# Creative Consultants LLC

Efficient Performance Computing

Greg Scantlen, CEO

GTC 2012  May 15

## Contents

- Stelletto – cluster features in portable package
- CAPS-Enterprise , Francois Bodin, CTO
- Black Dog Endeavors, Rob Farber, CEO

# Supercomputing 2008 (SC08)

5 GPUs 1TF SP NBODY + NanoTube 3D Stereo

Stereoscopic Computational Microscope

# GITerDone GTC 2009

Switchless IB TRIAD Network

Head node netboots three compute nodes

Nine GPUs in Cluster using three Dual Port IB HCA

# SC09 “Stella” ConSTELLAtion 4.5TF SP NBODY SIM

Custom Modular Cases

Tesla Partner, nine C1060 GPUs

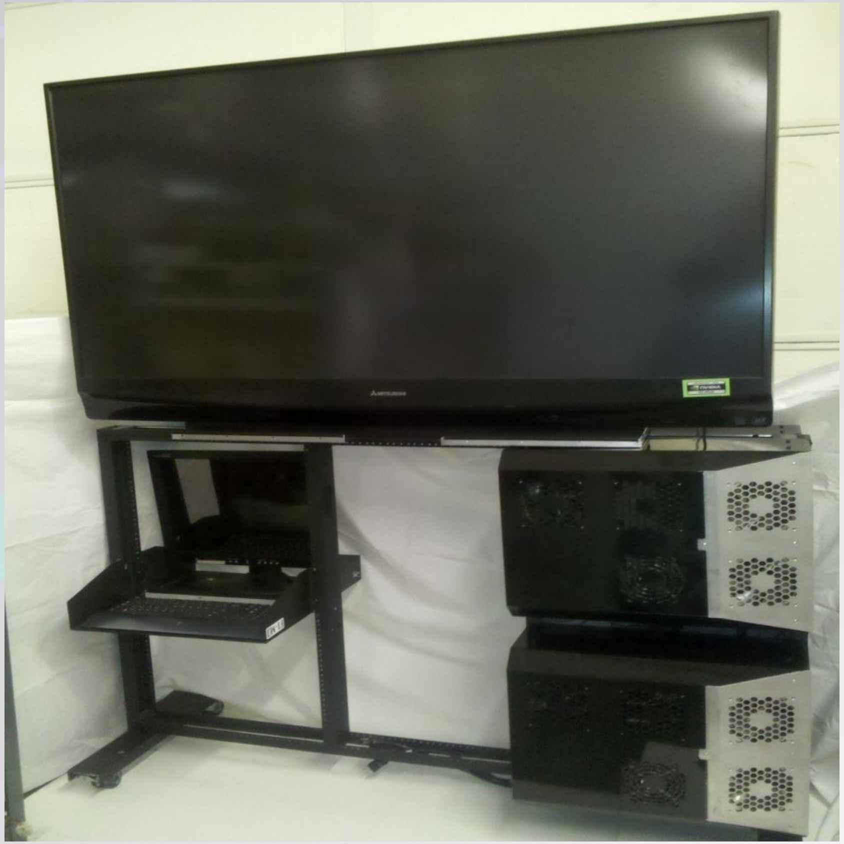
TRIAD IB Network

# SC09 Booth

Do you think GPUs will be important?

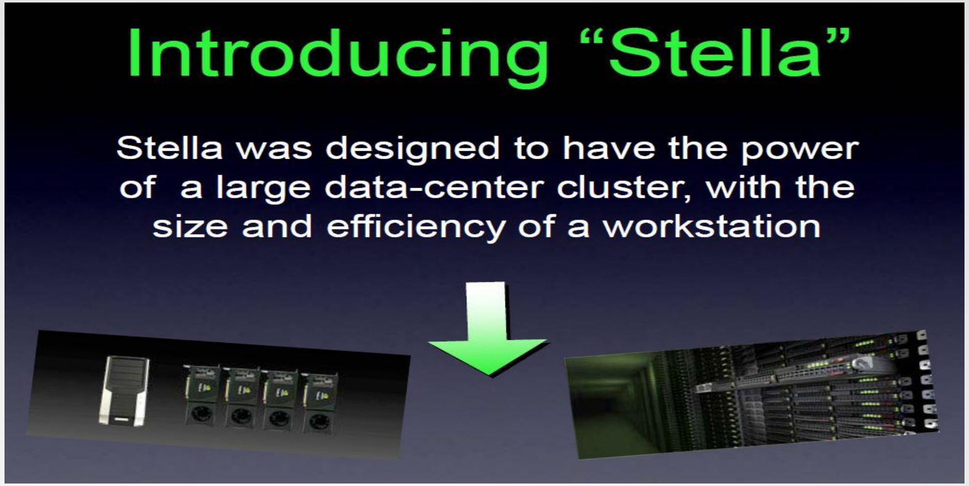
WHAT CAN I RUN ON IT?

## LAMMPS WorkShop Feb 2010



Stella modified with 60” Stereoscopic Display for Group Viewing of MD

Combined presentation with Axel Kohlmeyer



Introducing “Stella”

Stella was designed to have the power of a large data-center cluster, with the size and efficiency of a workstation

# GTC 2010

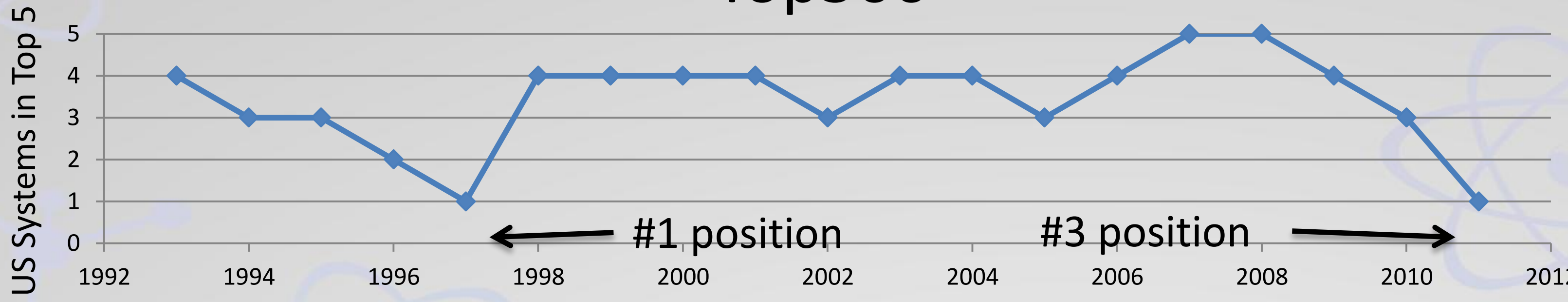
CAST & CREW:
BUCKY BALL Challenge
An interactive Molecular Dynamics demo created by Dr. Axel Kohlmeyer
Creative Consul
GPU

# Philly Science Festival   April 2011

PHILADELPHIA SCIENCE FESTIVAL
Temple University College of Science and Technology and Creative Consultants LLC:
The Buckyball Challenge
PROUD SPONSOR
TEMPLE

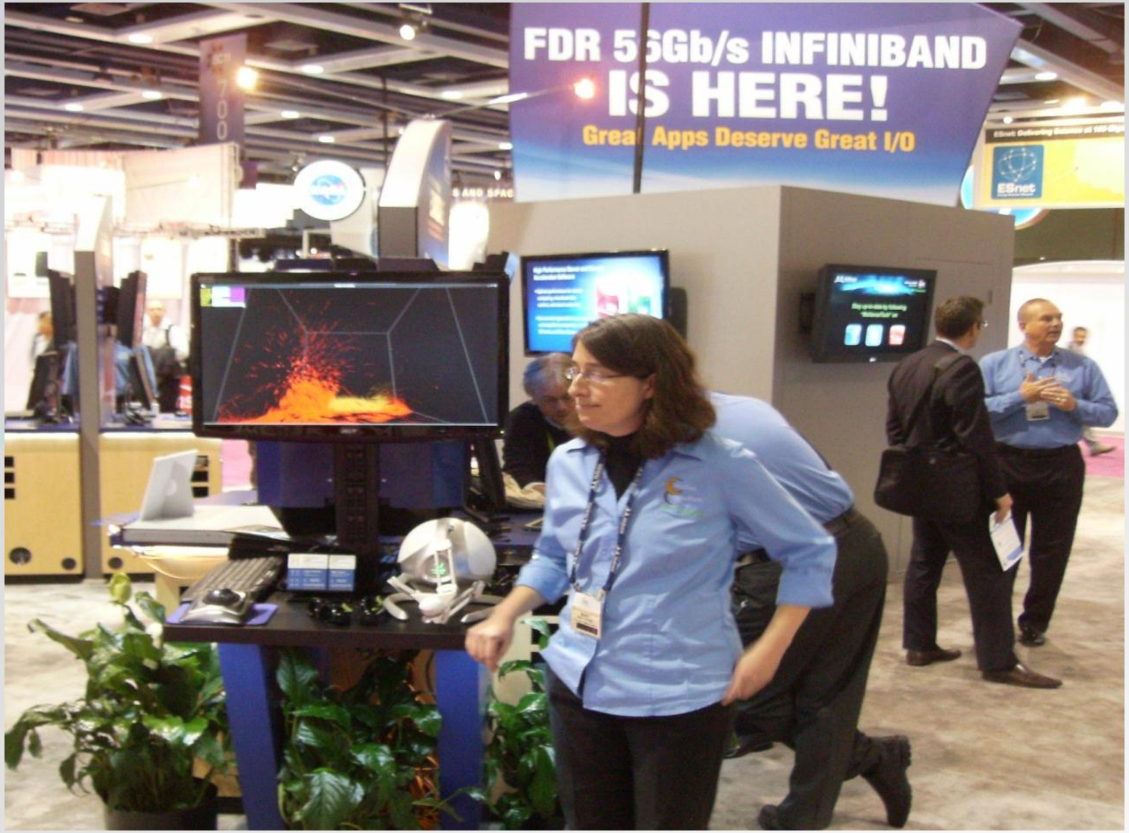
(Photos courtesy of Darryl W. Moran for the Philadelphia Science Festival)

# Top500



| June 2010 | | November 2010 | | June 2011 | |
|---|---|---|---|---|---|
| 1. Jaguar | 🇺🇸 | 1. Tianhe | 🇨🇳 | 1. K computer | 🇯🇵 |
| 2. Nebulae | 🇨🇳 | 2. Jaguar | 🇺🇸 | 2. Tianhe | 🇨🇳 |
| 3. Roadrunner | 🇺🇸 | 3. Nebulae | 🇨🇳 | 3. Jaguar | 🇺🇸 |
| 4. Kraken XT5 | 🇺🇸 | 4. TSUBAME 2.0 | 🇯🇵 | 4. Nebulae | 🇨🇳 |
| 5. JUGENE | 🇩🇪 | 5. Roadrunner | 🇺🇸 | 5. TSUBAME 2.0 | 🇯🇵 |
| *64. TSUBAME* | 🇯🇵 | 🟩 *Nvidia GPU's* | | 🟥 *SPARC64* | |

# SC11 Stelletto



Quiet

8x 16Core CPU

4x  Fermi GPUs

40Gbps IB

3D Stereo Display

# Stelletto Configurations

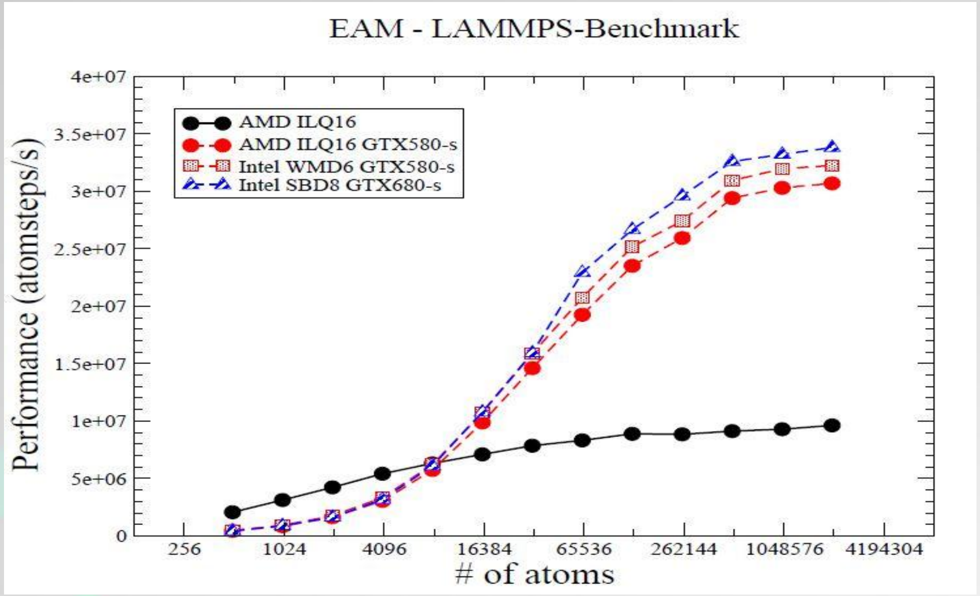
- 2x AMD Quad Socket Interlagos
- 4x PCIe 2.0 GPUs
- QDR InfiniBand

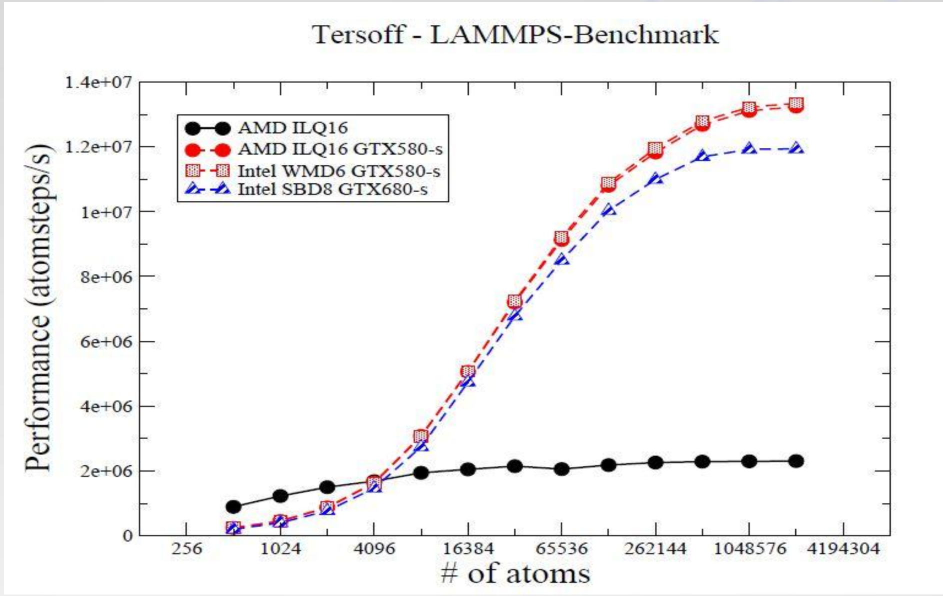
- 2x Intel Dual Socket Sandy Bridge
- 6x PCIe 3.0 GPUs
- FDR InfiniBand

## Hardware Comparison using LAMMPS

- Classical Molecular Dynamics simulation code
- Scales well on many nodes, used on simple workstation as well as 100K core systems
- One of the six main applications for Titan, GPU based successor of Jaguar

# Communications vs Compute Intense LAMMPS Simulations





Communications

Compute

## Conclusions

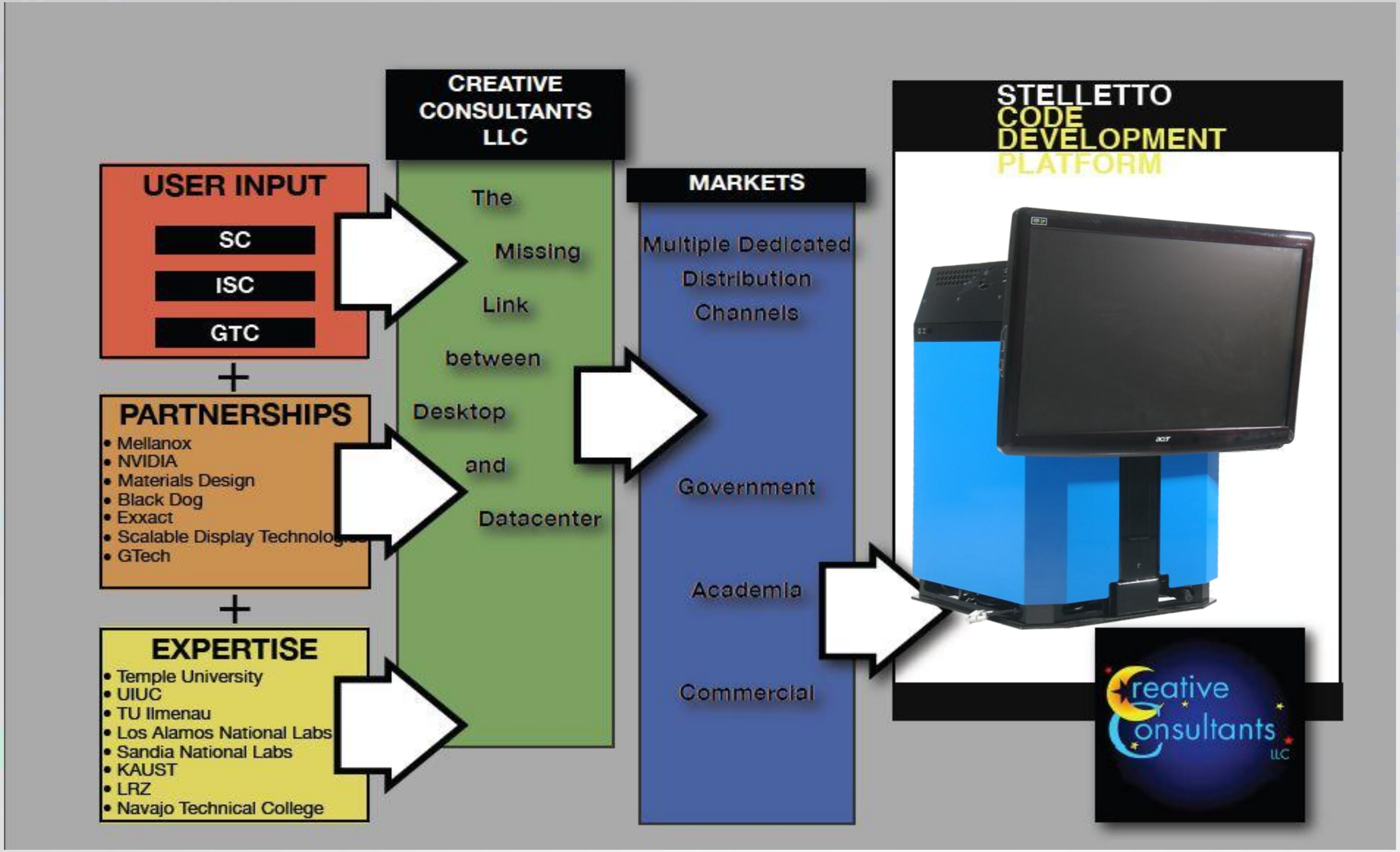
- Code is the problem,  Stelletto CDP is our solution
- Agility means Small to Large, Bottom Up Design
- Quickly Configure Rapidly Changing Architectures

## Desirable Features

- Quiet enough for office use
- Standard power using wall receptacles
- Full HPC capability in portable package

## Facilitate Science and Engineering

- **Compliment HPC Development Systems and/or Code**
- MD Script Development
- Low Node Count Production Runs


CREATIVE CONSULTANTS LLC
USER INPUT
SC
ISC
GTC
+
PARTNERSHIPS
• Mellanox
• NVIDIA
• Materials Design
• Black Dog
• Exxact
• Scalable Display Technolog
• GTech
+
EXPERTISE
• Temple University
• UIUC
• TU Ilmenau
• Los Alamos National Labs
• Sandia National Labs
• KAUST
• LRZ
• Navajo Technical College
The Missing Link between Desktop and Datacenter
MARKETS
Multiple Dedicated Distribution Channels
Government
Academia
Commercial
STELLETTO
CODE DEVELOPMENT PLATFORM
Creative Consultants LLC

Creative Consultants LLC
Efficient Performance Computing

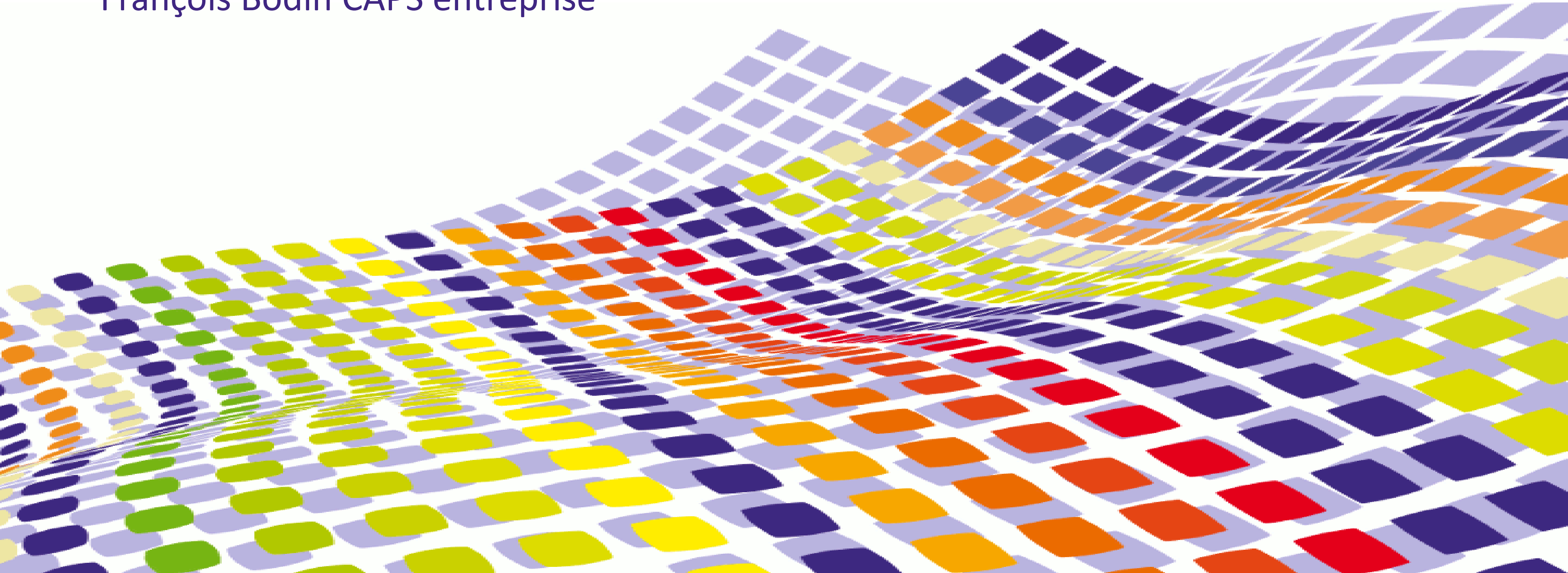
Thanks for Listening
Here’s Francois and Rob

NVIDIA

# Programming Heterogeneous Many-cores Using Directives

HMPP - OpenAcc

François Bodin CAPS entreprise



## Introduction

- Programming many-core systems faces the following dilemma
  - The constraint of keeping a unique version of codes, preferably mono-language
    - Reduces maintenance cost
    - Preserves code assets
    - Less sensitive to fast moving hardware targets
    - Codes last several generations of hardware architecture
  - Achieve "portable" performance
    - Multiple forms of parallelism cohabiting
      - Multiple devices (e.g. GPUs) with their own address space
      - Multiple threads inside a device
      - Vector/SIMD parallelism inside a thread
    - Massive parallelism
      - Tens of thousands of threads needed
- For legacy codes, directive-based approach may be an alternative

## Directives-based Approaches

- Supplement an existing serial language with directives to express parallelism and data management
  - Preserves code basis (e.g. C, Fortran) and serial semantic
  - Competitive with code written in the device dialect (e.g. CUDA)
  - Incremental approach to many-core programming
  - Mainly targets legacy codes
- Many variants
  - OpenHMPP
  - PGI Accelerator
  - OpenACC
  - OpenMP Accelerator extension
  - …
- OpenACC is a new initiative by CAPS, CRAY, PGI and NVidia
  - A first common subset presented at SC11

## HydroC Code

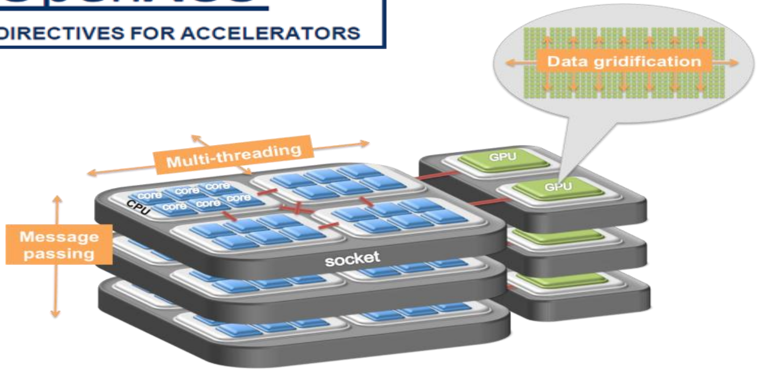
- HydroC* is a *summary* from RAMSES
  - Used to study large scale structure and galaxy formation.
  - Includes classical algorithms we can find in many applications codes for Tier-0 systems
  - Solves compressible Euler equations of hydrodynamics, based on finite volume numerical method using a second order Godunov scheme for Euler equations
  - **The algorithms have not been modified**
  - ~1500 LoC, two versions, Fortran and C, MPI
- GNU Compiler 4.4.5, MPICH, NV SDK 5.1, CAPS OpenACC, compiler flag -O3
- More at
  - http://irfu.cea.fr/Phocea/Vie_des_labos/Ast/ast_sstechnique.php?id_ast=904
  - http://hipacc.ucsc.edu/html/HIPACCLectures/lecture_hydro.pdf
  - http://calcul.math.cnrs.fr/Documents/Manifestations/CIRA2011/IDRIS_io_lyon2011.pdf

*Pierre-François Lavallée[a], Guillaume Colin de Verdière[b], Philippe Wautelet[a], Dimitri Lecas[a], Jean-Michel Dupays[a]
*[a]IDRIS/CNRS, [b]CEA,Centre DAM*

# OpenACC Initiative

- Express data and computations to be executed on an accelerator
    - Using marked code regions

OpenACC
DIRECTIVES FOR ACCELERATORS



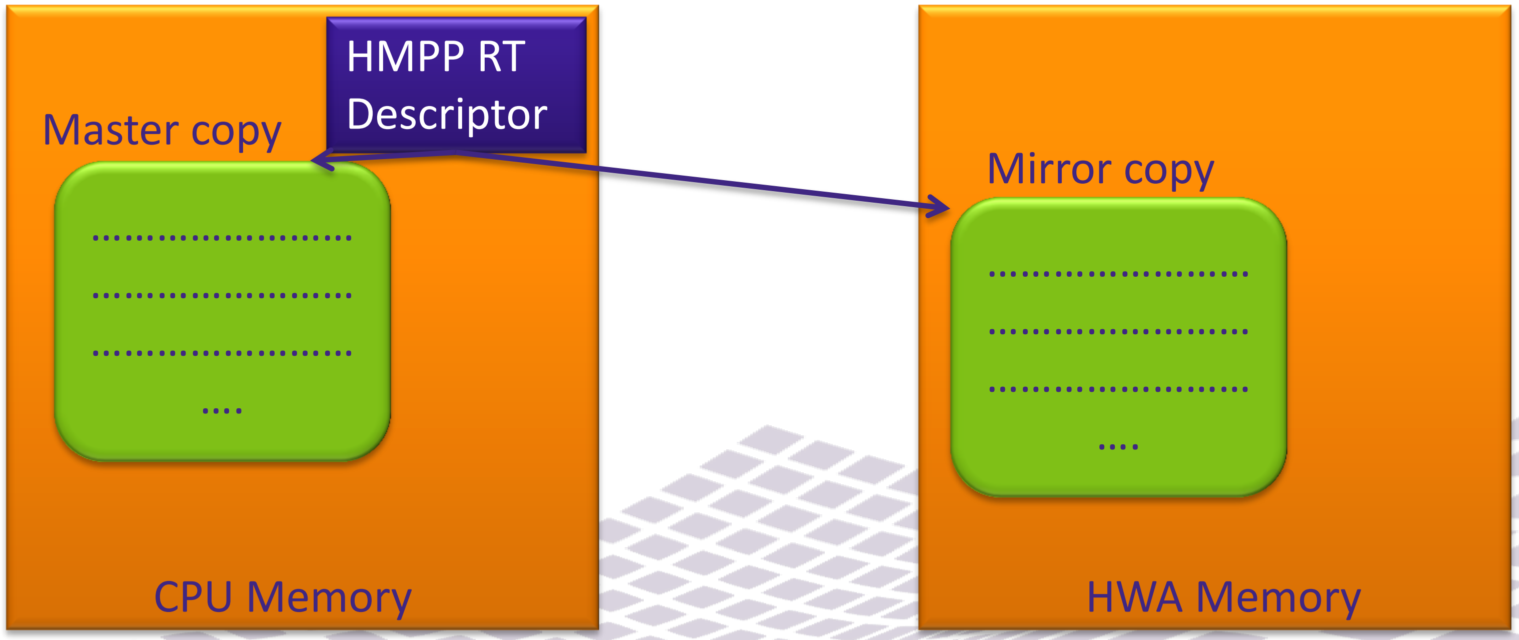
- Main OpenACC constructs
    - Parallel and kernel regions
    - Parallel loops
    - Data regions
    - Runtime API

- OpenACC support released in April 2012 (HMPP Workbench 3.1)
    - OpenACC Test Suite provided by University of Houston

- Visit http://www.openacc-standard.com for more information

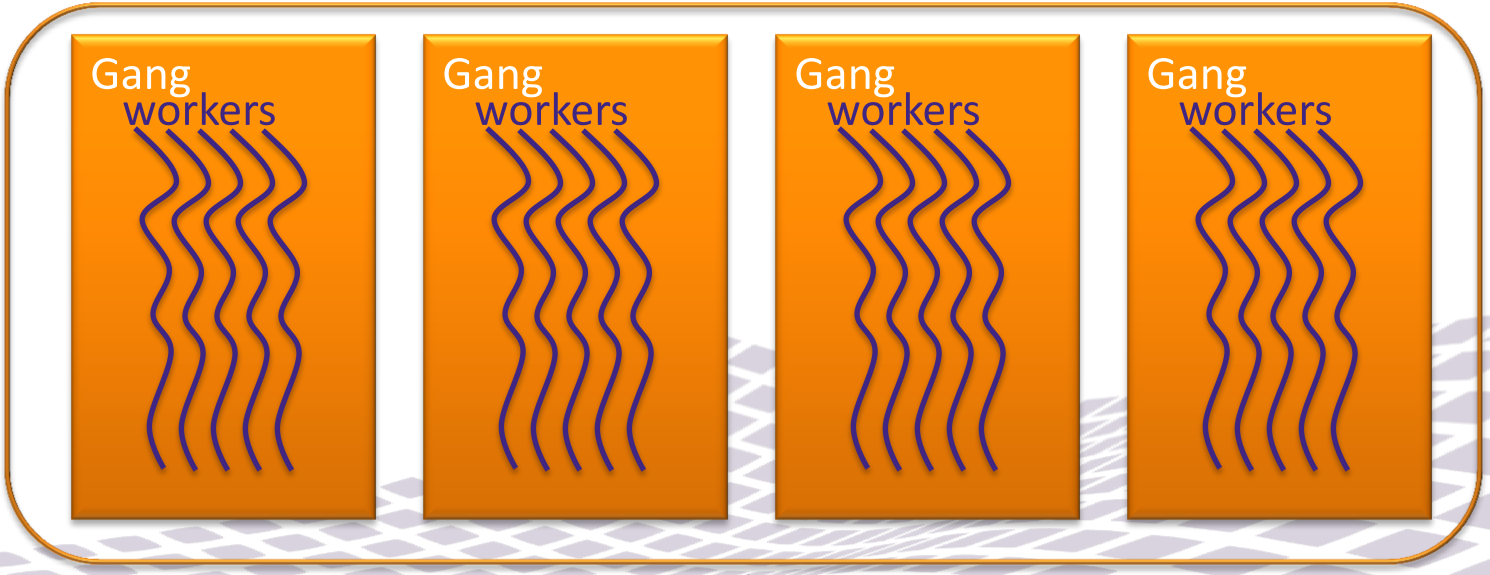
## OpenACC Data Management

- Mirroring duplicates a CPU memory block into the HWA memory
  - Mirror identifier is a CPU memory block address
  - Only one mirror per CPU block
  - Users ensure consistency of copies via directives



# OpenACC Execution Model

- Host-controlled execution
- Based on three parallelism levels
  - Gangs – coarse grain, Workers – fine grain, Vectors – finest grain



# Parallel Loops

- The loop directive describes iteration space partitioning to execute the loop; declares loop-private variables and arrays, and reduction operations
- Clauses
  - gang [(scalar-integer-expression)]
  - worker [(scalar-integer-expression)]
  - vector [(scalar-integer-expression)]
  - collapse(*n*)
  - seq
  - independent
  - private(list)
  - reduction(operator:list )

Iteration space distributed over NB gangs

```
#pragma acc loop gang(NB)
  for (int i = 0; i < n; ++i){
    #pragma acc loop worker(NT)
    for (int j = 0; j < m; ++j){
      B[i][j] = i * j * A[i][j];
    }
  }
```

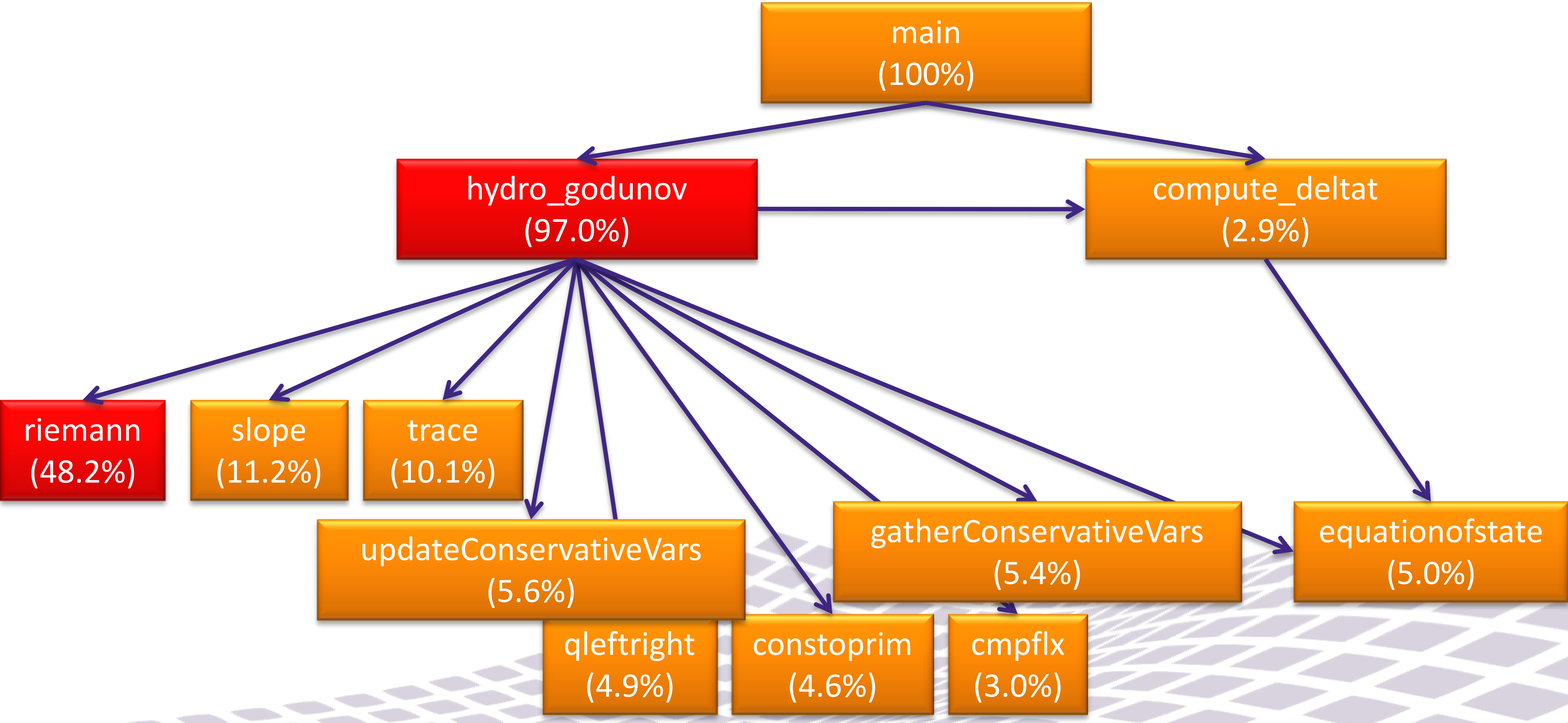
Iteration space distributed over NT workers

## Kernel Regions

- Parallel loops inside a region are transformed into accelerator kernels (e.g. CUDA kernels)
  - Each loop nest can have different values for gang and worker numbers
- Clauses
  - if(condition)
  - async[(scalar-integer-expression)]
  - copy(list)
  - copyin(list)
  - copyout(list)
  - create(list)
  - present(list)
  - present_or_copy(list)
  - present_or_copyin(list)
  - present_or_copyout(list)
  - present_or_create(list)
  - deviceptr(list)

```
#pragma acc kernels
{
#pragma acc loop independent
  for (int i = 0; i < n; ++i){
    for (int j = 0; j < n; ++j){
      for (int k = 0; k < n; ++k){
        B[i][j*k%n] = A[i][j*k%n];
      }
    }
  }
#pragma acc loop gang(NB)
  for (int i = 0; i < n; ++i){
    #pragma acc loop worker(NT)
    for (int j = 0; j < m; ++j){
      B[i][j] = i * j * A[i][j];
    }
  }
}
```

# HydroC Call-Graph



# Riemann Hotspot

```
void riemann ()
{
  #pragma acc kernels
    copy( qleft[0:Hnvar*Hstep*Hnxyt], \
          qright[0:Hnvar*Hstep*Hnxyt], \
          qgdnv[0:Hnvar*Hstep*Hnxyt], \
          sgnm[0:Hstep*Hnxyt] )
  {
    #pragma acc loop independent
    for (int s = 0; s < slices; s++){
      for (int i = 0; i < narray; i++){
        …
  }
  …
  #pragma acc kernels
    copy( qleft[0:Hnvar*Hstep*Hnxyt], \
          qright[0:Hnvar*Hstep*Hnxyt], \
          sgnm[0:Hstep*Hnxyt], \
          qgdnv[0:Hnvar*Hstep*Hnxyt] )
  {
    #pragma acc loop independent
    for (int invar = IP + 1; invar < Hnvar; invar++){
      for (int s = 0; s < slices; s++){
     ...
```

Allocate and copy data from host to device and device to host and deallocate at the end of the block.

1D *gridification*

Copy data from device to host and deallocate

Potential Speedup

$S_p = 1 / (1 – 0.4824) = 1.93$

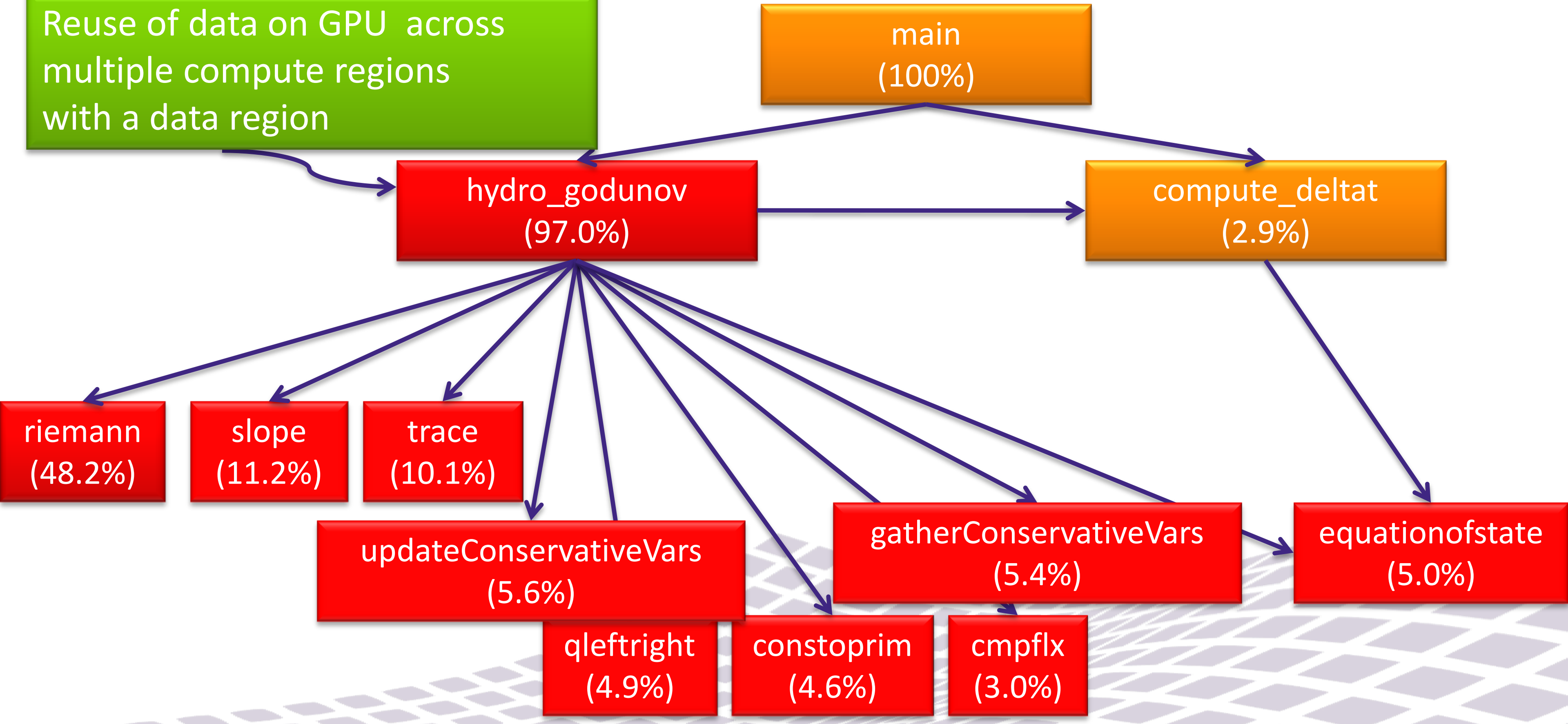
# Data Management Directives

- Data regions define scalars, arrays and sub-arrays to be allocated in the device memory for the duration of the region
    - Explicit management of data transfers using clauses or directives
- Many clauses
    - if(condition)
    - copy(list)
    - copyin(list )
    - copyout(list)
    - create(list)
    - present(list)
    - present_or_copy(list)
    - present_or_copyin(list)
    - present_or_copyout(list)
    - present_or_create(list)
    - deviceptr(list)

```
#pragma acc data copyin(A[1:N-2]),
  copyout(B[N])
{
  #pragma acc kernels
  {
    #pragma acc loop independant
    for (int i = 0; i < N; ++i){
     ...
    }}
  #pragma acc update host(A)
    ...
  #pragma acc kernels
  for (int i = 0; i < n; ++i){
      B[i] = ...;
  }}
```

## Further Optimizations

Reuse of data on GPU across multiple compute regions with a data region



# Adding Data Region

```
void hydro_godunov (…)
{
#pragma acc data \
  create(qleft[0:H.nvar], qright[0:H.nvar], \
         q[0:H.nvar], qgdnv[0:H.nvar], \
         flux[0:H.nvar], u[0:H.nvar], \
         dq[0:H.nvar], e[0:Hstep], c[0:Hstep], \
         sgnm[0:Hstep], qxm[0:H.nvar], qxp[0:H.nvar]) \
  copy(uold[0:H.nvar*H.nxt*H.nyt]) \
  copyin(Hstep)
 {
  for (j = Hmin; j < Hmax; j += Hstep){
                // compute many slices each pass
                 int jend = j + Hstep;
                 if (jend >= Hmax)
                   jend = Hmax;
          . . .// the work here
   } // for j
  }//end of data region
...
```

Data are left on the GPU during the step loop. pcopy clauses are used into called routines

## Full Application

- With the same strategy, the full application have been ported with OpenACC

- The following hotspots have been accelerated
  - cmplx
  - updateConservativeVar
  - gatherConservativeVar
  - constoprim
  - equationofstate
  - qleftright
  - riemann
  - slope
  - trace

1 week of development

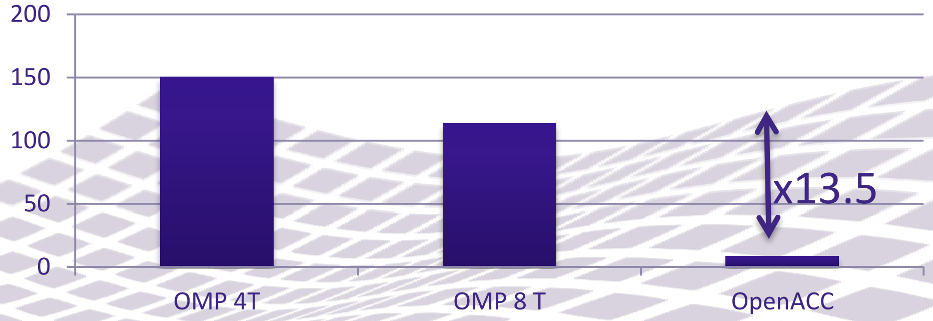
60 directives, 4% of the LoC

Achieved speedup = 3x
and still room for improvement

# DNA Distance Application with OpenACC

- Biomedical application part of Phylip package,
  - Main computation kernel takes as input a list of DNA sequences for each species
    - Code is based on an approximation using Newton-Raphson method (SP)
    - Produces a 2-dimension matrix of distances
  - Experiments performed in the context of the HMPP APAC CoC*
- Performance
  - OpenMP version, 4 & 8 threads, Intel(R) i7 CPU 920 @ 2.67GHz
  - 1 GPU Tesla C2070

**Execution time in seconds**

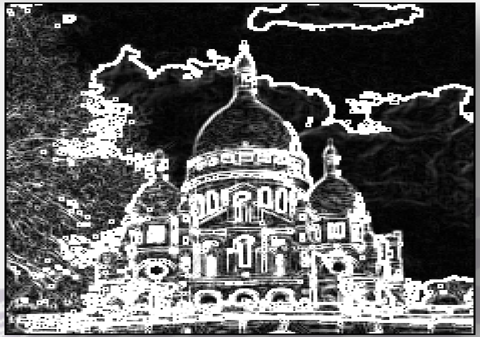


*http://competencecenter.hmpp.org/category/hmpp-coc-asia/

# Sobel Filter Performance Example

## Edge detection algorithm

- Sobel Filter benchmark
- Size
  - ~ 200 lines of C code
- GPU C2070 improvement
  - x 24 over serial code on Intel i7 CPU 920 @ 2.67GHz
- Main porting operation
  - Inserting 6 OpenACC directives



## Conclusion

- Directive-based approaches are currently one of the most promising track for heterogeneous many-cores
  - Preserve code assets
  - At node level help separating parallelism description from implementation
- Need to integrate libraries and user codes
  - Requires interoperability between runtimes
- Auto-tuning is one of the future keys to efficient portability
  - Has to be part of the programming API

## GTC Special Offering

- CAPS OpenACC Compiler for $199
  - OpenACC 1.0 directives
  - C and Fortran to CUDA - for Linux
  - One-year maintenance and support
  - 1 perpetual license
  - Ends June 15th

- See us on Booth #28 or on Exxact booth #36

Accelerator Programming Model Parallelization

Directive-based programming GPGPU Manycore programming

Hybrid Manycore Programming HPC community OpenACC

Petaflops Parallel computing HPC open standard

Multicore programming Exaflops NVIDIA Cuda

Code speedup Hardware accelerators programming

High Performance Computing OpenHMPP

Parallel programming interface

Massively parallel

Open CL

OpenACC DIRECTIVES FOR ACCELERATORS

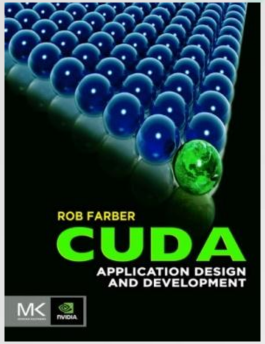
http://www.caps-entreprise.com
http://twitter.com/CAPSentreprise
http://www.openacc-standard.org/
http://www.openhmpp.org

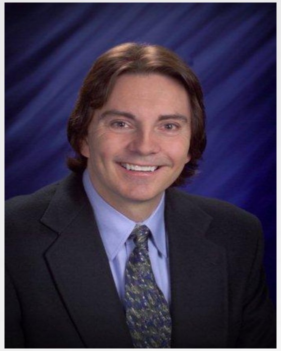
# The potential: GPU, multiGPU, and CPU+GPU

Rob Farber

Chief Scientist, BlackDog Endeavors, LLC

Author, “CUDA Application Design and Development”





Doctor Dobb’s Journal CUDA tutorials

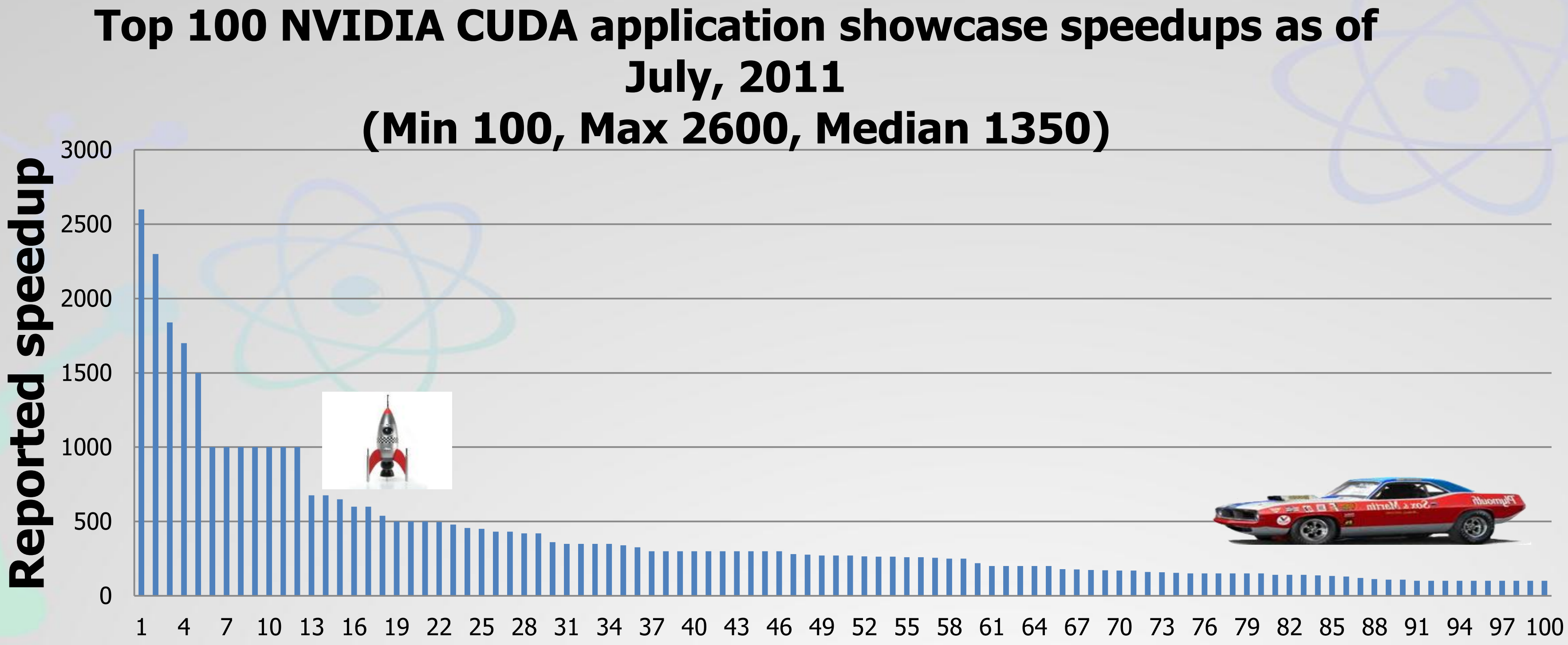
Dr.Dobb's THE WORLD OF SOFTWARE DEVELOPMENT

OpenCL “The Code Project” tutorials

THE CODE PROJECT Your Development Resource

Columnist Scientific Computing

# Performance is the reason for GPUs



**Ranked from highest to lowest speedup**

http://developer.nvidia.com/cuda-action-research-apps

## Supercomputing for the masses!

- Market forces evolved GPUs into massively parallel GPGPUs (General Purpose GPUs).
- **300+ million CUDA-enabled GPUs says it all!**
- CUDA: put supercomputing in the hands of the masses
  - December 1996, ASCI Red the first teraflop supercomputer
  - Today: kids buy GPUs with flop rates comparable to systems available to scientists with supercomputer access in the mid to late 1990s
    - GTX 560 $169 on newegg.com

Remember that Finnish kid who wrote some software to understand operating systems? Inexpensive commodity hardware enables:

- New thinking
- A large educated base of developers

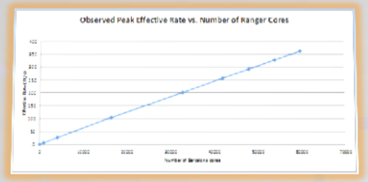
You can change the world!

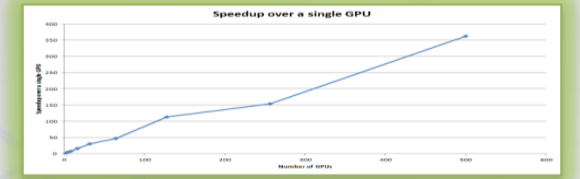
# Make your life easy!

- Four basic types of programming models:
  - Directive-based programming like OpenMP and OpenACC
    - Just heard about OpenACC and HMPP
  - Language platforms based on a strong-scaling execution model **(CUDA and OpenCL™)**
  - Common libraries providing FFT and BLAS functionality
  - MPI (Message Passing Interface)

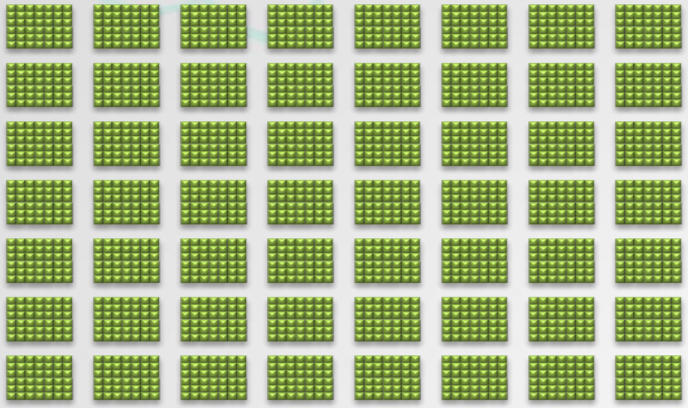
# Big idea: A strong scaling execution model!

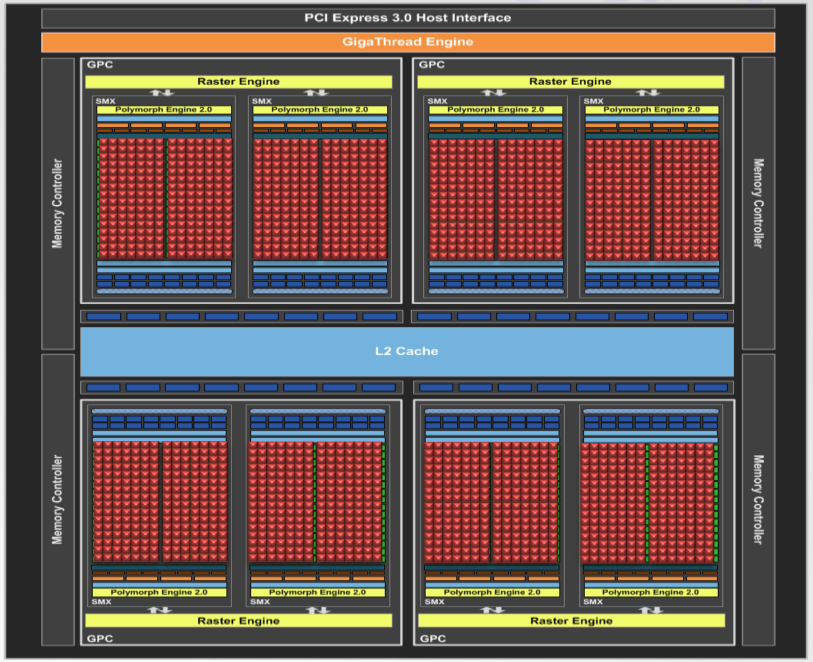
- Threads can only communicate within a thread block





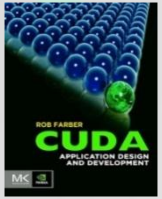
- Fast hardware scheduling
  - Both Grid and on SM/SMX





# If you know C++, you are already programming GPUs!

First two examples in



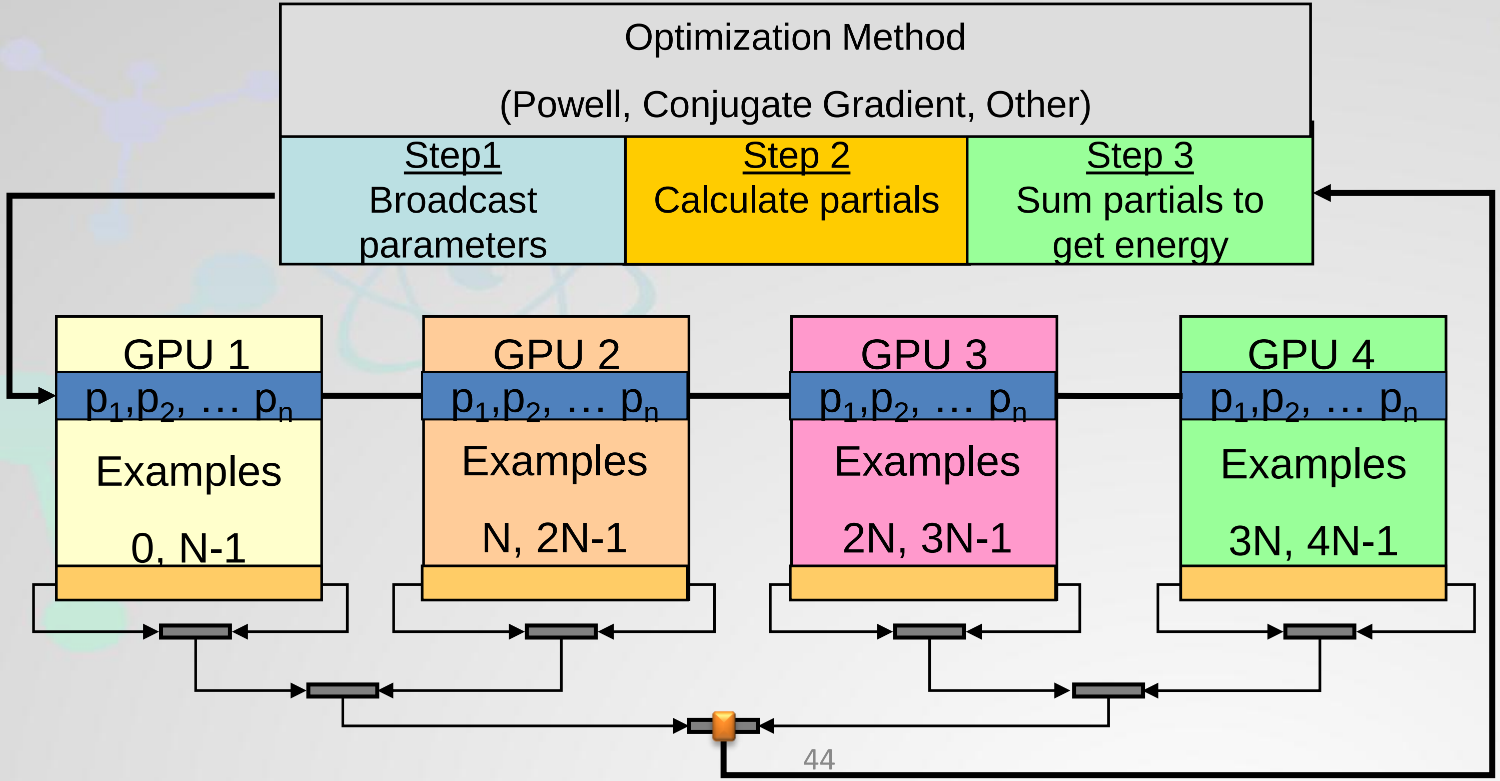
```
//seqSerial.cpp
#include <iostream>
#include <vector>
using namespace std;

int main()
{
  const int N=50000;

  // task 1: create the array
  vector<int> a(N);

  // task 2: fill the array
  for(int i=0; i < N; i++) a[i]=i;

  // task 3: calculate the sum of the array
  int sumA=0;
  for(int i=0; i < N; i++) sumA += a[i];

  // task 4: calculate the sum of 0 .. N-1
  int sumCheck=0;
  for(int i=0; i < N; i++) sumCheck += i;

  // task 5: check the results agree
  if(sumA == sumCheck) cout << "Test Succeeded!" << endl;
  else {cerr << "Test FAILED!" << endl; return(1);}

  return(0);
}
```

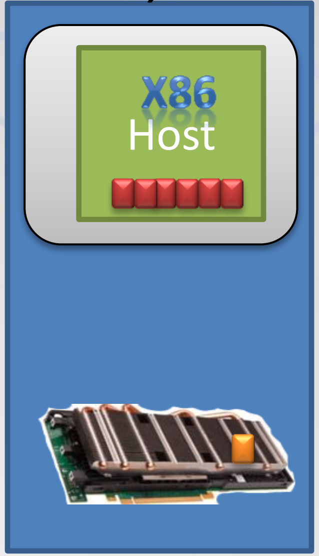
```
//seqCuda.cu
#include <iostream>
using namespace std;

#include <thrust/reduce.h>
#include <thrust/sequence.h>
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>

int main()
{
  const int N=50000;

  // task 1: create the array
  thrust::device_vector<int> a(N);

  // task 2: fill the array
  thrust::sequence(a.begin(), a.end(), 0);

  // task 3: calculate the sum of the array
  int sumA= thrust::reduce(a.begin(),a.end(), 0);

  // task 4: calculate the sum of 0 .. N-1
  int sumCheck=0;
  for(int i=0; i < N; i++) sumCheck += i;

  // task 5: check the results agree
  if(sumA == sumCheck) cout << "Test Succeeded!" << endl;
  else { cerr << "Test FAILED!" << endl; return(1);}

  return(0);
}
```

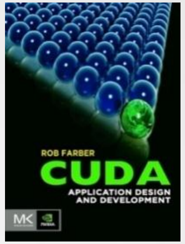
# A general mapping: use thrust::transform_reduce()

$$energy = objFunc(p_1, p_2, \ldots p_n)$$

*(efficient on SIMD, SIMT, MIMD, vector, vector parallel, cluster, cloud)*





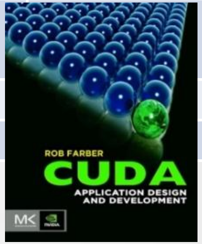


# Speedup over a quad core when learning XOR

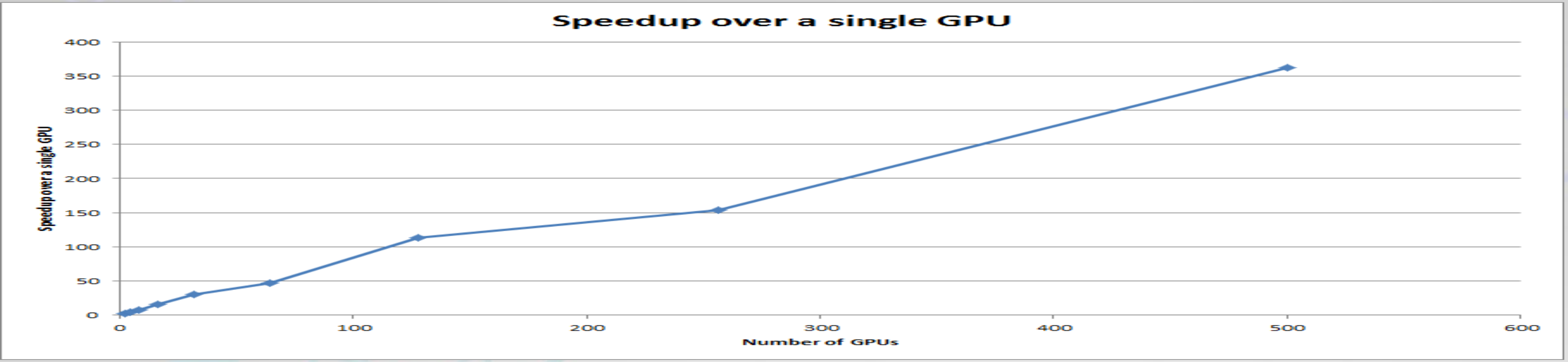
| OS | Machine | Opt method | Precision | Ave obj func time | % func time | Speedup over quad-core | Speedup over single-core |
|---|---|---|---|---|---|---|---|
| Linux | NVIDIA C2070 | Nelder-Mead | 32 | 0.00532 | 100.0 | 85 | 341 |
| Win7 | NVIDIA C2070 | Nelder-Mead | 32 | 0.00566 | 100.0 | 81 | 323 |
| Linux | NVIDIA GTX280 | Nelder-Mead | 32 | 0.01109 | 99.2 | 41 | 163 |
| Linux | NVIDIA C2070 | Nelder-Mead | 64 | 0.01364 | 100.0 | 40 | 158 |
| Win7 | NVIDIA C2070 | Nelder-Mead | 64 | 0.01612 | 100.0 | 22 | 87 |
| Linux | NVIDIA C2070 | Levenberg-Marquardt | 32 | 0.04313 | 2.7 | 10 | 38 |
| Linux | NVIDIA C2070 | Levenberg-Marquardt | 64 | 0.08480 | 4.4 | 6 | 23 |
| Linux | Intel e5630 | Levenberg-M | | | | | |
| Linux | Intel e5630 | Levenberg-M | | | | | |
| Linux | Intel e5630 | Nelder-M | | | | | |
| [illegible] | [illegible] | Nelder-M | | | | | |

```
#pragma omp parallel for reduction(+ : sum)
  for(int i=0; i < nExamples; ++i)
  {
   Real d = getError(i);
   sum += d;
  }
```

Code for CPU generated by thrust

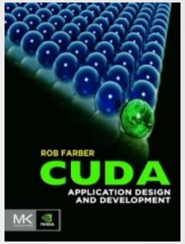


## So simple it’s the MPI example in Chapter 10



- Dominant runtime of code that scales to 500 GPUs

```
FcnOfInterest objFcn(input);

energy = thrust::transform_reduce(
            thrust::counting_iterator<int>(0),
             thrust::counting_iterator<int>(nExamples),
                                          objFcn, 0.0f, thrust::plus<Real>());
```



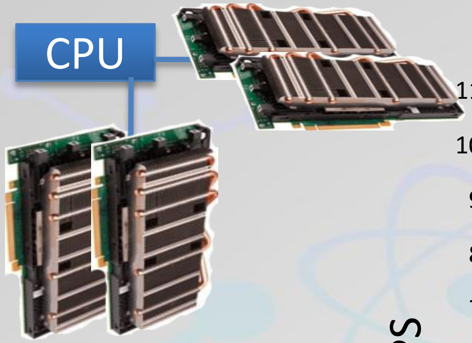
ICHEC
Irish Centre for High-End Computing

# Hybrid DGEMM
(matrix multiply)

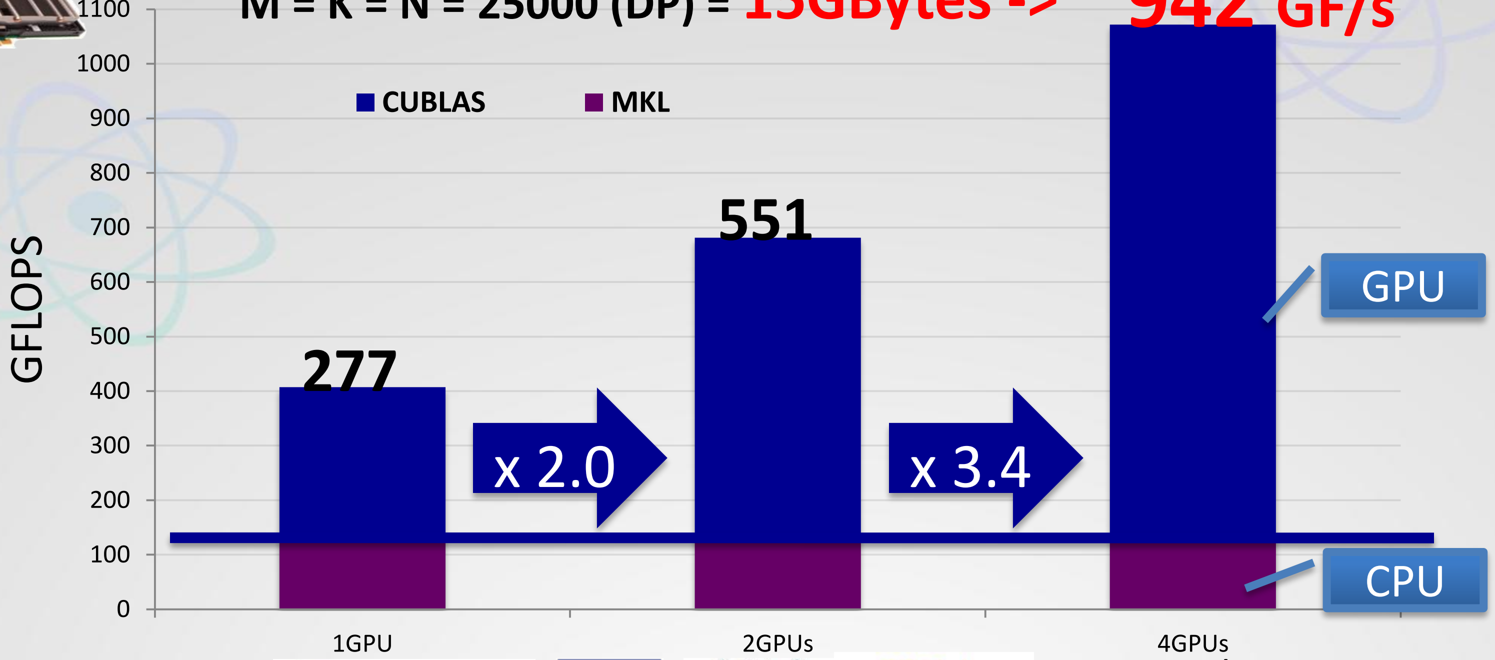
http://qe-forge.org/projects/phigemm/
(Ivan Girotto and Filippo Spiga)

**M = K = N = 25000 (DP) = 15GBytes -> 942 GF/s**

(2 GPUs per PCI Bus!!)



2 x Intel Xeon X5670 2.93GHz + 4 NVIDIA Tesla C2050



System provided by NVIDIA

Notice the size >> a single GPU

# 8-GPU Stelletto Configurations?

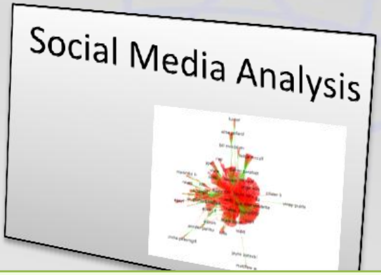
- 2x AMD Quad Socket Interlagos
- 4x PCIe 2.0 GPUs
- QDR InfiniBand

- 2x Intel Dual Socket Sandy Bridge
- 6x PCIe 3.0 GPUs
- FDR InfiniBand

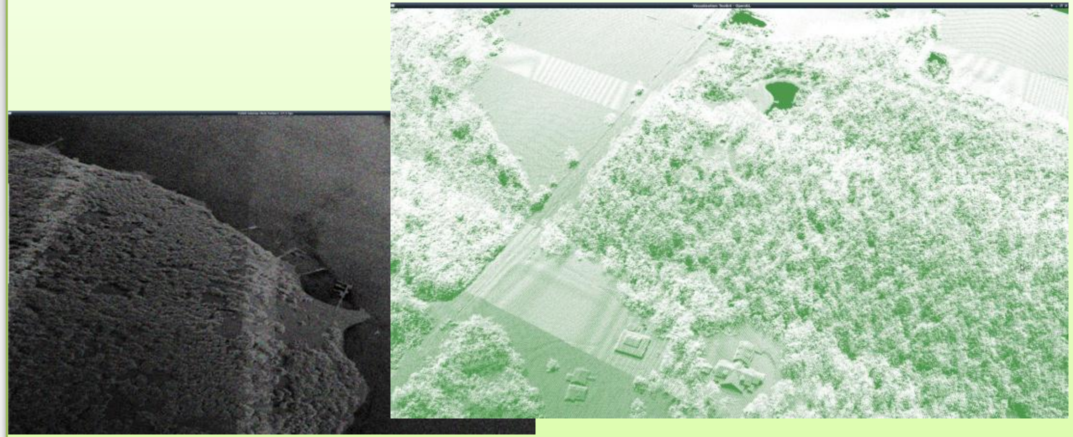
# CUDA + Primitive Restart (a potent combination!)

**Primitive restart:** ***Looking forward to Kepler!***

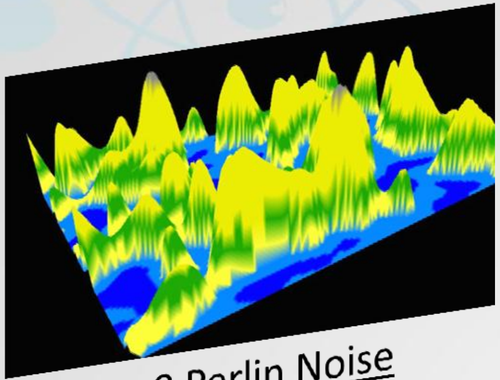
- A feature of OpenGL 3.1
- Roughly 60x faster than optimized OpenGL
- Avoids the PCIe bottleneck
- Variable length data works great!



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

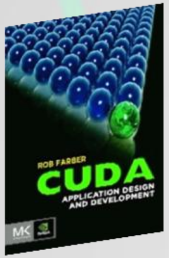


In collaboration with Global Navigation Sciences (http://http://globalnavigationsciences.com/



Chapter 9 Perlin Noise

Fly around in a 3D virtual world



## Note the speed difference between GPU and CPU

See on YouTube
http://www.youtube.com/watch?v=SzTQUCPtk80
http://www.youtube.com/watch?v=5zhL7JATkSI

Each test ran separately

rmfarber@bda: ~/AMD/AMD006/example
File Edit View Terminal Help

```
rmfarber@bda:~/AMD/AMD006/example$ ./gltest
rmfarber@bda:~/AMD/AMD006/example$
```

GPU

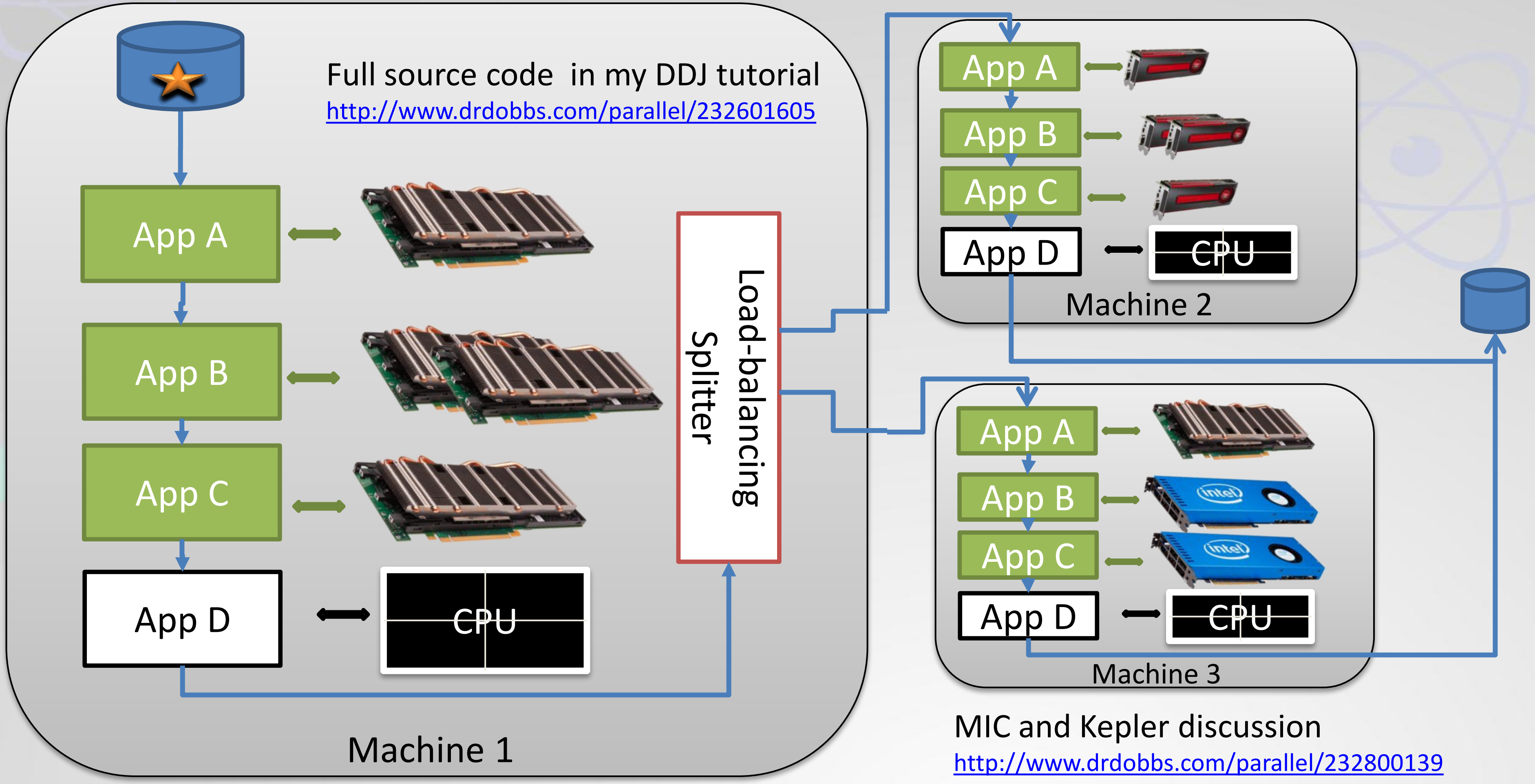
rmfarber@bda: ~/AMD/AMD006/example
File Edit View Terminal Help

```
rmfarber@bda:~/AMD/AMD006/example$ ./gltest CPU
```

CPU

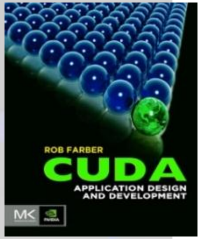
# Fast and scalable heterogeneous workflows

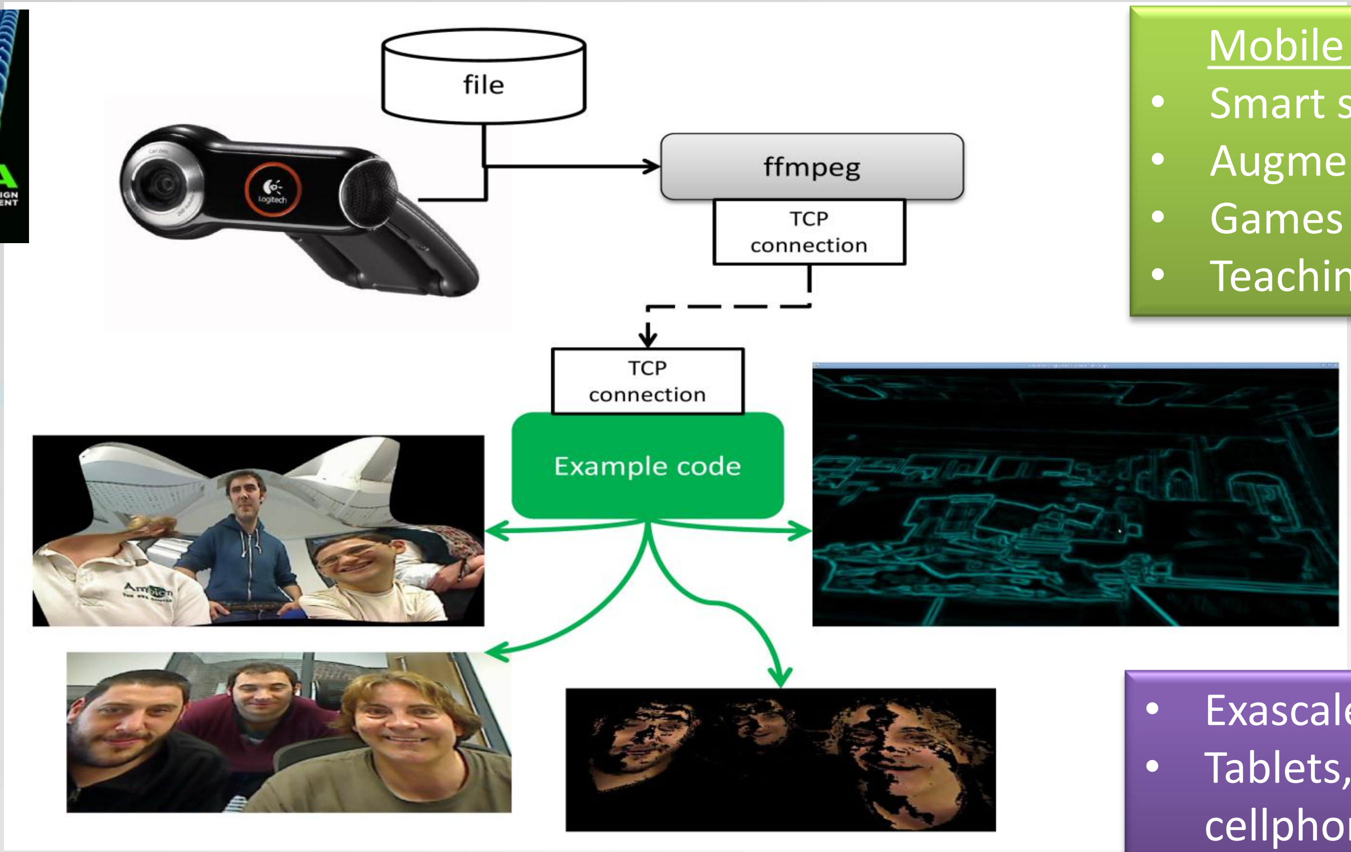
Full source code in my DDJ tutorial
http://www.drdobbs.com/parallel/232601605



MIC and Kepler discussion
http://www.drdobbs.com/parallel/232800139

# A cool real-time video workflow





**Mobile or desktop**

- Smart sensors
- Augmented Reality
- Games
- Teaching

- Exascale video analysis
- Tablets, notebooks … cellphones?

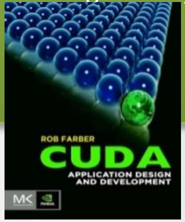
# For the demo, think Kinect and 3D morphing for augmented reality

(identify flesh colored blobs for hands)

Artifacts caused by picking a colorspace rectangle rather than an ellipse

## 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)
{
  un                       blockIdx.x*blockDim.x + threadIdx.x;
                           blockIdx.y*blockDim.y + threadIdx.y;
                     s[y*width+x].x;
                    Pos[y*width+x].y;
                  lorPos[y*width+x].z;
        reR = 255*( ((float)r)/(r+g+b));
       pureG = 255*( ((float)g)/(r+g+b));
  i  !( (p                 ) && (pureG < highPureG)
                                  PureR) && (pureR < highPureR)

                           _uchar4(0,0,0,0);
}
```



## Full source code provided in *"CUDA Application Design and Development"* in print and on Kindle.

Available from many booksellers.

- Kindle version (color) is also available) [http://www.amazon.com/CUDA-Application-Design-Development-Farber/dp/0123884268](http://www.amazon.com/CUDA-Application-Design-Development-Farber/dp/0123884268)

The Chinese edition is coming! (interest in other translations?)

Teaching aids (PowerPoint slides, code) available on [http://GPUcomputing.net/RobFarber](http://GPUcomputing.net/RobFarber)

20% off at GTC: I'll sign books after this talk