ACCELEREYES®

Jacket: Faster MATLAB® Genomics Codes

by Chris McClanahan, GPU Engineer

Outline

- Introduction to Jacket for MATLAB®
- GFOR
- Comparison with GPU-PCT alternative
- Case Studies: Genomics Examples

Matrix Types

gdouble
double precision

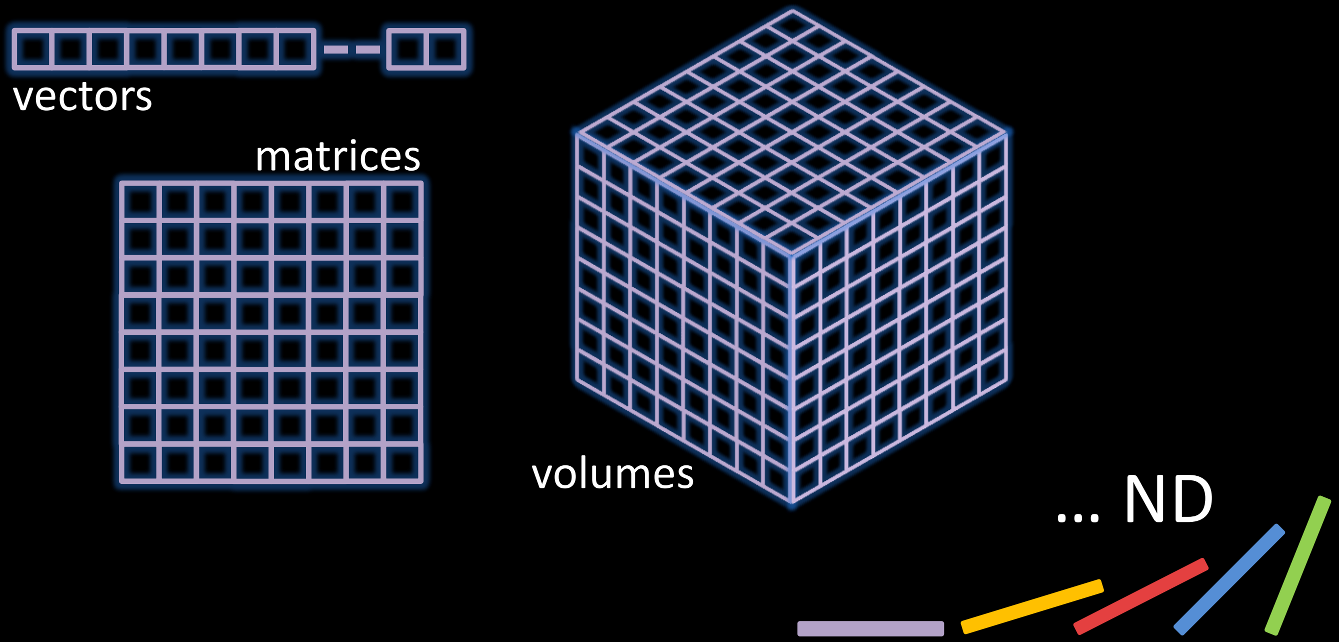
glogical
boolean

guint#
unsigned integers

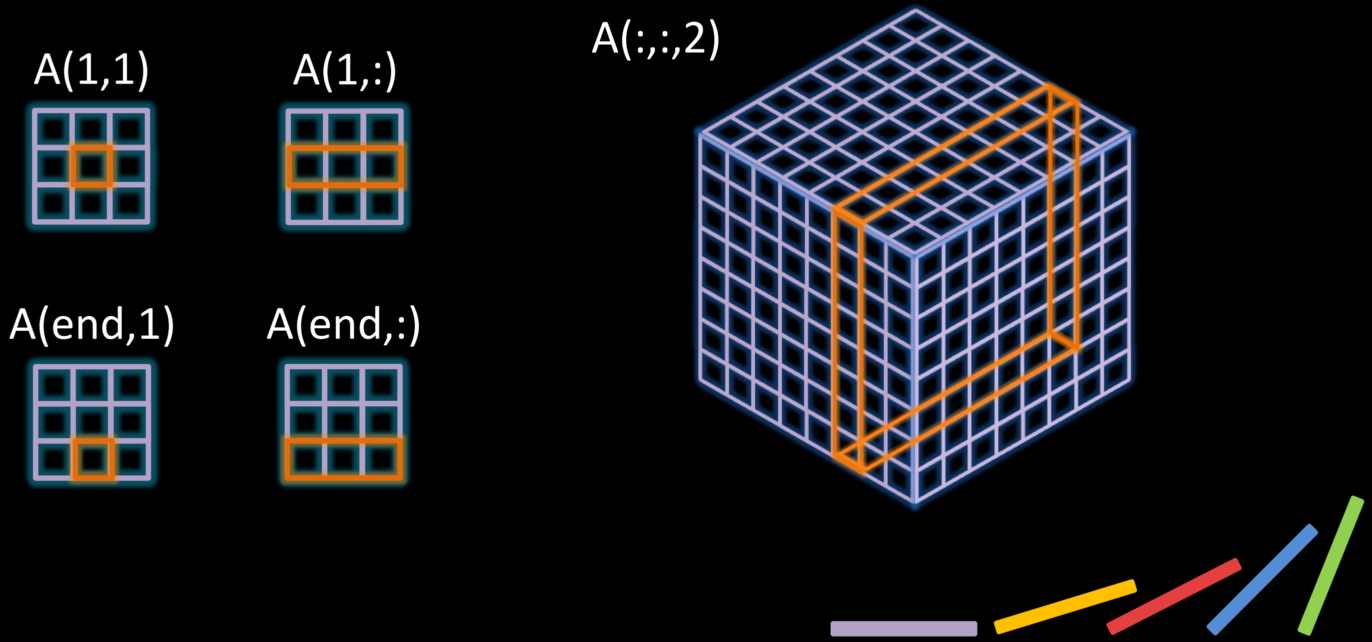
gint#
integers

gsingle
single precision

Matrix Types: ND Support

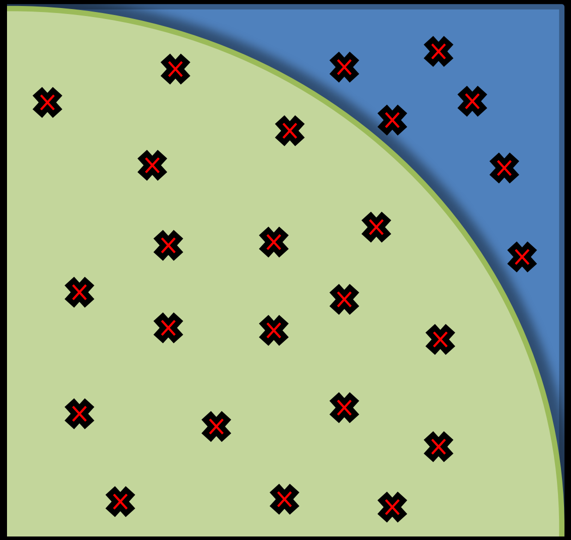


Matrix Types: Easy Manipulation



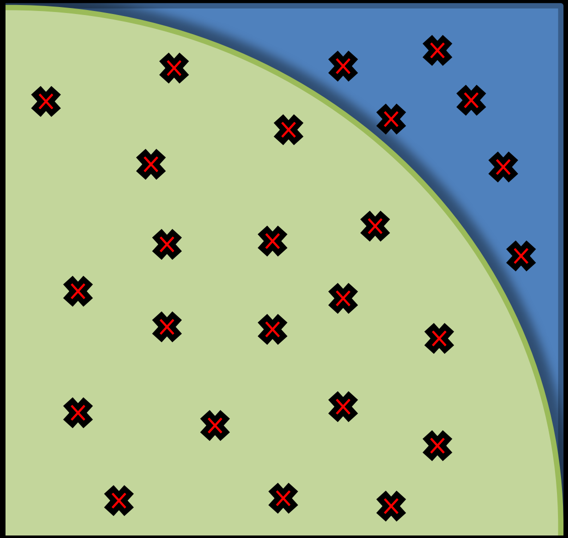
Easy GPU Acceleration of M code

```
n = 20e6;  % 20 million random samples
X = grand(1,n,'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;
```



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;
```



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

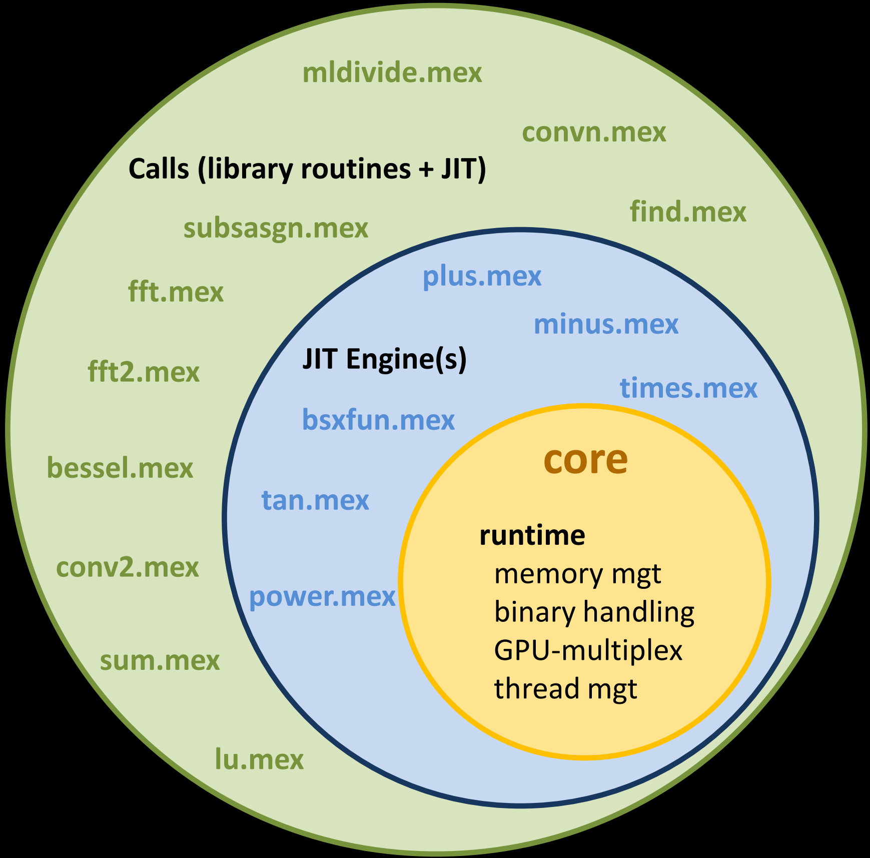
Easy Multi GPU Scaling

```
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
```

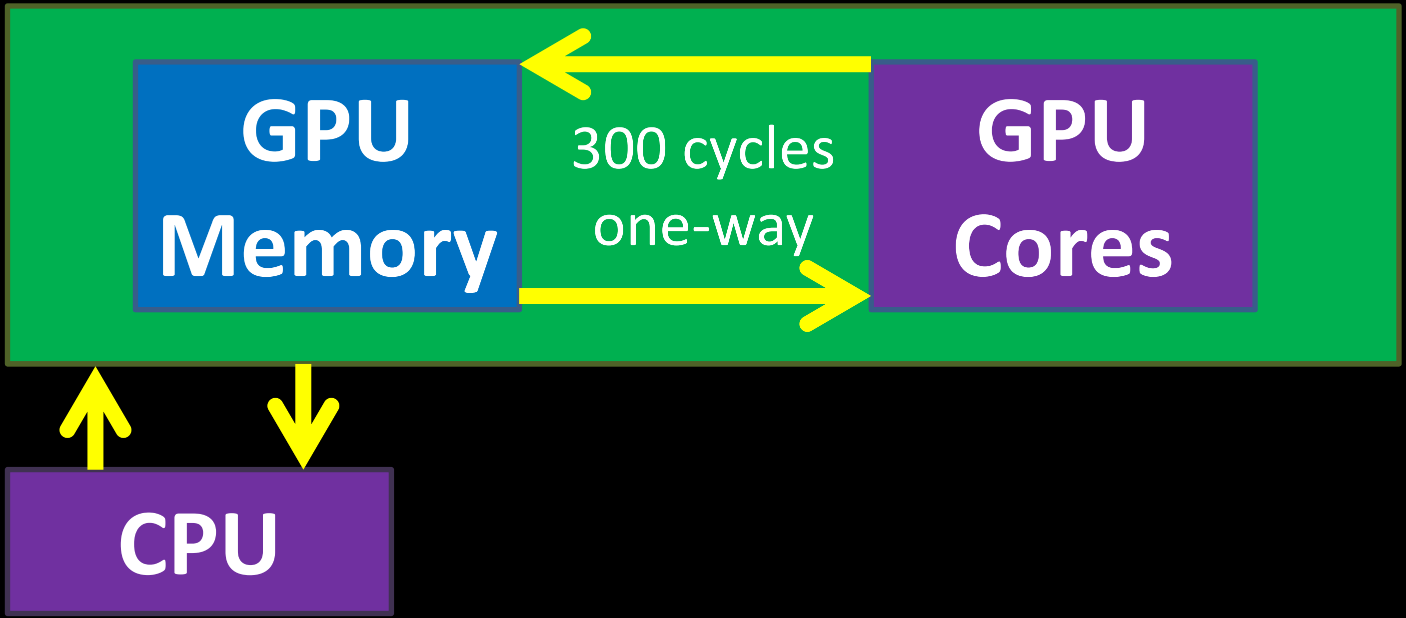
Technology Stack

- A full system making optimizations for you
- Including
  - “Core” runtime
  - “JIT” smart copy/exec
  - “Calls” functionality



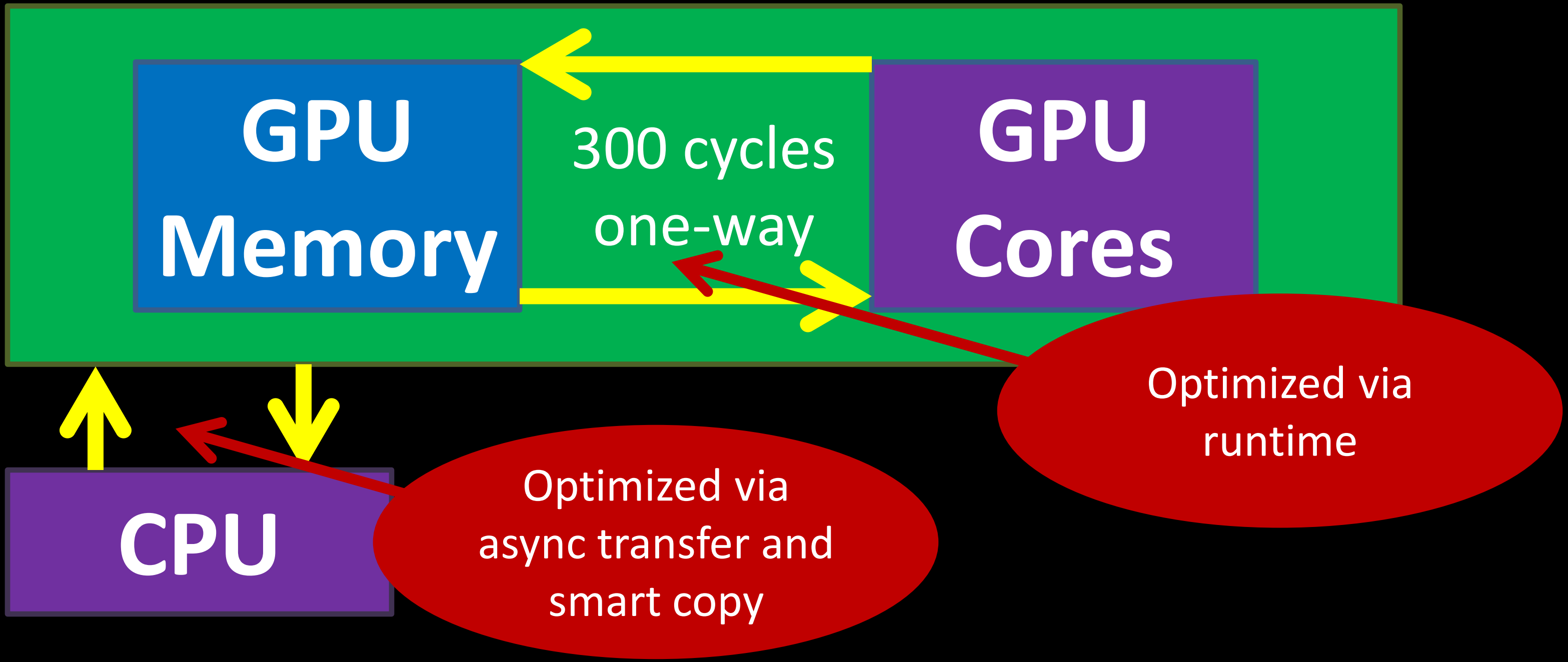
Automated Optimizations

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



Automated Optimizations

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



GFOR

GPU FOR-loops

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)

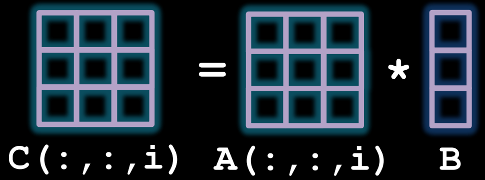
```
gfor i = 1:3
    C(:,:,i) = A(:,:,i) * B;
```

Example: Matrix Multiply

Regular FOR-loop (3 serial kernel launches)

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

iteration i = 1

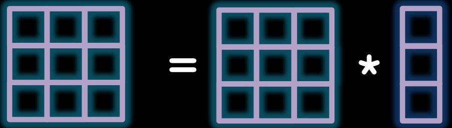


Example: Matrix Multiply

Regular FOR-loop (3 serial kernel launches)

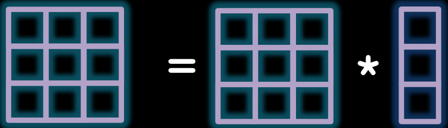
```
for i = 1:3
    C(:,:,i) = A(:,:,i) * B;
```

iteration i = 1



C(:,:,i) A(:,:,i) B

iteration i = 2

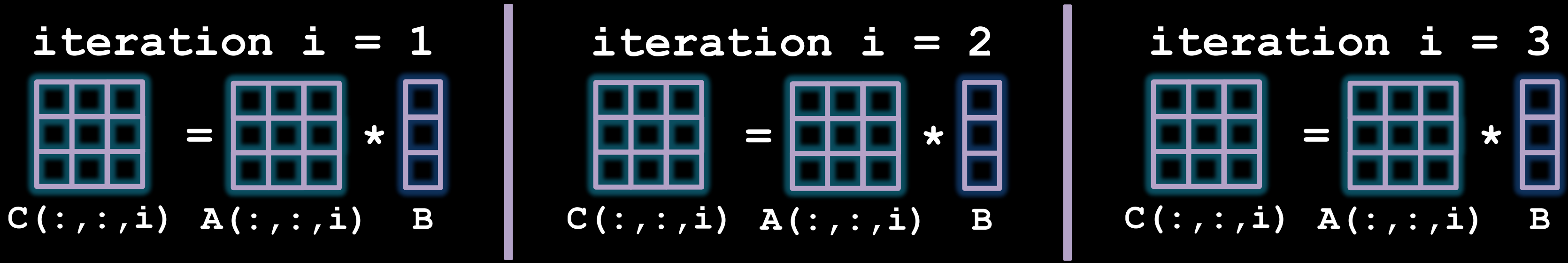


C(:,:,i) A(:,:,i) B

Example: Matrix Multiply

Regular FOR-loop (3 serial kernel launches)

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

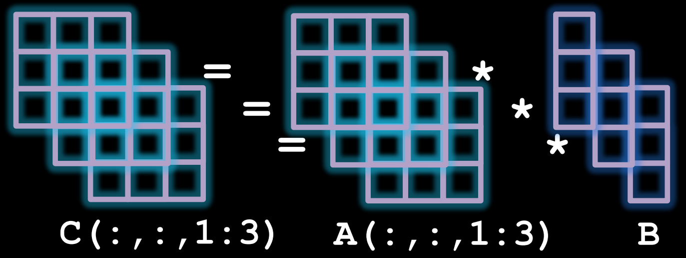


Example: Matrix Multiply

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

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

simultaneous iterations i = 1:3

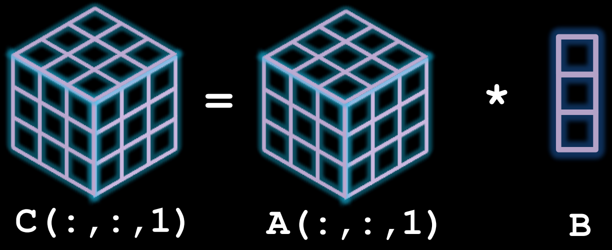


Example: Matrix Multiply

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)

Jacket versus PCT

$\left\{ \text{parallel computing toolbox} \right.$

Compare with R2012a PCT

- Jacket is faster
- Jacket does not use Java
- Jacket is very mature (~5 years)
- Jacket includes more functionality

Speedups with Jacket

	Jacket speedup over R2012A PCT™
Arithmetic: 1 + 2*x + 3*x.^2 + 4*x.^3 + 5*x.^4	109x
Trigonometric Functions: cos	76x
Sorting: sort	41x
Convolution 2D: conv2	29x
Elementwise Multiplication: times	19x
Data Manipulation: reshape	19x
Exponential Power: power	11x
Linear Algebra: transpose	5x
Convolution 1D: conv	3x
Search: find	3x
Reductions: sum, min, max	2x
Linear Algebra: matrix inverse, lu, qr, mldivide	1.4x

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

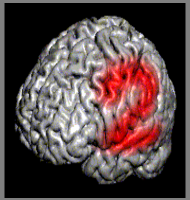
help
- gprofview

and many more…

Easy To Maintain

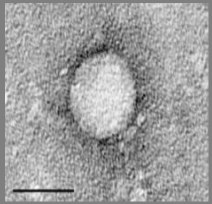
- Write your code once and let Jacket carry you through the coming hardware evolution.
  - Each new Jacket release improves the speed of your code, without any code modification.
  - Each new Jacket release leverages latest GPU hardware (e.g. Fermi, Kepler), without any code modification.

Case Studies



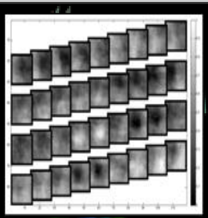
17X

Neuro-imaging
Georgia Tech



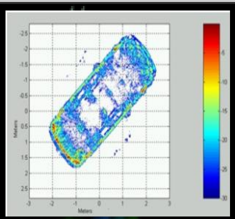
20X

Bio-Research
CDC



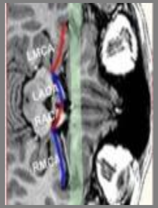
20X

Video Processing
Google



45X

Radar Imaging
System Planning



12X

Medical Devices
Spencer Tech

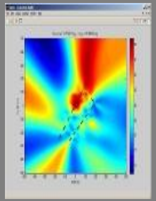
http://www.accelereyes.com/case_studies

5X

Weather Modeling
NCAR

35X

Power Engineering
IIT India

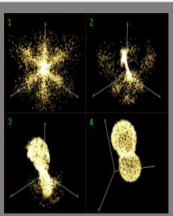


17X

Track Bad Guys
BAE Systems

70X

Drug Delivery
Georgia Tech



35X

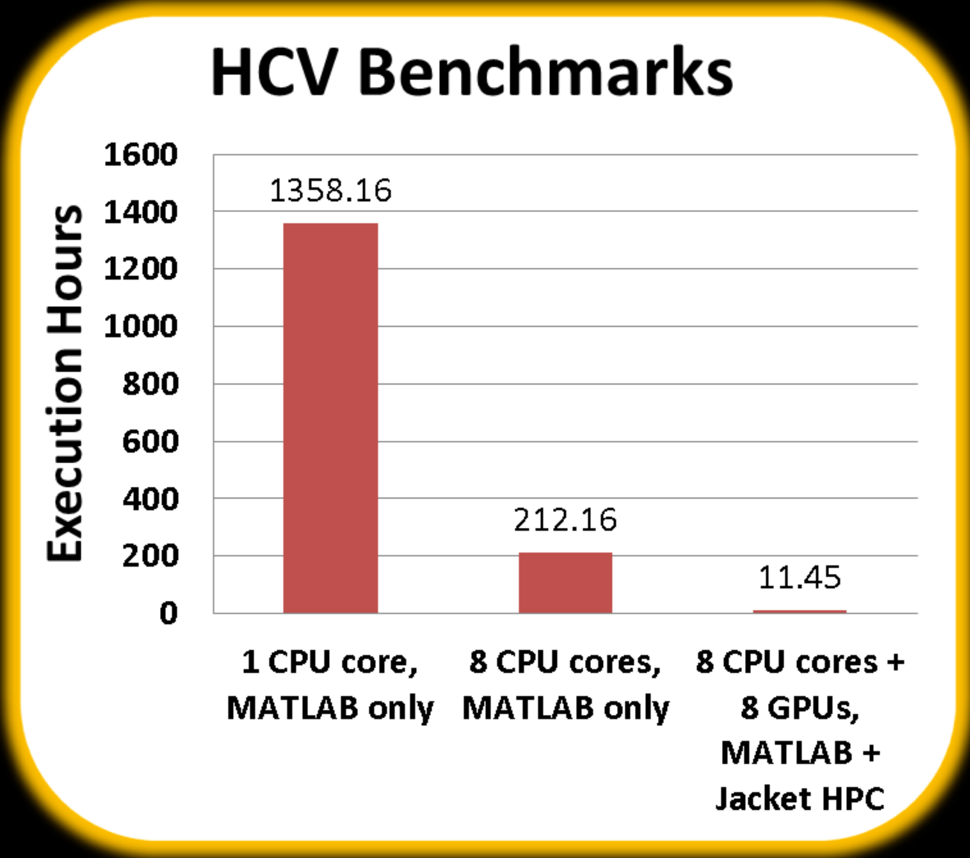
Bioinformatics
Leibniz

Case Study: CDC Genomics

- Hepatitis C Virus (HCV)
- Goal to explore random genetic mutations
- 10,000 random alignments
  - simulating the distribution of correlation values under the null hypothesis that substitutions of amino acids at two sites are statistically independent (how aa's mutate HCV)

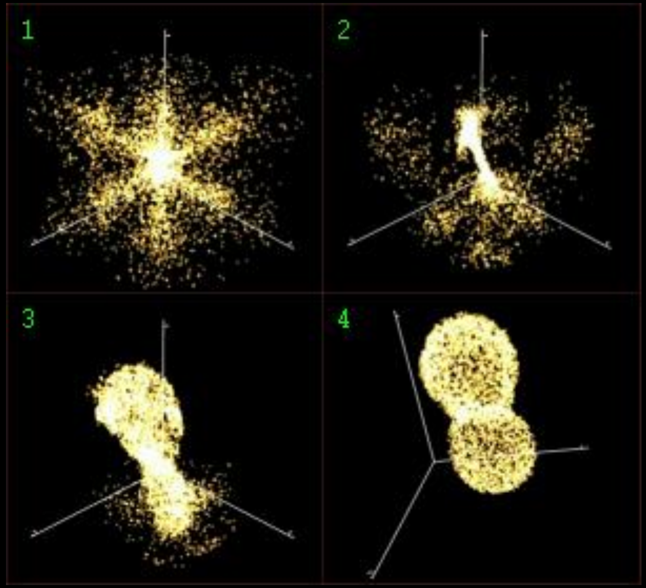
Case Study: CDC Genomics

- 10,000 random alignments intractable on CPU
- Addition of GPUs brings ~18X speedup



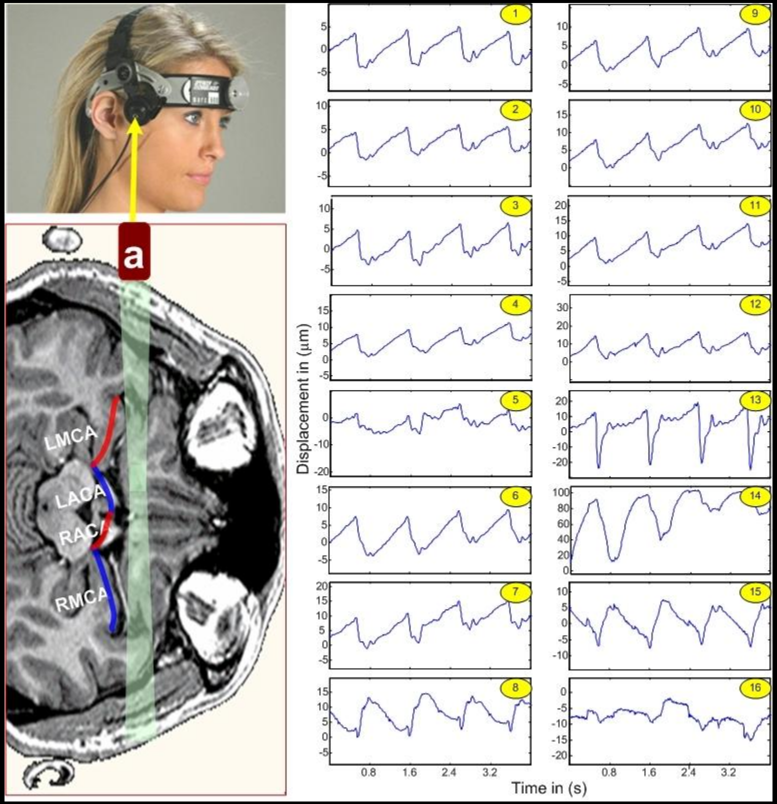
Case Study: Leibniz Institute

- High Throughput Multi-Dimensional Scaling (HiT-MDS)
- High dimensional data reconstructed for visualization
- Goal to understand data
- Speedup: 35X



Case Study: Spencer Technologies

- Real-time signal processing
- 64 ultrasound sensors
- Precise brain blood flow
- Speedup: 12X



Discussion

jacket

Faster MATLAB® through GPU computing