Inside the Accelerate Framework for iOS

Big performance on small devices

Session 209

Steve Canon
Senior Engineer

Accelerate.framework

What is it?

- "One-stop shopping" for high-performance computational libraries

Accelerate.framework in iOS 4

- vDSP—digital signal processing library
- BLAS—basic linear algebra subroutines
- LAPACK—linear algebra package

Accelerate.framework in Mac OS X

- vDSP—digital signal processing library
- BLAS—basic linear algebra subroutines
- LAPACK—linear algebra package
- vImage—vector image processing framework
- vForce—vector math library

Accelerate.framework in iOS 5

- vDSP—digital signal processing library
- BLAS—basic linear algebra subroutines
- LAPACK—linear algebra package
- vImage—vector image processing framework
- vForce—vector math library

Session Goals

- Introduce new-to-iOS components of Accelerate.framework
- Review improvements made to existing components in iOS 5
- Help you identify places in your code where you could use the Accelerate.framework

vImage

New in iOS 5

vImage

- vectorized Image Processing Framework
- Introduced in OS X 10.3

6 out of 7

Top grossing apps in the
Mac App Store use vImage

vImage Example

Convolution

- One of the most important (and most complex) operations in vImage
- The core of many common image functions

vImage Example

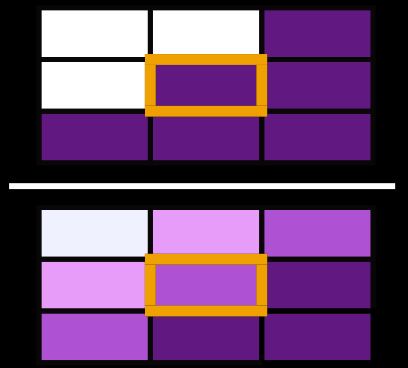
Convolution

- Weighted average of nearby pixels
  - Blur
  - Sharpen
  - Edge detection

Weights:

1	2	1
2	4	2
1	2	1

Pixels: ×



vImage Example

Convolution

- You could write your own

```
for (i=0; i<imageHeight; i++) {
    for (j=0; j<imageWidth; j++) {
        int accumulator = 0;
        for (ik=0; jk<kernelHeight; ik++) {
            for (jk=0; jk<kernelWidth; jk++) {
                accumulator += kernel[k][l] *
                               src[i+ik-kernelHeight/2][j+jk-kernelWidth/2];
            }
        }
        dst[i][j] = accumulator;
    }
}
```

vImage Example

Convolution

- You could write your own
- But you should not
  - Does not handle the edges of the image properly
  - Does not handle integer overflow properly
  - Really slow
- A good convolution requires hundreds of lines of code (or more)

vImage Example

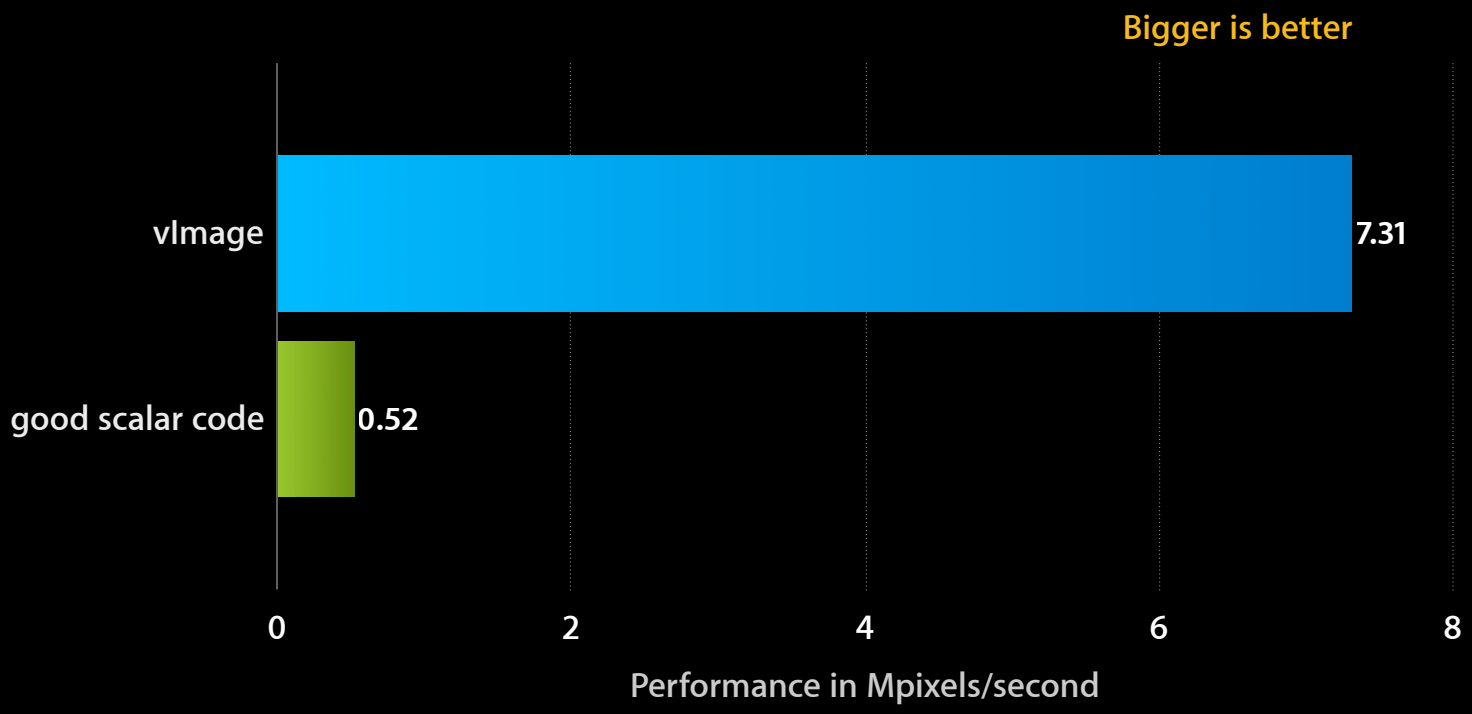
Convolution

- Use vImage instead

```
#include <Accelerate/Accelerate.h>

vImageErr error = vImageConvolve_ARGB8888(source, dest, NULL, 0, 0,
                                          kernel, kernelHeight,
                                          kernelWidth, divisor, NULL,
                                          flags );
```

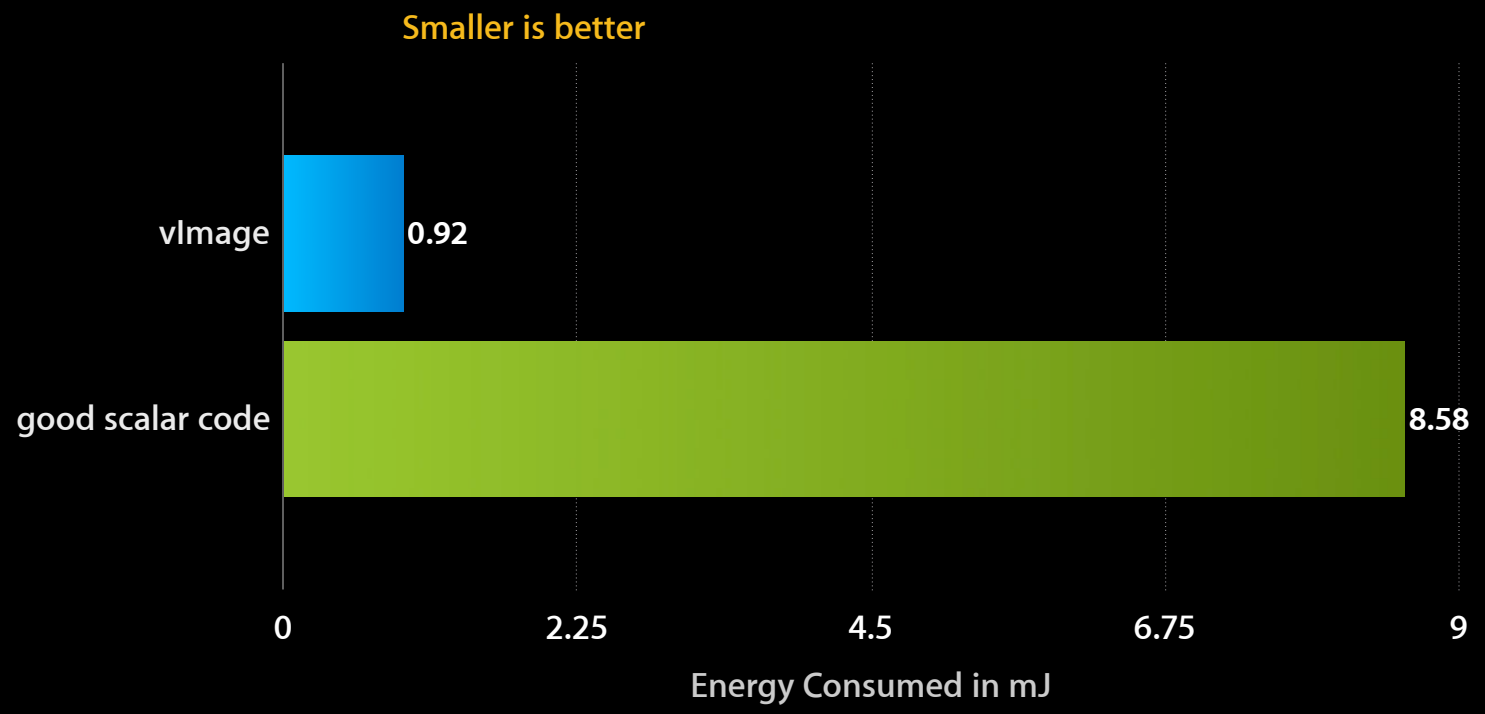
vImage Performance

7x7 convolution on a 1024x768 image, iPad 2



vImage Energy Consumption

7x7 convolution on a 1024x768 image, iPad 2



vImage

Operations

- Convolution
- Geometry
- Transform
- Morphology
- Histogram
- Alpha
- Conversion

vImage Operations

Convolution

- Weighted average of nearby pixels
- Optional bias
- Can use different weights per color channel
- Multiple edging options
  - Background color
  - Edge extend
  - Truncate kernel
  - “Do nothing”

vImage Operations

Geometry

- Rotate
- Shear
- Reduce and Enlarge
- Affine Warp
- Reflect

vImage Operations

Geometry

- Rotate
- Shear
- Reduce and Enlarge
- Affine Warp
- Reflect

vImage Operations

Geometry

- Rotate
- Shear
- Reduce and Enlarge
- Affine Warp
- Reflect

vImage Operations

Geometry

- Rotate
- Shear
- Reduce and Enlarge
- Affine Warp
- Reflect

vImage Operations

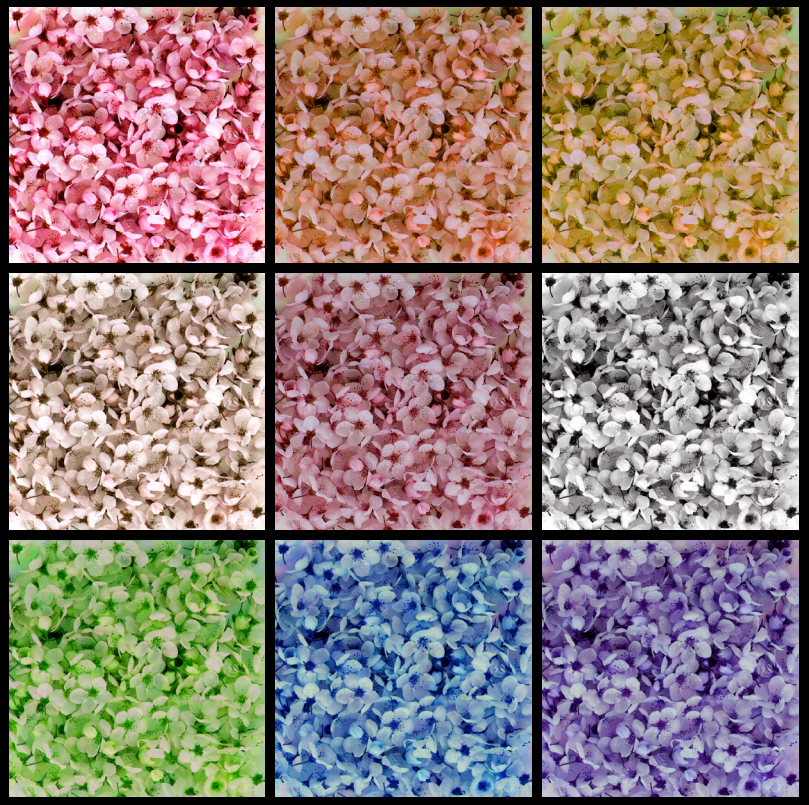
Geometry

- Rotate
- Shear
- Reduce and Enlarge
- Affine Warp
- Reflect

vImage Operations

Transformation

- Matrix multiplication
  - Color space conversion
  - Hue, saturation, brightness
  - Color twist
- Gamma correction
- Polynomial and rational evaluation



vImage Operations

Morphology

- Erode and Dilate
- Min and Max

vImage Operations

Histogram

vImage Operations

Alpha Compositing

- Premultiplied alpha
- Non-premultiplied
- Mixed
- Unpremultiplication
- Premultiplication
- Clip to Alpha

vImage

Data types

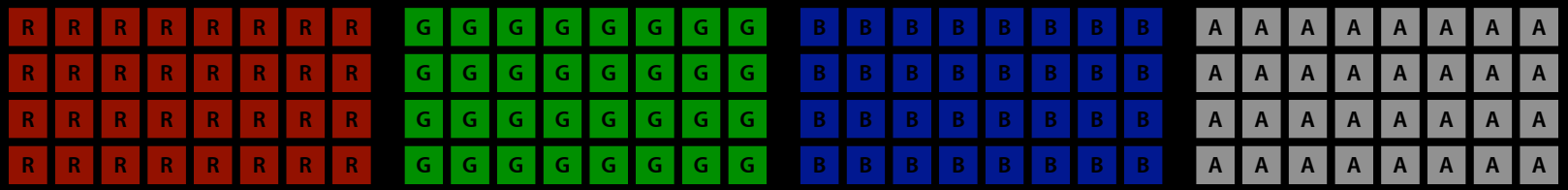
- Core formats
  - 4 channel, 8-bits per channel (UNORM8)
  - 4 channel, 32-bits per channel (floating-point)

vImage Data Layouts

- Interleaved
  - ARGB, RGBA, BGRA, etc.

vImage Data Layouts

- Interleaved
- Planar



vImage Operations

Conversion between core formats

- Planar ↔ Interleaved
- 8-Bit ↔ Float
- Swap channel orders e.g. RGBA ↔ BGRA

vImage Operations

Conversions between core formats and other formats

- RGB565
- ARGB1555
- 16-bit floating-point (“half float”)
- 16-bit unsigned integer
- 16-bit signed integer
- RGB888
- RGBFFF
- RGBX8888
- XRGB8888
- XBGR8888...

vImage Data Requirements

- Minimal alignment requirements
  - Float data requires 4-byte alignment
- Data is not containerized

```
typedef struct {
    void *data;
    vImagePixelCount height;
    vImagePixelCount width;
    size_t rowBytes;
} vImage_Buffer;
```

- vImage can operate on your image data in place

vImage Features

Vectorized for best performance

- Each function uses the best implementation for your hardware
  - On OS X we take advantage of SSE3, SSSE3, SSE4.1 when possible
  - On iOS we take advantage of NEON
    - We use hardware half ↔ float conversions on A5

vImage Features

Designed for low latency operation

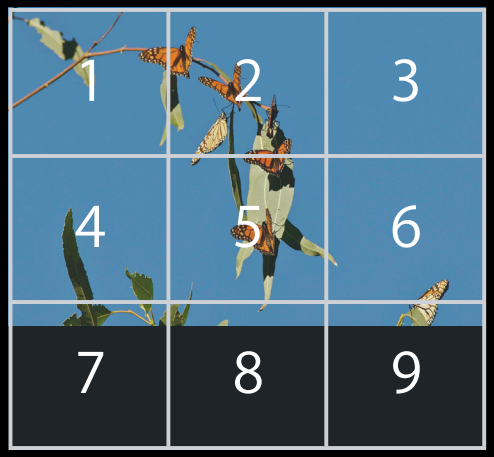
- Does not use JIT (all code is precompiled)
- You can provide temp buffers to avoid hidden malloc/zero-fill; use `kvImageGetTempBufferSize` flag to determine size

```
ssize_t tempSize = vImageMax_ARGB8888(&src, &dest, NULL, ... ,
                                      myFlags | kvImageGetTempBufferSize);
void *tempBuffer = malloc(tempSize);
for (i=0; i<filterCount; i++)
    vImageMax_ARGB8888(&src, &dst, tempBuffer, ... , myFlags);
free(tempBuffer);
```

vImage Features

Threaded using GCD

- Transparently take advantage of multiple processors
- Threading can be disabled using the `kvImageDoNotTile` flag, so it does not conflict with your tiling model
- APIs support your tiling model even in the presence of edging



vForce

New in iOS 5

Luke Chang
Engineer

Satisfy Your Computational Need

- Elementary math functions for arrays
- Introduced in Mac OS X 10.4, now on iOS 5.0

vForce in Action

Filling a buffer with sine wave using a for loop

```
float buffer[length];
float indices[length];

...

for (int i = 0; i < length; i++)
{
    buffer[i] = sinf(indices[i]);
}
```

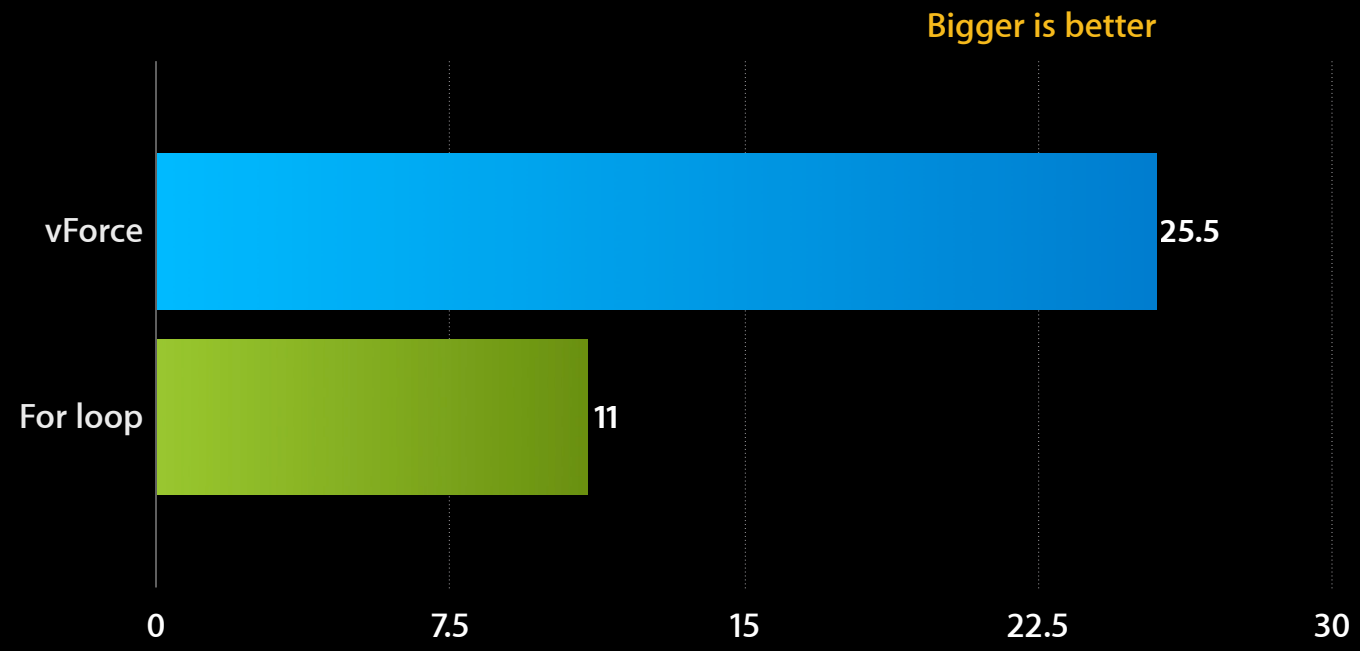
vForce in Action

Filling a buffer with sine wave using vForce

```
#include <Accelerate/Accelerate.h>

float buffer[length];
float indices[length];

...

vvsinf(buffer, indices, &length);
```

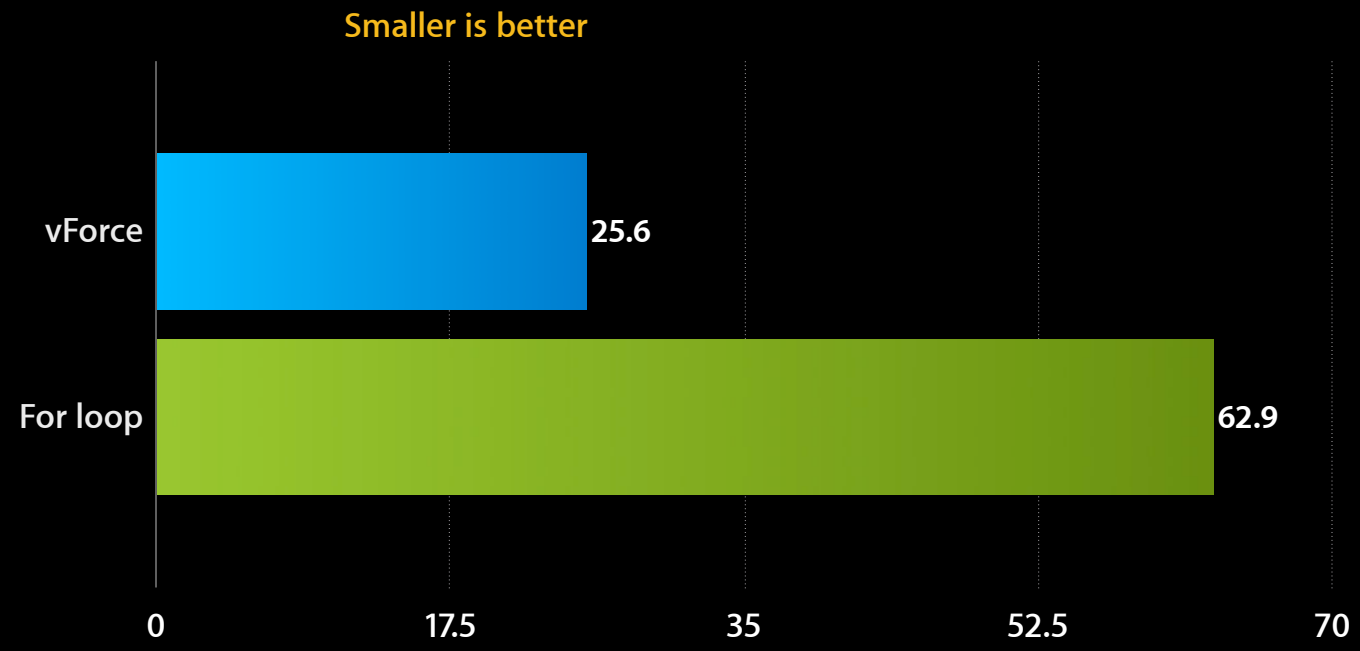
Better Performance

Sines computed per µs on iPad 2 (iOS 5)



Less Energy

nJ consumed per sine on iPad 2 (iOS 5)

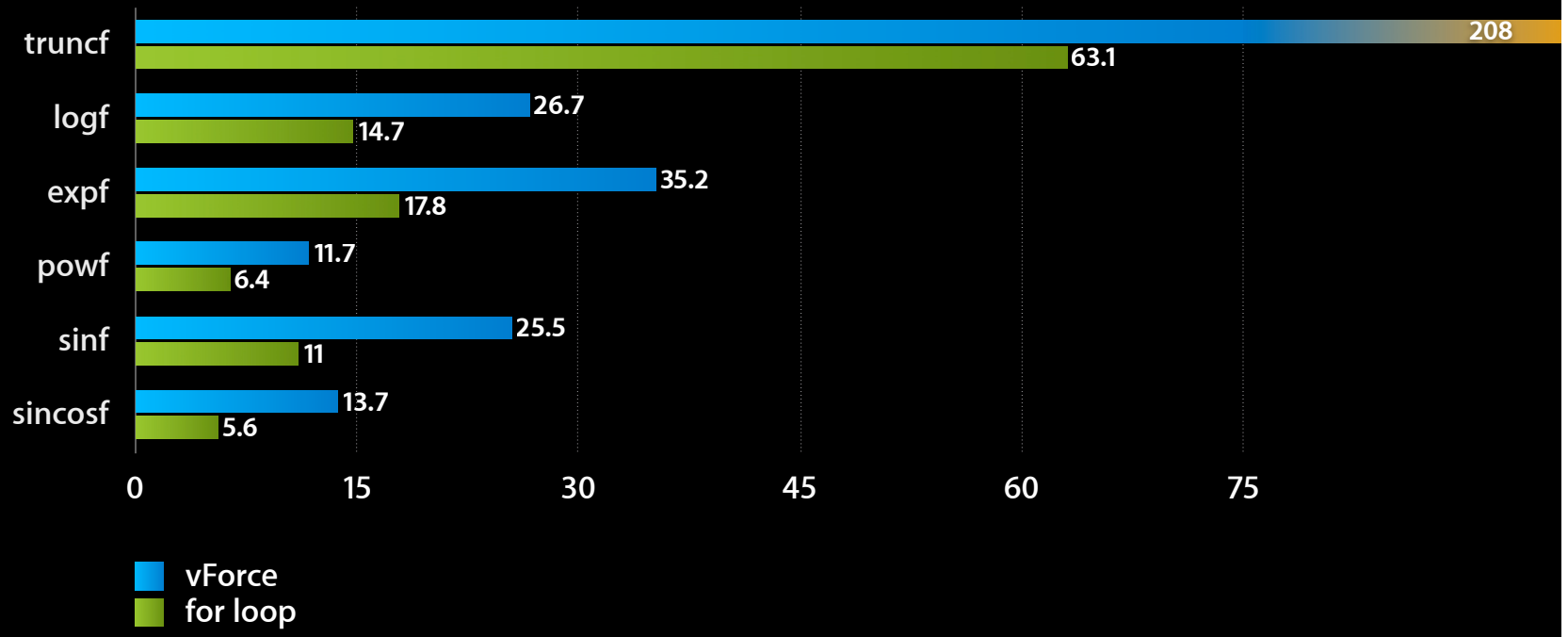


What Is Available?

- Commonly used transcendental functions
  - Power, sine, cosine, logarithm, exponential, etc.
- Rounding functions
  - Ceiling, floor, truncation, nearest integer
- Lots of other stuff
  - Square root, remainder, etc.

vForce Performance

Results computed per µs on iPad 2 (iOS 5)



vForce in Detail

- Support both float and double
- Correct edge case handling
- Minimal alignment requirements

How Did We Do It?

- vForce single precision is vectorized using NEON
- vForce exploits software pipelining and loop unrolling

LAPACK and BLAS

Improved in iOS 5

Steve Canon
Senior Engineer

LAPACK and BLAS

- Improved performance in iOS 5
- Takes advantage of the A5 processor for great double-precision performance

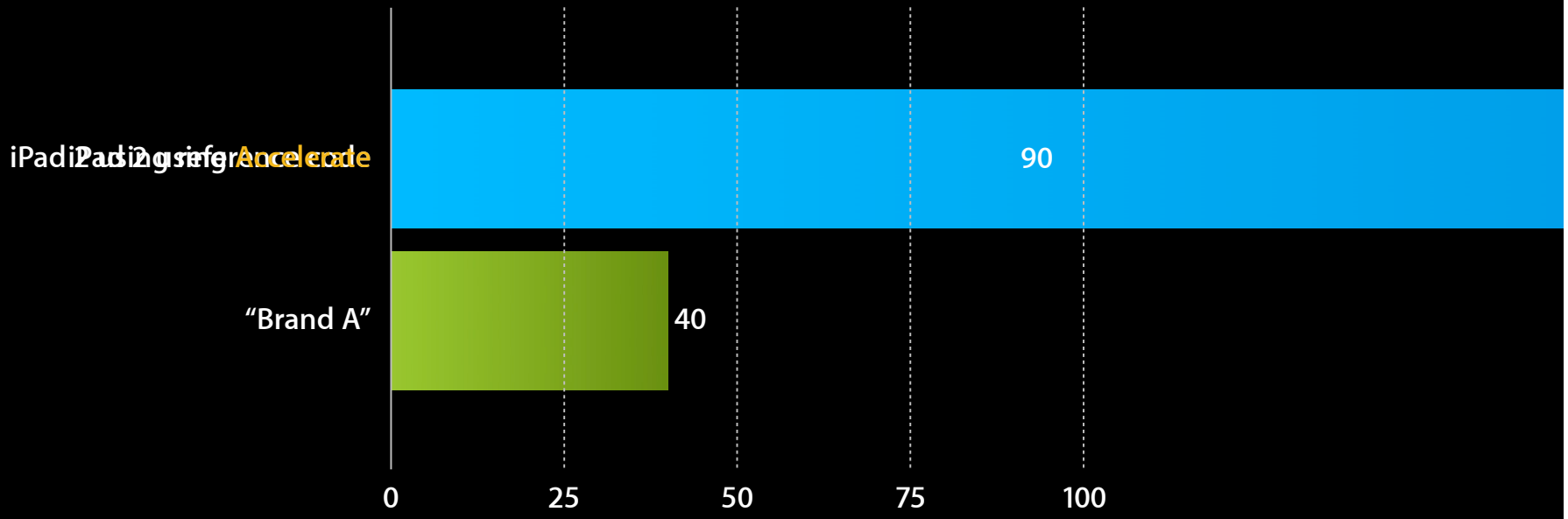
LAPACK and BLAS

LINPACK benchmark

- How fast can you solve a system of linear equations?
- Actually 3 different benchmarks
  - 100 equations, using the reference implementation
  - 1000 equations, using your tuned implementation
  - "No holds barred"

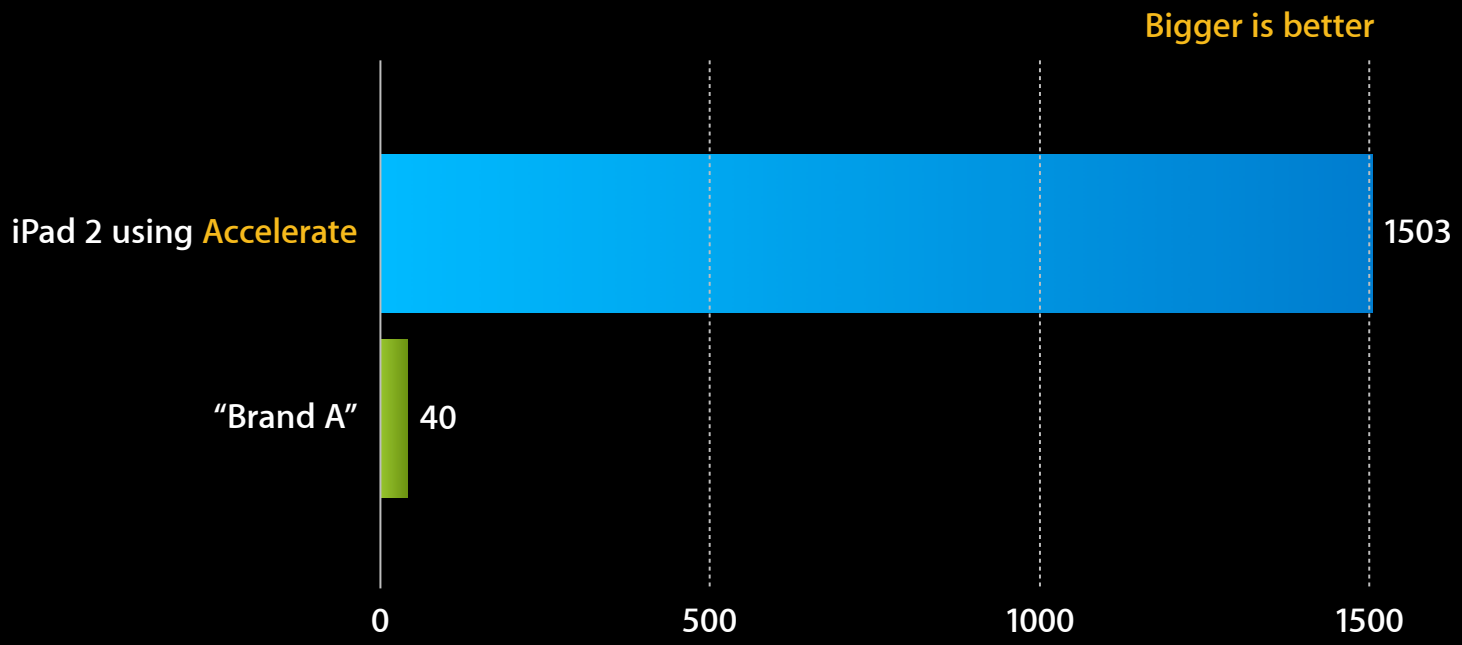
LAPACK and BLAS

LINPACK benchmark performance in Mflops



LAPACK and BLAS

LINPACK benchmark performance in Mflops



LAPACK and BLAS

LINPACK benchmark

- iPad 2 would have been one of the 500 fastest supercomputers in the world in 1994!

LAPACK and BLAS

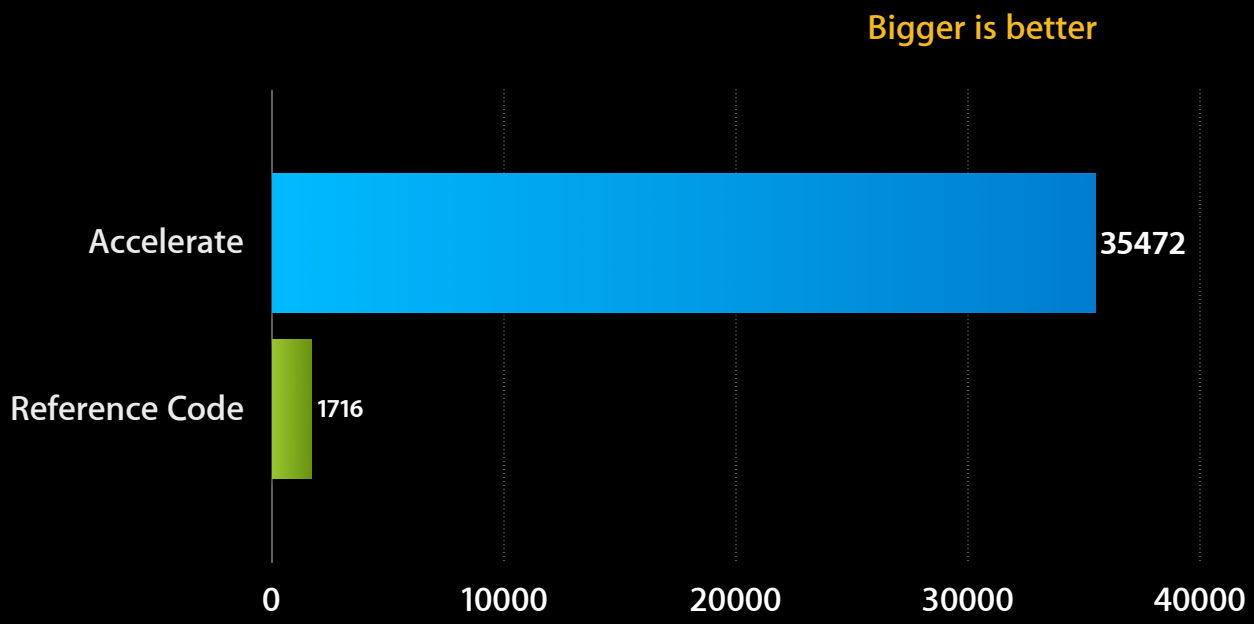
- Great performance without writing a lot of code

```
#include <Accelerate/Accelerate.h>

dgetrf_(&n, &n, a, &n, ipiv, &info);
dgetrs_("N", &n, &one, a, &n, ipiv, b, &n, &info);
```

- Same code works great on the Mac

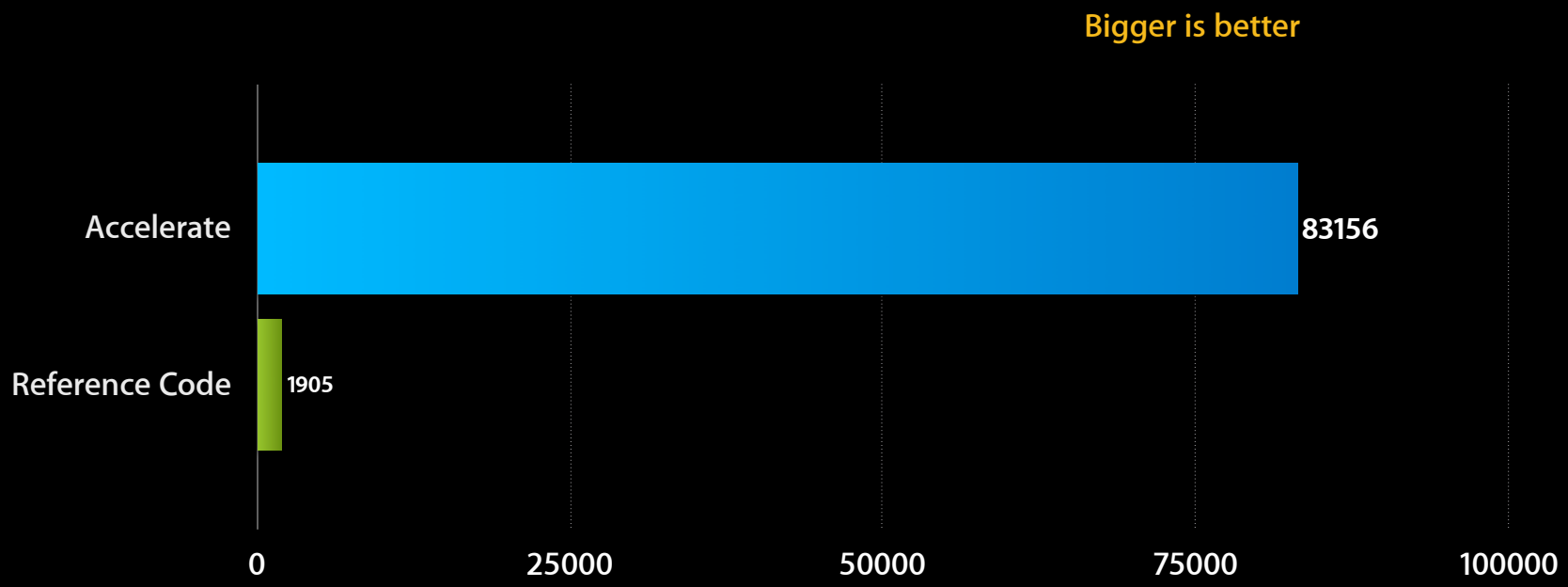
LAPACK and BLAS

MacBook Pro LINPACK performance in Mflops



LAPACK and BLAS

Mac Pro LINPACK performance in Mflops



Summary

- Easier than writing your own code
- Great performance on diverse hardware
- Low energy usage

More Information

Paul Danbold
Core OS Technologies 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

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

Labs

Accelerate Framework Lab	Core OS Lab A Thursday 11:30AM–1:30PM

Q&A


WWDC2011