitations
  - Easily compressed with snappy (google), gzip, bzip2, ...
Header fundamentals

• Machine independent
  • Always encoded in network standard order
• Version number: Essential on a per message basis
  – Streams are persistent
    • Version numbers identify format changes
    • I still use data created in the 1980s
  – Applications can transparently utilize old and new versions
Header fundamentals

- **Message size:** *Essential to speed*
  - Allows `malloc()` followed by `read()`, which is fast
- **Duplicating the size** is *essential for robustness!*
  - The size is a single point of failure per packet
    - Persistent streams can experience bit rot, truncation, I/O errors, ...
    - I have seen machine failures send valid TCP information containing garbage data!
      - Long running workflows using many supercomputers and numerous workstations
  - Once the message is read, more sophisticated error correction can happen even if an error is detected
  - If a message is corrupt, all is not necessarily lost
    - The remainder of the stream may still be intact (or at least usable/recoverable)
Header fundamentals

• **Message type:** *Identifies the type of the message*
  – Protobufs, raw vectors and arrays, etcetera
  – I still use streams created in the 1980s

• Use protobufs or other source generator to create the serialization methods
  – Flexible
Data packets can be any format

Three methods are statically or dynamically linked

If( init() ) {write header and message}
while ( read the binary header information == SUCCESS)
{
- Compare header sizes (a mismatch flags an unrecoverable error)
- Allocate size bytes (after converting from network standard byte order)
- Binary read of size bytes into the allocated memory.
  if( func(msg) ) {
    // perform the write
    - Binary write the header in network standard byte order
    - Binary write the packet information
  }
If( fini() ) {write header and message}
package tutorial;

define Packet {
    UNKNOWN = 0;
    PB_VEC_FLOAT = 1;
    PB_VEC_DOUBLE = 2;
    PB_VEC_INT = 3;
}

message FloatVector {
    repeated float values = 1 [packed = true];
    optional string name = 2;
}

message DoubleVector {
    repeated float values = 1 [packed = true];
    optional string name = 2;
}

*Used in my Doctor Dobb’s tutorial

Supporting language projects

Action Script: http://code.google.com/p/protobuf
C: http://code.google.com/p/protobuf/c/
C: http://koti.kapsi.fi/jpap/protobuf
C++: http://protobuf.googlecode.com/Google-official-implementation
C/C++: http://spbc.sf.net/
C#: http://code.google.com/p/protobuf-csharp
C#: http://code.google.com/p/protobuf-csharp-port
Java: http://code.google.com/p/protobuf-java/
Java ME: http://github.com/ponderingpanda/protobuf-j2me
Lua: http://code.google.com/p/protobuf-lua/
D: http://256.makerslocal.org/wiki/index.php/ProtocolBuffer
Erlang: http://github.com/gerakines/erlang_protobuffs/tree/master
Ruby: http://github.com/mozy/ruby-protobuf/
Scala: http://github.com/jeffplaisance/scala-protobuf
Vala: https://launchpad.net/protobuf-vala
Visual Basic: https://github.com/bmizerany/beefcake/tree/master/lib/beefcake

Go: http://code.google.com/p/protobuf/
Haskell: http://hackage.haskell.org/package/hprotoc
Java: http://github.com/ponderingpanda/protobuf-j2me
JavaScript: http://github.com/sirikata/protobuf
Lua: http://github.com/indygreg/lua-protobuf
Mercury: http://code.google.com/p/protobuf-mercury/
Objective C: http://code.google.com/p/protobuf

Perl: http://groups.google.com/group/protobuf-perl
Python: http://code.google.com/p/protobuf-python/
Python: http://github.com/mozyst/ruby-protobuf-buffers
Ruby: http://github.com/bmizerany/beefcake/tree/master/lib/beefcake
Scala: http://github.com/jpfplaisance/scala-protobuf
A C vector summation example

```c
extern "C"
#ifdef _WIN32
__declspec(dllexport)
#endif
char* init(const char* proiname, const char* sourcename,
uint32_t*size, uint32_t*type) {
    return(NULL);
}

extern "C"
#ifdef _WIN32
__declspec(dllexport)
#endif
char* fini(const char* proiname, const char* sourcename,
uint32_t*size, uint32_t*type) {
    return(NULL);
}

extern "C"
#ifdef _WIN32
__declspec(dllexport)
#endif
void dynFree(char* pt) {
    if(pt) delete [] pt;
}
```

//reduction.cc (Rob Farber)
extern "C"
#ifdef _WIN32
__declspec(dllexport)
#endif
char* func(const char* proiname, const char* sourcename,
uint32_t*size, uint32_t*type, char*blob) {
    switch(*type) {
    case tutorial::PB_VEC_FLOAT: {
        tutorial::FloatVector vec;
        if(!vec.ParseFromArray(blob,*size)) {
            cerr << proiname << "," << sourcename << "Illegal packet" << endl;
        } else {
            if(vec.has_name() == true) cerr << "vec_float" << vec.name() << " 
sum = 0.f;
for(int i=0; i < vec.values_size(); i++) sum += vec.values(i);
    cerr << "sum of vector" << sum << endl;
    cerr << "tlast value in vector is " << vec.values(vec.values_size()-1) << endl;
    cerr << "tvector size is " << vec.values_size() << endl;
    } }
break;
default:
    cerr << "Unknown packet type" << endl;
}
return(NULL);
```

init() // does nothing

fini() // does nothing

func() // performs the summation
Strive to limit data copies

Protobufs
1. Read blob from stream/call func
2. Unpack blob in func()
3. Copy to host buffer
4. Transport data to GPU

Better
1. Read blob from stream/call func()
2. Pass to GPU

Best
1. Read smart pointer from stream/call func()
2. Use GPUdirect to get data to GPU

S3266 - GPU Direct Support for RDMA and Green Multi-GPU Architectures, Dustin Franklin, GE
Why __declspec and other cruff?

- Dynamic plugins:
  - *dynFunc.exe reduction* (Windows)
  - *dynFunc reduction* (Linux)

A stream on one machine:

```
dynFunc vec2x < stream.dat | dynFunc reduction.cc
```

A stream across three machines:

```
dynFunc vec2x < stream.dat |
  | ssh machine1 dynFunc app1.cu | dynFunc app2.so |
  | ssh machine2 dynFunc reduction.c
```

Use CUDA, OpenCL and OpenACC as scripting languages!
Scalable collaborations require dynamic compilation!

Without dynamic plugins

**No scalability**
Collaborators need:
• All plugins
• For all machine types

With dynamic plugins

Full source for Windows and Linux in tutorial 23:
http://www.drdobbs.com/parallel/232601605

Only needs source for the required plugins
Flexible and Robust

• Flexibility:
  – Use any machine or collection of machines
    • (Local, LAN, WAN, Cloud)
  – Use your own code – in your own language
    • GPU acceleration!
  – CUDA as a dynamically compiled RAD (Rapid Application Development) language

• Application robustness
  1. Most workflows use “known working” code
     • Just a few different modules
  2. Stream integrity
Objective functions in Bioinformatics

NetTalk
Sejnowski, T. J. and Rosenberg, C. R. (1986)
NETtalk: a parallel network that learns to read aloud, Cognitive Science, 14, 179-211

"Applications of Neural Net and Other Machine Learning Algorithms to DNA Sequence Analysis",

The phoneme to be pronounced

T|F Exon region

Internal connections

t e x t

Internal connections

A C G T T
Predicting binding affinity
(The closer you look the greater the complexity)
How do we know you are not playing expensive computer games with our money?

Validation is essential
1995 drug design hardware vs 2013
(analyzed all available chemical databases ... TB of data)

• Quad-core 512 MB Sun workstation
  – My Samsung S3 is more powerful and has 2 GB RAM

• 80 GB disk and a TB DLT tape stacker
  – A TB laptop hard drive

• 60 Gflop/s Connection machine
  – A mobile GeForce GPU

You can change the world!
$30M of hardware replaced by a GPU accelerated laptop
Worked great on 20th century hardware

Full source code in my DDJ tutorial
http://www.drdobbs.com/parallel/232601605

Developed in 1980s by Farber
1995 Processed all available chemical databases, 2013 ....
Binding affinity required a blind test

Binding affinity for a specific antibody

Possible hexamers

$20^6 = 64M$

1k – 2k *pseudo*-random (hexamer, binding) affinity pairs

Approx. 0.001% sampling

Hill climbing to find high affinity

Learn:
\[ \text{Affinity}^{\text{Antibody}} = f(A_0, \ldots, A_5) \]

Predict P,C,T,N,S,L has the highest binding affinity

Confirm experimentally

\[ f(x, y) = e^{-(x^2 + y^2)} \]
Time series

Learn:

\[ X_{t+1} = f(X_t, X_{t-1}, X_{t-2}, \ldots) \]

Iterate

\[ X_{t+1} = f(X_t, X_{t-1}, X_{t-2}, \ldots) \]
\[ X_{t+2} = f(X_{t+1}, X_t, X_{t-1}, \ldots) \]
\[ X_{t+3} = f(X_{t+2}, X_{t+1}, X_t, \ldots) \]
\[ X_{t+4} = f(X_{t+3}, X_{t+2}, X_{t+1}, \ldots) \]

Internal connections

Works great (Lapedes and Farber 1987)
TF/s devices open the door to new topics

- Works great for manufacturing optimization
  - Best product for lowest cost of materials
  - Works great for color matching
- Multiterm objective functions
  - Best design for the lowest (cost, weight, \{your metric here\}, ...)
  - A teraflop/s per device can run many optimizations to map the decision space.
- Machine learning with memory or variable inputs 😊
  - Recurrent neural networks: have to iterate the network during training
- Realtime processing:
  - IIR filters, ....

You can change the world!
Big data social media

• Need a simplifying framework
  – A laptop can represent a billion node graph
  – People don’t understand billion node graphs!
    • Million node graphs are not comprehensible
    • Thousand node graphs are too complex
    • Hundred node graphs are still too big
    • A few to tens of nodes are potentially understandable

Validate against 3rd party experts and machine metrics
• Understand this is a lens looking into a social reality
• Cannot forget that the computer only represents reality!
PCA (Principle Components Analysis)

• Widely used in data-mining and data reduction
  – Discuss a method proposed by Sanger (1989)

• Extends to Nonlinear PCA (NLPCA)

The general mapping scales according to data

• Exascale capable
• 18,688 GPUs on Titan
• Got 112 TB of data for just the GPUs?
Dimension reduction

• The curse of dimensionality
  – People cannot visualize data beyond 3D + color
  – Search volume rapidly increases with dimension
    • Queries return too much data or no data
Thought-leaders: a power law concept

- Thought-leaders are defined as agents in a social network (nodes in a graph) who have a disproportionately large agency on the whole network.
  - Broadly, if one views a social media post as a stimulus then the node’s *agency* is a measure of the amount of response it induces.
  - Quantitative measures of this are the number of comments a blog post receives, the number of “mentions” (or retweets in Twitter parlance) of a user’s micro-blog status update, or the number of links to and from a blogger.
- Motivated by the analyst’s need to efficiently survey a deluge of data and extract priority leaders to inform robust courses of action.
Triangulate Three Metrics to Characterize a Thought-leader (think Nielson Rating)

• Utilizing simplest graph metric: all variants on degree centrality.
• Intuition underlying the metrics:
  • (metric: number of posts): Thought-leaders are active post authors - but is anybody listening?
  • (metric: comments per post): Post author stimulates a response - but is this response a flash in the pan or do they regularly get a response?
  • (metric: comments per post author): The overall response a post author has stimulated
Spectrum of graph Laplacian matrix

How does spectrum depend on “key” nodes?
- Removal of \textit{high-degree} nodes should lower maximum eigenvalue

How do Thought-leaders contribute to spectrum of a Social Media graph?
- Test! (Next slide)
Clearly see a drop in the highest ranked Laplacian eigenvalue

**Atlanta Flood Tweet Graph Largest Laplacian Eigenvalues**

- Blue diamonds: Randomly removed from all
- Red squares: Randomly removed from top half rank
- Green triangles: "Removed by Task 14 metrics only"

**Flu Tweet Graph Largest Laplacian Eigenvalues**

- Blue diamonds: Removed by metrics
- Red squares: Randomly removed from all
- Green triangles: Removed randomly from top half rank
Evaluating runtime performance and accuracy

Massive Social Network Analysis: Mining Twitter for Social Good

David Ediger Karl Jiang
Jason Riedy David A. Bader
Georgia Institute of Technology
Atlanta, GA, USA

Courtney Corley Rob Farber
Pacific Northwest National Lab
Richland, WA, USA

William N. Reynolds
Least Squares Software, Inc.
Albuquerque, NM, USA

Abstract—Social networks produce an enormous quantity of data. Facebook consists of over 400 million active users sharing over 5 billion pieces of information each month. Analyzing this vast quantity of unstructured data presents challenges for software and hardware. We present GraphCT, a Graph Characterization Toolkit for massive graphs representing social network data. On a 128-processor Cray XMT, GraphCT estimates the betweenness centrality of an artificially generated (R-MAT) 537 million vertex, 8.6 billion edge graph in 55 minutes. We use GraphCT to analyze public data from Twitter, a microblogging network. Twitter’s message connections appear primarily tree-structured as a news dissemination system. Within the public data, however, are clusters of conversations. Using GraphCT, we can rank actors within these conversations and help analysts focus attention on a much smaller data subset.

average of 120 ‘friendship’ connections each and sharing 5 billion references to items each month [11].

One analysis approach treats the interactions as graphs and applies tools from graph theory, social network analysis, and scale-free networks [28]. However, the volume of data that must be processed to apply these techniques overwhelms current computational capabilities. Even well-understood analytic methodologies require advances in both hardware and software to process the growing corpus of social media.

Social media provides staggering amounts of data. Extracting knowledge from these volumes requires automation. Computing quickly over this data is a challenge for both algorithms and architectures.

We present GraphCT, a Graph Characterization...
Sociolects (finding people by word usage)

- Myocardial infarction
- Infection
- Antibiotic
- Far out
- cool
- Hey man
- thread
- warp
- CUDA
Look at some large data sets

<table>
<thead>
<tr>
<th>Name</th>
<th>Language</th>
<th>Terms</th>
<th>Tweets Harvested</th>
<th>Distinct Posters (Nodes)/Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>English</td>
<td>20</td>
<td>1,127,895,980</td>
<td>20,325,708/30,411,053</td>
</tr>
<tr>
<td>Real N</td>
<td>Spanish</td>
<td>63</td>
<td>831,799</td>
<td>714,214/665,231</td>
</tr>
<tr>
<td>Real Sy</td>
<td>Arabic</td>
<td>20</td>
<td>27,159,300</td>
<td>237,538/615,397</td>
</tr>
<tr>
<td>Real B</td>
<td>Arabic</td>
<td>17</td>
<td>9,310,475</td>
<td>149,488/333,962</td>
</tr>
<tr>
<td>Real Sh</td>
<td>English</td>
<td>28</td>
<td>279,829,564</td>
<td>6,894,309/6,196,719</td>
</tr>
</tbody>
</table>

**Newman assortativity**

Broken by population effects
- Newman's r is very dependent on the size of the different classes
- Classes with large numbers of nodes will typically dominate

**Assortativity as Probability: Edge Density**
(Insight by Reynolds and Salter)
Assortativity as Probability: Edge Density

Normalized mixing matrix \( e = \frac{M}{||M||} \).

- An estimate of a probability distribution – it sums to 1. It assumes the existence of an edge.
- It is the probability that, given an edge, that it will connect a node in group \( i \) to group \( j \).

\[ e = \frac{M}{||M||} = P(\text{source } \epsilon \ i, \text{destination } \epsilon \ j \mid \text{Edge}(i,j) ) \]

Given the probability of group membership function \( P(i) \), expressions can be developed for other probability density functions.

\[ P(\text{Edge}(i, j) \mid s \epsilon \ i, d \epsilon \ j) = \frac{P(s \epsilon i, d \epsilon j \mid \text{Edge}(i, j))P(\text{Edge}(i, j))}{P(s \epsilon i)P(d \epsilon j)} \]

\( P(\text{Edge} \mid i,j) \) is an interesting quantity – it is the probability of an interaction given that source is in \( i \) and dest is in \( j \).

It is a quantity that applies to a network as a whole (compared to quantities like average degree, which also apply to nodes within a network)

Only unknown quantity is \( P(\text{Edge}(i,j)) \) – the probability in a network of finding an edge between two arbitrary nodes. A consistent expression for this is given by:

\[ P(\text{Edge}(i, j) \mid s \epsilon i, d \epsilon j) = \frac{N_{ij}}{N_i N_j}, \quad P(\text{Edge}(i, j)) = \frac{N_E}{N_N^2} \]

Where \( N_E \) is the number of edges in the network and \( N_N \) is the number of nodes. This ratios are the Edge Density: the number of observed edges, to the number of possible edges (pairs) in the network, allowing for self-connection.

- Expect drop in population to be geometric (fastest)
- For a sociolect, we expect the drop to be sub-geometric (slower)
Cut to the finish... a clear signal

Ratio of Observed vs Expected Populations Under Independence Hypothesis: $\text{Pop}(N_T) = P_T \text{Pop}(N_T-1)$

Minimum TC TCt

Real
GPUs are great big data visualization supercomputers!

**Primitive restart:**
- A feature of OpenGL 3.1
- Roughly 60x faster than optimized OpenGL
- Avoids the PCIe bottleneck
- Variable length data works great!

LiDAR: 131M points 15 – 33 FPS (C2070)

Chapter 9 Perlin Noise
Fly around in a 3D virtual world

In collaboration with Global Navigation Sciences (http://www.globalnavigationsciences.com)
The entire segmentation method
__global__ void kernelSkin(float4* pos, uchar4 *colorPos,
    unsigned int width, unsigned int height,
    int lowPureG, int highPureG,
    int lowPureR, int highPureR)
{
    unsigned int x = blockIdx.x*blockDim.x + threadIdx.x;
    unsigned int y = blockIdx.y*blockDim.y + threadIdx.y;
    int r = colorPos[y*width+x].x;
    int g = colorPos[y*width+x].y;
    int b = colorPos[y*width+x].z;
    int pureR = 255*((float)r)/(r+g+b);
    int pureG = 255*((float)g)/(r+g+b);
    if( !(pureG > lowPureG) && (pureG < highPureG)
    && (pureR > lowPureR) && (pureR < highPureR))
    colorPos[y*width+x] = make_uchar4(0,0,0,0);
}
Interactive 100+M LiDar data points!

- **Worst case**: each data point is recalculated
  - Useful for onboard triangulation
  - Custom metrics
  - Etcetera
- With a simple modification of the Chapter 9 “CUDA Application Design and Development” example code
TF/s devices + Big data
(So much more I can say)

• Works great for manufacturing optimization
  – Best product for lowest cost of materials
  – Works great for color matching
• Multiterm objective functions
  – Best design for the lowest (cost, weight, {your metric here}, ...)
  – A teraflop/s per device can run many optimizations to map the decision space.
• Machine learning with memory or variable inputs 😊
  – Recurrent neural networks: have to iterate the network during training
• Realtime processing:
  • IIR filters, ....
Thank you!

Rob Farber  
Chief Scientist, BlackDog Endeavors, LLC  
Author, “CUDA Application Design and Development”  
Research consultant: ICHEC, Fortune 100 companies, and others

Scientist:  

Dr. Dobb’s Journal CUDA & OpenACC tutorials

- OpenCL “The Code Project” tutorials  
- Columnist Scientific Computing, and other venues