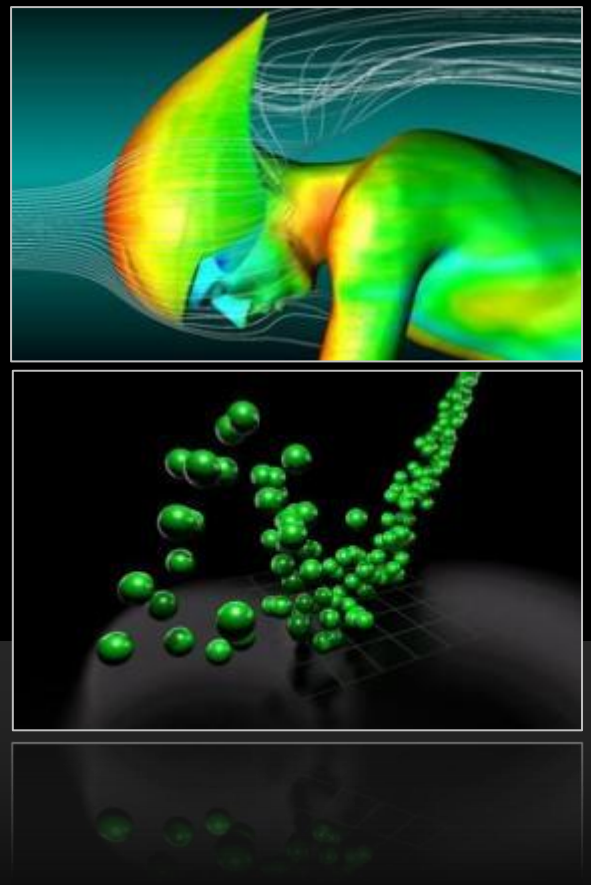
Gear Up for Speedup

Delivering 4x Speedups with ArrayFire



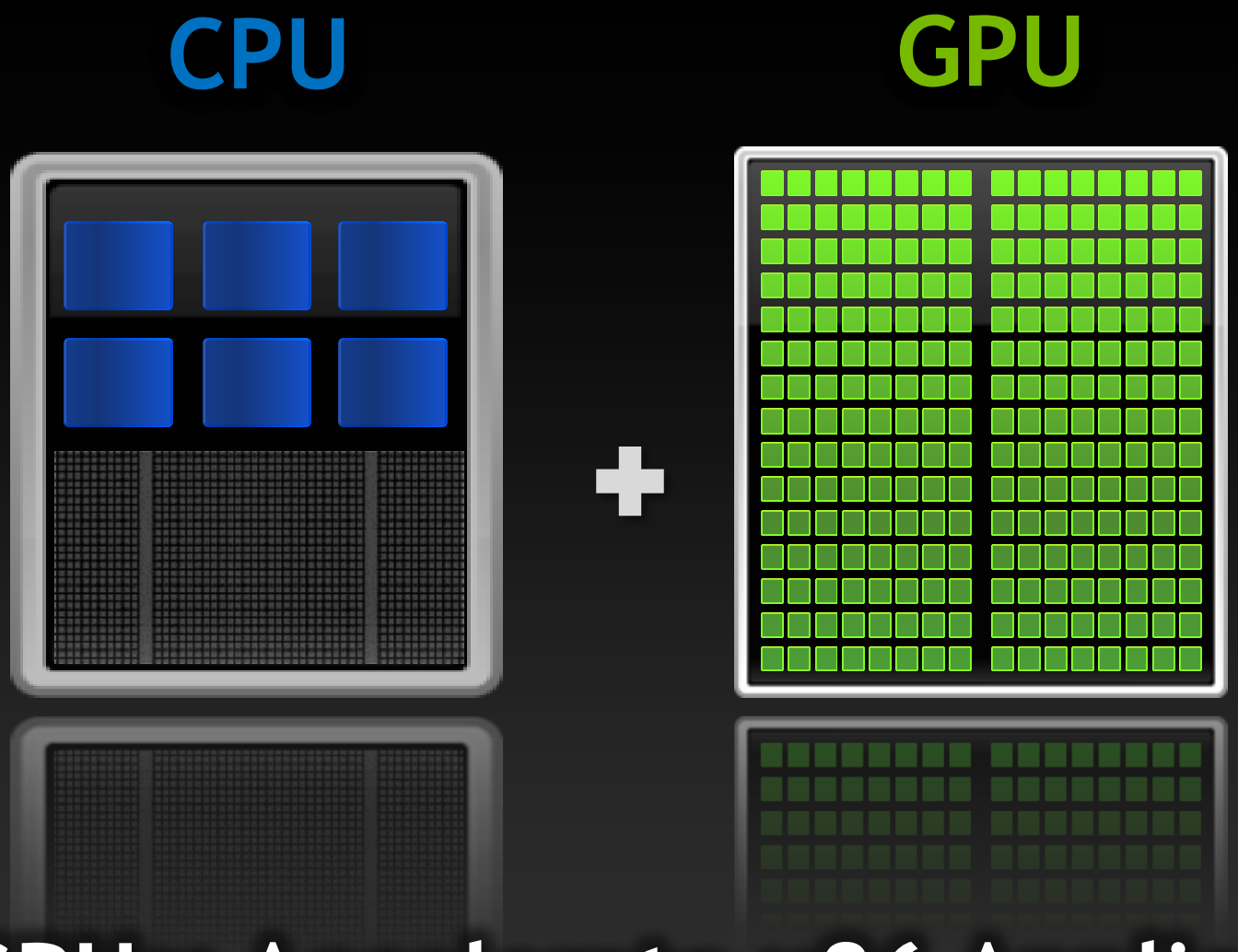
Julia Levites – NVIDIA
John Melonakos – AccelerEyes

Introduction

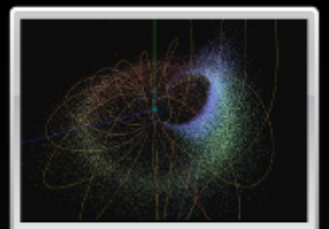
- What is GPU computing?
- GPU computing with ArrayFire

Gear Up for Speedup!

Experience 4X speed up with ArrayFire and GPUs



Add GPUs: Accelerate x86 Applications



100X

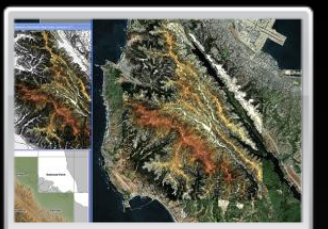
Astrophysics
RIKEN

130X

Quantum Chemistry
U of Illinois, Urbana

25X

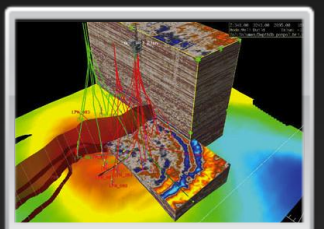
Fast Flow Simulation
FluiDyna



60X

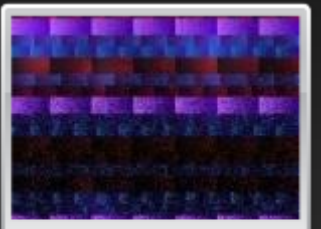
Geospatial Imagery
PCI Geomatics

GPUs Accelerate HPC



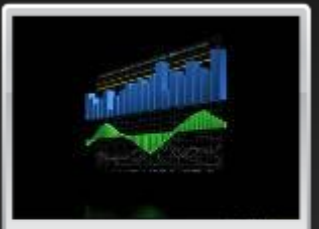
12X

Kirchoff Time Migration
Acceleware



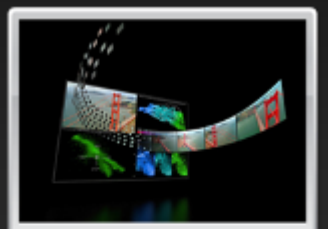
30X

Gene Sequencing
U of Maryland



149X

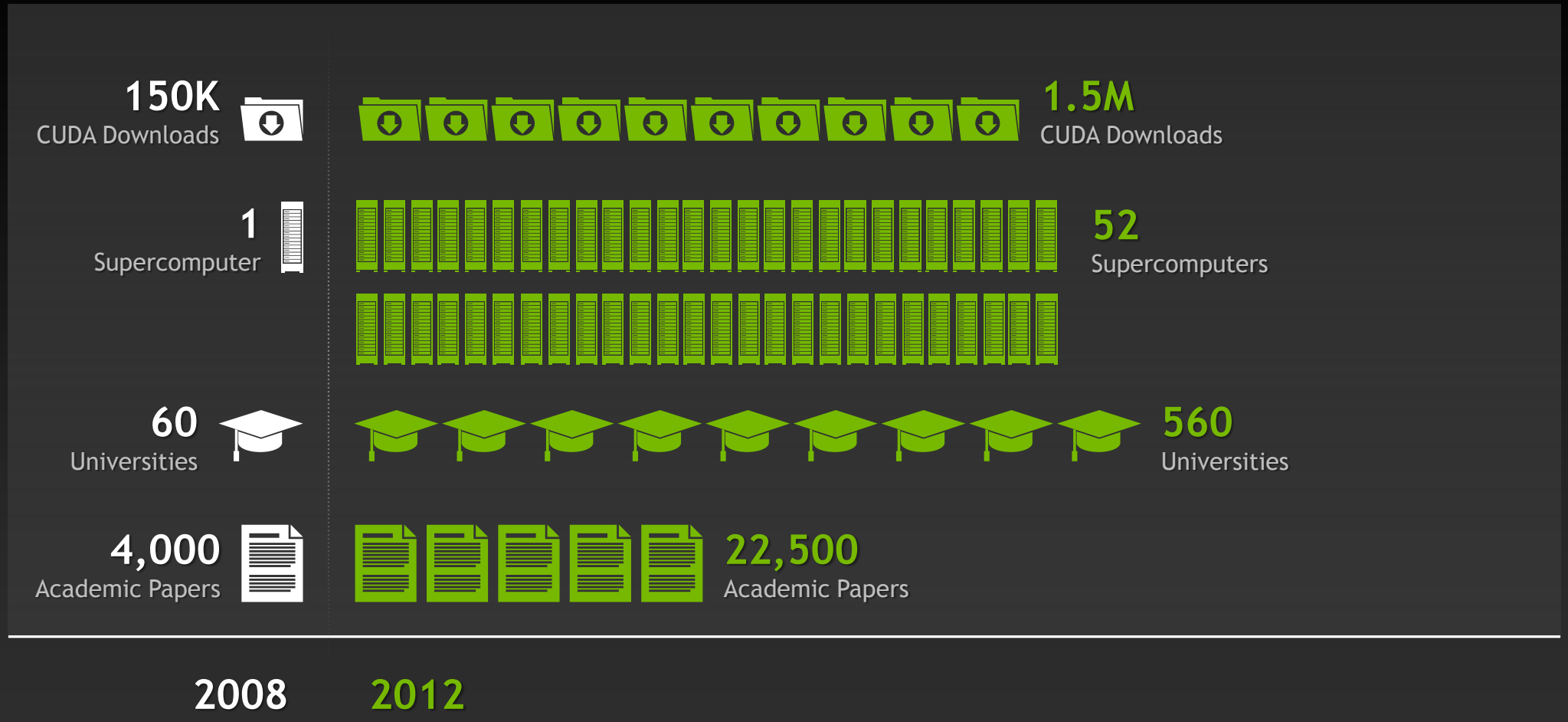
Financial Simulation
Oxford



18X

Video Transcoding
Elemental Tech

Explosive Growth of GPU Computing

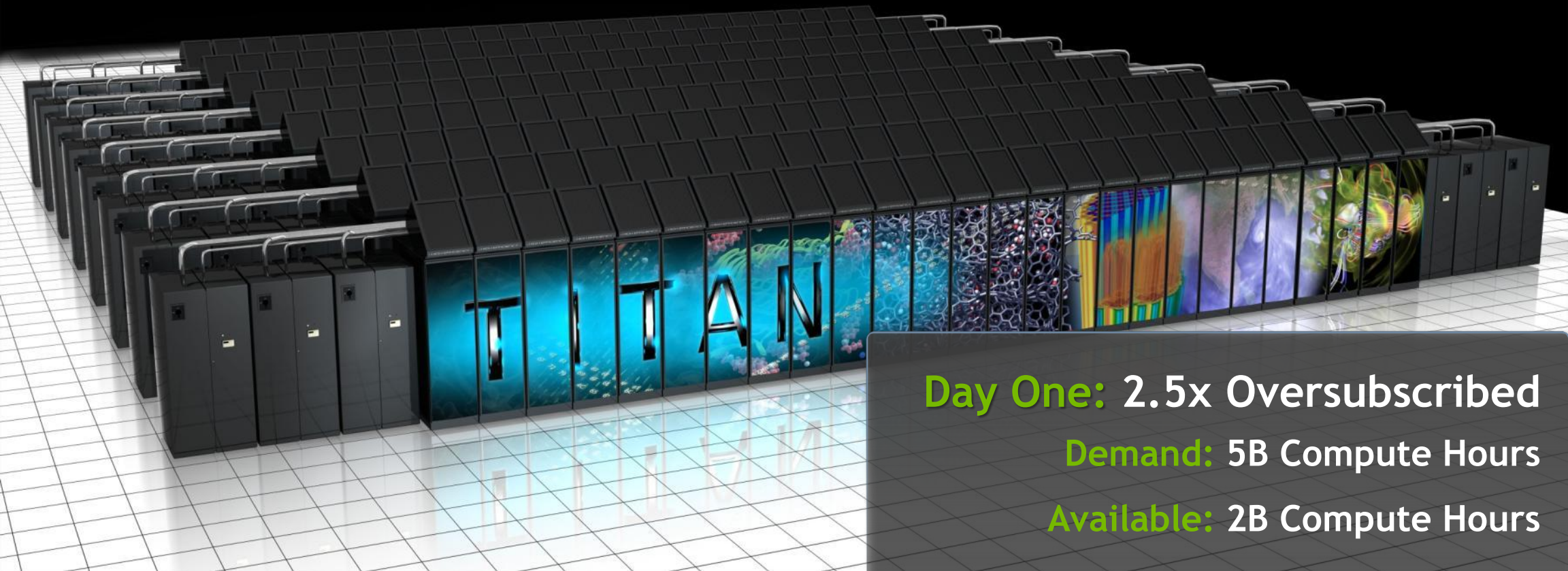


Titan: World's #1 Open Science Supercomputer

18,688 Tesla Kepler GPUs

Fully GPU Accelerated Computing System

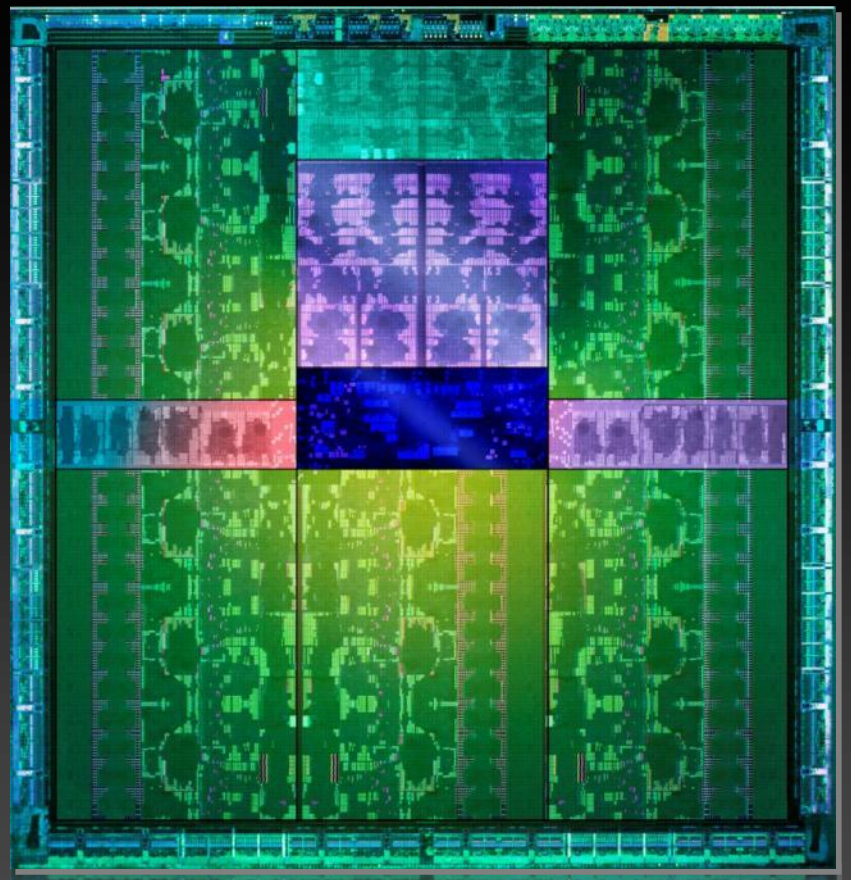
20+ Petaflops: 90% of Performance from GPUs



Day One: 2.5x Oversubscribed

Demand: 5B Compute Hours

Available: 2B Compute Hours



KEPLER

THE WORLD'S FASTEST, MOST EFFICIENT HPC ACCELERATOR

3x Perf/Watt

Max Utilization

Easier to Program

Test Drive Kepler with ArrayFire

How can Gear Up help me?

Tonči Jukić
@toncijukic

Follow

Today we have completed #ArrayFire implementation for our GPGPU farm and ported over 30% of code within only 3 days.
accelereyes.com/products/array...

Reply Retweeted Favorite

Fast, simple & comprehensive GPU library

- Hand-optimized GPU functions – Top speeds
- C, C++, Fortran
- CUDA 5.0 and Kepler K20 support
- v2.0 (coming this month)

accelereyes.com/arrayfire

estimate π

estimate π

```
#include <stdio.h>
#include <arrayfire.h>
using namespace af;
int main() {
    // 20 million random samples
    int n = 20e6;
    array x = randu(n,1), y = randu(n,1);
    // how many fell inside unit circle?
    float pi = 4 * sum<float>(sqrt(x*x+y*y)<1) / n;
    printf("pi = %g\n", pi);
    return 0;
}
```

estimate π

```
#include <stdio.h>
#include <arrayfire.h>
using namespace af;
int main() {
    // 20 million random samples
    int n = 20e6;
    array x = randu(n,1), y = randu(n,1);
    // how many fell inside unit circle?
    float pi = 4 * sum<float>(sqrt(x*x+y*y)<1) / n;
    printf("pi = %g\n", pi);
    return 0;
}
```

estimate π

```
#include <stdio.h>
#include <arrayfire.h>
using namespace af;
int main() {
    // 20 million random samples
    int n = 20e6;
    array x = randu(n,1), y = randu(n,1);
    // how many fell inside unit circle?
    float pi = 4 * sum<float>(sqrt(x*x+y*y)<1) / n;
    printf("pi = %g\n", pi);
    return 0;
}
```

estimate π

```
#include <stdio.h>
#include <arrayfire.h>
using namespace af;
int main() {
    // 20 million random samples
    int n = 20e6;
    array x = randu(n,1), y = randu(n,1);
    // how many fell inside unit circle?
    float pi = 4 * sum<float>(sqrt(x*x+y*y)<1) / n;
    printf("pi = %g\n", pi);
    return 0;
}
```

Data types

```
array x = randu(n,f32);
array y = randu(n, f64);
array z = rand(n, u32);
```

f32
real single precision

f64
real double precision

c32
complex single precision

array
container object

u32
unsigned integer

c64
complex double precision

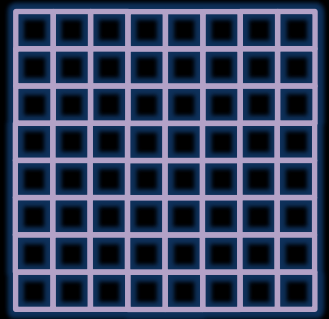
b8
boolean byte

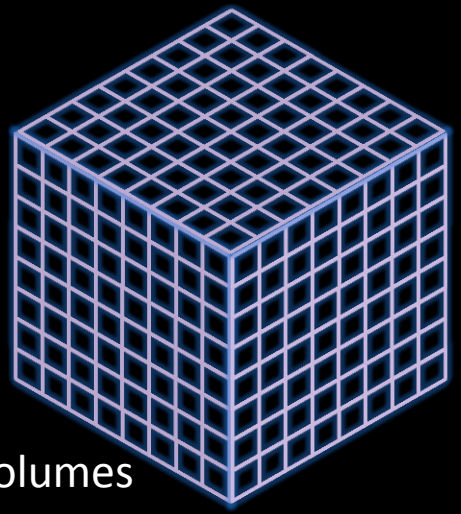
s32
signed integer

Up to 4D Support

vectors

matrices





volumes

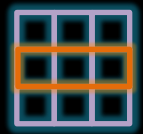
... 4D

Subscripting

ArrayFire Keywords: **end, span**

A(1,1)

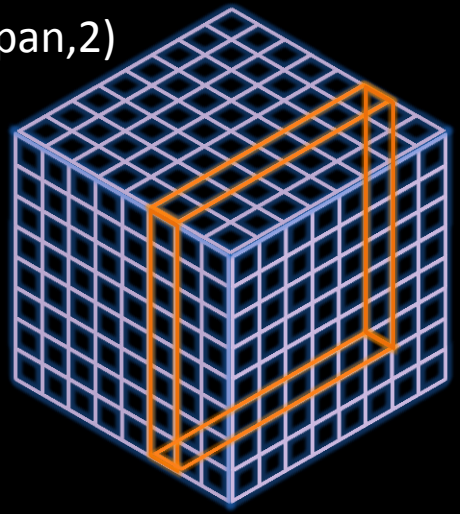
A(1,span)

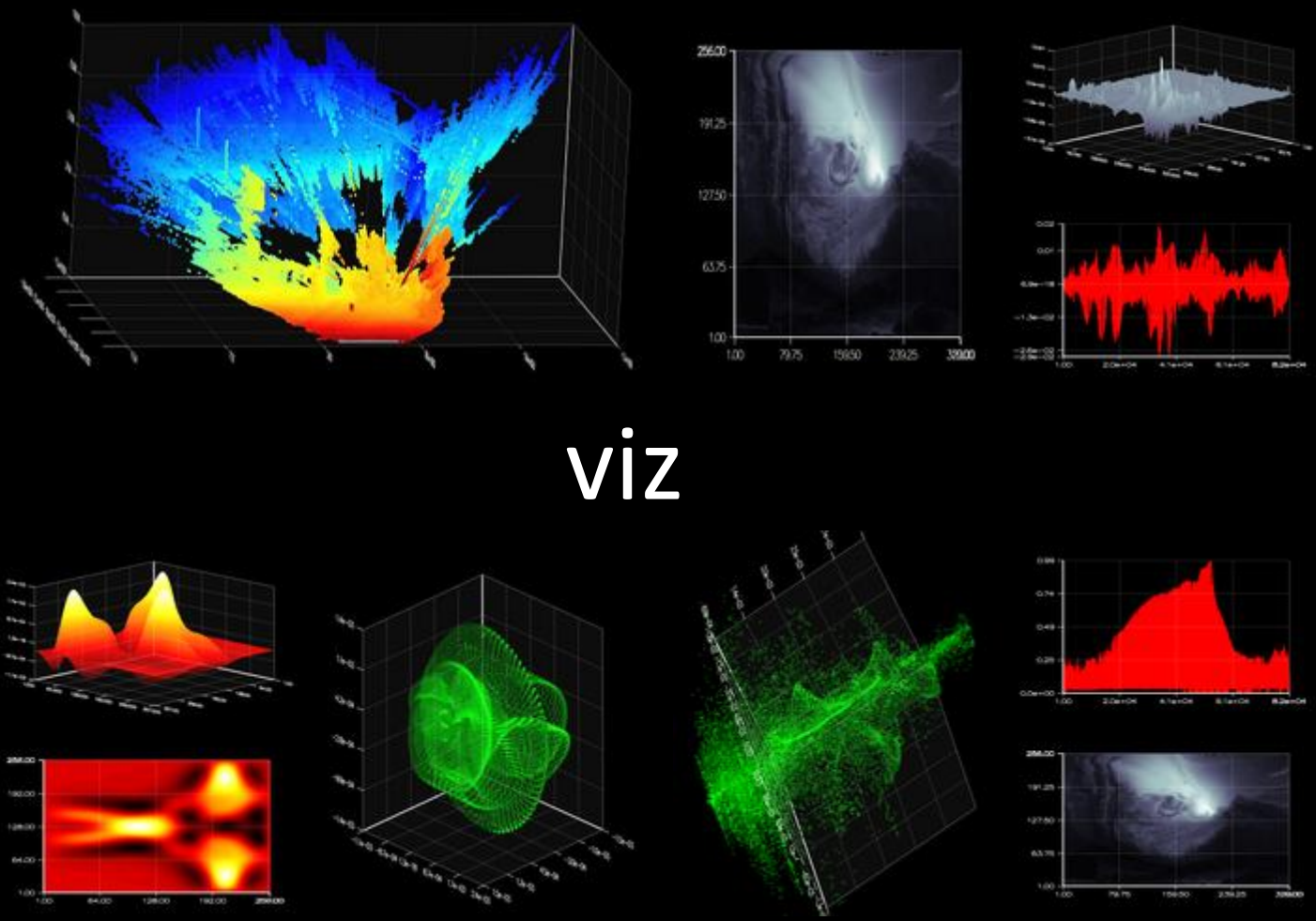


A(end,1)

A(end,span)

A(span,span,2)

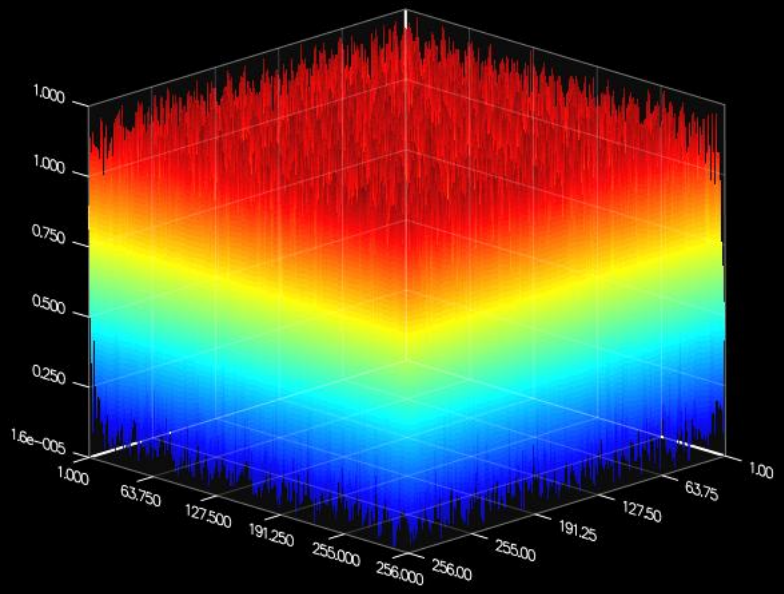



viz

Compute + Render

- asynchronous
- non-blocking
- throttled at 35 Hz

```
#include <arrayfire.h>
using namespace af;
int main() {
    // Infinite number of random 3d surfaces
    const int n = 256;
    while (1) {
        array x = randu(n,n);
        // 3d surface plot
        surface(x);
    }
    return 0;
}
```

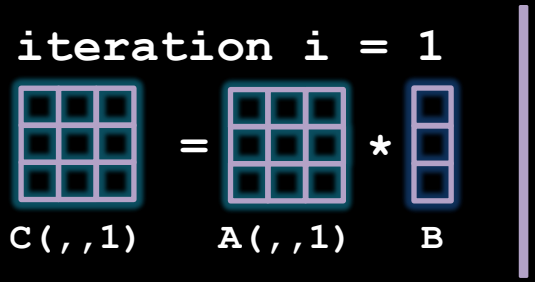


GFOR -- data parallel for-loops

Example: Matrix Multiply

Serial matrix multiplications (3 kernel launches)

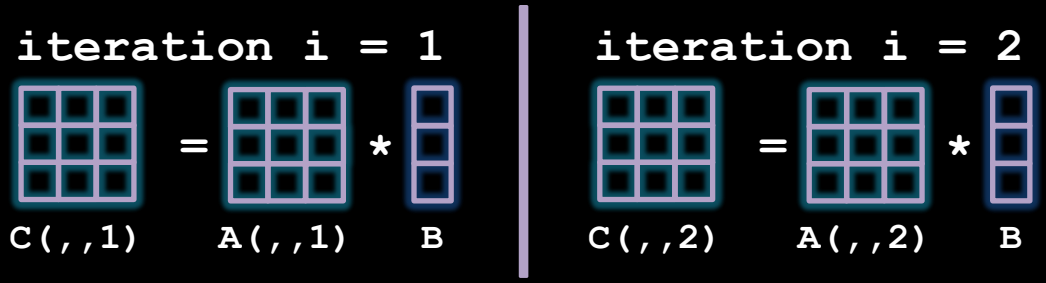
```
for (i = 0; i < 3; i++)
    C(span,span,i) = A(span,span,i) * B;
```



Example: Matrix Multiply

Serial matrix multiplications (3 kernel launches)

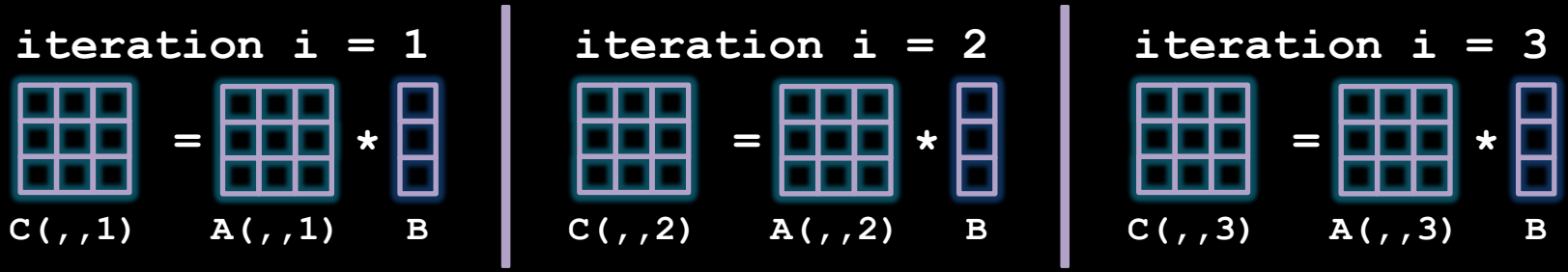
```
for (i = 0; i < 3; i++)
    C(span,span,i) = A(span,span,i) * B;
```



Example: Matrix Multiply

Serial matrix multiplications (3 kernel launches)

```
for (i = 0; i < 3; i++)
    C(span,span,i) = A(span,span,i) * B;
```



GFOR: data parallel for-loop

- Good for lots of small problems

Serial matrix multiplications (3 kernel launches)

```
for (i = 0; i < 3; i++)
    C(span,span,i) = A(span,span,i) * B;
```

Parallel matrix multiplications (**1** kernel launch)

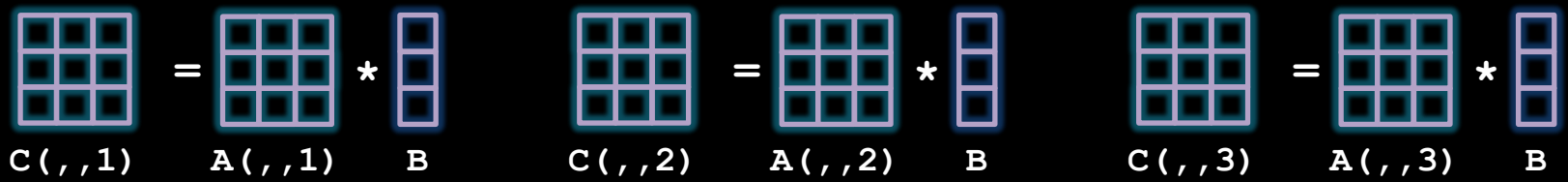
```
gfor (array i, 3)
    C(span,span,i) = A(span,span,i) * B;
```

Example: Matrix Multiply

Parallel matrix multiplications (**1** kernel launch)

```
gfor (array i, 3)
    C(span,span,i) = A(span,span,i) * B;
```

```
simultaneous iterations i = 1:3
```



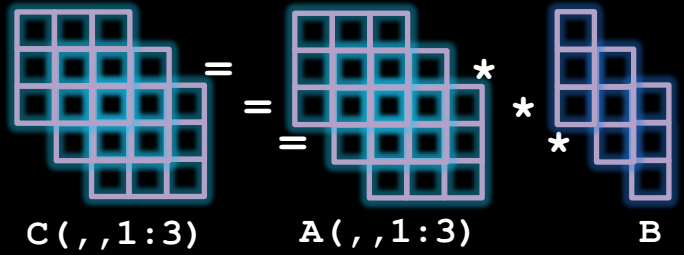
Example: Matrix Multiply

Parallel matrix multiplications (**1** kernel launch)

```
gfor (array i, 3)
    C(span,span,i) = A(span,span,i) * B;
```

Think of GFOR as compiling 1 stacked kernel with all iterations.

```
simultaneous iterations i = 1:3
```



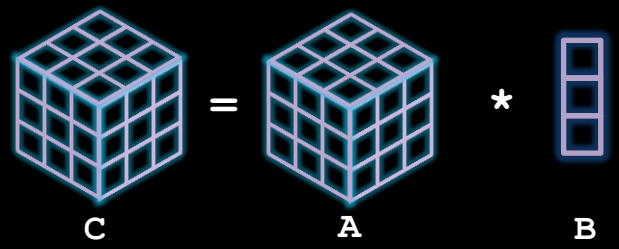
Example: Matrix Multiply

Parallel matrix multiplications (**1** kernel launch)

```
gfor (array i, 3)
    C(span,span,i) = A(span,span,i) * B;
```

Think of GFOR as compiling 1 stacked kernel with all iterations.

```
simultaneous iterations i = 1:3
```



Easy Multi GPU Scaling

```
array *y = new array[n];
for (int i = 0; i < n; ++i) {
    deviceset(i);                // change GPUs
    array x = randu(5,5);        // add work to GPU's queue
    y[i] = fft(x);               // more work in queue
}

// all GPUs are now computing simultaneously
```

Hundreds of Functions…

reductions
- sum, min, max, count, prod
- vectors, columns, rows, etc

dense linear algebra
- LU, QR, Cholesky, SVD, Eigenvalues, Inversion, Solvers, Determinant, Matrix Power

convolutions
- 2D, 3D, ND

FFTs
- 2D, 3D, ND

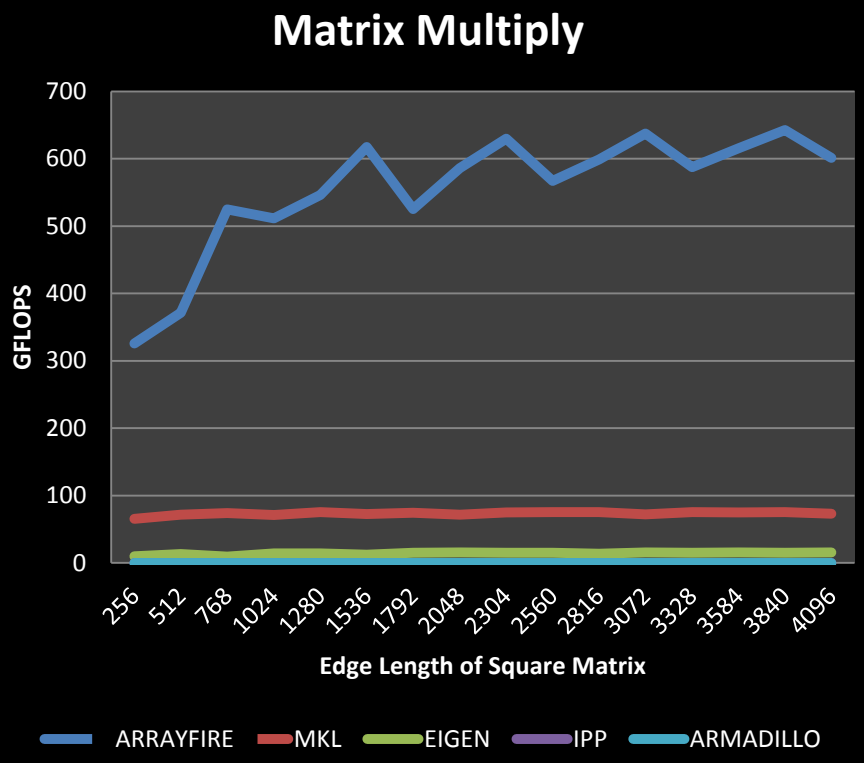
image processing
- filter, rotate, erode, dilate, morph, resize, rgb2gray, histograms

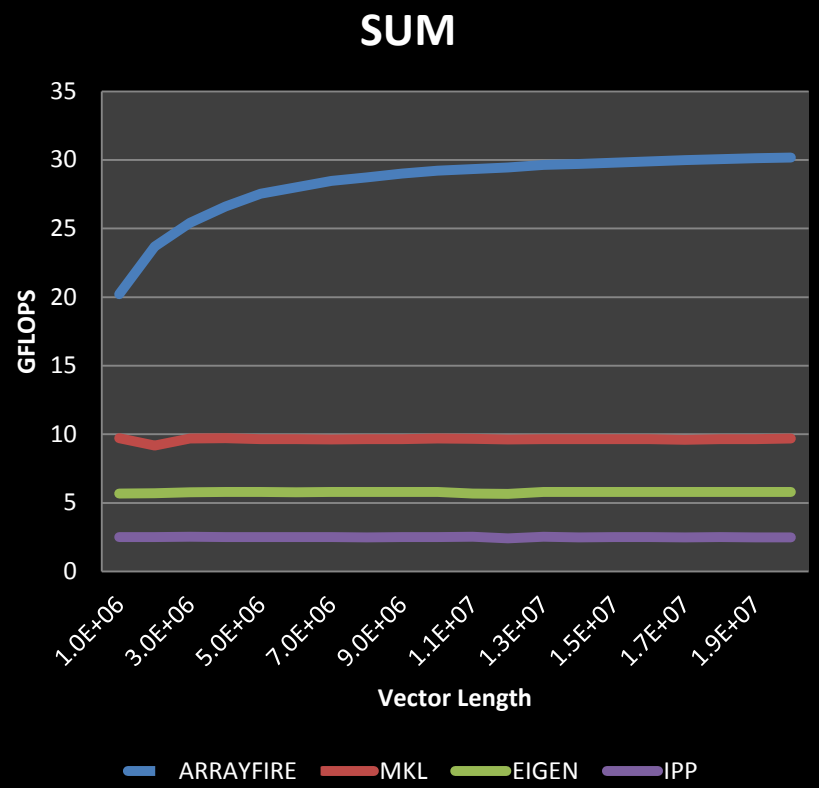
interpolate & scale
- vectors, matrices
- rescaling

sorting
- along any dimension
- sort detection

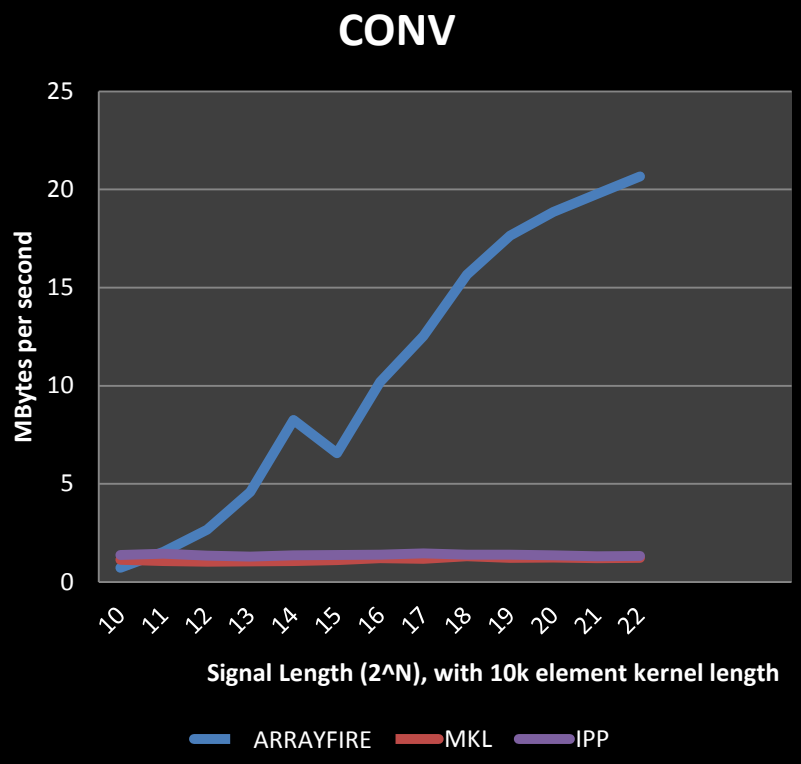
and many more…

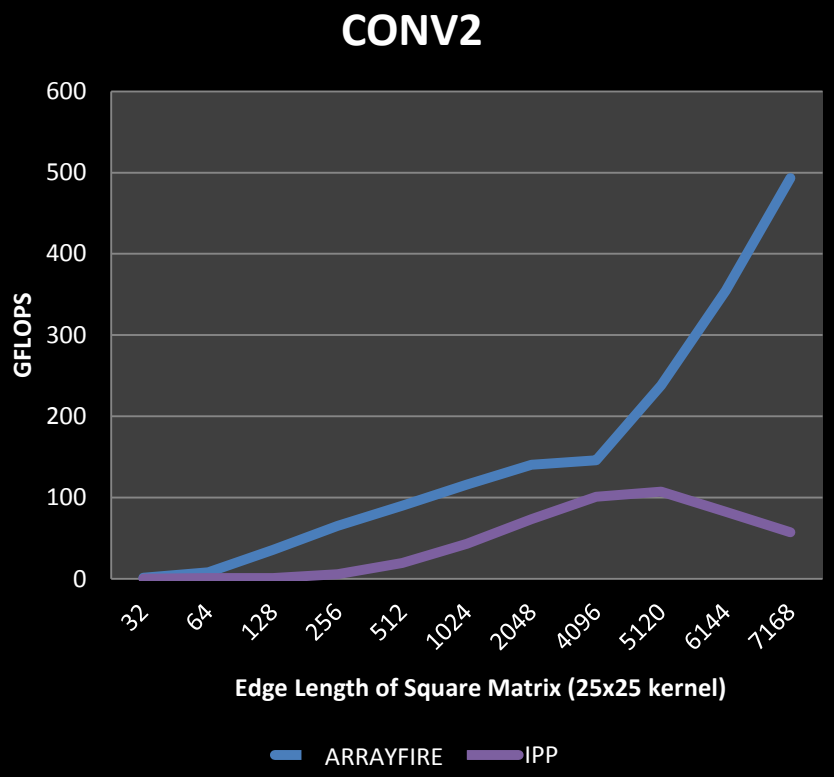
speed


Matrix Multiply
GFLOPS
700
600
500
400
300
200
100
0
256
512
768
1024
1280
1536
1792
2048
2304
2560
2816
3072
3328
3584
3840
4096
Edge Length of Square Matrix
ARRAYFIRE
MKL
EIGEN
IPP
ARMADILLO

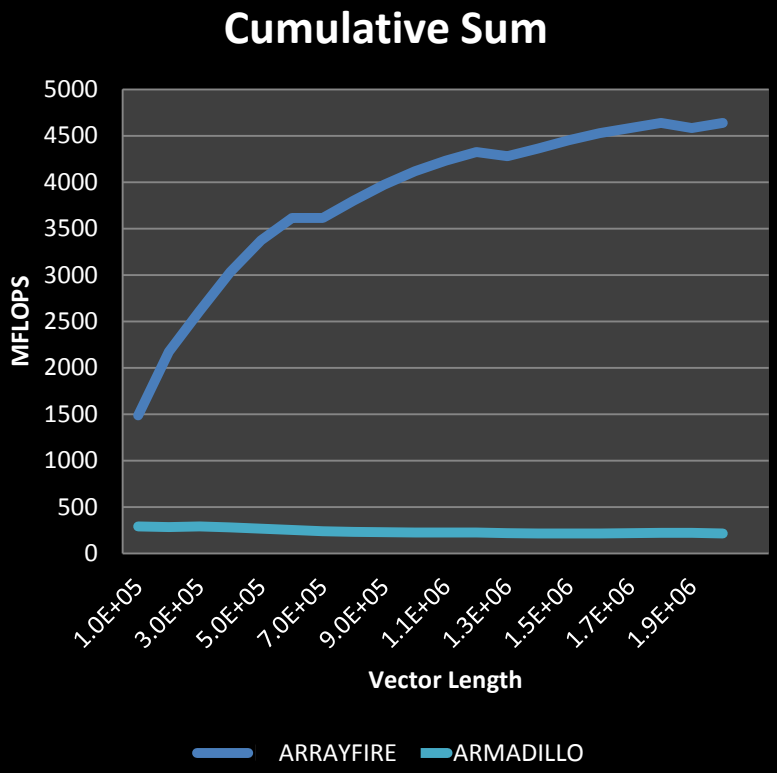

SUM
GFLOPS
35
30
25
20
15
10
5
0
1.0E+06
3.0E+06
5.0E+06
7.0E+06
9.0E+06
1.1E+07
1.3E+07
1.5E+07
1.7E+07
1.9E+07
Vector Length
ARRAYFIRE
MKL
EIGEN
IPP

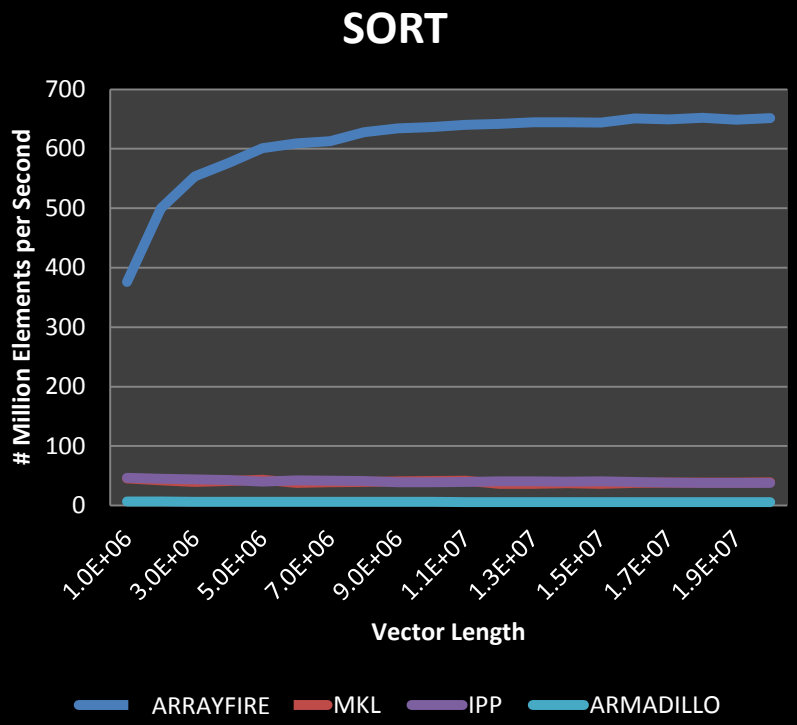
speed


CONV
MBytes per second
25
20
15
10
5
0
10
11
12
13
14
15
16
17
18
19
20
21
22
Signal Length (2^N), with 10k element kernel length
ARRAYFIRE
MKL
IPP


CONV2
GFLOPS
600
500
400
300
200
100
0
32
64
128
256
512
1024
2048
4096
5120
6144
7168
Edge Length of Square Matrix (25x25 kernel)
ARRAYFIRE
IPP

speed


Cumulative Sum
MFLOPS
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
1.0E+05
3.0E+05
5.0E+05
7.0E+05
9.0E+05
1.1E+06
1.3E+06
1.5E+06
1.7E+06
1.9E+06
Vector Length
ARRAYFIRE
ARMADILLO


SORT
Million Elements per Second
700
600
500
400
300
200
100
0
1.0E+06
3.0E+06
5.0E+06
7.0E+06
9.0E+06
1.1E+07
1.3E+07
1.5E+07
1.7E+07
1.9E+07
Vector Length
ARRAYFIRE
MKL
IPP
ARMADILLO

Easy To Maintain

- Write your code once
  - Each new release improves the speed of your code
  - Tesla, Fermi, Kepler, ..
- Behind the scenes…we update

Licensing

- Free
  - Full performance, single GPU, internet connection to license server
- Pro
  - Multi-GPU, full linear algebra, offline

Get Help

Custom development

- Port CPU code
- Extend existing code to use the GPU
- Numerical, image, signal, algebra, ..

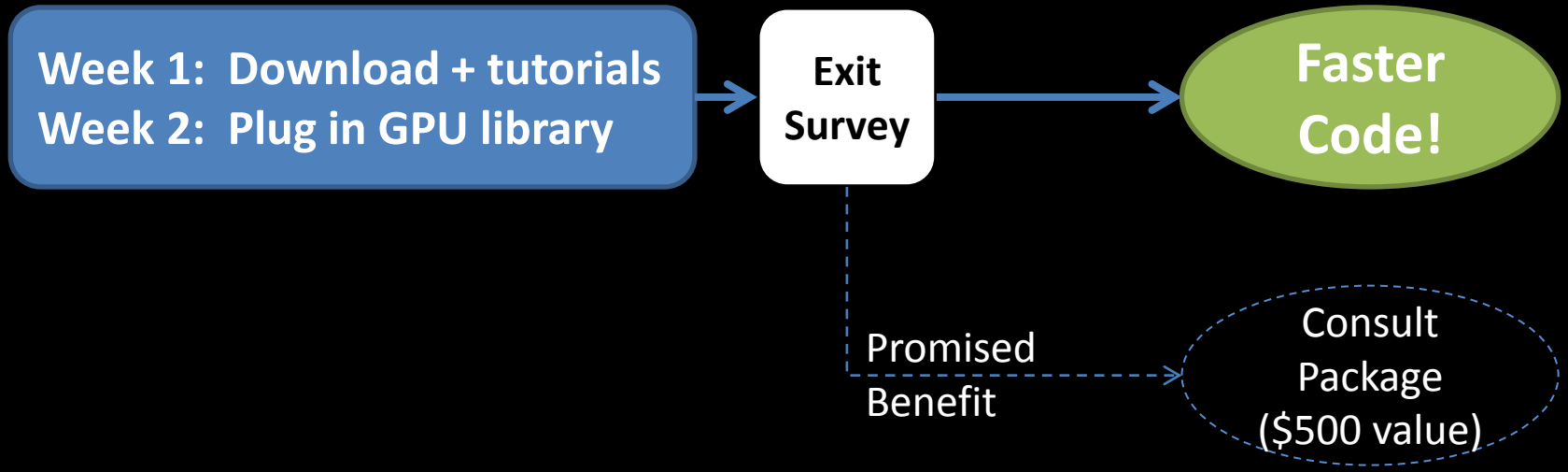
Training courses

- 1-4 days
- CUDA, OpenCL
- Hands on work with your code

Gear Up for Speedup

- NVIDIA and AccelerEyes guarantee 4X code improvements in just 2 weeks
- Use ArrayFire GPU library for CUDA!

How it works…


Week 1: Download + tutorials
Week 2: Plug in GPU library
Exit Survey
Faster Code!
Promised Benefit
Consult Package ($500 value)

What others are saying…

Dominic

“At Catsaras Investments Ltd, ArrayFire gave us 50X speedups on our financial models and is really easy-to-use. With ArrayFire, we avoid the time-sink of writing and optimizing GPU kernels by hand.”

Gear Up for Speedup!