Using Accelerate and simd

Session 701

Matthew Badin, CoreOS, Vector and Numerics
Luke Chang, CoreOS, Vector and Numerics
Accelerate
vDSP
simd
vImage
What Is Accelerate Framework?
Functionality
Performance
Energy Efficiency
Composition of Accelerate Framework

vDSP — Signal processing
vImage — Image processing
vForce — Vector transcendental functions
BLAS, LAPACK, LinearAlgebra — Dense matrix computations
Sparse BLAS, Sparse Solvers — Sparse matrix computations
BNNS — Neural networks
Closely Related

**simd**—Small vector and matrix computation for CPU

**Compression**—Lossless data compression
Signal Processing Library

Basic operations on arrays
- Add, subtract, multiply, conversion, etc.

Discrete Fourier/Cosine Transform
- 1D DFT/DCT/FFT
- 2D FFT

Convolution and correlation
Demo

Extract signal from noise
Essential Computations

Analyze noisy signal with forward DCT
Remove frequency components of amplitude less than threshold
Reconstruct clean signal with inverse DCT
// Create a Forward DCT Setup Object for vDSP_DCT_Execute

let dctSetup_FORWARD: vDSP_DFT_Setup = {
    guard let dctSetup = vDSP_DCT_CreateSetup(
        nil, vDSP_Length(numSamples), .II) else {
        fatalError("can't create FORWARD vDSP_DFT_Setup")
    }
    return dctSetup

}()
// Perform Forward DCT

var forwardDCT = [Float](repeating: 0,
    count: numSamples)

vDSP_DCT_Execute(dctSetup_FORWARD, noisySignalReal, &forwardDCT)
// All Values in `forwardDCT` Less Than `threshold` To 0

vDSP_vthres(forwardDCT, stride, &threshold, &forwardDCT, stride, count)
// Create an Inverse DCT Setup Object for vDSP_DCT_Execute

let dctSetup_INVERSE: vDSP_DFT_Setup = {
    guard let dctSetup = vDSP_DCT_CreateSetup(
        nil, vDSP_Length(numSamples), .III) else {
        fatalError("can't create INVERSE vDSP_DFT_Setup")
    }
    return dctSetup
}()
// Reconstruct Clean Signal
vDSP_DCT_Execute(dctSetup_INVERSE, forwardDCT, &inverseDCT)

// Apply Normalization Factor
var divisor = Float(count)
vDSP_vsdiv(inverseDCT, stride, &divisor, &inverseDCT, stride, count)
Demo

Halftone de-screening
Essential Computations

Transform halftone image with 2D FFT

Remove frequency components of the halftone screen

Reconstruct continuous tone image
// Create a FFT Setup Object, No Direction Specified

let fftSetUp: FFTSetup = {
    let log2n = vDSP_Length(log2(1024.0 * 1024.0))
    guard let fftSetUp = vDSP_create_fftsetup(log2n, FFTRadix(kFFTRadix2)) else {
        fatalError("can't create FFT Setup")
    }
    return fftSetUp
} ()
// Perform 2D FFT

let sourceImage_floatPixels_frequency = DSPSplitComplex(
    realp: &sourceImage_floatPixelsReal_spatial,
    imagp: &sourceImage_floatPixelsImag_frequency)

vDSP_fft2d_zrop(fftSetUp, &sourceImageSplitComplex, vDSP_Stride(1), vDSP_Stride(0),
    &sourceImage_floatPixels_frequency, vDSP_Stride(1), vDSP_Stride(0),
    vDSP_Length(log2(Float(width))),
    vDSP_Length(log2(Float(height))),
    FFTDirection(kFFTDirection_Foward))
Frequency Removal

zvmags
vthrsc
vclip
zrvmul
Perform Inverse FFT to Create Image from Frequency Domain Data

```swift
var floatPixels_spatial = DSPSplitComplex(realp: &floatPixelsReal_spatial,
                                          imagp: &floatPixelsImag_spatial)

vDSP_fft2d_zrop(fftSetUp, &sourceImage_floatPixels_frequency,
                 stride, 0,
                 &floatPixels_spatial,
                 stride, 0,
                 vDSP_Length(log2(Float(width))),
                 vDSP_Length(log2(Float(height))),
                 FFTDirection(kFFTDirection_Inverse))
```
The simd Module
The simd Module

Simplified vector programming

Small (fixed-size) vector and matrices

Abstract architecture-specific types and instrinsics
// Average Two Vectors

var x: [Float] = [1, 2, 3, 4]
var y: [Float] = [3, 3, 3, 3]
var z: [Float](repeating: 0, count: 4)

for i in 0..<4 {
    z[i] = (x[i] + y[i]) / 2.0
}

// Average Two Vectors

let x = simd_float4(1,2,3,4)
let y = simd_float4(3,3,3,3)

let z = 0.5 * (x + y)
Matrices of size 2, 3, or 4 and vectors of 2, 3, 4, 8, 16, 32, or 64

Arithmetic operators (+, -, *, /) work with both vectors and scalars

Supports common vector math and geometry operations (dot, length, clamp)

Support for transcendental functions

Quaternions
Matrices of size 2, 3, or 4 and vectors of 2, 3, 4, 8, 16, 32, or 64

Arithmetic operators (+, -, *, /) work with both vectors and scalars

Supports common vector math and geometry operations (dot, length, clamp)

Support for transcendental functions

Quaternions
// Vector to Rotate
let original = simd_float3(0, 0, 1)

// Axis of Rotation
let quaternion = simd_quatf(angle: .pi / -3,
                        axis: simd_float3(1, 0, 0))

// Apply the Rotation
let rotatedVector = simd_act(quaternion, original)
// Vector to Rotate
let original = simd_float3(0,0,1)

// Axis of Rotation
let quaternion = simd_quatf(angle: .pi / 3,
    axis: simd_float3(1,0,0))

// Apply the Rotation
let rotatedVector = simd_act(quaternion, original)
let original = simd_float3(0,0,1)

let quaternion = simd_quatf(angle: .pi / -3,
                           axis: simd_float3(1,0,0))

let rotatedVector = simd_act(quaternion, original)
// Vector to Rotate
let original = simd_float3(0,0,1)

// Axis of Rotation
let quaternion = simd_quatf(angle: .pi / -3,
    axis: simd_float3(1,0,0))

// Apply the Rotation
let rotatedVector = simd_act(quaternion, original)
// Vector to Rotate
let original = simd_float3(0,0,1)

// Axis of Rotation
let quaternion = simd_quatf(angle: .pi / -3,
                          axis: simd_float3(1,0,0))

let quaternion2 = simd_quatf(angle: .pi / 3,
                          axis: simd_float3(0,1,0))

// Apply the Rotation
let rotatedVector = simd_act(quaternion, original)
let original = simd_float3(0,0,1)

let quaternion = simd_quatf(angle: .pi / -3,
                           axis: simd_float3(1,0,0))

let quaternion2 = simd_quatf(angle: .pi / 3,
                               axis: simd_float3(0,1,0))

let quaternion3 = quaternion2 * quaternion

let rotatedVector = simd_act(quaternion, original)
// Vector to Rotate
let original = simd_float3(0,0,1)

// Axis of Rotation
let quaternion = simd_quatf(angle: .pi / -3,
    axis: simd_float3(1,0,0))

let quaternion2 = simd_quatf(angle: .pi / 3,
    axis: simd_float3(0,1,0))

// Combine the Two Rotations
let quaternion3 = quaternion2 * quaternion

// Apply the Rotation
let rotatedVector = simd_act(quaternion3, original)
// Vector to Rotate
let original = simd_float3(0,0,1)

// Axis of Rotation
let quaternion = simd_quatf(angle: .pi / -3,
axis: simd_float3(1,0,0))

let quaternion2 = simd_quatf(angle: .pi / 3,
axis: simd_float3(0,1,0))

// Combine the Two Rotations
let quaternion3 = quaternion2 * quaternion

// Apply the Rotation
let rotatedVector = simd_act(quaternion3, original)
// Slerp Interpolation
let blue = simd_quatf(…)

let green = simd_quatf(…)

let red = simd_quatf(…)

for t: Float in stride(from: 0, to: 1, by: 0.001) {
    let q = simd_slerp(blue, green, t)
    // Code to Add Line Segment at `q.act(original)`
}

for t: Float in stride(from: 0, to: 1, by: 0.001) {
    let q = simd_slerp_longest(green, red, t)
    // Code to Add Line Segment at `q.act(original)`
}
// Slerp Interpolation

let blue = simd_quatf(…)

let green = simd_quatf(…)

let red = simd_quatf(…)

for t: Float in stride(from: 0, to: 1, by: 0.001) {
    let q = simd_slerp(blue, green, t)
    // Code to Add Line Segment at `q.act(original)`
}

for t: Float in stride(from: 0, to: 1, by: 0.001) {
    let q = simd_slerp_longest(green, red, t)
    // Code to Add Line Segment at `q.act(original)`
}
// Spline Interpolation

let original = simd_float3(0, 0, 1)
let rotations: [simd_quatf] = ...

for i in 1 ... rotations.count - 3 {
    for t: Float in stride(from: 0, to: 1, by: 0.001) {
        let q = simd_spline(rotations[i - 1],
                            rotations[i],
                            rotations[i + 1],
                            rotations[i + 2],
                            t)

        // Code to Add Line Segment at `q.act(original)`
    }
}
// Spline Interpolation

let original = simd_float3(0,0,1)

let rotations: [simd_quatf] = ...

for i in 1...rotations.count - 3 {
    for t: Float in stride(from: 0, to: 1, by: 0.001) {
        let q = simd_spline(rotations[i - 1],
                            rotations[i],
                            rotations[i + 1],
                            rotations[i + 2],
                            t)

        // Code to Add Line Segment at `q.act(original)`
    }
}
// Spline Interpolation

let original = simd_float3(0,0,1)
let rotations: [simd_quatf] = ...

for i in 1 ... rotations.count - 3 {
    for t: Float in stride(from: 0, to: 1, by: 0.001) {
        let q = simd_spline(rotations[i - 1],
                           rotations[i],
                           rotations[i + 1],
                           rotations[i + 2],
                           t)
        // Code to Add Line Segment at `q.act(original)`
    }
}
vForce  BNNS
BLAS  LAPACK  LinearAlgebra
vDSP  simd  vImage
Sparse BLAS  Sparse
Compression
vForce, BNNS, BLAS, LAPACK, Linear Algebra, vDSP, simd, vImage, Sparse BLAS, Sparse, Compression
Image Processing Library

Conversion
Geometry
Convolution
Transform
Morphology
Demo

Video effects app
Workflow

Receive captured image from camera
Prepare vImage input/output buffers
Apply effects with vImage functions
Display output image on screen
Workflow

Receive captured image from camera
Prepare vImage input/output buffers
Apply effects with vImage functions
Display output image on screen
Adjust Color Saturation

The formula to adjust color saturation is:

\[ Cb = ((Cb - 128) \times \text{saturation}) + 128 \]
\[ Cr = ((Cr - 128) \times \text{saturation}) + 128 \]
```swift
var preBias: Int16 = -128

// Fixed-Point Arithmetics in Q12 Format
let divisor: Int32 = 0x1000
var postBias: Int32 = 128 * divisor

// Convert Saturation to Q12 Format
var matrix = [ Int16(saturation * Float(divisor)) ]

vImageMatrixMultiply_Planar8(&sources, &destinations, 1, 1,
    &matrix, divisor, &preBias, &postBias,
    vImage_Flags(kvImageNoFlags))
```
Workflow

Receive captured image from camera
Prepare vImage input/output buffers
Apply effects with vImage functions
Display output image on screen
func captureOutput(_ output: AVCaptureOutput, didOutput sampleBuffer: CMSampleBuffer, from connection: AVCaptureConnection) {

    // Get CVImageBuffer from CMSampleBuffer
    let pixelBuffer = sampleBuffer.imageBuffer

    // Make Sure pixelBuffer Is Accessible to CPU
    CVPixelBufferLockBaseAddress(pixelBuffer, .readOnly)

    // Apply Effects with vImage Functions
    ...

    // Unlock the BaseAddress of pixelBuffer
    CVPixelBufferUnlockBaseAddress(pixelBuffer, .readOnly)
}

// AVCaptureVideoDataOutputSampleBufferDelegate
Workflow

Receive captured image from camera

Prepare `vlImage` input/output buffers

Apply effects with `vlImage` functions

Display output image on screen
// Prepare vImage Input Buffer for Luminance

let lumaBaseAddress = CVPixelBufferGetBaseAddressOfPlane(pixelBuffer, 0)
let lumaWidth = CVPixelBufferGetWidthOfPlane(pixelBuffer, 0)
let lumaHeight = CVPixelBufferGetHeightOfPlane(pixelBuffer, 0)
let lumaRowBytes = CVPixelBufferGetBytesPerRowOfPlane(pixelBuffer, 0)

var sourceLumaBuffer = vImage_Buffer(data: lumaBaseAddress,
   height: vImagePixelCount(lumaHeight),
   width: vImagePixelCount(lumaWidth),
   rowBytes: lumaRowBytes)

// Prepare vImage Input Buffer for Chrominance

...
// Initialize vImage Output Buffer

var destinationBuffer = vImage_Buffer()

vImageBuffer_Init(&destinationBuffer,
    sourceLumaBuffer.height, sourceLumaBuffer.width,
    cgImageFormat.bitsPerPixel, vImage.Flags(kvImageNoFlags))
Workflow

Receive captured image from camera

Prepare vImage input/output buffers

Apply effects with vImage functions

Display output image on screen
// Convert YCbCr Image to ARGB

vImageConvert_420Yp8_CbCr8ToARGB8888(&sourceLumaBuffer, &sourceChromaBuffer,
   &destinationBuffer, &infoYpCbCrToARGB,
   nil, 255, vImage_Flags(kvImageNoFlags))

// Create CGImage from vImage_Buffer for Display
// Creating CGImage Object Does Not Copy Image Buffers

let cgImage = vImageCreateCGImageFromBuffer(&destinationBuffer, &cgImageFormat,
   nil, nil, vImage_Flags(kvImageNoFlags),
   &error)
// Display the Image on Screen
if let cgImage = cgImage, error == kvImageNoError {
    DispatchQueue.main.async {
        self.imageView.image = UIImage(cgImage: cgImage.takeRetainedValue())
    }
}
Other Effects

Rotation
Blur
Dither
Color quantization
Demo
More video effects
let backcolor: [UInt8] = [255, 255, 255, 255]

vImageRotate_ARGB8888(&destinationBuffer, &destinationBuffer, nil,
             fxValue, backColor, vImage_Flags(kvImageBackgroundColorFill))
Blur Effect

vImageTentConvolve_ARGB8888(&tmpBuffer, &destinationBuffer, nil,
0, 0, kernelSize, kernelSize, nil,
vImage_Flags(kvImageEdgeExtend))
Dither Effect

vImageConvert_Planar8toPlanar1(&sourceLumaBuffer,
    &ditheredLuma,
    nil,
    Int32(kvImageConvert_DitherAtkinson),
    vImage_Flags(kvImageNoFlags))
Color Quantization Effect

```swift
var lookUpTable = (0...255).map {
    return Pixel_8(($0 / quantizationLevel) * quantizationLevel)
}

vImageTableLookUp_ARGB8888(&destinationBuffer, &destinationBuffer, nil, &lookUpTable, &lookUpTable, &lookUpTable, vImage_Flags(kvImageNoFlags))
```
LINPACK Benchmark
LINPACK Benchmark

How fast can you solve a system of equations?

Actually three separate benchmarks:

• 100-by-100 system
• 1000-by-1000 system
• “No holds barred”
LINPACK Benchmark
Performance in GFLOPS (bigger is better)

- iPhone 5S: 7.5
- iPhone 6: 9
- iPhone 6S: 14.2
- iPhone 7: 24.8
- iPhone X: 28.7

using Accelerate
LINPACK Benchmark
Performance in GFLOPS (bigger is better)

- iPhone 5S: 18.2 GFLOPS
- iPhone 6: 21.1 GFLOPS
- iPhone X using Accelerate: 28.7 GFLOPS
LINPACK Benchmark
Performance in GFLOPS (bigger is better)

- iPhone 5S: 18.2
- iPhone 6: 21.1
- iPhone 6S: 36.9
- iPhone 7: 55
- iPhone X using Accelerate: 68
  - Single: 28.7
  - Double: 68
macOS  iOS  tvOS  watchOS
Summary

Wide varieties of functionalities

Easy to use

Fast and energy efficient

Portable across platforms and architectures
More Information

https://developer.apple.com/wwdc18/701

Accelerate Lab

Technology Lab 2

Wednesday 2:00PM