Productive GPU Software

This conference uses integrated audio. To interact with the host, you need a working microphone and speakers.

To speak, please click on the “Raise Hand” button in the Participants box. You can speak into your microphone after the host allows you to.

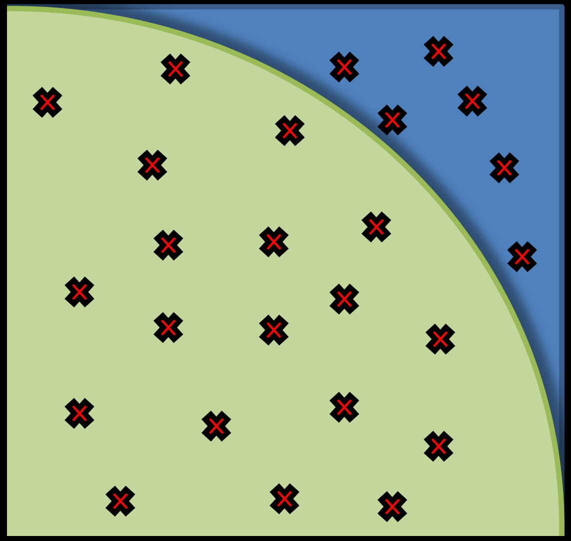
If you cannot hear the host, or if your voice is not being transmitted, please let us know using the Chat window.

Outline

- Introduction to Jacket for MATLAB®
- GFOR
- Comparison with PCT™ alternative
- Moving into the future
- Case studies and code demos

Easy GPU Acceleration of M code

```
n = 20e6;  % 20 million random samples
X = grand(1,n,'gdouble');
Y = grand(1,n,'gdouble');
distance_to_origin = sqrt( X.*X + Y.*Y );
is_inside = (distance_to_origin <= 1);
pi = 4 * sum(is_inside) / n;
```



Matrix Types

gdouble
double precision

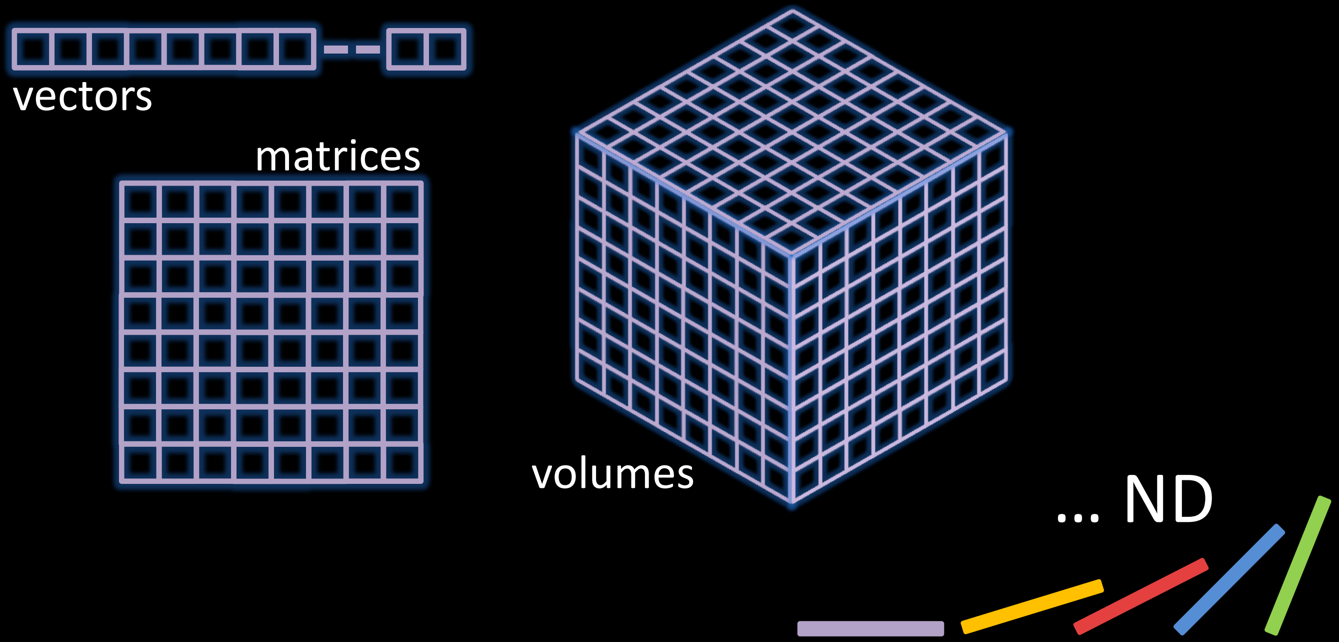
glogical
boolean

guint#
unsigned integers

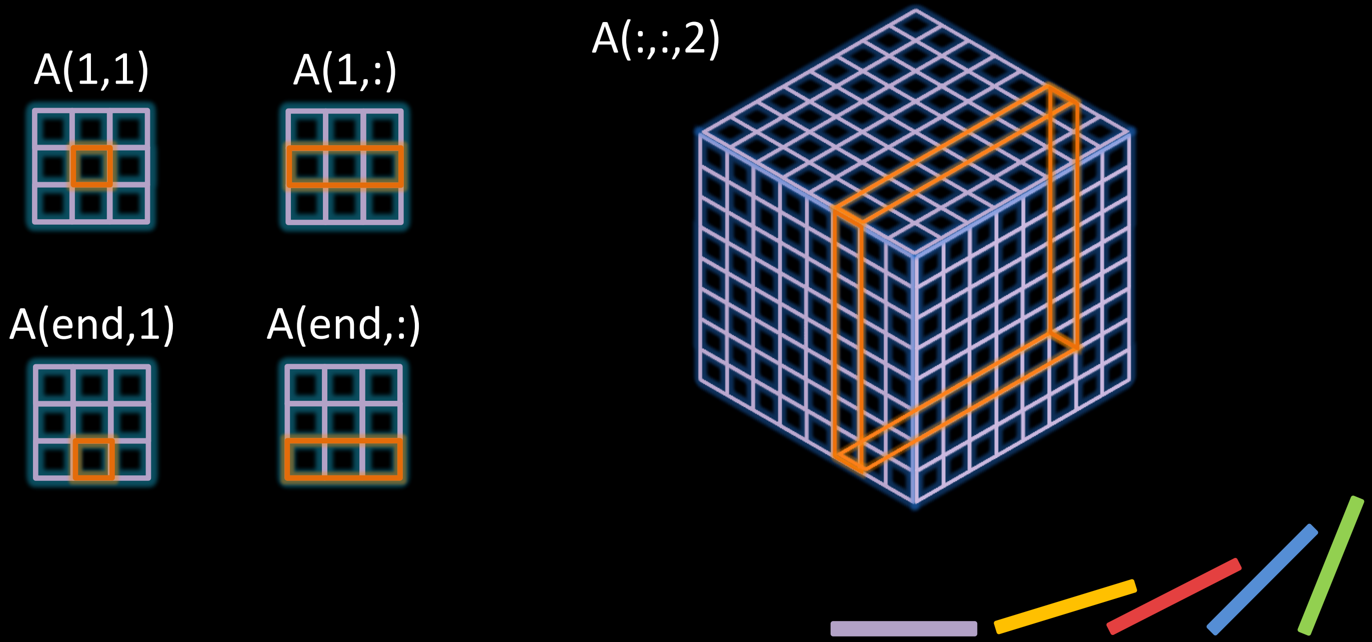
gint#
integers

gsingle
single precision

Matrix Types: ND Support

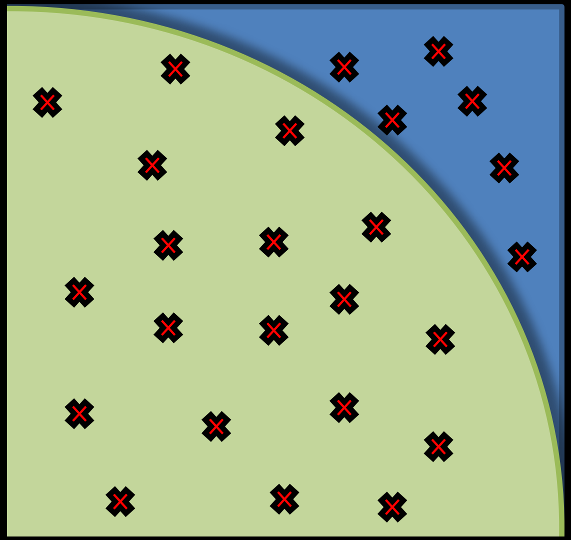


Matrix Types: Easy Manipulation



Easy GPU Acceleration of M code

```
n = 20e6;  % 20 million random samples
X = grand(1,n);
Y = grand(1,n);
distance_to_origin = sqrt( X.*X + Y.*Y );
is_inside = (distance_to_origin <= 1);
pi = 4 * sum(is_inside) / n;
```



Easy GPU Acceleration of M code

No GPU-specific stuff involved (no kernels, no threads, no blocks, just regular M code)

"Very little recoding was needed to promote our Lattice Boltzmann Model code to run on the GPU." –Dr. Kevin Tubbs, HPTi

GFOR – Parallel FOR-loop for GPUs

- Like a normal FOR-loop, but faster

Regular FOR-loop (3 serial kernel launches)

```
for i = 1:3
    C(:,:,i) = A(:,:,i) * B;
```

Parallel GPU FOR-loop (only **1** kernel launch)

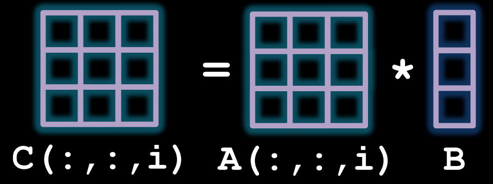
```
gfor i = 1:3
    C(:,:,i) = A(:,:,i) * B;
```

Example: Matrix Multiply

Regular FOR-loop (3 serial kernel launches)

```
for i = 1:3
    C(:,:,i) = A(:,:,i) * B;
```

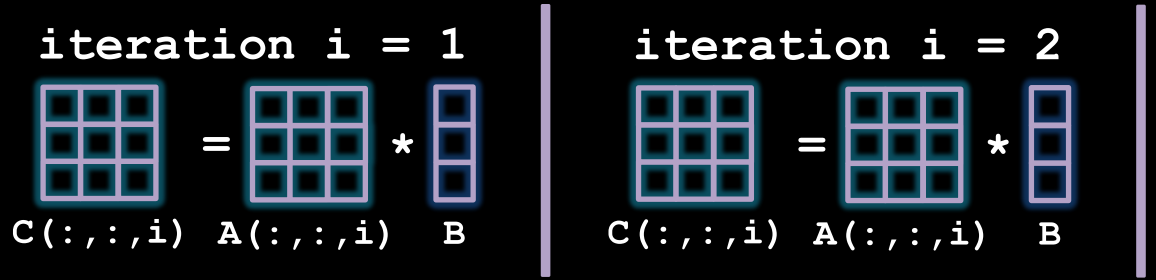
iteration i = 1



Example: Matrix Multiply

Regular FOR-loop (3 serial kernel launches)

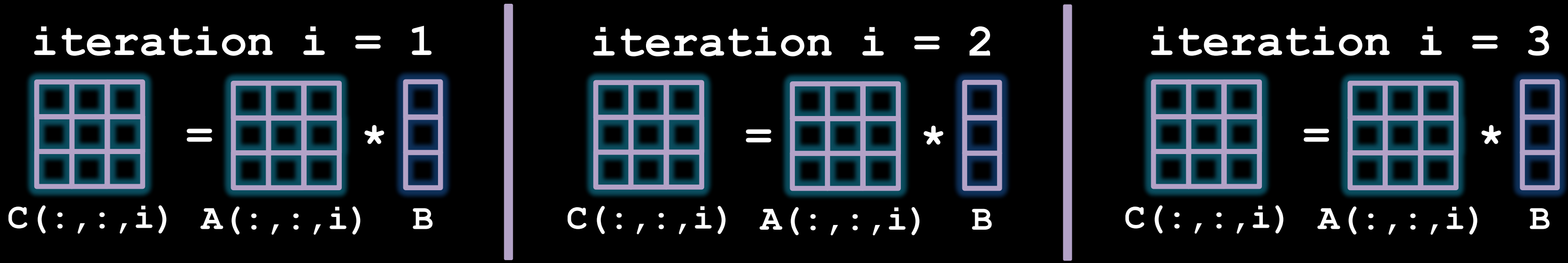
```
for i = 1:3
    C(:,:,i) = A(:,:,i) * B;
```



Example: Matrix Multiply

Regular FOR-loop (3 serial kernel launches)

```
for i = 1:3
    C(:,:,i) = A(:,:,i) * B;
```

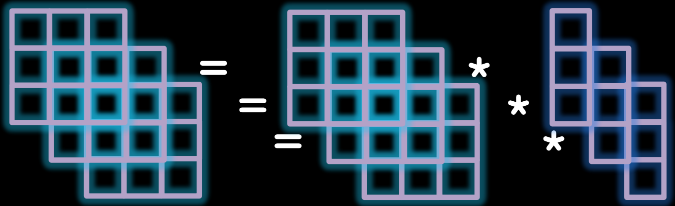


Example: Matrix Multiply

Parallel GPU FOR-loop (only **1** kernel launch)

```
gfor i = 1:3
    C(:,:,i) = A(:,:,i) * B;
```

simultaneous iterations i = 1:3



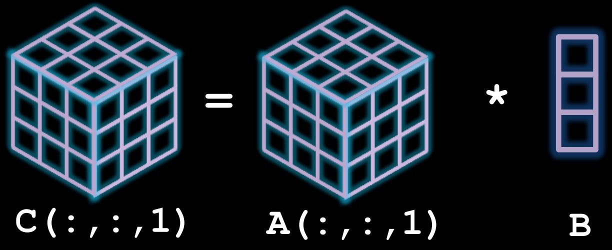
C(:,:,1:3) A(:,:,1:3) B

Example: Matrix Multiply

Parallel GPU FOR-loop (only **1** kernel launch)

```
gfor i = 1:3
    C(:,:,i) = A(:,:,i) * B;
```

`simultaneous iterations i = 1:3`



Example: Summing over Columns

- Think of gfor as "syntactic sugar" to write vectorized code in an iterative style.

Three passes to sum all columns of B

```
for i = 1:3
    A(i) = sum(B(:,i));
```

One pass to sum all columns of B

```
gfor i = 1:3
    A(i) = sum(B(:,i));
```

Both equivalent to "**sum(B)**", but latter is faster (more explicitly written)

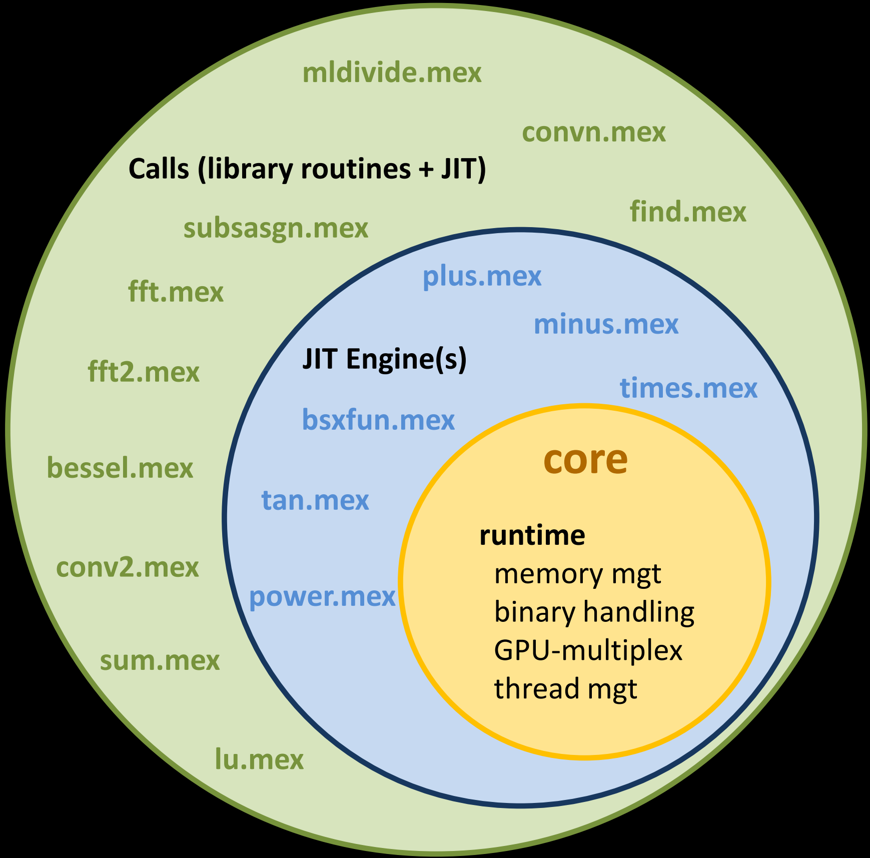
Easy Multi GPU Scaling

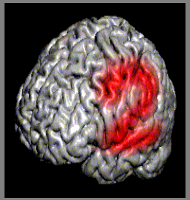
```
y = gzeros( 5, 5, n );
for i = 1:n,
    gselect(i);              % choose GPU for this iteration
    x = grand(5,5);          % add work to GPU's queue
    y(:,:,i) = fft(x);       % more work in queue
end

% all GPUs are now computing simultaneously, until done
```

Technology Stack

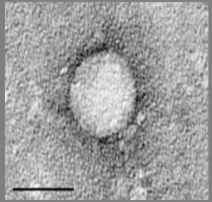
- A full system making optimizations for you
- Including
  - "Core" brains
  - "JIT" speed
  - "Calls" heavy-lifting





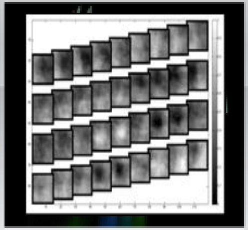
17X

Neuro-imaging
Georgia Tech



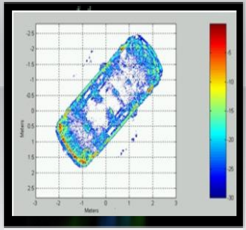
20X

Bio-Research
CDC



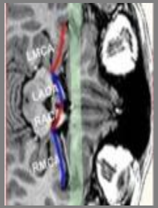
20X

Video Processing
Google



45X

Radar Imaging
System Planning



12X

Medical Devices
Spencer Tech

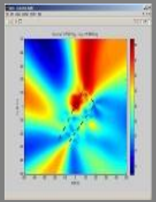
http://www.accelereyes.com/case_studies

5X

Weather Modeling
NCAR

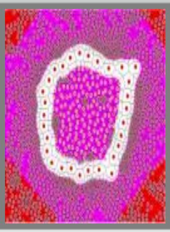
35X

Power Engineering
IIT India



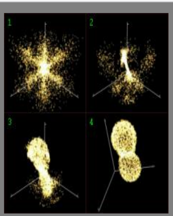
17X

Track Bad Guys
BAE
Systems



70X

Drug Delivery
Georgia Tech

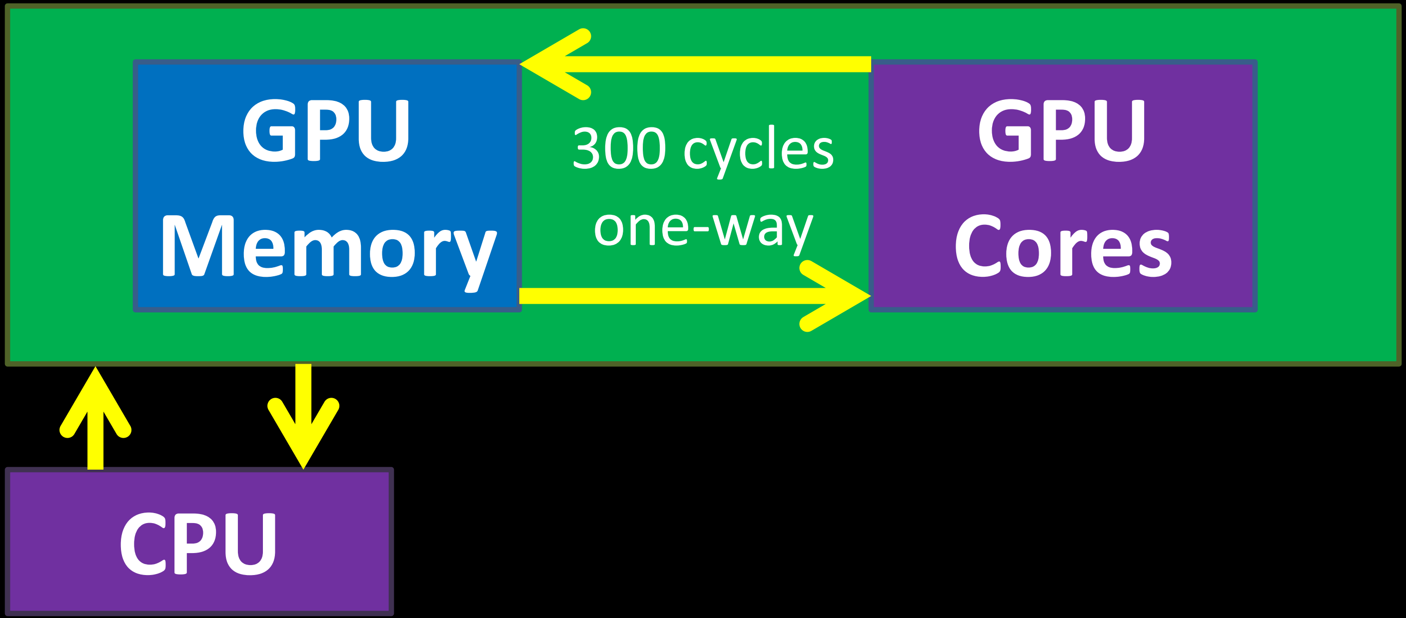


35X

Bioinformatics
Leibniz

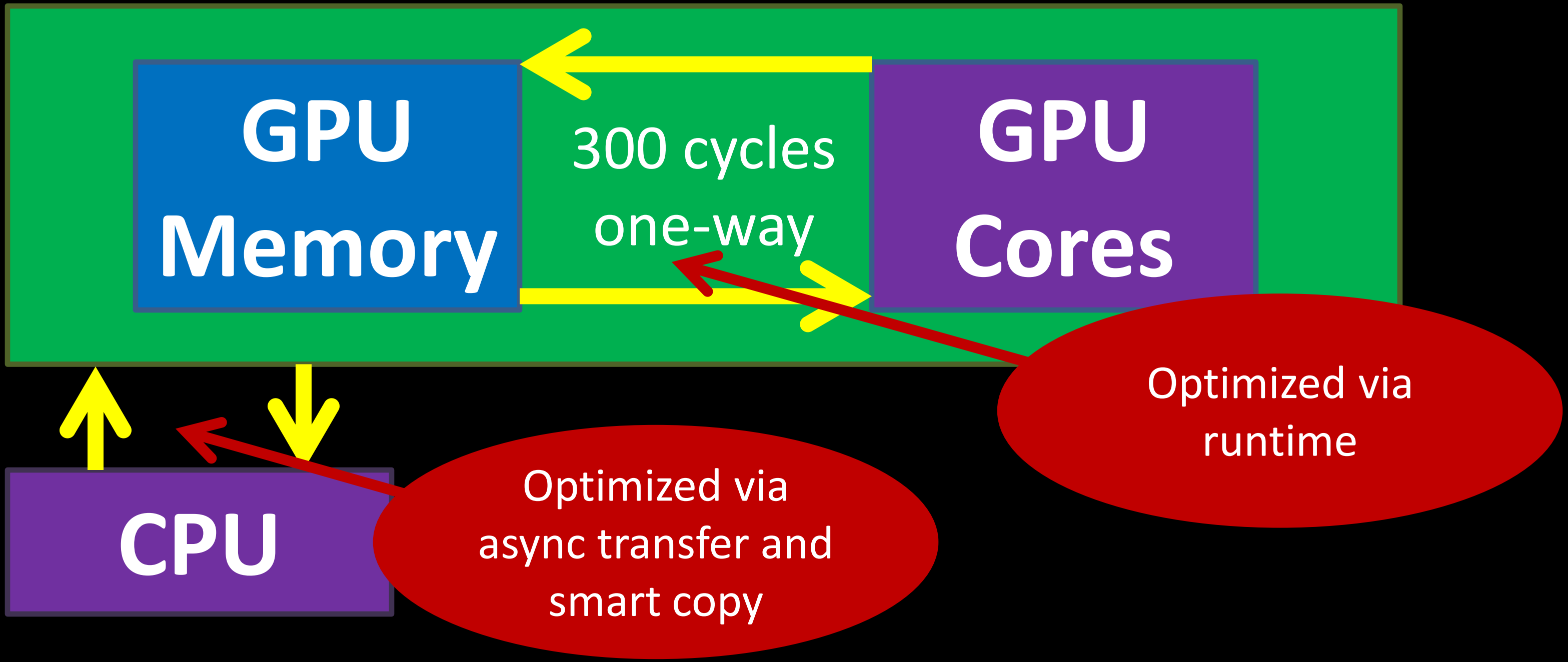
Automated Optimizations

```
A = sin( x + y ).^2
```



Automated Optimizations

```
A = sin( x + y ).^2
```



Compare versus PCT { parallel computing toolbox™

$$A = \sin(x + y).\hat{}2$$

<u>PCT</u>
Load x, y (300 cycles)
+ (4 cycles)
Store Temp1 (300 cycles)
Load Temp1 (300 cycles)
Sin (~20 cycles)
Store Temp2 (300 cycles)
Load Temp2 (300 cycles)
.^ (~10 cycles)
Store A (300 cycles)

<u>Jacket</u>
Load x, y (300 cycles)
Sin(x+y).^2 (34 cycles)
Store A (300 cycles)

Compare versus PCT { parallel computing toolbox™

```
A = sin( x + y ).^2
```

<u>PCT</u>

Load x, y (300 cycles)
+ (4 cycles)
Store Temp1 (300 cycles)
Load Temp1 (300 cycles)
Sin (~20 cycles)
Store Temp2 (300 cycles)
Load Temp2 (300 cycles)
.^ (~10 cycles)
Store A (300 cycles)

1834 cycles

<u>Jacket</u>

Load x, y (300 cycles)
Sin(x+y).^2 (34 cycles)
Store A (300 cycles)

634 cycles

Compare versus PCT { parallel computing toolbox™

$$A = \sin(x + y).\hat{}2$$

PCT

Load x, y (300 cycles)
+ (4 cycles)
Store Temp1 (300 cycles)
Load Temp1 (300 cycles)
Sin (~20 cycles)
Store Temp2 (300 cycles)
Load Temp2 (300 cycles)
.^ (~10 cycles)
Store A (300 cycles)

1834 cycles

Jacket

Load x, y (300 cycles)
Sin(x+y).^2 (34 cycles)
Store A (300 cycles)

634 cycles

Theoretically, a 3x increase. Actually, a 20x difference:

- **Legacy Java system**
- **Better GPU code**

Jacket has 10X more functions…

gfor (loops)

reductions
- sum, min, max, any, all, nnz, prod
- vectors, columns, rows, etc

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

gcompile (fine-grain)

convolutions
- 2D, 3D, ND

FFTs
- 2D, 3D, ND

image processing
- filter, rotate, erode, dilate, bwmorph, resize, rgb2gray
- hist, histeq

gselect (multi-GPU)

interp and rescale
- vectors, matrices
- rescaling

sorting
- along any dimension
- find

help
- gprofview

and many more…

Easy To Maintain

- Write your code once and let Jacket carry you through the coming hardware evolution.
  - Each new Jacket release improves the speed of your code, without any code modification.
  - Each new Jacket release leverages latest GPU hardware (e.g. Fermi, Kepler), without any code modification.

New in Jacket 2.1: Optimization

- Unconstrained Optimization in 2.1
  - Gradient Descent and BFGS methods
  - Jacobian computation with GFOR
- Batched-mode Optimization in 2.2
- Search-based Optimization in 2.2
- Constrained Optimization in 2.3

Sparse Roadmap

Current functions supported:

- Matrix multiply
- Triangular matrix solve
- Iterative solvers with no pre-conditioning.
- Examples: CG, BICG, BICGSTAB, BICGSTABL, GMRES, LSQR

Under development:

- Iterative solvers with pre-conditioning and improved performance
- Examples: CG, BICG, BICGSTAB, GMRES

Move to C/C++, Fortran, or Python

ArrayFire GPU library

- Free version for most users (single GPU usage)
- Pro version (multi-GPU usage)
- Available for CUDA or OpenCL devices

ArrayFire

The World's Largest, Fastest GPU Library

ArrayFire Example (C++)

```
#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(mul(x,x)+mul(y,y))<1) / n;
    printf("pi = %g\n", pi);
    return 0;
}
```

Case Studies

See more examples:

http://www.accelereyes.com/examples/case_studies

http://blog.accelereyes.com/blog/

Case Study: Australian Brokerage

- Description: Nonlinear regressive model fitting
- Speedup: 115x
- Solution: Jacket, Jacket DLA, ArrayFire Pro, Consulting

Case Study: Australian Brokerage

- Description: Modified conjugate gradient for sparse matrices
- Speedup: 10-30x (Depends on data size. Larger data gives bigger speedups.)
- Solution: Jacket, Jacket SLA, ArrayFire Pro, Consulting

Case Study: Koch Industries

- Description: Option pricing based on Monte-Carlo simulation
- Speedup: 60 - 70x
- Solution: Jacket

Case Study: Bank of America

- Description: Visualization of server utilization and workloads, required to run in MATLAB®
- Focus only on visualization, not computation
- Result: Beautiful OpenGL 3D renderings
- Solution: Jacket with the Graphics Library

Automotive Trader Example

- Description: Algorithmic trading
- Speedup: 37x on 3 GPUs (14x on 1 GPU)
- Solution: Jacket, Jacket MGL for 3 GPUs
- Learn more: http://www.automatedtrader.net/articles/software-review/107768/mashup

Demos

Discussion

jacket

Faster MATLAB® through GPU computing