Core OS

#WWDC14

What's New in the Accelerate Framework

Session 703

Geoff Belter
Engineer, Vector and Numerics Group

What is Available?

Image processing (vImage)

Digital signal processing (vDSP)

Math functions (vForce, vMathLib, vBigNum)

Linear algebra (LAPACK, BLAS)

What is the Accelerate Framework?

High performance

- Fast
- Energy efficient

OS X and iOS

All generations of hardware

Session Goals

New features in vImage

Introduce

- LinearAlgebra
- <simd/simd.h>

vlmage

High performance image processing

vImage

Some things you can do

vImage

Some things you can do

Getting Data into vImage

CGImageRef

Getting Data into vImage

CGImageRef

```
// CGImageRef --> vImage_Buffer
vImage_Buffer buf;
vImage_CGImageFormat fmt = { .bitsPerComponent = 8, .bitsPerPixel = 32, … };
vImage_Error err = vImageBuffer_initWithCGImage( &buf, &fmt, NULL,
                                                 cgImage, kvImageNoFlags );
```

Getting Data into vImage

CGImageRef

```
// CGImageRef --> vImage_Buffer
vImage_Buffer buf;
vImage_CGImageFormat fmt = { .bitsPerComponent = 8, .bitsPerPixel = 32, … };
vImage_Error err = vImageBuffer_initWithCGImage( &buf, &fmt, NULL,
                                                 cgImage, kvImageNoFlags );
```

```
// vImage_Buffer --> CGImageRef
cgImage = vImageCreateCGImageFromBuffer( &buf, &fmt, NULL, NULL,
                                         kvImageNoFlags, &err );
```

Conversion Support

vImageConvert_AnyToAny

```
// 1) Make converter: srcFormat --> destFormat
vImage_CGImageFormat srcFormat = { .bitsPerComponent = 8, … };
vImage_CGImageFormat destFormat = { .bitsPerComponent = 16, … };

vImageConverterRef c = vImageConverter_CreateWithCGImageFormat(
    &srcFormat, &destFormat, NULL, kvImageNoFlags, &err);

// 2) Convert
vImage_Buffer srcBuf = {…}, destBuf = {…};
err = vImageConvert_AnyToAny(c,  &srcBuf, &destBuf, NULL, flags);
```

Conversion Support

vImageConvert_AnyToAny

```
// 1) Make converter: srcFormat --> destFormat
vImage_CGImageFormat srcFormat = { .bitsPerComponent = 8, … };
vImage_CGImageFormat destFormat = { .bitsPerComponent = 16, … };

vImageConverterRef c = vImageConverter_CreateWithCGImageFormat(
    &srcFormat, &destFormat, NULL, kvImageNoFlags, &err);

// 2) Convert
vImage_Buffer srcBuf = {…}, destBuf = {…};
err = vImageConvert_AnyToAny(c,  &srcBuf, &destBuf, NULL, flags);
```

Conversion Support

vImageConvert_AnyToAny

```
// 1) Make converter: srcFormat --> destFormat
vImage_CGImageFormat srcFormat = { .bitsPerComponent = 8, … };
vImage_CGImageFormat destFormat = { .bitsPerComponent = 16, … };

vImageConverterRef c = vImageConverter_CreateWithCGImageFormat(
    &srcFormat, &destFormat, NULL, kvImageNoFlags, &err);

// 2) Convert
vImage_Buffer srcBuf = {…}, destBuf = {…};
err = vImageConvert_AnyToAny(c,  &srcBuf, &destBuf, NULL, flags);
```

Conversion Support

vImageConvert_AnyToAny

```
// 1) Make converter: srcFormat --> destFormat
vImage_CGImageFormat srcFormat = { .bitsPerComponent = 8, … };
vImage_CGImageFormat destFormat = { .bitsPerComponent = 16, … };

vImageConverterRef c = vImageConverter_CreateWithCGImageFormat(
    &srcFormat, &destFormat, NULL, kvImageNoFlags, &err);

// 2) Convert
vImage_Buffer srcBuf = {…}, destBuf = {…};
err = vImageConvert_AnyToAny(c,  &srcBuf, &destBuf, NULL, flags);
```

What You Had to Say

“functions that convert
vImage_Buffer objects to
CGImageRef objects and back
👍👍👍👍👍👍”

Twitter user

What You Had to Say

"functions that convert vImage_Buffer objects to CGImageRef objects and back 👍👍👍👍👍👍"

Twitter user

"vImageConvert_AnyToAny is magical. Threaded and vectorized conversion between nearly any two pixel formats."

Twitter user

Video—RGB, Grayscale, and Y'CbCr

CVPixelBufferRef (a video frame)

Video—RGB, Grayscale, and Y'CbCr

CVPixelBufferRef (a video frame)


```
// CVPixelBufferRef --> vImageBuffer
vImageBuffer_InitWithCVPixelBuffer( &buf, &desiredFormat, cvPixelBuffer,
                                    NULL, NULL, kvImageNoFlags );
```

Video—RGB, Grayscale, and Y'CbCr

CVPixelBufferRef (a video frame)


```
// CVPixelBufferRef --> vImageBuffer
vImageBuffer_InitWithCVPixelBuffer( &buf, &desiredFormat, cvPixelBuffer,
                                    NULL, NULL, kvImageNoFlags );

// vImageBuffer --> CVPixelBufferRef
vImageBuffer_CopyToCVPixelBuffer( &buf, &bufFormat, cvPixelBuffer,
                                  NULL, NULL, kvImageNoFlags );
```

Getting video into vImage

NEW

Lower level interfaces

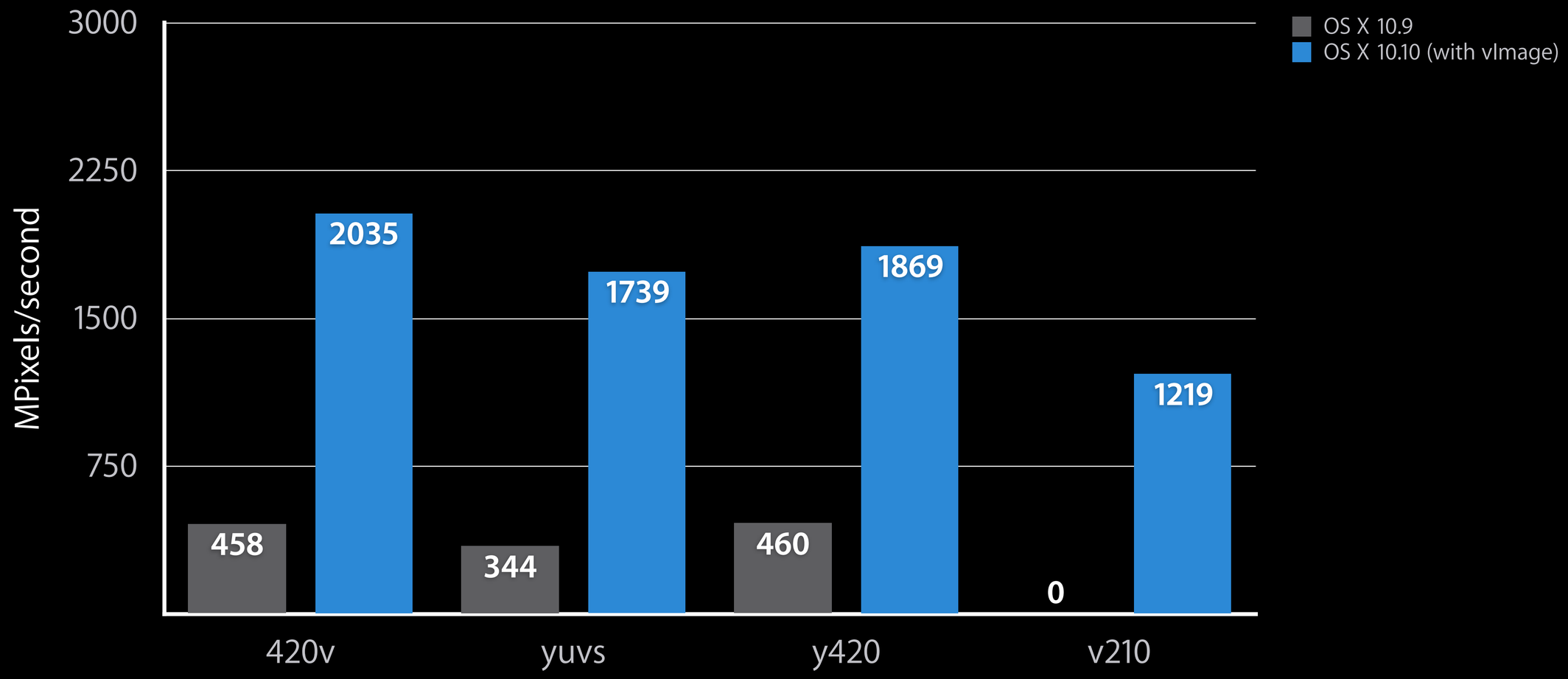
- 41 video conversions
- Manage chroma siting, transfer function, conversion matrix, etc.
- RGB colorspaces for video formats

vImageConvert_AnyToAny() for video

```
- vImageConverter_CreateForCGtoCVImageFormat
- vImageConverter_CreateForCVtoCGImageFormat
```

VideoToolbox Performance

Higher is better



LinearAlgebra (LA)

Simple to use high performance

Solving System of Linear Equations

With LAPACK

Given the system A, and the right-hand-side B, how do you find X (AX = B)?

Solving System of Linear Equations

With LAPACK

Given the system A, and the right-hand-side B, how do you find X (AX = B)?

```
__CLPK_integer n = matrix_size;
__CLPK_integer nrhs = number_right_hand_sides;
__CLPK_integer lda = column_stride_A;
__CLPK_integer ldb = column_stride_B;
__CLPK_integer *ipiv = malloc(sizeof(__CLPK_integer)*n);
__CLPK_integer info;
sgesv_(&n, &nrhs, A, &lda, ipiv, B, &ldb, &info);
free(ipiv);
```

Solving System of Linear Equations

With LA

Given the system A, and the right-hand-side B, how do you find X ($AX = B$)?

Solving System of Linear Equations

With LA

Given the system A, and the right-hand-side B, how do you find X (AX = B)?

```
la_object_t X = la_solve(A,B);
```

LinearAlgebra

New in iOS 8.0 and OS X Yosemite

LinearAlgebra

New in iOS 8.0 and OS X Yosemite

Simple with good performance

LinearAlgebra

New in iOS 8.0 and OS X Yosemite

Simple with good performance

Single and double precision

LinearAlgebra

New in iOS 8.0 and OS X Yosemite

Simple with good performance

Single and double precision

Native Objective-C Object

What's Available?

Element-wise arithmetic

Matrix product

Transpose

Norms / normalization

Linear systems

Slice

Splat

LA Objects

LA Objects

Reference counted opaque objects

- Objective-C objects when appropriate

LA Objects

Reference counted opaque objects

- Objective-C objects when appropriate

Managed for you

- Data buffer/memory
- Dimension details
- Errors and warnings
- Scalar data type (float/double)

Memory Management

Memory Management

C

---

la_release()
la_retain()

```
la_object_t A, B, C;
// create A and B
C = la_sum(A,B);
la_release(A);
la_release(B);
// use C
la_release(C);
```

Memory Management

C	Objective-C no ARC
la_release() la_retain()	-[release] -[retain]

```
la_object_t A, B, C;
// create A and B
C = la_sum(A,B);
la_release(A);
la_release(B);
// use C
la_release(C);
```

```
la_object_t A, B, C;
// create A and B
C = la_sum(A,B);
[A release];
[B release];
// use C
[C release];
```

Memory Management

C	Objective-C no ARC	Objective-C with ARC
la_release() la_retain()	-[release] -[retain]	

```
la_object_t A, B, C;
// create A and B
C = la_sum(A,B);
la_release(A);
la_release(B);
// use C
la_release(C);
```

```
la_object_t A, B, C;
// create A and B
C = la_sum(A,B);
[A release];
[B release];
// use C
[C release];
```

```
la_object_t A, B, C;
// create A and B
C = la_sum(A,B);

// use C
```

Buffer to LA Object

With copy

```
double *A = malloc(sizeof(double) * num_rows * row_stride);
// Fill A as row major matrix

// Data copied into object, user still responsible for A
la_object_t Aobj = la_matrix_from_double_buffer(A, num_rows, num_cols,
                                                row_stride, LA_NO_HINT,
                                                LA_DEFAULT_ATTRIBUTES);

// User retains all rights to A.  User must clean up A
free(A);
```

Buffer to LA Object

With copy

```
double *A = malloc(sizeof(double) * num_rows * row_stride);
// Fill A as row major matrix

// Data copied into object, user still responsible for A
la_object_t Aobj = la_matrix_from_double_buffer(A, num_rows, num_cols,
                                                row_stride, LA_NO_HINT,
                                                LA_DEFAULT_ATTRIBUTES);

// User retains all rights to A.  User must clean up A
free(A);
```

Buffer to LA Object

With copy

```
double *A = malloc(sizeof(double) * num_rows * row_stride);
// Fill A as row major matrix

// Data copied into object, user still responsible for A
la_object_t Aobj = la_matrix_from_double_buffer(A, num_rows, num_cols,
                                                row_stride, LA_NO_HINT,
                                                LA_DEFAULT_ATTRIBUTES);

// User retains all rights to A.  User must clean up A
free(A);
```

Buffer to LA Object

With copy

```
double *A = malloc(sizeof(double) * num_rows * row_stride);
// Fill A as row major matrix

// Data copied into object, user still responsible for A
la_object_t Aobj = la_matrix_from_double_buffer(A, num_rows, num_cols,
                                                row_stride, LA_NO_HINT,
                                                LA_DEFAULT_ATTRIBUTES);

// User retains all rights to A.  User must clean up A
free(A);
```

Hints

```
la_object_t o = la_matrix_from_double_buffer(A, num_rows, num_cols,
                                             row_stride, LA_SHAPE_DIAGONAL,
                                             LA_DEFAULT_ATTRIBUTES);
```

Hints

```
la_object_t o = la_matrix_from_double_buffer(A, num_rows, num_cols,
                                             row_stride, LA_SHAPE_DIAGONAL,
                                             LA_DEFAULT_ATTRIBUTES);
```

Allow for better performance

Hints

```
la_object_t o = la_matrix_from_double_buffer(A, num_rows, num_cols,
                                             row_stride, LA_SHAPE_DIAGONAL,
                                             LA_DEFAULT_ATTRIBUTES);
```

Allow for better performance

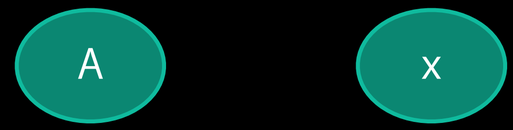
Insight about the data buffer

- Diagonal
- Triangular
- Symmetric
- Positive Definite

Lazy Evaluation

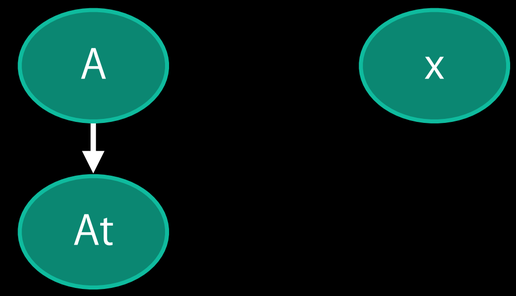
```
la_object_t foo(la_object_t A, la_object_t x) {
    // At = A’
    la_object_t At = la_transpose(A);

    // sum odd elements of x to even elements of x
    la_object_t x2 = la_sum(la_vector_slice(x,0,2,la_vector_length(x)/2),
                            la_vector_slice(x,1,2,la_vector_length(x)/2));

    // Atx2 = A’ * x2 * 3.2
    la_object_t Atx2 = la_scale_with_float(la_matrix_product(At,x2), 3.2f);
    if (la_status(Atx2) < 0) { // error }
    return Atx2
}
```

Lazy Evaluation



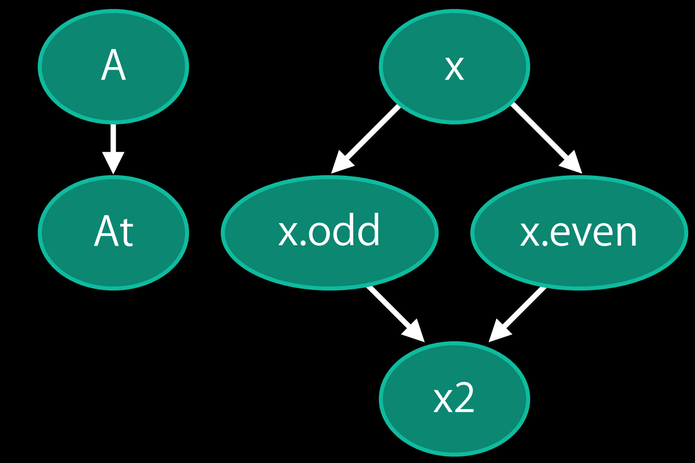

```
la_object_t foo(la_object_t A, la_object_t x) {
    // At = A’
    la_object_t At = la_transpose(A);
    // sum odd elements of x to even elements of x
    la_object_t x2 = la_sum(la_vector_slice(x,0,2,la_vector_length(x)/2),
                            la_vector_slice(x,1,2,la_vector_length(x)/2));

    // Atx2 = A’ * x2 * 3.2
    la_object_t Atx2 = la_scale_with_float(la_matrix_product(At,x2), 3.2f);
    if (la_status(Atx2) < 0) { // error }
    return Atx2
}
```

Lazy Evaluation



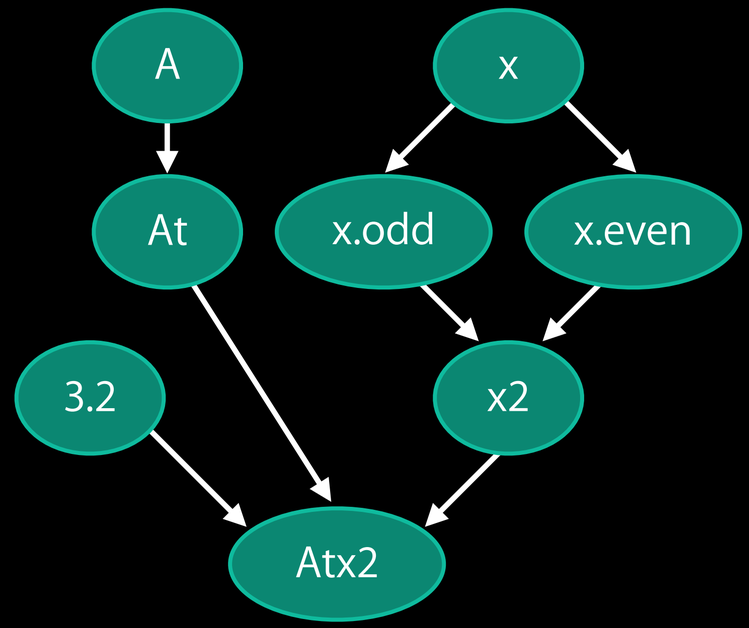

```
la_object_t foo(la_object_t A, la_object_t x) {
    // At = A’
    la_object_t At = la_transpose(A);
    // sum odd elements of x to even elements of x
    la_object_t x2 = la_sum(la_vector_slice(x,0,2,la_vector_length(x)/2),
                            la_vector_slice(x,1,2,la_vector_length(x)/2));

    // Atx2 = A’ * x2 * 3.2
    la_object_t Atx2 = la_scale_with_float(la_matrix_product(At,x2), 3.2f);
    if (la_status(Atx2) < 0) { // error }
    return Atx2
}
```

Lazy Evaluation




```
la_object_t foo(la_object_t A, la_object_t x) {
    // At = A'
    la_object_t At = la_transpose(A);

    // sum odd elements of x to even elements of x
    la_object_t x2 = la_sum(la_vector_slice(x,0,2,la_vector_length(x)/2),
                            la_vector_slice(x,1,2,la_vector_length(x)/2));

    // Atx2 = A' * x2 * 3.2
    la_object_t Atx2 = la_scale_with_float(la_matrix_product(At,x2), 3.2f);
    if (la_status(Atx2) < 0) { // error }
    return Atx2
}
```

Lazy Evaluation




```
la_object_t foo(la_object_t A, la_object_t x) {
    // At = A'
    la_object_t At = la_transpose(A);

    // sum odd elements of x to even elements of x
    la_object_t x2 = la_sum(la_vector_slice(x,0,2,la_vector_length(x)/2),
                            la_vector_slice(x,1,2,la_vector_length(x)/2));

    // Atx2 = A' * x2 * 3.2
    la_object_t Atx2 = la_scale_with_float(la_matrix_product(At,x2), 3.2f);
    if (la_status(Atx2) < 0) { // error }
    return Atx2
}
```

Lazy Evaluation

Details

No computation

No data buffer allocation

Triggered by

- la_matrix_to_float_buffer
- la_matrix_to_double_buffer
- la_vector_to_float_buffer
- la_vector_to_double_buffer

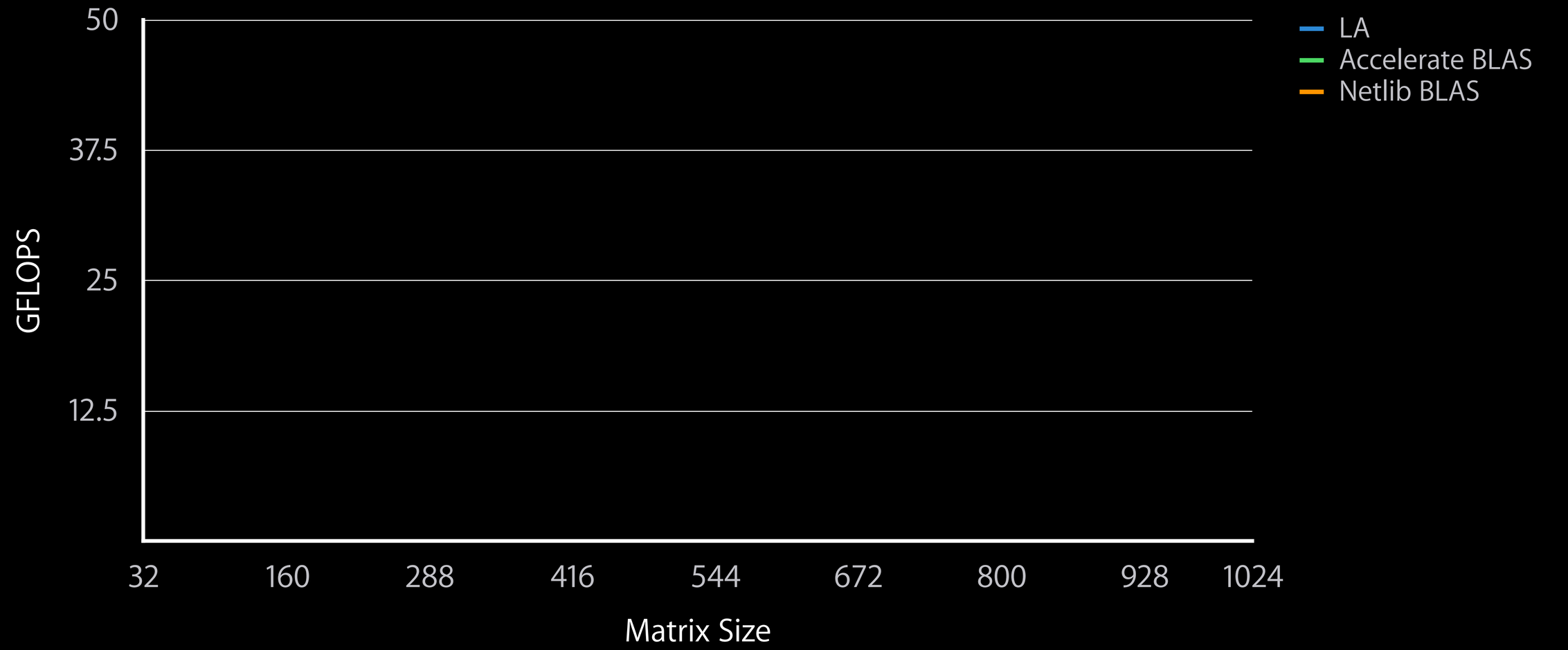
Performance Comparison

Netlib BLAS

- Open source

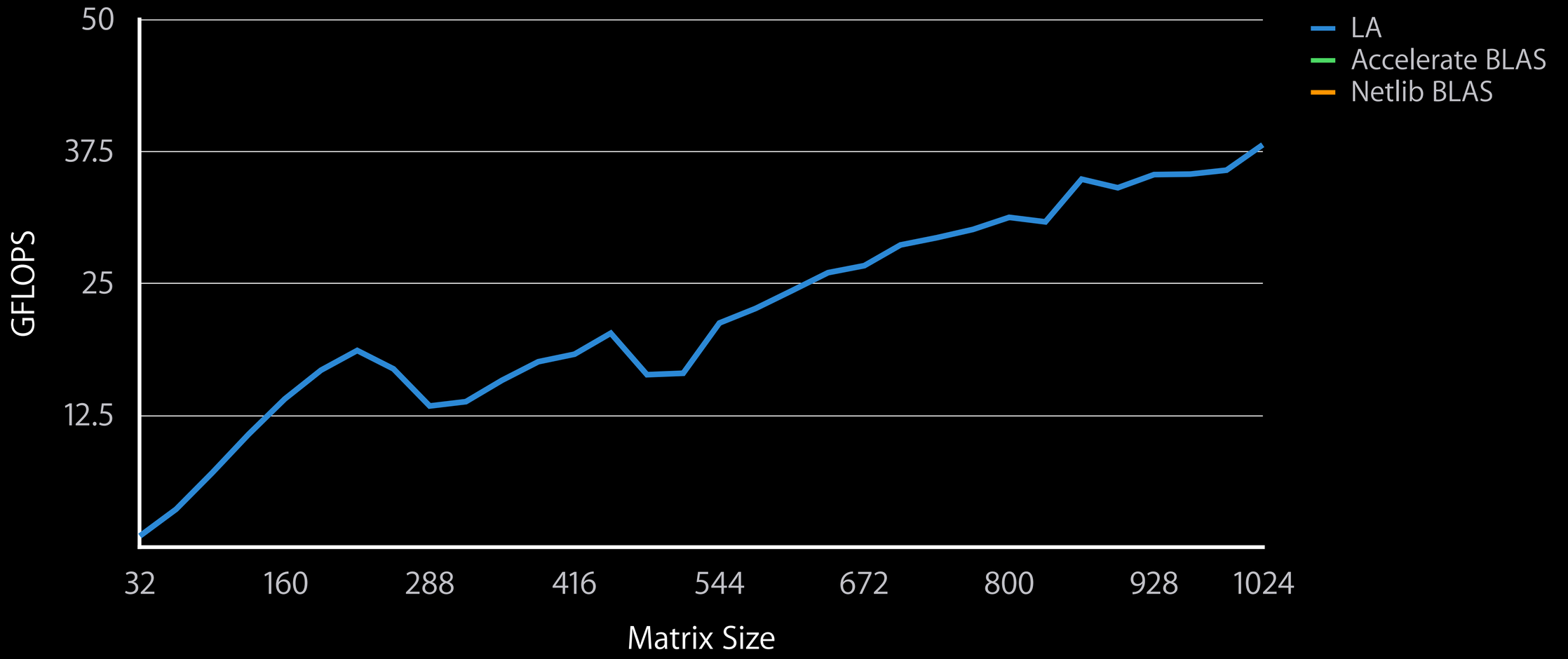
Performance Comparison

Higher is better



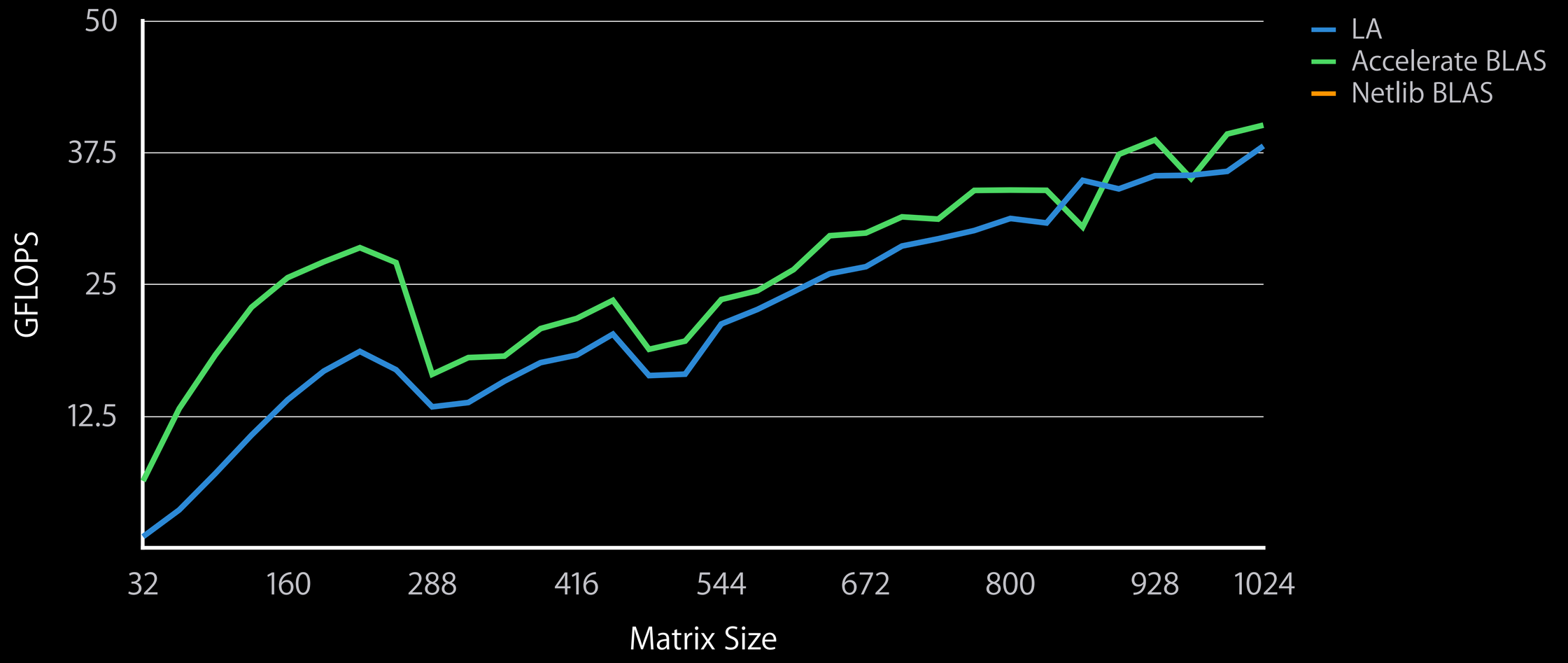
Performance Comparison

Higher is better



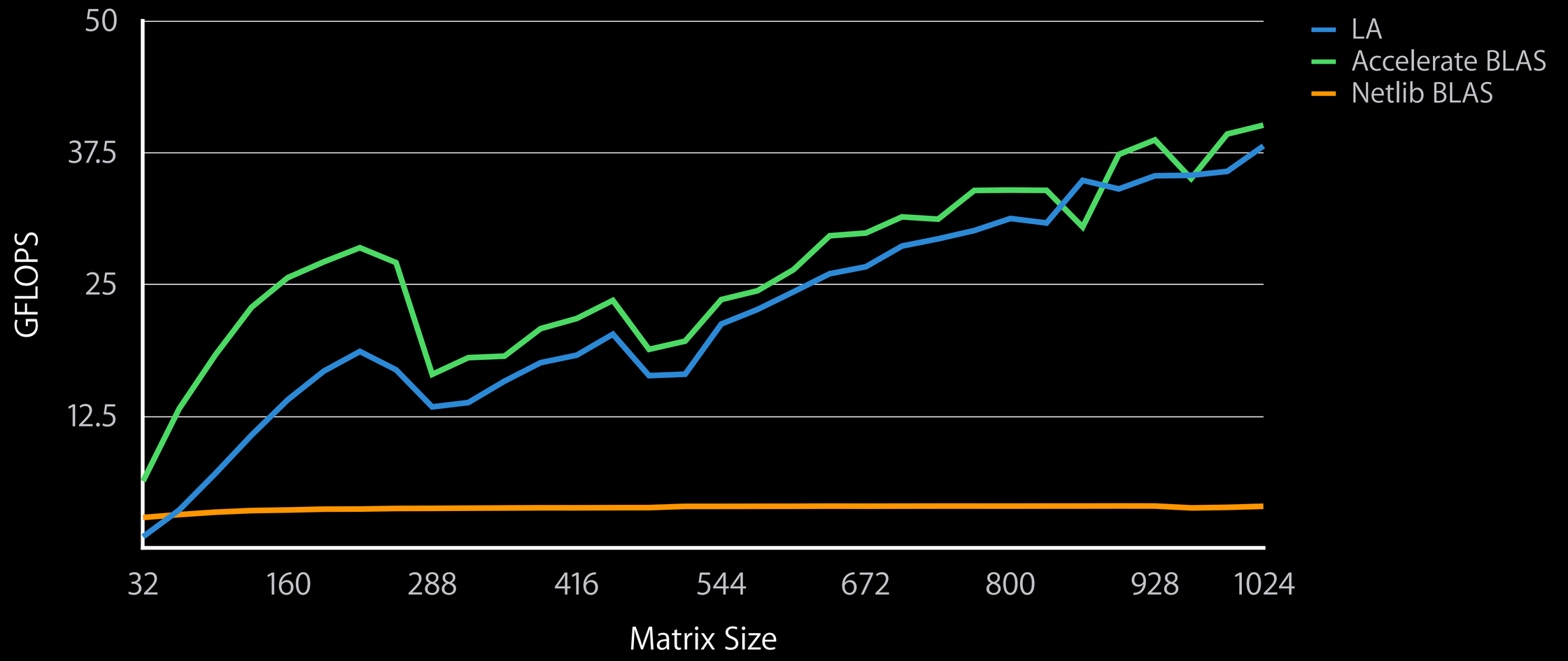
Performance Comparison

Higher is better



Performance Comparison

Higher is better



Error Handling

```
la_object_t AB = la_matrix_product( A, la_transpose(B) );
if (la_status(AB) < 0) { // handle error }

la_object_t result = la_sum( AB, la_scale_with_float( C, 3.2f ) );
if (la_status(result) < 0) { // handle error }
```

```
la_status_t status = la_matrix_to_float_buffer(buffer, leading_dim, result);
```

Error Handling

```
la_object_t AB = la_matrix_product( A, la_transpose(B) );
if (la_status(AB) < 0) { // handle error }

la_object_t result = la_sum( AB, la_scale_with_float( C, 3.2f ) );
if (la_status(result) < 0) { // handle error }

la_status_t status = la_matrix_to_float_buffer(buffer, leading_dim, result);
```

Error Handling

```
la_object_t AB = la_matrix_product( A, la_transpose(B) );

la_object_t result = la_sum( AB, la_scale_with_float( C, 3.2f ) );
```

```
la_status_t status = la_matrix_to_float_buffer(buffer, leading_dim, result);
if (status == LA_SUCCESS) {
    // No errors, buffer is filled with good data.
} else if (status > 0) {
    // No errors occurred, but result does not have full accuracy.
} else {
    // An error occurred.
    assert(0);
}
```

Debugging

Enable logging

- LA_ATTRIBUTE_ENABLE_LOGGING

Debugging

Enable logging

- LA_ATTRIBUTE_ENABLE_LOGGING

Error log

```
la_object_t la_sum(la_object_t, la_object_t):
   LA_DIMENSION_MISMATCH_ERROR: Encountered a dimension mismatch
   obj_left rows must be equal to obj_right rows; failed comparison: 8 == 9
```

Solve

```
la_object_t x = la_solve(A,b);
```

- If A is square and non-singular, compute the solution x to Ax = b
- If A is square and singular, produce an error

Slicing

What is slicing

Light weight access to partial object

- No buffer allocations
- No buffer copies

Slicing

What is slicing

Light weight access to partial object

- No buffer allocations
- No buffer copies

Three pieces of information

- Offset
- Stride
- Dimension

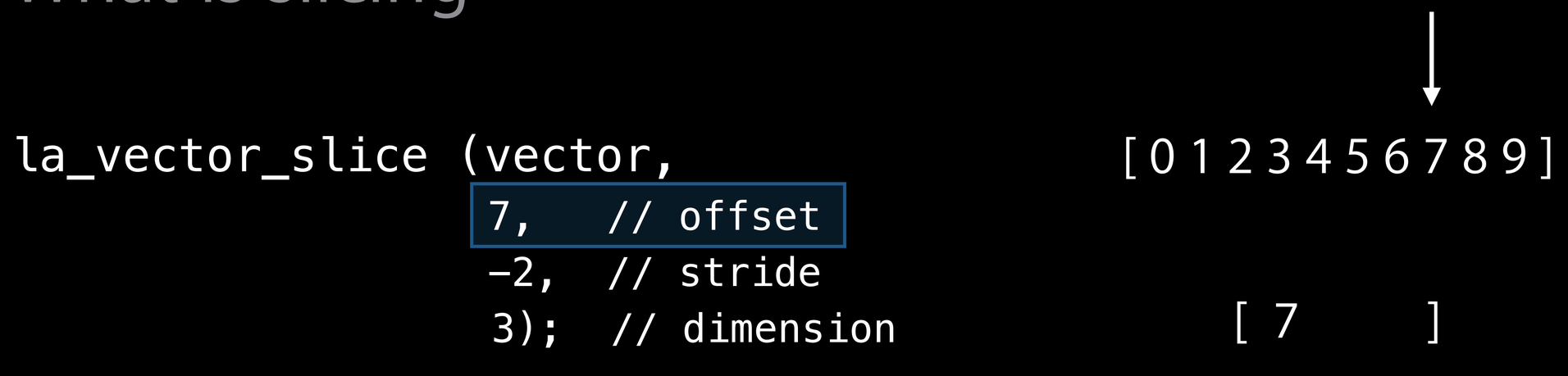
Slicing

What is slicing

```
la_vector_slice (vector,                    [0 1 2 3 4 5 6 7 8 9]
                 7,   // offset
                 -2,  // stride
                 3);  // dimension
```

Slicing

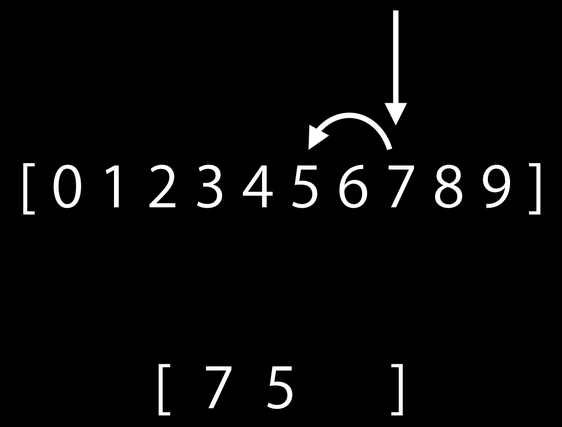
What is slicing



Slicing

What is slicing

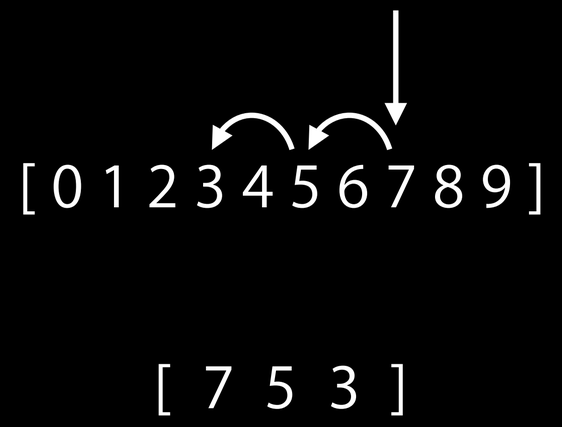
```
la_vector_slice (vector,
                 7,   // offset
                 -2,  // stride
                 3);  // dimension
```



Slicing

What is slicing

```
la_vector_slice (vector,
                 7,   // offset
                 -2,  // stride
                 3);  // dimension
```



Slice Example

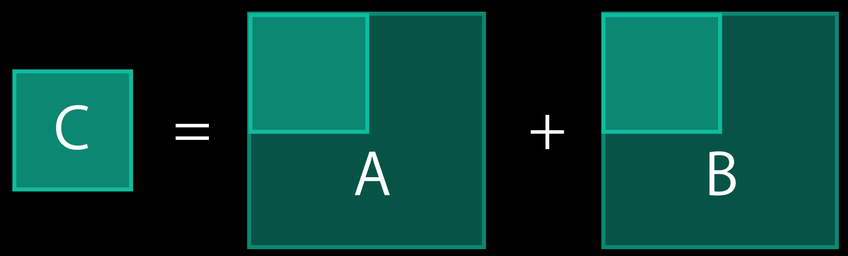
Tiling

```
la_object_t A,B,C;
// A and B are matrices of dimension MxN

for (int i = 0; i < 2; ++i) {
    for (int j = 0; j < 2; ++j) {
        la_object_t Atile = la_matrix_slice(A,i*M/2,j*N/2,1,1,M/2,N/2);
        la_object_t Btile = la_matrix_slice(B,i*M/2,j*N/2,1,1,M/2,N/2);
        C = la_sum(Atile, Btile);
        // use of C tile
    }
}
```

Slice Example

Tiling

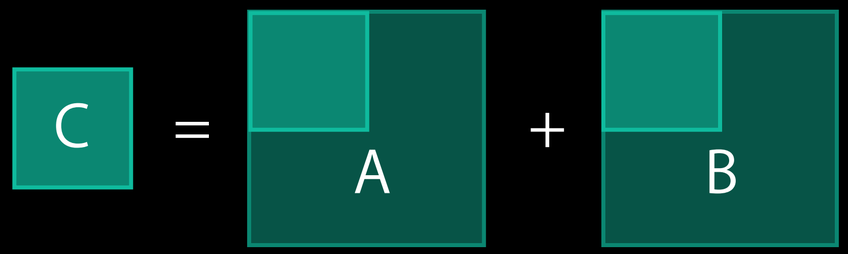
```
la_object_t A,B,C;
// A and B are matrices of dimension MxN

for (int i = 0; i < 2; ++i) {
    for (int j = 0; j < 2; ++j) {
        la_object_t Atile = la_matrix_slice(A,i*M/2,j*N/2,1,1,M/2,N/2);
        la_object_t Btile = la_matrix_slice(B,i*M/2,j*N/2,1,1,M/2,N/2);
        C = la_sum(Atile, Btile);
        // use of C tile
    }
}
```



Slice Example

Tiling

```
la_object_t A,B,C;
// A and B are matrices of dimension MxN

for (int i = 0; i < 2; ++i) {
    for (int j = 0; j < 2; ++j) {
        la_object_t Atile = la_matrix_slice(A,i*M/2,j*N/2,1,1,M/2,N/2);
        la_object_t Btile = la_matrix_slice(B,i*M/2,j*N/2,1,1,M/2,N/2);
        C = la_sum(Atile, Btile);
        // use of C tile
    }
}
```



Slice Example

Tiling

```
la_object_t A,B,C;
// A and B are matrices of dimension MxN

la_object_t sum = la_sum(A,B);

for (int i = 0; i < 2; ++i) {
    for (int j = 0; j < 2; ++j) {
        C = la_matrix_slice(sum,i*M/2,j*N/2,1,1,M/2,N/2);
        // use of C tile
    }
}
```

Slice Example

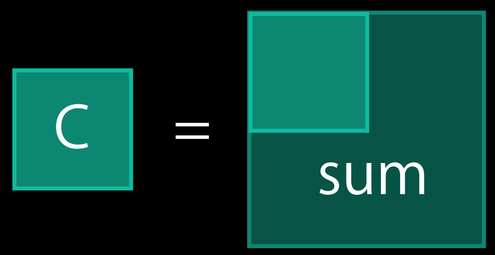
Tiling

```
la_object_t A,B,C;
// A and B are matrices of dimension MxN

la_object_t sum = la_sum(A,B);

for (int i = 0; i < 2; ++i) {
    for (int j = 0; j < 2; ++j) {
        C = la_matrix_slice(sum,i*M/2,j*N/2,1,1,M/2,N/2);
        // use of C tile
    }
}
```

Slice Example

Tiling

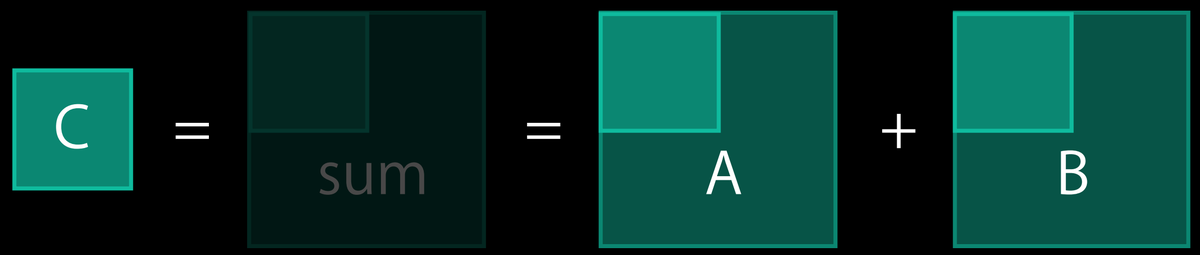
```
la_object_t A,B,C;
// A and B are matrices of dimension MxN

la_object_t sum = la_sum(A,B);

for (int i = 0; i < 2; ++i) {
    for (int j = 0; j < 2; ++j) {
        C = la_matrix_slice(sum,i*M/2,j*N/2,1,1,M/2,N/2);
        // use of C tile
    }
}
```



Slice Example

Tiling

```
la_object_t A,B,C;
// A and B are matrices of dimension MxN

la_object_t sum = la_sum(A,B);

for (int i = 0; i < 2; ++i) {
    for (int j = 0; j < 2; ++j) {
        C = la_matrix_slice(sum,i*M/2,j*N/2,1,1,M/2,N/2);
        // use of C tile
    }
}
```



Splat

What is splat

```
la_splat_from_float(5.0f);
```

Splat

What is splat

```
la_splat_from_float(5.0f);
```

Splat

What is splat

```
la_splat_from_float(5.0f);
```

Add 2 to every element of a vector

- la_sum(vector, la_splat_from_double(2.0));

LinearAlgebra

Summary

Simple API

Modern language and run-time features

Good performance

LINPACK

The joy of benchmarking

Stephen Canon
Engineer, Vector and Numerics Group

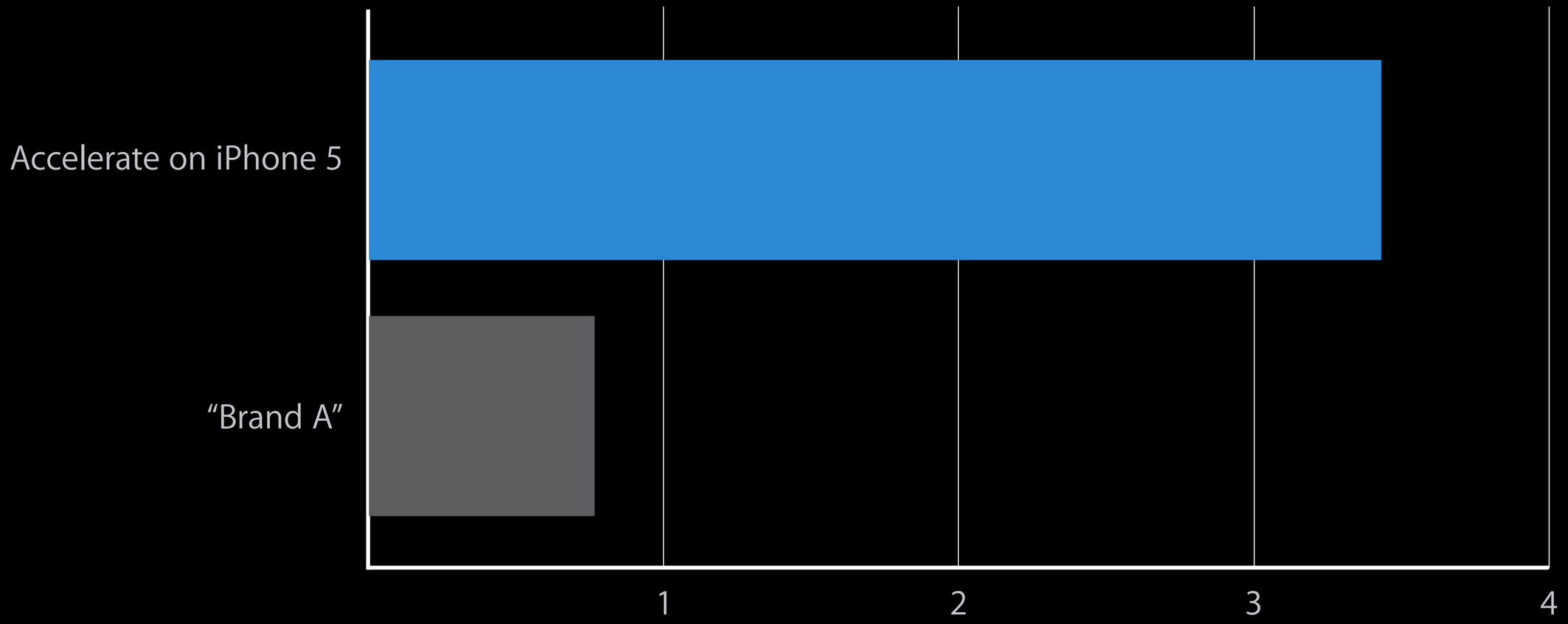
How fast can you solve a system of linear equations?

LINPACK tests both hardware and software

Accelerate vs. “Brand A”

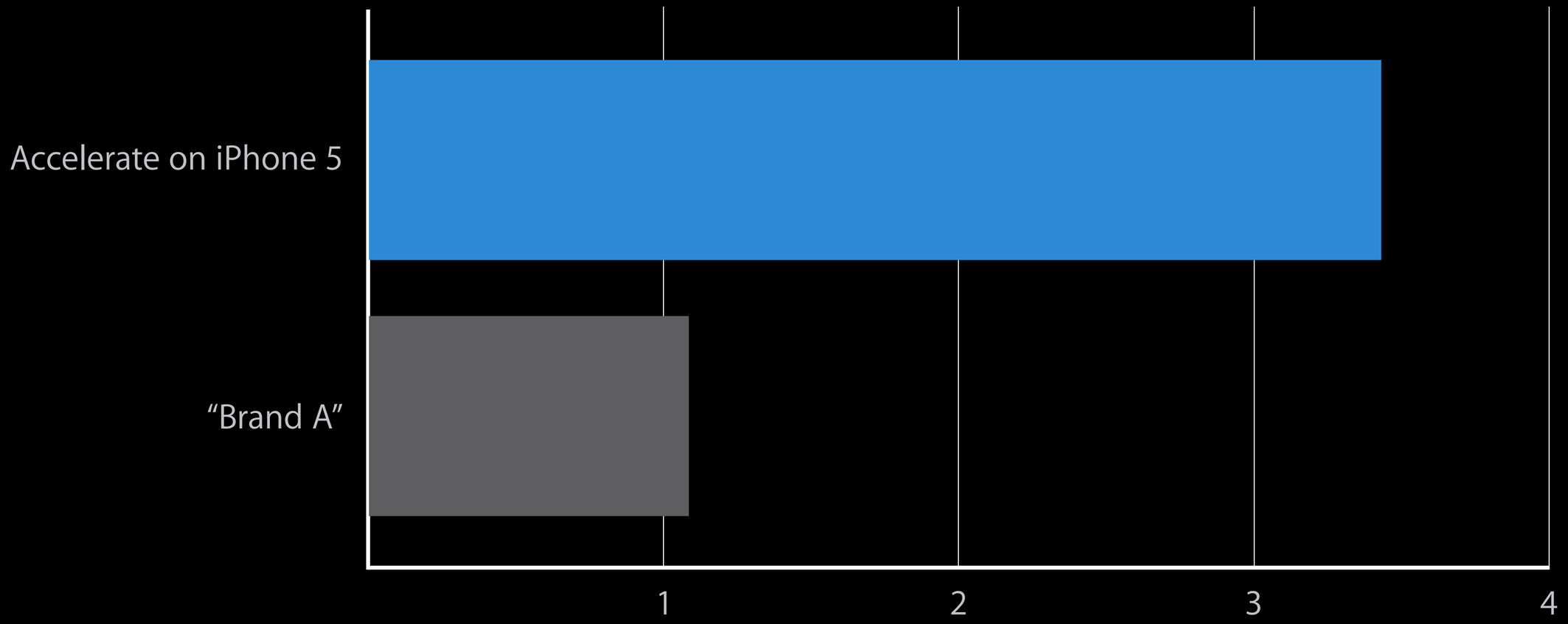
2013

LINPACK performance in GFLOPS (bigger is better)



2014

LINPACK performance in GFLOPS (bigger is better)



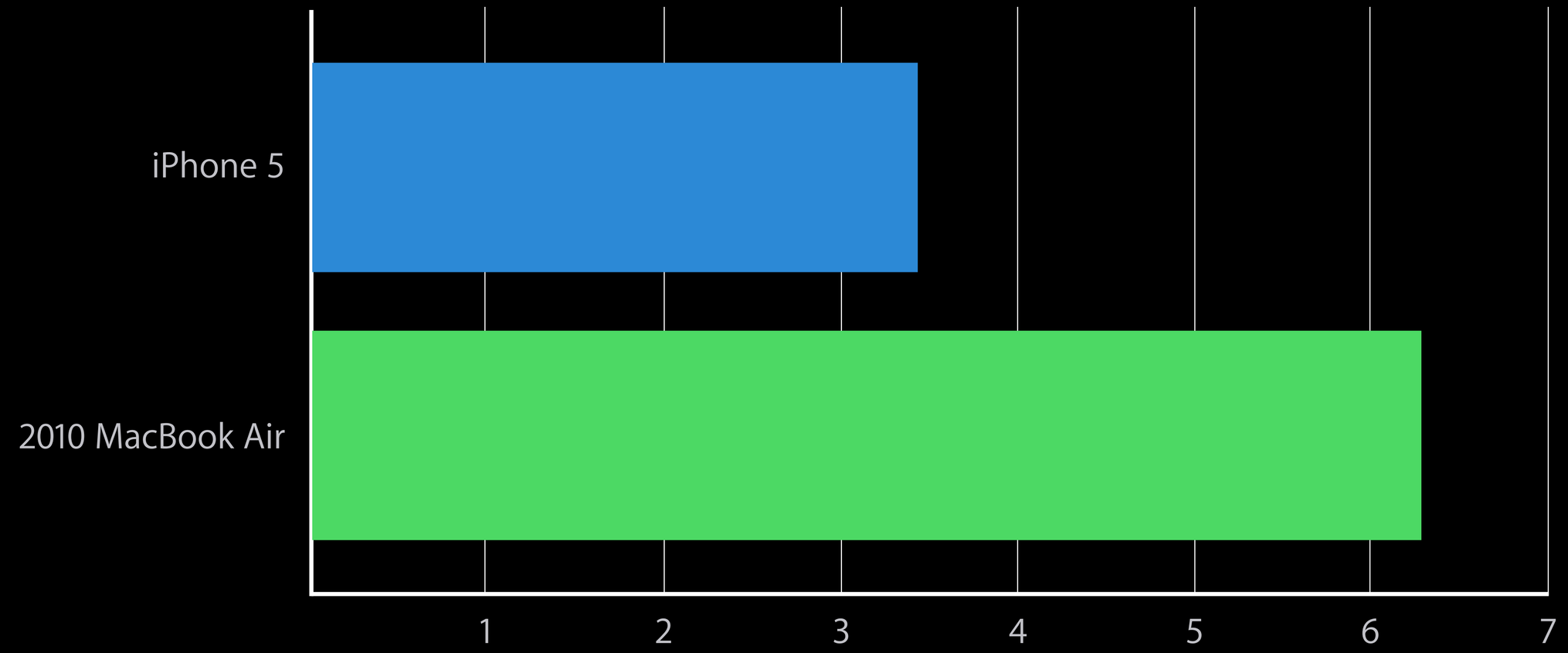
Let’s find some new competition

Let’s find some new competition

Ourselves

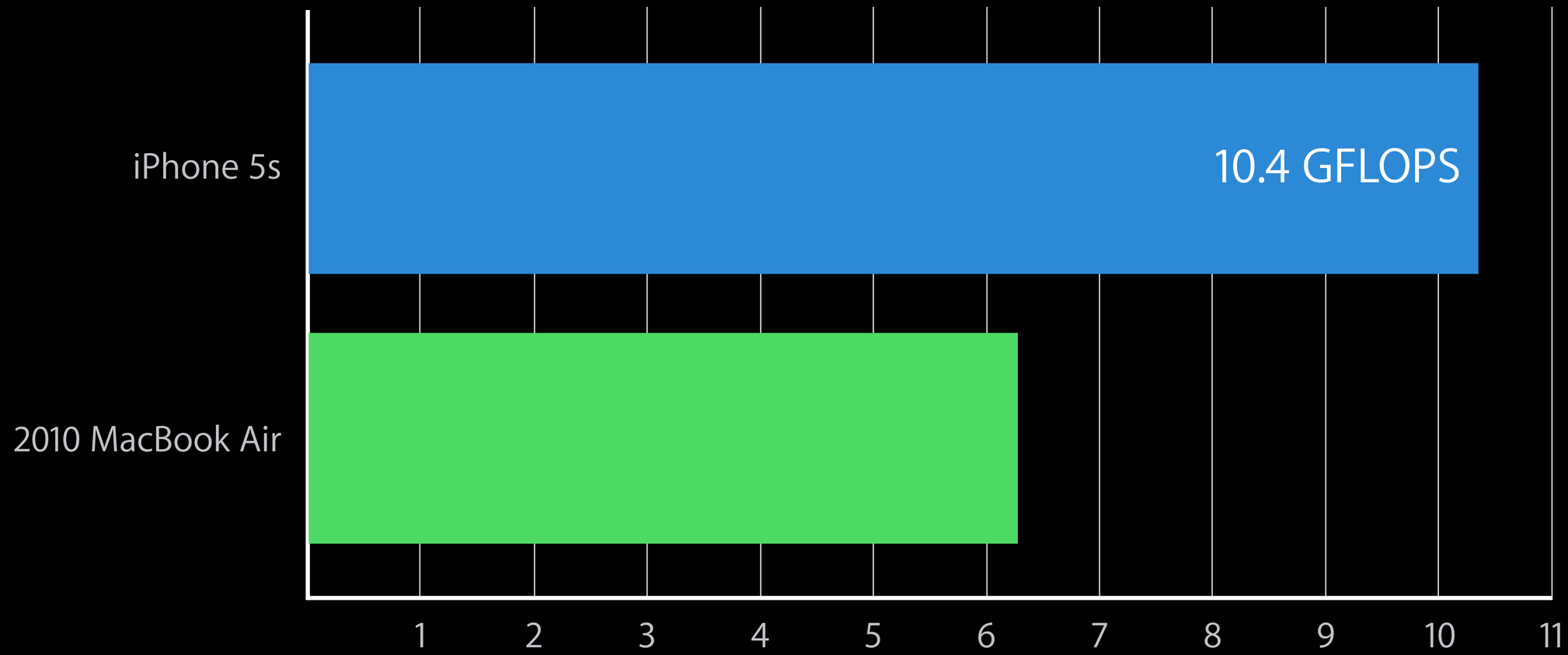
2014

LINPACK performance in GFLOPS (bigger is better)



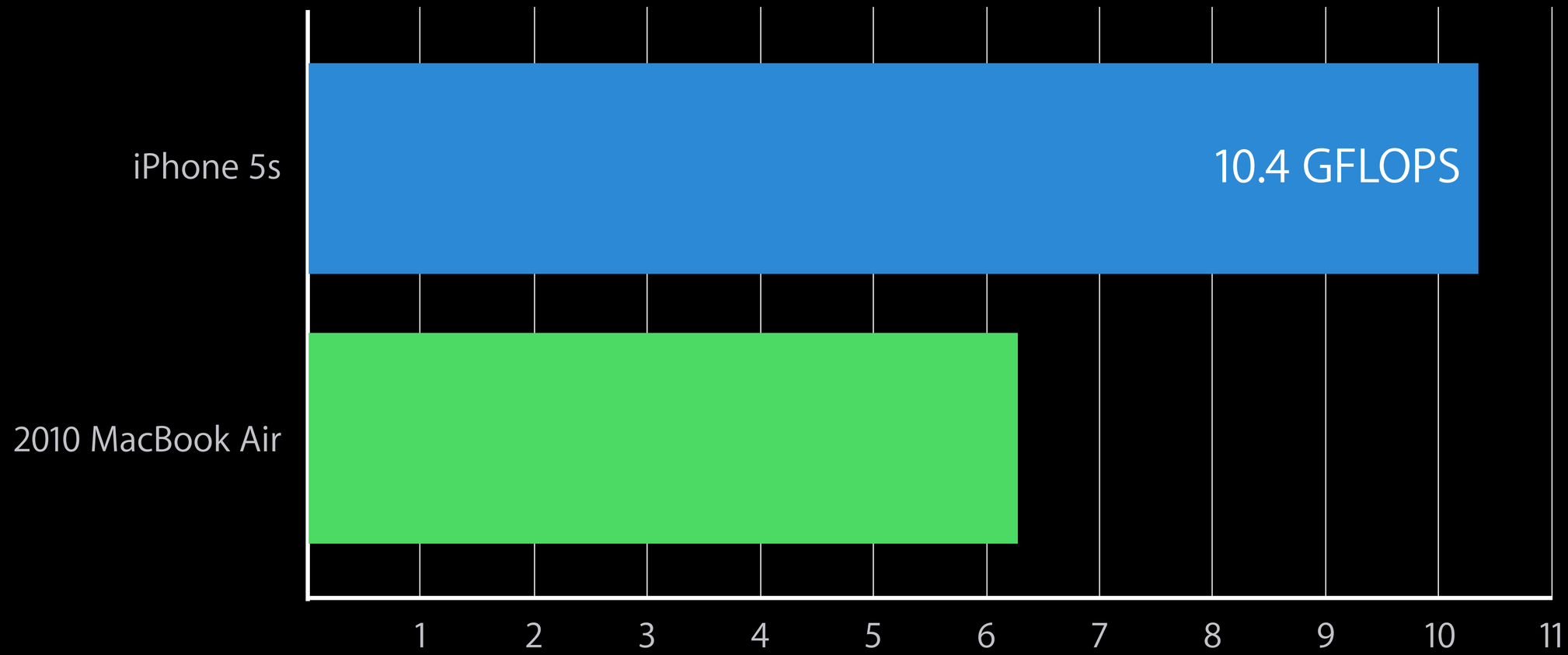
2014

LINPACK performance in GFLOPS (bigger is better)



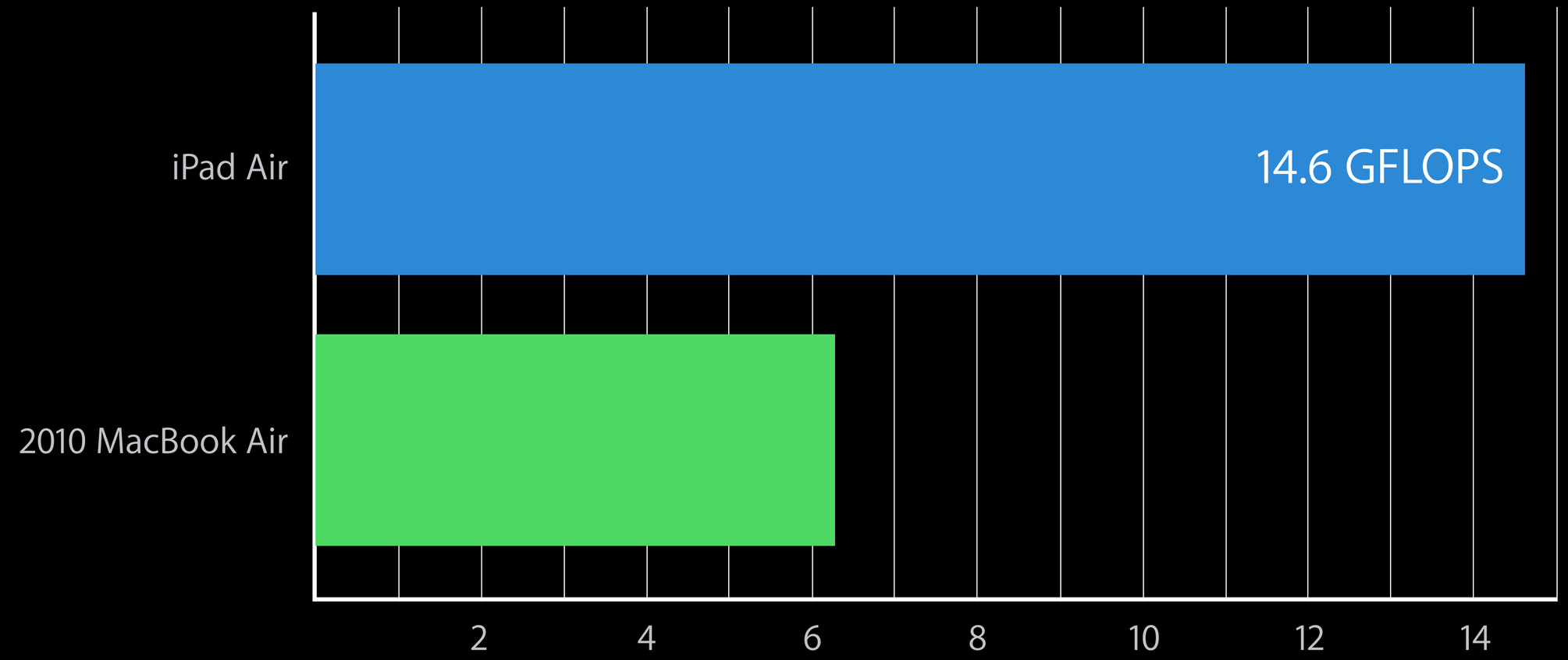
2014

LINPACK performance in GFLOPS (bigger is better)



2014

LINPACK performance in GFLOPS (bigger is better)



<simd/simd.h>

Short vector and matrix math

<simd/simd.h>

New (iOS 8 and OS X Yosemite) library with three purposes:

<simd/simd.h>

New (iOS 8 and OS X Yosemite) library with three purposes:

- 2D, 3D, and 4D vector math and geometry

<simd/simd.h>

New (iOS 8 and OS X Yosemite) library with three purposes:

- 2D, 3D, and 4D vector math and geometry
- Features of Metal in C, Objective-C, and C++ on the CPU

<simd/simd.h>

New (iOS 8 and OS X Yosemite) library with three purposes:

- 2D, 3D, and 4D vector math and geometry
- Features of Metal in C, Objective-C, and C++ on the CPU
- Abstraction over architecture-specific SIMD types and intrinsics

Vector Math and Geometry

Wish list

Vector Math and Geometry

Wish list

Inline implementations

Vector Math and Geometry

Wish list

Inline implementations

Concise functions without extra parameters

Vector Math and Geometry

Wish list

Inline implementations

Concise functions without extra parameters

```
float a = cblas_sdot(3, 1.0f, x, 1, y, 1);
```

Vector Math and Geometry

Wish list

Inline implementations

Concise functions without extra parameters

```
float a = GLKVector3DotProduct(x, y);
```

Vector Math and Geometry

Wish list

Inline implementations

Concise functions without extra parameters

```
float a = vector_dot(x, y);
```

Vector Math and Geometry

Wish list

Inline implementations

Concise functions without extra parameters

```
using namespace simd;
float a = dot(x, y);
```

Vector Math and Geometry

Wish list

Inline implementations

Concise functions without extra parameters

Vector Math and Geometry

Wish list

Inline implementations

Concise functions without extra parameters

Arithmetic should use operators

Vector Math and Geometry

Wish list

Inline implementations

Concise functions without extra parameters

Arithmetic should use operators

```
z = GLKVector4MultiplyScalar(GLKVector4Add(x,y),0.5);
```

Vector Math and Geometry

Wish list

Inline implementations

Concise functions without extra parameters

Arithmetic should use operators

```
z = 0.5*(x + y);
```

Vector Math and Geometry

Wish list

Inline implementations

Concise functions without extra parameters

Arithmetic should use operators

```
z = 0.5*(x + y);
```

Vector Math and Geometry

Types

In C and Objective-C, the primary type is vector_float*N*, where N is 2, 3, or 4

In C++ you can use simd::float*N*

Based on clang “extended vectors”

Vector Math and Geometry

Arithmetic

Your favorite arithmetic operators (+,–,∗,/) work with both vectors and scalars

Vector Math and Geometry

Arithmetic

Your favorite arithmetic operators (+,–,∗,/) work with both vectors and scalars

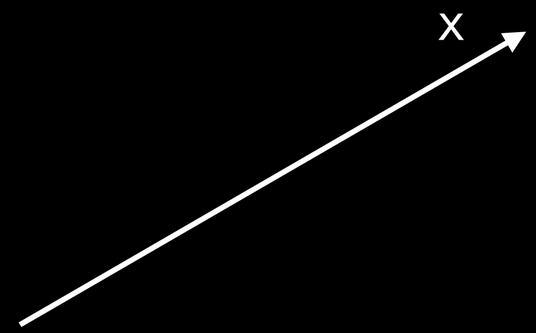
```
vector_float3 vector_reflect(vector_float3 x, vector_float3 n) {

}
```

Vector Math and Geometry

Arithmetic

Your favorite arithmetic operators (+,–,∗,/) work with both vectors and scalars

```
vector_float3 vector_reflect(vector_float3 x, vector_float3 n) {

}
```



Vector Math and Geometry

Arithmetic

Your favorite arithmetic operators (+,–,∗,/) work with both vectors and scalars

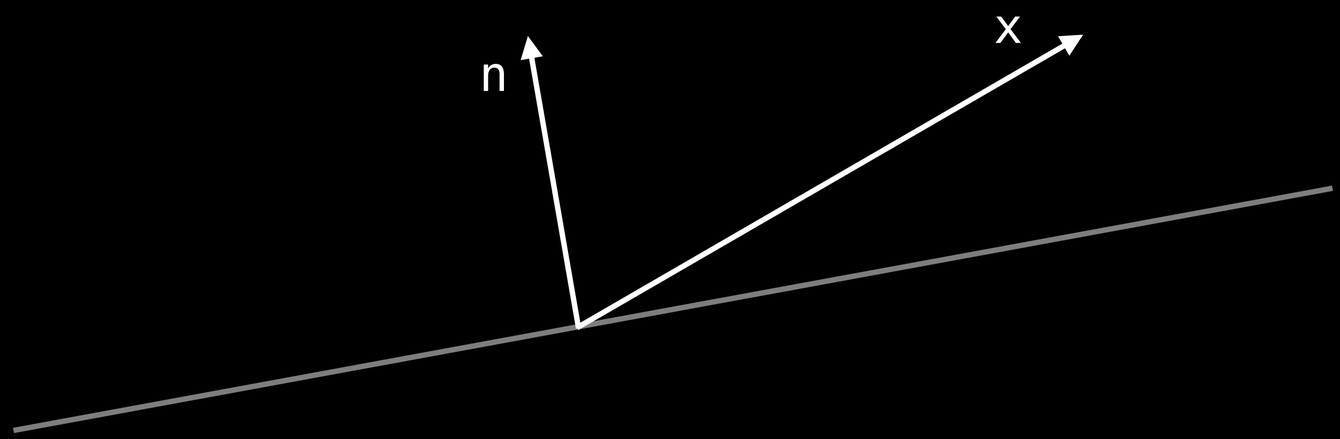
```
vector_float3 vector_reflect(vector_float3 x, vector_float3 n) {

}
```

Vector Math and Geometry

Arithmetic

Your favorite arithmetic operators (+,−,∗,/) work with both vectors and scalars

```
vector_float3 vector_reflect(vector_float3 x, vector_float3 n) {

}
```

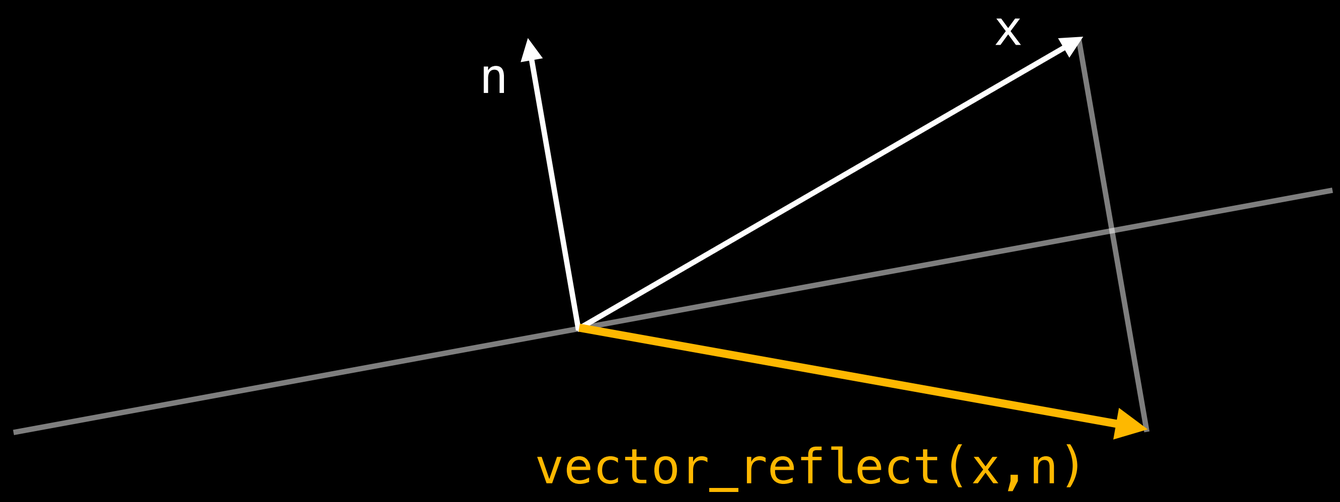


Vector Math and Geometry

Arithmetic

Your favorite arithmetic operators (+,–,∗,/) work with both vectors and scalars

```
vector_float3 vector_reflect(vector_float3 x, vector_float3 n) {

}
```

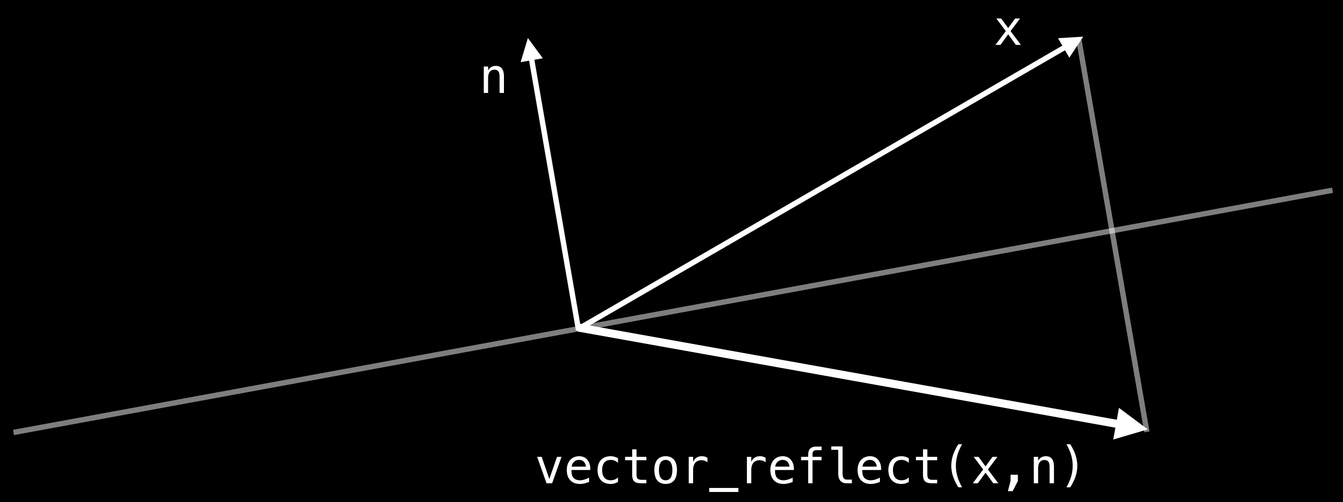


Vector Math and Geometry

Arithmetic

Your favorite arithmetic operators (+,–,∗,/) work with both vectors and scalars

```
vector_float3 vector_reflect(vector_float3 x, vector_float3 n) {
    return x - 2∗vector_dot(x,n)∗n;
}
```



Vector Math and Geometry

Elements and subvectors

Array subscripting

```
vector_float4 a = { 0, 1, 2, 3 };
float z = x[2];                 // z = 2
```

Vector Math and Geometry

Elements and subvectors

Named subvectors

```
vector_float4 a = { 0, 1, 2, 3 };
vector_float2 b = a.lo;      // b = { 0, 1 }
a.even = -b;                 // a = { 0, 1,-1, 3 }
```

Vector Math and Geometry

`<simd/math.h>`

`<simd/common.h>`

`<simd/geometry.h>`

Vector Math and Geometry

	C/Objective-C
<simd/math.h>	fabs(x) sqrt(x) floor(x) sin(x)…
<simd/common.h>	vector_clamp(x,min,max) vector_mix(x,y,t) vector_recip(x) vector_step(x,edge)…
<simd/geometry.h>	vector_dot(x,y) vector_length(x) vector_normalize(x) vector_reflect(x,n)…

Vector Math and Geometry

	C/Objective-C	C++ (and Metal)
<simd/math.h>	fabs(x) sqrt(x) floor(x) sin(x)…	fabs(x) sqrt(x) floor(x) sin(x)…
<simd/common.h>	vector_clamp(x,min,max) vector_mix(x,y,t) vector_recip(x) vector_step(x,edge)…	clamp(x,min,max) mix(x,y,t) recip(x) step(x,edge)…
<simd/geometry.h>	vector_dot(x,y) vector_length(x) vector_normalize(x) vector_reflect(x,n)…	dot(x,y) length(x) normalize(x) reflect(x,n)…

Vector Math and Geometry

Some functions have two flavors: “precise” and “fast”

Vector Math and Geometry

Some functions have two flavors: “precise” and “fast”

- “precise” is the default…

Vector Math and Geometry

Some functions have two flavors: “precise” and “fast”

- “precise” is the default…
- … but if you compile with `-ffast-math`, you get the “fast” versions

Vector Math and Geometry

Even with `-ffast-math`, you can call the precise functions by name:

```
float len = vector_precise_length(x);
```

Vector Math and Geometry

You can call the fast versions by name too:

```
x = fast::normalize(x);
```

Matrices

C and Objective-C

“`matrix_float`*`N`*`x`*`M`*”, where *N* and *M* are 2, 3, or 4

- *N* is number of *columns*, *M* is number of *rows*.

Matrices

C and Objective-C

Create matrices

```
matrix_from_diagonal(vector)
matrix_from_columns(vector, vector, …)
matrix_from_rows(vector, vector, …)
```

Arithmetic

```
matrix_scale(matrix, scalar)
matrix_linear_combination(matrix, scalar, matrix, scalar)
matrix_transpose(matrix)
matrix_invert(matrix)
matrix_multiply(matrix/vector, matrix/vector)
```

Matrices

C++ (and Metal)

Create matrices

```
float4x4()
float2x2(diagonal)
float3x4(column0, column1, …)
```

Arithmetic

```
float3x4 A, B;
A -= 2.f * B;
float4x3 C = transpose(A)
vector3 x;
vector4 y = A*x;
```

Abstract SIMD

Additional types

Doubles, signed and unsigned integers

Longer vectors (8, 16, and 32 elements)

Unaligned vectors

Abstract SIMD

Integer operators and conversions

Abstract SIMD

Integer operators and conversions

Arithmetic operators: +, −, ∗, /, %

Abstract SIMD

Integer operators and conversions

Arithmetic operators: +, −, ∗, /, %

Bitwise operators: >>, <<, &, |, ^, ~

Abstract SIMD

Integer operators and conversions

Arithmetic operators: +, −, ∗, /, %

Bitwise operators: >>, <<, &, |, ^, ~

Conversions:

```
vector_float x;
vector_ushort y = vector_ushort(x);
vector_char z = vector_char_sat(x);
```

Abstract SIMD

Comparisons

Vector comparisons: ==, !=, >, <, >=, <=

- Result is a vector of integers; each lane is –1 if comparison is true, 0 if false

Abstract SIMD

Comparisons

Vector comparisons: ==, !=, >, <, >=, <=

- Result is a vector of integers; each lane is –1 if comparison is true, 0 if false

Abstract SIMD

Comparisons

Vector comparisons: ==, !=, >, <, >=, <=

- Result is a vector of integers; each lane is –1 if comparison is true, 0 if false

x	0.0	1.0	2.0	3.0
y	0.0	3.14159	–infinity	42.0

Abstract SIMD

Comparisons

Vector comparisons: ==, !=, >, <, >=, <=

- Result is a vector of integers; each lane is –1 if comparison is true, 0 if false

x	0.0	1.0	2.0	3.0
y	0.0	3.14159	–infinity	42.0

x < y

Abstract SIMD

Comparisons

Vector comparisons: ==, !=, >, <, >=, <=

- Result is a vector of integers; each lane is –1 if comparison is true, 0 if false

x	0.0	1.0	2.0	3.0
y	0.0	3.14159	–infinity	42.0
x < y	0x00000000			

Abstract SIMD

Comparisons

Vector comparisons: ==, !=, >, <, >=, <=

- Result is a vector of integers; each lane is –1 if comparison is true, 0 if false

x	0.0	1.0	2.0	3.0
y	0.0	3.14159	–infinity	42.0
x < y	0x00000000	0xffffffff		

Abstract SIMD

Comparisons

Vector comparisons: ==, !=, >, <, >=, <=

- Result is a vector of integers; each lane is –1 if comparison is true, 0 if false

x	0.0	1.0	2.0	3.0
y	0.0	3.14159	–infinity	42.0
x < y	0x00000000	0xffffffff	0x00000000	

Abstract SIMD

Comparisons

Vector comparisons: ==, !=, >, <, >=, <=

- Result is a vector of integers; each lane is –1 if comparison is true, 0 if false

x	0.0	1.0	2.0	3.0
y	0.0	3.14159	–infinity	42.0
x < y	0x00000000	0xffffffff	0x00000000	0xffffffff

Abstract SIMD

Comparisons

Vector comparisons: ==, !=, >, <, >=, <=

- Result is a vector of integers; each lane is –1 if comparison is true, 0 if false
- Type of result usually isn't important, because you'll use one of the following:

```
if (vector_any(x < 0)) { /* executed if any lane of x is negative */ }
if (vector_all(y != 0)) { /* executed if every lane of y is non-zero */ }
z = vector_bitselect(x, y, x > y); /* minimum of x and y */
```

String Copy

Scalar implementation

String Copy

Scalar implementation

```
void string_copy(char *dst, const char *src) {
    while ((*dst++ = *src++));
}
```

String Copy

SSE intrinsic implementation

```
void vector_string_copy(char *dst, const char *src) {
    while ((uintptr_t)src % 16)
        if ((*dst++ = *src++) == 0) return;
    while (1) {
        __m128i data = _mm_load_si128((const __m128i *)src);
        __m128i contains_zero = _mm_cmpeq_epi8(data, _mm_set1_epi8(0));
        if (_mm_movemask_epi8(contains_zero))
            break;
        _mm_storeu_si128((__m128i *)dst, data);
        src += 16;
        dst += 16;
    }
    string_copy((char *)vec_dst, (const char *)vec_src);
}
```

String Copy

<simd/simd.h> implementation

```
void vector_string_copy(char *dst, const char *src) {
    while ((uintptr_t)src % 16)
        if ((*dst++ = *src++) == 0) return;
    const vector_char16 *vec_src = (const vector_char16 *)src;
    packed_char16 *vec_dst = (packed_char16 *)dst;
    while (!vector_any(*vec_src == 0))
        *vec_dst++ = *vec_src++;
    string_copy((char *)vec_dst, (const char *)vec_src);
}
```

String Copy

<simd/simd.h> implementation

```
void vector_string_copy(char *dst, const char *src) {
    while ((uintptr_t)src % 16)
        if ((*dst++ = *src++) == 0) return;
    const vector_char16 *vec_src = (const vector_char16 *)src;
    packed_char16 *vec_dst = (packed_char16 *)dst;
    while (!vector_any(*vec_src == 0))
        *vec_dst++ = *vec_src++;
    string_copy((char *)vec_dst, (const char *)vec_src);
}
```

String Copy

<simd/simd.h> implementation

```
void vector_string_copy(char *dst, const char *src) {
    while ((uintptr_t)src % 16)
        if ((*dst++ = *src++) == 0) return;
    const vector_char16 *vec_src = (const vector_char16 *)src;
    packed_char16 *vec_dst = (packed_char16 *)dst;
    while (!vector_any(*vec_src == 0))
        *vec_dst++ = *vec_src++;
    string_copy((char *)vec_dst, (const char *)vec_src);
}
```

String Copy

<simd/simd.h> implementation

```
void vector_string_copy(char *dst, const char *src) {
    while ((uintptr_t)src % 16)
        if ((*dst++ = *src++) == 0) return;
    const vector_char16 *vec_src = (const vector_char16 *)src;
    packed_char16 *vec_dst = (packed_char16 *)dst;
    while (!vector_any(*vec_src == 0))
        *vec_dst++ = *vec_src++;
    string_copy((char *)vec_dst, (const char *)vec_src);
}
```

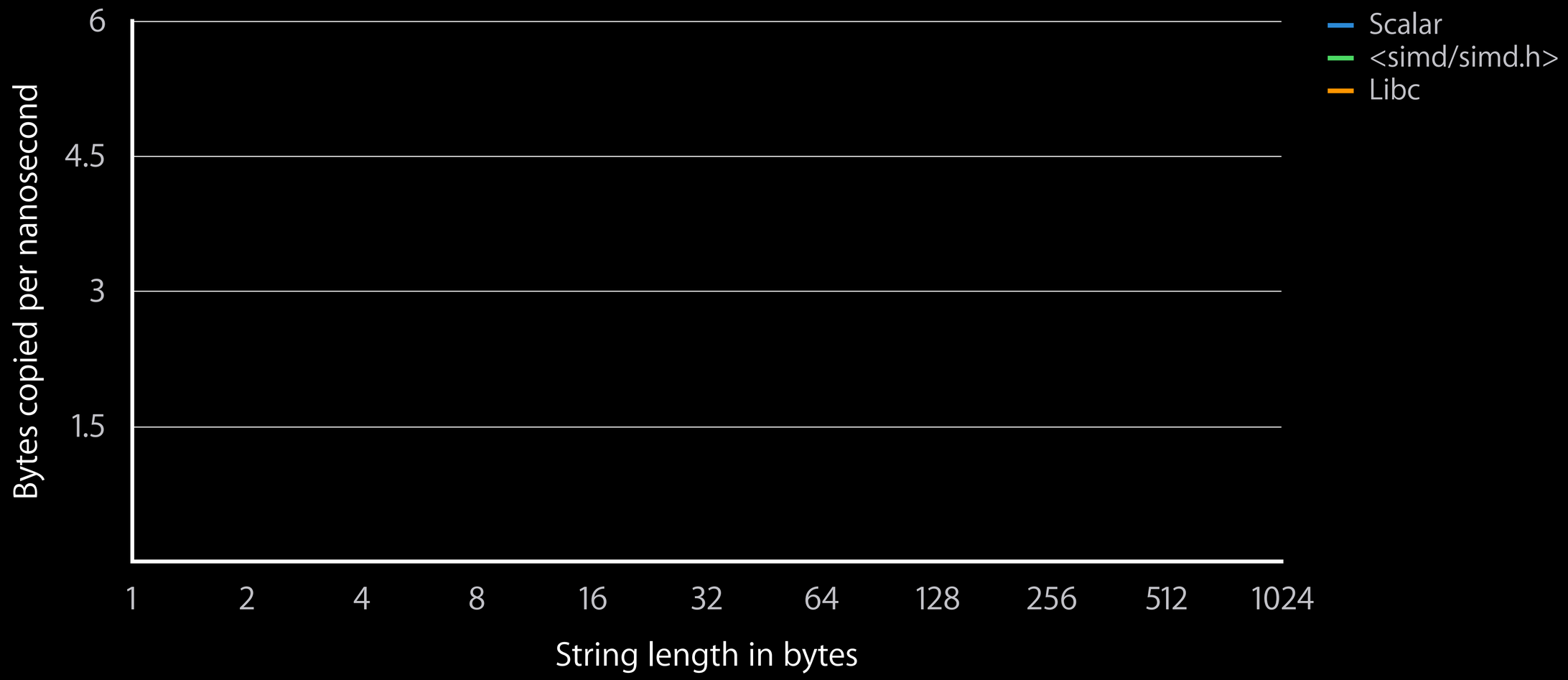
String Copy

<simd/simd.h> implementation

```
void vector_string_copy(char *dst, const char *src) {
    while ((uintptr_t)src % 16)
        if ((*dst++ = *src++) == 0) return;
    const vector_char16 *vec_src = (const vector_char16 *)src;
    packed_char16 *vec_dst = (packed_char16 *)dst;
    while (!vector_any(*vec_src == 0))
        *vec_dst++ = *vec_src++;
    string_copy((char *)vec_dst, (const char *)vec_src);
}
```

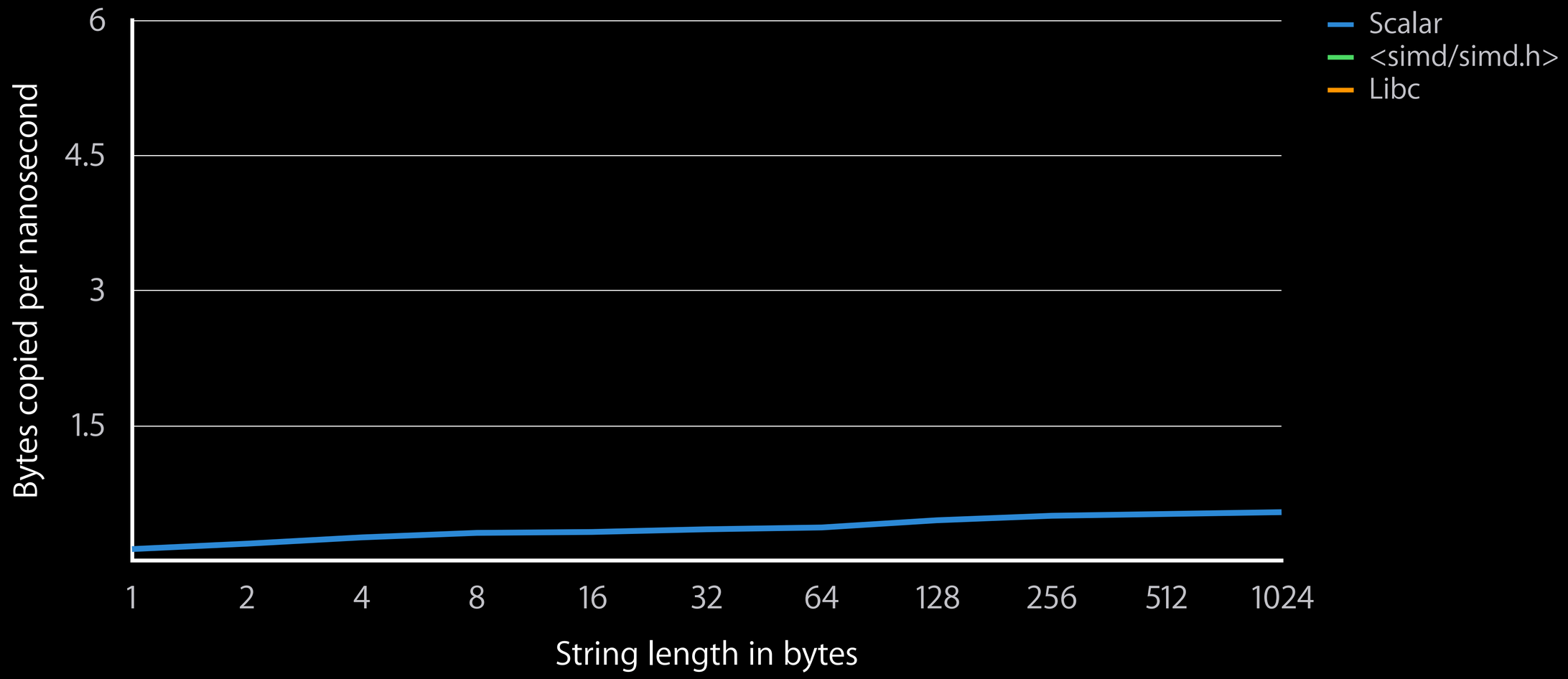
Performance

Higher is better



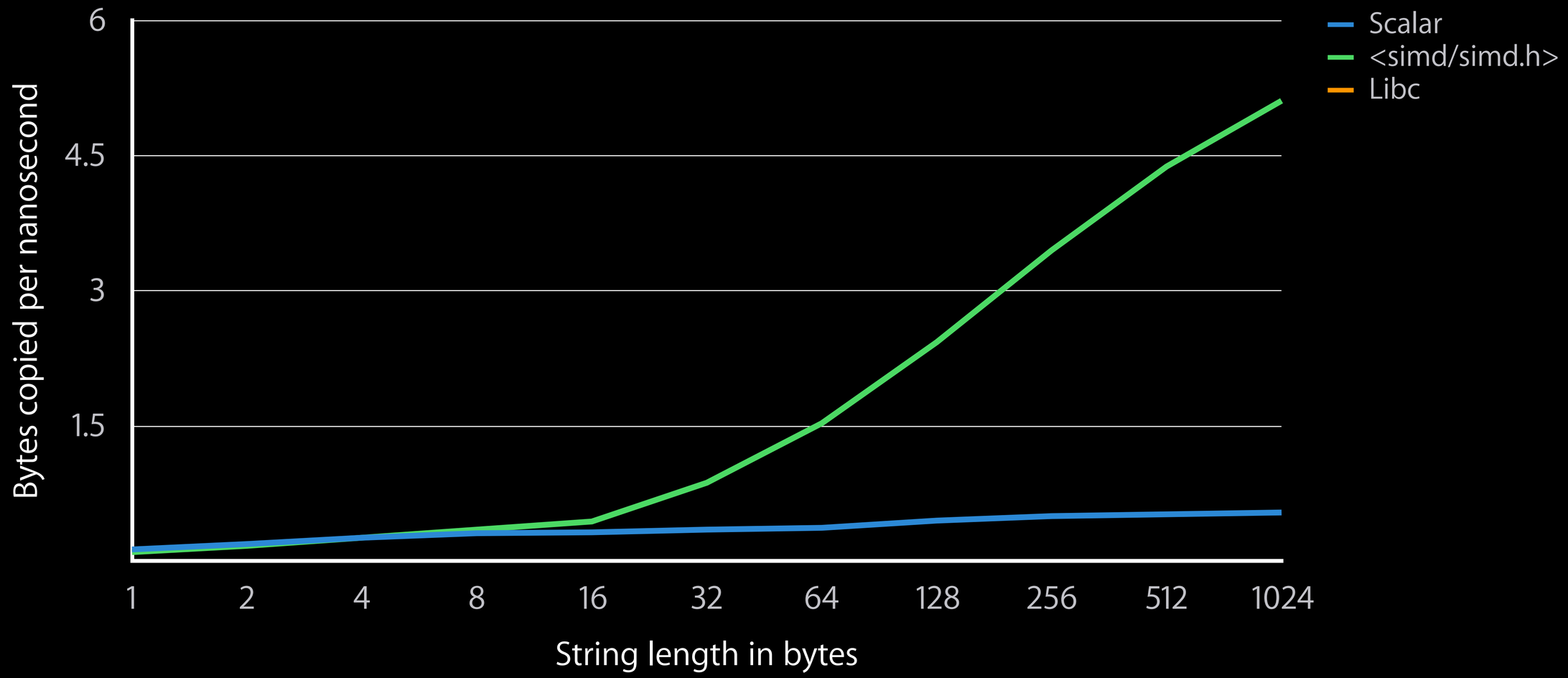
Performance

Higher is better



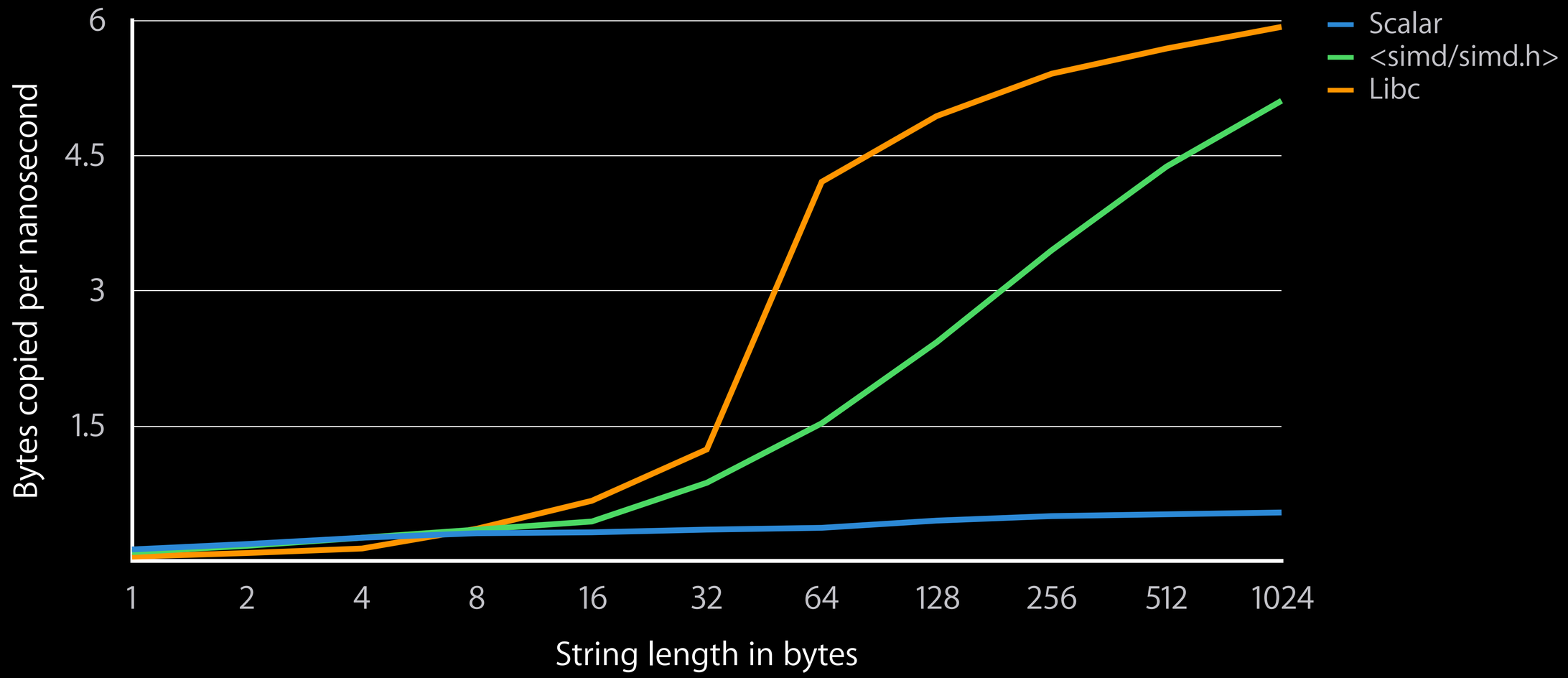
Performance

Higher is better



Performance

Higher is better



Summary

Simple interfaces for complex operations

New libraries still have a few rough edges

Let us know what use cases matter to you, and what additional features you need

More Information

Paul Danbold
Core OS Technology Evangelist
danbold@apple.com

George Warner
DTS Sr. Support Scientist
geowar@apple.com

Documentation
vImage Programming Guide
http://developer.apple.com/library/mac/#documentation/Performance/Conceptual/vImage/Introduction/Introduction.html

More Information

Documentation
vDSP Programming Guide
http://developer.apple.com/library/mac/#documentation/Performance/Conceptual/vDSP_Programming_Guide/Introduction/Introduction.html

vImage Headers
/System/Library/Frameworks/Accelerate.framework/Frameworks/vImage.framework/Headers/vImage.h

vDSP Headers
/System/Library/Frameworks/Accelerate.framework/Frameworks/vecLib.framework/Headers/vDSP.h

More Information

Documentation
LinearAlgebra Headers
/System/Library/Frameworks/Accelerate.framework/Frameworks/vecLib.framework/Headers/LinearAlgebra/LinearAlgebra.h

<simd/simd.h>
/usr/include/simd/simd.h

Apple Developer Forums
http://devforums.apple.com

Bug Report
http://bugreport.apple.com

Related Sessions

• Working with Metal: Overview	Pacific Heights	Wednesday 9:00AM
• Working with Metal: Fundamentals	Pacific Heights	Wednesday 10:15AM
• Working with Metal: Advanced	Pacific Heights	Wednesday 11:30AM

Labs

• Accelerate Lab	Core OS Lab B	Tuesday 11:30AM
• Metal Lab	Graphics and Games Lab A	Wednesday 2:00PM

WWDC14