vDSP Library

2005-09-08

Contents

Figures, Tables, and Listings

Introduction to the vDSP Library

Framework: Accelerate

The vDSP Library provides mathematical functions for applications such as speech, sound, audio, and video processing, diagnostic medical imaging, radar signal processing, seismic analysis, and scientific data processing.

The vDSP functions operate on real and complex data types. The functions include data type conversions, fast Fourier transforms (FFTs), and vector-to-vector and vector-to-scalar operations.

The vDSP functions have been implemented in two ways: as vectorized code, which uses the Velocity Engine technology in the PowerPC G4 microprocessor, and as scalar code, which runs on Macintosh models that have a G3 microprocessor.

The vDSP Library itself is included as part of vecLib in both Mac OS 9 and Mac OS X. The header file, `vDSP.h`, is provided for Macintosh software developers; it defines nonstandard data types used by the vDSP functions and symbols accepted as flag arguments to vDSP functions.

This chapter publishes calling conventions and operational descriptions for each function supported by the vDSP Library, which is part of vecLib.

Each function's synopsis specifies calling conventions using the function's ANSI C prototype.

Function descriptions appear here in alphabetical order. Where functions are grouped together with a shared description, they are alphabetized by the first function listed in the group. The functions are listed by logical groupings in Chapter 3, "Function Type Reference."

The single-precision functions have been implemented as vectorized code, which uses the Velocity Engine technology in the PowerPC G4 or G5 microprocessor, and as scalar code, which runs on Macintosh models that have a G3 microprocessor. When the Velocity Engine is present, the vectorized code is used as long as certain criteria are met. Those criteria are listed for each function; if none is listed, then the following general rules should be followed:

- Vectors that are passed to the function should be "relatively aligned," i.e., their physical addresses should be congruent modulo 16. This applies to both real and imaginary parts of complex vectors.
- Stride values should be 1.

There is no Velocity Engine support for double-precision functions; they are implemented as scalar code only.

Note: The criteria for invoking vectorized code may change in future implementations, but they will only get less restrictive.

Calling Conventions

This section describes the calling conventions used with the vDSP functions and discusses the use of address strides.

Passing Parameters

Most vDSP functions are provided with input data and produce output data. All such data arguments, whether vector or scalar, are passed by reference; that is, callers pass pointers to memory locations from which input data is read and to which output data is written.

Other argument types passed to vDSP functions include:

- Address strides, which tell a function how to step through input and output data. Stride examples include every element, every other element, every third element, and so forth.
- Flags, which influence a function's behavior in some way. Examples include forward versus inverse directional flags for discrete Fourier transforms.
- Element counts, which tell functions how many elements to process.

Address strides, flags, and element counts are integer values and, unlike input and output data, these values are passed to a function directly, not by reference. For example, the vector multiply, `vDSP_vmul`, accepts three of the four argument types:

```
void vDSP_vmul(
float *input_1, /* input vector 1 */
SInt32 stride_1, /* address stride for input vector 1 */
float *input_2, /* input vector 2 */
SInt32 stride_2, /* address stride for input vector 2 */
float *result, /* output vector */
SInt32 strideResult, /* address stride for output vector */
UInt32 size /* real output count */
);
```

A typical call to `vDSP_vmul()` illustrates how the input and output data is passed by reference, whereas address strides, flags, and element counts pass directly:

```
/* Multiply sequential values of two 1,024-point vectors */
float a[1024], b[1024], c[1024];
vDSP_vmul( a, 1, b, 1, c, 1, 1024 );
```

Specifying Address Strides

As mentioned earlier, address strides tell functions how to step through input and output data: every element, every second element, every third element, and so on. Though usually positive, address strides can be specified as negative for most vDSP functions. Developers should be aware, however, that negative address strides can often change the arithmetic operation of a function, so experimentation with input values that produce known results is recommended.

When operating on complex vectors, vector elements, which are complex numbers, are stored as ordered pairs of real elements. Therefore, a stride of 2 is specified to process every vector element; a stride of 4 is specified to process every other element; and so forth.

When operating on real vectors, address strides of 1 address every element, address strides of 2 address every other element, and so forth. For some functions, address strides of 1 for real vectors result in superior performance over non-unit strides. The use of split complex yields similar performance benefits for complex vectors.

Important: In the present implementation, the use of strides other than 1 in some functions causes those functions to be performed in scalar code—that is, without using the Velocity Engine. See the descriptions of individual functions in Chapter 4, "vDSP Reference."

Return Values

Repeating the synopsis of `vDSP_vmul()`, the definition line shows `vDSP_vmul()` to be of type void; that is, it returns no value to the caller:

```
void vDSP_vmul(a, i, b, j, c, k, n)
```

Important: To achieve the best possible performance, the vDSP routines perform no error checking and returns no completion codes. All arguments are assumed to be specified and passed correctly, and these responsibilities lie with the caller.

Data Types

Four numeric data types used by vDSP functions equate to standard C data types; those are real, integer, short integer, and character.

The vDSP Library defines two additional data types:

- DSPComplex: Ordered pair of real numbers
- DSPSplitComplex: Ordered pair of real vectors

DSPComplex

The complex data type is defined in the header file `vDSP.h`. Its definition is:

```
struct DSPComplex {
    float                       real;
    float                       imag;
};
typedef struct DSPComplex           DSPComplex;
```

Complex data are stored as ordered pairs of floating-point numbers. Because they are stored as ordered pairs, complex vectors require address strides that are multiples of two.

DSPSplitComplex

The split complex data type is defined in the header file vDSP.h. Its definition is:

```
struct DSPSplitComplex {
    float *                     realp;
    float *                     imagp;
};
typedef struct DSPSplitComplex      DSPSplitComplex;
```

Split complex vectors are stored as two vectors, a vector containing all the real values and a vector containing all the imaginary values.

Using Fourier Transforms

The vDSP Library provides Fourier transforms for transforming data from the time domain to the frequency domain and back. Functions are available for one-dimensional and two-dimensional transforms.

FFT Weights Arrays

To boost performance, vDSP functions that process frequency-domain data expect an array of complex exponentials, sometimes called twiddle factors, to exist prior to calling the function. Once created, this array, called an FFT weights array, can be used over and over by the same Fourier function and can be shared by several Fourier functions.

FFT weights arrays are created by calling `vDSP_create_fftsetup`. Before calling a function that processes in the frequency domain, you must call `vDSP_create_fftsetup`. The caller specifies the required array size to the setup function. The function:

- Creates a data structure to hold the array.
- Builds the array.
- Returns a pointer to the data structure. (If adequate storage to build the array is unavailable, the function returns zero.)

The pointer to the data structure is then passed as an argument to subsequent Fourier functions. You must check the value of the pointer before supplying the pointer to a frequency domain function. The `vDSP_create_fftsetup` function might not find enough storage to allocate for the weights array, in which case the function returns a zero. A returned value of zero is the only explicit notification that the array allocation failed.

Argument `log2n` to `vDSP_create_fftsetup` is the base-2 exponent of `n`, where `n` is the number of complex unit circle divisions the array represents, and thus specifies the largest number of elements that can be processed by a subsequent Fourier function. Argument `log2n` must equal or exceed argument `log2n` supplied to any functions using the weights array. Functions automatically adjust their strides through the array when the table has more resolution, or larger `n`, than required. Thus, a single call to `vDSP_create_fftsetup()` can serve a series of calls to FFT functions as long as `n` is large enough for each function called and as long as the weights array is preserved.

For example, before making this call to `vDSP_fft_zrip()` for 256 data points,

```
vDSP_fft_zrip( setup, c, 1, 8, FFT_FORWARD); /* 256 (2**8) points  */
```

the `vDSP_create_fftsetup()` call can be

```
FFTsetup setup; setup = vDSP_create_fftsetup ( 8, 0 );
```

or the call can specify a larger value for n. For example, if the same application also calls `vDSP_fft_zrip()` to operate on 2048 data points,

```
vDSP_fft_zrip( setup, c, 1, 11, FFT_FORWARD); /* 2048 (2**11) points  */
```

an initial call to `vDSP_create_fftsetup()` can be made with these values:

```
FFTsetup setup; vDSP_create_fftsetup( 11, 0 ); /* supports up to 2048 (2**11)
points  */
```

Reusing a weights array for similarly sized FFTs is economical. However, using a weights array built for an FFT that processes a large number of elements can degrade performance for an FFT that processes a smaller number of elements.

For a fully developed example of the use of `create_fftsetup` and FFT functions, see Chapter A, "Sample Code." (page 381)

Data Packing for Real FFTs

The discrete Fourier transform functions in the vDSP library provide a unique case in data formatting to conserve memory. Real-to-complex discrete Fourier transforms write their output data in special packed formats so that the complex output requires no more memory than the real input.

Packing and Transformation Functions

Applications that call the real FFT may have to use two transformation functions, one before the FFT call and one after. This is required if the input array is not in the even-odd split configuration.

A real array `A = {A[0],...,A[n]}` has to be transformed into an even-odd array `AEvenOdd = {A[0],A[2],...,A[n-1],A[1],A[3],...A[n]}` by means of the call `vDSP_ctoz()`.

The result of the real FFT of `AEvenOdd` of dimension `n` is a complex array of dimension `2n`, with a very special format:

```
{[DC,0],C[1],C[2],...,C[n/2],[NY,0],Cc[n/2],...,Cc[2],Cc[1]}
```

where

- Values `DC` and `NY` are the DC and Nyquist components (real valued)
- Array `C` is complex in a split representation,
- Array `Cc` is the complex conjugate of `C` in a split representation.

For a real array `A` of size n , the results, which are complex, require 2n spaces. In order to fit the 2n-sized result into an n size input array, and since the complex conjugates are duplicate information, the real FFT produces its results as follows:

```
{[DC,NY],C[1],C[2],...,C[n/2]}.
```

The sections "Packing for One-Dimensional Arrays" (page 21) and "Packing for Two-Dimensional Arrays" (page 22) describe the packing formats in more detail.

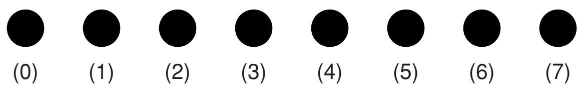
Packing for One-Dimensional Arrays

Due to its inherent symmetry, the forward Fourier transform of n floating-point inputs from the time domain to the frequency domain produces n/2 + 1 unique complex outputs. Because data point n/2 – i equals the complex conjugate of point n/2 + i, only the first half of the frequency data, up to and including point n/2 + 1, is sufficient to preserve the original information. In addition, the FFT data packing takes advantage of the fact that the imaginary parts of the first complex output (element zero) and of the last complex output (element n/2, the Nyquist frequency), are always zero.

This makes it possible to store the real part of the last output where the imaginary part of the first output would have been. The zero-valued imaginary parts of the first and last outputs are implied.

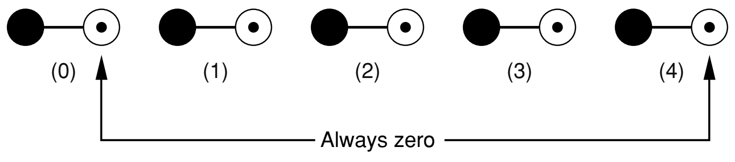
For example, consider this eight-point real vector as it exists prior to being transformed to the frequency domain (Figure 1-1 (page 21)).

Figure 1-1 Eight point real vector



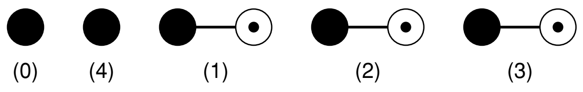
Transforming these eight real points to the frequency domain results in five complex values (Figure 1-2 (page 21)).

Figure 1-2 Results of an eight-point real-to-complex DFT



The five complex values are packed in the output vector shown in Figure 1-3 (page 21).

Figure 1-3 Packed format of an eight-point real-to-complex DFT



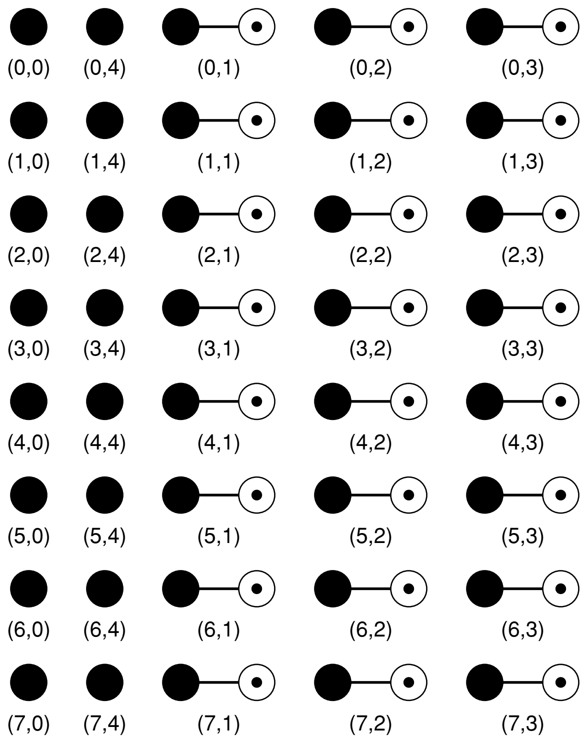
Notice that the real part of output four is stored where the imaginary part of output zero would have been stored with no packing. The imaginary parts of the first output and the last output, always zero, are implied.

Packing for Two-Dimensional Arrays

Two-dimensional real-to-complex DFTs are packed in a similar way. When processing two-dimensional real inputs, the DFTs first transform each row and then transform down columns. As each row is transformed, it is packed as previously described for one-dimensional transforms.

For example, to transform an 8-by-8 real image to the frequency domain, first each row is transformed and packed as shown in Figure 1-4 (page 22).

Figure 1-4 Interim format of an 8-by-8 real-to-complex DFT

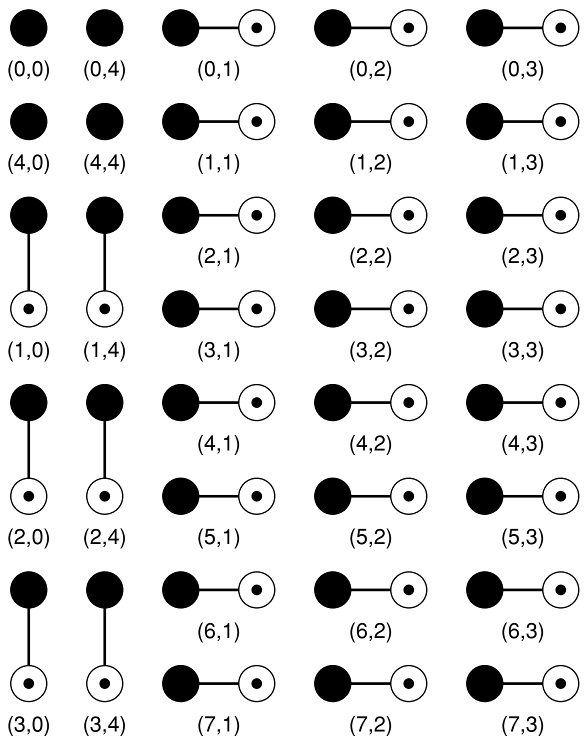


Notice that each row is packed like the one-dimensional transform in the earlier example. Packing yields 16 real values and 24 complex values in a total of 64 `sizeof(float)` memory locations, the size of the original matrix.

After each row is processed and packed, the DFT then transforms each column. Since the first pass leaves real values in columns zero and one, these two columns are transformed from real to complex using the same packing method vertically that was used horizontally on each row. Thus, transforming these 16 real values yields six complex points and four real points, occupying the 16 memory locations in columns zero and one.

The remaining three columns of complex values are transformed complex-to-complex, yielding 24 complex points as output, and requiring 48 memory locations to store. This brings the total storage requirement to 64 locations, the size of the original matrix. The completed 8-by-8 transform's format is shown in Figure 1-5 (page 23).

Figure 1-5 Packed format of an 8-by-8 real-to-complex 2D DFT



Scaling Fourier Transforms

To provide the best possible execution speeds, the vDSP library's functions don't always adhere strictly to textbook formulas for Fourier transforms, and must be scaled accordingly. The following sections specify the scaling for each type of Fourier transform implemented by the vDSP Library. The scaling factors are also stated explicitly in the formulas that accompany the function definitions in the reference chapter.

The scaling factors are summarized in Figure 1-6 (page 24), which shows the implemented values in terms of the mathematical values. The sections that follow describe the scaling factors individually.

Figure 1-6 Summary of the scaling factors

One-Dimensional Transforms

Real forward transforms: $RF_{imp} = RF_{math} * 2$

Real inverse transforms: $RI_{imp} = RI_{math} * n$

Complex forward transforms: $CF_{imp} = CF_{math}$

Complex inverse transforms: $CI_{imp} = CI_{math} * n$

Two-Dimensional Transforms

Real forward transforms: $RF2D_{imp} = RF2D_{math} * 2$

Real inverse transforms: $RI2D_{imp} = RI2D_{math} * mn$

Complex forward transforms: $CF2D_{imp} = CF2D_{math}$

Complex inverse transforms: $CI2D_{imp} = CI2D_{math} * mn$

Real Fourier Transforms

Figure 1-7 (page 24) shows the mathematical formula for the one-dimensional forward Fourier transform.

Figure 1-7 1D forward Fourier transforms, general formula

$$C_m = \sum_{n=0}^{N-1} C_n e^{\left[\frac{-i2\pi}{N}\right]^{nm}}$$

The values of the Fourier coefficients returned by the real forward transform as implemented (RFimp) are equal to the mathematical values (RFmath) times 2:

$RF_{imp} = RF_{math} * 2$

Figure 1-8 (page 24) shows the mathematical formula for the inverse Fourier transform.

Figure 1-8 1D inverse Fourier transforms, general formula

$$C_m = \sum_{n=0}^{N-1} C_n e^{\left[\frac{i2\pi}{N}\right]^{nm}}$$

The values of the Fourier coefficients returned by the real inverse transform as implemented (RIimp) are equal to the mathematical values (RImath) times N:

$RI_{imp} = RI_{math} * N$

Complex Fourier Transforms

Figure 1-7 (page 24) shows the mathematical formula for the one-dimensional forward Fourier transform.

The values of the Fourier coefficients returned by the complex forward transform as implemented (CFimp) are equal to the mathematical values (CFmath):

$CF_{imp} = CF_{math}$

The values of the Fourier coefficients returned by the complex inverse transform as implemented (CIimp) are equal to the mathematical values (CImath) times N:

$CI_{imp} = CI_{math} * N$

Real 2-Dimensional Fourier Transforms

Figure 1-9 (page 25) shows the mathematical formula for the 2-dimensional forward Fourier transform.

Figure 1-9 2D forward Fourier transforms, general formula

$$C_{nm} = \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} C_{pq} e^{\left[\frac{-i2\pi}{N}\right]^{pn}} e^{\left[\frac{-i2\pi}{M}\right]^{qm}}$$

The values of the Fourier coefficients returned by the 2-dimensional real forward transform as implemented (RF2Dimp) are equal to the mathematical values (R2DFmath) times 2:

$RF2D_{imp} = RF2D_{math} * 2$

Figure 1-10 (page 25) shows the mathematical formula for the 2-dimensional inverse Fourier transform.

Figure 1-10 2D inverse Fourier transforms, general formula

$$C_{nm} = \frac{1}{MN} \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} C_{pq} e^{\left[\frac{i2\pi}{N}\right]^{pn}} e^{\left[\frac{i2\pi}{M}\right]^{qm}}$$

The values of the Fourier coefficients returned by the 2-dimensional real inverse transform as implemented (RF2Dimp) are equal to the mathematical values (R2DFmath) times MN:

$RI2D_{imp} = RI2D_{math} * MN$

Complex 2-Dimensional Fourier Transforms

Figure 1-9 (page 25) shows the mathematical formula for the 2-dimensional forward Fourier transform.

The values of the Fourier coefficients returned by the 2-dimensional complex forward transform as implemented (CF2Dimp) are equal to the mathematical values (CF2Dmath):

$CF2D_{imp} = CF2D_{math}$

Figure 1-10 (page 25) shows the mathematical formula for the 2-dimensional inverse Fourier transform.

The values of the Fourier coefficients returned by the 2-dimensional complex inverse transform as implemented (CI2Dimp) are equal to the mathematical values (CI2Dmath) TIMES MN:

$CI2D_{imp} = CI2D_{math} * mn$

Scaling Example

Using complex two-dimensional transforms as an example, when transforming data from the time domain to the frequency domain with `vDSP_fft_zop`, no scaling factor is introduced. Transforming the image from the frequency domain to the time domain with `vDSP_fft_zop()`, however, introduces a factor of N, where N is the vector length.

In the following example, a vector is transformed to the frequency domain, an inverse transform reconstructs it in the time domain, and then it is scaled by 1/N to recover the original input (perhaps altered by machine rounding errors).

Listing 1-1 Scaling example

```
    ~
    ~
scale = 1.0/((float) FFT_LENGTH) ;
    ~
    ~
/* to the frequency domain */
vDSP_fft_zop(   setup,
           &A,
           1,
           &B,
           1,
           FFT_LENGTH_LOG2,
           FFT_FORWARD,
           ) ;

/* back to the time domain */
vDSP_fft_zop(   setup,
           &B,
           1,
           &A,
           1,
           FFT_LENGTH_LOG2,
           FFT_INVERSE,
           ) ;

/*scale the result */
vDSP_vsmul(     A.realp,
           1,
           &scale,
           A.realp,
           1,
           FFT_LENGTH
           ) ;

vDSP_vsmul(     A.imagp,
           1,
           &scale,
           A.imagp,
           1,
           FFT_LENGTH
           ) ;
    ~
    ~
```

Function Type Reference

This chapter presents a list of vDSP functions listed by function type. It is provided to assist users not familiar with the vDSP library to find the name of a function for implementing a particular algorithm.

The functions in this chapter are classified into eight different functional categories as shown in Table 2-1 (page 29).

Table 2-1 Categories of functions

Category	Types of Functions
Table 2-2 (page 30), Complex Vector Conversion (page 30)	Converting a complex vector from interleaved to split format, and the reverse.
Table 2-3 (page 30), Vector Scalar Arithmetic Functions (page 30)	Combining a scalar with a vector, such as multiplying a vector by a scalar.
Table 2-4 (page 30), Single Vector Functions (page 30)	Transforming each element of a vector.
Table 2-5 (page 32), Vector-to-Scalar Operations (page 32)	Producing a scalar result from a combination of two equal-length vectors: vector dot-products.
Table 2-6 (page 33),Vector-to-Vector Arithmetic Operations (page 33)	Arithmetically combining two vectors of equal length to produce a single-vector result. Examples are vector addition and element-by-element vector multiplication.
Table 2-7 (page 35), One-Dimensional FFTs (page 35)	Performing one-dimensional FFTs.
Table 2-8 (page 36), Two-Dimensional FFTs (page 36)	Performing two-dimensional FFTs.
Table 2-9 (page 36), Correlation and Convolution (page 36)	Performing correlation and convolution operations.
Table 2-10 (page 37), Miscellaneous Functions (page 37)	Hamming and Hanning windows; submatrix copy

Complex Vector Conversion

Table 2-2 Complex vector conversion

Routine	Description
vDSP_ctoz (page 46), vDSP_ctozD (page 46)	Interleaved-complex copy to split
vDSP_ztoc (page 353), vDSP_ztocD (page 354)	Split copy to interleaved-complex

Vector Scalar Arithmetic Functions

Table 2-3 Vector scalar arithmetic functions

Routine	Description
vDSP_vsadd (page 282), vDSP_vsaddD (page 283), vDSP_vsaddi (page 284)	Vector scalar add
vDSP_vsdiv (page 292), vDSP_vsdivD (page 293), vDSP_vsdivi (page 294), vDSP_zvdiv (page 363), vDSP_zvdivD (page 363)	Vector scalar divide
vDSP_svdiv (page 181), vDSP_svdivD (page 182)	Divide scalar by vector
vDSP_vsma (page 297), vDSP_vsmaD (page 298), vDSP_zvsma (page 374), vDSP_zvsmaD (page 375)	Vector scalar multiply and vector add
vDSP_vsmsaD (page 300), vDSP_vsmsaD (page 300)	Vector scalar multiply and scalar add
vDSP_vsmsb (page 301), vDSP_vsmsbD (page 303)	Vector scalar multiply and scalar subtract
vDSP_vsmul (page 304), vDSP_vsmulD (page 304), vDSP_zvzsml (page 378), vDSP_zvzsmlD (page 378)	Vector scalar multiply

Single Vector Functions

Table 2-4 Single vector functions

Routine	Description
vDSP_vabs (page 192), vDSP_vabsD (page 192), vDSP_vabsi (page 193), vDSP_zvabs (page 356), vDSP_zvabsD (page 356)	Vector absolute value

Routine	Description
vDSP_vnabs (page 267), vDSP_vnabsD (page 268)	Vector negative absolute value
vDSP_vneg (page 269), vDSP_vnegD (page 269), vDSP_zvneg (page 371), vDSP_zvnegD (page 372)	Vector negate
vDSP_vfill (page 223), vDSP_vfillD (page 223), vDSP_vfilli (page 224), vDSP_zvfill (page 364), vDSP_zvfillD (page 365)	Vector fill
vDSP_vramp (page 278), vDSP_vrampD (page 278)	Generate ramped vector
vDSP_vsq (page 308), vDSP_vsqD (page 309).	Vector square
vDSP_vssq (page 309), vDSP_vssqD (page 310)	Vector signed square
vDSP_vclr (page 208), vDSP_vclrD (page 208)	Clear vector
vDSP_vgen (page 230), vDSP_vgenD (page 231)	Generate tapered vector
vDSP_vgenp (page 232), vDSP_vgenpD (page 233)	Vector generate by extrapolation and interpolation
vDSP_polar (page 175), vDSP_polarD (page 176), vDSP_rect (page 177), vDSP_rectD (page 178)	Polar/rectangular coordinate conversions
vDSP_vdbcon (page 211), vDSP_vdbconD (page 212)	Vector convert power or amplitude to decibels
vDSP_vfrac (page 225), vDSP_vfracD (page 226)	Vector truncate to fraction
vDSP_zvconj (page 361), vDSP_zvconjD (page 362)	Vector complex conjugate
vDSP_zvmags (page 366), vDSP_zvmagsD (page 366)	Complex vector magnitudes squared
vDSP_zvmgsa (page 367), vDSP_zvmgsaD (page 368)	Complex vector magnitudes square and add
vDSP_zvphas (page 373), vDSP_zvphasD (page 374)	Complex vector phase
vDSP_vclip (page 203), vDSP_vclipD (page 207)	Vector clip
vDSP_vclipc (page 204), vDSP_vclipcD (page 205)	Vector clip and count
vDSP_viclip (page 235), vDSP_viclipD (page 236)	Vector inverted clip
vDSP_vlim (page 241), vDSP_vlimD (page 242)	Vector test limit
vDSP_vthr (page 318), vDSP_vthrD (page 319)	Vector threshold
vDSP_vthres (page 319), vDSP_vthresD (page 320)	Vector threshold with zero fill
vDSP_vthrsc (page 321), vDSP_vthrscD (page 322)	Vector threshold with signed constant
vDSP_vcmprs (page 209), vDSP_vcmprsD (page 210)	Vector compress

Routine	Description
vDSP_vgathr (page 226), vDSP_vgathrD (page 229), vDSP_vgathra (page 227), vDSP_vgathraD (page 228)	Vector gather
vDSP_vindex (page 237), vDSP_vindexD (page 238)	Vector index
vDSP_vrvrs (page 281), vDSP_vrvrsD (page 282)	Vector reverse order, in place
vDSP_zvmov (page 369), vDSP_zvmovD (page 370)	Complex vector move
vDSP_nzcros (page 173), vDSP_nzcrosD (page 174)	Count zero crossings
vDSP_vavlin (page 201), vDSP_vavlinD (page 202)	Vector linear average
vDSP_vlint (page 243), vDSP_vlintD (page 244)	Vector linear interpolation
vDSP_vrsum (page 279), vDSP_vrsumD (page 280)	Vector running sum
vDSP_vsimps (page 295), vDSP_vsimpsD (page 296)	Simpson integration
vDSP_vsort (page 305), vDSP_vsortD (page 305), vDSP_vsorti (page 306), vDSP_vsortiD (page 307)	Vector in-place sort
vDSP_vswsum (page 313), vDSP_vswsumD (page 314)	Vector sliding window sum
vDSP_vtabi (page 315), vDSP_vtabiD (page 316)	Vector interpolation, table lookup
vDSP_vtrapz (page 326), vDSP_vtrapzD (page 327)	Vector trapezoidal integration
vDSP_vdpsp (page 218), vDSP_vspdp (page 307)	Vector convert between double and single precision

Vector-to-Scalar Operations

Table 2-5 Vector-to-scalar operations

Routine	Description
vDSP_dotpr (page 51), vDSP_dotprD (page 52)	Vector dot product
vDSP_zdotpr (page 335), vDSP_zdotprD (page 336)	Vector dot product, complex split
vDSP_zidotpr (page 336), vDSP_zidotprD (page 337)	Vector inner dot product, complex split
vDSP_zrdotpr (page 348), vDSP_zrdotprD (page 348)	Vector dot product, complex split and real
vDSP_maxv (page 151), vDSP_maxvD (page 152), vDSP_maxvi (page 153), vDSP_maxviD (page 154)	Vector maximum value

Routine	Description
vDSP_maxmgv (page 148), vDSP_maxmgvD (page 149), vDSP_maxmgvi (page 150), vDSP_maxmgviD (page 151)	Vector maximum magnitude
vDSP_minv (page 163), vDSP_minvD (page 163), vDSP_minvi (page 164), vDSP_minviD (page 165)	Vector minimum value
vDSP_minmgv (page 159), vDSP_minmgvD (page 160), vDSP_minmgvi (page 161), vDSP_minmgviD (page 162)	Vector minimum magnitude
vDSP_meanv (page 156), vDSP_meanvD (page 157)	Vector mean value
vDSP_meamgv (page 155), vDSP_meamgvD (page 155)	Vector mean magnitude
vDSP_measqv (page 158), vDSP_measqvD (page 159)	Vector mean square value
vDSP_mvessq (page 171), vDSP_mvessqD (page 172)	Vector mean of signed squares
vDSP_rmsqv (page 179), vDSP_rmsqvD (page 180)	Vector root mean square
vDSP_sve (page 183), vDSP_sveD (page 183)	Vector sum
vDSP_svemg (page 184), vDSP_svemgD (page 185)	Vector sum of magnitudes
vDSP_svesq (page 186), vDSP_svesqD (page 187)	Vector sum of squares
vDSP_svs (page 187), vDSP_svsD (page 188)	Vector sum of signed squares

Vector-to-Vector Arithmetic Operations

Table 2-6 Vector-to-vector arithmetic operations

Routine	Description
vDSP_veqvi (page 222)	Vector equivalence, 32-bit logical
vDSP_vadd (page 194), vDSP_vaddD (page 195)	Vector add
vDSP_vsub (page 310), vDSP_vsubD (page 311)	Vector subtract
vDSP_vam (page 195), vDSP_vamD (page 196)	Vector add and multiply
vDSP_vsbm (page 285), vDSP_vsbmD (page 286)	Vector subtract and multiply
vDSP_vaam (page 189), vDSP_vaamD (page 190)	Vector add, add, and multiply
vDSP_vsbsbm (page 288), vDSP_vsbsbmD (page 289)	Vector subtract, subtract, and multiply
vDSP_vasbm (page 196), vDSP_vasbmD (page 198)	Vector add, subtract, and multiply
vDSP_vasm (page 199), vDSP_vasmD (page 200)	Vector add and scalar multiply

Routine	Description
vDSP_vsbsm (page 290), vDSP_vsbsmD (page 291)	Vector subtract and scalar multiply
vDSP_vmsa (page 261), vDSP_vmsaD (page 262)	Vector multiply and scalar add
vDSP_vdiv (page 215), vDSP_vdivD (page 216), vDSP_vdivi (page 217)	Vector divide
vDSP_vmul (page 266), vDSP_vmulD (page 266)	Vector multiply
vDSP_vma (page 246), vDSP_vmaD (page 247)	Vector multiply and add
vDSP_vmsb (page 264), vDSP_vmsbD (page 265)	Vector multiply and subtract
vDSP_vmma (page 256), vDSP_vmmaD (page 257)	Vector multiply, multiply, and add
vDSP_vmmsb (page 259), vDSP_vmmsbD (page 260)	Vector multiply, multiply, and subtract
vDSP_vmax (page 248), vDSP_vmaxD (page 249)	Vector maximum corresponding values
vDSP_vmaxmg (page 250), vDSP_vmaxmgD (page 251)	Vector maximum corresponding magnitudes
vDSP_vmin (page 252), vDSP_vminD (page 253)	Vector minimum corresponding values
vDSP_vminmg (page 254), vDSP_vminmgD (page 255)	Vector minimum corresponding magnitudes
vDSP_vdist (page 213), vDSP_vdistD (page 214)	Vector distance
vDSP_vintb (page 239), vDSP_vintbD (page 240)	Vector interpolation between vectors
vDSP_vqint (page 275), vDSP_vqintD (page 276)	Vector quadratic interpolation
vDSP_vpoly (page 270), vDSP_vpolyD (page 271)	Vector polynomial
vDSP_vpythg (page 272), vDSP_vpythgD (page 274)	Vector pythagoras
vDSP_venvlp (page 219), vDSP_venvlpD (page 220)	Vector envelope
vDSP_vswap (page 311), vDSP_vswapD (page 312)	Vector swap
vDSP_vtmerg (page 323), vDSP_vtmergD (page 324)	Vector tapered merge
vDSP_zaspec (page 331), vDSP_zaspecD (page 331)	Accumulating autospectrum
vDSP_zcspec (page 334), vDSP_zcspecD (page 334)	Accumulating cross-spectrum
vDSP_zcoher (page 331), vDSP_zcoherD (page 332)	Coherence function
vDSP_zrvdiv (page 350), vDSP_zrvdivD (page 350)	Vector complex split multiply by real vector
vDSP_zrvmul (page 351), vDSP_zrvmulD (page 351)	Vector complex split divide by real vector

Routine	Description
vDSP_zrvsub (page 352), vDSP_zrvsubD (page 353)	Vector subtract real from complex split vector
vDSP_zrvadd (page 349), vDSP_zrvaddD (page 349)	Vector complex split, add to real vector
vDSP_zvadd (page 357), vDSP_zvaddD (page 358)	Vector add, complex split
vDSP_zvcmul (page 359), vDSP_zvmulD (page 371)	Vector multiply, complex split
vDSP_zvsub (page 376), vDSP_zvsubD (page 377)	Vector subtract, complex split
vDSP_zvcma (page 358), vDSP_zvcmaD (page 359)	Vector conjugate multiply and add complex split
vDSP_ztrans (page 354), vDSP_ztransD (page 355)	Complex transfer function
vDSP_vgenp (page 232), vDSP_vgenpD (page 233)	Generate vector by extrapolation and interpolation
vDSP_deq22 (page 47), vDSP_deq22D (page 48)	Difference equation, 2 poles, 2 zeros

One-Dimensional FFTs

Table 2-7 1-D FFTs

Routine	Description
vDSP_create_fftsetup (page 44), vDSP_create_fftsetupD (page 45), vDSP_destroy_fftsetup (page 50), vDSP_destroy_fftsetupD (page 51)	Create complex exponentials for Fourier transforms. See Chapter 2, "Using Fourier Transforms."
vDSP_fft_zip (page 120), vDSP_fft_zipD (page 121) , vDSP_fft_zipt (page 123), vDSP_fft_ziptD (page 124)	In-place split complex Fourier transforms
vDSP_fft_zop (page 125), vDSP_fft_zopD (page 127), vDSP_fft_zopt (page 128), vDSP_fft_zoptD (page 130)	Out-of-place split complex Fourier transforms
vDSP_fft_zrip (page 131), vDSP_fft_zripD (page 132), vDSP_fft_zript (page 134),vDSP_fft_zriptD (page 135)	In-place split real Fourier transforms
vDSP_fft_zrop (page 137), vDSP_fft_zropD (page 138), vDSP_fft_zropt (page 140), vDSP_fft_zroptD (page 141)	Out-of-place split real Fourier transforms

Two-Dimensional FFTs

Table 2-8 2-D FFTs

Routine	Description
vDSP_create_fftsetup (page 44), vDSP_create_fftsetupD (page 45), vDSP_destroy_fftsetup (page 50), vDSP_destroy_fftsetupD (page 51)	Create complex exponentials for Fourier transforms. See Chapter 2, "Using Fourier Transforms."
vDSP_fft2d_zip (page 56), vDSP_fft2d_zipD (page 57), vDSP_fft2d_zipt (page 59), vDSP_fft2d_ziptD (page 61)	In-place 2-d split complex Fourier transforms
vDSP_fft2d_zop (page 63), vDSP_fft2d_zopD (page 65), vDSP_fft2d_zopt (page 66), vDSP_fft2d_zoptD (page 68)	Out-of-place, 2-d split complex Fourier transforms
vDSP_fft2d_zrip (page 70), vDSP_fft2d_zripD (page 72), vDSP_fft2d_zript (page 74), vDSP_fft2d_zriptD (page 77)	In-place 2-d split real Fourier transforms
vDSP_fft2d_zrop (page 78), vDSP_fft2d_zropD (page 80), vDSP_fft2d_zropt (page 82), vDSP_fft2d_zroptD (page 85)	Out-of-place 2-d split real Fourier transforms

Correlation and Convolution

Table 2-9 Correlation and convolution

Routine	Description
vDSP_acorD (page 39)	Autocorrelation with automatic selection of domain
vDSP_acorfD (page 40)	Frequency-domain autocorrelation
vDSP_acortD (page 40)	Time-domain autocorrelation
vDSP_blkman_window (page 41), vDSP_convD (page 43)	Convolution and correlation
vDSP_zcoher (page 331), vDSP_zconvD (page 333)	Convolution and correlation, split complex
vDSP_wiener (page 328), vDSP_wienerD (page 329)	Wiener-Levinson general convolution
vDSP_desamp (page 49), vDSP_desampD (page 50)	Convolution with decimation
vDSP_zrdesamp (page 346), vDSP_zrdesampD (page 347)	Complex/real downsample with anti-aliasing

Miscellaneous Functions

Table 2-10 Miscellaneous functions

Routine	Description
vDSP_hamm_window (page 143), vDSP_hamm_windowD (page 144)	Create a Hamming window
vDSP_hann_window (page 144), vDSP_hann_windowD (page 145)	Create a Hanning window
vDSP_mmov (page 166), vDSP_mmovD (page 167)	The contents of a submatrix are copied to another submatrix

vDSP Functions

vDSP_acorD

Autocorrelation with automatic selection of domain.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_acorD (double * A,
    double * C,
    int N,
    int M);
```

Parameters

A

Double-precision real input vector

C

Double-precision real output vector

N

Output count (lags)

N

Input count from *A*

Discussion

Performs the operation

$$C_m = \sum_{p=0}^{N-m-1} A_{m+n} \cdot A_n \qquad m = \{0, M\text{-}1\}$$

Performs linear autocorrelation of vector *A*. Either time- or frequency-domain correlation is selected automatically, evaluating *M* and *N* to choose the domain that yields the best execution time. Frequency-domain correlation does not preserve vector *A*, so calling functions must assume that vector *A* was modified by this function.

Note that vDSP_acorfD explicit does frequency-domain correlation, and vDSP_acortD explicitly does time-domain correlation.

Availability
Available in Mac OS X v10.4 and later.

vDSP_acorfD

Frequency-domain autocorrelation.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_acorfD (double * A,
    double * C,
    int N,
    int M);
```

Parameters

A

Double-precision real input vector

C

Double-precision real output vector

N

Output count (lags)

N

Input count from *A*

Discussion
Performs the operation

$$C_m = \sum_{p=0}^{N-m-1} A_{m+n} \cdot A_n \qquad \text{m = \{0, M-1\}}$$

Performs linear frequency-domain autocorrelation of vector *A*. Frequency-domain correlation does not preserve vector *A*, so calling functions must assume that vector *A* was modified by this function.

Note that vDSP_acorD automatically selects the domain for best execution time, and vDSP_acortD explicitly does time-domain correlation.

Availability
Available in Mac OS X v10.4 and later.

vDSP_acortD

Time-domain autocorrelation.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_acortD (double * A,
    double * C,
    int N,
    int M);
```

Parameters

A

Double-precision real input vector

C

Double-precision real output vector

N

Output count (lags)

N

Input count from *A*

Discussion

Performs the operation

$$C_m = \sum_{p=0}^{N-m-1} A_{m+n} \cdot A_n \qquad m = \{0, M-1\}$$

Performs linear time-domain autocorrelation of vector *A*.

Note that vDSP_acorD automatically selects the domain for best execution time, and vDSP_acorfD explicitly does frequency-domain correlation.

Availability

Available in Mac OS X v10.4 and later.

vDSP_blkman_window

Creates a Blackman window.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_blkman_window (float * C,
    unsigned int N,
    int FLAG);
```

Discussion

$$C_n = 0.42 - 0.5\cos\frac{2\pi n}{N} + 0.08\cos\frac{4\pi n}{N} \qquad n = \{0, N-1\}$$

`vDSP_blkman_window` creates a single-precision Blackman window function *C*, which can be multiplied by a vector using `vDSP_vmul` . Specify the `vDSP_HALF_WINDOW` flag to create only the first (n+1)/2 points, or 0 (zero) for full size window.

See also `vDSP_vmul`.

Important: The constant `vDSP_HALF_WINDOW` is not declared in the vDSP header file. For the initial release in Mac OS X v10.4, declare it in a separate header with the value 1.

Availability
Available in Mac OS X v10.4 and later.

vDSP_blkman_windowD

Creates a Blackman window.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_blkman_windowD (double * C,
    unsigned int N,
    int FLAG);
```

Discussion

$$C_n = 0.42 - 0.5\cos\frac{2\pi n}{N} + 0.08\cos\frac{4\pi n}{N} \qquad \text{n = \{0, N-1\}}$$

`vDSP_blkman_window` creates a double-precision Blackman window function `C`, which can be multiplied by a vector using `vDSP_vmulD` . Specify the `vDSP_HALF_WINDOW` flag to create only the first (n+1)/2 points, or 0 (zero) for full size window.

See also `vDSP_vmulD`.

Important: The constant `vDSP_HALF_WINDOW` is not declared in the vDSP header file. For the initial release in Mac OS X v10.4, declare it in a separate header with the value 1.

Availability
Available in Mac OS X v10.4 and later.

vDSP_conv

Performs either correlation or convolution on two vectors.

```
void
vDSP_conv (const float signal[],
    SInt32signalStride,
    const floatfilter[],
    SInt32 filterStride,
    float result[],
    SInt32 resultStride,
    SInt32 lenResult,
    SInt32 lenFilter);
```

Discussion

$$C_{nK} = \sum_{p=0}^{P-1} A_{(n+p)I} B_{pJ} \qquad \text{n = \{0, N-1\}}$$

If filterStride is positive, `vDSP_conv` performs correlation. If filterStride is negative, it performs convolution and *filter must point to the last vector element. The function can run in place, but result cannot be in place with filter.

The value of `lenFilter` must be less than or equal to 2044. The length of vector `signal` must satisfy two criteria: it must be

- equal to or greater than 12
- equal to or greater than the sum of `N-1` plus the nearest multiple of 4 that is equal to or greater than the value of `lenFilter`.

Criteria to invoke vectorized code:

- The vectors `signal` and `result` must be relatively aligned.
- The value of `lenFilter` must be between 4 and 256, inclusive.
- The value of `lenResult` must be greater than 36.
- The values of `signalStride` and `resultStride` must be 1.
- The value of `filterStride` must be either 1 or –1.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_convD

Performs either correlation or convolution on two vectors.

```
void
vDSP_convD (const double signal[],
    SInt32 signalStride,
    const double filter[],
    SInt32 filterStride,
    double result[],
    SInt32 resultStride,
    SInt32 lenResult,
    SInt32 lenFilter);
```

Discussion

$$C_{nK} = \sum_{p=0}^{P-1} A_{(n+p)I} B_{pJ} \qquad \text{n = \{0, N-1\}}$$

If filterStride is positive, `vDSP_convD` performs correlation. If *`filterStride`* is negative, it performs convolution and **`filter`* must point to the last vector element. The function can run in place, but result cannot be in place with filter.

The value of `lenFilter` must be less than or equal to 2044. The length of vector `signal` must satisfy two criteria: it must be

- equal to or greater than 12
- equal to or greater than the sum of `N-1` plus the nearest multiple of 4 that is equal to or greater than the value of `lenFilter`.

Criteria to invoke vectorized code:

No Altivec support for double precision. This function always invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_create_fftsetup

Builds a data structure that contains precalculated data for use by Fourier Transform functions.

```
FFTSetup
vDSP_create_fftsetup (UInt32 log2n,
    FFTRadix radix);
```

Discussion
A single weights array can serve multiple FFT functions as long as the array is large enough for each function called and as long as the weights array is preserved. Functions that use the weights array automatically adjust their strides through the array when the array is larger than required. Using a shared weights array for similarly sized FFTs conserves memory. However, if the size disparity is great, using a large weights array for an FFT that processes a small number of elements can degrade performance.

Parameter `log2n` is a base 2 exponent that represents the number of divisions of the complex unit circle and thus specifies the largest power of two that can be processed by a subsequent frequency-domain function. Parameter `log2n` must equal or exceed the largest power of 2 that any subsequent function processes using the weights array.

Parameter *`radix`* specifies radix options. Radix 2, radix 3, and radix 5 functions are supported.

`vDSP_create_fftsetup` builds a weights array and returns an address (type `FFTSetup`) that points to a data structure that contains the weights array to any functions that subsequently use the array. The address is subsequently passed as an argument to FFT functions.

If zero is returned, the `vDSP_create_fftsetup` function failed to find enough storage to allocate the weights array.

When forecasting memory requirements for building a weights array, keep in mind that this function generates complex numbers.

See also Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_create_fftsetupD

Builds a data structure that contains precalculated data for use by Fourier Transform functions.

```
FFTSetupD
create_fftsetupD (UInt32 log2n,
    FFTRadix radix);
```

Discussion
A single weights array can serve multiple FFT functions as long as the array is large enough for each function called and as long as the weights array is preserved. Functions that use the weights array automatically adjust their strides through the array when the array is larger than required. Using a shared weights array for similarly sized FFTs conserves memory. However, if the size disparity is great, using a large weights array for an FFT that processes a small number of elements can degrade performance.

Parameter `log2n` is a base 2 exponent that represents the number of divisions of the complex unit circle and thus specifies the largest power of two that can be processed by a subsequent frequency-domain function. Parameter `log2n` must equal or exceed the largest power of 2 that any subsequent function processes using the weights array.

Parameter *`radix`* specifies radix options. Radix 2, radix 3, and radix 5 functions are supported.

`vDSP_create_fftsetupD` builds a weights array and returns an address (type `FFTSetupD`) that points to a data structure that contains the weights array to any functions that subsequently use the array. The address is subsequently passed as an argument to FFT functions.

If zero is returned, the `vDSP_create_fftsetupD` function failed to find enough storage to allocate the weights array.

When forecasting memory requirements for building a weights array, keep in mind that this function generates complex numbers.

See also Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_ctoz

Copies the contents of an interleaved complex vector C to a split complex vector Z.

```
void
vDSP_ctoz (const DSPComplex C[],
    SInt32 cStride,
    DSPSplitComplex * Z,
    SInt32 zStride,
    SInt32 size);
```

Discussion
Performs the operation

$A_{nI} = Re(C_{nK}) \quad ; \quad A_{nK+1} = Im(C_{n;K}) \qquad n = \{0, N\text{-}1\}$

cStride is an address stride through cStride. zStride is an address stride through Z. The value of cStride must be a multiple of 2.

For best performance, `C.realp`, `C.imagp`, `Z.realp`, and `Z.imagp` should be 16-byte aligned.

Criteria to invoke vectorized code:

- The value of `size` must be greater than 3.
- The value of cStride must be 2.
- The value of `zStride` must be 1.
- Vectors `Z.realp` and `Z.imagp` must be relatively aligned.
- Vector `C` must 8 bytes aligned if `Z.realp` and `Z.imagp` are 4 bytes aligned or `C` must be 16 bytes aligned if `Z.realp` and `Z.imagp` are at least 8 bytes aligned.

If any of these criteria is not satisfied, the function invokes scalar code.

See also functions"vDSP_ztoc" (page 353) and "vDSP_ztocD" (page 354).

Availability
Available in Mac OS X v10.4 and later.

vDSP_ctozD

Copies the contents of an interleaved complex vector C to a split complex vector Z.

```
void
vDSP_ctozD (const DSPDoubleComplex C[],
    SInt32 cStride,
    DSPDoubleSplitComplex * Z,
    SInt32 zStride,
    SInt32 size);
```

Discussion
This performs the operation

$$A_{nI} = Re(C_{nK}) \quad ; \quad A_{nK+1} = Im(C_{n;K}) \qquad n = \{0, N\text{-}1\}$$

cStride is an address stride through cStride. zStride is an address stride through Z. The value of cStride must be a multiple of 2.

For best performance, `C.realp`, `C.imagp`, `Z.realp`, and `Z.imagp` should be 16-byte aligned.

Criteria to invoke vectorized code:

No altivec support for double precision. This function always invokes scalar code.

See also functions "vDSP_ztoc" (page 353) and "vDSP_ztocD" (page 354).

Availability
Available in Mac OS X v10.4 and later.

vDSP_deq22

Difference equation, 2 poles, 2 zeros.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_deq22 (float * A,
    int I,
    float * B,
    float * C,
    int K,
    unsigned int N);
```

Parameters

A
    Single-precision real input vector; must have at least *N*+2 elements

I
    Stride for *A*

B
    5 single-precision inputs, filter coefficients

C
    Single-precision real output vector; must have at least*N*+2elements

K
    Stride for *C*

N

Number of new output elements to produce

Discussion

Performs two-pole two-zero recursive filtering on real input vector `A`. Since the computation is recursive, the first two elements in vector `C` must be initialized prior to calling `deq22x`. `deq22x` creates `N` new values for vector `C` beginning with its third element and requires `N`+2 input values from vector `A`. This function can only be done out of place.

$$C_{nk} = \sum_{p=0}^{2} A_{(n-p)i} B_p - \sum_{p=3}^{4} C_{(n-p+2)k} B_p \qquad n = \{2, N+1\}$$

Availability

Available in Mac OS X v10.4 and later.

vDSP_deq22D

Difference equation, 2 poles, 2 zeros.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_deq22D (double * A,
    int I,
    double * B,
    double * C,
    int K,
    unsigned int N);
```

Parameters

A

Double-precision real input vector; must have at least `N`+2 elements

I

Stride for `A`

B

5 double-precision inputs, filter coefficients

C

Double-precision real output vector; must have at least`N`+2elements

K

Stride for `C`

N

Number of new output elements to produce

Discussion

Performs two-pole two-zero recursive filtering on real input vector `A`. Since the computation is recursive, the first two elements in vector `C` must be initialized prior to calling `deq22x`. `deq22x` creates `N` new values for vector `C` beginning with its third element and requires `N`+2 input values from vector `A`. This function can only be done out of place.

$$C_{nk} = \sum_{p=0}^{2} A_{(n-p)i} B_p - \sum_{p=3}^{4} C_{(n-p+2)k} B_p \qquad n = \{2, N+1\}$$

Availability
Available in Mac OS X v10.4 and later.

vDSP_desamp

Convolution with decimation.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_desamp (float * A,
    int I,
    float * B,
    float * C,
    unsigned int N,
    unsigned int P);
```

Parameters

A
Single-precision real input vector, 8-byte aligned; length of *A* >= 12

I
Desampling factor

B
Double-precision input filter coefficients

C
Single-precision real output vector

N
Output count

M
Filter coefficient count

Discussion
Performs finite impulse response (FIR) filtering at selected positions of vector *A*. `desampx` can run in place, but *C* cannot be in place with *B*. Length of *A* must be >=(*N*-1)**I*+(nearest multiple of 4 >=*M*).

$$C_n = \sum_{p=0}^{P-1} A_{nI+p} B_p \qquad n = \{0, N-1\}$$

Availability
Available in Mac OS X v10.4 and later.

vDSP_desampD

Convolution with decimation.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_desampD (double * A,
    int I,
    double * B,
    double * C,
    unsigned int N,
    unsigned int P);
```

Parameters

A

Double-precision real input vector, 8-byte aligned; length of *A* >= 12

I

Desampling factor

B

Double-precision input filter coefficients

C

Double-precision real output vector

N

Output count

M

Filter coefficient count

Discussion

Performs finite impulse response (FIR) filtering at selected positions of vector *A*. `desampx` can run in place, but *C* cannot be in place with *B*. Length of *A* must be >=(*N*-1)**I*+(nearest multiple of 4 >=*M*).

$$C_n = \sum_{p=0}^{P-1} A_{nI+p} B_p \qquad \text{n = \{0, N-1\}}$$

Availability

Available in Mac OS X v10.4 and later.

vDSP_destroy_fftsetup

Frees an existing Fourier Transforms data structure.

```
void
vDSP_destroy_fftsetup (FFTSetup setup);
```

Discussion
`vDSP_destroy_fftsetup` frees an existing weights array. Any memory allocated for the array is released. Parameter `setup` identifies the weights array, and must point to a data structure previously created by `vDSP_create_fftsetup`. After the `vDSP_destroy_fftsetup` function returns, the structure is no longer valid and cannot be passed to any subsequent frequency-domain functions.

When forecasting memory requirements for building a weights array, keep in mind that this function generate complex numbers. See also Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_destroy_fftsetupD

Frees an existing Fourier Transforms data structure.

```
void
vDSP_destroy_fftsetupD (FFTSetupD setup);
```

Discussion
`vDSP_destroy_fftsetupD` frees an existing weights array. Any memory allocated for the array is released. Parameter `setup` identifies the weights array, and must point to a data structure previously created by `vDSP_create_fftsetupD`. After the `vDSP_destroy_fftsetupD` function returns, the structure is no longer valid and cannot be passed to any subsequent frequency-domain functions.

When forecasting memory requirements for building a weights array, keep in mind that this function generate complex numbers. See also Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_dotpr

Computes the dot or scalar product of vectors `A` and `B` and leaves the result in scalar `C`.

```
void
vDSP_dotpr (const A[],
    SInt32 I,
    const float B[],
    SInt32 J,
    float * C,
    UInt32 N);
```

Discussion
This performs the operation

$$C = \sum_{n=0}^{N-1} A_{nI} \cdot B_{nJ} \qquad n = \{0, N\text{-}1\}$$

Criteria to invoke vectorized code:

- The vectors A and B must be relatively aligned.
- The value of N must be equal to or greater than 20.
- The values of I and J must be 1.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_dotprD

Computes the dot or scalar product of vectors A and B and leaves the result in scalar C.

```
void
vDSP_dotprD (const A[],
    SInt32 I,
    const double B[],
    SInt32 J,
    double * C,
    UInt32 N);
```

Discussion
This performs the operation

$$C = \sum_{n=0}^{N-1} A_{nI} \cdot B_{nJ} \qquad n = \{0, N\text{-}1\}$$

Criteria to invoke vectorized code:

- The vectors A and B must be relatively aligned.
- The value of N must be equal to or greater than 20.
- The values of I and J must be 1.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_f3x3

Filters an image by performing a two-dimensional convolution with a 3x3 kernel on the input matrix A. The resulting image is placed in the output matrix C.

```
void
vDSP_f3x3 (float * A,
    SInt32 M,
 SInt32 * N,
 float * B,
 float * C);
```

Discussion
This performs the operation

$$C_{(m+1,n+1)} = \sum_{p=0}^{2} \sum_{q=0}^{2} A_{(m+p,n+q)} \bullet B_{(p,q)} \qquad m = \{0, M-1\} \text{ and } n = \{0, N-3\}$$

The function pads the perimeter of the output image with a border of zeros of width 1.

`B` is the 3x3 kernel. `M` and `N` are the number of rows and columns, respectively, of the two-dimensional input matrix `A`. `M` must be greater than or equal to 3. `N` must be even and greater than or equal to 4.

Criteria to invoke vectorized code:

- `A`, `B`, and `C` must be 16-byte aligned.
- `N` must be greater than or equal to 18.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_f3x3D

Filters an image by performing a two-dimensional convolution with a 3x3 kernel on the input matrix `A`. The resulting image is placed in the output matrix `C`.

```
void
vDSP_f3x3D (double * A,
    SInt32 M,
 SInt32 * N,
 double * B,
 double * C);
```

Discussion
This performs the operation

$$C_{(m+1,n+1)} = \sum_{p=0}^{2} \sum_{q=0}^{2} A_{(m+p,n+q)} \bullet B_{(p,q)} \qquad \text{m = \{0, M-1\} and n = \{0, N-3\}}$$

The function pads the perimeter of the output image with a border of zeros of width 1.

`B` is the 3x3 kernel. `M` and `N` are the number of rows and columns, respectively, of the two-dimensional input matrix `A`. `M` must be greater than or equal to 3. `N` must be even and greater than or equal to 4.

Criteria to invoke vectorized code:

- `A`, `B`, and `C` must be 16-byte aligned.
- `N` must be greater than or equal to 18.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_f5x5

Filters an image by performing a two-dimensional convolution with a 5x5 kernel on the input matrix `signal`. The resulting image is placed in the output matrix result.

```
void
vDSP_f5x5 (float * A,

 SInt32 M,

 SInt32 * N,

 float * B,

 float * C);
```

Discussion
This performs the operation

$$C_{(m+2,n+2)} = \sum_{p=0}^{4} \sum_{q=0}^{4} A_{(m+p,n+q)} \bullet B_{(p,q)} \qquad \text{m = \{0, M-5\} and n = \{0, N-5\}}$$

The function pads the perimeter of the output image with a border of zeros of width 2.

`B` is the 3x3 kernel. `M` and `N` are the number of rows and columns, respectively, of the two-dimensional input matrix `A`. `M` must be greater than or equal to 5. `N` must be even and greater than or equal to 6.

Criteria to invoke vectorized code:

- `A`, `B`, and `C` must be 16-byte aligned.

- `N` must be greater than or equal to 20.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_f5x5D

Filters an image by performing a two-dimensional convolution with a 5x5 kernel on the input matrix `signal`. The resulting image is placed in the output matrix result.

```
void
vDSP_f5x5D (double * A,

 SInt32 M,

 SInt32 * N,

 double * B,

 double * C);
```

Discussion
This performs the operation

$$C_{(m+2,n+2)} = \sum_{p=0}^{4} \sum_{q=0}^{4} A_{(m+p,n+q)} \bullet B_{(p,q)} \qquad \text{m = \{0, M-5\} and n = \{0, N-5\}}$$

The function pads the perimeter of the output image with a border of zeros of width 2.

`B` is the 3x3 kernel. *`M`* and *`N`* are the number of rows and columns, respectively, of the two-dimensional input matrix *`A`*. *`M`* must be greater than or equal to 5. *`N`* must be even and greater than or equal to 6.

Criteria to invoke vectorized code:

- `A`, `B`, and `C` must be 16-byte aligned.
- `N` must be greater than or equal to 20.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_fft2d_zip

Computes an in-place complex discrete Fourier transform of matrix represented by `signal`, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse).

```
void
vDSP_fft2d_zip (FFTSetup setup,
    DSPSplitComplex * signal,
    SInt32 rowStride,
    SInt32 colStride,
    UInt32 log2nInCol,
    UInt32 log2nInRow,
    FFTDirection direction);
```

Parameters

`setup`

Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the values supplied as parameters `log2nInCol` and `log2nInRow` of the transform function.

`rowStride`

Specifies a stride across each row of the matrix `signal`. Specifying 1 for `rowStride` processes every element across each row, specifying 2 processes every other element across each row, and so forth.

`colStride`

Specifies a column stride for the matrix, and should generally be allowed to default unless the matrix is a submatrix. Parameter `colStride` can be defaulted by specifying 0. The default column stride equals the row stride multiplied by the column count. Thus, if `rowStride` is 1 and `colStride` is 0, every element of the input /output matrix is processed. If `rowStride` is 2 and `colStride` is 0, every other element of each row is processed.

If not 0, parameter `colStride` represents the distance between each row of the matrix. If parameter `colStride` is 1024, for instance, complex element 512 of the matrix equates to element (1,0), element 1024 equates to element (2,0), and so forth.

`log2nInCol`

The base 2 exponent of the number of columns to process for each row. `log2nInCol` must be between 2 and 10, inclusive.

`log2nInRow`

The base 2 exponent of the number of rows to process. For example, to process 64 rows of 128 columns, specify 7 for parameter `log2nInCol` and 6 for parameter log2nInRow. `log2nInRow` must be between 2 and 10, inclusive. See also the FFT Limitations sections in the Target chapters of the Developer's Guide.

`direction`
: A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`
: Forward transform

`FFT_INVERSE`
: Inverse transform

Results are undefined for other values of `direction`.

Discussion

This performs the operation

If F = 1 $\quad C_{nm} = \mathrm{FDFT2D}(C_{nm}) \quad n = \{0, N\text{-}1\} \text{ and } m = \{0, M\text{-}1\}$

If F = -1 $\quad C_{nm} = \mathrm{IDFT2D}(C_{nm}) \bullet MN \quad n = \{0, N\text{-}1\} \text{ and } m = \{0, M\text{-}1\}$

$$\mathrm{FDFT2D}(X_{nm}) = \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} X_{pq} \bullet e^{(-j2\pi pn)/N} \bullet e^{(-j2\pi qm)/M}$$

$$\mathrm{IDFT2D}(X_{nm}) = \frac{1}{MN} \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} X_{pq} \bullet e^{(j2\pi pn)/N} \bullet e^{(j2\pi qm)/M}$$

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability

Available in Mac OS X v10.4 and later.

vDSP_fft2d_zipD

Computes an in-place complex discrete Fourier transform of matrix represented by `signal`, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse).

```
void
vDSP_fft2d_zipD (FFTSetupD setup,
    DSPSplitComplex * signal,
    SInt32 rowStride,
    SInt32 colStride,
    UInt32 log2nInCol,
    UInt32 log2nInRow,
    FFTDirection direction);
```

Parameters

`setup`

Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the values supplied as parameters `log2nInCol` and `log2nInRow` of the transform function.

`rowStride`

Specifies a stride across each row of the matrix `signal`. Specifying 1 for `rowStride` processes every element across each row, specifying 2 processes every other element across each row, and so forth.

`colStride`

Specifies a column stride for the matrix, and should generally be allowed to default unless the matrix is a submatrix. Parameter `colStride` can be defaulted by specifying 0. The default column stride equals the row stride multiplied by the column count. Thus, if `rowStride` is 1 and `colStride` is 0, every element of the input /output matrix is processed. If `rowStride` is 2 and `colStride` is 0, every other element of each row is processed.

If not 0, parameter `colStride` represents the distance between each row of the matrix. If parameter `colStride` is 1024, for instance, complex element 512 of the matrix equates to element (1,0), element 1024 equates to element (2,0), and so forth.

`log2nInCol`

The base 2 exponent of the number of columns to process for each row. `log2nInCol` must be between 2 and 10, inclusive.

`log2nInRow`

The base 2 exponent of the number of rows to process. For example, to process 64 rows of 128 columns, specify 7 for parameter `log2nInCol` and 6 for parameter log2nInRow. `log2nInRow` must be between 2 and 10, inclusive. See also the FFT Limitations sections in the Target chapters of the Developer's Guide.

`direction`

A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`

Forward transform

`FFT_INVERSE`

Inverse transform

Results are undefined for other values of `direction`.

Discussion

This performs the operation

If F = 1 $\quad C_{nm} = \mathrm{FDFT2D}(C_{nm})$ $\quad$ n = {0, N-1} and m = {0, M-1}

If F = -1 $\quad C_{nm} = \mathrm{IDFT2D}(C_{nm}) \bullet MN$ $\quad$ n = {0, N-1} and m = {0, M-1}

$$\mathrm{FDFT2D}(X_{nm}) = \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} X_{pq} \bullet e^{(-j2\pi pn)/N} \bullet e^{(-j2\pi qm)/M}$$

$$\mathrm{IDFT2D}(X_{nm}) = \frac{1}{MN} \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} X_{pq} \bullet e^{(j2\pi pn)/N} \bullet e^{(j2\pi qm)/M}$$

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_fft2d_zipt

Computes an in-place complex discrete Fourier transform of matrix represented by `signal`, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse).

```
void
vDSP_fft2d_zipt (FFTSetup setup,
    DSPSplitComplex * signal,
    SInt32 rowStride,
    SInt32 colStride,
    DSPSplitComplex * tempBuffer,
    UInt32 log2nInCol,
    UInt32 log2nInRow,
    FFTDirection direction);
```

Parameters

`setup`
: Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the values supplied as parameters `log2nInCol` and `log2nInRow` of the transform function.

`rowStride`
: Specifies a stride across each row of the matrix `signal`. Specifying 1 for `rowStride` processes every element across each row, specifying 2 processes every other element across each row, and so forth.

`colStride`

Specifies a column stride for the matrix, and should generally be allowed to default unless the matrix is a submatrix. Parameter `colStride` can be defaulted by specifying 0. The default column stride equals the row stride multiplied by the column count. Thus, if `rowStride` is 1 and `colStride` is 0, every element of the input /output matrix is processed. If `rowStride` is 2 and `colStride` is 0, every other element of each row is processed.

If not 0, parameter `colStride` represents the distance between each row of the matrix. If parameter `colStride` is 1024, for instance, complex element 512 of the matrix equates to element (1,0), element 1024 equates to element (2,0), and so forth.

`tempBuffer`

A temporary matrix used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 16 KB or 4*`n`, where `log2n = log2nInCol + log2nInRow`.

`log2nInCol`

The base 2 exponent of the number of columns to process for each row. `log2nInCol` must be between 2 and 10, inclusive.

`log2nInRow`

The base 2 exponent of the number of rows to process. For example, to process 64 rows of 128 columns, specify 7 for parameter `log2nInCol` and 6 for parameter log2nInRow. `log2nInRow` must be between 2 and 10, inclusive. See also the FFT Limitations sections in the Target chapters of the Developer's Guide.

`direction`

A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`

Forward transform

`FFT_INVERSE`

Inverse transform

Results are undefined for other values of `direction`.

Discussion

This performs the operation

If F = 1 $\quad C_{nm} = \mathrm{FDFT2D}(C_{nm}) \qquad n = \{0, N-1\} \text{ and } m = \{0, M-1\}$

If F = -1 $\quad C_{nm} = \mathrm{IDFT2D}(C_{nm}) \bullet MN \qquad n = \{0, N-1\} \text{ and } m = \{0, M-1\}$

$$\mathrm{FDFT2D}(X_{nm}) = \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} X_{pq} \bullet e^{(-j2\pi pn)/N} \bullet e^{(-j2\pi qm)/M}$$

$$\mathrm{IDFT2D}(X_{nm}) = \frac{1}{MN} \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} X_{pq} \bullet e^{(j2\pi pn)/N} \bullet e^{(j2\pi qm)/M}$$

Criteria to invoke vectorized code:

- The input/output vectors `signal.realp` and `signal.imagp` and the temporary vectors `tempBuffer.realp` and `tempBuffer.imagp` must be aligned on 16-byte boundaries.
- The value of `rowStride` must be 1.
- The value of `colStride` must be a multiple of 4.
- The values of `log2nInRow` and `log2nInCol` must be between 2 and 10, inclusive.

If any of these criteria is not satisfied, the function invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_fft2d_ziptD

Computes an in-place complex discrete Fourier transform of matrix represented by `signal`, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse).

```
void
vDSP_fft2d_ziptD (FFTSetupD setup,
    DSPSplitComplex * signal,
    SInt32 rowStride,
    SInt32 colStride,
    DSPSplitComplex * tempBuffer,
    UInt32 log2nInCol,
    UInt32 log2nInRow,
    FFTDirection direction);
```

Parameters

setup
Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the values supplied as parameters `log2nInCol` and `log2nInRow` of the transform function.

rowStride
Specifies a stride across each row of the matrix `signal`. Specifying 1 for `rowStride` processes every element across each row, specifying 2 processes every other element across each row, and so forth.

colStride

Specifies a column stride for the matrix, and should generally be allowed to default unless the matrix is a submatrix. Parameter `colStride` can be defaulted by specifying 0. The default column stride equals the row stride multiplied by the column count. Thus, if `rowStride` is 1 and `colStride` is 0, every element of the input /output matrix is processed. If `rowStride` is 2 and `colStride` is 0, every other element of each row is processed.

If not 0, parameter `colStride` represents the distance between each row of the matrix. If parameter `colStride` is 1024, for instance, complex element 512 of the matrix equates to element (1,0), element 1024 equates to element (2,0), and so forth.

tempBuffer

A temporary matrix used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 16 KB or 4*`n`, where `log2n = log2nInCol + log2nInRow`.

log2nInCol

The base 2 exponent of the number of columns to process for each row. `log2nInCol` must be between 2 and 10, inclusive.

log2nInRow

The base 2 exponent of the number of rows to process. For example, to process 64 rows of 128 columns, specify 7 for parameter `log2nInCol` and 6 for parameter log2nInRow. `log2nInRow` must be between 2 and 10, inclusive. See also the FFT Limitations sections in the Target chapters of the Developer's Guide.

direction

A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`

Forward transform

`FFT_INVERSE`

Inverse transform

Results are undefined for other values of `direction`.

Discussion

This performs the operation

If F = 1 $C_{nm} = \mathrm{FDFT2D}(C_{nm})$ n = {0, N-1} and m = {0, M-1}

If F = -1 $C_{nm} = \mathrm{IDFT2D}(C_{nm}) \cdot MN$ n = {0, N-1} and m = {0, M-1}

$$\mathrm{FDFT2D}(X_{nm}) = \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} X_{pq} \cdot e^{(-j2\pi pn)/N} \cdot e^{(-j2\pi qm)/M}$$

$$\mathrm{IDFT2D}(X_{nm}) = \frac{1}{MN} \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} X_{pq} \cdot e^{(j2\pi pn)/N} \cdot e^{(j2\pi qm)/M}$$

Criteria to invoke vectorized code:

Double Precision - `vDSP_fft2d_zipD` and `vDSP_fft2d_ziptD`

No altivec support for double precision. The function always invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_fft2d_zop

Computes an out-of-place complex discrete Fourier transform of the matrix represented by `signal`, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse).

```
void
vDSP_fft2d_zop (FFTSetup setup,
    DSPSplitComplex * signal,
    SInt32 signalRowStride,
    SInt32 signalColStride,
    DSPSplitComplex * result,
    SInt32 resultRowStride,
    SInt32 resultColStride,
    UInt32 log2nInCol,
    UInt32 log2nInRow,
    FFTDirection direction);
```

Parameters

`setup`
Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the values supplied as parameters `log2nInCol` and log2nInRow of the transform function.

`signalRowStride`
Specifies a stride across each row of matrix `a`. Specifying 1 for signalRowStride processes every element across each row, specifying 2 processes every other element across each row, and so forth.

`signalColStride`
If not 0, this parameter represents the distance between each row of the input /output matrix. If parameter signalColStride is 1024, for instance, element 512 equates to element (1,0) of matrix `a`, element 1024 equates to element (2,0), and so forth.

`colStride`
Specifies a column stride for the input /output matrix, and should generally be allowed to default unless the input /output matrix is a submatrix. Parameter colStride can be defaulted by specifying 0. The default stride equals the row stride multiplied by the column count. Thus, if signalRowStride is 1 and signalColStride is 0, every element of matrix `a` is processed. If signalRowStride is 2 and signalColStride is 0, every other element of each row is processed.

resultRowStride
: Specifies a row stride for output matrix `result` in the same way that `signalRowStride` specifies a stride for input the input /output matrix.

resultColStride
: Specifies a column stride for output matrix `result` in the same way that `signalColStride` specifies a stride for input the input /output matrix.

log2nInCol
: The base 2 exponent of the number of columns to process for each row. `log2nInCol` must be between 2 and 10, inclusive.

log2nInRow
: The base 2 exponent of the number of rows to process. For example, to process 64 rows of 128 columns, specify 7 for `log2nInCol` and 6 for `log2nInRow`. `log2nInRow` must be between 2 and 10, inclusive.

direction
: A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`
: Forward transform

`FFT_INVERSE`
: Inverse transform

Results are undefined for other values of `direction`.

Discussion

This performs the operation

If F = 1 $C_m = \text{FDFT}(A_m)$ If F = -1 $C_m = \text{IDFT}(A_m) \bullet N$ m = {0, N-1}

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

Criteria to invoke vectorized code:

- The input vectors `signal.realp` and `signal.imagp`, the output vectors `result.realp` and `result.imagp`, and the temporary vectors `tempBuffer.realp` and `tempBuffer.imagp` must be aligned on 16-byte boundaries.
- The values of signalRowStride and `resultRowStride` must be 1.
- The values of signalColStride and `resultColStride` must be multiples of 4.
- The values of `log2nInRow` and `log2nInCol` must be between 2 and 10, inclusive.

If any of these criteria is not satisfied, the function invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_fft2d_zopD

Computes an out-of-place complex discrete Fourier transform of the matrix represented by `signal`, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse).

```
void
vDSP_fft2d_zopD (FFTSetupD setup,
    DSPDoubleSplitComplex * signal,
    SInt32 signalRowStride,
    SInt32 signalColStride,
    DSPDoubleSplitComplex * result,
    SInt32 resultRowStride,
    SInt32 resultColStride,
    UInt32 log2nInCol,
    UInt32 log2nInRow,
    FFTDirection direction);
```

Parameters

setup
Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the values supplied as parameters `log2nInCol` and log2nInRow of the transform function.

signalRowStride
Specifies a stride across each row of matrix `a`. Specifying 1 for signalRowStride processes every element across each row, specifying 2 processes every other element across each row, and so forth.

signalColStride
If not 0, this parameter represents the distance between each row of the input /output matrix. If parameter signalColStride is 1024, for instance, element 512 equates to element (1,0) of matrix `a`, element 1024 equates to element (2,0), and so forth.

colStride
Specifies a column stride for the input /output matrix, and should generally be allowed to default unless the input /output matrix is a submatrix. Parameter colStride can be defaulted by specifying 0. The default stride equals the row stride multiplied by the column count. Thus, if signalRowStride is 1 and signalColStride is 0, every element of matrix `a` is processed. If signalRowStride is 2 and signalColStride is 0, every other element of each row is processed.

resultRowStride
Specifies a row stride for output matrix `result` in the same way that `signalRowStride` specifies a stride for input the input /output matrix.

resultColStride
Specifies a column stride for output matrix `result` in the same way that `signalColStride` specifies a stride for input the input /output matrix.

log2nInCol
The base 2 exponent of the number of columns to process for each row. `log2nInCol` must be between 2 and 10, inclusive.

log2nInRow
The base 2 exponent of the number of rows to process. For example, to process 64 rows of 128 columns, specify 7 for `log2nInCol` and 6 for `log2nInRow`. `log2nInRow` must be between 2 and 10, inclusive.

direction
A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`
Forward transform

`FFT_INVERSE`
Inverse transform

Results are undefined for other values of `direction`.

Discussion
This performs the operation

$$\text{If F = 1} \quad C_m = \text{FDFT}(A_m) \qquad \text{If F = -1} \quad C_m = \text{IDFT}\ (A_m) \bullet N \qquad \text{m} = \{0, \text{N-1}\}$$

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_fft2d_zopt

Computes an out-of-place complex discrete Fourier transform of the matrix represented by `signal`, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse).

```
void
vDSP_fft2d_zopt (FFTSetup setup,
    DSPSplitComplex * signal,
    SInt32 signalRowStride,
    SInt32 signalColStride,
    DSPSplitComplex * result,
    SInt32 resultRowStride,
    SInt32 resultColStride,
    DSPSplitComplex * tempBuffer,
    UInt32 log2nInCol,
    UInt32 log2nInRow,
    FFTDirection direction);
```

Parameters

setup

Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the values supplied as parameters `log2nInCol` and log2nInRow of the transform function.

signalRowStride

Specifies a stride across each row of matrix `a`. Specifying 1 for signalRowStride processes every element across each row, specifying 2 processes every other element across each row, and so forth.

signalColStride

If not 0, this parameter represents the distance between each row of the input /output matrix. If parameter signalColStride is 1024, for instance, element 512 equates to element (1,0) of matrix `a`, element 1024 equates to element (2,0), and so forth.

colStride

Specifies a column stride for the input /output matrix, and should generally be allowed to default unless the input /output matrix is a submatrix. Parameter colStride can be defaulted by specifying 0. The default stride equals the row stride multiplied by the column count. Thus, if signalRowStride is 1 and signalColStride is 0, every element of matrix `a` is processed. If signalRowStride is 2 and signalColStride is 0, every other element of each row is processed.

resultRowStride

Specifies a row stride for output matrix `result` in the same way that `signalRowStride` specifies a stride for input the input /output matrix.

resultColStride

Specifies a column stride for output matrix `result` in the same way that `signalColStride` specifies a stride for input the input /output matrix.

tempBuffer

A temporary matrix used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 16 KB or 4*`n`, where `log2n = log2nInCol + log2nInRow`.

log2nInCol

The base 2 exponent of the number of columns to process for each row. `log2nInCol` must be between 2 and 10, inclusive.

`log2nInRow`

The base 2 exponent of the number of rows to process. For example, to process 64 rows of 128 columns, specify 7 for `log2nInCol` and 6 for `log2nInRow`. `log2nInRow` must be between 2 and 10, inclusive.

`direction`

A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`

Forward transform

`FFT_INVERSE`

Inverse transform

Results are undefined for other values of `direction`.

Discussion

This performs the operation

If F = 1 $\quad C_m = \text{FDFT}(A_m) \quad$ If F = -1 $\quad C_m = \text{IDFT}(A_m) \bullet N \quad$ m = {0, N-1}

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

Criteria to invoke vectorized code:

- The input vectors `signal.realp` and `signal.imagp`, the output vectors `result.realp` and `result.imagp`, and the temporary vectors `tempBuffer.realp` and `tempBuffer.imagp` must be aligned on 16-byte boundaries.
- The values of signalRowStride and `resultRowStride` must be 1.
- The values of signalColStride and `resultColStride` must be multiples of 4.
- The values of `log2nInRow` and `log2nInCol` must be between 2 and 10, inclusive.

If any of these criteria is not satisfied, the function invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability

Available in Mac OS X v10.4 and later.

vDSP_fft2d_zoptD

Computes an out-of-place complex discrete Fourier transform of the matrix represented by `signal`, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse).

```
void
vDSP_fft2d_zoptD (FFTSetupD setup,
    DSPDoubleSplitComplex * signal,
    SInt32 signalRowStride,
    SInt32 signalColStride,
    DSPDoubleSplitComplex * result,
    SInt32 resultRowStride,
    SInt32 resultColStride,
    DSPDoubleSplitComplex * tempBuffer,
    UInt32 log2nInCol,
    UInt32 log2nInRow,
    FFTDirection direction);
```

Parameters

setup

Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the values supplied as parameters `log2nInCol` and log2nInRow of the transform function.

signalRowStride

Specifies a stride across each row of matrix `a`. Specifying 1 for signalRowStride processes every element across each row, specifying 2 processes every other element across each row, and so forth.

signalColStride

If not 0, this parameter represents the distance between each row of the input /output matrix. If parameter signalColStride is 1024, for instance, element 512 equates to element (1,0) of matrix `a`, element 1024 equates to element (2,0), and so forth.

colStride

Specifies a column stride for the input /output matrix, and should generally be allowed to default unless the input /output matrix is a submatrix. Parameter colStride can be defaulted by specifying 0. The default stride equals the row stride multiplied by the column count. Thus, if signalRowStride is 1 and signalColStride is 0, every element of matrix `a` is processed. If signalRowStride is 2 and signalColStride is 0, every other element of each row is processed.

resultRowStride

Specifies a row stride for output matrix `result` in the same way that `signalRowStride` specifies a stride for input the input /output matrix.

resultColStride

Specifies a column stride for output matrix `result` in the same way that `signalColStride` specifies a stride for input the input /output matrix.

tempBuffer

A temporary matrix used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 16 KB or 4*`n`, where `log2n = log2nInCol + log2nInRow`.

log2nInCol

The base 2 exponent of the number of columns to process for each row. `log2nInCol` must be between 2 and 10, inclusive.

`log2nInRow`
The base 2 exponent of the number of rows to process. For example, to process 64 rows of 128 columns, specify 7 for `log2nInCol` and 6 for `log2nInRow`. `log2nInRow` must be between 2 and 10, inclusive.

`direction`
A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`
Forward transform

`FFT_INVERSE`
Inverse transform

Results are undefined for other values of `direction`.

Discussion
This performs the operation

If F = 1 $C_m = \text{FDFT}(A_m)$ If F = -1 $C_m = \text{IDFT}(A_m) \cdot N$ m = {0, N-1}

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \cdot e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \cdot e^{(j2\pi nm)/N}$$

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_fft2d_zrip

Computes an in-place real discrete Fourier transform, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse).

```
void
vDSP_fft2d_zrip (FFTSetup setup,
    DSPSplitComplex * signal,
    SInt32 rowStride,
    SInt32 colStride,
    UInt32 log2nInCol,
    UInt32 log2nInRow,
    FFTDirection direction);
```

Parameters

`setup`

Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the values supplied as parameters log2nInCol and log2nInRow of the transform function.

`rowStride`

Specifies a stride across each row of the input matrix signal. Specifying 1 for rowStride processes every element across each row, specifying 2 processes every other element across each row, and so forth.

`colStride`

Specifies a column stride for the matrix, and should generally be allowed to default unless the matrix is a submatrix. Parameter `colStride` can be defaulted by specifying 0. The default column stride equals the row stride multiplied by the column count. Thus, if rowStride is 1 and colStride is 0, every element of the input /output matrix is processed. If rowStride is 2 and colStride is 0, every other element of each row is processed.

If not 0, `colStride` represents the distance between each row of the matrix. If `colStride` is 1024, for instance, complex element 512 of the matrix equates to element (1,0), element 1024 equates to element (2,0), and so forth.

`log2nInCol`

The base 2 exponent of the number of columns to process for each row. `log2nInCol` must be between 2 and 10, inclusive.

`log2nInRow`

The base 2 exponent of the number of rows to process. For example, to process 64 rows of 128 columns, specify 7 for `log2nInCol` and 6 for `log2nInRow`. `log2nInRow` must be between 2 and 10, inclusive.

`direction`

A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`

Forward transform

`FFT_INVERSE`

Inverse transform

Results are undefined for other values of `direction`.

Discussion

Forward transforms read real input and write packed complex output. Inverse transforms read packed complex input and write real output. As a result of packing the frequency-domain data, spatial-domain data and its equivalent frequency-domain data have the same storage requirements; see "Data Packing for Real FFTs" (page 20).

If F = 1 $C_{nm} = \text{FDFT2D}(C_{nm}) \cdot 2$ n = {0, N-1} and m = {0, M-1}

If F = -1 $C_{nm} = \text{IDFT2D}(C_{nm}) \cdot MN$ n = {0, N-1} and m = {0, M-1}

$$\text{FDFT2D}(X_{nm}) = \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} X_{pq} \cdot e^{(-j2\pi pn)/N} \cdot e^{(-j2\pi qm)/M}$$

$$\text{IDFT2D}(X_{nm}) = \frac{1}{MN} \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} X_{pq} \cdot e^{(j2\pi pn)/N} \cdot e^{(j2\pi qm)/M}$$

Real data is stored in split complex form, with odd reals stored on the imaginary side of the split complex form and even reals in stored on the real side.

```
int nr, nc, tempSize;
nr = 1<<log2InRow;
nc = 1<<log2InCol;
if ( ( (log2InCol-1) < 3 ) || ( log2InRow > 9)
{
    tempSize = 9 * nr;
}
else
{
    tempSize = 17 * nr
}
tempBuffer.realp = ( float* ) malloc (tempSize * sizeOf ( float  ) );
tempBuffer.imagp = ( float* ) malloc (tempSize * sizeOf ( float  ) );
```

Criteria to invoke vectorized code:

- The input/output vectors `signal.realp` and `signal.imagp` and the temporary vectors `tempBuffer.realp` and `tempBuffer.imagp` must be aligned on 16-byte boundaries.
- The value of `rowStride` must be 1.
- The value of `colStride` must be a multiple of 4.
- The value of `log2nInRow` and `log2nInCol` must be between 3 and 10, inclusive.

If any of these criteria is not satisfied, the function invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_fft2d_zripD

Computes an in-place real discrete Fourier transform, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse).

```
void
vDSP_fft2d_zripD (FFTSetupD setup,
    DSPDoubleSplitComplex * signal,
    SInt32 rowStride,
    SInt32 colStride,
    UInt32 log2nInCol,
    UInt32 log2nInRow,
    FFTDirection direction);
```

Parameters

setup

Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the values supplied as parameters log2nInCol and log2nInRow of the transform function.

rowStride

Specifies a stride across each row of the input matrix signal. Specifying 1 for rowStride processes every element across each row, specifying 2 processes every other element across each row, and so forth.

colStride

Specifies a column stride for the matrix, and should generally be allowed to default unless the matrix is a submatrix. Parameter `colStride` can be defaulted by specifying 0. The default column stride equals the row stride multiplied by the column count. Thus, if rowStride is 1 and colStride is 0, every element of the input /output matrix is processed. If rowStride is 2 and colStride is 0, every other element of each row is processed.

If not 0, `colStride` represents the distance between each row of the matrix. If `colStride` is 1024, for instance, complex element 512 of the matrix equates to element (1,0), element 1024 equates to element (2,0), and so forth.

log2nInCol

The base 2 exponent of the number of columns to process for each row. `log2nInCol` must be between 2 and 10, inclusive.

log2nInRow

The base 2 exponent of the number of rows to process. For example, to process 64 rows of 128 columns, specify 7 for `log2nInCol` and 6 for `log2nInRow`. `log2nInRow` must be between 2 and 10, inclusive.

direction

A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`

Forward transform

`FFT_INVERSE`

Inverse transform

Results are undefined for other values of `direction`.

Discussion

Forward transforms read real input and write packed complex output. Inverse transforms read packed complex input and write real output. As a result of packing the frequency-domain data, spatial-domain data and its equivalent frequency-domain data have the same storage requirements; see "Data Packing for Real FFTs" (page 20).

If F = 1 $C_{nm} = \mathrm{FDFT2D}(C_{nm}) \bullet 2$ n = {0, N-1} and m = {0, M-1}

If F = -1 $C_{nm} = \mathrm{IDFT2D}(C_{nm}) \bullet MN$ n = {0, N-1} and m = {0, M-1}

$$\mathrm{FDFT2D}(X_{nm}) = \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} X_{pq} \bullet e^{(-j2\pi pn)/N} \bullet e^{(-j2\pi qm)/M}$$

$$\mathrm{IDFT2D}(X_{nm}) = \frac{1}{MN} \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} X_{pq} \bullet e^{(j2\pi pn)/N} \bullet e^{(j2\pi qm)/M}$$

Real data is stored in split complex form, with odd reals stored on the imaginary side of the split complex form and even reals in stored on the real side.

```
int nr, nc, tempSize;
nr = 1<<log2InRow;
nc = 1<<log2InCol;
if ( ( (log2InCol-1) < 3 ) || ( log2InRow > 9)
{
    tempSize = 9 * nr;
}
else
{
    tempSize = 17 * nr
}
tempBuffer.realp = ( float* ) malloc (tempSize * sizeOf ( float  ) );
tempBuffer.imagp = ( float* ) malloc (tempSize * sizeOf ( float  ) );
```

No altivec support for double precision. The function always invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability

Available in Mac OS X v10.4 and later.

vDSP_fft2d_zript

Computes an in-place real discrete Fourier transform, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse).

```
void
vDSP_fft2d_zript (FFTSetup setup,
    DSPSplitComplex * signal,
    SInt32 rowStride,
    SInt32 colStride,
    DSPSplitComplex * tempBuffer,
    UInt32 log2nInCol,
    UInt32 log2nInRow,
    FFTDirection direction);
```

Parameters

setup

Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the values supplied as parameters log2nInCol and log2nInRow of the transform function.

rowStride

Specifies a stride across each row of the input matrix signal. Specifying 1 for rowStride processes every element across each row, specifying 2 processes every other element across each row, and so forth.

colStride

Specifies a column stride for the matrix, and should generally be allowed to default unless the matrix is a submatrix. Parameter `colStride` can be defaulted by specifying 0. The default column stride equals the row stride multiplied by the column count. Thus, if rowStride is 1 and colStride is 0, every element of the input /output matrix is processed. If rowStride is 2 and colStride is 0, every other element of each row is processed.

If not 0, `colStride` represents the distance between each row of the matrix. If `colStride` is 1024, for instance, complex element 512 of the matrix equates to element (1,0), element 1024 equates to element (2,0), and so forth.

tempBuffer

A temporary matrix used for storing interim results. The size of temporary memory for each part (real and imaginary) should be at least as great as the actual data, or 17 columns of data, whichever is greater.

log2nInCol

The base 2 exponent of the number of columns to process for each row. `log2nInCol` must be between 3 and 10, inclusive.

log2nInRow

The base 2 exponent of the number of rows to process. For example, to process 64 rows of 128 columns, specify 7 for `log2nInCol` and 6 for `log2nInRow`. `log2nInRow` must be between 3 and 10, inclusive.

`direction`
: A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`
: Forward transform

`FFT_INVERSE`
: Inverse transform

Results are undefined for other values of `direction`.

Discussion

Forward transforms read real input and write packed complex output. Inverse transforms read packed complex input and write real output. As a result of packing the frequency-domain data, spatial-domain data and its equivalent frequency-domain data have the same storage requirements; see "Data Packing for Real FFTs" (page 20).

If F = 1 $\quad C_{nm} = \mathrm{FDFT2D}(C_{nm}) \cdot 2 \qquad$ n = {0, N-1} and m = {0, M-1}

If F = -1 $\quad C_{nm} = \mathrm{IDFT2D}(C_{nm}) \cdot MN \qquad$ n = {0, N-1} and m = {0, M-1}

$$\mathrm{FDFT2D}(X_{nm}) = \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} X_{pq} \cdot e^{(-j2\pi pn)/N} \cdot e^{(-j2\pi qm)/M}$$

$$\mathrm{IDFT2D}(X_{nm}) = \frac{1}{MN} \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} X_{pq} \cdot e^{(j2\pi pn)/N} \cdot e^{(j2\pi qm)/M}$$

Real data is stored in split complex form, with odd reals stored on the imaginary side of the split complex form and even reals in stored on the real side.

Criteria to invoke vectorized code:

- The input/output vectors `signal.realp` and `signal.imagp` and the temporary vectors `tempBuffer.realp` and `tempBuffer.imagp` must be aligned on 16-byte boundaries.
- The value of `rowStride` must be 1.
- The value of `colStride` must be a multiple of 4.
- The value of `log2nInRow` and `log2nInCol` must be between 3 and 10, inclusive.

If any of these criteria is not satisfied, the function invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability

Available in Mac OS X v10.4 and later.

vDSP_fft2d_zriptD

Computes an in-place real discrete Fourier transform, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse).

```
void
vDSP_fft2d_zriptD (FFTSetupD setup,
    DSPDoubleSplitComplex * signal,
    SInt32 rowStride,
    SInt32 colStride,
    DSPDoubleSplitComplex * tempBuffer,
    UInt32 log2nInCol,
    UInt32 log2nInRow,
    FFTDirection direction);
```

Parameters

setup

Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the values supplied as parameters log2nInCol and log2nInRow of the transform function.

rowStride

Specifies a stride across each row of the input matrix signal. Specifying 1 for rowStride processes every element across each row, specifying 2 processes every other element across each row, and so forth.

colStride

Specifies a column stride for the matrix, and should generally be allowed to default unless the matrix is a submatrix. Parameter `colStride` can be defaulted by specifying 0. The default column stride equals the row stride multiplied by the column count. Thus, if rowStride is 1 and colStride is 0, every element of the input /output matrix is processed. If rowStride is 2 and colStride is 0, every other element of each row is processed.

If not 0, `colStride` represents the distance between each row of the matrix. If `colStride` is 1024, for instance, complex element 512 of the matrix equates to element (1,0), element 1024 equates to element (2,0), and so forth.

tempBuffer

The size of temporary memory for each part (real and imaginary)should be at least as great as the actual data, or 17 columns of data, whichever is greater.

log2nInCol

The base 2 exponent of the number of columns to process for each row. `log2nInCol` must be between 3 and 10, inclusive.

log2nInRow

The base 2 exponent of the number of rows to process. For example, to process 64 rows of 128 columns, specify 7 for `log2nInCol` and 6 for `log2nInRow`. `log2nInRow` must be between 3 and 10, inclusive.

direction
A forward/inverse directional flag, and must specify one of the following:

FFT_FORWARD
Forward transform

FFT_INVERSE
Inverse transform

Results are undefined for other values of direction.

Discussion
Forward transforms read real input and write packed complex output. Inverse transforms read packed complex input and write real output. As a result of packing the frequency-domain data, spatial-domain data and its equivalent frequency-domain data have the same storage requirements; see "Data Packing for Real FFTs" (page 20).

If F = 1 $C_{nm} = \text{FDFT2D}(C_{nm}) \cdot 2$ n = {0, N-1} and m = {0, M-1}

If F = -1 $C_{nm} = \text{IDFT2D}(C_{nm}) \cdot MN$ n = {0, N-1} and m = {0, M-1}

$$\text{FDFT2D}(X_{nm}) = \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} X_{pq} \cdot e^{(-j2\pi pn)/N} \cdot e^{(-j2\pi qm)/M}$$

$$\text{IDFT2D}(X_{nm}) = \frac{1}{MN} \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} X_{pq} \cdot e^{(j2\pi pn)/N} \cdot e^{(j2\pi qm)/M}$$

Real data is stored in split complex form, with odd reals stored on the imaginary side of the split complex form and even reals in stored on the real side.

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_fft2d_zrop

Computes an out-of-place real discrete Fourier transform, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse).

```
void
vDSP_fft2d_zrop (FFTSetup setup,
    DSPSplitComplex * signal,
    SInt32 signalRowStride,
    SInt32 signalColStride,
    DSPSplitComplex * result,
    SInt32 resultRowStride,
    SInt32 resultColStride,
    UInt32 log2nInCol,
    UInt32 log2nInRow,
    FFTDirection direction);
```

Parameters

setup

Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the value supplied as parameter `log2n` or `log2m`, whichever is larger, of the transform function.

signalRowStride

Specifies a stride across each row of matrix signal. Specifying 1 for signalRowStride processes every element across each row, specifying 2 processes every other element across each row, and so forth.

signalColStride

If not 0, represents the distance between each row of the input /output matrix. If parameter signalColStride is 1024, for instance, element 512 equates to element (1,0) of matrix `a`, element 1024 equates to element (2,0), and so forth.

colStride

Specifies a column stride for the matrix, and should generally be allowed to default unless the matrix is a submatrix. Parameter colStride can be defaulted by specifying 0. The default stride equals the row stride multiplied by the column count. Thus, if signalRowStride is 1 and signalColStride is 0, every element of matrix signal is processed. If signalRowStride is 2 and signalColStride is 0, every other element of each row is processed.

resultRowStride

Specifies a row stride for output matrix `c` in the same way that `signalRowStride` specifies strides for input the matrix.

resultColStride

Specifies a column stride for output matrix `c` in the same way that signalColStride specify strides for input the matrix.

log2nInCol

The base 2 exponent of the number of columns to process for each row. `log2nInCol` must be between 3 and 10, inclusive.

log2nInRow

The base 2 exponent of the number of rows to process. For example, to process 64 rows of 128 columns, specify 7 for `log2nInCol` and 6 for `log2nInRow`. `log2nInRow` must be between 3 and 10, inclusive.

direction

A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`

Forward transform

`FFT_INVERSE`

Inverse transform

Results are undefined for other values of `direction`.

Discussion

Forward transforms read real input and write packed complex output. Inverse transforms read packed complex input and write real output. As a result of packing the frequency-domain data, spatial-domain data and its equivalent frequency-domain data have the same storage requirements; see "Data Packing for Real FFTs" (page 20).

Where $N = 3(2^m)$ for Radix-3 functions, and $N = 5(2^m)$ for Radix-5 functions

If F = 1 $\quad C_m = \mathrm{RDFT}(A_m) \bullet 2 \qquad$ If F = -1 $\quad C_m = \mathrm{IDFT}\ (A_m) \bullet N \qquad m = \{0, N\text{-}1\}$

$$\mathrm{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \mathrm{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

Real data is stored in split complex form, with odd reals stored on the imaginary side of the split complex form and even reals in stored on the real side.

Criteria to invoke vectorized code:

- The input vectors `signal.realp` and `signal.imagp`, the output vectors `result.realp` and `result.imagp`, and the temporary vectors `tempBuffer.realp` and `tempBuffer.imagp` must be aligned on 16-byte boundaries.
- The values of `signalRowStride` and `resultRowStride` must be 1.
- The values of `signalColStride` and `resultColStride` must be multiples of 4.
- The value of `log2nInRow` and `log2nInCol` must be between 3 and 10, inclusive.

If any of these criteria is not satisfied, the function invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability

Available in Mac OS X v10.4 and later.

vDSP_fft2d_zropD

Computes an out-of-place real discrete Fourier transform, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse).

```
void
vDSP_fft2d_zropD (FFTSetupD setup,
    DSPDoubleSplitComplex * signal,
    SInt32 signalRowStride,
    SInt32 signalColStride,
    DSPDoubleSplitComplex * result,
    SInt32 resultRowStride,
    SInt32 resultColStride,
    UInt32 log2nInCol,
    UInt32 log2nInRow,
    FFTDirection direction);
```

Parameters

setup

Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the value supplied as parameter `log2n` or `log2m`, whichever is larger, of the transform function.

signalRowStride

Specifies a stride across each row of matrix signal. Specifying 1 for signalRowStride processes every element across each row, specifying 2 processes every other element across each row, and so forth.

signalColStride

If not 0, represents the distance between each row of the input /output matrix. If parameter signalColStride is 1024, for instance, element 512 equates to element (1,0) of matrix `a`, element 1024 equates to element (2,0), and so forth.

colStride

Specifies a column stride for the matrix, and should generally be allowed to default unless the matrix is a submatrix. Parameter colStride can be defaulted by specifying 0. The default stride equals the row stride multiplied by the column count. Thus, if signalRowStride is 1 and signalColStride is 0, every element of matrix signal is processed. If signalRowStride is 2 and signalColStride is 0, every other element of each row is processed.

resultRowStride

Specifies a row stride for output matrix `c` in the same way that `signalRowStride` specifies strides for input the matrix.

resultColStride

Specifies a column stride for output matrix `c` in the same way that signalColStride specify strides for input the matrix.

log2nInCol

The base 2 exponent of the number of columns to process for each row. `log2nInCol` must be between 3 and 10, inclusive.

log2nInRow

The base 2 exponent of the number of rows to process. For example, to process 64 rows of 128 columns, specify 7 for `log2nInCol` and 6 for `log2nInRow`. `log2nInRow` must be between 3 and 10, inclusive.

`direction`
A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`
Forward transform

`FFT_INVERSE`
Inverse transform

Results are undefined for other values of `direction`.

Discussion
Forward transforms read real input and write packed complex output. Inverse transforms read packed complex input and write real output. As a result of packing the frequency-domain data, spatial-domain data and its equivalent frequency-domain data have the same storage requirements; see "Data Packing for Real FFTs" (page 20).

Where $N = 3(2^m)$ for Radix-3 functions, and $N = 5(2^m)$ for Radix-5 functions

If $F = 1$ $\quad C_m = \mathrm{RDFT}(A_m) \bullet 2$ $\quad$ If $F = -1$ $\quad C_m = \mathrm{IDFT}(A_m) \bullet N$ $\quad m = \{0, N-1\}$

$$\mathrm{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \mathrm{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

Real data is stored in split complex form, with odd reals stored on the imaginary side of the split complex form and even reals in stored on the real side.

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_fft2d_zropt

Computes an out-of-place real discrete Fourier transform, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse).

```
void
vDSP_fft2d_zropt (FFTSetup setup,
    DSPSplitComplex * signal,
    SInt32 signalRowStride,
    SInt32 signalColStride,
    DSPSplitComplex * result,
    SInt32 resultRowStride,
    SInt32 resultColStride,
    DSPSplitComplex * tempBuffer,
    UInt32 log2nInCol,
    UInt32 log2nInRow,
    FFTDirection direction);
```

Parameters

setup

Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the value supplied as parameter `log2n` or `log2m`, whichever is larger, of the transform function.

signalRowStride

Specifies a stride across each row of matrix signal. Specifying 1 for signalRowStride processes every element across each row, specifying 2 processes every other element across each row, and so forth.

signalColStride

If not 0, represents the distance between each row of the input /output matrix. If parameter signalColStride is 1024, for instance, element 512 equates to element (1,0) of matrix `a`, element 1024 equates to element (2,0), and so forth.

colStride

Specifies a column stride for the matrix, and should generally be allowed to default unless the matrix is a submatrix. Parameter colStride can be defaulted by specifying 0. The default stride equals the row stride multiplied by the column count. Thus, if signalRowStride is 1 and signalColStride is 0, every element of matrix signal is processed. If signalRowStride is 2 and signalColStride is 0, every other element of each row is processed.

resultRowStride

Specifies a row stride for output matrix `c` in the same way that `signalRowStride` specifies strides for input the matrix.

resultColStride

Specifies a column stride for output matrix `c` in the same way that signalColStride specify strides for input the matrix.

tempBuffer

A temporary matrix used for storing interim results. The size of temporary memory for each part (real and imaginary) can be calculated using the following algorithm:

```
int nr, nc, tempSize;
nr = 1<<log2InRow;
nc = 1<<log2InCol;
if ( ( (log2InCol-1) < 3 ) || ( log2InRow > 9)
{
    tempSize = 9 * nr;
}
else
{
    tempSize = 17 * nr
}
tempBuffer.realp = (float*) malloc (tempSize * sizeOf (float));
tempBuffer.imagp = (float*) malloc (tempSize * sizeOf (float));
```

log2nInCol

The base 2 exponent of the number of columns to process for each row. `log2nInCol` must be between 3 and 10, inclusive.

log2nInRow

The base 2 exponent of the number of rows to process. For example, to process 64 rows of 128 columns, specify 7 for `log2nInCol` and 6 for `log2nInRow`. `log2nInRow` must be between 3 and 10, inclusive.

direction

A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`

Forward transform

`FFT_INVERSE`

Inverse transform

Results are undefined for other values of `direction`.

Discussion

Forward transforms read real input and write packed complex output. Inverse transforms read packed complex input and write real output. As a result of packing the frequency-domain data, spatial-domain data and its equivalent frequency-domain data have the same storage requirements; see "Data Packing for Real FFTs" (page 20).

Where N = $3(2^m)$ for Radix-3 functions, and N = $5(2^m)$ for Radix-5 functions

If F = 1 $C_m = \text{RDFT}(A_m) \bullet 2$ If F = -1 $C_m = \text{IDFT}(A_m) \bullet N$ m = {0, N-1}

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

Real data is stored in split complex form, with odd reals stored on the imaginary side of the split complex form and even reals in stored on the real side.

Criteria to invoke vectorized code:

- The input vectors `signal.realp` and `signal.imagp`, the output vectors `result.realp` and `result.imagp`, and the temporary vectors `tempBuffer.realp` and `tempBuffer.imagp` must be aligned on 16-byte boundaries.
- The values of `signalRowStride` and `resultRowStride` must be 1.
- The values of `signalColStride` and `resultColStride` must be multiples of 4.
- The value of `log2nInRow` and `log2nInCol` must be between 3 and 10, inclusive.

If any of these criteria is not satisfied, the function invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_fft2d_zroptD

Computes an out-of-place real discrete Fourier transform, either from the spatial domain to the frequency domain (forward) or from the frequency domain to the spatial domain (inverse).

```
void
vDSP_fft2d_zroptD (FFTSetupD setup,
    DSPDoubleSplitComplex * signal,
    SInt32 signalRowStride,
    SInt32 signalColStride,
    DSPDoubleSplitComplex * result,
    SInt32 resultRowStride,
    SInt32 resultColStride,
    DSPDoubleSplitComplex * tempBuffer,
    UInt32 log2nInCol,
    UInt32 log2nInRow,
    FFTDirection direction);
```

Parameters

`setup`
: Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the value supplied as parameter `log2n` or `log2m`, whichever is larger, of the transform function.

`signalRowStride`
: Specifies a stride across each row of matrix signal. Specifying 1 for signalRowStride processes every element across each row, specifying 2 processes every other element across each row, and so forth.

`signalColStride`
: If not 0, represents the distance between each row of the input /output matrix. If parameter signalColStride is 1024, for instance, element 512 equates to element (1,0) of matrix `a`, element 1024 equates to element (2,0), and so forth.

colStride

Specifies a column stride for the matrix, and should generally be allowed to default unless the matrix is a submatrix. Parameter colStride can be defaulted by specifying 0. The default stride equals the row stride multiplied by the column count. Thus, if signalRowStride is 1 and signalColStride is 0, every element of matrix signal is processed. If signalRowStride is 2 and signalColStride is 0, every other element of each row is processed.

resultRowStride

Specifies a row stride for output matrix `c` in the same way that `signalRowStride` specifies strides for input the matrix.

resultColStride

Specifies a column stride for output matrix `c` in the same way that signalColStride specify strides for input the matrix.

tempBuffer

A temporary matrix used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 16 KB or 4*`n`, where `log2n = log2nInCol + log2nInRow`.

log2nInCol

The base 2 exponent of the number of columns to process for each row. `log2nInCol` must be between 3 and 10, inclusive.

log2nInRow

The base 2 exponent of the number of rows to process. For example, to process 64 rows of 128 columns, specify 7 for `log2nInCol` and 6 for `log2nInRow`. `log2nInRow` must be between 3 and 10, inclusive.

direction

A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`

Forward transform

`FFT_INVERSE`

Inverse transform

Results are undefined for other values of `direction`.

Discussion

Forward transforms read real input and write packed complex output. Inverse transforms read packed complex input and write real output. As a result of packing the frequency-domain data, spatial-domain data and its equivalent frequency-domain data have the same storage requirements; see "Data Packing for Real FFTs" (page 20).

Where N = $3(2^m)$ for Radix-3 functions, and N = $5(2^m)$ for Radix-5 functions

If F = 1 $\quad C_m = \mathrm{RDFT}(A_m) \cdot 2$ $\quad$ If F = -1 $\quad C_m = \mathrm{IDFT}\ (A_m) \cdot N$ $\quad$ m = {0, N-1}

$$\mathrm{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \cdot e^{(-j2\pi nm)/N} \qquad \mathrm{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \cdot e^{(j2\pi nm)/N}$$

Real data is stored in split complex form, with odd reals stored on the imaginary side of the split complex form and even reals in stored on the real side.

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_fft3_zop

Computes an out-of-place radix-3 complex Fourier transform, either forward or inverses. The number of input and output values processed equals 3 times the power of 2 specified by parameter `log2n`.

```
void
vDSP_fft3_zop (FFTSetup setup,

 DSPSplitComplex * signal,

 SInt32 signalStride,

 DSPSplitComplex * result,

 SInt32 resultStride,

 UInt32 log2n,

 FFTDirection direction);
```

Parameters

setup
: Use `vDSP_create_fftsetup`, to initialize this function. `FFT_RADIX3` must be specified in the call to `vDSP_create_fftsetup` . `setup` is preserved for reuse.

signalStride
: Specifies an address stride through the input vector `signal`. To process every element of the vector, specify 1 for parameter `signalStride`; to process every other element, specify 2. The value of `signalStride` should be 1 for best performance.

resultStride
: Specifies an address stride for the result. The value of `resultStride` should be 1 for best performance.

log2n
: The base 2 exponent of the number of elements to process in a single input signal. . `log2n` must be between 3 and 15, inclusive.

`direction`
A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`
Forward transform

`FFT_INVERSE`
Inverse transform

Results are undefined for other values of `direction`.

Discussion

This performs the operation

Where N = $3(2^m)$ for Radix-3 functions, and N = $5(2^m)$ for Radix-5 functions

If F = 1 $C_m = \mathrm{RDFT}(A_m) \cdot 2$ If F = -1 $C_m = \mathrm{IDFT}\,(A_m) \cdot N$ m = {0, N-1}

$$\mathrm{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \cdot e^{(-j2\pi nm)/N} \qquad \mathrm{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \cdot e^{(j2\pi nm)/N}$$

See also the FFT Limitations sections in the Target chapters of the Developer's Guide.

Criteria to invoke vectorized code:

- `signal.realp` and `signal.imagp` must be 16-byte aligned.
- `signalStride` and `resultStride` must be 1.
- `result.realp` and `result.imagp` must be 16-byte aligned.
- `log2n` must be between 3 and 15, inclusive.

If any of these criteria is not satisfied, the function invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability

Available in Mac OS X v10.4 and later.

vDSP_fft3_zopD

Computes an out-of-place radix-3 complex Fourier transform, either forward or inverse. The number of input and output values processed equals 3 times the power of 2 specified by parameter `log2n`.

```
void
vDSP_fft3_zopD (FFTSetupD setup,

 DSPDoubleSplitComplex * signal,
 SInt32 signalStride,

 DSPDoubleSplitComplex * result,

 SInt32 resultStride,

 UInt32 log2n,

 FFTDirection direction);
```

Parameters

setup

Use `vDSP_create_fftsetup`, to initialize this function. `FFT_RADIX3` must be specified in the call to `vDSP_create_fftsetup` . `setup` is preserved for reuse.

signalStride

Specifies an address stride through the input vector `signal`. To process every element of the vector, specify 1 for parameter `signalStride`; to process every other element, specify 2. The value of `signalStride` should be 1 for best performance.

resultStride

Specifies an address stride for the result. The value of `resultStride` should be 1 for best performance.

log2n

The base 2 exponent of the number of elements to process in a single input signal. `log2n` must be between 3 and 15, inclusive.

direction

A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`

Forward transform

`FFT_INVERSE`

Inverse transform

Results are undefined for other values of `direction`.

Discussion

This performs the operation

Where N = $3(2^m)$ for Radix-3 functions, and N = $5(2^m)$ for Radix-5 functions

If F = 1 $\quad C_m = \text{RDFT}(A_m) \bullet 2$ $\quad$ If F = -1 $\quad C_m = \text{IDFT}\ (A_m) \bullet N$ $\quad$ m = {0, N-1}

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

See also the FFT Limitations sections in the Target chapters of the Developer's Guide.

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_fft5_zop

Computes an out-of-place radix-5 complex Fourier transform, either forward or inverse. The number of input and output values processed equals 5 times the power of 2 specified by parameter `log2n`.

```
void
vDSP_fft5_zop (FFTSetup setup,

 DSPSplitComplex * signal,

 SInt32 signalStride,

 DSPSplitComplex * result,

 SInt32 resultStride,

 UInt32 log2n,

 FFTDirection direction);
```

Parameters

`setup`
Use `vDSP_create_fftsetup`, to initialize this function. `FFT_RADIX5` must be specified in the call to `vDSP_create_fftsetup`. `setup` is preserved for reuse.

`signalStride`
Specifies an address stride through the input vector `signal`. To process every element of the vector, specify 1 for parameter `signalStride`; to process every other element, specify 2. The value of `signalStride` should be 1 for best performance.

`resultStride`
Specifies an address stride for the result. The value of `resultStride` should be 1 for best performance.

`log2n`
The base 2 exponent of the number of elements to process in a single input signal. . `log2n` must be between 3 and 15, inclusive.

`direction`
A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`
Forward transform

`FFT_INVERSE`
Inverse transform

Results are undefined for other values of `direction`.

Discussion
This performs the operation

Where $N = 3(2^m)$ for Radix-3 functions, and $N = 5(2^m)$ for Radix-5 functions

If F = 1 $C_m = \text{RDFT}(A_m) \cdot 2$ If F = -1 $C_m = \text{IDFT}(A_m) \cdot N$ m = {0, N-1}

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \cdot e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \cdot e^{(j2\pi nm)/N}$$

See also the FFT Limitations sections in the Target chapters of the Developer's Guide.

Criteria to invoke vectorized code:

Single Precision - `vDSP_fft3_zop` and `vDSP_fft5_zop`

- `signal.realp` and `signal.imagp` must be 16-byte aligned.
- `signalStride` and `resultStride` must be 1.
- `result.realp` and `result.imagp` must be 16-byte aligned.
- `log2n` must be between 3 and 15, inclusive.

If any of these criteria is not satisfied, the function invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_fft5_zopD

Computes an out-of-place radix-5 complex Fourier transform, either forward or inverse. The number of input and output values processed equals 5 times the power of 2 specified by parameter `log2n`.

```
vDSP_fft5_zopD (FFTSetupD setup,
 DSPDoubleSplitComplex * signal,

 SInt32 signalStride,

 DSPDoubleSplitComplex * result,

 SInt32 resultStride,

 UInt32 log2n,

 FFTDirection direction);
```

Parameters

setup
: Use `vDSP_create_fftsetupD`, to initialize this function. `FFT_RADIX5` must be specified in the call to `vDSP_create_fftsetupD`. `setup` is preserved for reuse.

signalStride
: Specifies an address stride through the input vector `signal`. To process every element of the vector, specify 1 for parameter `signalStride`; to process every other element, specify 2. The value of `signalStride` should be 1 for best performance.

resultStride
: Specifies an address stride for the result. The value of `resultStride` should be 1 for best performance.

log2n
: The base 2 exponent of the number of elements to process in a single input signal. . `log2n` must be between 3 and 15, inclusive.

direction
: A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`
: Forward transform

`FFT_INVERSE`
: Inverse transform

Results are undefined for other values of `direction`.

Discussion

This performs the operation

Where N = $3(2^m)$ for Radix-3 functions, and N = $5(2^m)$ for Radix-5 functions

If F = 1 $C_m = \text{RDFT}(A_m) \bullet 2$ If F = -1 $C_m = \text{IDFT}\,(A_m) \bullet N$ m = {0, N-1}

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

See also the FFT Limitations sections in the Target chapters of the Developer's Guide.

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_fftm_zip

Performs multiple Fourier transforms with a single call.

```
void

vDSP_fftm_zip (FFTSetup setup,

 DSPSplitComplex * signal,

 SInt32 signalStride,

 SInt32 fftStride,

 UInt32 log2n,

 UInt32 numFFT,

 FFTDirection direction);
```

Parameters

setup
: Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`.The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of the transform function.

signalStride
: Specifies an address stride through the input signals. To process every element of each signal, specify 1 for parameter signalStride; to process every other element, specify 2. The value of signalStride should be 1 for best performance.

fftStride
: The number of elements between the first element of one input signal and the first element of the next (which is also to length of each input signal, measured in elements).

log2n
: The base 2 exponent of the number of elements to process in a single input signal. For example, to process 512 elements, specify 9 for parameter `log2n`. The value of `log2n` must be between 2 and 12, inclusive.

numFFT
The number of different input signals.

direction
A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`
Forward transform

`FFT_INVERSE`
Inverse transform

Results are undefined for other values of `direction`.

Discussion

The functions allow you to perform Fourier transforms on a number of different input signals at once, using a single call. They can be used for efficient processing of small input signals (less than 512 points). They will work for input signals of 4 points or greater. Each of the input signals processed by a given call must have the same length and address stride. The input signals are concatenated into a single input/output vector, the parameter `signal`.

The functions compute in-place complex discrete Fourier transforms of the input signals, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

Criteria to invoke vectorized code:

- Input/output vectors `tempBuffer.realp` and `tempBuffer.imagp` and temporary vectors `tempBuffer.realp` and `tempBuffer.imagp` must be aligned on 16-byte boundaries.
- The value of `signalStride` must be 1.

If any of these criteria is not satisfied, the function invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability

Available in Mac OS X v10.4 and later.

vDSP_fftm_zipD

Performs multiple Fourier transforms with a single call.

```
void

vDSP_fftm_zipD (FFTSetupD setup,

 DSPDoubleSplitComplex * signal,

 SInt32 signalStride,

 SInt32 fftStride,
 UInt32 log2n,
 UInt32 numFFT,
 FFTDirection direction);
```

Parameters

`setup`
: Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`.The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of the transform function.

`signalStride`
: Specifies an address stride through the input signals. To process every element of each signal, specify 1 for parameter signalStride; to process every other element, specify 2. The value of signalStride should be 1 for best performance.

`fftStride`
: The number of elements between the first element of one input signal and the first element of the next (which is also to length of each input signal, measured in elements).

`log2n`
: The base 2 exponent of the number of elements to process in a single input signal. For example, to process 512 elements, specify 9 for parameter `log2n`. The value of `log2n` must be between 2 and 12, inclusive.

`numFFT`
: The number of different input signals.

`direction`
: A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`
: Forward transform

`FFT_INVERSE`
: Inverse transform

Results are undefined for other values of `direction`.

Discussion

The function allows you to perform Fourier transforms on a number of different input signals at once, using a single call. It can be used for efficient processing of small input signals (less than 512 points). It will work for input signals of 4 points or greater. Each of the input signals processed by a given call must have the same length and address stride. The input signals are concatenated into a single input/output vector, the parameter `signal`.

The function computes in-place complex discrete Fourier transforms of the input signals, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_fftm_zipt

Performs multiple Fourier transforms with a single call.

```
void

vDSP_fftm_zipt (FFTSetup setup,

 DSPSplitComplex * signal,

 SInt32 signalStride,

 SInt32 fftStride,

 DSPSplitComplex * tempBuffer,

 UInt32 log2n,

 UInt32 numFFT,

 FFTDirection direction);
```

Parameters

setup
: Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`.The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of the transform function.

signalStride
: Specifies an address stride through the input signals. To process every element of each signal, specify 1 for parameter signalStride; to process every other element, specify 2. The value of signalStride should be 1 for best performance.

fftStride
: The number of elements between the first element of one input signal and the first element of the next (which is also to length of each input signal, measured in elements).

tempBuffer
: A temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k for best performance. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, `tempBuffer.realp` and `tempBuffer.imagp` should be 32-byte aligned for best performance.

log2n
The base 2 exponent of the number of elements to process in a single input signal. For example, to process 512 elements, specify 9 for parameter `log2n`. The value of `log2n` must be between 2 and 12, inclusive.

numFFT
The number of different input signals.

direction
A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`
Forward transform

`FFT_INVERSE`
Inverse transform

Results are undefined for other values of `direction`.

Discussion
The function allows you to perform Fourier transforms on a number of different input signals at once, using a single call. It can be used for efficient processing of small input signals (less than 512 points). It will work for input signals of 4 points or greater. Each of the input signals processed by a given call must have the same length and address stride. The input signals are concatenated into a single input/output vector, the parameter `signal`.

The function computes in-place complex discrete Fourier transforms of the input signals, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

Criteria to invoke vectorized code:

- Input/output vectors `tempBuffer.realp` and `tempBuffer.imagp` and temporary vectors `tempBuffer.realp` and `tempBuffer.imagp` must be aligned on 16-byte boundaries.
- The value of `signalStride` must be 1.

If any of these criteria is not satisfied, the function invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_fftm_ziptD

Performs multiple Fourier transforms with a single call.

```
void

vDSP_fftm_ziptD (FFTSetupD setup,
 DSPDoubleSplitComplex * signal,
 SInt32 signalStride,
 SInt32 fftStride,
 DSPDoubleSplitComplex * tempBuffer,
 UInt32 log2n,
 UInt32 numFFT,
 FFTDirection direction);
```

Parameters

setup

Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`.The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of the transform function.

signalStride

Specifies an address stride through the input signals. To process every element of each signal, specify 1 for parameter signalStride; to process every other element, specify 2. The value of signalStride should be 1 for best performance.

fftStride

The number of elements between the first element of one input signal and the first element of the next (which is also to length of each input signal, measured in elements).

tempBuffer

A temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k for best performance. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, `tempBuffer.realp` and `tempBuffer.imagp` should be 32-byte aligned for best performance.

log2n

The base 2 exponent of the number of elements to process in a single input signal. For example, to process 512 elements, specify 9 for parameter `log2n`. The value of `log2n` must be between 2 and 12, inclusive.

numFFT

The number of different input signals.

direction

A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`

Forward transform

`FFT_INVERSE`

Inverse transform

Results are undefined for other values of `direction`.

Discussion

The function allows you to perform Fourier transforms on a number of different input signals at once, using a single call. It can be used for efficient processing of small input signals (less than 512 points). It will work for input signals of 4 points or greater. Each of the input signals processed by a given call must have the same length and address stride. The input signals are concatenated into a single input/output vector, the parameter `signal`.

The function computes in-place complex discrete Fourier transforms of the input signals, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability

Available in Mac OS X v10.4 and later.

vDSP_fftm_zop

Performs multiple Fourier transforms with a single call.

```
void

vDSP_fftm_zop (FFTSetup setup,

 DSPSplitComplex * signal,

 SInt32 signalStride,

 SInt32 fftStride,

 DSPSplitComplex * result,

 SInt32 resultStride,

 SInt32 rfftStride,

 UInt32 log2n,

 UInt32 numFFT,

 SInt32 flag);
```

Parameters

`setup`

Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`.The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of the transform function.

signalStride

Specifies an address stride through the input signals. To process every element of each signal, specify 1 for parameter signalStride; to process every other element, specify 2. The value of signalStride should be 1 for best performance.

fftStride

The number of elements between the first element of one input signal and the first element of the next (which is also to length of each input signal, measured in elements).

rfftStride

The number of elements between the first element of one result vector and the next in the output vector `result`.

log2n

The base 2 exponent of the number of elements to process in a single input signal. For example, to process 512 elements, specify 9 for parameter `log2n`. The value of `log2n` must be between 2 and 12, inclusive.

numFFT

The number of different input signals.

direction

A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`

Forward transform

`FFT_INVERSE`

Inverse transform

Results are undefined for other values of `direction`.

Discussion

The function allows you to perform Fourier transforms on a number of different input signals at once, using a single call. It can be used for efficient processing of small input signals (less than 512 points). It will work for input signals of 4 points or greater. Each of the input signals processed by a given call must have the same length and address stride. The input signals are concatenated into a single input/output vector, the parameter `signal`.

The function computes out-of-place complex discrete Fourier transforms of the input signals, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

Criteria to invoke vectorized code:

- The input vectors `signal.realp` and `signal.imagp`, the output vectors `result.realp` and `result.imagp`, and the buffer vectors `tempBuffer.realp` and `tempBuffer.imagp` must be aligned on 16-byte boundaries.
- The values of `signalStride` and `resultStride` must be 1.
- The value of `log2n` must be between 2 and 12, inclusive.

If any of these criteria is not satisfied, the function invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_fftm_zopD

Performs multiple Fourier transforms with a single call.

```
void

vDSP_fftm_zopD (FFTSetupD setup,

 DSPDoubleSplitComplex * signal,

 SInt32 signalStride,

 SInt32 fftStride,

 DSPDoubleSplitComplex * result,

 SInt32 resultStride,

 SInt32 rfftStride,

 UInt32 log2n,

 UInt32 numFFT,

 SInt32 flag);
```

Parameters

setup
: Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`.The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of the transform function.

signalStride
: Specifies an address stride through the input signals. To process every element of each signal, specify 1 for parameter signalStride; to process every other element, specify 2. The value of signalStride should be 1 for best performance.

fftStride
: The number of elements between the first element of one input signal and the first element of the next (which is also to length of each input signal, measured in elements).

rfftStride
: The number of elements between the first element of one result vector and the next in the output vector `result`.

`log2n`
: The base 2 exponent of the number of elements to process in a single input signal. For example, to process 512 elements, specify 9 for parameter `log2n`. The value of `log2n` must be between 2 and 12, inclusive.

`numFFT`
: The number of different input signals.

`direction`
: A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`
: Forward transform

`FFT_INVERSE`
: Inverse transform

Results are undefined for other values of `direction`.

Discussion

The function allows you to perform Fourier transforms on a number of different input signals at once, using a single call. It can be used for efficient processing of small input signals (less than 512 points). It will work for input signals of 4 points or greater. Each of the input signals processed by a given call must have the same length and address stride. The input signals are concatenated into a single input/output vector, the parameter `signal`.

The function computes out-of-place complex discrete Fourier transforms of the input signals, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability

Available in Mac OS X v10.4 and later.

vDSP_fftm_zopt

Performs multiple Fourier transforms with a single call.

```
void

vDSP_fftm_zopt (FFTSetup setup,

 DSPSplitComplex * signal,

 SInt32 signalStride,

 SInt32 fftStride,
 DSPSplitComplex * result,

 SInt32 resultStride,

 SInt32 rfftStride,

 DSPSplitComplex * tempBuffer,

 UInt32 log2n,

 UInt32 numFFT,

 SInt32 flag);
```

Parameters

setup

Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`.The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of the transform function.

signalStride

Specifies an address stride through the input signals. To process every element of each signal, specify 1 for parameter signalStride; to process every other element, specify 2. The value of signalStride should be 1 for best performance.

fftStride

The number of elements between the first element of one input signal and the first element of the next (which is also to length of each input signal, measured in elements).

rfftStride

The number of elements between the first element of one result vector and the next in the output vector `result`.

tempBuffer

A temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k for best performance. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, `tempBuffer.realp` and `tempBuffer.imagp` should be 32-byte aligned for best performance.

log2n

The base 2 exponent of the number of elements to process in a single input signal. For example, to process 512 elements, specify 9 for parameter `log2n`. The value of `log2n` must be between 2 and 12, inclusive.

numFFT

The number of different input signals.

`direction`
: A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`
: Forward transform

`FFT_INVERSE`
: Inverse transform

Results are undefined for other values of `direction`.

Discussion

The function allows you to perform Fourier transforms on a number of different input signals at once, using a single call. It can be used for efficient processing of small input signals (less than 512 points). It will work for input signals of 4 points or greater. Each of the input signals processed by a given call must have the same length and address stride. The input signals are concatenated into a single input/output vector, the parameter `signal`.

The function computes out-of-place complex discrete Fourier transforms of the input signals, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

Criteria to invoke vectorized code:

- The input vectors `signal.realp` and `signal.imagp`, the output vectors `result.realp` and `result.imagp`, and the buffer vectors `tempBuffer.realp` and `tempBuffer.imagp` must be aligned on 16-byte boundaries.
- The values of `signalStride` and `resultStride` must be 1.
- The value of `log2n` must be between 2 and 12, inclusive.

If any of these criteria is not satisfied, the function invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability

Available in Mac OS X v10.4 and later.

vDSP_fftm_zoptD

Performs multiple Fourier transforms with a single call.

```
void

vDSP_fftm_zoptD (FFTSetupD setup,

 DSPDoubleSplitComplex * signal,

 SInt32 signalStride,

 SInt32 fftStride,
 DSPDoubleSplitComplex * result,
 SInt32 resultStride,
 SInt32 rfftStride,
 DSPDoubleSplitComplex * tempBuffer,
 UInt32 log2n,
 UInt32 numFFT,
 SInt32 flag);
```

Parameters

setup

Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`.The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of the transform function.

signalStride

Specifies an address stride through the input signals. To process every element of each signal, specify 1 for parameter signalStride; to process every other element, specify 2. The value of signalStride should be 1 for best performance.

fftStride

The number of elements between the first element of one input signal and the first element of the next (which is also to length of each input signal, measured in elements).

rfftStride

The number of elements between the first element of one result vector and the next in the output vector `result`.

tempBuffer

A temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k for best performance. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, `tempBuffer.realp` and `tempBuffer.imagp` should be 32-byte aligned for best performance.

log2n

The base 2 exponent of the number of elements to process in a single input signal. For example, to process 512 elements, specify 9 for parameter `log2n`. The value of `log2n` must be between 2 and 12, inclusive.

numFFT

The number of different input signals.

`direction`
A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`
Forward transform

`FFT_INVERSE`
Inverse transform

Results are undefined for other values of `direction`.

Discussion

The function allows you to perform Fourier transforms on a number of different input signals at once, using a single call. It can be used for efficient processing of small input signals (less than 512 points). It will work for input signals of 4 points or greater. Each of the input signals processed by a given call must have the same length and address stride. The input signals are concatenated into a single input/output vector, the parameter `signal`.

The function computes out-of-place complex discrete Fourier transforms of the input signals, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability

Available in Mac OS X v10.4 and later.

vDSP_fftm_zrip

Performs multiple Fourier transform with a single call.

```
void

vDSP_fftm_zrip (FFTSetup setup,

 DSPSplitComplex * signal,

 SInt32 signalStride,

 SInt32 fftStride,

 UInt32 log2n,

 UInt32 numFFT,

 FFTDirection direction);
```

Parameters

setup

Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`.The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of the transform function.

signalStride

Specifies an address stride through the input signals. To process every element of each signal, specify 1 for parameter signalStride; to process every other element, specify 2. The value of signalStride should be 1 for best performance.

fftStride

The number of elements between the first element of one input signal and the first element of the next (which is also to length of each input signal, measured in elements).

rfftStride

The number of elements between the first element of one result vector and the next in the output vector `result`.

log2n

The base 2 exponent of the number of elements to process in a single input signal. For example, to process 512 elements, specify 9 for parameter `log2n`. The value of `log2n` must be between 3 and 12, inclusive.

numFFT

The number of different input signals.

direction

A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`

Forward transform

`FFT_INVERSE`

Inverse transform

Results are undefined for other values of `direction`.

Discussion

The functions allow you to perform Fourier transforms on a number of different input signals at once, using a single call. They can be used for efficient processing of small input signals (less than 512 points). They will work for input signals of 4 points or greater. Each of the input signals processed by a given call must have the same length and address stride. The input signals are concatenated into a single input/output vector, the parameter `signal`.

The functions compute in-place real discrete Fourier transforms of the input signals, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

Criteria to invoke vectorized code:

- The input/output vectors `signal.realp` and `signal.imagp` must be aligned on 16-byte boundaries.
- The value of `signalStride` must be 1.

- The value of `log2n` must be between 3 and 12, inclusive.

If any of these criteria is not satisfied, the function invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_fftm_zripD

Performs multiple Fourier transform with a single call.

```
void

vDSP_fftm_zripD (FFTSetupD setup,

 DSPDoubleSplitComplex * signal,

 SInt32 signalStride,

 SInt32 fftStride,

 UInt32 log2n,

 UInt32 numFFT,

 FFTDirection direction);
```

Parameters

`setup`
Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`.The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of the transform function.

`signalStride`
Specifies an address stride through the input signals. To process every element of each signal, specify 1 for parameter signalStride; to process every other element, specify 2. The value of signalStride should be 1 for best performance.

`fftStride`
The number of elements between the first element of one input signal and the first element of the next (which is also to length of each input signal, measured in elements).

`rfftStride`
The number of elements between the first element of one result vector and the next in the output vector `result`.

log2n
: The base 2 exponent of the number of elements to process in a single input signal. For example, to process 512 elements, specify 9 for parameter `log2n`. The value of `log2n` must be between 3 and 12, inclusive.

numFFT
: The number of different input signals.

direction
: A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`
: Forward transform

`FFT_INVERSE`
: Inverse transform

Results are undefined for other values of `direction`.

Discussion

The functions allow you to perform Fourier transforms on a number of different input signals at once, using a single call. They can be used for efficient processing of small input signals (less than 512 points). They will work for input signals of 4 points or greater. Each of the input signals processed by a given call must have the same length and address stride. The input signals are concatenated into a single input/output vector, the parameter `signal`.

The functions compute in-place real discrete Fourier transforms of the input signals, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability

Available in Mac OS X v10.4 and later.

vDSP_fftm_zript

Performs multiple Fourier transform with a single call.

```
vDSP_fftm_zript (FFTSetup setup,

 DSPSplitComplex * signal,

 SInt32 signalStride,

 SInt32 fftStride,

 DSPSplitComplex * tempBuffer,
 UInt32 log2n,
 UInt32 numFFT,
 FFTDirection direction);
```

Parameters

setup
: Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`.The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of the transform function.

signalStride
: Specifies an address stride through the input signals. To process every element of each signal, specify 1 for parameter signalStride; to process every other element, specify 2. The value of signalStride should be 1 for best performance.

fftStride
: The number of elements between the first element of one input signal and the first element of the next (which is also to length of each input signal, measured in elements).

rfftStride
: The number of elements between the first element of one result vector and the next in the output vector `result`.

tempBuffer
: A temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k for best performance. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, `tempBuffer.realp` and `tempBuffer.imagp` should be 32-byte aligned for best performance.

log2n
: The base 2 exponent of the number of elements to process in a single input signal. For example, to process 512 elements, specify 9 for parameter `log2n`. The value of `log2n` must be between 3 and 12, inclusive.

numFFT
: The number of different input signals.

direction
: A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`
: Forward transform

`FFT_INVERSE`
: Inverse transform

Results are undefined for other values of `direction`.

Discussion

The functions allow you to perform Fourier transforms on a number of different input signals at once, using a single call. They can be used for efficient processing of small input signals (less than 512 points). They will work for input signals of 4 points or greater. Each of the input signals processed by a given call must have the same length and address stride. The input signals are concatenated into a single input/output vector, the parameter `signal`.

The functions compute in-place real discrete Fourier transforms of the input signals, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

Criteria to invoke vectorized code:

- The input/output vectors `signal.realp` and `signal.imagp` must be aligned on 16-byte boundaries.
- The value of `signalStride` must be 1.
- The value of `log2n` must be between 3 and 12, inclusive.

If any of these criteria is not satisfied, the function invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability

Available in Mac OS X v10.4 and later.

vDSP_fftm_zriptD

Performs multiple Fourier transform with a single call.

```
void

vDSP_fftm_zriptD (FFTSetupD setup,

 DSPDoubleSplitComplex * signal,

 SInt32 signalStride,

 SInt32 fftStride,

 DSPDoubleSplitComplex * tempBuffer,

 UInt32 log2n,

 UInt32 numFFT,

 FFTDirection direction);
```

Parameters

setup

Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`.The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of the transform function.

signalStride

Specifies an address stride through the input signals. To process every element of each signal, specify 1 for parameter signalStride; to process every other element, specify 2. The value of signalStride should be 1 for best performance.

fftStride

The number of elements between the first element of one input signal and the first element of the next (which is also to length of each input signal, measured in elements).

rfftStride

The number of elements between the first element of one result vector and the next in the output vector `result`.

tempBuffer

A temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k for best performance. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, `tempBuffer.realp` and `tempBuffer.imagp` should be 32-byte aligned for best performance.

log2n

The base 2 exponent of the number of elements to process in a single input signal. For example, to process 512 elements, specify 9 for parameter `log2n`. The value of `log2n` must be between 3 and 12, inclusive.

numFFT

The number of different input signals.

direction

A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`

Forward transform

`FFT_INVERSE`

Inverse transform

Results are undefined for other values of `direction`.

Discussion

The functions allow you to perform Fourier transforms on a number of different input signals at once, using a single call. They can be used for efficient processing of small input signals (less than 512 points). They will work for input signals of 4 points or greater. Each of the input signals processed by a given call must have the same length and address stride. The input signals are concatenated into a single input/output vector, the parameter `signal`.

The functions compute in-place real discrete Fourier transforms of the input signals, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

Criteria to invoke vectorized code:

Double Precision - `vDSP_fftm_zripD` and `vDSP_fftm_zriptD`

No altivec support for double precision. The function always invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_fftm_zrop

Performs multiple Fourier transforms with a single call.

```
void

vDSP_fftm_zrop( FFTSetup setup,

 DSPSplitComplex * signal,

 SInt32 signalStride,

 SInt32 fftStride,

 DSPSplitComplex * result,

 SInt32 resultStride,

 SInt32 rfftStride,

 UInt32 log2n,

 UInt32 numFFT,

 FFTDirection direction);
```

Parameters

setup
Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`.The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of the transform function.

signalStride
Specifies an address stride through input signals . To process every element of each signal, specify a stride of 1; to process every other element, specify 2. The value of `signalStride` should be 1 for best performance.

fftStride
The number of elements between the first element of one input signal and the first element of the next (which is also to length of each input signal, measured in elements).

resultStride
: Specifies an address stride through output vector result. Thus, to process every element, specify a stride of 1; to process every other element, specify 2. The value of `resultStride` should be 1 for best performance.

rfftStride
: The number of elements between the first element of one result vector and the next in the output vector `result`.

log2n
: The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`. The value of `log2n` must be between 3 and 20, inclusive. See also the FFT Limitations sections in the Target chapters of the Developer's Guide.

numFFT
: The number of different input signals.

direction
: A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`
: Forward transform

`FFT_INVERSE`
: Inverse transform

Results are undefined for other values of `direction`.

Discussion
The functions allow you to perform Fourier transforms on a number of different input signals at once, using a single call. They can be used for efficient processing of small input signals (less than 512 points). They will work for input signals of 4 points or greater. Each of the input signals processed by a given call must have the same length and address stride. The input signals are concatenated into a single input vector, the parameter `signal`.

The functions compute out-of-place real discrete Fourier transforms of the input signals, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

Criteria to invoke vectorized code:

- The input vectors `signal.realp` and `signal.imagp` and the output vectors `result.realp` and `result.imagp` must be aligned on 16-byte boundaries.
- The values of `signalStride` and `resultStride` must be 1.
- The value of `log2n` must be between 3 and 12, inclusive.

If any of these criteria is not satisfied, the function invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_fftm_zropD

Performs multiple Fourier transforms with a single call.

```
void

vDSP_fftm_zropD (FFTSetupD setup,

 DSPDoubleSplitComplex * signal,

 SInt32 signalStride,

 SInt32 fftStride,

 DSPDoubleSplitComplex * result,

 SInt32 resultStride,

 SInt32 rfftStride,

 UInt32 log2n,

 UInt32 numFFT,

 FFTDirection direction);
```

Parameters

setup

Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`.The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of the transform function.

signalStride

Specifies an address stride through input signals . To process every element of each signal, specify a stride of 1; to process every other element, specify 2. The value of `signalStride` should be 1 for best performance.

fftStride

The number of elements between the first element of one input signal and the first element of the next (which is also to length of each input signal, measured in elements).

resultStride

Specifies an address stride through output vector result. Thus, to process every element, specify a stride of 1; to process every other element, specify 2. The value of `resultStride` should be 1 for best performance.

rfftStride

The number of elements between the first element of one result vector and the next in the output vector `result`.

log2n

The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`. The value of `log2n` must be between 3 and 20, inclusive. See also the FFT Limitations sections in the Target chapters of the Developer's Guide.

numFFT

The number of different input signals.

direction

A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`

Forward transform

`FFT_INVERSE`

Inverse transform

Results are undefined for other values of `direction`.

Discussion

The functions allow you to perform Fourier transforms on a number of different input signals at once, using a single call. They can be used for efficient processing of small input signals (less than 512 points). They will work for input signals of 4 points or greater. Each of the input signals processed by a given call must have the same length and address stride. The input signals are concatenated into a single input vector, the parameter `signal`.

The functions compute out-of-place real discrete Fourier transforms of the input signals, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability

Available in Mac OS X v10.4 and later.

vDSP_fftm_zropt

Performs multiple Fourier transforms with a single call.

```
void

vDSP_fftm_zropt (FFTSetup setup,

 DSPSplitComplex * signal,

 SInt32 signalStride,

 SInt32 fftStride,

 DSPSplitComplex * result,

 SInt32 resultStride,

 SInt32 rfftStride,
 DSPSplitComplex * tempBuffer,
 UInt32 log2n,
 UInt32 numFFT,
 FFTDirection direction);
```

Parameters

setup

Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`.The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of the transform function.

signalStride

Specifies an address stride through input signals . To process every element of each signal, specify a stride of 1; to process every other element, specify 2. The value of `signalStride` should be 1 for best performance.

fftStride

The number of elements between the first element of one input signal and the first element of the next (which is also to length of each input signal, measured in elements).

resultStride

Specifies an address stride through output vector result. Thus, to process every element, specify a stride of 1; to process every other element, specify 2. The value of `resultStride` should be 1 for best performance.

rfftStride

The number of elements between the first element of one result vector and the next in the output vector `result`.

tempBuffer

A temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k for best performance. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, `tempBuffer.realp` and `tempBuffer.imagp` should be 32-byte aligned for best performance.

log2n

The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`. The value of `log2n` must be between 3 and 20, inclusive. See also the FFT Limitations sections in the Target chapters of the Developer's Guide.

numFFT

The number of different input signals.

`direction`
A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`
Forward transform

`FFT_INVERSE`
Inverse transform

Results are undefined for other values of `direction`.

Discussion

The functions allow you to perform Fourier transforms on a number of different input signals at once, using a single call. They can be used for efficient processing of small input signals (less than 512 points). They will work for input signals of 4 points or greater. Each of the input signals processed by a given call must have the same length and address stride. The input signals are concatenated into a single input vector, the parameter `signal`.

The functions compute out-of-place real discrete Fourier transforms of the input signals, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

Criteria to invoke vectorized code:

- The input vectors `signal.realp` and `signal.imagp` and the output vectors `result.realp` and `result.imagp` must be aligned on 16-byte boundaries.
- The values of `signalStride` and `resultStride` must be 1.
- The value of `log2n` must be between 3 and 12, inclusive.

If any of these criteria is not satisfied, the function invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability

Available in Mac OS X v10.4 and later.

vDSP_fftm_zroptD

Performs multiple Fourier transforms with a single call.

```
void

vDSP_fftm_zroptD (FFTSetupD setup,

 DSPDoubleSplitComplex * signal,

 SInt32 signalStride,

 SInt32 fftStride,

 DSPDoubleSplitComplex * result,
 SInt32 resultStride,
 SInt32 rfftStride,
 DSPDoubleSplitComplex * tempBuffer,
 UInt32 log2n,
 UInt32 numFFT,
 FFTDirection direction);
```

Parameters

setup

Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`.The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of the transform function.

signalStride

Specifies an address stride through input signals . To process every element of each signal, specify a stride of 1; to process every other element, specify 2. The value of `signalStride` should be 1 for best performance.

fftStride

The number of elements between the first element of one input signal and the first element of the next (which is also to length of each input signal, measured in elements).

resultStride

Specifies an address stride through output vector result. Thus, to process every element, specify a stride of 1; to process every other element, specify 2. The value of `resultStride` should be 1 for best performance.

rfftStride

The number of elements between the first element of one result vector and the next in the output vector `result`.

tempBuffer

A temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k for best performance. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, `tempBuffer.realp` and `tempBuffer.imagp` should be 32-byte aligned for best performance.

log2n

The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`. The value of `log2n` must be between 3 and 20, inclusive. See also the FFT Limitations sections in the Target chapters of the Developer's Guide.

numFFT

The number of different input signals.

direction
A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`
Forward transform

`FFT_INVERSE`
Inverse transform

Results are undefined for other values of `direction`.

Discussion
The functions allow you to perform Fourier transforms on a number of different input signals at once, using a single call. They can be used for efficient processing of small input signals (less than 512 points). They will work for input signals of 4 points or greater. Each of the input signals processed by a given call must have the same length and address stride. The input signals are concatenated into a single input vector, the parameter `signal`.

The functions compute out-of-place real discrete Fourier transforms of the input signals, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_fft_zip

Computes an in-place complex discrete Fourier transform of the input/output vector signal, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

```
void
vDSP_fft_zip (FFTSetup setup,
    DSPSplitComplex * signal,
    SInt32 signalStride,
    UInt32 log2n,
    FFTDirection direction);
```

Parameters

setup
Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`.The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of the transform function.

signalStride
Specifies an address stride through the input/output vector signal. To process every element of the vector, specify 1 for parameter `signalStride`; to process every other element, specify 2. The value of `signalStride` should be 1 for best performance.

log2n
The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`. The value of `log2n` must be between 2 and 20, inclusive.

direction
A forward/inverse directional flag, and must specify one of the following:

FFT_FORWARD
Forward transform

FFT_INVERSE
Inverse transform

Results are undefined for other values of `direction`.

Discussion
This performs the operation

If F = 1 $\quad C_m = \mathrm{FDFT}(C_m) \quad$ If F = -1 $\quad C_m = \mathrm{IDFT}\ (C_m) \bullet N \quad$ m = {0, N-1}

$$\mathrm{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \mathrm{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

Criteria to invoke vectorized code:

- Input/output vectors `tempBuffer.realp` and `tempBuffer.imagp` and temporary vectors `tempBuffer.realp` and `tempBuffer.imagp` must be aligned on 16-byte boundaries.
- The value of `signalStride` must be 1.

If any of these criteria is not satisfied, the function invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_fft_zipD

Computes an in-place complex discrete Fourier transform of the input/output vector signal, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

```
void
vDSP_fft_zipD (FFTSetupD setup,
    DSPDoubleSplitComplex * signal,
    SInt32 signalStride,
    UInt32 log2n,
    FFTDirection direction);
```

Parameters

`setup`
Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`.The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of the transform function.

`signalStride`
Specifies an address stride through the input/output vector signal. To process every element of the vector, specify 1 for parameter `signalStride`; to process every other element, specify 2. The value of `signalStride` should be 1 for best performance.

`log2n`
The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`. The value of `log2n` must be between 2 and 20, inclusive.

`direction`
A forward/inverse directional flag, and must specify one of the following:

FFT_FORWARD
Forward transform

FFT_INVERSE
Inverse transform

Results are undefined for other values of `direction`.

Discussion
This performs the operation

If F = 1 $\quad C_m = \text{FDFT}(C_m) \qquad$ If F = -1 $\quad C_m = \text{IDFT}(C_m) \bullet N \qquad m = \{0, N\text{-}1\}$

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_fft_zipt

Computes an in-place complex discrete Fourier transform of the input/output vector signal, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

```
void
vDSP_fft_zipt (FFTSetup setup,
    DSPSplitComplex * signal,
    SInt32 signalStride,
    DSPSplitComplex * tempBuffer,
    UInt32 log2n,
    FFTDirection direction);
```

Parameters

setup

Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`.The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of the transform function.

signalStride

Specifies an address stride through the input/output vector signal. To process every element of the vector, specify 1 for parameter `signalStride`; to process every other element, specify 2. The value of `signalStride` should be 1 for best performance.

tempBuffer

A temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k for best performance. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, `tempBuffer.realp` and `tempBuffer.imagp` should be 32-byte aligned for best performance.

log2n

The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`. The value of `log2n` must be between 2 and 20, inclusive.

direction

A forward/inverse directional flag, and must specify one of the following:

FFT_FORWARD

Forward transform

FFT_INVERSE

Inverse transform

Results are undefined for other values of `direction`.

Discussion

This performs the operation

If F = 1 $\quad C_m = \text{FDFT}(C_m) \quad$ If F = -1 $\quad C_m = \text{IDFT}(C_m) \cdot N \quad$ m = {0, N-1}

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \cdot e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \cdot e^{(j2\pi nm)/N}$$

Criteria to invoke vectorized code:

- Input/output vectors `tempBuffer.realp` and `tempBuffer.imagp` and temporary vectors `tempBuffer.realp` and `tempBuffer.imagp` must be aligned on 16-byte boundaries.
- The value of `signalStride` must be 1.

If any of these criteria is not satisfied, the function invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_fft_ziptD

Computes an in-place complex discrete Fourier transform of the input/output vector signal, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

```
void
vDSP_fft_ziptD (FFTSetupD setup,
    DSPDoubleSplitComplex * signal,
    SInt32 signalStride,
    DSPDoubleSplitComplex * tempBuffer,
    UInt32 log2n,
    FFTDirection direction);
```

Parameters

setup
Points to a structure initialized by a prior call to the FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`.The value supplied as parameter `log2n` of the earlier call to the setup function must equal or exceed the value supplied as parameter `log2n` of the transform function.

signalStride
Specifies an address stride through the input/output vector signal. To process every element of the vector, specify 1 for parameter `signalStride`; to process every other element, specify 2. The value of `signalStride` should be 1 for best performance.

tempBuffer

A temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k for best performance. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, `tempBuffer.realp` and `tempBuffer.imagp` should be 32-byte aligned for best performance.

log2n

The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`. The value of `log2n` must be between 2 and 20, inclusive.

direction

A forward/inverse directional flag, and must specify one of the following:

FFT_FORWARD
Forward transform

FFT_INVERSE
Inverse transform

Results are undefined for other values of `direction`.

Discussion

This performs the operation

$$\text{If F = 1} \quad C_m = \text{FDFT}(C_m) \qquad \text{If F = -1} \quad C_m = \text{IDFT}\,(C_m) \cdot N \qquad \text{m} = \{0, \text{N-1}\}$$

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \cdot e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \cdot e^{(j2\pi nm)/N}$$

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability

Available in Mac OS X v10.4 and later.

vDSP_fft_zop

Computes an out-of-place complex discrete Fourier transform of the input vector, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

```
void
vDSP_fft_zop (FFTSetup setup,
    DSPSplitComplex * signal,
    SInt32 signalStride,
    DSPSplitComplex * result,
    SInt32 resultStride,
    UInt32 log2n,
    FFTDirection direction);
```

Parameters

setup

Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the value supplied as parameter `log2n` of the transform function.

signalStride

Specifies an address stride through input vector signal. Parameter resultStride specifies an address stride through output vector result. Thus, to process every element, specify a signalStride of 1; to process every other element, specify 2. The values of signalStride and resultStride should be 1 for best performance.

For `vDSP_fft_zopt` and `vDSP_fft_zoptD`, parameter tempBuffer is a temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, tempBuffer`.realp` and tempBuffer`.imagp` should be 32-byte aligned for best performance.

log2n

The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`. See also the FFT Limitations sections in the Target chapters of the Developer's Guide. The value of `log2n` must be between 2 and 20, inclusive.

direction

A forward/inverse directional flag, and must specify one of the following:

FFT_FORWARD

Forward transform

FFT_INVERSE

Inverse transform

Results are undefined for other values of `direction`.

Discussion

This performs the operation

If F = 1 $C_m = \text{FDFT}(A_m)$ If F = -1 $C_m = \text{IDFT}(A_m) \bullet N$ m = {0, N-1}

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

Criteria to invoke vectorized code:

- The input vectors `signal.realp` and `signal.imagp`, the output vectors `result.realp` and `result.imagp`, and the buffer vectors `tempBuffer.realp` and tempBuffer`.imagp` must be aligned on 16-byte boundaries.
- The values of `signalStride` and `resultStride` must be 1.
- The value of `log2n` must be between 2 and 20, inclusive.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_fft_zopD

Computes an out-of-place complex discrete Fourier transform of the input vector, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

```
void
vDSP_fft_zopD (FFTSetupD setup,
    DSPDoubleSplitComplex * signal,
    SInt32 signalStride,
    DSPDoubleSplitComplex * result,
    SInt32 resultStride,
    UInt32 log2n,
    FFTDirection direction);
```

Parameters

`setup`
Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the value supplied as parameter `log2n` of the transform function.

`signalStride`
Specifies an address stride through input vector signal. Parameter resultStride specifies an address stride through output vector result. Thus, to process every element, specify a signalStride of 1; to process every other element, specify 2. The values of signalStride and resultStride should be 1 for best performance.

`log2n`
The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`. See also the FFT Limitations sections in the Target chapters of the Developer's Guide. The value of `log2n` must be between 2 and 20, inclusive.

direction
A forward/inverse directional flag, and must specify one of the following:

FFT_FORWARD
Forward transform

FFT_INVERSE
Inverse transform

Results are undefined for other values of `direction`.

Discussion
This performs the operation

If F = 1 $C_m = \mathrm{FDFT}(A_m)$ If F = -1 $C_m = \mathrm{IDFT}\ (A_m) \bullet N$ m = {0, N-1}

$$\mathrm{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \mathrm{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_fft_zopt

Computes an out-of-place complex discrete Fourier transform of the input vector, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

```
void
vDSP_fft_zopt (FFTSetup setup,
    DSPSplitComplex * signal,
    SInt32 signalStride,
    DSPSplitComplex * result,
    SInt32 resultStride,
    DSPSplitComplex * tempBuffer,
    UInt32 log2n,
    FFTDirection direction);
```

Parameters

setup
Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the value supplied as parameter `log2n` of the transform function.

`signalStride`

Specifies an address stride through input vector signal. Parameter resultStride specifies an address stride through output vector result. Thus, to process every element, specify a signalStride of 1; to process every other element, specify 2. The values of signalStride and resultStride should be 1 for best performance.

`tempBuffer`

A temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, tempBuffer`.realp` and tempBuffer`.imagp` should be 32-byte aligned for best performance.

`log2n`

The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`. See also the FFT Limitations sections in the Target chapters of the Developer's Guide. The value of `log2n` must be between 2 and 20, inclusive.

`direction`

A forward/inverse directional flag, and must specify one of the following:

FFT_FORWARD

Forward transform

FFT_INVERSE

Inverse transform

Results are undefined for other values of `direction`.

Discussion

This performs the operation

$$\text{If } F = 1 \quad C_m = \text{FDFT}(A_m) \qquad \text{If } F = \text{-}1 \quad C_m = \text{IDFT}\,(A_m) \bullet N \qquad m = \{0, N\text{-}1\}$$

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

Criteria to invoke vectorized code:

- The input vectors `signal.realp` and `signal.imagp`, the output vectors `result.realp` and `result.imagp`, and the buffer vectors `tempBuffer.realp` and tempBuffer`.imagp` must be aligned on 16-byte boundaries.
- The values of `signalStride` and `resultStride` must be 1.
- The value of `log2n` must be between 2 and 20, inclusive.

If any of these criteria is not satisfied, the function invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability

Available in Mac OS X v10.4 and later.

vDSP_fft_zoptD

Computes an out-of-place complex discrete Fourier transform of the input vector, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

```
void
vDSP_fft_zoptD (FFTSetupD setup,
    DSPDoubleSplitComplex * signal,
    DSPDoubleSplitComplex * result,
    SInt32 signalStride,
    SInt32 resultStride,
    DSPDoubleSplitComplex * tempBuffer,
    UInt32 log2n,
    FFTDirection direction);
```

Parameters

setup

Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the value supplied as parameter `log2n` of the transform function.

signalStride

Specifies an address stride through input vector signal. Parameter resultStride specifies an address stride through output vector result. Thus, to process every element, specify a signalStride of 1; to process every other element, specify 2. The values of signalStride and resultStride should be 1 for best performance.

tempBuffer

A temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, tempBuffer`.realp` and tempBuffer`.imagp` should be 32-byte aligned for best performance.

log2n

The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`. See also the FFT Limitations sections in the Target chapters of the Developer's Guide. The value of `log2n` must be between 2 and 20, inclusive.

direction

A forward/inverse directional flag, and must specify one of the following:

FFT_FORWARD

Forward transform

FFT_INVERSE

Inverse transform

Results are undefined for other values of `direction`.

Discussion

This performs the operation

If F = 1 $\quad C_m = \text{FDFT}(A_m)$ $\quad$ If F = -1 $\quad C_m = \text{IDFT}(A_m) \bullet N$ $\quad$ m = {0, N-1}

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N} \sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability

Available in Mac OS X v10.4 and later.

vDSP_fft_zrip

Computes an in-place real discrete Fourier transform, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

```
void
vDSP_fft_zrip (FFTSetup setup,
    DSPSplitComplex * signal,
    SInt32 signalStride,
    UInt32 log2n,
    FFTDirection direction);
```

Parameters

setup

Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the value supplied as parameter `log2n` of the transform function.

signalStride

Specifies an address stride through the input/output vector . To process every element of the vector, specify 1 for parameter signalStride; to process every other element, specify 2. The value of signalStride should be 1 for best performance.

log2n

The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`. The value of `log2n` must be between 3 and 20, inclusive. See also the FFT Limitations sections in the Target chapters of the Developer's Guide.

`direction`
A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`
Forward transform

`FFT_INVERSE`
Inverse transform

Results are undefined for other values of `direction`.

Discussion
Forward transforms read real input and write packed complex output. Inverse transforms read packed complex input and write real output. As a result of packing the frequency-domain data, time-domain data and its equivalent frequency-domain data have the same storage requirements.

If F = 1 $C_m = \mathrm{RDFT}(C_m) \cdot 2$ If F = -1 $C_m = \mathrm{IDFT}\ (C_m) \cdot N$ m = {0, N-1}

$$\mathrm{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \cdot e^{(-j2\pi nm)/N} \qquad \mathrm{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \cdot e^{(j2\pi nm)/N}$$

Real data is stored in split complex form, with odd reals stored on the imaginary side of the split complex form and even reals in stored on the real side. See the section "Data Packing for Real FFTs" (page 20).

Criteria to invoke vectorized code:

- The input/output vectors `signal.realp` and `signal.imagp` must be aligned on 16-byte boundaries.
- The value of `signalStride` must be 1.
- The value of `log2n` must be between 3 and 20, inclusive.

If any of these criteria is not satisfied, the function invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_fft_zripD

Computes an in-place real discrete Fourier transform, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

```
void
vDSP_fft_zripD (FFTSetupD setup,
    DSPDoubleSplitComplex * signal,
    SInt32 signalStride,
    UInt32 log2n,
    FFTDirection direction);
```

Parameters

setup
Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the value supplied as parameter `log2n` of the transform function.

signalStride
Specifies an address stride through the input/output vector . To process every element of the vector, specify 1 for parameter `signalStride`; to process every other element, specify 2. The value of `signalStride` should be 1 for best performance.

log2n
The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`. The value of `log2n` must be between 3 and 20, inclusive. See also the FFT Limitations sections in the Target chapters of the Developer's Guide.

direction
A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`
Forward transform

`FFT_INVERSE`
Inverse transform

Results are undefined for other values of `direction`.

Discussion

Forward transforms read real input and write packed complex output. Inverse transforms read packed complex input and write real output. As a result of packing the frequency-domain data, time-domain data and its equivalent frequency-domain data have the same storage requirements.

If F = 1 $C_m = \mathrm{RDFT}(C_m) \bullet 2$ If F = -1 $C_m = \mathrm{IDFT}\ (C_m) \bullet N$ m = {0, N-1}

$$\mathrm{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \mathrm{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

Real data is stored in split complex form, with odd reals stored on the imaginary side of the split complex form and even reals in stored on the real side. See the section "Data Packing for Real FFTs" (page 20).

Criteria to invoke vectorized code:

- The input/output vectors `signal.realp` and `signal.imagp` must be aligned on 16-byte boundaries.

- The value of `signalStride` must be 1.
- The value of `log2n` must be between 3 and 20, inclusive.

If any of these criteria is not satisfied, the function invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_fft_zript

Computes an in-place real discrete Fourier transform, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

```
void
vDSP_fft_zript (FFTSetup setup,
    DSPSplitComplex * signal,
    SInt32 signalStride,
    DSPSplitComplex * tempBuffer,
    UInt32 log2n,
    FFTDirection direction);
```

Parameters

`setup`
Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the value supplied as parameter `log2n` of the transform function.

`signalStride`
Specifies an address stride through the input/output vector . To process every element of the vector, specify 1 for parameter signalStride; to process every other element, specify 2. The value of signalStride should be 1 for best performance.

`tempBuffer`
A temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k for best performance. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, tempBuffer`.realp` and tempBuffer`.imagp` should be 32-byte aligned for best performance.

`log2n`
The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`. The value of `log2n` must be between 3 and 20, inclusive. See also the FFT Limitations sections in the Target chapters of the Developer's Guide.

`direction`
A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`
Forward transform

`FFT_INVERSE`
Inverse transform

Results are undefined for other values of `direction`.

Discussion
Forward transforms read real input and write packed complex output. Inverse transforms read packed complex input and write real output. As a result of packing the frequency-domain data, time-domain data and its equivalent frequency-domain data have the same storage requirements.

If F = 1 $C_m = \text{RDFT}(C_m) \cdot 2$ If F = -1 $C_m = \text{IDFT}\ (C_m) \cdot N$ m = {0, N-1}

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \cdot e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \cdot e^{(j2\pi nm)/N}$$

Real data is stored in split complex form, with odd reals stored on the imaginary side of the split complex form and even reals in stored on the real side. See the section "Data Packing for Real FFTs" (page 20).

Criteria to invoke vectorized code:

- The input/output vectors `signal.realp` and `signal.imagp` must be aligned on 16-byte boundaries.
- The value of `signalStride` must be 1.
- The value of `log2n` must be between 3 and 20, inclusive.

If any of these criteria is not satisfied, the function invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_fft_zriptD

Computes an in-place real discrete Fourier transform, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

```
void
vDSP_fft_zriptD (FFTSetupD setup,
    DSPDoubleSplitComplex * signal,
    SInt32 signalStride,
    DSPDoubleSplitComplex * tempBuffer,
    UInt32 log2n,
    FFTDirection direction);
```

Parameters

setup

Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the value supplied as parameter `log2n` of the transform function.

signalStride

Specifies an address stride through the input/output vector . To process every element of the vector, specify 1 for parameter signalStride; to process every other element, specify 2. The value of signalStride should be 1 for best performance.

tempBuffer

A temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k for best performance. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, tempBuffer`.realp` and tempBuffer`.imagp` should be 32-byte aligned for best performance.

log2n

The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`. The value of `log2n` must be between 3 and 20, inclusive. See also the FFT Limitations sections in the Target chapters of the Developer's Guide.

direction

A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`

Forward transform

`FFT_INVERSE`

Inverse transform

Results are undefined for other values of `direction`.

Discussion

Forward transforms read real input and write packed complex output. Inverse transforms read packed complex input and write real output. As a result of packing the frequency-domain data, time-domain data and its equivalent frequency-domain data have the same storage requirements.

If F = 1 $C_m = \text{RDFT}(C_m) \bullet 2$ If F = -1 $C_m = \text{IDFT}\ (C_m) \bullet N$ m = {0, N-1}

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

Real data is stored in split complex form, with odd reals stored on the imaginary side of the split complex form and even reals in stored on the real side. See the section "Data Packing for Real FFTs" (page 20).

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_fft_zrop

Computes an out-of-place real discrete Fourier transform, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

```
void
vDSP_fft_zrop (FFTSetup setup,
    DSPSplitComplex * signal,
    SInt32 signalStride,
    DSPSplitComplex * result,
    SInt32 resultStride,
    UInt32 log2n,
    FFTDirection direction);
```

Parameters

setup
Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the value supplied as parameter `log2n` of the transform function.

signalStride
Specifies an address stride through input vector signal. Thus, to process every element, specify a stride of 1; to process every other element, specify 2. The value of `signalStride` should be 1 for best performance.

resultStride
Specifies an address stride through output vector result. Thus, to process every element, specify a stride of 1; to process every other element, specify 2. The value of `resultStride` should be 1 for best performance.

log2n
The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`. The value of `log2n` must be between 3 and 20, inclusive. See also the FFT Limitations sections in the Target chapters of the Developer's Guide.

direction
A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`
Forward transform

`FFT_INVERSE`
Inverse transform

Results are undefined for other values of `direction`.

Discussion
This performs the operation

If F = 1 $\quad C_m = \mathrm{RDFT}(A_m) \cdot 2 \qquad$ If F = -1 $\quad C_m = \mathrm{IDFT}\ (A_m) \cdot N \qquad$ m = {0, N-1}

$$\mathrm{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \cdot e^{(-j2\pi nm)/N} \qquad \mathrm{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \cdot e^{(j2\pi nm)/N}$$

Forward transforms read real input and write packed complex output. Inverse transforms read packed complex input and write real output. As a result of packing the frequency-domain data, time-domain data and its equivalent frequency-domain data have the same storage requirements; see "Data Packing for Real FFTs" (page 20). Real data is stored in split complex form, with odd reals stored on the imaginary side of the split complex form and even reals in stored on the real side.

Criteria to invoke vectorized code:

- The input vectors `signal.realp` and `signal.imagp` and the output vectors `result.realp` and `result.imagp` must be aligned on 16-byte boundaries.
- The values of `signalStride` and `resultStride` must be 1.
- The value of `log2n` must be between 3 and 20, inclusive.

If any of these criteria is not satisfied, the function invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_fft_zropD

Computes an out-of-place real discrete Fourier transform, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

```
void
vDSP_fft_zropD (FFTSetupD setup,
    DSPDoubleSplitComplex * signal,
    SInt32 signalStride,
    DSPDoubleSplitComplex * result,
    SInt32 resultStride,
    UInt32 log2n,
    FFTDirection direction)
```

Parameters

setup

Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the value supplied as parameter `log2n` of the transform function.

signalStride

Specifies an address stride through input vector signal. Thus, to process every element, specify a stride of 1; to process every other element, specify 2. The value of `signalStride` should be 1 for best performance.

resultStride

Specifies an address stride through output vector result. Thus, to process every element, specify a stride of 1; to process every other element, specify 2. The value of `resultStride` should be 1 for best performance.

log2n

The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`. The value of `log2n` must be between 3 and 20, inclusive. See also the FFT Limitations sections in the Target chapters of the Developer's Guide.

direction

A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`

Forward transform

`FFT_INVERSE`

Inverse transform

Results are undefined for other values of `direction`.

Discussion

This performs the operation

If F = 1 $C_m = \text{RDFT}(A_m) \bullet 2$ If F = -1 $C_m = \text{IDFT}\ (A_m) \bullet N$ m = {0, N-1}

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

Forward transforms read real input and write packed complex output. Inverse transforms read packed complex input and write real output. As a result of packing the frequency-domain data, time-domain data and its equivalent frequency-domain data have the same storage requirements; see "Data Packing for Real FFTs" (page 20). Real data is stored in split complex form, with odd reals stored on the imaginary side of the split complex form and even reals in stored on the real side.

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_fft_zropt

Computes an out-of-place real discrete Fourier transform, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

```
void
vDSP_fft_zropt (FFTSetup setup,
    DSPSplitComplex * signal,
    SInt32 signalStride,
    DSPSplitComplex * result,
    SInt32 resultStride,
    DSPSplitComplex * tempBuffer,
    UInt32 log2n,
    FFTDirection direction);
```

Parameters

setup
Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the value supplied as parameter `log2n` of the transform function.

signalStride
Specifies an address stride through input vector signal. Thus, to process every element, specify a stride of 1; to process every other element, specify 2. The value of `signalStride` should be 1 for best performance.

resultStride
Specifies an address stride through output vector result. Thus, to process every element, specify a stride of 1; to process every other element, specify 2. The value of `resultStride` should be 1 for best performance.

tempBuffer
A temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k for best performance. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, tempBuffer.`realp` and tempBuffer.`imagp` should be 32-byte aligned for best performance.

`log2n`
: The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`. The value of `log2n` must be between 3 and 20, inclusive. See also the FFT Limitations sections in the Target chapters of the Developer's Guide.

`direction`
: A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`
: Forward transform

`FFT_INVERSE`
: Inverse transform

Results are undefined for other values of `direction`.

Discussion

This performs the operation

If F = 1 $\quad C_m = \text{RDFT}(A_m) \bullet 2 \qquad$ If F = -1 $\quad C_m = \text{IDFT}\ (A_m) \bullet N \qquad$ m = {0, N-1}

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

Forward transforms read real input and write packed complex output. Inverse transforms read packed complex input and write real output. As a result of packing the frequency-domain data, time-domain data and its equivalent frequency-domain data have the same storage requirements; see "Data Packing for Real FFTs" (page 20). Real data is stored in split complex form, with odd reals stored on the imaginary side of the split complex form and even reals in stored on the real side.

Criteria to invoke vectorized code:

- The input vectors `signal.realp` and `signal.imagp` and the output vectors `result.realp` and `result.imagp` must be aligned on 16-byte boundaries.
- The values of `signalStride` and `resultStride` must be 1.
- The value of `log2n` must be between 3 and 20, inclusive.

If any of these criteria is not satisfied, the function invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability

Available in Mac OS X v10.4 and later.

vDSP_fft_zroptD

Computes an out-of-place real discrete Fourier transform, either from the time domain to the frequency domain (forward) or from the frequency domain to the time domain (inverse).

```
void
vDSP_fft_zroptD (FFTSetupD setup,
    DSPDoubleSplitComplex * signal,
    SInt32 signalStride,
    DSPDoubleSplitComplex * result,
    SInt32 resultStride,
    DSPDoubleSplitComplex * tempBuffer,
    UInt32 log2n,
    FFTDirection direction);
```

Parameters

setup

Points to a structure initialized by a prior call to FFT weights array function `vDSP_create_fftsetup` or `vDSP_create_fftsetupD`. The value supplied as parameter `log2n` of the setup function must equal or exceed the value supplied as parameter `log2n` of the transform function.

signalStride

Specifies an address stride through input vector signal. Thus, to process every element, specify a stride of 1; to process every other element, specify 2. The value of `signalStride` should be 1 for best performance.

resultStride

Specifies an address stride through output vector result. Thus, to process every element, specify a stride of 1; to process every other element, specify 2. The value of `resultStride` should be 1 for best performance.

tempBuffer

A temporary vector used for storing interim results. The size of temporary memory for each part (real and imaginary) is the lower value of 4*n or 16k for best performance. Or you can simply pass the buffer of size 2^(log2n) for each part (real and imaginary). If possible, tempBuffer`.realp` and tempBuffer`.imagp` should be 32-byte aligned for best performance.

log2n

The base 2 exponent of the number of elements to process. For example, to process 1024 elements, specify 10 for parameter `log2n`. The value of `log2n` must be between 3 and 20, inclusive. See also the FFT Limitations sections in the Target chapters of the Developer's Guide.

direction

A forward/inverse directional flag, and must specify one of the following:

`FFT_FORWARD`

Forward transform

`FFT_INVERSE`

Inverse transform

Results are undefined for other values of `direction`.

Discussion

This performs the operation

If F = 1 $C_m = \text{RDFT}(A_m) \bullet 2$ If F = -1 $C_m = \text{IDFT}(A_m) \bullet N$ m = {0, N-1}

$$\text{FDFT}(X_m) = \sum_{n=0}^{N-1} X_n \bullet e^{(-j2\pi nm)/N} \qquad \text{IDFT}(X_m) = \frac{1}{N}\sum_{n=0}^{N-1} X_n \bullet e^{(j2\pi nm)/N}$$

Forward transforms read real input and write packed complex output. Inverse transforms read packed complex input and write real output. As a result of packing the frequency-domain data, time-domain data and its equivalent frequency-domain data have the same storage requirements; see "Data Packing for Real FFTs" (page 20). Real data is stored in split complex form, with odd reals stored on the imaginary side of the split complex form and even reals in stored on the real side.

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

See also functions "vDSP_create_fftsetup" (page 44), "vDSP_create_fftsetupD" (page 45), "vDSP_destroy_fftsetup" (page 50), "vDSP_destroy_fftsetupD" (page 51), and Chapter 2, "Using Fourier Transforms."

Availability
Available in Mac OS X v10.4 and later.

vDSP_hamm_window

Creates a Hamming window.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_hamm_window (float * C,
    unsigned int N,
    int FLAG);
```

Discussion

$$C_n = 0.54 - 0.46\cos\frac{2\pi n}{N} \qquad n = \{0, N-1\}$$

`vDSP_hamm_window` creates a single-precision Hamming window function *C*, which can be multiplied by a vector using `vDSP_vmul` . Specify the `vDSP_HALF_WINDOW` flag to create only the first (n+1)/2 points, or 0 (zero) for full size window.

See also `vDSP_vmul`.

Important: The constant `vDSP_HALF_WINDOW` is not declared in the vDSP header file. For the initial release in Mac OS X v10.4, declare it in a separate header with the value 1.

Availability
Available in Mac OS X v10.4 and later.

vDSP_hamm_windowD

Creates a Hamming window.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_hamm_windowD (double * C,
    unsigned int N,
    int FLAG);
```

Discussion

$$C_n = 0.54 - 0.46 \cos \frac{2\pi n}{N} \qquad n = \{0, N\text{-}1\}$$

`vDSP_hamm_windowD` creates a double-precision Hamming window function `C`, which can be multiplied by a vector using `vDSP_vmulD` . Specify the `vDSP_HALF_WINDOW` flag to create only the first (n+1)/2 points, or 0 (zero) for full size window.

See also `vDSP_vmulD`.

Important: The constant `vDSP_HALF_WINDOW` is not declared in the vDSP header file. For the initial release in Mac OS X v10.4, declare it in a separate header with the value 1.

Availability

Available in Mac OS X v10.4 and later.

vDSP_hann_window

Creates a Hanning window.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_hann_window (float * C,
    unsigned int N,
    int FLAG);
```

Discussion

$$C_n = W\left(1.0 - \cos \frac{2\pi n}{N}\right) \qquad n = \{0, N\text{-}1\}$$

`vDSP_hann_window` creates a single-precision Hanning window function `C`, which can be multiplied by a vector using `vDSP_vmul` .

The FLAG parameter can have the following values:

- `vDSP_HANN_DENORM` creates a denormalized window.
- *`vDSP_HANN_NORM`* creates a normalized window.
- *`vDSP_HALF_WINDOW`* creates only the first (N+1)/2 points.

vDSP_HALF_WINDOWvDSP_HALF_WINDOW can ORed with any of the other values (i.e., using the C operator |.

See also vDSP_vmul.

Important: The constants vDSP_HANN_DENORM, vDSP_HANN_NORM, and vDSP_HALF_WINDOW are not declared in the vDSP header file. For the initial release in Mac OS X v10.4, declare them in a separate header with the values vDSP_HANN_DENORM = 0, vDSP_HANN_NORM = 2, and vDSP_HALF_WINDOW = 1.

Availability
Available in Mac OS X v10.4 and later.

vDSP_hann_windowD

Creates a Hanning window.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_hann_windowD (double * C,
    unsigned int N,
    int FLAG);
```

Discussion

$$C_n = W\left(1.0 - \cos\frac{2\pi n}{N}\right) \qquad n = \{0, N-1\}$$

vDSP_hann_window creates a double-precision Hanning window function *C*, which can be multiplied by a vector using vDSP_vmul .

The FLAG parameter can have the following values:

- vDSP_HANN_DENORM creates a denormalized window.
- *vDSP_HANN_NORM* creates a normalized window.
- *vDSP_HALF_WINDOW* creates only the first (N+1)/2 points.

vDSP_HALF_WINDOWvDSP_HALF_WINDOW can ORed with any of the other values (i.e., using the C operator |.

See also vDSP_vmul.

Important: The constants vDSP_HANN_DENORM, vDSP_HANN_NORM, and vDSP_HALF_WINDOW are not declared in the vDSP header file. For the initial release in Mac OS X v10.4, declare them in a separate header with the values vDSP_HANN_DENORM = 0, vDSP_HANN_NORM = 2, and vDSP_HALF_WINDOW = 1.

Availability
Available in Mac OS X v10.4 and later.

vDSP_imgfir

Filters an image by performing a two-dimensional convolution with a kernel.

```
void
vDSP_imgfir (float * A,

 SInt32 M,

 SInt32 * N,

 float * B,

 float * C

 SInt32 * P

 SInt32 * Q);
```

Parameters

A
: Single-precision input matrix

B
: Single-precision filter kernel

M
: Number of rows in *A* and in *C*

N
: Number of columns in *A* and in *C*

C
: Single-precision result matrix

P
: Number of rows in *B*

Q
: Number of columns in *B*

Discussion

The image is given by the input matrix A. It has M rows and N columns. M and N must both be odd and greater than or equal to 5.

$$C_{(m+(P-1)/2,n+(Q-1)/2)} = \sum_{p=0}^{P-1}\sum_{q=0}^{Q-1} A_{(m+p,n+q)} \bullet B_{(p,q)} \qquad \text{m = \{0, M-P\} and n = \{0, N-Q\}}$$

B is the filter kernel. It has P rows and Q columns. P must be greater than or equal to Q.

M must be even and greater than P. *N* must be even and greater than *Q*.

The filtered image is placed in the output matrix `C`. The functions pad the perimeter of the output image with a border of (`P`-1)/2 rows of zeros on the top and bottom and (`Q`-1)/2 columns of zeros on the left and right.

Criteria to invoke vectorized code:

- The parameters `A`, `B`, and `C` must be 16-byte aligned.
- `N` must be greater than or equal to 20.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_imgfirD

Filters an image by performing a two-dimensional convolution with a kernel.

```
void
vDSP_imgfirD (double * A,

 SInt32 M,

 SInt32 * N,

 double * B,

 double * C

 SInt32 * P

 SInt32 * Q);
```

Parameters

A
    Double-precision input matrix

B
    Double-precision filter kernel

M
    Number of rows in *A* and in *C*

N
    Number of columns in *A* and in *C*

C
    Double-precision result matrix

P
    Number of rows in *B*

Q
: Number of columns in *B*

Discussion

The image is given by the input matrix A. It has M rows and N columns. M and N must both be odd and greater than or equal to 5.

$$C_{(m+(P\text{-}1)/2,n+(Q\text{-}1)/2)} = \sum_{p=0}^{P-1}\sum_{q=0}^{Q-1} A_{(m+p,n+q)} \cdot B_{(p,q)} \qquad \text{m = \{0, M-P\} and n = \{0, N-Q\}}$$

B is the filter kernel. It has P rows and Q columns. P must be greater than or equal to Q.

M must be even and greater than P. *N* must be even and greater than *Q*.

The filtered image is placed in the output matrix C. The functions pad the perimeter of the output image with a border of (P-1)/2 rows of zeros on the top and bottom and (Q-1)/2 columns of zeros on the left and right.

Criteria to invoke vectorized code:

- The parameters A, B, and C must be 16-byte aligned.
- N must be greater than or equal to 20.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability

Available in Mac OS X v10.4 and later.

vDSP_maxmgv

Vector maximum magnitude.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_maxmgv (float * A,

 int I,

 float * C,

 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

C

Output scalar

N

Count

Discussion

This performs the operation

$c = |A_0| \quad \text{If} \quad c < |A_{ni}| \quad \text{then} \quad c = |A_{ni}| \qquad n = \{1, N\text{-}1\}$

Finds the element with the greatest magnitude in vector *A* and copies this value to scalar *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_maxmgvD

Vector maximum magnitude.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_maxmgvD (double * A,

 int I,

 double * C,

 unsigned int N);
```

Parameters

A

Double-precision real input vector

I

Stride for *A*

C

Output scalar

N

Count

Discussion

This performs the operation

$c = |A_0| \quad \text{If} \quad c < |A_{ni}| \quad \text{then} \quad c = |A_{ni}| \qquad n = \{1, N\text{-}1\}$

Finds the element with the greatest magnitude in vector *A* and copies this value to scalar *C*.

Availability
Available in Mac OS X v10.4 and later.

vDSP_maxmgvi

Vector maximum magnitude with index.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_maxmgvi (float * A,

 int I,

 float * C,

 int * IC,

 unsigned int N);
```

Parameters

A
Single-precision real input vector

I
Stride for *A*

C
Output scalar

IC
Output scalar

N
Count

Discussion
This performs the operation

$$\begin{cases} c = |A_0| \\ d = 0 \end{cases} \quad \text{If} \quad c < |A_{ni}| \quad \text{then} \quad \begin{cases} c = |A_{ni}| \\ d = ni \end{cases} \qquad n = \{1, N\text{-}1\}$$

Copies the element with the greatest magnitude from real vector *A* to real scalar *C*, and writes its zero-based index to integer scalar *IC*. The index is the actual array index, not the pre-stride index. If vector *A* contains more than one instance of the maximum magnitude, *IC* contains the index of the first instance.

Availability
Available in Mac OS X v10.4 and later.

vDSP_maxmgviD

Vector maximum magnitude with index.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_maxmgviD (double * A,

 int I,

 double * C,

 int * IC,

 unsigned int N);
```

Parameters

A

Double-precision real input vector

I

Stride for *A*

C

Output scalar

IC

Output scalar

N

Count

Discussion

This performs the operation

$$\begin{cases} c = |A_0| \\ d = 0 \end{cases} \quad \text{If} \quad c < |A_{ni}| \quad \text{then} \quad \begin{cases} c = |A_{ni}| \\ d = ni \end{cases} \qquad \text{n} = \{1, \text{N-1}\}$$

Copies the element with the greatest magnitude from real vector *A* to real scalar *C*, and writes its zero-based index to integer scalar *IC*. The index is the actual array index, not the pre-stride index. If vector *A* contains more than one instance of the maximum magnitude, *IC* contains the index of the first instance.

Availability

Available in Mac OS X v10.4 and later.

vDSP_maxv

Vector maximum value.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_maxv (float * A,

 int I,

 float * C,

 unsigned int N);
```

Parameters

A

Single-precision real input vector

I

Stride for *A*

C

Output scalar

N

Count

Discussion

This performs the operation

$c = A_0 \quad \text{If} \quad c < A_{ni} \quad \text{then} \quad c = A_{ni} \qquad n = \{1, N-1\}$

Finds the element with the greatest value in vector *A* and copies this value to scalar *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_maxvD

Vector maximum value.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_maxvD (double * A,

 int I,

 double * C,

 unsigned int N);
```

Parameters

A

Double-precision real input vector

I

Stride for *A*

C
: Output scalar

N
: Count

Discussion

This performs the operation

$c = A_0 \quad \text{If} \quad c < A_{ni} \quad \text{then} \quad c = A_{ni} \qquad n = \{1, N\text{-}1\}$

Finds the element with the greatest value in vector *A* and copies this value to scalar *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_maxvi

Vector maximum value with index.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_maxvi (float * A,

 int I,

 float * C,

 int * IC,

 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

C
: Output scalar

IC
: Output scalar

N
: Count

Discussion

This performs the operation

$$\begin{array}{l} c = A_0 \\ d = 0 \end{array} \quad \text{If} \quad c < A_{ni} \quad \text{then} \quad \begin{array}{l} c = A_{ni} \\ d = ni \end{array} \qquad n = \{1, N\text{-}1\}$$

Copies the element with the greatest value from real vector `A` to real scalar `C`, and writes its zero-based index to integer scalar `IC`. The index is the actual array index, not the pre-stride index. If vector `A` contains more than one instance of the maximum value, `IC` contains the index of the first instance.

Availability
Available in Mac OS X v10.4 and later.

vDSP_maxviD

Vector maximum value with index.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_maxviD (double * A,

 int I,

 double * C,

 int * IC,

 unsigned int N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

C
: Output scalar

IC
: Output scalar

N
: Count

Discussion
This performs the operation

$$\begin{array}{l} c = A_0 \\ d = 0 \end{array} \quad \text{If} \quad c < A_{ni} \quad \text{then} \quad \begin{array}{l} c = A_{ni} \\ d = ni \end{array} \qquad n = \{1, N\text{-}1\}$$

Copies the element with the greatest value from real vector *A* to real scalar *C*, and writes its zero-based index to integer scalar *IC*. The index is the actual array index, not the pre-stride index. If vector *A* contains more than one instance of the maximum value, *IC* contains the index of the first instance.

Availability
Available in Mac OS X v10.4 and later.

vDSP_meamgv

Vector mean magnitude.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_meamgv (float * A,

 int I,

 float * C,

 unsigned int N);
```

Parameters

A
Single-precision real input vector

I
Stride for *A*

C
Output scalar

N
Count

Discussion
This performs the operation

$$C = \frac{1}{N} \sum_{n=0}^{N-1} \left| A_{ni} \right|$$

Finds the mean of the magnitudes of elements of vector *A* and stores this value in scalar *C*.

Availability
Available in Mac OS X v10.4 and later.

vDSP_meamgvD

Vector mean magnitude.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_meamgvD (double * A,

 int I,

 double * C,

 unsigned int N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

C
: Output scalar

N
: Count

Discussion

This performs the operation

$$C = \frac{1}{N} \sum_{n=0}^{N-1} \left|A_{ni}\right|$$

Finds the mean of the magnitudes of elements of vector *A* and stores this value in scalar *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_meanv

Vector mean value.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_meanv (float * A,

 int I,

 float * C,

 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

C
: Output scalar

N
: Count

Discussion
This performs the operation

$$C = \frac{1}{N}\sum_{n=0}^{N-1} A_{ni}$$

Finds the mean value of the elements of vector *A* and stores this value in scalar *C*.

Availability
Available in Mac OS X v10.4 and later.

vDSP_meanvD

Vector mean value.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_meanvD (double * A,

 int I,

 double * C,

 unsigned int N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

C
: Output scalar

N
: Count

Discussion
This performs the operation

$$C = \frac{1}{N}\sum_{n=0}^{N-1} A_{ni}$$

Finds the mean value of the elements of vector *A* and stores this value in scalar *C*.

Availability
Available in Mac OS X v10.4 and later.

vDSP_measqv

Vector mean square value.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_measqv (float * A,

 int I,

 float * C,

 unsigned int N);
```

Parameters

A
Single-precision real input vector

I
Stride for *A*

C
Output scalar

N
Count

Discussion
This performs the operation

$$C = \frac{1}{N}\sum_{n=0}^{N-1} A_{ni}^{2}$$

Finds the mean value of the squares of the elements of vector *A* and stores this value in scalar *C*.

Availability
Available in Mac OS X v10.4 and later.

vDSP_measqvD

Vector mean square value.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_measqvD (double * A,

 int I,

 double * C,

 unsigned int N);
```

Parameters

A

Double-precision real input vector

I

Stride for *A*

C

Output scalar

N

Count

Discussion

This performs the operation

$$C = \frac{1}{N}\sum_{n=0}^{N-1} A_{ni}^{2}$$

Finds the mean value of the squares of the elements of vector *A* and stores this value in scalar *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_minmgv

Vector minimum magnitude.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_minmgv (float * A,

 int I,

 float * C,
 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

C
: Output scalar

N
: Count

Discussion

This performs the operation

$$c = |A_0| \quad \text{If} \quad c > |A_{ni}| \quad \text{then} \quad c = |A_{ni}| \qquad n = \{1, N\text{-}1\}$$

Finds the element with the least magnitude in vector *A* and copies this value to scalar *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_minmgvD

Vector minimum magnitude.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_minmgvD (double * A,

 int I,

 double * C,

 unsigned int N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

C
: Output scalar

N
: Count

Discussion

This performs the operation

$c = |A_0| \quad \text{If} \quad c > |A_{ni}| \quad \text{then} \quad c = |A_{ni}| \qquad n = \{1, N\text{-}1\}$

Finds the element with the least magnitude in vector *A* and copies this value to scalar *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_minmgvi

Vector minimum magnitude with index.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_minmgvi (float * A,

 int I,

 float * C,

 int * IC,

 unsigned int N);
```

Parameters

A

Single-precision real input vector

I

Stride for *A*

C

Output scalar

IC

Output scalar

N

Count

Discussion

This performs the operation

$$\begin{matrix} c = |A_0| \\ d = 0 \end{matrix} \quad \text{If} \quad c > |A_{ni}| \quad \text{then} \quad \begin{matrix} c = |A_{ni}| \\ d = ni \end{matrix} \qquad n = \{1, N\text{-}1\}$$

Copies the element with the least magnitude from real vector *A* to real scalar *C*, and writes its zero-based index to integer scalar *IC*. The index is the actual array index, not the pre-stride index. If vector *A* contains more than one instance of the least magnitude, *IC* contains the index of the first instance.

Availability
Available in Mac OS X v10.4 and later.

vDSP_minmgviD

Vector minimum magnitude with index.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_minmgviD (double * A,

 int I,

 double * C,

 int * IC,

 unsigned int N);
```

Parameters

A
Double-precision real input vector

I
Stride for *A*

C
Output scalar

IC
Output scalar

N
Count

Discussion
This performs the operation

$$\begin{matrix} c = |A_0| \\ d = 0 \end{matrix} \quad \text{If} \quad c > |A_{ni}| \quad \text{then} \quad \begin{matrix} c = |A_{ni}| \\ d = ni \end{matrix} \quad n = \{1, N\text{-}1\}$$

Copies the element with the least magnitude from real vector *A* to real scalar *C*, and writes its zero-based index to integer scalar *IC*. The index is the actual array index, not the pre-stride index. If vector *A* contains more than one instance of the least magnitude, *IC* contains the index of the first instance.

Availability
Available in Mac OS X v10.4 and later.

vDSP_minv

Vector minimum value.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_minv (float * A,

 int I,

 float * C,

 unsigned int N);
```

Parameters

A

Single-precision real input vector

I

Stride for *A*

C

Output scalar

N

Count

Discussion

This performs the operation

$c = A_0 \quad \text{If} \quad c > A_{ni} \quad \text{then} \quad c = A_{ni} \qquad n = \{1, N\text{-}1\}$

Finds the element with the least value in vector *A* and copies this value to scalar *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_minvD

Vector minimum value.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_minvD (double * A,

 int I,

 double * C,

 unsigned int N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

C
: Output scalar

N
: Count

Discussion

This performs the operation

$$c = A_0 \qquad \text{If} \quad c > A_{ni} \quad \text{then} \quad c = A_{ni} \qquad n = \{1, N-1\}$$

Finds the element with the least value in vector *A* and copies this value to scalar *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_minvi

Vector minimum value with index.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_minvi (float * A,

 int I,

 float * C,

 int * IC,

 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

C
: Output scalar

IC
: Output scalar

N
Count

Discussion
This performs the operation

$$\begin{matrix} c = A_0 \\ d = 0 \end{matrix} \quad \text{If} \quad c > A_{ni} \quad \text{then} \quad \begin{matrix} c = A_{ni} \\ d = ni \end{matrix} \qquad n = \{1, N\text{-}1\}$$

Copies the element with the least value from real vector *A* to real scalar *C*, and writes its zero-based index to integer scalar *IC*. The index is the actual array index, not the pre-stride index. If vector *A* contains more than one instance of the least value, *IC* contains the index of the first instance.

Availability
Available in Mac OS X v10.4 and later.

vDSP_minviD

Vector minimum value with index.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_minviD (double * A,

 int I,

 double * C,

 int * IC,

 unsigned int N);
```

Parameters

A
Double-precision real input vector

I
Stride for *A*

C
Output scalar

IC
Output scalar

N
Count

Discussion
This performs the operation

$$\begin{matrix} c = A_0 \\ d = 0 \end{matrix} \quad \text{If} \quad c > A_{ni} \quad \text{then} \quad \begin{matrix} c = A_{ni} \\ d = ni \end{matrix} \qquad n = \{1, N-1\}$$

Copies the element with the least value from real vector *A* to real scalar *C*, and writes its zero-based index to integer scalar *IC*. The index is the actual array index, not the pre-stride index. If vector *A* contains more than one instance of the least value, *IC* contains the index of the first instance.

Availability

Available in Mac OS X v10.4 and later.

vDSP_mmov

The contents of a submatrix are copied to another submatrix.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_mmov (float * A,

 float * C,

 unsigned int NC,

 unsigned int NR,

 unsigned int TCA,

 unsigned int TCC);
```

Parameters

A

Single-precision real input submatrix

C

Single-precision real output submatrix

NC

Number of columns in *A* and *C*

NR

Number of rows in *A* and *C*

TCA

Number of columns in the matrix of which *A* is a submatrix

TCC

Number of columns in the matrix of which *C* is a submatrix

Discussion

This performs the operation

$$C_{(x,y)} = A_{(x,y)} \qquad x = \{0, X-1\}; \quad y = \{0, Y-1\}$$

The matrices are assumed to be stored in row-major order. Thus elements `A[i][j]` and `A[i][j+1]` are adjacent. Elements `A[i][j]` and `A[i+1][j]` are *`TCA`* elements apart.

This function may be used to move a subarray beginning at any point in a larger embedding array by passing for *`A`* the address of the first element of the subarray. For example, to move a subarray starting at `A[3][4]`, pass `&A[3][4]`. Similarly, the address of the first destination element is passed for *`C`*

`NC` may equal *`TCA`*, and it may equal *`TCC`*. To copy all of an array to all of another array, pass the number of rows in *`NR`* and the number of columns in *`NC`*, *`TCA`*, and *`TCC`*.

Availability
Available in Mac OS X v10.4 and later.

vDSP_mmovD

The contents of a submatrix are copied to another submatrix.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_mmovD (double * A,

 double * C,

 unsigned int NC,

 unsigned int NR,

 unsigned int TCA,

 unsigned int TCC);
```

Parameters

`A`
    Double-precision real input submatrix

`C`
    Double-precision real output submatrix

`NC`
    Number of columns in *`A`* and *`C`*

`NR`
    Number of rows in *`A`* and *`C`*

`TCA`
    Number of columns in the matrix of which *`A`* is a submatrix

`TCC`
    Number of columns in the matrix of which *`C`* is a submatrix

Discussion
This performs the operation

$$C_{(x,y)} = A_{(x,y)} \qquad x = \{0, X-1\}; \quad y = \{0, Y-1\}$$

The matrices are assumed to be stored in row-major order. Thus elements `A[i][j]` and `A[i][j+1]` are adjacent. Elements `A[i][j]` and `A[i+1][j]` are *`TCA`* elements apart.

This function may be used to move a subarray beginning at any point in a larger embedding array by passing for *`A`* the address of the first element of the subarray. For example, to move a subarray starting at `A[3][4]`, pass `&A[3][4]`. Similarly, the address of the first destination element is passed for *`C`*

`NC` may equal *`TCA`*, and it may equal *`TCC`*. To copy all of an array to all of another array, pass the number of rows in *`NR`* and the number of columns in *`NC`*, *`TCA`*, and *`TCC`*.

Availability
Available in Mac OS X v10.4 and later.

vDSP_mmul

Multiplies an M-by-P matrix `A` by a P-by-N matrix `B` and stores the results in an M-by-N matrix `C`. This function can only be performed out-of-place.

```
void
vDSP_mmul (float * A,

 SInt32 I,

 float * B,

 SInt32 J,

 float * C,

 SInt32 K,

 SInt32 M,

 SInt32 N,

 SInt32 P);
```

Discussion
This performs the operation

$$C_{(mN+n)K} = \sum_{p=0}^{P-1} A_{(mP+p)I} \bullet B_{(pN+n)J} \qquad n = \{0, N-1\} \text{ and } m = \{0, M-1\}$$

Parameters `A` and `B` are the matrixes to be multiplied. `I` is an address stride through `A`. `J` is an address stride through `B`.

Parameter `C` is the result matrix. `K` is an address stride through `C`.

Parameter `M` is the row count for both `A` and `C`. Parameter `N` is the column count for both `B` and `C`. Parameter `P` is the column count for `A` and the row count for `B`.

Criteria to invoke vectorized code:

- `A`, `B`, and `C` must be 16-byte aligned.
- `I`, `J`, and `K` must all be 1.
- `M`, `N`, and `P` must be multiples of 4 and greater than 32.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_mmulD

Multiplies an M-by-P matrix `A` by a P-by-N matrix `B` and stores the results in an M-by-N matrix `C`. This function can only be performed out-of-place.

```
void
vDSP_mmulD (double * A,

 SInt32 I,

 double * B,

 SInt32 J,

 double * C,

 SInt32 K,

 SInt32 M,

 SInt32 N,

 SInt32 P);
```

Discussion
This performs the operation

$$C_{(mN+n)K} = \sum_{p=0}^{P-1} A_{(mP+p)I} \cdot B_{(pN+n)J} \qquad n = \{0, N\text{-}1\} \text{ and } m = \{0, M\text{-}1\}$$

Parameters `A` and `B` are the matrixes to be multiplied. `I` is an address stride through `A`. `J` is an address stride through `B`.

Parameter `C` is the result matrix. `K` is an address stride through `C`.

Parameter `M` is the row count for both `A` and `C`. Parameter `N` is the column count for both `B` and `C`. Parameter `P` is the column count for `A` and the row count for `B`.

Criteria to invoke vectorized code:

- `A`, `B`, and `C` must be 16-byte aligned.
- `I`, `J`, and `K` must all be 1.
- `M`, `N`, and `P` must be multiples of 4 and greater than 32.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_mtrans

Creates a transposed matrix `C` from a source matrix `A`.

```
void
vDSP_mtrans (float * A,

 SInt32 I,

 float * C,

 SInt32 K,

 SInt32 M,

 SInt32 N);
```

Discussion
This performs the operation

$C_{(mN+n)K} = A_{(nM+m)I}$ n = {0, N-1} and m = {0, M-1}

Parameter `A` is the source matrix. `I` is an address stride through the source matrix.

Parameter `C` is the resulting transposed matrix. `K` is an address stride through the result matrix.

Parameter `M` is the row count for `C` (and the column count for `A`). Parameter `N` is the column count for `C` (and the row count for `A`).

Criteria to invoke vectorized code:

- *`A`* and *`C`* must be equal (that is, the function must be performed in place).
- *`A`* must be a square matrix.
- *`A`* must be have a 32-byte aligned base address.
- *`M`* must be a multiple of 8.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_mtransD

Creates a transposed matrix C from a source matrix A.

```
void
vDSP_mtransD (double * A,

 SInt32 I,

 double * C,

 SInt32 K,

 SInt32 M,

 SInt32 N);
```

Discussion
This performs the operation

$$C_{(mN+n)K} = A_{(nM+m)I} \qquad n = \{0, N-1\} \text{ and } m = \{0, M-1\}$$

Parameter A is the source matrix. I is an address stride through the source matrix.

Parameter C is the resulting transposed matrix. K is an address stride through the result matrix.

Parameter M is the row count for C (and the column count for A). Parameter N is the column count for C (and the row count for A).

Criteria to invoke vectorized code:

- *A* and *C* must be equal (that is, the function must be performed in place).
- *A* must be a square matrix.
- *A* must be have a 32-byte aligned base address.
- *M* must be a multiple of 8.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_mvessq

Vector mean of signed squares.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_mvessq (float * A,

 int I,

 float * C,

 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

C
: Output scalar

N
: Count

Discussion

This performs the operation

$$C = \frac{1}{N} \sum_{n=0}^{N-1} A_{ni} \bullet \left|A_{ni}\right|$$

Finds the mean value of the signed squares of the elements of vector *A* and stores this value in scalar *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_mvessqD

Vector mean of signed squares.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_mvessqD (double * A,

 int I,

 double * C,

 unsigned int N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

C
: Output scalar

N
: Count

Discussion

This performs the operation

$$C = \frac{1}{N} \sum_{n=0}^{N-1} A_{ni} \bullet \left|A_{ni}\right|$$

Finds the mean value of the signed squares of the elements of vector *A* and stores this value in scalar *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_nzcros

Find zero crossings.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_nzcros (float * A,

 int I,

 unsigned int B,

 int * C,

 unsigned int * D,

 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

B
: Maximum number of crossings to find

C
: Index of last crossing found

N
: Count of elements in *A*

Discussion

This performs the operation

```
$d = c = 0$
For integer $n$,  $0 < n < N$;
  If $\mathrm{sign}(A_{nI})! = \mathrm{sign}(A_{(n-1)I})$ then
    $d = d+1$
    If $d = b$ then
      $c = nI$,
      exit.    n = {1, N-1}
```

The “function” sign(x) above has the value -1 if the sign bit of x is 1 (x is negative or -0), and +1 if the sign bit is 0 (x is positive or +0).

Scans vector *A* to locate transitions from positive to negative values and from negative to positive values. The scan terminates when the number of crossings specified by *B* is found, or the end of the vector is reached. The zero-based index of the last crossing is returned in *C*. *C* is the actual array index, not the pre-stride index. If the zero crossing that *B* specifies is not found, zero is returned in *C*. The total number of zero crossings found is returned in *D*.

Note that a transition from -0 to +0 or from +0 to -0 is counted as a zero crossing.

Availability

Available in Mac OS X v10.4 and later.

vDSP_nzcrosD

Find zero crossings.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_nzcrosD (float * A,

 int I,

 unsigned int B,

 int * C,
 unsigned int * D,
 unsigned int N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

B
: Maximum number of crossings to find

C
: Index of last crossing found

N
: Count of elements in *A*

Discussion

This performs the operation

```
$d = c = 0$
For integer $n$,  $0 < n < N$;
  If $\text{sign}(A_{nI})! = \text{sign}(A_{(n-1)I})$ then
    $d = d+1$
    If $d = b$ then
      $c = nI$,
      exit.     $n = \{1, N-1\}$
```

The “function” sign(x) above has the value -1 if the sign bit of x is 1 (x is negative or -0), and +1 if the sign bit is 0 (x is positive or +0).

Scans vector *A* to locate transitions from positive to negative values and from negative to positive values. The scan terminates when the number of crossings specified by *B* is found, or the end of the vector is reached. The zero-based index of the last crossing is returned in *C*. *C* is the actual array index, not the pre-stride index. If the zero crossing that *B* specifies is not found, zero is returned in *C*. The total number of zero crossings found is returned in *D*.

Note that a transition from -0 to +0 or from +0 to -0 is counted as a zero crossing.

Availability

Available in Mac OS X v10.4 and later.

vDSP_polar

Rectangular to polar conversion.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_polar (float * A,

 int I,

 float * C,

 int K,

 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*, must be even

C
: Single-precision output vector

K
: Stride for *C*, must be even

N
: Number of ordered pairs processed

Discussion

This performs the operation

$$C_{nK} = \sqrt{A_{nI}^2 + A_{nI+1}^2} \qquad C_{nK+1} = \text{atan2}(A_{nI+1}, A_{nI}), \qquad n = \{0, N-1\}$$

Converts rectangular coordinates to polar coordinates. Cartesian (x,y) pairs are read from vector `A`. Polar (rho, theta) pairs, where rho is the radius and theta is the angle in the range [–pi, pi] are written to vector `C`. `N` specifies the number of coordinate pairs in `A` and `C`.

Coordinate pairs are adjacent elements in the array, regardless of stride; stride is the distance from one coordinate pair to the next.

This function performs the inverse operation of `vDSP_rect`, which converts polar to rectangular coordinates.

Availability

Available in Mac OS X v10.4 and later.

vDSP_polarD

Rectangular to polar conversion.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_polarD (double * A,

 int I,

 double * C,

 int K,

 unsigned int N);
```

Parameters

A

Double-precision real input vector

I

Stride for *A*, must be even

C

Double-precision output vector

K

Stride for *C*, must be even

N

Number of ordered pairs processed

Discussion

This performs the operation

$$C_{nK} = \sqrt{A_{nI}^2 + A_{nI+1}^2} \qquad C_{nK+1} = \text{atan2}\,(A_{nI+1}, A_{nI}) \,, \qquad n = \{0, N\text{-}1\}$$

Converts rectangular coordinates to polar coordinates. Cartesian (x,y) pairs are read from vector `A`. Polar (rho, theta) pairs, where rho is the radius and theta is the angle in the range [–pi, pi] are written to vector `C`. `N` specifies the number of coordinate pairs in `A` and `C`.

Coordinate pairs are adjacent elements in the array, regardless of stride; stride is the distance from one coordinate pair to the next.

This function performs the inverse operation of `vDSP_rectD`, which converts polar to rectangular coordinates.

Availability

Available in Mac OS X v10.4 and later.

vDSP_rect

Polar to rectangular conversion.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_rect (float * A,

 int I,

 float * C,

 int K,

 unsigned int N);
```

Parameters

A

Single-precision real input vector

I

Stride for *A*, must be even

C

Single-precision real output vector

K

Stride for *C*, must be even

N

Number of ordered pairs processed

Discussion

This performs the operation

$$C_{nK} = A_{nI} \cdot \cos(A_{nI+1}) \qquad C_{nK+1} = A_{nI} \cdot \sin(A_{nI+1}) \qquad n = \{0, N\text{-}1\}$$

Converts polar coordinates to rectangular coordinates. Polar (rho, theta) pairs, where rho is the radius and theta is the angle in the range [–pi, pi] are read from vector *A*. Cartesian (x,y) pairs are written to vector *C*. *N* specifies the number of coordinate pairs in *A* and *C*.

Coordinate pairs are adjacent elements in the array, regardless of stride; stride is the distance from one coordinate pair to the next.

This function performs the inverse operation of `vDSP_polar`, which converts rectangular to polar coordinates.

Availability

Available in Mac OS X v10.4 and later.

vDSP_rectD

Polar to rectangular conversion.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_rectD (double * A,
 int I,

 double * C,

 int K,

 unsigned int N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*, must be even

C
: Double-precision real output vector

K
: Stride for *C*, must be even

N
: Number of ordered pairs processed

Discussion

This performs the operation

$$C_{nK} = A_{nI} \bullet \cos(A_{nI+1}) \qquad C_{nK+1} = A_{nI} \bullet \sin(A_{nI+1}) \qquad n = \{0, N\text{-}1\}$$

Converts polar coordinates to rectangular coordinates. Polar (rho, theta) pairs, where rho is the radius and theta is the angle in the range [–pi, pi] are read from vector `A`. Cartesian (x,y) pairs are written to vector `C`. `N` specifies the number of coordinate pairs in `A` and `C`.

Coordinate pairs are adjacent elements in the array, regardless of stride; stride is the distance from one coordinate pair to the next.

This function performs the inverse operation of `vDSP_polarD`, which converts rectangular to polar coordinates.

Availability

Available in Mac OS X v10.4 and later.

vDSP_rmsqv

Vector root-mean-square.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_rmsqv (float * A,

 int I,
 float * C,

 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

C
: Single-precision real output scalar

N
: Count

Discussion

This performs the operation

$$C = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1} A_{nI}^{2}}$$

Calculates the root mean square of the elements of *A* and stores the result in *C*

Availability

Available in Mac OS X v10.4 and later.

vDSP_rmsqvD

Vector root-mean-square.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_rmsqvD (double * A,

 int I,

 double * C,

 unsigned int N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

C

Double-precision real output scalar

N

Count

Discussion

This performs the operation

$$C = \sqrt{\frac{1}{N} \sum_{n=0}^{N-1} A_{nI}^{2}}$$

Calculates the root mean square of the elements of *A* and stores the result in *C*

Availability

Available in Mac OS X v10.4 and later.

vDSP_svdiv

Divide scalar by vector.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_svdiv (float * A,

 float * B,

 int J,

 float * C,

 int K,

 unsigned int N);
```

Parameters

A

Single-precision real input scalar

B

Single-precision real input vector

J

Stride for *B*

C

Single-precision real output vector

K

Stride for *C*

N

Count

Discussion

This performs the operation

$$C_{nK} = \frac{A}{B_{nJ}}, \qquad n = \{0, N-1\}$$

Divides scalar *A* by each element of vector *B*, storing the results in vector *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_svdivD

Divide scalar by vector.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_svdivD (double * A,

 double * B,

 int J,

 double * C,

 int K,

 unsigned int N);
```

Parameters

A

Double-precision real input scalar

B

Double-precision real input vector

J

Stride for *B*

C

Double-precision real output vector

K

Stride for *C*

N

Count

Discussion

This performs the operation

$$C_{nK} = \frac{A}{B_{nJ}}, \qquad n = \{0, N-1\}$$

Divides scalar *A* by each element of vector *B*, storing the results in vector *C*.

Availability
Available in Mac OS X v10.4 and later.

vDSP_sve

Vector sum.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_sve (float * A,

 int I,

 float * C,

 unsigned int N);
```

Parameters

A
: Single-precision real input scalar

I
: Stride for *A*

C
: Single-precision real output scalar

N
: Count

Discussion
This performs the operation

$$C = \sum_{n=0}^{N-1} A_{nI}$$

Writes the sum of the elements of *A* into *C*.

Availability
Available in Mac OS X v10.4 and later.

vDSP_sveD

Vector sum.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_sveD (double * A,

 int I,

 double * C,

 unsigned int N);
```

Parameters

A
: Double-precision real input scalar

I
: Stride for *A*

C
: Double-precision real output scalar

N
: Count

Discussion

This performs the operation

$$C = \sum_{n=0}^{N-1} A_{nI}$$

Writes the sum of the elements of *A* into *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_svemg

Vector sum of magnitudes.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_svemg (float * A,

 int I,

 float * C,

 unsigned int N);
```

Parameters

A
: Single-precision real input scalar

I
: Stride for *A*

C
: Single-precision real output scalar

N
: Count

Discussion

This performs the operation

$$C = \sum_{n=0}^{N-1} \left| A_{nI} \right|$$

Writes the sum of the magnitudes of the elements of *A* into *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_svemgD

Vector sum of magnitudes.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_svemgD (double * A,

 int I,

 double * C,

 unsigned int N);
```

Parameters

A
: Double-precision real input scalar

I
: Stride for *A*

C
: Double-precision real output scalar

N
: Count

Discussion
This performs the operation

$$C = \sum_{n=0}^{N-1} \left|A_{nI}\right|$$

Writes the sum of the magnitudes of the elements of *A* into *C*.

Availability
Available in Mac OS X v10.4 and later.

vDSP_svesq

Vector sum of squares.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_svesq (float * A,

 int I,

 float * C,

 unsigned int N);
```

Parameters

A
: Single-precision real input scalar

I
: Stride for *A*

C
: Single-precision real output scalar

N
: Count

Discussion
This performs the operation

$$C = \sum_{n=0}^{N-1} A_{nI}^{2}$$

Writes the sum of the squares of the elements of *A* into *C*.

Availability
Available in Mac OS X v10.4 and later.

vDSP_svesqD

Vector sum of squares.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_svesqD (double * A,

 int I,

 double * C,

 unsigned int N);
```

Parameters

A

Double-precision real input scalar

I

Stride for *A*

C

Double-precision real output scalar

N

Count

Discussion

This performs the operation

$$C = \sum_{n=0}^{N-1} A_{nI}^2$$

Writes the sum of the squares of the elements of *A* into *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_svs

Vector sum of signed squares.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_svs (float * A,

 int I,

 float * C,
 unsigned int N);
```

Parameters

A
: Single-precision real input scalar

I
: Stride for *A*

C
: Single-precision real output scalar

N
: Count

Discussion

This performs the operation

$$C = \sum_{n=0}^{N-1} A_{nI} \cdot \left|A_{nI}\right|$$

Writes the sum of the signed squares of the elements of *A* into *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_svsD

Vector sum of signed squares.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_svsD (double * A,

 int I,

 double * C,

 unsigned int N);
```

Parameters

A
: Double-precision real input scalar

I
: Stride for *A*

C
: Double-precision real output scalar

N

Count

Discussion

This performs the operation

$$C = \sum_{n=0}^{N-1} A_{nI} \bullet \left|A_{nI}\right|$$

Writes the sum of the signed squares of the elements of *A* into *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vaam

Vector add, add, and multiply.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vaam (float * A,

 int I,

 float * B,

 int J,

 float * C,

 int K,

 float * D,

 int L,

 float * E,

 int M,

 unsigned int N);
```

Parameters

A

Single-precision real input vector

I

Stride for *A*

B

Single-precision real input vector

J
: Stride for *B*

C
: Single-precision real input vector

K
: Stride for *C*

D
: Single-precision real input vector

L
: Stride for *D*

E
: Single-precision real output vector

M
: Stride for *E*

N
: Count ; each vector must have at least *N* elements

Discussion

This performs the operation

$$E_{nm} = (A_{ni} + B_{nj})(C_{nk} + D_{nl}) \qquad n = \{0, N-1\}$$

Multiplies the sum of vectors *A* and *B* by the sum of vectors *C* and *D*. Results are stored in vector *E*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vaamD

Vector add, add, and multiply.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vaamD (double * A,

 int I,

 double * B,

 int J,

 double * C,

 int K,

 double * D,
 int L,
 double * E,
 int M,
 unsigned int N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

B
: Double-precision real input vector

J
: Stride for *B*

C
: Double-precision real input vector

K
: Stride for *C*

D
: Double-precision real input vector

L
: Stride for *D*

E
: Double-precision real output vector

M
: Stride for *E*

N
: Count ; each vector must have at least N elements

Discussion

This performs the operation

$$E_{nm} = (A_{ni} + B_{nj})(C_{nk} + D_{nl}) \qquad n = \{0, N-1\}$$

Multiplies the sum of vectors *A* and *B* by the sum of vectors *C* and *D*. Results are stored in vector *E*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vabs

Vector absolute values.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vabs (float * A,

 int I,

 float * C,

 int K,

 unsigned int N);
```

Parameters

A

Single-precision real input vector

I

Stride for *A*

C

Single-precision real output vector

K

Stride for *C*

N

Count

Discussion

This performs the operation

$$C_{nK} = |A_{nI}|, \qquad n = \{0, N-1\}$$

Writes the absolute values of the elements of *A* into corresponding elements of *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vabsD

Vector absolute values.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vabsD (double * A,

 int I,
 double * C,

 int K,

 unsigned int N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

C
: Double-precision real output vector

K
: Stride for *C*

N
: Count

Discussion

This performs the operation

$$C_{nK} = \left|A_{nI}\right|, \qquad n = \{0, N\text{-}1\}$$

Writes the absolute values of the elements of *A* into corresponding elements of *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vabsi

Integer vector absolute values.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vabsi (int * A,

 int I,

 int * C,

 int K,

 unsigned int N);
```

Parameters

A
: Integer input vector

`I`

Stride for `A`

`C`

Integer output vector

`K`

Stride for `C`

`N`

Count

Discussion

This performs the operation

$$C_{nK} = |A_{nI}|, \qquad n = \{0, N-1\}$$

Writes the absolute values of the elements of `A` into corresponding elements of `C`.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vadd

Adds vector `A` to vector `B` and leaves the result in vector `C`.

```
void
vDSP_vadd (const float A[],
    SInt32 I,
    const float B[],
    SInt32 J,
    float C[],
    SInt32 K,
    UInt32 N);
```

Discussion

This performs the operation

$$C_{nK} = A_{nI} + B_{nJ} \qquad n = \{0, N-1\}$$

Criteria to invoke vectorized code:

- The vectors `A`,`B`, and `C` must be relatively aligned.
- The value of `N` must be equal to or greater than 8.
- The values of `I`, `J`, and `K` must be 1.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vaddD

Adds vector A to vector B and leaves the result in vector C.

```
void
vDSP_vaddD (const float A[],
    SInt32 I,
    const float B[],
    SInt32 J,
    float C[],
    SInt32 K,
    UInt32 N);
```

Discussion

This performs the operation

$$C_{nK} = A_{nI} + B_{nJ} \qquad n = \{0, N-1\}$$

Criteria to invoke vectorized code:

- The vectors A,B, and C must be relatively aligned.
- The value of N must be equal to or greater than 8.
- The values of I, J, and K must be 1.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vam

Adds vectors A and B, multiplies the sum by vector C, and leaves the result in vector D.

```
void
vDSP_vam (const float A[],
    SInt32 I,
    const float B[],
    SInt32 J,
    const float C[],
    SInt32 K,
    float D[],
    SInt32 L,
    UInt32 N);
```

Discussion

This performs the operation

$$D_{nL} = (A_{nI} + B_{nJ})\, C_{nK} \qquad n = \{0, N-1\}$$

Criteria to invoke vectorized code:

- The vectors A, B,C, and D must be relatively aligned.

- The value of `N` must be equal to or greater than 8.
- The values of `I`, `J`, `K`, and `L` must be 1.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_vamD

Adds vectors `A` and `B`, multiplies the sum by vector `C`, and leaves the result in vector `D`.

```
void
vDSP_vamD (const double A[],
    SInt32 I,
    const double B[],
    SInt32 J,
    const double C[],
    SInt32 K,
    double D[],
    SInt32 L,
    UInt32 N);
```

Discussion
This performs the operation

$$D_{nL} = (A_{nI} + B_{nJ}) C_{nK} \qquad n = \{0, N-1\}$$

Criteria to invoke vectorized code:

- The vectors `A`, `B`,`C`, and `D` must be relatively aligned.
- The value of `N` must be equal to or greater than 8.
- The values of `I`, `J`, `K`, and `L` must be 1.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_vasbm

Vector add, subtract, and multiply.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vasbm (float * A,

 int I,
 float * B,
 int J,
 float * C,
 int K,
 float * D,
 int L,
 float * E,
 int M,
 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

B
: Single-precision real input vector

J
: Stride for *B*

C
: Single-precision real input vector

K
: Stride for *C*

D
: Single-precision real input vector

L
: Stride for *D*

E
: Single-precision real output vector

M
: Stride for *E*

N
: Count ; each vector must have at least *N* elements

Discussion

This performs the operation

$$E_{nM} = (A_{nI} + B_{nJ})(C_{nK} - D_{nL}) \ , \qquad n = \{0, N-1\}$$

Multiplies the sum of vectors *A* and *B* by the result of subtracting vector *D* from vector *C*. Results are stored in vector *E*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vasbmD

Vector add, subtract, and multiply.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vasbmD (double * A,

 int I,

 double * B,

 int J,

 double * C,

 int K,

 double * D,

 int L,

 double * E,

 int M,

 unsigned int N);
```

Parameters

A

Double-precision real input vector

I

Stride for *A*

B

Double-precision real input vector

J

Stride for *B*

C

Double-precision real input vector

K

Stride for *C*

D
    Double-precision real input vector

L
    Stride for *D*

E
    Double-precision real output vector

M
    Stride for *E*

N
    Count ; each vector must have at least N elements

Discussion

This performs the operation

$$E_{nM} = (A_{nI} + B_{nJ})(C_{nK} - D_{nL}) \ , \qquad n = \{0, N\text{-}1\}$$

Multiplies the sum of vectors *A* and *B* by the result of subtracting vector *D* from vector *C*. Results are stored in vector *E*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vasm

Vector add and scalar multiply.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vasm (float * A,

 int I,

 float * B,

 int J,

 float * C,

 float * D,

 int L,

 unsigned int N);
```

Parameters

A
    Single-precision real input vector

I
Stride for *A*

B
Single-precision real input vector

J
Stride for *B*

C
Single-precision real input scalar

D
Single-precision real output vector

L
Stride for *D*

N
Count ; each vector must have at least N elements

Discussion

This performs the operation

$D_{nM} = (A_{nI} + B_{nJ})\, c$, n = {0, N-1}

Multiplies the sum of vectors *A* and *B* by scalar *C*. Results are stored in vector *D*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vasmD

Vector add and scalar multiply.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vasmD (double * A,

 int I,

 double * B,

 int J,

 double * C,

 double * D,

 int L,

 unsigned int N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

B
: Double-precision real input vector

J
: Stride for *B*

C
: Double-precision real input scalar

D
: Double-precision real output vector

L
: Stride for *D*

N
: Count ; each vector must have at least N elements

Discussion

This performs the operation

$$D_{nM} = (A_{nI} + B_{nJ})\,c\ , \qquad \text{n} = \{0, \text{N-1}\}$$

Multiplies the sum of vectors *A* and *B* by scalar *C*. Results are stored in vector *D*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vavlin

Vector linear average.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vavlin (float * A,

 int I,

 float * B,

 float * C,

 int K,

 unsigned int N);
```

Parameters

A

Single-precision real input vector

I

Stride for *A*

B

Single-precision real input scalar

C

Single-precision real input-output vector

K

Stride for *C*

N

Count ; each vector must have at least N elements

Discussion

This performs the operation

$$C_{nK} = \frac{C_{nK}B + A_{nI}}{B + 1.0} \quad , \qquad n = \{0, N\text{-}1\}$$

Recalculates the linear average of input-output vector *C* to include input vector *A*. Input scalar *B* specifies the number of vectors included in the current average.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vavlinD

Vector linear average.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vavlinD (double * A,

 int I,

 double * B,

 double * C,

 int K,

 unsigned int N);
```

Parameters

A

Double-precision real input vector

I
: Stride for *A*

B
: Double-precision real input scalar

C
: Double-precision real input-output vector

K
: Stride for *C*

N
: Count ; each vector must have at least N elements

Discussion

This performs the operation

$$C_{nK} = \frac{C_{nK} B + A_{nI}}{B + 1.0} \quad , \qquad n = \{0, N\text{-}1\}$$

Recalculates the linear average of input-output vector *C* to include input vector *A*. Input scalar *B* specifies the number of vectors included in the current average.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vclip

Vector clip.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vclip (float * A,

 int I,

 float * B,

 float * C,

 float * D,

 int L,

 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

B
: Single-precision real input scalar: low clipping threshold

C
: Single-precision real input scalar: high clipping threshold

D
: Single-precision real output vector

L
: Stride for *D*

N
: Count

Discussion

This performs the operation

$$\begin{cases} \text{If } A_{nI} < b & \text{then } D_{nM} = B \\ \text{If } A_{nI} > c & \text{then } D_{nM} = C \ , \qquad n = \{0, N\text{-}1\} \\ \text{If } b \le A_{nI} \le c & \text{then } D_{nM} = A_{nI} \end{cases}$$

Elements of *A* are copied to *D* while clipping elements that are outside the interval [*B*, *C*] to the endpoints.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vclipc

Vector clip and count.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vclipc (float * A,

 int I,

 float * B,

 float * C,

 float * D,

 int L,

 unsigned int N,
 unsigned int * NLOW,
 unsigned int * NHI);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

B
: Single-precision real input scalar: low clipping threshold

C
: Single-precision real input scalar: high clipping threshold

D
: Single-precision real output vector

L
: Stride for *D*

N
: Count of elements in *A* and *D*

NLOW
: Number of elements that were clipped to *B*

NHI
: Number of elements that were clipped to *C*

Discussion

This performs the operation

$$\begin{cases} \text{If } A_{nI} < b & \text{then } D_{nM} = B \\ \text{If } A_{nI} > c & \text{then } D_{nM} = C \;, \qquad \text{n = \{0, N-1\}} \\ \text{If } b \le A_{nI} \le c & \text{then } D_{nM} = A_{nI} \end{cases}$$

Elements of *A* are copied to *D* while clipping elements that are outside the interval [*B*, *C*] to the endpoints.

The count of elements clipped to *B* is returned in **NLOW*, and the count of elements clipped to *C* is returned in **NHI*

Availability

Available in Mac OS X v10.4 and later.

vDSP_vclipcD

Vector clip and count.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vclipcD (float * A,

 int I,

 double * B,

 double * C,

 double * D,

 int L,

 unsigned int N,

 unsigned int * NLOW,

 unsigned int * NHI);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

B
: Double-precision real input scalar: low clipping threshold

C
: Double-precision real input scalar: high clipping threshold

D
: Double-precision real output vector

L
: Stride for *D*

N
: Count of elements in *A* and *D*

NLOW
: Number of elements that were clipped to *B*

NHI
: Number of elements that were clipped to *C*

Discussion

This performs the operation

$$\begin{cases} \text{If} \quad A_{nI} < b & \text{then} \quad D_{nM} = B \\ \text{If} \quad A_{nI} > c & \text{then} \quad D_{nM} = C \ , \qquad n = \{0, \text{N-1}\} \\ \text{If} \quad b \le A_{nI} \le c & \text{then} \quad D_{nM} = A_{nI} \end{cases}$$

Elements of *A* are copied to *D* while clipping elements that are outside the interval [*B*, *C*] to the endpoints.

The count of elements clipped to *B* is returned in **NLOW*, and the count of elements clipped to *C* is returned in **NHI*

Availability
Available in Mac OS X v10.4 and later.

vDSP_vclipD

Vector clip.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vclipD (double * A,

 int I,

 double * B,

 double * C,

 double * D,

 int L,

 unsigned int N);
```

Parameters

A
  Double-precision real input vector

I
  Stride for *A*

B
  Double-precision real input scalar: low clipping threshold

C
  Double-precision real input scalar: high clipping threshold

D
  Double-precision real output vector

L
  Stride for *D*

N
  Count

Discussion
This performs the operation

$$\begin{cases} \text{If } A_{nI} < b & \text{then } D_{nM} = B \\ \text{If } A_{nI} > c & \text{then } D_{nM} = C \;, \qquad n = \{0, N\text{-}1\} \\ \text{If } b \le A_{nI} \le c & \text{then } D_{nM} = A_{nI} \end{cases}$$

Elements of *A* are copied to *D* while clipping elements that are outside the interval [*B*, *C*] to the endpoints.

Availability
Available in Mac OS X v10.4 and later.

vDSP_vclr

Vector clear.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vclr (float * C,

 int K,

 unsigned int N);
```

Parameters

C
: Single-precision real input-output vector

K
: Stride for *C*

N
: Count

Discussion
All elements of vector *C* are set to zeros.

Availability
Available in Mac OS X v10.4 and later.

vDSP_vclrD

Vector clear.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vclrD (double * C,

 int K,
 unsigned int N);
```

Parameters

C
: Double-precision real input-output vector

K
: Stride for *C*

N
: Count

Discussion

All elements of vector *C* are set to zeros.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vcmprs

Vector compress.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vcmprs (float * A,

 int I,

 float * B,

 int J,

 float * C,

 int K,

 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

B
: Single-precision real input vector

J
: Stride for *B*

C
: Single-precision real output vector

K

Stride for *C*

N

Count

Discussion

Performs the operation

$p = 0$

If $B_{nJ} \neq 0.0$ then $C_{pK} = A_{nI}$; $p = p + 1$; $n = \{0, N-1\}$

Compresses vector *A* based on the nonzero values of gating vector *B*. For nonzero elements of *B*, corresponding elements of *A* are sequentially copied to output vector *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vcmprsD

Vector compress.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vcmprsD (double * A,

 int I,

 double * B,

 int J,

 double * C,

 int K,

 unsigned int N);
```

Parameters

A

Double-precision real input vector

I

Stride for *A*

B

Double-precision real input vector

J

Stride for *B*

C
: Double-precision real output vector

K
: Stride for *C*

N
: Count

Discussion

Performs the operation

$p = 0$

If $B_{nJ} \neq 0.0$ then $C_{pK} = A_{nI};\ p = p + 1;\qquad n = \{0, N-1\}$

Compresses vector *A* based on the nonzero values of gating vector *B*. For nonzero elements of *B*, corresponding elements of *A* are sequentially copied to output vector *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vdbcon

Vector convert power or amplitude to decibels.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vdbcon (float * A,

 int I,

 float * B,

 float * C,

 int K,

 unsigned int N,

 unsigned int F);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

B
: Single-precision real input scalar: zero reference

C
: Single-precision real output vector

K
: Stride for *C*

N
: Count

F
: Power (0) or amplitude (1) flag

Discussion

Performs the operation

$$C_{nK} = \alpha\left(\log_{10}\left(\frac{A_{nI}}{B}\right)\right) \qquad n = \{0, N\text{-}1\}$$

Converts inputs from vector *A* to their decibel equivalents, calculated in terms of power or amplitude according to flag *F*. As a relative reference point, the value of input scalar *B* is considered to be zero decibels.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vdbconD

Vector convert power or amplitude to decibels.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vdbconD (double * A,

 int I,

 double * B,

 double * C,

 int K,

 unsigned int N,

 unsigned int F);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

B
: Double-precision real input scalar: zero reference

C
: Double-precision real output vector

K
: Stride for *C*

N
: Count

F
: Power (0) or amplitude (1) flag

Discussion

Performs the operation

$$C_{nK} = \alpha\left(\log_{10}\left(\frac{A_{nI}}{B}\right)\right) \qquad n = \{0, N\text{-}1\}$$

Converts inputs from vector *A* to their decibel equivalents, calculated in terms of power or amplitude according to flag *F*. As a relative reference point, the value of input scalar *B* is considered to be zero decibels.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vdist

Vector distance.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vdist (float * A,

 int I,

 float * B,

 int J,

 float * C,

 int K,

 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
    Stride for *A*

B
    Single-precision real input vector

J
    Stride for *B*

C
    Single-precision real output vector

K
    Stride for *C*

N
    Count

Discussion
Performs the operation

$$C_{nk} = \sqrt{A_{ni}^2 + B_{nj}^2} \qquad n = \{0, N\text{-}1\}$$

Computes the square root of the sum of the squares of corresponding elements of vectors *A* and *B*, and stores the result in the corresponding element of vector *C*.

Availability
Available in Mac OS X v10.4 and later.

vDSP_vdistD

Vector distance.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vdistD (double * A,

 int I,

 double * B,

 int J,

 double * C,

 int K,

 unsigned int N);
```

Parameters

A
    Double-precision real input vector

I

Stride for *A*

B

Double-precision real input vector

J

Stride for *B*

C

Double-precision real output vector

K

Stride for *C*

N

Count

Discussion

Performs the operation

$$C_{nk} = \sqrt{A_{ni}^2 + B_{nj}^2} \qquad n = \{0, N\text{-}1\}$$

Computes the square root of the sum of the squares of corresponding elements of vectors *A* and *B*, and stores the result in the corresponding element of vector *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vdiv

Vector divide.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vdiv (float * A,

 int I,

 float * B,

 int J,

 float * C,

 int K,

 unsigned int N);
```

Parameters

A

Single-precision real input vector

I
: Stride for *A*

B
: Single-precision real input vector

J
: Stride for *B*

C
: Single-precision real output vector

K
: Stride for *C*

N
: Count

Discussion
Performs the operation

$$C_{nK} = \frac{B_{nI}}{A_{nJ}} \qquad n = \{0, N\text{-}1\}$$

Divides elements of vector *A* by corresponding elements of vector *B*, and stores the results in corresponding elements of vector *C*.

Availability
Available in Mac OS X v10.4 and later.

vDSP_vdivD

Vector divide.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vdivD (double * A,

 int I,

 double * B,

 int J,

 double * C,

 int K,

 unsigned int N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

B
: Double-precision real input vector

J
: Stride for *B*

C
: Double-precision real output vector

K
: Stride for *C*

N
: Count

Discussion

Performs the operation

$$C_{nK} = \frac{B_{nI}}{A_{nJ}} \qquad n = \{0, N-1\}$$

Divides elements of vector *A* by corresponding elements of vector *B*, and stores the results in corresponding elements of vector *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vdivi

Vector divide.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vdivi (int * A,

 int I,

 int * B,

 int J,

 int * C,

 int K,
 unsigned int N);
```

Parameters

A
: Integer input vector

I
: Stride for *A*

B
: Integer input vector

J
: Stride for *B*

C
: Integer output vector

K
: Stride for *C*

N
: Count

Discussion

Performs the operation

$$C_{nK} = \frac{B_{nI}}{A_{nJ}} \qquad n = \{0, N-1\}$$

Divides elements of vector *A* by corresponding elements of vector *B*, and stores the results in corresponding elements of vector *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vdpsp

Vector convert double-precision to single-precision.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vdpsp (double * A,
    int I,
    float * C,
    int K,
    unsigned int N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

C
: Single-precision real output vector

K
: Stride for *C*

N
: Count

Discussion

This performs the operation

$$C_{nK} = A_{nI}, \qquad n = \{0, N-1\}$$

Creates single-precision vector *C* by converting double-precision inputs from vector *A*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_venvlp

Vector envelope.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_venvlp (float * A,

 int I,

 float * B,

 int J,

 float * C,

 int K,

 float * D,

 int L,

 unsigned int N);
```

Parameters

A
: Single-precision real input vector: high envelope

I
: Stride for *A*

B
: Single-precision real input vector: low envelope

J
: Stride for *B*

C
: Single-precision real input vector

K
: Stride for *C*

D
: Single-precision real output vector

L
: Stride for *D*

N
: Count

Discussion

Performs the operation

$$\text{If } C_{nK} > A_{nI} \text{ or } C_{nK} < B_{nJ} \text{ then } D_{nM} = C_{nK}$$

$$\text{else } D_{nM} = 0.0 \qquad n = \{0, N-1\}$$

Finds the extrema of vector C. For each element of C, the corresponding element of A provides an upper-threshold value, and the corresponding element of B provides a lower-threshold value. If the value of an element of C falls outside the range defined by these thresholds, it is copied to the corresponding element of vector D. If its value is within the range, the corresponding element of vector D is set to zero.

Availability

Available in Mac OS X v10.4 and later.

vDSP_venvlpD

Vector envelope.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_venvlpD (double * A,

 int I,

 double * B,

 int J,

 double * C,
 int K,
 double * D,
 int L,
 unsigned int N);
```

Parameters

A
: Double-precision real input vector: high envelope

I
: Stride for *A*

B
: Double-precision real input vector: low envelope

J
: Stride for *B*

C
: Double-precision real input vector

K
: Stride for *C*

D
: Double-precision real output vector

L
: Stride for *D*

N
: Count

Discussion

Performs the operation

$$\text{If } \; C_{nK} > A_{nI} \;\text{ or }\; C_{nK} < B_{nJ} \;\text{ then }\; D_{nM} = C_{nK}$$

$$\text{else } \; D_{nM} = 0.0 \qquad n = \{0, N\text{-}1\}$$

Finds the extrema of vector C. For each element of C, the corresponding element of A provides an upper-threshold value, and the corresponding element of B provides a lower-threshold value. If the value of an element of C falls outside the range defined by these thresholds, it is copied to the corresponding element of vector D. If its value is within the range, the corresponding element of vector D is set to zero.

Availability

Available in Mac OS X v10.4 and later.

vDSP_veqvi

Vector equivalence, 32-bit logical.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_veqvi (int * A,

 int I,

 int * B,

 int J,

 int * C,

 int K,

 unsigned int N);
```

Parameters

A

Integer input vector

I

Stride for *A*

B

Integer input vector

J

Stride for *B*

C

Integer output vector

K

Stride for *C*

N

Count

Discussion

Performs the operation

$$C_{nk} = A_{ni} \bullet XNOR \bullet B_{nj} \qquad n = \{0, N-1\}$$

Outputs the bitwise logical equivalence, exclusive NOR, of the integers of vectors *A* and *B*. For each pair of input values, bits in each position are compared. A bit in the output value is set if both input bits areset, or both are clear; otherwise it is cleared.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vfill

Vector fill.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vfill (float * A,

 float * C,

 int K,

 unsigned int N);
```

Parameters

A

Single-precision real input scalar

C

Single-precision real output vector

K

Stride for *C*

N

Count

Discussion

Performs the operation

$C_{nK} = A \qquad n = \{0, N-1\}$

Sets each element of vector *C* to the value of *A*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vfillD

Vector fill.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vfillD (double * A,

 double * C,

 int K,

 unsigned int N);
```

Parameters

A

Double-precision real input scalar

C

Double-precision real output vector

K

Stride for *C*

N

Count

Discussion

Performs the operation

$$C_{nK} = A \qquad n = \{0, N\text{-}1\}$$

Sets each element of vector *C* to the value of *A*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vfilli

Integer vector fill.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vfilli (int * A,

 int * C,

 int K,

 unsigned int N);
```

Parameters

A

Integer input scalar

C

Integer output vector

K

Stride for *C*

N

Count

Discussion

Performs the operation

$C_{nK} = A \qquad n = \{0, N-1\}$

Sets each element of vector *C* to the value of *A*.

Availability
Available in Mac OS X v10.4 and later.

vDSP_vfrac

Vector truncate to fraction.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vfrac (float * A,

 int I,

 float * C,

 int K,

 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

C
: Single-precision real output vector

K
: Stride for *C*

N
: Count

Discussion
Performs the operation

$C_{nK} = A_{nI} - \text{truncate}\,(A_{nI}) \qquad n = \{0, N-1\}$

The “function” truncate(x) is the integer farthest from 0 but not farther than x. Thus, for example, `vDSP_vFrac(-3.25)` produces the result `-0.25`.

Sets each element of vector *C* to the signed fractional part of the corresponding element of *A*.

Availability
Available in Mac OS X v10.4 and later.

vDSP_vfracD

Vector truncate to fraction.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vfracD (double * A,

 int I,

 double * C,

 int K,

 unsigned int N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

C
: Double-precision real output vector

K
: Stride for *C*

N
: Count

Discussion

Performs the operation

$$C_{nK} = A_{nI} - \text{truncate}\,(A_{nI}) \qquad n = \{0, N\text{-}1\}$$

The “function” truncate(x) is the integer farthest from 0 but not farther than x. Thus, for example, `vDSP_vFrac(-3.25)` produces the result `-0.25`.

Sets each element of vector *C* to the signed fractional part of the corresponding element of *A*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vgathr

Vector gather.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vgathr (float * A,

 int * B,

 int J,

 float * C,

 int K,

 unsigned int N);
```

Parameters

A
: Single-precision real input vector

B
: Integer vector containing indices

J
: Stride for *B*

C
: Single-precision real output vector

K
: Stride for *C*

N
: Count

Discussion

Performs the operation

$$C_{nK} = A_{B_{nJ}} \qquad n = \{0, N-1\}$$

Uses elements of vector *B* as indices to copy selected elements of vector *A* to sequential locations in vector *C*. Note that 1, not zero, is treated as the first location in the input vector when evaluating indices. This function can only be done out of place.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vgathra

Vector gather, absolute pointers.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vgathra (float ** A,
 int I,

 float * C,

 int K,

 unsigned int N);
```

Parameters

A
: Pointer input vector

I
: Stride for *A*

C
: Single-precision real output vector

K
: Stride for *C*

N
: Count

Discussion

Performs the operation

$C_{nK} = *(A_{nI}) \qquad n = \{0, N\text{-}1\}$

Uses elements of vector *A* as pointers to copy selected single-precision values from memory to sequential locations in vector *C*. This function can only be done out of place.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vgathraD

Vector gather, absolute pointers.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vgathraD (double ** A,

 int I,

 double * C,

 int K,

 unsigned int N);
```

Parameters

A
: Pointer input vector

I
: Stride for *A*

C
: Double-precision real output vector

K
: Stride for *C*

N
: Count

Discussion

Performs the operation

$$C_{nK} = *(A_{nI}) \qquad n = \{0, N-1\}$$

Uses elements of vector *A* as pointers to copy selected double-precision values from memory to sequential locations in vector *C*. This function can only be done out of place.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vgathrD

Vector gather.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vgathrD (double * A,

 int * B,

 int J,

 double * C,

 int K,

 unsigned int N);
```

Parameters

A
: Double-precision real input vector

B
: Integer vector containing indices

J
: Stride for *B*

C
: Double-precision real output vector

K
: Stride for *C*

N
: Count

Discussion

Performs the operation

$$C_{nK} = A_{B_{nJ}} \qquad n = \{0, N-1\}$$

Uses elements of vector *B* as indices to copy selected elements of vector *A* to sequential locations in vector *C*. Note that 1, not zero, is treated as the first location in the input vector when evaluating indices. This function can only be done out of place.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vgen

Vector tapered ramp.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vgen (float * A,

 float * B,

 float * C,

 int K,

 unsigned int N);
```

Parameters

A
: Single-precision real input scalar: base value

B
: Single-precision real input scalar: end value

C
: Single-precision real output vector

K
: Stride for *C*

N

Count

Discussion

Performs the operation

$$C_{nK} = A + \frac{n(B-A)}{N-1} \qquad n = \{0, N\text{-}1\}$$

Creates ramped vector *C* with element zero equal to scalar *A* and element N–1 equal to scalar *B*. Output values between element zero and element N–1 are evenly spaced and increase or decrease monotonically.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vgenD

Vector tapered ramp.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vgenD (double * A,

 double * B,

 double * C,

 int K,

 unsigned int N);
```

Parameters

A

Double-precision real input scalar: base value

B

Double-precision real input scalar: end value

C

Double-precision real output vector

K

Stride for *C*

N

Count

Discussion

Performs the operation

$$C_{nK} = A + \frac{n(B-A)}{N-1} \qquad n = \{0, N\text{-}1\}$$

Creates ramped vector *C* with element zero equal to scalar *A* and element N–1 equal to scalar *B*. Output values between element zero and element N–1 are evenly spaced and increase or decrease monotonically.

Availability
Available in Mac OS X v10.4 and later.

vDSP_vgenp

Vector generate by extrapolation and interpolation.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vgenp (float * A,

 int I,

 float * B,

 int J,

 float * C,

 int K,

 unsigned int N,

 unsigned int M);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

B
: Single-precision real input vector

J
: Stride for *B*

C
: Single-precision real output vector

K
: Stride for *C*

N

Count for *C*

M

Count for *A* and *B*

Discussion

Performs the operation

$$C_{nK} = A_0 \qquad \text{for } 0 \le n \le \text{trunc}(B_0)$$

$$C_{nK} = A_{[M-1]I} \qquad \text{for } \text{trunc}(B_{[M-1]J}) < n \le N-1$$

$$C_{nK} = A_{mI} + \frac{A_\Delta\ (n - B_{mJ})}{B_\Delta} \qquad \text{for } \text{trunc}(B_{mJ}) < n \le \text{trunc}(B_{[m+1]J})$$

$$\text{where: } A_\Delta = A_{[m+1]I} - A_{mI}$$

$$B_\Delta = B_{[m+1]J} - B_{mI} \qquad \text{m} = \{0, \text{M-2}\}$$

Generates vector *C* by extrapolation and linear interpolation from the ordered pairs (A,B) provided by corresponding elements in vectors *A* and *B*. Vector *B* provides index values and should increase monotonically. Vector *A* provides intensities, magnitudes, or some other measurable quantities, one value associated with each value of *B*. This function can only be done out of place.

Vectors *A* and *B* define a piecewise linear function, f(x):

- In the interval [-infinity, trunc(*B*[0*J]], the function is the constant *A*[0*I].
- In each interval (trunc(*B*[m*J]), trunc(*B*[(m+1)*J])], the function is the line passing through the two points (*B*[m*J], *A*[m*I]) and (*B*[(m+1)*J], *A*[(m+1)*I]). (This is for each integer m, 0 <= m < *M*-1.)
- In the interval (*B*[(*M*-1)*J], infinity], the function is the constant *A*[(*M*-1)*I].
- For 0 <= n < *N*, *C*[n**K*] = f(n).

This function can only be done out of place.

Output values are generated for integral indices in the range zero through *N* - 1, deriving output values by interpolating and extrapolating from vectors *A* and *B*. For example, if vectors *A* and *B* define velocity and time pairs (v, t), `vDSP_vgenp` writes one velocity to vector *C* for every integral unit of time from zero to *N* - 1.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vgenpD

Vector generate by extrapolation and interpolation.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vgenpD (double * A,

 int I,

 double * B,

 int J,

 double * C,

 int K,

 unsigned int N,

 unsigned int M);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

B
: Double-precision real input vector

J
: Stride for *B*

C
: Double-precision real output vector

K
: Stride for *C*

N
: Count for *C*

M
: Count for *A* and *B*

Discussion

Performs the operation

$$C_{nK} = A_0 \qquad \text{for } 0 \le n \le \operatorname{trunc}(B_0)$$

$$C_{nK} = A_{[M-1]I} \qquad \text{for } \operatorname{trunc}(B_{[M-1]J}) < n \le N-1$$

$$C_{nK} = A_{mI} + \frac{A_\Delta\ (n-B_{mJ})}{B_\Delta} \qquad \text{for } \operatorname{trunc}(B_{mJ}) < n \le \operatorname{trunc}(B_{[m+1]J})$$

where: $A_\Delta = A_{[m+1]I} - A_{mI}$

$B_\Delta = B_{[m+1]J} - B_{mI}$ m = {0, M-2}

Generates vector *C* by extrapolation and linear interpolation from the ordered pairs (A,B) provided by corresponding elements in vectors *A* and *B*. Vector *B* provides index values and should increase monotonically. Vector *A* provides intensities, magnitudes, or some other measurable quantities, one value associated with each value of *B*. This function can only be done out of place.

Vectors *A* and *B* define a piecewise linear function, f(x):

- In the interval [-infinity, trunc(*B*[0*J]], the function is the constant *A*[0*I].
- In each interval (trunc(*B*[m*J]), trunc(*B*[(m+1)*J])], the function is the line passing through the two points (*B*[m*J], *A*[m*I]) and (*B*[(m+1)*J], *A*[(m+1)*I]). (This is for each integer m, 0 <= m < *M*-1.)
- In the interval (*B*[(*M*-1)*J], infinity], the function is the constant *A*[(*M*-1)*I].
- For 0 <= n < *N*, *C*[n**K*] = f(n).

This function can only be done out of place.

Output values are generated for integral indices in the range zero through *N* - 1, deriving output values by interpolating and extrapolating from vectors *A* and *B*. For example, if vectors *A* and *B* define velocity and time pairs (v, t), `vDSP_vgenp` writes one velocity to vector *C* for every integral unit of time from zero to *N* - 1.

Availability
Available in Mac OS X v10.4 and later.

vDSP_viclip

Vector inverted clip.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_viclip (float * A,

 int I,

 float * B,

 float * C,

 float * D,

 int L,

 unsigned int N);
```

Parameters

A
Single-precision real input vector

I
Stride for *A*

`B`
: Single-precision real input scalar: lower threshold

`C`
: Single-precision real input scalar: upper threshold

`D`
: Single-precision real output vector

`L`
: Stride for `D`

`N`
: Count

Discussion

Performs the operation

$$D_{nj} = A_{ni} \quad \text{if} \quad A_{ni} \le b$$

$$D_{nj} = A_{ni} \quad \text{if} \quad A_{ni} \ge c$$

$$D_{nj} = b \quad \text{if} \quad b < A_{ni} < 0.0$$

$$D_{nj} = c \quad \text{if} \quad 0.0 \le A_{ni} < c \qquad \text{n} = \{0, \text{N-1}\}$$

Performs an inverted clip of vector `A` using lower-threshold and upper-threshold input scalars `B` and `C`.

Availability

Available in Mac OS X v10.4 and later.

vDSP_viclipD

Vector inverted clip.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_viclipD (double * A,

 int I,

 double * B,

 double * C,

 double * D,

 int L,

 unsigned int N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

B
: Double-precision real input scalar: lower threshold

C
: Double-precision real input scalar: upper threshold

D
: Double-precision real output vector

L
: Stride for *D*

N
: Count

Discussion

Performs the operation

$$D_{nj} = A_{ni} \quad \text{if} \quad A_{ni} \le b$$

$$D_{nj} = A_{ni} \quad \text{if} \quad A_{ni} \ge c$$

$$D_{nj} = b \quad \text{if} \quad b < A_{ni} < 0.0$$

$$D_{nj} = c \quad \text{if} \quad 0.0 \le A_{ni} < c \qquad n = \{0, N-1\}$$

Performs an inverted clip of vector *A* using lower-threshold and upper-threshold input scalars *B* and *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vindex

Vector index.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vindex (float * A,

 float * B,

 int J,

 float * C,
 int K,

 unsigned int N);
```

Parameters

A
: Single-precision real input vector

B
: Single-precision real input vector: indices

J
: Stride for *B*

C
: Single-precision real output vector

K
: Stride for *C*

N
: Count

Discussion

Performs the operation

$$C_{nK} = A_{\mathrm{truncate}(B_{nJ})} \qquad \mathrm{n} = \{0, \mathrm{N}\text{-}1\}$$

Uses vector *B* as zero-based subscripts to copy selected elements of vector *A* to vector *C*. Fractional parts of vector *B* are ignored.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vindexD

Vector index.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vindexD (double * A,

 double * B,

 int J,

 double * C,

 int K,

 unsigned int N);
```

Parameters

A
: Double-precision real input vector

B
: Double-precision real input vector: indices

J
: Stride for *B*

C
: Double-precision real output vector

K
: Stride for *C*

N
: Count

Discussion

Performs the operation

$$C_{nK} = A_{\text{truncate}(B_{nJ})} \qquad \text{n = \{0, N-1\}}$$

Uses vector *B* as zero-based subscripts to copy selected elements of vector *A* to vector *C*. Fractional parts of vector *B* are ignored.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vintb

Vector linear interpolation between vectors.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vintb (float * A,

 int I,

 float * B,

 int J,

 float * C,

 float * D,

 int L,

 unsigned int N);
```

Parameters

`A`
: Single-precision real input vector

`I`
: Stride for `A`

`B`
: Single-precision real input vector

`J`
: Stride for `B`

`C`
: Single-precision real input scalar: interpolation constant

`D`
: Single-precision real output vector

`L`
: Stride for `D`

`N`
: Count

Discussion

Performs the operation

$$D_{nK} = A_{nI} + C[B_{nJ} - A_{nI}] \qquad n = \{0, N\text{-}1\}$$

Creates vector `D` by interpolating between vectors `A` and `B`.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vintbD

Vector linear interpolation between vectors.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vintbD (double * A,

 int I,

 double * B,

 int J,

 double * C,

 double * D,
 int L,

 unsigned int N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

B
: Double-precision real input vector

J
: Stride for *B*

C
: Double-precision real input scalar: interpolation constant

D
: Double-precision real output vector

L
: Stride for *D*

N
: Count

Discussion

Performs the operation

$$D_{nK} = A_{nI} + C[B_{nJ} - A_{nI}] \qquad n = \{0, N\text{-}1\}$$

Creates vector *D* by interpolating between vectors *A* and *B*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vlim

Vector test limit.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vlim (float * A,

 int I,

 float * B,

 float * C,
 float * D,

 int L,

 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

B
: Single-precision real input scalar: limit

C
: Single-precision real input scalar

D
: Single-precision real output vector

L
: Stride for *D*

N
: Count

Discussion

Compares values from vector *A* to limit scalar *B*. For inputs greater than or equal to *B*, scalar *C* is written to *D* . For inputs less than *B*, the negated value of scalar *C* is written to vector *D*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vlimD

Vector test limit.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vlimD (double * A,

 int I,

 double * B,

 double * C,

 double * D,

 int L,
 unsigned int N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

B
: Double-precision real input scalar: limit

C
: Double-precision real input scalar

D
: Double-precision real output vector

L
: Stride for *D*

N
: Count

Discussion

Compares values from vector *A* to limit scalar *B*. For inputs greater than or equal to *B*, scalar *C* is written to *D* . For inputs less than *B*, the negated value of scalar *C* is written to vector *D*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vlint

Vector linear interpolation between neighboring values.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vlint (float * A,

 float * B,

 int J,

 float * C,

 int K,

 unsigned int N,

 unsigned int M);
```

Parameters

A

Single-precision real input vector

B

Single-precision real input vector: integer parts are indices into *A* and fractional parts are interpolation constants

J

Stride for *B*

C

Single-precision real output vector

K

Stride for *C*

N

Count for C

M

Length of A

Discussion

Performs the operation

$$C_{nK} = A_{\beta} + \alpha(A_{\beta+1} - A_{\beta}) \qquad n = \{0, N-1\}$$

$$\text{where: } \beta = \text{trunc}(B_{nJ})$$
$$\alpha = B_{nJ} - \text{float}(\beta)$$

Generates vector *C* by interpolating between neighboring values of vector *A* as controlled by vector *B*. The integer portion of each element in *B* is the zero-based index of the first element of a pair of adjacent values in vector *A*.

The value of the corresponding element of *C* is derived from these two values by linear interpolation, using the fractional part of the value in B.

Argument *M* is not used in the calculation. However, the integer parts of the values in *B* must be greater than or equal to zero and less than or equal to *M* - 2.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vlintD

Vector linear interpolation between neighboring values.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vlintD (double * A,
 double * B,

 int J,

 double * C,

 int K,

 unsigned int N,

 unsigned int M);
```

Parameters

A
: Double-precision real input vector

B
: Double-precision real input vector: integer parts are indices into *A* and fractional parts are interpolation constants

J
: Stride for *B*

C
: Double-precision real output vector

K
: Stride for *C*

N
: Count for C

M
: Length of A

Discussion

Performs the operation

$$C_{nK} = A_{\beta} + \alpha(A_{\beta+1} - A_{\beta}) \qquad n = \{0, N\text{-}1\}$$

$$\text{where: } \beta = \text{trunc}(B_{nJ})$$
$$\alpha = B_{nJ} - \text{float}(\beta)$$

Generates vector *C* by interpolating between neighboring values of vector *A* as controlled by vector *B*. The integer portion of each element in *B* is the zero-based index of the first element of a pair of adjacent values in vector *A*.

The value of the corresponding element of *C* is derived from these two values by linear interpolation, using the fractional part of the value in B.

Argument *M* is not used in the calculation. However, the integer parts of the values in *B* must be greater than or equal to zero and less than or equal to *M* - 2.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vma

Vector multiply and add.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vma (float * A,

 int I,

 float * B,

 int J,

 float * C,

 int K,

 float * D,

 int L,

 unsigned int N);
```

Parameters

A

Single-precision real input vector

I

Stride for *A*

B

Single-precision real input vector

J

Stride for *B*

C

Single-precision real input vector

K

Stride for *C*

D

Single-precision real output vector

L

Stride for *D*

N

Count

Discussion

This performs the operation

$D_{nM} = A_{nI} \cdot B_{nJ} + C_{nK} \qquad n = \{0, N\text{-}1\}$

Multiplies corresponding elements of vectors `A` and `B`, add the corresponding elements of vector `C`, and stores the results in vector `D`.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vmaD

Vector multiply and add.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vmaD (double * A,

 int I,

 double * B,

 int J,

 double * C,

 int K,

 double * D,

 int L,

 unsigned int N);
```

Parameters

A

Double-precision real input vector

I

Stride for *A*

B

Double-precision real input vector

J

Stride for *B*

C

Double-precision real input vector

K

Stride for *C*

D

Double-precision real output vector

L

Stride for *D*

N

Count

Discussion

This performs the operation

$$D_{nM} = A_{nI} \bullet B_{nJ} + C_{nK} \qquad n = \{0, N\text{-}1\}$$

Multiplies corresponding elements of vectors *A* and *B*, add the corresponding elements of vector *C*, and stores the results in vector *D*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vmax

Vector maxima.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vmax (float * A,

 int I,

 float * B,

 int J,

 float * C,

 int K,

 unsigned int N);
```

Parameters

A

Single-precision real input vector

I

Stride for *A*

B

Single-precision real input vector

J

Stride for *B*

C
: Single-precision real output vector

K
: Stride for *C*

N
: Count

Discussion

This performs the operation

If $A_{nI} \geq B_{nJ}$ then $C_{nK} = A_{nI}$ else $C_{nK} = B_{nJ}$ n = {0, N-1}

Each element of output vector *D* is the greater of the corresponding values from input vectors *A* and *B*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vmaxD

Vector maxima.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vmaxD (double * A,

 int I,

 double * B,

 int J,

 double * C,

 int K,

 unsigned int N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

B
: Double-precision real input vector

J
: Stride for *B*

C
: Double-precision real output vector

K
: Stride for *C*

N
: Count

Discussion

This performs the operation

If $A_{nI} \geq B_{nJ}$ then $C_{nK} = A_{nI}$ else $C_{nK} = B_{nJ}$ $n = \{0, N\text{-}1\}$

Each element of output vector *D* is the greater of the corresponding values from input vectors *A* and *B*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vmaxmg

Vector maximum magnitudes.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vmaxmg (float * A,

 int I,

 float * B,

 int J,

 float * C,

 int K,

 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

B
: Single-precision real input vector

J
: Stride for *B*

C

Single-precision real output vector

K

Stride for *C*

N

Count

Discussion

This performs the operation

If $|A_{nI}| \geq |B_{nJ}|$ then $C_{nK} = |A_{nI}|$ else $C_{nK} = |B_{nJ}|$ n = {0, N-1}

Each element of output vector *D* is the larger of the magnitudes of corresponding values from input vectors *A* and *B*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vmaxmgD

Vector maximum magnitudes.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vmaxmgD (double * A,

 int I,

 double * B,

 int J,

 double * C,

 int K,

 unsigned int N);
```

Parameters

A

Double-precision real input vector

I

Stride for *A*

B

Double-precision real input vector

J

Stride for *B*

C

Double-precision real output vector

K

Stride for *C*

N

Count

Discussion

This performs the operation

$$\text{If} \quad |A_{nI}| \geq |B_{nJ}| \quad \text{then} \quad C_{nK} = |A_{nI}| \quad \text{else} \quad C_{nK} = |B_{nJ}| \qquad n = \{0, N\text{-}1\}$$

Each element of output vector *D* is the larger of the magnitudes of corresponding values from input vectors *A* and *B*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vmin

Vector minima.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vmin (float * A,

 int I,

 float * B,

 int J,

 float * C,

 int K,

 unsigned int N);
```

Parameters

A

Single-precision real input vector

I

Stride for *A*

B

Single-precision real input vector

J

Stride for *B*

C

Single-precision real output vector

K

Stride for *C*

N

Count

Discussion

This performs the operation

If $A_{nI} \leq B_{nJ}$ then $C_{nK} = A_{nI}$ else $C_{nK} = B_{nJ}$ $n = \{0, N-1\}$

Each element of output vector *D* is the lesser of the corresponding values from input vectors *A* and *B*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vminD

Vector minima.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vminD (double * A,

 int I,

 double * B,

 int J,

 double * C,

 int K,

 unsigned int N);
```

Parameters

A

Double-precision real input vector

I

Stride for *A*

B

Double-precision real input vector

J

Stride for *B*

C

Double-precision real output vector

K

Stride for *C*

N

Count

Discussion

This performs the operation

If $A_{nI} \leq B_{nJ}$ then $C_{nK} = A_{nI}$ else $C_{nK} = B_{nJ}$ n = {0, N-1}

Each element of output vector *D* is the lesser of the corresponding values from input vectors *A* and *B*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vminmg

Vector minimum magnitudes.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vmin (float * A,

 int I,

 float * B,

 int J,

 float * C,

 int K,

 unsigned int N);
```

Parameters

A

Single-precision real input vector

I

Stride for *A*

B

Single-precision real input vector

J

Stride for *B*

C
: Single-precision real output vector

K
: Stride for *C*

N
: Count

Discussion

This performs the operation

If $|A_{nI}| \le |B_{nJ}|$ then $C_{nK} = |A_{nI}|$ else $C_{nK} = |B_{nJ}|$ n = {0, N-1}

Each element of output vector *D* is the smaller of the magnitudes of corresponding values from input vectors *A* and *B*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vminmgD

Vector minimum magnitudes.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vminD (double * A,

 int I,

 double * B,

 int J,

 double * C,

 int K,

 unsigned int N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

B
: Double-precision real input vector

J
: Stride for *B*

C
: Double-precision real output vector

K
: Stride for *C*

N
: Count

Discussion

This performs the operation

$$\text{If} \quad |A_{nI}| \leq |B_{nJ}| \quad \text{then} \quad C_{nK} = |A_{nI}| \quad \text{else} \quad C_{nK} = |B_{nJ}| \qquad n = \{0, N\text{-}1\}$$

Each element of output vector *D* is the smaller of the magnitudes of corresponding values from input vectors *A* and *B*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vmma

Vector multiply, multiply, and add.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vmma (float * A,

 int I,

 float * B,

 int J,

 float * C,

 int K,

 float * D,

 int L,

 float * E,

 int M,

 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

B
: Single-precision real input vector

J
: Stride for *B*

C
: Single-precision real input vector

K
: Stride for *C*

D
: Single-precision real input vector

L
: Stride for *D*

E
: Single-precision real output vector

M
: Stride for *E*

N
: Count

Discussion

This performs the operation

$$E_{nM} = A_{nI} \cdot B_{nJ} + C_{nK} \cdot D_{nL} \qquad n = \{0, N-1\}$$

Corresponding elements of *A* and *B* are multiplied, corresponding values of *C* and *D* are multiplied, and these products are added together and stored in *E*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vmmaD

Vector multiply, multiply, and add.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vmmaD (double * A,

 int I,

 double * B,
 int J,

 double * C,

 int K,

 double * D,

 int L,

 double * E,

 int M,

 unsigned int N);
```

Parameters

A
Double-precision real input vector

I
Stride for *A*

B
Double-precision real input vector

J
Stride for *B*

C
Double-precision real input vector

K
Stride for *C*

D
Double-precision real input vector

L
Stride for *D*

E
Double-precision real output vector

M
Stride for *E*

N
Count

Discussion

This performs the operation

$$E_{nM} = A_{nI} \bullet B_{nJ} + C_{nK} \bullet D_{nL} \qquad n = \{0, N-1\}$$

Corresponding elements of *A* and *B* are multiplied, corresponding values of *C* and *D* are multiplied, and these products are added together and stored in *E*.

Availability
Available in Mac OS X v10.4 and later.

vDSP_vmmsb

Vector multiply, multiply, and subtract.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vmmsb (float * A,

 int I,

 float * B,

 int J,

 float * C,

 int K,

 float * D,

 int L,

 float * E,

 int M,

 unsigned int N);
```

Parameters

A
Single-precision real input vector

I
Stride for *A*

B
Single-precision real input vector

J
Stride for *B*

C
Single-precision real input vector

K
Stride for *C*

`D`
: Single-precision real input vector

`L`
: Stride for `D`

`E`
: Single-precision real output vector

`M`
: Stride for `E`

`N`
: Count

Discussion

This performs the operation

$$E_{nM} = A_{nI} B_{nJ} - C_{nK} D_{nL} \qquad n = \{0, N\text{-}1\}$$

Corresponding elements of `A` and `B` are multiplied, corresponding values of `C` and `D` are multiplied, and the second product is subtracted from the first. The result is stored in `E`.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vmmsbD

Vector multiply, multiply, and subtract.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vmmsbD (double * A,

 int I,

 double * B,

 int J,

 double * C,

 int K,

 double * D,

 int L,

 double * E,

 int M,

 unsigned int N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

B
: Double-precision real input vector

J
: Stride for *B*

C
: Double-precision real input vector

K
: Stride for *C*

D
: Double-precision real input vector

L
: Stride for *D*

E
: Double-precision real output vector

M
: Stride for *E*

N
: Count

Discussion

This performs the operation

$$E_{nM} = A_{nI} B_{nJ} - C_{nK} D_{nL} \qquad \mathrm{n} = \{0, \mathrm{N}\text{-}1\}$$

Corresponding elements of *A* and *B* are multiplied, corresponding values of *C* and *D* are multiplied, and the second product is subtracted from the first. The result is stored in *E*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vmsa

Vector multiply and scalar add.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vmsa (float * A,
 int I,
 float * B,
 int J,
 float * C,
 float * D,
 int L,
 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

B
: Single-precision real input vector

J
: Stride for *B*

C
: Single-precision real input scalar

D
: Single-precision real output vector

L
: Stride for *D*

N
: Count

Discussion

This performs the operation

$$D_{nK} = A_{nI} \bullet B_{nJ} + C \qquad n = \{0, N\text{-}1\}$$

Corresponding elements of *A* and *B* are multiplied and the scalar *C* is added. The result is stored in *D*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vmsaD

Vector multiply and scalar add.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vmsaD (double * A,

 int I,

 double * B,

 int J,

 double * C,

 double * D,

 int L,

 unsigned int N);
```

Parameters

A
    Double-precision real input vector

I
    Stride for *A*

B
    Double-precision real input vector

J
    Stride for *B*

C
    Double-precision real input scalar

D
    Double-precision real output vector

L
    Stride for *D*

N
    Count

Discussion

This performs the operation

$$D_{nK} = A_{nI} \cdot B_{nJ} + C \qquad n = \{0, N-1\}$$

Corresponding elements of *A* and *B* are multiplied and the scalar *C* is added. The result is stored in *D*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vmsb

Vector multiply and subtract.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vmsb (float * A,

 int I,

 float * B,

 int J,

 float * C,

 int K,

 float * D,

 int L,

 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

B
: Single-precision real input vector

J
: Stride for *B*

C
: Single-precision real input vector

K
: Stride for *C*

D
: Single-precision real output vector

L
: Stride for *D*

N
: Count

Discussion

This performs the operation

$$D_{nM} = A_{nI} \cdot B_{nJ} - C_{nK} \qquad n = \{0, N\text{-}1\}$$

Corresponding elements of *A* and *B* are multiplied and the corresponding value of *C* is subtracted. The result is stored in *D*.

Availability
Available in Mac OS X v10.4 and later.

vDSP_vmsbD

Vector multiply and subtract.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vmsbD (double * A,

 int I,

 double * B,

 int J,

 double * C,

 int K,

 double * D,

 int L,

 unsigned int N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

B
: Double-precision real input vector

J
: Stride for *B*

C
: Double-precision real input vector

K
: Stride for *C*

D
: Double-precision real output vector

L
: Stride for *D*

`N`

Count

Discussion

This performs the operation

$$D_{nM} = A_{nI} \cdot B_{nJ} - C_{nK} \qquad n = \{0, N-1\}$$

Corresponding elements of `A` and `B` are multiplied and the corresponding value of `C` is subtracted. The result is stored in `D`.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vmul

Multiplies vector `signal1` by vector `signal2` and leaves the result in vector result.

```
void
vDSP_vmul (const float A[],
    SInt32 I,
    const float B[],
    SInt32 J,
    float C[],
    SInt32 K,
    UInt32 N);
```

Discussion

This performs the operation

$$C_{nK} = A_{nI} \cdot B_{nJ} \qquad n = \{0, N-1\}$$

Criteria to invoke vectorized code:

- The vectors `signal1`,`signal2`, and `result` must be relatively aligned.
- The value of `size` must be equal to or greater than 8.
- The values of `signal1stride`, `signal2stride`, and `resultStride` must be 1.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vmulD

Multiplies vector `signal1` by vector `signal2` and leaves the result in vector result.

```
void
vDSP_vmulD (const double A[],
    SInt32 I,
    const double B[],
    SInt32 J,
    double C[],
    SInt32 K,
    UInt32 N);
```

Discussion

This performs the operation

$$C_{nK} = A_{nI} \bullet B_{nJ} \qquad n = \{0, N-1\}$$

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vnabs

Vector negative absolute value.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vnabs (float * A,

 int I,

 float * C,

 int K,

 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

C
: Single-precision real output vector

K
: Stride for *C*

N
: Count

Discussion

This performs the operation

$$C_{nK} = -|A_{nI}| \qquad n = \{0, N-1\}$$

Each value in *C* is the negated absolute value of the corresponding element in *A*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vnabsD

Vector negative absolute value.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vnabsD (double * A,

 int I,

 double * C,

 int K,

 unsigned int N);
```

Parameters

A

Double-precision real input vector

I

Stride for *A*

C

Double-precision real output vector

K

Stride for *C*

N

Count

Discussion

This performs the operation

$$C_{nK} = -|A_{nI}| \qquad n = \{0, N-1\}$$

Each value in *C* is the negated absolute value of the corresponding element in *A*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vneg

Vector negative value.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vneg (float * A,

 int I,

 float * C,

 int K,

 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

C
: Single-precision real output vector

K
: Stride for *C*

N
: Count

Discussion

Each value in *C* is the negated value of the corresponding element in *A*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vnegD

Vector negative value.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vnegD (double * A,

 int I,

 double * C,

 int K,

 unsigned int N);
```

Parameters

A

Double-precision real input vector

I

Stride for *A*

C

Double-precision real output vector

K

Stride for *C*

N

Count

Discussion

Each value in *C* is the negated value of the corresponding element in *A*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vpoly

Vector polynomial.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vpoly (float * A,

 int I,

 float * B,

 int J,

 float * C,

 int K,

 unsigned int N,

 unsigned int P);
```

Parameters

A

Single-precision real input vector: coefficients

I

Stride for *A*

B

Single-precision real input vector: variable values

J
Stride for *B*

C
Single-precision real output vector

K
Stride for *C*

N
Count

P
Degree of polynomial

Discussion
Performs the operation

$$C_{nK} = \sum_{p=0}^{P} A_{pI} \cdot B_{nJ}^{P-p} \qquad n = \{0, N\text{-}1\}$$

Evaluates polynomials using vector *B* as independent variables and vector *A* as coefficients. A polynomial of degree p requires p+1 coefficients, so vector *A* should contain *P*+1 values.

Availability
Available in Mac OS X v10.4 and later.

vDSP_vpolyD

Vector polynomial.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vpolyD (double * A,

 int I,

 double * B,

 int J,

 double * C,

 int K,

 unsigned int N,

 unsigned int P);
```

Parameters

A
: Double-precision real input vector: coefficients

I
: Stride for *A*

B
: Double-precision real input vector: variable values

J
: Stride for *B*

C
: Double-precision real output vector

K
: Stride for *C*

N
: Count

P
: Degree of polynomial

Discussion

Performs the operation

$$C_{nK} = \sum_{p=0}^{P} A_{pI} \cdot B_{nJ}^{P-p} \qquad n = \{0, N\text{-}1\}$$

Evaluates polynomials using vector *B* as independent variables and vector *A* as coefficients. A polynomial of degree p requires p+1 coefficients, so vector *A* should contain *P*+1 values.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vpythg

Vector pythagoras.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vpythg (float * A,

 int I,

 float * B,

 int J,
 float * C,
 int K,
 float * D,
 int L,
 float * E,
 int M,
 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

B
: Single-precision real input vector

J
: Stride for *B*

C
: Single-precision real input vector

K
: Stride for *C*

D
: Single-precision real input vector

L
: Stride for *D*

E
: Single-precision real output vector

M
: Stride for *E*

N
: Count

Discussion

Performs the operation

$$E_{nM} = \sqrt{(A_{nI} - C_{nK})^2 + (B_{nJ} - D_{nL})^2} \qquad n = \{0, N-1\}$$

Subtracts vector *C* from *A* and squares the differences, subtracts vector *D* from *B* and squares the differences, adds the two sets of squared differences, and then writes the square roots of the sums to vector *E*.

Availability
Available in Mac OS X v10.4 and later.

vDSP_vpythgD

Vector pythagoras.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vpythgD (double * A,

 int I,

 double * B,

 int J,

 double * C,

 int K,

 double * D,

 int L,

 double * E,

 int M,

 unsigned int N);
```

Parameters

A
    Double-precision real input vector

I
    Stride for *A*

B
    Double-precision real input vector

J
    Stride for *B*

C
    Double-precision real input vector

K
    Stride for *C*

D
Double-precision real input vector

L
Stride for *D*

E
Double-precision real output vector

M
Stride for *E*

N
Count

Discussion
Performs the operation

$$E_{nM} = \sqrt{(A_{nI} - C_{nK})^2 + (B_{nJ} - D_{nL})^2} \qquad n = \{0, N-1\}$$

Subtracts vector *C* from *A* and squares the differences, subtracts vector *D* from *B* and squares the differences, adds the two sets of squared differences, and then writes the square roots of the sums to vector *E*.

Availability
Available in Mac OS X v10.4 and later.

vDSP_vqint

Vector quadratic interpolation.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vqint (float * A,

 float * B,

 int J,

 float * C,

 int K,

 unsigned int N,

 unsigned int M);
```

Parameters
A
Single-precision real input vector

B
: Single-precision real input vector: integer parts are indices into *A* and fractional parts are interpolation constants

J
: Stride for *B*

C
: Single-precision real output vector

K
: Stride for *C*

N
: Count for C

M
: Length of A: must be greater than or equal to 3

Discussion

Performs the operation

$$C_{nK} = \frac{A_{\beta-1}[\alpha^2 - \alpha] + A_{\beta}[2.0 - 2.0\alpha^2] + A_{\beta+1}[\alpha^2 + \alpha]}{2}$$

where: $\beta = \max(\text{trunc}(B_{nJ}), 1)$ $\qquad n = \{0, N\text{-}1\}$

$\alpha = B_{nJ} - \text{float}(\beta)$

Generates vector *C* by interpolating between neighboring values of vector *A* as controlled by vector *B*. The integer portion of each element in *B* is the zero-based index of the first element of a triple of adjacent values in vector *A*.

The value of the corresponding element of *C* is derived from these three values by quadratic interpolation, using the fractional part of the value in B.

Argument *M* is not used in the calculation. However, the integer parts of the values in *B* must be less than or equal to *M* - 2.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vqintD

Vector quadratic interpolation.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vqintD (double * A,

 double * B,
 int J,
 double * C,
 int K,
 unsigned int N,
 unsigned int M);
```

Parameters

A
: Double-precision real input vector

B
: Double-precision real input vector: integer parts are indices into *A* and fractional parts are interpolation constants

J
: Stride for *B*

C
: Double-precision real output vector

K
: Stride for *C*

N
: Count for C

M
: Length of A: must be greater than or equal to 3

Discussion

Performs the operation

$$C_{nK} = \frac{A_{\beta-1}[\alpha^2 - \alpha] + A_{\beta}[2.0 - 2.0\alpha^2] + A_{\beta+1}[\alpha^2 + \alpha]}{2}$$

where: $\beta = \max(\mathrm{trunc}(B_{nJ}), 1)$ $\quad n = \{0, N-1\}$

$\alpha = B_{nJ} - \mathrm{float}(\beta)$

Generates vector *C* by interpolating between neighboring values of vector *A* as controlled by vector *B*. The integer portion of each element in *B* is the zero-based index of the first element of a triple of adjacent values in vector *A*.

The value of the corresponding element of *C* is derived from these three values by quadratic interpolation, using the fractional part of the value in B.

Argument *M* is not used in the calculation. However, the integer parts of the values in *B* must be less than or equal to *M* - 2.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vramp

Build ramped vector.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vramp (float * A,

 float * B,

 float * C,

 int K,

 unsigned int N);
```

Parameters

A
: Single-precision real input scalar: initial value

B
: Single-precision real input scalar: increment or decrement

C
: Single-precision real output vector

K
: Stride for `C`

N
: Count

Discussion

Performs the operation

$$C_{nk} = a + nb \qquad n = \{0, N-1\}$$

Creates a monotonically incrementing or decrementing vector. Scalar `A` is the initial value written to vector `C`. Scalar `B` is the increment or decrement for each succeeding element.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vrampD

Build ramped vector.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vrampD (double * A,
 double * B,

 double * C,

 int K,

 unsigned int N);
```

Parameters

A
: Double-precision real input scalar: initial value

B
: Double-precision real input scalar: increment or decrement

C
: Double-precision real output vector

K
: Stride for *C*

N
: Count

Discussion

Performs the operation

$$C_{nk} = a + nb \qquad \text{n} = \{0, \text{N-1}\}$$

Creates a monotonically incrementing or decrementing vector. Scalar *A* is the initial value written to vector *C*. Scalar *B* is the increment or decrement for each succeeding element.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vrsum

Vector running sum integration.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vrsum (float * A,

 int I,

 float * S,

 float * C,

 int K,

 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

S
: Single-precision real input scalar: weighting factor

C
: Single-precision real output vector

K
: Stride for *C*

N
: Count

Discussion

Performs the operation

$$C_0 = 0$$

$$C_{mK} = C_{(m-1)K} + SA_{mI} \qquad \text{m = \{1, N-1\}}$$

Integrates vector *A* using a running sum from vector *C*. Vector *A* is weighted by scalar *S* and added to the previous output point. The first element from vector *A* is not used in the sum.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vrsumD

Vector running sum integration.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vrsumD (double * A,

 int I,

 double * S,

 double * C,

 int K,

 unsigned int N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

S
: Double-precision real input scalar: weighting factor

C
: Double-precision real output vector

K
: Stride for *C*

N
: Count

Discussion

Performs the operation

$$C_0 = 0$$

$$C_{mK} = C_{(m-1)K} + SA_{mI} \qquad m = \{1, N\text{-}1\}$$

Integrates vector *A* using a running sum from vector *C*. Vector *A* is weighted by scalar *S* and added to the previous output point. The first element from vector *A* is not used in the sum.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vrvrs

Vector reverse order, in place.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vrvrs (float * C,

 int K,

 unsigned int N);
```

Parameters

C
: Single-precision real input-output vector

K
: Stride for *C*

N

Count

Discussion

Performs the operation

$C_{nK} \leftrightarrow C_{[N-n-1]K}$ n = {0, (N/2)-1}

Reverses the order of vector C in place.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vrvrsD

Vector reverse order, in place.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vrvrsD (double * C,

 int K,

 unsigned int N);
```

Parameters

C

Double-precision real input-output vector

K

Stride for *C*

N

Count

Discussion

Performs the operation

$C_{nK} \leftrightarrow C_{[N-n-1]K}$ n = {0, (N/2)-1}

Reverses the order of vector C in place.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vsadd

Vector scalar add.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vsadd (float * A,

 int I,

 float * B,

 float * C,

 int K,

 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

B
: Single-precision real input scalar

C
: Single-precision real output vector

K
: Stride for *C*

N
: Count

Discussion

Performs the operation

$$C_{nK} = A_{nI} + B \qquad n = \{0, N-1\}$$

Adds scalar *B* to each element of vector *A* and stores the result in the corresponding element of vector *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vsaddD

Vector scalar add.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vsaddD (double * A,

 int I,
 double * B,

 double * C,

 int K,

 unsigned int N);
```

Parameters

A
Double-precision real input vector

I
Stride for *A*

B
Double-precision real input scalar

C
Double-precision real output vector

K
Stride for *C*

N
Count

Discussion

Performs the operation

$$C_{nK} = A_{nI} + B \qquad n = \{0, N\text{-}1\}$$

Adds scalar *B* to each element of vector *A* and stores the result in the corresponding element of vector *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vsaddi

Integer vector scalar add.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vsaddi (int * A,

 int I,

 int * B,

 int * C,
 int K,

 unsigned int N);
```

Parameters

A
: Integer input vector

I
: Stride for *A*

B
: Integer input scalar

C
: Integer output vector

K
: Stride for *C*

N
: Count

Discussion

Performs the operation

$$C_{nK} = A_{nI} + B \qquad n = \{0, N-1\}$$

Adds scalar *B* to each element of vector *A* and stores the result in the corresponding element of vector *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vsbm

Vector subtract and multiply.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vsbm (float * A,

 int I,

 float * B,

 int J,

 float * C,

 int K,
 float * D,
 int L,
 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

B
: Single-precision real input vector

J
: Stride for *B*

C
: Single-precision real input vector

K
: Stride for *C*

D
: Single-precision real output vector

L
: Stride for *D*

N
: Count

Discussion

Performs the operation

$$D_{nM} = (A_{nI} - B_{nJ}) C_{nK} \qquad n = \{0, N\text{-}1\}$$

Subtracts vector *B* from vector *A* and then multiplies the differences by vector *C*. Results are stored in vector *D*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vsbmD

Vector subtract and multiply.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vsbmD (double * A,
 int I,
 double * B,
 int J,
 double * C,
 int K,
 double * D,
 int L,
 unsigned int N);
```

Parameters

A
Double-precision real input vector

I
Stride for *A*

B
Double-precision real input vector

J
Double for *B*

C
Double-precision real input vector

K
Stride for *C*

D
Double-precision real output vector

L
Stride for *D*

N
Count

Discussion

Performs the operation

$$D_{nM} = (A_{nI} - B_{nJ})\,C_{nK} \qquad \text{n} = \{0, \text{N-1}\}$$

Subtracts vector *B* from vector *A* and then multiplies the differences by vector *C*. Results are stored in vector *D*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vsbsbm

Vector subtract, subtract, and multiply.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vsbsbm (float * A,

 int I,

 float * B,

 int J,

 float * C,

 int K,

 float * D,

 int L,

 float * E,

 int M,

 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

B
: Single-precision real input vector

J
: Stride for *B*

C
: Single-precision real input vector

K
: Stride for *C*

D
: Single-precision real input vector

L
: Stride for *D*

E
: Single-precision real output vector

M

Stride for *E*

N

Count

Discussion

Performs the operation

$$E_{nM} = (A_{nI} - B_{nJ})(C_{nK} - D_{nL}) \qquad n = \{0, N\text{-}1\}$$

Subtracts vector *B* from *A*, subtracts vector *D* from *C*, and multiplies the differences. Results are stored in vector *E*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vsbsbmD

Vector subtract, subtract, and multiply.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vsbsbmD (double * A,

 int I,

 double * B,

 int J,

 double * C,

 int K,

 double * D,

 int L,

 double * E,

 int M,

 unsigned int N);
```

Parameters

A

Double-precision real input vector

I

Stride for *A*

B
: Double-precision real input vector

J
: Stride for *B*

C
: Double-precision real input vector

K
: Stride for *C*

D
: Double-precision real input vector

L
: Stride for *D*

E
: Double-precision real output vector

M
: Stride for *E*

N
: Count

Discussion

Performs the operation

$$E_{nM} = (A_{nI} - B_{nJ})(C_{nK} - D_{nL}) \qquad n = \{0, N-1\}$$

Subtracts vector *B* from *A*, subtracts vector *D* from *C*, and multiplies the differences. Results are stored in vector *E*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vsbsm

Vector subtract and scalar multiply.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vsbsm (float * A,

 int I,

 float * B,

 int J,

 float * C,
 float * D,

 int L,

 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

B
: Single-precision real input vector

J
: Stride for *B*

C
: Single-precision real input scalar

D
: Single-precision real output vector

L
: Stride for *D*

N
: Count

Discussion

Performs the operation

$$D_{nK} = (A_{nI} - B_{nJ})C \qquad n = \{0, N-1\}$$

Subtracts vector *B* from vector *A* and then multiplies each difference by scalar *C*. Results are stored in vector *D*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vsbsmD

Vector subtract and scalar multiply.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vsbsmD (double * A,

 int I,
 double * B,
 int J,
 double * C,
 double * D,
 int L,
 unsigned int N);
```

Parameters

A
Double-precision real input vector

I
Stride for *A*

B
Double-precision real input vector

J
Stride for *B*

C
Double-precision real input scalar

D
Double-precision real output vector

L
Stride for *D*

N
Count

Discussion

Performs the operation

$$D_{nK} = (A_{nI} - B_{nJ})C \qquad n = \{0, N\text{-}1\}$$

Subtracts vector *B* from vector *A* and then multiplies each difference by scalar *C*. Results are stored in vector *D*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vsdiv

Vector scalar divide.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vsdiv (float * A,

 int I,

 float * B,

 float * C,

 int K,

 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

B
: Single-precision real input scalar

C
: Single-precision real output vector

K
: Stride for *C*

N
: Count

Discussion

Performs the operation

$$C_{nK} = \frac{A_{nI}}{B} \qquad n = \{0, N-1\}$$

Divides each element of vector *A* by scalar *B* and stores the result in the corresponding element of vector *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vsdivD

Vector scalar divide.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vsdivD (double * A,
 int I,

 double * B,

 double * C,

 int K,

 unsigned int N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

B
: Double-precision real input scalar

C
: Double-precision real output vector

K
: Stride for *C*

N
: Count

Discussion

Performs the operation

$$C_{nK} = \frac{A_{nI}}{B} \qquad n = \{0, N-1\}$$

Divides each element of vector *A* by scalar *B* and stores the result in the corresponding element of vector *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vsdivi

Integer vector scalar divide.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vsdivi (int * A,

 int I,

 int * B,
 int * C,

 int K,

 unsigned int N);
```

Parameters

A
: Integer input vector

I
: Stride for *A*

B
: Integer input scalar

C
: Integer output vector

K
: Stride for *C*

N
: Count

Discussion

Performs the operation

$$C_{nK} = \frac{A_{nI}}{B} \qquad n = \{0, N\text{-}1\}$$

Divides each element of vector *A* by scalar *B* and stores the result in the corresponding element of vector *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vsimps

Simpson integration.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vsimps (float * A,

 int I,

 float * B,

 float * C,

 int K,
 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

B
: Single-precision real input scalar

C
: Single-precision real output vector

K
: Stride for *C*

N
: Count

Discussion

Performs the operation

$$C_0 = 0.0$$

$$C_K = \frac{[A_0 + A_I]B}{2}$$

$$C_{nK} = C_{[n-2]K} + \frac{[A_{[n-2]I} + 4.0 \times A_{[n-1]I} + A_{nI}]B}{3} \qquad n = \{2, N\text{-}1\}$$

Integrates vector *A* using Simpson integration, storing results in vector *C*. Scalar *B* specifies the integration step size. This function can only be done out of place.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vsimpsD

Simpson integration.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vsimpsD (double * A,

 int I,

 double * B,

 double * C,
 int K,

 unsigned int N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

B
: Double-precision real input scalar

C
: Double-precision real output vector

K
: Stride for *C*

N
: Count

Discussion

Performs the operation

$$C_0 = 0.0$$

$$C_K = \frac{[A_0 + A_I]B}{2}$$

$$C_{nK} = C_{[n-2]K} + \frac{[A_{[n-2]I} + 4.0 \times A_{[n-1]I} + A_{nI}]B}{3} \qquad n = \{2, N\text{-}1\}$$

Integrates vector *A* using Simpson integration, storing results in vector *C*. Scalar *B* specifies the integration step size. This function can only be done out of place.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vsma

Vector scalar multiply and vector add.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vsma (float * A,

 int I,

 float * B,
 float * C,
 int K,
 float * D,
 int L,
 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

B
: Single-precision real input scalar

C
: Single-precision real input vector

K
: Stride for *C*

D
: Single-precision real output vector

L
: Stride for *D*

N
: Count

Discussion

Performs the operation

$$D_{nM} = A_{nI} \bullet B + C_{nK} \qquad n = \{0, N-1\}$$

Multiplies vector *A* by scalar *B* and then adds the products to vector *C*. Results are stored in vector *D*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vsmaD

Vector scalar multiply and vector add.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vsmaD (double * A,
 int I,
 double * B,
 double * C,
 int K,
 double * D,
 int L,
 unsigned int N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

B
: Double-precision real input scalar

C
: Double-precision real input vector

K
: Stride for *C*

D
: Double-precision real output vector

L
: Stride for *D*

N
: Count

Discussion

Performs the operation

$$D_{nM} = A_{nI} \bullet B + C_{nK} \qquad \mathrm{n} = \{0, \mathrm{N}-1\}$$

Multiplies vector *A* by scalar *B* and then adds the products to vector *C*. Results are stored in vector *D*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vsmsa

Vector scalar multiply and scalar add.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vsmsa (float * A,

 int I,

 float * B,

 float * C,

 float * D,

 int L,

 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

B
: Single-precision real input scalar

C
: Single-precision real input scalar

D
: Single-precision real output vector

L
: Stride for *D*

N
: Count

Discussion
Performs the operation

$$D_{nM} = A_{nI} \bullet B + C \qquad n = \{0, N-1\}$$

Multiplies vector *A* by scalar *B* and then adds scalar *C* to each product. Results are stored in vector *D*.

Availability
Available in Mac OS X v10.4 and later.

vDSP_vsmsaD

Vector scalar multiply and scalar add.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vsmsaD (double * A,

 int I,

 double * B,

 double * C,

 double * D,

 int L,

 unsigned int N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

B
: Double-precision real input scalar

C
: Double-precision real input scalar

D
: Double-precision real output vector

L
: Stride for *D*

N
: Count

Discussion

Performs the operation

$$D_{nM} = A_{nI} \cdot B + C \qquad n = \{0, N-1\}$$

Multiplies vector *A* by scalar *B* and then adds scalar *C* to each product. Results are stored in vector *D*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vsmsb

Vector scalar multiply and vector subtract.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vsmsb (float * A,

 int I,

 float * B,

 float * C,

 int K,

 float * D,

 int L,

 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

B
: Single-precision real input scalar

C
: Single-precision real input vector

K
: Stride for *C*

D
: Single-precision real output vector

L
: Stride for *D*

N
: Count

Discussion

Performs the operation

$$D_{nM} = A_{nI} \bullet B - C_{nK} \qquad n = \{0, N-1\}$$

Multiplies vector *A* by scalar *B* and then subtracts vector *C* from the products. Results are stored in vector *D*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vsmsbD

Vector scalar multiply and vector subtract.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vsmsbD (double * A,

 int I,

 double * B,

 double * C,

 int K,

 double * D,

 int L,

 unsigned int N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

B
: Double-precision real input scalar

C
: Double-precision real input vector

K
: Stride for *C*

D
: Double-precision real output vector

L
: Stride for *D*

N
: Count

Discussion

Performs the operation

$$D_{nM} = A_{nI} \bullet B - C_{nK} \qquad n = \{0, N\text{-}1\}$$

Multiplies vector *A* by scalar *B* and then subtracts vector *C* from the products. Results are stored in vector *D*.

Availability
Available in Mac OS X v10.4 and later.

vDSP_vsmul

Multiplies vector `signal1` by scalar signal2 and leaves the result in vector result.

```
void
vDSP_vsmul (const float signal1[],
    SInt32 signal1stride,
    const float * signal2,
    float result[],
    SInt32 resultStride,
    UInt32 size);
```

Discussion
This performs the operation

$$C_{nK} = A_{nI} \cdot B \qquad n = \{0, N-1\}$$

Criteria to invoke vectorized code:

- The vectors `signal1` and `result` must be relatively aligned.
- The value of `size` must be equal to or greater than 8.
- The values of `signal1stride` and `resultStride` must be 1.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_vsmulD

Multiplies vector `signal1` by scalar signal2 and leaves the result in vector result.

```
void
vDSP_vsmulD (const double signal1[],
    SInt32 signal1stride,
    const double * signal2,
    double result[],
    SInt32 resultStride,
    UInt32 size);
```

Discussion
This performs the operation

$$C_{nK} = A_{nI} \cdot B \qquad n = \{0, N-1\}$$

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_vsort

Vector in-place sort.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vsort (float * C,

 unsigned int N,

 int OFLAG);
```

Parameters

C
: Single-precision real input-output vector

N
: Count

OFLAG
: Flag for sort order: 1 for ascending, -1 for descending

Discussion
Performs an in-place sort of vector `C` in the order specified by parameter `OFLAG`.

Availability
Available in Mac OS X v10.4 and later.

vDSP_vsortD

Vector in-place sort.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vsortD (double * C,

 unsigned int N,

 int OFLAG);
```

Parameters

C
: Double-precision real input-output vector

N
: Count

OFLAG
: Flag for sort order: 1 for ascending, -1 for descending

Discussion

Performs an in-place sort of vector *C* in the order specified by parameter *OFLAG*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vsorti

Vector integer in-place sort.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vsorti (float * C,

 int * IC,

 int * List_addr,

 unsigned int N,

 int OFLAG);
```

Parameters

C
: Single-precision real input vector

IC
: Integer output vector. Must be initialized with the indices of vector *C*, from 0 to *N*-1.

List_addr
: Temporary vector. This is currently not used and NULL should be passed.

N
: Count

OFLAG
: Flag for sort order: 1 for ascending, -1 for descending

Discussion

Leaves input vector *C* unchanged and performs an in-place sort of the indices in vector *IC* according to the values in *A*. The sort order is specified by parameter *OFLAG*.

The values in *C* can then be obtained in sorted order, by taking indices in sequence from *IC*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vsortiD

Vector integer in-place sort.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vsortiD (double * C,

 int * IC,

 int * List_addr,

 unsigned int N,

 int OFLAG);
```

Parameters

C
: Double-precision real input vector

IC
: Integer output vector. Must be initialized with the indices of vector *C*, from 0 to *N*-1.

List_addr
: Temporary vector. This is currently not used and NULL should be passed.

N
: Count

OFLAG
: Flag for sort order: 1 for ascending, -1 for descending

Discussion

Leaves input vector *C* unchanged and performs an in-place sort of the indices in vector *IC* according to the values in *A*. The sort order is specified by parameter *OFLAG*.

The values in *C* can then be obtained in sorted order, by taking indices in sequence from *IC*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vspdp

Vector convert single-precision to double-precision.

```
void
vDSP_vspdp (float * A,
    int I,
    double * C,
    int K,
    unsigned int N);
```

Parameters

A

Single-precision real input vector

I

Stride for *A*

C

Double-precision real output vector

K

Stride for *C*

N

Count

Discussion

This performs the operation

$$C_{nk} = A_{ni} \qquad n = \{0, N\text{-}1\}$$

Creates double-precision vector `C` by converting single-precision inputs from vector `A`. This function can only be done out of place.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vsq

Computes the squared values of vector `signal1` and leaves the result in vector result.

```
void
vDSP_vsq (const float A[],
    SInt32 I,
    float C[],
    SInt32 K,
    UInt32 N);
```

Discussion

This performs the operation

$$C_{nK} = A_{nI}^{2} \qquad n = \{0, N\text{-}1\}$$

Criteria to invoke vectorized code:

- The vectors `signal` and `result` must be relatively aligned.

- The value of `size` must be equal to or greater than 8.
- The values of signalStride and `resultStride` must be 1.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_vsqD

Computes the squared values of vector `signal1` and leaves the result in vector result.

```
void
vDSP_vsqD (const double A[],
    SInt32 I,
    double C[],
    SInt32 K,
    UInt32 N);
```

Discussion
This performs the operation

$$C_{nK} = A_{nI}^2 \qquad n = \{0, N-1\}$$

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_vssq

Computes the signed squares of vector `signal1` and leaves the result in vector result.

```
void
vDSP_vssq (const float A[],
    SInt32 I,
    float C[],
    SInt32 K,
    UInt32 N);
```

Discussion
This performs the operation

$$C_{nK} = A_{nI} \cdot \left|A_{nI}\right| \qquad n = \{0, N-1\}$$

Criteria to invoke vectorized code:

- The vectors `signal` and `result` must be relatively aligned.

- The value of `size` must be equal to or greater than 8.
- The values of signalStride and `resultStride` must be 1.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_vssqD

Computes the signed squares of vector `signal1` and leaves the result in vector result.

```
void
vDSP_vssqD (const double A[],
    SInt32 I,
    double C[],
    SInt32 K,
    UInt32 N);
```

Discussion
This performs the operation

$$C_{nK} = A_{nI} \bullet |A_{nI}| \qquad n = \{0, N\text{-}1\}$$

Criteria to invoke vectorized code:

No altivec support for double precision. This function always invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_vsub

Subtracts vector `signal2` from vector `signal1` and leaves the result in vector `result`.

```
void
vDSP_vsub (const float A[],
    SInt32 I,
    const float B[],
    SInt32 J,
    float C[],
    SInt32 K,
    UInt32 N);
```

Discussion
This performs the operation

$$C_{nK} = A_{nI} - B_{nJ} \qquad n = \{0, N\text{-}1\}$$

Criteria to invoke vectorized code:

- The vectors `signal1`,`signal2`, and `result` must be relatively aligned.
- The value of `size` must be equal to or greater than 8.
- The values of `signal1stride`, `signal2stride`, and `resultStride` must be 1.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_vsubD

Subtracts vector `signal2` from vector `signal1` and leaves the result in vector `result`.

```
void
vDSP_vsub (const float A[],
    SInt32 I,
    const float B[],
    SInt32 J,
    float C[],
    SInt32 K,
    UInt32 N);
```

Discussion
This performs the operation

$$C_{nK} = A_{nI} - B_{nJ} \qquad n = \{0, N\text{-}1\}$$

Criteria to invoke vectorized code:

- The vectors `signal1`,`signal2`, and `result` must be relatively aligned.
- The value of `size` must be equal to or greater than 8.
- The values of `signal1stride`, `signal2stride`, and `resultStride` must be 1.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_vswap

Vector swap.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vswap (float * A,

 int I,
 float * C,

    int K
 unsigned int N);
```

Parameters

A
: Single-precision real input-output vector

I
: Stride for *A*

C
: Single-precision real input-output vector

K
: Stride for *C*

N
: Count

Discussion

Performs the operation

$$C_{nK} \Leftrightarrow A_{nI} \qquad n = \{0, N\text{-}1\}$$

Exchanges the elements of vectors *A* and *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vswapD

Vector swap.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vswapD (double * A,

 int I,

 double * C,

    int K
 unsigned int N);
```

Parameters

A
: Double-precision real input-output vector

I
: Stride for *A*

C

Double-precision real input-output vector

J

Stride for *C*

N

Count

Discussion

Performs the operation

$C_{nK} \Leftrightarrow A_{nI} \qquad n = \{0, N\text{-}1\}$

Exchanges the elements of vectors *A* and *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vswsum

Vector sliding window sum.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vswsum (float * A,

 int I,

 float * C,

 int K,

 unsigned int N,

 unsigned int P);
```

Parameters

A

Single-precision real input vector

I

Stride for *A*

C

Single-precision real output vector

K

Stride for *C*

N

Count of output points

P

Length of window

Discussion

Performs the operation

$$C_0(P) = \sum_{p=0}^{P-1} A_{qi} \quad (C_{nk}(P) = C_{(n-1)k}(P) + A_{(n+P-1)i} - A_{(n-1)i}) \qquad \text{n = \{1, N-1\}}$$

Writes the sliding window sum of *P* consecutive elements of vector *A* to vector *C*, for each of *N* possible starting positions of the *P*-element window in vector *A*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vswsumD

Vector sliding window sum.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vswsumD (double * A,

 int I,

 double * C,

 int K,

 unsigned int N,

 unsigned int P);
```

Parameters

A

Double-precision real input vector

I

Stride for *A*

C

Double-precision real output vector

K

Stride for *C*

N

Count of output points

P

Length of window

Discussion

Performs the operation

$$C_0(P) = \sum_{p=0}^{P-1} A_{qi} \quad (C_{nk}(P) = C_{(n-1)k}(P) + A_{(n+P-1)i} - A_{(n-1)i}) \qquad \text{n = \{1, N-1\}}$$

Writes the sliding window sum of *P* consecutive elements of vector *A* to vector *C*, for each of *N* possible starting positions of the *P*-element window in vector *A*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vtabi

Vector interpolation, table lookup.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vtabi (float * A,

 int I,

 float * F,

 float * G,

 float * C,

 unsigned int M,

 float * D,

 int L,

 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

F
: Single-precision real input scalar: scale factor

G
: Single-precision real input scalar: base offset

C
: Single-precision real input vector: lookup table

M
Lookup table size

D
Single-precision real output vector

L
Stride for *D*

N
Count

Discussion

Performs the operation

$$p = F \bullet A_{nI} + G \qquad q = \text{floor}(p) \qquad r = p - \text{float}(q) \qquad C_{nk} = (1.0 - r)B_q + rB_{q-1} \qquad \text{n} = \{0, \text{N-1}\}$$

Evaluates elements of vector A for use as offsets into vector *B*. Vector *B* is a zero-based lookup table supplied by the caller that generates output values for vector *C*. Linear interpolation is used to compute output values when offsets do not evaluate integrally. Scale factor *F* and base offset *G* map the anticipated range of input values to the range of the lookup table and are typically assigned values such that:

```
    floor(F * minimum input value + G) = 0
    floor(F * maximum input value + G) = M-1
```

Input values that evaluate to zero or less derive their output values from table location zero. Values that evaluate beyond the table, greater than *M*-1, derive their output values from the last table location. For inputs that evaluate integrally, the table location indexed by the integral is copied as the output value. All other inputs derive their output values by interpolation between the two table values surrounding the evaluated input.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vtabiD

Vector interpolation, table lookup.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vtabiD (double * A,

 int I,

 double * F,

 double * G,

 double * C,

 unsigned int M,
 double * D,

 int L,

 unsigned int N);
```

Parameters

A
Double-precision real input vector

I
Stride for *A*

F
Double-precision real input scalar: scale factor

G
Double-precision real input scalar: base offset

C
Double-precision real input vector: lookup table

M
Lookup table size

D
Double-precision real output vector

L
Stride for *D*

N
Count

Discussion
Performs the operation

$$p = F \bullet A_{nI} + G \qquad q = \text{floor}(p) \qquad r = p - \text{float}(q) \qquad C_{nk} = (1.0 - r)B_q + rB_{q-1} \qquad \text{n} = \{0, \text{N-1}\}$$

Evaluates elements of vector A for use as offsets into vector *B*. Vector *B* is a zero-based lookup table supplied by the caller that generates output values for vector *C*. Linear interpolation is used to compute output values when offsets do not evaluate integrally. Scale factor *F* and base offset *G* map the anticipated range of input values to the range of the lookup table and are typically assigned values such that:

```
    floor(F * minimum input value + G) = 0
    floor(F * maximum input value + G) = M-1
```

Input values that evaluate to zero or less derive their output values from table location zero. Values that evaluate beyond the table, greater than *M*-1, derive their output values from the last table location. For inputs that evaluate integrally, the table location indexed by the integral is copied as the output value. All other inputs derive their output values by interpolation between the two table values surrounding the evaluated input.

Availability
Available in Mac OS X v10.4 and later.

vDSP_vthr

Vector threshold.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vthr (float * A,

 int I,

 float * B,

 float * C,

 int K,

 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

B
: Single-precision real input scalar: lower threshold

C
: Single-precision real output vector

K
: Stride for *C*

N
: Count

Discussion
Performs the operation

If $A_{nI} \geq B$ then $C_{nK} = A_{nI}$ else $C_{nK} = B$ n = {0, N-1}

Creates vector *C* by comparing each input from vector *A* with scalar *B*. If an input value is less than *B*, *B* is copied to *C*; otherwise, the input value from *A* is copied to *C*.

Availability
Available in Mac OS X v10.4 and later.

vDSP_vthrD

Vector threshold.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vthrD (double * A,

 int I,

 double * B,

 double * C,

 int K,

 unsigned int N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

B
: Double-precision real input scalar: lower threshold

C
: Double-precision real output vector

K
: Stride for *C*

N
: Count

Discussion

Performs the operation

If $A_{nI} \geq B$ then $C_{nK} = A_{nI}$ else $C_{nK} = B$ n = {0, N-1}

Creates vector *C* by comparing each input from vector *A* with scalar *B*. If an input value is less than *B*, *B* is copied to *C*; otherwise, the input value from *A* is copied to *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vthres

Vector threshold with zero fill.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vthres (float * A,

 int I,

 float * B,

 float * C,

 int K,

 unsigned int N);
```

Parameters

A

Single-precision real input vector

I

Stride for *A*

B

Single-precision real input scalar: lower threshold

C

Single-precision real output vector

K

Stride for *C*

N

Count

Discussion

Performs the operation

If $A_{nI} \geq B$ then $C_{nK} = A_{nI}$ else $C_{nK} = 0.0$ $n = \{0, N\text{-}1\}$

Creates vector *C* by comparing each input from vector *A* with scalar *B*. If an input value is less than *B*, zero is written to *C*; otherwise, the input value from *A* is copied to *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vthresD

Vector threshold with zero fill.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vthresD (double * A,

 int I,
 double * B,

 double * C,

 int K,

 unsigned int N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

B
: Double-precision real input scalar: lower threshold

C
: Double-precision real output vector

K
: Stride for *C*

N
: Count

Discussion

Performs the operation

If $A_{nI} \geq B$ then $C_{nK} = A_{nI}$ else $C_{nK} = 0.0$ $n = \{0, N-1\}$

Creates vector *C* by comparing each input from vector *A* with scalar *B*. If an input value is less than *B*, zero is written to *C*; otherwise, the input value from *A* is copied to *C*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vthrsc

Vector threshold with signed constant.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vthrsc (float * A,

 int I,

 float * B,

 float * C,
 float * D,

 int L,

 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

B
: Single-precision real input scalar: lower threshold

C
: Single-precision real input scalar

D
: Single-precision real output vector

L
: Stride for *D*

N
: Count

Discussion

Performs the operation

If $A_{nI} \geq B$ then $D_{nM} = C$ else $D_{nM} = -C$ n = {0, N-1}

Creates vector *D* using the plus or minus value of scalar *C*. The sign of the output element is determined by comparing input from vector *A* with threshold scalar *B*. For input values less than *B*, the negated value of *C* is written to vector *D*. For input values greater than or equal to *B*, *C* is copied to vector *D*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vthrscD

Vector threshold with signed constant.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vthrscD (double * A,

 int I,

 double * B,
 double * C,

 double * D,

 int L,

 unsigned int N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

B
: Double-precision real input scalar: lower threshold

C
: Double-precision real input scalar

D
: Double-precision real output vector

L
: Stride for *D*

N
: Count

Discussion

Performs the operation

$$\text{If} \quad A_{nI} \geq B \quad \text{then} \quad D_{nM} = C \quad \text{else} \quad D_{nM} = -C \qquad n = \{0, N-1\}$$

Creates vector *D* using the plus or minus value of scalar *C*. The sign of the output element is determined by comparing input from vector *A* with threshold scalar *B*. For input values less than *B*, the negotiated value of *C* is written to vector *D*. For input values greater than or equal to *B*, *C* is copied to vector *D*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vtmerg

Vector tapered merge of two vectors.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vtmerg (float * A,

 int I,
 float * B,
 int J,
 float * C,
 int K,
 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

B
: Single-precision real input vector

J
: Stride for *B*

C
: Single-precision real output vector

K
: Stride for *C*

N
: Count

Discussion

Performs the operation

$$C_{nK} = A_{nI} + \frac{n(B_{nJ} - A_{nI})}{N-1} \qquad n = \{0, N\text{-}1\}$$

Performs a tapered merge of vectors *A* and *B*. Values written to vector *C* range from element zero of vector *A* to element *N*–1 of vector *B*. Output values between these endpoints reflect varying amounts of their corresponding inputs from vectors *A* and *B*, with the percentage of vector *A* decreasing and the percentage of vector *B* increasing as the index increases.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vtmergD

Vector tapered merge of two vectors.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vtmergD (double * A,

 int I,

 double * B,

 int J,

 double * C,

 int K,

 unsigned int N);
```

Parameters

A
: Double-precision real input vector

I
: Stride for *A*

B
: Double-precision real input vector

J
: Stride for *B*

C
: Double-precision real output vector

K
: Stride for *C*

N
: Count

Discussion

Performs the operation

$$C_{nK} = A_{nI} + \frac{n(B_{nJ} - A_{nI})}{N-1} \qquad \mathrm{n} = \{0, \mathrm{N}\text{-}1\}$$

Performs a tapered merge of vectors *A* and *B*. Values written to vector *C* range from element zero of vector *A* to element *N*–1 of vector *B*. Output values between these endpoints reflect varying amounts of their corresponding inputs from vectors *A* and *B*, with the percentage of vector *A* decreasing and the percentage of vector *B* increasing as the index increases.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vtrapz

Vector trapezoidal integration.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vtrapz (float * A,

 int I,

 float * B,

 float * C,

 int K,

 unsigned int N);
```

Parameters

A
: Single-precision real input vector

I
: Stride for *A*

B
: Single-precision real input scalar: step size

C
: Single-precision real output vector

K
: Stride for *C*

N
: Count

Discussion

Performs the operation

$$C_0 = 0.0$$

$$C_{nK} = C_{[n-1]K} + \frac{B[A_{[n-1]I} + A_{nI}]}{2} \qquad n = \{1, N\text{-}1\}$$

Estimates the integral of vector *A* using the trapezoidal rule. Scalar *B* specifies the integration step size. This function can only be done out of place.

Availability

Available in Mac OS X v10.4 and later.

vDSP_vtrapzD

Vector trapezoidal integration.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_vtrapzD (double * A,

 int I,

 double * B,

 double * C,

 int K,

 unsigned int N);
```

Parameters

A

Double-precision real input vector

I

Stride for *A*

B

Double-precision real input scalar: step size

C

Double-precision real output vector

K

Stride for *C*

N

Count

Discussion

Performs the operation

$$C_0 = 0.0$$

$$C_{nK} = C_{[n-1]K} + \frac{B[A_{[n-1]I} + A_{nI}]}{2} \qquad n = \{1, N\text{-}1\}$$

Estimates the integral of vector *A* using the trapezoidal rule. Scalar *B* specifies the integration step size. This function can only be done out of place.

Availability

Available in Mac OS X v10.4 and later.

vDSP_wiener

Wiener-Levinson general convolution.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_wiener (int L,

 float * A,

 float * C,

 float * F,

 float * P,

 int IFLG,

 int * IERR);
```

Parameters

L
Input filter length

A
Single-precision real input vector: coefficients

C
Single-precision real input vector: input coefficients

F
Single-precision real output vector: filter coefficients

P
Single-precision real output vector: error prediction operators

IFLG
Not currently used, pass zero

IERR
Error flag

Discussion

Performs the operation

$$\text{Find} \quad C_m \quad \text{such that} \quad B_n = \sum_{m=0}^{N-1} A_{n-m} \cdot C_m \qquad m,n = \{0, N-1\}$$

solves a set of single-channel normal equations described by:

```
    B[n] = C[0] * A[n] + C[1] * A[n-1] +, . . . ,+ C[N-1] *  A[n-N+1]
        for n = {0, N-1}
```

where matrix *A* contains elements of the symmetric Toeplitz matrix shown below. This function can only be done out of place.

Note that *A*[-n] is considered to be equal to *A*[n].

`vDSP_wiener` solves this set of simultaneous equations using a recursive method described by Levinson. See Robinson, E.A., *Multichannel Time Series Analysis with Digital Computer Programs*. San Francisco: Holden-Day, 1967, pp. 43-46.

```
    |A[0]  A[1]  A[2] ... A[N-1] |    |C[0]  |   |B[0]  |
    |A[1]  A[0]  A[1] ... A[N-2] |    |C[1]  |   |B[1]  |
    |A[2]  A[1]  A[0] ... A[N-3] |  * |C[2]  | = |B[2]  |
    | ...   ...   ... ... ...    |    | ...  |   | ...  |
    |A[N-1]A[N-2]A[N-3] ... A[0] |    |C[N-1]|   |B[N-1]|
```

Typical methods for solving N equations in N unknowns have execution times proportional to N cubed, and memory requirements proportional to N squared. By taking advantage of duplicate elements, the recursion method executes in a time proportional to N squared and requires memory proportional to N. The Wiener-Levinson algorithm recursively builds a solution by computing the m+1 matrix solution from the m matrix solution.

With successful completion, `vDSP_wiener` returns zero in error flag *IERR*. If `vDSP_wiener` fails, *IERR* indicates in which pass the failure occurred.

Availability
Available in Mac OS X v10.4 and later.

vDSP_wienerD

Wiener-Levinson general convolution.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_wienerD (int L,

 double * A,

 double * C,

 double * F,

 double * P,

 int IFLG,

 int * IERR);
```

Parameters

L
: Input filter length

A
: Double-precision real input vector: coefficients

C
: Double-precision real input vector: input coefficients

F
: Double-precision real output vector: filter coefficients

P
: Double-precision real output vector: error prediction operators

IFLG
: Not currently used, pass zero

IERR
: Error flag

Discussion

Performs the operation

$$\text{Find} \quad C_m \quad \text{such that} \quad B_n = \sum_{m=0}^{N-1} A_{n-m} \cdot C_m \qquad m,n = \{0, N-1\}$$

solves a set of single-channel normal equations described by:

```
    B[n] = C[0] * A[n] + C[1] * A[n-1] +, . . . ,+ C[N-1] *  A[n-N+1]
        for n = {0, N-1}
```

where matrix *A* contains elements of the symmetric Toeplitz matrix shown below. This function can only be done out of place.

Note that *A*[-n] is considered to be equal to *A*[n].

`vDSP_wiener` solves this set of simultaneous equations using a recursive method described by Levinson. See Robinson, E.A., *Multichannel Time Series Analysis with Digital Computer Programs*. San Francisco: Holden-Day, 1967, pp. 43-46.

```
    |A[0]  A[1]  A[2] ... A[N-1] |     |C[0]   |   |B[0]   |
    |A[1]  A[0]  A[1] ... A[N-2] |     |C[1]   |   |B[1]   |
    |A[2]  A[1]  A[0] ... A[N-3] |  *  |C[2]   | = |B[2]   |
    | ...   ...   ... ... ...    |     | ...   |   | ...   |
    |A[N-1]A[N-2]A[N-3] ... A[0] |     |C[N-1]|   |B[N-1]|
```

Typical methods for solving N equations in N unknowns have execution times proportional to N cubed, and memory requirements proportional to N squared. By taking advantage of duplicate elements, the recursion method executes in a time proportional to N squared and requires memory proportional to N. The Wiener-Levinson algorithm recursively builds a solution by computing the m+1 matrix solution from the m matrix solution.

With successful completion, `vDSP_wiener` returns zero in error flag *IERR*. If `vDSP_wiener` fails, *IERR* indicates in which pass the failure occurred.

Availability

Available in Mac OS X v10.4 and later.

vDSP_zaspec

Computes an accumulating autospectrum.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zaspec (DSPSplitComplex * A,
    float* C,
    unsigned int N);
```

Discussion

`vDSP_zaspec` multiplies single-precision complex vector *A* by its complex conjugates, yielding the sums of the squares of the complex and real parts: (x + iy) (x – iy) = (x*x + y*y). The results are added to real single-precision input-output vector *C*. Vector *C* must contain valid data from previous processing or should be initialized according to your needs before calling `vDSP_zaspec`.

$$C_n = C_n + (Re(A_n))^2 + (Im(A_n))^2 \qquad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

vDSP_zaspecD

Computes an accumulating autospectrum.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zaspecD (DSPDoubleSplitComplex * A,
    double* C,
    unsigned int N);
```

Discussion

`vDSP_zaspecD` multiplies double-precision complex vector *A* by its complex conjugates, yielding the sums of the squares of the complex and real parts: (x + iy) (x – iy) = (x*x + y*y). The results are added to real double-precision input-output vector *C*. Vector *C* must contain valid data from previous processing or should be initialized according to your needs before calling `vDSP_zaspec`.

$$C_n = C_n + (Re(A_n))^2 + (Im(A_n))^2 \qquad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

vDSP_zcoher

Coherence function of two signals.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zcoher (float * A,
    float * B,
    DSPSplitComplex * C,
    float * D,
    unsigned int N);
```

Discussion

Computes the single-precision coherence function `D` of two signals. The inputs are the signals' autospectra, real single-precision vectors `A` and `B`, and their cross-spectrum, single-precision complex vector `C`.

$$D_n = \frac{[Re(C_n)]^2 + [Im(C_n)]^2}{A_n B_n} \qquad n = \{0, N\text{-}1\}$$

Availability

Available in Mac OS X v10.4 and later.

vDSP_zcoherD

Coherence function of two signals.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zcoherD (double * A,
    double * B,
    DSPDoubleSplitComplex * C,
    double * D,
    unsigned int N);
```

Discussion

Computes the double-precision coherence function `D` of two signals. The inputs are the signals' autospectra, real double-precision vectors `A` and `B`, and their cross-spectrum, double-precision complex vector `C`.

$$D_n = \frac{[Re(C_n)]^2 + [Im(C_n)]^2}{A_n B_n} \qquad n = \{0, N\text{-}1\}$$

Availability

Available in Mac OS X v10.4 and later.

vDSP_zconv

Performs either correlation or convolution on two complex vectors.

```
void
vDSP_zconv (DSPSplitComplex * A,
    SInt32 I,
    DSPSplitComplex * B,
    SInt32 J,
    DSPSplitComplex * C,
    SInt32 K,
    SInt32 N,
    SInt32 P);
```

Discussion

`A` is the input vector, with stride `I`, and `C` is the output vector, with stride `K` and length `N`.

`B` is a filter vector, with stride `I` and length `P`. If `J` is positive, the function performs correlation. If `J` is negative, it performs convolution and `B` must point to the last element in the filter vector. The function can run in place, but `C` cannot be in place with `B`.

$$C_{nK} = \sum_{p=0}^{P-1} A_{(n+p)I} B_{pJ} \qquad n = \{0, N-1\}$$

The value of `N` must be less than or equal to 512.

Criteria to invoke vectorized code:

- Both the real parts and the imaginary parts of vectors `A` and `C` must be relatively aligned.
- The values of `I` and `K` must be 1.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability

Available in Mac OS X v10.4 and later.

vDSP_zconvD

Performs either correlation or convolution on two complex vectors.

```
void
vDSP_zconvD (DSPDoubleSplitComplex * A,
    SInt32 I,
    DSPDoubleSplitComplex * B,
    SInt32 J,
    DSPDoubleSplitComplex * C,
    SInt32 K,
    SInt32 N,
    SInt32 P);
```

Discussion

`A` is the input vector, with stride `I`, and `C` is the output vector, with stride `K` and length `N`.

`B` is a filter vector, with stride `I` and length `P`. If `J` is positive, the function performs correlation. If `J` is negative, it performs convolution and `B` must point to the last element in the filter vector. The function can run in place, but `C` cannot be in place with `B`.

$$C_{nK} = \sum_{p=0}^{P-1} A_{(n+p)I} B_{pJ} \qquad n = \{0, N-1\}$$

The value of N must be less than or equal to 512.

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_zcspec

Accumulating cross-spectrum on two complex vectors.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zcspec (DSPSplitComplex * A,
    DSPSplitComplex * B,
    DSPSplitComplex * C,
    unsigned int N);
```

Parameters

A
: Single-precision complex input vector

B
: Single-precision complex input vector

C
: Single-precision complex input-output vector

N
: Count

Discussion
Computes the cross-spectrum of complex vectors *A* and *B* and then adds the results to complex input-output vector *C*. Vector *C* should contain valid data from previous processing or should be initialized with zeros before calling `vDSP_zcspec`.

$$C_n = C_n + A_n^* B_n \qquad n = \{0, N-1\}$$

Availability
Available in Mac OS X v10.4 and later.

vDSP_zcspecD

Accumulating cross-spectrum on two complex vectors.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zcspecD (DSPDoubleSplitComplex * A,
    DSPDoubleSplitComplex * B,
    DSPDoubleSplitComplex * C,
    unsigned int N);
```

Parameters

A
: Double-precision complex input vector

B
: Double-precision complex input vector

C
: Double-precision complex input-output vector

N
: Count

Discussion

Computes the cross-spectrum of complex vectors `A` and `B` and then adds the results to complex input-output vector `C`. Vector `C` should contain valid data from previous processing or should be initialized with zeros before calling `vDSP_zcspecD`.

$$C_n = C_n + A_n^* B_n \qquad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

vDSP_zdotpr

Calculates the complex dot product of complex vectors signal1 and signal2 and leaves the result in complex vector `result`.

```
void
vDSP_zdotpr (DSPSplitComplex * A,
    SInt32 I,
    DSPSplitComplex * B,
    SInt32 J,
    DSPSplitComplex * C,
    UInt32 N);
```

Discussion

This performs the operation

$$C = \sum_{n=0}^{N-1} A_{nI} B_{nJ}$$

Criteria to invoke vectorized code:

- The vectors `A->realp`, `A->imagp`, `B->realp`, and `B->imagp` must be relatively aligned.
- The value of `N` must be equal to or greater than 20.
- The values of `I`, and `J` must be 1.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_zdotprD

Calculates the complex dot product of complex vectors signal1 and signal2 and leaves the result in complex vector `result`.

```
void
vDSP_zdotprD (DSPDoubleSplitComplex * A,
    SInt32 I,
    DSPDoubleSplitComplex * B,
    SInt32 J,
    DSPDoubleSplitComplex * C,
    UInt32 N);
```

Discussion
This performs the operation

$$C = \sum_{n=0}^{N-1} A_{nI} B_{nJ}$$

Criteria to invoke vectorized code:

No altivec support for double precision. This function always invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_zidotpr

Calculates the conjugate dot product (or inner dot product) of complex vectors signal1 and signal2 and leave the result in complex vector `result`.

```
void
vDSP_zidotpr (DSPSplitComplex * A,
    SInt32 I,
    DSPSplitComplex * B,
    SInt32 J,
    DSPSplitComplex * C,
    UInt32 N);
```

Discussion
This performs the operation

$$C = \sum_{n=0}^{N-1} A^*_{nI} B_{nJ}$$

Criteria to invoke vectorized code:

- The vectors `A->realp`, `A->imagp`, `B->realp`, and `B->imagp` must be relatively aligned.
- The value of `N` must be equal to or greater than 20.
- The values of `I`, and `J` must be 1.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_zidotprD

Calculates the conjugate dot product (or inner dot product) of complex vectors signal1 and signal2 and leave the result in complex vector `result`.

```
void
vDSP_zidotprD (DSPDoubleSplitComplex * A,
    SInt32 I,
    DSPDoubleSplitComplex * B,
    SInt32 J,
    DSPDoubleSplitComplex * C,
    UInt32 N);
```

Discussion
This performs the operation

$$C = \sum_{n=0}^{N-1} A^*_{nI} B_{nJ}$$

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_zmma

Multiplies a matrix `A` by matrix `B` , adds the product to matrix `C`, and stores the result in matrix `D`. `A` is an `M`-by-`P` matrix, `B` is a `P`-by-`N` matrix, `C` and `D` are by `M`-by-`N` matrixes. This function can only be performed out-of-place.

```
void
vDSP_zmma( DSPSplitComplex * A,

 SInt32 I,

 DSPSplitComplex * B,

 SInt32 J,

 DSPSplitComplex * C,

 SInt32 K,

 DSPSplitComplex * D,

 SInt32 L,

 SInt32 M,

 SInt32 N,

 SInt32 P);
```

Discussion
This performs the operation

$$D_{(rN+q)L} = C_{(rN+q)K} + \sum_{p=0}^{P-1} A_{(rP+p)I} B_{(pN+q)J}$$

$$0 \le r < M, \quad 0 \le q < N$$

Parameters `A` and `C` are the matrixes to be multiplied, and `C` the matrix to be added. `I` is an address stride through `A`. `J` is an address stride through `B`. `K` is an address stride through `C`. `L` is an address stride through `D`.

Parameter `D` is the result matrix.

Parameter `M` is the row count for `A`, `C` and `D`. Parameter `N` is the column count of `B`, `C`, and `D`. Parameter `P` is the column count of `A` and the row count of `B`.

Criteria to invoke vectorized code:

- `A`, `B`, `C`, and `D` must be 16-byte aligned.
- `I`, `J`, `K`, and `L` must all be 1.
- `N` and `P` must be multiples of 4 and greater than or equal to 16.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_zmmaD

Multiplies a matrix A by matrix B , adds the product to matrix C, and stores the result in matrix D. A is an M-by-P matrix, B is a P-by-N matrix, C and D are by M-by-N matrixes. This function can only be performed out-of-place.

```
void
vDSP_zmmaD (DSPDoubleSplitComplex * A,

 SInt32 I,

 DSPDoubleSplitComplex * B,

 SInt32 J,

 DSPDoubleSplitComplex * C,

 SInt32 K,

 DSPDoubleSplitComplex * D,

 SInt32 L,

 SInt32 M,

 SInt32 N,

 SInt32 P);
```

Discussion

This performs the operation

$$D_{(rN+q)L} = C_{(rN+q)K} + \sum_{p=0}^{P-1} A_{(rP+p)I} B_{(pN+q)J}$$

$$0 \le r < M, \quad 0 \le q < N$$

Parameters A and C are the matrixes to be multiplied, and *C* the matrix to be added. I is an address stride through A. J is an address stride through B. K is an address stride through C. L is an address stride through D.

Parameter D is the result matrix.

Parameter M is the row count for A, C and D. Parameter N is the column count of B, C, and D. Parameter P is the column count of A and the row count of B.

Criteria to invoke vectorized code:

No altivec support for double precision. This function always invokes scalar code.

Availability

Available in Mac OS X v10.4 and later.

vDSP_zmms

Multiplies an matrix `a` by matrix `b` , subtracts matrix c from the product, and stores the result in matrix d. a is an M-by-P matrix, b is a P-by-N matrix, c and d are by M-by-N matrixes. The function can only be performed out of place.

```
void
vDSP_zmms (DSPSplitComplex * A,

 SInt32 I,

 DSPSplitComplex * B,

 SInt32 bStriJde,

 DSPSplitComplex * C,

 SInt32 K,

 DSPSplitComplex * D,

 SInt32 L,

 SInt32 M,

 SInt32 N,

 SInt32 P);
```

Discussion

This performs the operation

$$D_{(rN+q)L} = \sum_{p=0}^{P-1} A_{(rP+p)I} B_{(pN+q)J} - C_{(rN+q)K}$$

$$0 \le r < M, \quad 0 \le q < N$$

Parameters `A` and `B` are the matrixes to be multiplied, and C the matrix to be subtracted. `I` is an address stride through `A`. `J` is an address stride through `B`. `K` is an address stride through `C`. `L` is an address stride through `D`.

Parameter `D` is the result matrix.

Parameter `M` is the row count for `A`, `C` and `D`. Parameter `N` is the column count of `B`, `C`, and `D`. Parameter `P` is the column count of `A` and the row count of `B`.

Criteria to invoke vectorized code:

- `A`, `B`, `C`, and `D` must be 16-byte aligned.
- `I`, `J`, `K`, and `L` must all be 1.
- `N` and `P` must be multiples of 4 and greater than or equal to 16.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_zmmsD

Multiplies an matrix `a` by matrix `b` , subtracts matrix c from the product, and stores the result in matrix d. a is an M-by-P matrix, b is a P-by-N matrix, c and d are by M-by-N matrixes. The function can only be performed out of place.

```
vDSP_zmmsD (DSPDoubleSplitComplex * A,

 SInt32 I,

 DSPDoubleSplitComplex * B,

 SInt32 J,

 DSPDoubleSplitComplex * C,

 SInt32 K,

 DSPDoubleSplitComplex * D,

 SInt32 L,

 SInt32 M,

 SInt32 N,

 SInt32 P);
```

Discussion
This performs the operation

$$D_{(rN+q)L} = \sum_{p=0}^{P-1} A_{(rP+p)I} B_{(pN+q)J} - C_{(rN+q)K}$$

$$0 \le r < M, \quad 0 \le q < N$$

Parameters `A` and `B` are the matrixes to be multiplied, and C the matrix to be subtracted. `I` is an address stride through `A`. `J` is an address stride through `B`. `K` is an address stride through `C`. `L` is an address stride through `D`.

Parameter `D` is the result matrix.

Parameter `M` is the row count for `A`, `C` and `D`. Parameter `N` is the column count of `B`, `C`, and `D`. Parameter `P` is the column count of `A` and the row count of `B`.

Criteria to invoke vectorized code:

No altivec support for double precision. This function always invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_zmmul

Multiplies an `M`-by-`P` matrix `A` by a `P`-by-`N` matrix `B` and stores the results in an `M`-by-`N` matrix `C`. The function can only be performed out of place

```
void
vDSP_zmmul (DSPSplitComplex * A,

 SInt32 I,

 DSPSplitComplex * B,

 SInt32 J,

 DSPSplitComplex * C,

 SInt32 K,

 SInt32 M,

 SInt32 N,

 SInt32 P);
```

Discussion
This performs the operation

$$C_{(mN+n)K} = \sum_{p=0}^{P-1} A_{(mP+p)I} \cdot B_{(pN+n)J} \qquad n = \{0, N\text{-}1\} \text{ and } m = \{0, M\text{-}1\}$$

Parameters `A` and `B` are the matrixes to be multiplied. `I` is an address stride through `A`. `J` is an address stride through `B`.

Parameter `C` is the result matrix. `K` is an address stride through `C`.

Parameter `M` is the row count for both `A` and `C`. Parameter `N` is the column count for both `B` and `C`. Parameter `P` is the column count for `A` and the row count for `B`.

Criteria to invoke vectorized code:

- `A`, `B`, and `C` must be 16-byte aligned.
- `I`, `J`, and `K` must all be 1.
- `N` and `P` must be multiples of 4 and greater than or equal to 16.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_zmmulD

Multiplies an `M`-by-`P` matrix `A` by a `P`-by-`N` matrix `B` and stores the results in an `M`-by-`N` matrix `C`. The function can only be performed out of place.

```
void
vDSP_zmmulD (DSPDoubleSplitComplex * a,

 SInt32 aStride,

 DSPDoubleSplitComplex * b,

 SInt32 bStride,

 DSPDoubleSplitComplex * c,

 SInt32 cStride,

 SInt32 M,

 SInt32 N,

 SInt32 P);
```

Discussion
This performs the operation

$$C_{(mN+n)K} = \sum_{p=0}^{P-1} A_{(mP+p)I} \bullet B_{(pN+n)J} \qquad n = \{0, N\text{-}1\} \text{ and } m = \{0, M\text{-}1\}$$

Parameters `A` and `B` are the matrixes to be multiplied. `I` is an address stride through `A`. `J` is an address stride through `B`.

Parameter `C` is the result matrix. `K` is an address stride through `C`.

Parameter `M` is the row count for both `A` and `C`. Parameter `N` is the column count for both `B` and `C`. Parameter `P` is the column count for `A` and the row count for `B`.

Criteria to invoke vectorized code:

No altivec support for double precision. This function always invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_zmsm

Multiplies an matrix `a` by matrix `b` , subtracts the product from matrix c, and stores the result in matrix d. a is an M-by-P matrix, b is a P-by-N matrix, c and d are by M-by-N matrixes. The function can only be performed out of place.

```
void
vDSP_zmsm (DSPSplitComplex * a,

 SInt32 aStride,

 DSPSplitComplex * b,

 SInt32 bStride,

 DSPSplitComplex * c,

 SInt32 cStride,

 DSPSplitComplex * d,

 SInt32 dStride,

 SInt32 M,

 SInt32 N,

 SInt32 P);
```

Discussion

This performs the operation

$$D_{(rN+q)L} = C_{(rN+q)K} - \sum_{p=0}^{P-1} A_{(rP+p)I} B_{(pN+q)J}$$

Parameters `a` and `b` are the matrixes to be multiplied, and c is the matrix from which the product is to be subtracted. `aStride` is an address stride through `a`. `bStride` is an address stride through `b`. `cStride` is an address stride through `c`. `dStride` is an address stride through `d`.

Parameter `d` is the result matrix.

Parameter `M` is the row count for `a`, `c` and `d`. Parameter `N` is the column count of `b`, `c`, and `d`. Parameter `P` is the column count of `a` and the row count of `b`.

Criteria to invoke vectorized code:

- `a`, `b`, `c`, and `d` must be 16-byte aligned.
- `aStride`, `bStride`, `cStride`, and `dStride` must all be 1.
- `N` and `P` must be multiples of 4 and greater than or equal to 16.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_zmsmD

Multiplies an matrix `a` by matrix `b` , subtracts the product from matrix c, and stores the result in matrix d. a is an M-by-P matrix, b is a P-by-N matrix, c and d are by M-by-N matrixes. The function can only be performed out of place.

```
vDSP_zmsmD (DSPDoubleSplitComplex * a,

 SInt32 aStride,

 DSPDoubleSplitComplex * b,

 SInt32 bStride,

 DSPDoubleSplitComplex * c,

 SInt32 cStride,

 DSPDoubleSplitComplex * d,

 SInt32 dStride,

 SInt32 M,

 SInt32 N,

 SInt32 P);
```

Discussion
This performs the operation

$$D_{(rN+q)L} \quad = \quad C_{(rN+q)K} \; - \sum_{p=0}^{P-1} A_{(rP+p)I} \, B_{(pN+q)J}$$

Parameters `a` and `b` are the matrixes to be multiplied, and c is the matrix from which the product is to be subtracted. `aStride` is an address stride through `a`. `bStride` is an address stride through `b`. `cStride` is an address stride through `c`. `dStride` is an address stride through `d`.

Parameter `d` is the result matrix.

Parameter `M` is the row count for `a`, `c` and `d`. Parameter `N` is the column count of `b`, `c`, and `d`. Parameter `P` is the column count of `a` and the row count of `b`.

Criteria to invoke vectorized code:

No altivec support for double precision. This function always invokes scalar code.

Availability

Available in Mac OS X v10.4 and later.

vDSP_zrdesamp

Complex/real downsample with anti-aliasing.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zrdesamp (DSPSplitComplex * A,

 int I,

 float * B,

 DSPSplitComplex * C,

 int N,

 int M);
```

Parameters

A

Single-precision complex input vector.

I

Complex decimation factor.

B

Filter coefficient vector.

C

Single-precision complex output vector.

N

Length of output vector.

M

Length of real filter vector.

Discussion

Performs finite impulse response (FIR) filtering at selected positions of input vector *A*.

$$C_m = \sum_{p=0}^{P-1} A(mi+q) \cdot Bq, \qquad (\mathrm{m} = \{0, \mathrm{N-1}\})$$

Length of *A* must be at least (N+M-1)*i. This function can run in place, but *C* cannot be in place with *B*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_zrdesampD

Complex/real downsample with anti-aliasing.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zrdesampD (DSPDoubleSplitComplex * A,

 int I,

 double * B,

 DSPDoubleSplitComplex * C,

 int N,

 int M);
```

Parameters

A

Double-precision complex input vector.

I

Complex decimation factor.

B

Filter coefficient vector.

C

Double-precision complex output vector.

N

Length of output vector.

M

Length of real filter vector.

Discussion

Performs finite impulse response (FIR) filtering at selected positions of input vector *A*.

$$C_m = \sum_{p=0}^{P-1} A(mi+q) \cdot Bq, \qquad (\mathrm{m} = \{0, \mathrm{N}\text{-}1\})$$

Length of *A* must be at least (N+M-1)*i. This function can run in place, but *C* cannot be in place with *B*.

Availability

Available in Mac OS X v10.4 and later.

vDSP_zrdotpr

Calculates the complex dot product of complex vector `A` and real vector `B` and leaves the result in complex vector `C`.

```
void
vDSP_zrdotpr (DSPSplitComplex * A,
    SInt32 I,
    const float B[],
    SInt32 J,
    DSPSplitComplex * C,
    UInt32 N);
```

Discussion

This performs the operation

$$C = \sum_{n=0}^{N-1} A_{nI} B_{nJ}$$

Criteria to invoke vectorized code:

- The vectors `A->realp`, `A->imagp`, `B->realp`, and `B->imagp` must be relatively aligned.
- The value of `N` must be equal to or greater than 16.
- The values of `I`, and `J` must be 1.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability

Available in Mac OS X v10.4 and later.

vDSP_zrdotprD

Calculates the complex dot product of complex vector `A` and real vector `B` and leaves the result in complex vector `C`.

```
void
vDSP_zrdotprD (DSPDoubleSplitComplex * A,
    SInt32 I,
    const double B[],
    SInt32 J,
    DSPDoubleSplitComplex * C,
    UInt32 N);
```

Discussion

This performs the operation

$$C = \sum_{n=0}^{N-1} A_{nI} B_{nJ}$$

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_zrvadd

Adds real vector `B` to complex vector `A` and leaves the result in complex vector `C`.

```
void
vDSP_zrvadd (DSPSplitComplex * A,
    SInt32 I,
    const float B[],
    SInt32 J,
    DSPSplitComplex * C,
    SInt32 K,
    UInt32 N);
```

Discussion
This performs the operation

$$C_{nK} = A_{nI} + B_{nJ} \qquad n = \{0, N\text{-}1\}$$

Criteria to invoke vectorized code:

- The vectors `A->realp`, `A->imagp`, `B->realp`, `B->imagp`, `C->realp`, and `C->imagp` must be relatively aligned.
- The value of `N` must be equal to or greater than 8.
- The values of `I`, `J`, and K must be 1.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_zrvaddD

Adds real vector `B` to complex vector `A` and leaves the result in complex vector `C`.

```
void
vDSP_zrvaddD (DSPDoubleSplitComplex * signaAll,
    SInt32 I,
    const float B[],
    SInt32 J,
    DSPSplitComplex * C,
    SInt32 K,
    UInt32 N);
```

Discussion
This performs the operation

$$C_{nK} = A_{nI} + B_{nJ} \qquad n = \{0, N-1\}$$

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_zrvdiv

Divides complex vector `A` by real vector `B` and leaves the result in vector `C`.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zrvdiv (DSPSplitComplex * A,
    int I,
    float *B,
    int J,
    DSPSplitComplex * C,
    int K,
    unsigned int N);
```

Discussion
This performs the operation

$$C_{nk} = \frac{A_{ni}}{B_{nj}} \qquad n = \{0, N-1\}$$

Availability
Available in Mac OS X v10.4 and later.

vDSP_zrvdivD

Divides complex vector `A` by real vector `B` and leaves the result in vector `C`.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zrvdivD (DSPDoubleSplitComplex * A,
    int I,
    double * B,
    int J,
    DSPDoubleSplitComplex * C,
    int K,
    unsigned int N);
```

Discussion
This peforms the operation

$$C_{nk} = \frac{A_{ni}}{B_{nj}} \qquad n = \{0, N-1\}$$

Availability
Available in Mac OS X v10.4 and later.

vDSP_zrvmul

Multiplies complex vector `A` by real vector `B` and leaves the result in vector `C`.

```
void
vDSP_zrvmul (DSPSplitComplex * A,
    SInt32 I,
    const float B[],
    SInt32 J,
    DSPSplitComplex * C,
    SInt32 K,
    UInt32 N);
```

Discussion
This peforms the operation

$$C_{nK} = A_{nI} \cdot B_{nJ} \qquad n = \{0, N-1\}$$

Criteria to invoke vectorized code:

- The vectors `A->realp`, `A->imagp`, `B->realp`,`B->imagp`, `C->realp`, and `C->imagp` must be relatively aligned.
- The value of `N` must be equal to or greater than 8.
- The values of `I`, `J`, and K must be 1.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_zrvmulD

Multiplies complex vector `A` by real vector `B` and leaves the result in vector `C`.

```
void
vDSP_zrvmulD (DSPDoubleSplitComplex * A,
    SInt32 I,
    const double B[],
    SInt32 J,
    DSPDoubleSplitComplex * C,
    SInt32 K,
    UInt32 N);
```

Discussion

This performs the operation

$$C_{nK} = A_{nI} \bullet B_{nJ} \qquad n = \{0, N-1\}$$

Criteria to invoke vectorized code:

No altivec support for double precision. This function always invokes scalar code.

Availability

Available in Mac OS X v10.4 and later.

vDSP_zrvsub

Subtracts real vector `B` from complex vector `A` and leaves the result in complex vector `C`.

```
void
vDSP_zrvsub (DSPSplitComplex * A,
    SInt32 I,
    const float B[],
    SInt32 J,
    DSPSplitComplex * C,
    SInt32 K,
    UInt32 N);
```

Discussion

This performs the operation

$$C_{nK} = A_{nI} - B_{nJ} \qquad n = \{0, N-1\}$$

Criteria to invoke vectorized code:

- The vectors `A->realp`, `A->imagp`, `B->realp`,`B->imagp`, `C->realp`, and `C->imagp` must be relatively aligned.
- The value of `N` must be equal to or greater than 8.
- The values of `I`, `J`, and K must be 1.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability

Available in Mac OS X v10.4 and later.

vDSP_zrvsubD

Subtracts real vector B from complex vector A and leaves the result in complex vector C.

```
void
vDSP_zrvsubD (DSPDoubleSplitComplex * A,
    SInt32 I,
    const double B[],
    SInt32 J,
    DSPDoubleSplitComplex * C,
    SInt32 K,
    UInt32 N);
```

Discussion

This peforms the operation

$$C_{nK} = A_{nI} - B_{nJ} \qquad n = \{0, N-1\}$$

Criteria to invoke vectorized code:

No altivec support for double precision. This function always invokes scalar code.

Availability

Available in Mac OS X v10.4 and later.

vDSP_ztoc

Copies the contents of a split complex vector A to an interleaved complex vector C.

```
void
vDSP_ztoc (const DSPSplitComplex * A,
    SInt32 I,
    float *C,
    SInt32 K,
    SInt32 N);
```

Discussion

This peforms the operation

$$C_{nK} = Re(A_{nI})$$

$$C_{nK+1} = Im(A_{nI}) \qquad n = \{0, N-1\}$$

K is an address stride through *C*. *I* is an address stride through *A*.

For best performance, C->realp, C->imagp, A->realp, and A->imagp should be 16-byte aligned.

Criteria to invoke vectorized code:

- The value of N must be greater than 3.
- The value of K must be 2.
- The value of I must be 1.
- Vectors A->realp and A->imagp must be relatively aligned.

- Vector `C` must 8 bytes aligned if `A->realp` and `A->imagp` are 4-byte aligned or `C` must be 16 bytes aligned if `A->realp` and `A->imagp` are at least 8-byte aligned.

If any of these criteria is not satisfied, the function invokes scalar code.

See also "vDSP_ctoz" (page 46) and "vDSP_ctozD" (page 46).

Availability
Available in Mac OS X v10.4 and later.

vDSP_ztocD

Copies the contents of a split complex vector `A` to an interleaved complex vector `C`.

```
void
vDSP_ztocD (const DSPDoubleSplitComplex * A,
    SInt32 I,
    DSPDoubleComplex * C,
    SInt32 K,
    SInt32 N);
```

Discussion
This peforms the operation

$$C_{nK} = Re(A_{nI})$$

$$C_{nK+1} = Im(A_{nI}) \qquad n = \{0, N-1\}$$

`K` is an address stride through `C`. `I` is an address stride through `A`.

For best performance, `C->realp`, `C->imagp`, `A->realp`, and `A->imagp` should be 16-byte aligned.

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

See also "vDSP_ctoz" (page 46) and "vDSP_ctozD" (page 46).

Availability
Available in Mac OS X v10.4 and later.

vDSP_ztrans

Transfer function.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_ztrans (float * A,
    DSPSplitComplex * B,
    DSPSplitComplex * C,
    unsigned int N);
```

Parameters

A
: Single-precision real input vector

B
: Single-precision complex input vector

C
: Single-precision complex output vector

N
: Count

Discussion

This peforms the operation

$$C_n = \frac{B_n}{A_n} \qquad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

vDSP_ztransD

Transfer function.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_ztransD (double * A,
    DSPDoubleSplitComplex * B,
    DSPDoubleSplitComplex * C,
    unsigned int N);
```

Parameters

A
: Double-precision real input vector

B
: Double-precision complex input vector

C
: Double-precision complex output vector

N
: Count

Discussion

This peforms the operation

$$C_n = \frac{B_n}{A_n} \qquad n = \{0, N\text{-}1\}$$

Availability

Available in Mac OS X v10.4 and later.

vDSP_zvabs

Complex vector absolute value.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zvabs (DSPSplitComplex * A,
    int I,
    float * C,
    int K,
    unsigned int N);
```

Parameters

A
: Single-precision complex input vector

I
: Stride for *A*

C
: Single-precision real output vector

K
: Stride for C

N
: Count

Discussion

This peforms the operation

$$C_{nk} = \sqrt{Re[A_{ni}]^2 + Im[A_{ni}]^2} \qquad n = \{0, N\text{-}1\}$$

Availability

Available in Mac OS X v10.4 and later.

vDSP_zvabsD

Complex vector absolute value.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zvabsD (DSPDoubleSplitComplex * A,
    int I,
    double * C,
    int K,
    unsigned int N);
```

Parameters

A
: Double-precision complex input vector

I
: Stride for `A`

C
: Double-precision real output vector

K
: Stride for C

N
: Count

Discussion

This peforms the operation

$$C_{nk} = \sqrt{Re[A_{ni}]^2 + Im[A_{ni}]^2} \qquad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

vDSP_zvadd

Adds complex vectors `A` and `B` and leaves the result in complex vector `C`

```
void
vDSP_zvadd (DSPSplitComplex * A,
    SInt32 I,
    DSPSplitComplex * B,
    SInt32 J,
    DSPSplitComplex * C,
    SInt32 K,
    UInt32 N);
```

Discussion

This peforms the operation

$$C_{nK} = A_{nI} + B_{nJ} \qquad n = \{0, N-1\}$$

Criteria to invoke vectorized code:

- The vectors `A->realp`, `A->imagp`, `B->realp`, `B->imagp`, `C->realp`, and `C->imagp` must be relatively aligned.

- The value of C must be equal to or greater than 8.
- The values of I, J, and K must be 1.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_zvaddD

Adds complex vectors A and B and leaves the result in complex vector C

```
void
vDSP_zvaddD (DSPDoubleSplitComplex * A,
    SInt32 I,
    DSPDoubleSplitComplex * B,
    SInt32 J,
    DSPDoubleSplitComplex * C,
    SInt32 K,
    UInt32 size);
```

Discussion
This peforms the operation

$$C_{nK} = A_{nI} + B_{nJ} \qquad n = \{0, N-1\}$$

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_zvcma

Multiplies complex vector B by the complex conjugates of complex vector A, adds the products to complex vector C, and stores the results in complex vector D.

```
void
vDSP_zvcma (DSPSplitComplex * A,
    SInt32 I,
    DSPSplitComplex * B,
    SInt32 J,
    DSPSplitComplex * C,
    SInt32 K,
    DSPSplitComplex * D,
    SInt32 L,
    UInt32 N);
```

Discussion
This peforms the operation

$D_{nL} \;=\; A^{*}_{nI} B_{nJ} \;+\; C_{nK} \qquad n = \{0, N\text{-}1\}$

Criteria to invoke vectorized code:

- The vectors `A->realp`, `A->imagp`, `B->realp`, `B->imagp`, `C->realp`, `C->imagp`, `D->realp`, and `D->imagp` must be relatively aligned.
- The value of `N` must be equal to or greater than 8.
- The values of `I`, `J`, `K`, and `L` must be 1.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_zvcmaD

Multiplies complex vector `B` by the complex conjugates of complex vector `A`, adds the products to complex vector `C`, and stores the results in complex vector `D`.

```
void
vDSP_zvcmaD (DSPDoubleSplitComplex * A,
    SInt32 I,
    DSPDoubleSplitComplex * B,
    SInt32 J,
    DSPDoubleSplitComplex * C,
    SInt32 K,
    DSPDoubleSplitComplex * D,
    SInt32 L,
    UInt32 size);
```

Discussion
This peforms the operation

$D_{nL} \;=\; A^{*}_{nI} B_{nJ} \;+\; C_{nK} \qquad n = \{0, N\text{-}1\}$

Criteria to invoke vectorized code:

No altivec support for double precision. This function always invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_zvcmul

Complex vector conjugate and multiply.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zvcmul (DSPSplitComplex * A,
    int I,
    DSPSplitComplex * B,
    int J,
    DSPSplitComplex * C,
    int K,
    unsigned int N);
```

Parameters

A
: Single-precision complex input vector

I
: Stride for A

C
: Single-precision complex input vector

J
: Stride for B

C
: Single-precision complex output vector

K
: Stride for C

N
: Count

Discussion

Multiplies vector *B* by the complex conjugates of vector *A* and stores the results in vector *C*.

$$C_{nK} = A^*_{nI} B_{nJ}$$

Availability

Available in Mac OS X v10.4 and later.

vDSP_zvcmulD

Complex vector conjugate and multiply.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zvcmulD (DSPDoubleSplitComplex * A,
    int I,
    DSPDoubleSplitComplex * B,
    int J,
    DSPDoubleSplitComplex * C,
    int K,
    unsigned int N);
```

Parameters

A
: Double-precision complex input vector

I
: Stride for `A`

C
: Double-precision complex input vector

J
: Stride for `B`

C
: Double-precision complex output vector

K
: Stride for `C`

N
: Count

Discussion

Multiplies vector *B* by the complex conjugates of vector *A* and stores the results in vector *C*.

$$C_{nK} = A^*_{nI} B_{nJ}$$

Availability

Available in Mac OS X v10.4 and later.

vDSP_zvconj

Complex vector conjugate.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zvconj (DSPSplitComplex * A,
    int I,
    DSPSplitComplex * C,
    int K,
    unsigned int N);
```

Parameters

A
: Single-precision complex input vector

I
: Stride for `A`

C
: Single-precision complex output vector

K
: Stride for `C`

N
: Count

Discussion

Conjugates vector *A*.

$$C_{nk} = A^*_{ni}$$

Availability

Available in Mac OS X v10.4 and later.

vDSP_zvconjD

Complex vector conjugate.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zvconjD (DSPDoubleSplitComplex * A,
    int I,
    DSPDoubleSplitComplex * C,
    int K,
    unsigned int N);
```

Parameters

A
: Double-precision complex input vector

I
: Stride for `A`

C
: Double-precision complex output vector

K
: Stride for `C`

N
: Count

Discussion

Conjugates vector *A*.

$$C_{nk} = A^*_{ni}$$

Availability
Available in Mac OS X v10.4 and later.

vDSP_zvdiv

Complex vector divide.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zvdiv (DSPSplitComplex * A,
    int I,
    DSPSplitComplex * B,
    intJ,
    DSPSplitComplex * C,
    int K,
    unsigned int N);
```

Parameters

A
Single-precision complex input vector

I
Stride for *A*

B
Single-precision complex input vector

J
Stride for *B*

C
Single-precision complex output vector

K
Stride for *C*

N
Count

Discussion
Divides vector *B* by vector *A*.

$$C_{nK} = \frac{B_{nJ}}{A_{nI}} \qquad n = \{0, N-1\}$$

Availability
Available in Mac OS X v10.4 and later.

vDSP_zvdivD

Complex vector divide.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zvdivD (DSPDoubleSplitComplex * A,
    int I,
    DSPDoubleSplitComplex * B,
    intJ,
    DSPDoubleSplitComplex * C,
    int K,
    unsigned int N);
```

Parameters

A

Double-precision complex input vector

I

Stride for *A*

B

Double-precision complex input vector

J

Stride for *B*

C

Double-precision complex output vector

K

Stride for *C*

N

Count

Discussion

Divides vector *B* by vector *A*.

$$C_{nK} = \frac{B_{nJ}}{A_{nI}} \qquad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

vDSP_zvfill

Complex vector fill.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zvfill (DSPSplitComplex * A,
    DSPSplitComplex * C,
    int K,
    unsigned int N);
```

Parameters

A

Single-precision complex input scalar

C

Single-precision complex output vector

K

Stride for *C*

N

Count

Discussion

Sets each element in complex vector *C* to complex scalar *A*.

$$C_{nK} = A \qquad n = \{0, N\text{-}1\}$$

Availability

Available in Mac OS X v10.4 and later.

vDSP_zvfillD

Complex vector fill.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zvfillD (DSPDoubleSplitComplex * A,
    DSPDoubleSplitComplex * C,
    int K,
    unsigned int N);
```

Parameters

A

Double-precision complex input scalar

C

Double-precision complex output vector

K

Stride for *C*

N

Count

Discussion

Sets each element in complex vector *C* to complex scalar *A*.

$$C_{nK} = A \qquad n = \{0, N\text{-}1\}$$

Availability
Available in Mac OS X v10.4 and later.

vDSP_zvmags

Complex vector magnitudes squared.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zvmags (DSPSplitComplex * A,
    int I,
    float * C,
    int K,
    unsigned int N);
```

Parameters

A
Single-precision complex input vector

I
Stride for *A*

C
Single-precision real output vector

K
Stride for *C*

N
Count

Discussion
Calculates the squared magnitudes of complex vector *A*.

$$C_{nK} = (Re[A_{nI}])^2 + (Im[A_{nI}])^2 \qquad n = \{0, N\text{-}1\}$$

Availability
Available in Mac OS X v10.4 and later.

vDSP_zvmagsD

Complex vector magnitudes squared.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zvmagsD (DSPDoubleSplitComplex * A,
    int I,
    double * C,
    int K,
    unsigned int N);
```

Parameters

A

Double-precision complex input vector

I

Stride for *A*

C

Double-precision real output vector

K

Stride for *C*

N

Count

Discussion

Calculates the squared magnitudes of complex vector *A*.

$$C_{nK} = (Re[A_{nI}])^2 + (Im[A_{nI}])^2 \qquad \text{n} = \{0, \text{N-1}\}$$

Availability

Available in Mac OS X v10.4 and later.

vDSP_zvmgsa

Complex vector magnitudes square and add.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zvmgsa (DSPSplitComplex * A,
    int I,
    float * B,
    int J,
    float * C,
    int K,
    unsigned int N);
```

Parameters

A

Single-precision complex input vector

I

Stride for *A*

B

Single-precision real input vector

J

Stride for *B*

C

Single-precision real output vector

K

Stride for *C*

N

Count

Discussion

Adds the squared magnitudes of complex vector *A* to real vector *B* and store the results in real vector *C*.

$$C_{nK} = [Re[A_{nI}]]^2 + [Im[A_{nI}]]^2 + B_{nJ} \qquad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

vDSP_zvmgsaD

Complex vector magnitudes square and add.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zvmgsaD (DSPDoubleSplitComplex * A,
    int I,
    double * B,
    int J,
    double * C,
    int K,
    unsigned int N);
```

Parameters

A

Double-precision complex input vector

I

Stride for *A*

B

Double-precision real input vector

J

Stride for *B*

C

Double-precision real output vector

K
: Stride for *C*

N
: Count

Discussion

Adds the squared magnitudes of complex vector *A* to real vector *B* and store the results in real vector *C*.

$$C_{nK} = [Re[A_{nI}]]^2 + [Im[A_{nI}]]^2 + B_{nJ} \qquad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

vDSP_zvmov

Complex vector move.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zvmov (DSPSplitComplex * A,
    int I,
    DSPSplitComplex * C,
    int K,
    unsigned int N);
```

Parameters

A
: Single-precision complex input vector

I
: Stride for *A*

C
: Single-precision complex output vector

K
: Stride for *C*

N
: Count

Discussion

Copies complex vector *A* to complex vector *C*.

$$C_{nK} = A_{nI}$$

Availability

Available in Mac OS X v10.4 and later.

vDSP_zvmovD

Complex vector move.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zvmovD (DSPDoubleSplitComplex * A,
    int I,
    DSPDoubleSplitComplex * C,
    int K,
    unsigned int N);
```

Parameters

A
: Double-precision complex input vector

I
: Stride for *A*

C
: Double-precision complex output vector

K
: Stride for *C*

N
: Count

Discussion

Copies complex vector *A* to complex vector *C*.

$C_{nK} = A_{nI}$

Availability

Available in Mac OS X v10.4 and later.

vDSP_zvmul

Multiplies complex vectors `A` and `B` and leaves the result in complex vector `C`.

```
void
vDSP_zvmul (DSPSplitComplex * A,
    SInt32 I,
    DSPSplitComplex * B,
    SInt32 J,
    DSPSplitComplex * C,
    SInt32 K,
    UInt32 N,
    SInt32 F);
```

Discussion

Pass 1 or -1 for *F*, for normal or conjugate multiplication, respectively. Results are undefined for other values of *F*.

$Re[C_{nK}] = Re[A_{nI}]Re[B_{nJ}] - F(Im[A_{nI}]Im[B_{nJ}])$

$Im[C_{nK}] = Re[A_{nI}]Im[B_{nJ}] + F(Im[A_{nI}]Re[B_{nJ}])$

n = {0, N-1}

Criteria to invoke vectorized code:

- The vectors `A->realp`, `A->imagp`, `B->realp`, `B->imagp`, `C->realp`, and `C->imagp` must be relatively aligned.
- The value of `N` must be equal to or greater than 8.
- The values of `I`, `J`, and `K` must be 1.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_zvmulD

Multiplies complex vectors `A` and `B` and leaves the result in complex vector `C`.

```
void
vDSP_zvmulD (DSPDoubleSplitComplex * A,
    SInt32 I,
    DSPDoubleSplitComplex * B,
    SInt32 J,
    DSPDoubleSplitComplex * C,
    SInt32 K,
    UInt32 N,
    SInt32 F);
```

Discussion
Pass 1 or -1 for `F`, for normal or conjugate multiplication, respectively. Results are undefined for other values of `F`.

$Re[C_{nK}] = Re[A_{nI}]Re[B_{nJ}] - F(Im[A_{nI}]Im[B_{nJ}])$

$Im[C_{nK}] = Re[A_{nI}]Im[B_{nJ}] + F(Im[A_{nI}]Re[B_{nJ}])$

n = {0, N-1}

Criteria to invoke vectorized code:

No altivec support for double precision. This function always invokes scalar code.

Availability
Available in Mac OS X v10.4 and later.

vDSP_zvneg

Complex vector negate.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zvneg (DSPSplitComplex * A,
    int I,
    DSPSplitComplex * C,
    int K,
    unsigned int N);
```

Parameters

A

Single-precision complex input vector

I

Stride for *A*

C

Single-precision complex output vector

K

Stride for *C*

N

Count

Discussion

Computes the negatives of the values of complex vector *A* and puts them into complex vector *C*.

$C_{nK} = -A_{nI}$

Availability

Available in Mac OS X v10.4 and later.

vDSP_zvnegD

Complex vector negate.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zvnegD (DSPDoubleSplitComplex * A,
    int I,
    DSPDoubleSplitComplex * C,
    int K,
    unsigned int N);
```

Parameters

A

Double-precision complex input vector

I

Stride for *A*

C

Double-precision complex output vector

K

Stride for *C*

N

Count

Discussion

Computes the negatives of the values of complex vector *A* and puts them into complex vector *C*.

$$C_{nK} = -A_{nI}$$

Availability

Available in Mac OS X v10.4 and later.

vDSP_zvphas

Complex vector phase.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zvphas (DSPSplitComplex * A,
    int I,
    float * C,
    int K,
    unsigned int N);
```

Parameters

A

Single-precision complex input vector

I

Stride for *A*

C

Single-precision real output vector

K

Stride for *C*

N

Count

Discussion

Finds the phase values, in radians, of complex vector *A* and store the results in real vector *C*. The results are between –pi and +pi. The sign of the result is the sign of the second coordinate in the input vector.

$$C_{nK} = \operatorname{atan}\frac{Im[A_{nI}]}{Re[A_{nI}]} \qquad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

vDSP_zvphasD

Complex vector phase.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zvphasD (DSPDoubleSplitComplex * A,
    int I,
    double * C,
    int K,
    unsigned int N);
```

Parameters

A
: Double-precision complex input vector

I
: Stride for *A*

C
: Double-precision real output vector

K
: Stride for *C*

N
: Count

Discussion

Finds the phase values, in radians, of complex vector *A* and store the results in real vector *C*. The results are between –pi and +pi. The sign of the result is the sign of the second coordinate in the input vector.

$$C_{nK} = \operatorname{atan}\frac{Im[A_{nI}]}{Re[A_{nI}]} \qquad n = \{0, N-1\}$$

Availability

Available in Mac OS X v10.4 and later.

vDSP_zvsma

Complex vector scalar multiply and add.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zvsma (DSPSplitComplex * A,
    int I,
    DSPSplitComplex * B,
    DSPSplitComplex * C,
    int K,
    DSPSplitComplex * D,
    int L,
    unsigned int N);
```

Parameters

A

Single-precision complex input vector

I

Stride for *A*

B

Single-precision complex input scalar

C

Single-precision real input vector

K

Stride for *C*

D

Single-precision real output vector

L

Stride for *C*

N

Count

Discussion

Multiplies vector *A* by scalar *B* and add the products to vector *C*. The result is stored in vector *D*.

$$D_{nL} = A_{nI}B + C_{nK}$$

Availability

Available in Mac OS X v10.4 and later.

vDSP_zvsmaD

Complex vector scalar multiply and add.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zvsmaD (DSPDoubleSplitComplex * A,
    int I,
    DSPDoubleSplitComplex * B,
    DSPDoubleSplitComplex * C,
    int K,
    DSPDoubleSplitComplex * D,
    int L,
    unsigned int N);
```

Parameters

A
Double-precision complex input vector

I
Stride for *A*

B
Double-precision complex input scalar

C
Double-precision real input vector

K
Stride for *C*

D
Double-precision real output vector

L
Stride for *C*

N
Count

Discussion

Multiplies vector *A* by scalar *B* and add the products to vector *C*. The result is stored in vector *D*.

$$D_{nL} = A_{nI}B + C_{nK}$$

Availability

Available in Mac OS X v10.4 and later.

vDSP_zvsub

Subtracts complex vector `B` from complex vector `A` and leaves the result in complex vector `C`

```
void
vDSP_zvsub (DSPSplitComplex * A,
    SInt32 I,
    DSPSplitComplex * B,
    SInt32 J,
    DSPSplitComplex * C,
    SInt32 K,
    UInt32 N);
```

Discussion

This peforms the operation

$$C_{nK} = A_{nI} - B_{nJ} \qquad n = \{0, N\text{-}1\}$$

Criteria to invoke vectorized code:

- The vectors `A->realp`, `A->imagp`, `B->realp`, `B->imagp`, `C->realp`, and `C->imagp` must be relatively aligned.
- The value of `N` must be equal to or greater than 8.
- The values of `I`, `J`, and `K` must be 1.

If any of these criteria is not satisfied, the function invokes scalar code.

Availability

Available in Mac OS X v10.4 and later.

vDSP_zvsubD

Subtracts complex vector `B` from complex vector `A` and leaves the result in complex vector `C`

```
void
vDSP_zvsubD (DSPDoubleSplitComplex * A,
    SInt32 I,
    DSPDoubleSplitComplex * B,
    SInt32 J,
    DSPDoubleSplitComplex * C,
    SInt32 K,
    UInt32 N);
```

Discussion

This peforms the operation

$$C_{nK} = A_{nI} - B_{nJ} \qquad n = \{0, N\text{-}1\}$$

Criteria to invoke vectorized code:

No altivec support for double precision. The function always invokes scalar code.

Availability

Available in Mac OS X v10.4 and later.

vDSP_zvzsml

Complex vector multiply by complex scalar.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zvzsml (DSPSplitComplex * A,
    int I,
    DSPSplitComplex * B,
    DSPSplitComplex * C,
    int K,
    unsigned int N);
```

Parameters

A
: Single-precision complex input vector

I
: Stride for *A*

B
: Single-precision complex input scalar

C
: Single-precision complex output vector

K
: Stride for *C*

N
: Count

Discussion

This peforms the operation

$$C_{nK} = A_{nI}B$$

Availability

Available in Mac OS X v10.4 and later.

vDSP_zvzsmlD

Complex vector multiply by complex scalar.

AVAILABLE IN MAC OS X VERSION 10 4 AND LATER

```
void
vDSP_zvzsml (DSPDoubleSplitComplex * A,
    int I,
    DSPDoubleSplitComplex * B,
    DSPDoubleSplitComplex * C,
    int K,
    unsigned int N);
```

Parameters

A
: Double-precision complex input vector

I
: Stride for *A*

B
: Double-precision complex input scalar

C
: Double-precision complex output vector

K
: Stride for *C*

N
: Count

Discussion

This peforms the operation

$$C_{nK} = A_{nI}B$$

Availability

Available in Mac OS X v10.4 and later.

Sample Code

This chapter contains sample code showing how the vDSP functions are used. The sample code also includes performance tests using the computer's microsecond timer.

vecLib Simple Sample

This is a sample program showing how to use some of the vecLib functions.

Listing A-1 vecLib sample

```
/***************************************************************************

     File Name: vecLibSimpleSample.c

     A sample program to illustrate the usage of some of the vecLib
     functions.

     Copyright © 2000 Apple Computer, Inc.  All rights reserved.

***************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <Types.h>
#include "vDSP.h"

main()
      {

 /************************************************************************
     Use of vecLib simple functions such as vector addition, subtraction,
     multiply, square and dot product.
 ************************************************************************/

      float  *inputOne, *inputTwo, *result, dotProduct;
      SInt32 strideOne, strideTwo, resultStride;
      UInt32 i, size;

      size = 1024;
```

```
    // use vec_malloc to insure proper alignment.
    inputOne = ( float* ) vec_malloc ( size * sizeof ( float )  );
    inputTwo = ( float* ) vec_malloc ( size * sizeof ( float )  );
    result   = ( float* ) vec_malloc ( size * sizeof ( float )  );

    if( inputOne == NULL || inputTwo == NULL || result == NULL  )
          {
          printf ( "\nmalloc failed to allocate memory.\n");
          exit ( 0 );
          }

    // Set the input arguments of length "size" to:  [1,...,1] and [2,...,2].
    for( i = 0; i < size; i++ )
          {
          inputOne[i] = 1.0;
          inputTwo[i] = 2.0;
          }

    strideOne = strideTwo = resultStride = 1;

     // Add the input vectors "inputOne" and "inputTwo".
     // The result is returned in the vector "result".
     vDSP_vadd ( inputOne, strideOne, inputTwo, strideTwo, result,
       resultStride, size );
    printf ( "the result of the addition is:\n" );
    for ( i = 0; i < size; i++ )
          printf ( "%13.6e\n", result[i] );

    // Subtract the input vectors "inputOne" and "inputTwo".
    // The result is returned in the vector "result".
    vDSP_vsub ( inputOne, strideOne, inputTwo, strideTwo, result,
      resultStride, size );
    printf ( "the result of the subtraction is:\n" );
    for ( i = 0; i < size; i++ )
          printf ( "%13.6e\n", result[i] );

    // Multiply the input vectors "inputOne" and "inputTwo".
    // The result is returned in the vector "result".
    vDSP_vmul ( inputOne, strideOne, inputTwo, strideTwo, result,
      resultStride, size );
    printf ( "the result of the multiplication is:\n"  );
    for ( i = 0; i < size; i++ )
          printf ( "%13.6e\n", result[i] );

    // Square the input vector "inputTwo".
    // The result is returned in the vector "result".
    vDSP_vsq ( inputTwo, strideTwo, result, resultStride, size  );
    printf ( "the result of the square is:\n" );
    for ( i = 0; i < size; i++ )
          printf ( "%13.6e\n", result[i] );

    // Form dot product of the input vectors "inputOne"  and "inputTwo".
    // The result is returned in the scalar "dotProduct".
    vDSP_dotpr ( inputOne, strideOne, inputTwo, strideTwo, &dotProduct,
      size );
    printf ( "the result of the dot product is: %13.6e\n",  dotProduct );

    // free all allocated memory
```

```
      vec_free ( inputOne );
      vec_free ( inputTwo );
      vec_free ( result );

      return 0;
      }
```

FFT Sample

This is a sample program showing how to use the FFT functions.

Listing A-2 FFT sample

```
/************************************************************************

     File Name: FFTSample.c

     A sample program to illustrate the usage of complex, real,  1-d and 2-d
     FFT functions.  This program also times the functions using  the
     microsecond timer.

     Copyright © 2000 Apple Computer, Inc.  All rights reserved.

************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <Types.h>
//#include <OSUtils.h>
#include <Timer.h>
#include <Gestalt.h>
//#include <Errors.h>
#include <fp.h>
#include "vDSP.h"

#define MAX_LOOP_NUM       10000  // Number of iterations used in  the timing
loop
#define kHasAltiVecMask    ( 1 << gestaltPowerPCHasVectorInstructions  )
// used in looking for a g4
#define MAX(a,b)           ( (a>b)?a:b )
#define N                  10     // This is a power of 2 defining  the length
 of the FFTs

static UnsignedWide StartTime, endTime;

Boolean HasAltivec     ( );
void TurnJavaModeOff   ( vector unsigned long *oldJavaMode );
void RestoreJavaMode   ( vector unsigned long *oldJavaMode );
void Start_Clock       ( void );
void Stop_Clock        ( float *call_time );
void Compare           ( float *original, float *computed, long  length );
void Dummy_fft_zip    ( FFTSetup setup, COMPLEX_SPLIT  *C, long  stride, unsigned
  long log2n, long flag );
void Dummy_fft2d_zip   ( FFTSetup setup, COMPLEX_SPLIT *A, long  rowStride, long
  columnStride, unsigned log2nr, unsigned log2nc, long flag );
```

```
void Dummy_fft_zrip    ( FFTSetup setup, COMPLEX_SPLIT  *C, long  stride, unsigned
  long log2n, long flag );
void Dummy_ctoz        ( COMPLEX *C, SInt32 strideC, COMPLEX_SPLIT  *Z, SInt32
  strideZ, SInt32 size );
void Dummy_ztoc        ( COMPLEX_SPLIT *A, SInt32 strideZ, COMPLEX  *C, SInt32
  strideC, SInt32 size );
void RealFFTUsageAndTiming     ( );
void ComplexFFTUsageAndTiming  ( );
void Complex2dFFTUsageAndTiming( );


main ( )
      {
      RealFFTUsageAndTiming ( );
      ComplexFFTUsageAndTiming ( );
      Complex2dFFTUsageAndTiming ( );

      return 0;
    }


/*************************************************************************
     Real FFT.

     The real FFT, unlike the complex FFT, may possibly have to  use two
     transformation functions, one before the FFT call and one after.
     This is if the input array is not in the even-odd split configuration.

     A real array A = {A[0],...,A[n]} has to be transformed into  an
     even-odd array

        AEvenOdd = {A[0],A[2],...,A[n-1],A[1],A[3],...A[n]}

     via the vDSP_ctoz call.

     The result of the real FFT of AEvenOdd of dimension n is a  complex
     array of the dimension 2n, with a special format:

       {[DC,0],C[1],C[2],...,C[n/2],[NY,0],Cc[n/2],...,Cc[2],Cc[1]}

     where

     1. DC and NY are the dc and nyquist components (real valued),
     2. C is complex in a split representation,
     3. Cc is the complex conjugate of C in a split representation.

     For an n size real array A, the complex results require 2n
     spaces.  In order to fit the 2n size result into an n size  input and
     since the complex conjugates are duplicate information, the  real
     FFT produces its results as follows:

       {[DC,NY],C[1],C[2],...,C[n/2]}.

     The time for a length 1024 real FFT with the transformation  functions
     on a 500mhz machine is 14.9s.  If the data structure of the  calling
     program renders the transformation functions unecessary, the  same
     signal is processed in 11.3s.
```

```
**************************************************************************/

void RealFFTUsageAndTiming ( )
    {
      COMPLEX_SPLIT A;
      FFTSetup      setupReal;
      UInt32        log2n;
      UInt32        n, nOver2;
      SInt32        stride;
      UInt32        i;
      float         *originalReal, *obtainedReal;
      float         scale;

      // Set the size of FFT.
      log2n = N;
      n = 1 << log2n;

      stride = 1;
      nOver2 = n / 2;

      printf ( "1D real FFT of length log2 ( %d ) = %d\n\n",  n, log2n );

      // Allocate memory for the input operands and check its availability,
      // use the vector version to get 16-byte alignment.
      A.realp = ( float* ) vec_malloc ( nOver2 * sizeof ( float  ) );
      A.imagp = ( float* ) vec_malloc ( nOver2 * sizeof ( float  ) );
      originalReal = ( float* ) vec_malloc ( n * sizeof ( float  ) );
      obtainedReal = ( float* ) vec_malloc ( n * sizeof ( float  ) );

      if ( originalReal == NULL || A.realp == NULL || A.imagp ==  NULL  )
        {
            printf ( "\nmalloc failed to allocate memory for  the real FFT
              section of the sample.\n" );
            exit ( 0 );
        }

     // Generate an input signal in the real domain.
     for ( i = 0; i < n; i++ )
       originalReal[i] = ( float ) ( i+1 );

     // Look at the real signal as an interleaved complex vector  by
      // casting it.
      // Then call the transformation function vDSP_ctoz to get  a split
      // complex vector,
      // which for a real signal, divides into an even-odd configuration.
      vDSP_ctoz ( ( COMPLEX * ) originalReal, 2, &A, 1, nOver2  );

      // Set up the required memory for the FFT routines and check  its
      // availability.
     setupReal = vDSP_create_fftsetup ( log2n, FFT_RADIX2);
     if ( setupReal == NULL )
     {
          printf ( "\nFFT_Setup failed to allocate enough memory  for
          the real FFT.\n" );
          exit ( 0 );
     }

      // Carry out a Forward and Inverse FFT transform.
      vDSP_fft_zrip ( setupReal, &A, stride, log2n, FFT_FORWARD  );
      vDSP_fft_zrip ( setupReal, &A, stride, log2n, FFT_INVERSE  );

      // Verify correctness of the results, but first scale it by  2n.
      scale = (float)1.0/(2*n);

      vDSP_vsmul( A.realp, 1, &scale, A.realp, 1, nOver2 );
      vDSP_vsmul( A.imagp, 1, &scale, A.imagp, 1, nOver2 );

      // The output signal is now in a split real form.  Use the  function
      // vDSP_ztoc to get a split real vector.
      vDSP_ztoc ( &A, 1, ( COMPLEX * ) obtainedReal, 2, nOver2  );

      // Check for accuracy by looking at the inverse transform  results.
      Compare ( originalReal, obtainedReal, n );

       // Timing section for the real 1d FFT plus the translation  functions.
          {
            float time, overheadTime;
            vector unsigned long oldJavaMode;

     // Turn Java mode off.  Otherwise, an extra cycle is added  to the vfpu.
     // WARNING:  Java mode has to be treated with care.  Some algorithms
     // may be sensitive to flush to zero and need proper IEEE-754
     // denormal handling.
            TurnJavaModeOff ( &oldJavaMode );

            Start_Clock ( );
            for ( i = 0; i < MAX_LOOP_NUM; i++ )
                {
                   vDSP_ctoz ( (COMPLEX *) originalReal, 2, &A,  1, nOver2 );
                   vDSP_fft_zrip (setupReal, &A, stride, log2n,
                     FFT_FORWARD );
                   vDSP_ztoc ( &A, 1, (COMPLEX *) obtainedReal,  2, nOver2 );
                }
            Stop_Clock ( &time );

            // Restore Java mode.
            RestoreJavaMode( &oldJavaMode );

            // Measure and take off the calling overhead of the  FFT and the
            // translation functions vDSP_ctoz and vDSP_ztoc
            // (minimal impact).

            Start_Clock();
            for ( i = 0; i < MAX_LOOP_NUM; i++ )
            {
              Dummy_ctoz ( ( COMPLEX * ) originalReal, 2, &A,  1, nOver2 );
              Dummy_fft_zrip (setupReal, &A, stride, log2n,  FFT_FORWARD );
              Dummy_ztoc ( &A, 1, ( COMPLEX * ) obtainedReal,  2, nOver2 );
            }
            Stop_Clock ( &overheadTime );

            time -= overheadTime;
            time /= MAX_LOOP_NUM;

            printf ( "\nTime for 1D real FFT of length log2  ( %d ) = %d
              plus the translation functions is %4.4f secs.",  n, log2n,
              time );

        }

        // Timing section for the real 1d FFT.
        {
            float time, overheadTime;
            vector unsigned long oldJavaMode;

     // Turn Java mode off.  Otherwise, an extra cycle is added  to the vfpu.
     // WARNING:  Java mode has to be treated with care.  Some algorithms
     // may be sensitive to flush to zero and need proper IEEE-754
     // denormal handling.
            TurnJavaModeOff ( &oldJavaMode );

            Start_Clock ( );
            for ( i = 0; i < MAX_LOOP_NUM; i++ )
            vDSP_fft_zrip ( setupReal, &A, stride, log2n, FFT_FORWARD  );
            Stop_Clock ( &time );

            // Restore Java mode.
            RestoreJavaMode( &oldJavaMode );

     // Measure and take off the calling overhead of the FFT and  the
     // translation functions vDSP_ctoz and vDSP_ztoc (minimal impact).
            Start_Clock ( );
            for ( i = 0; i < MAX_LOOP_NUM; i++ )
                  Dummy_fft_zrip ( setupReal, &A, stride, log2n,  FFT_FORWARD
);
            Stop_Clock ( &overheadTime );

            time -= overheadTime;
            time /= MAX_LOOP_NUM;

            printf ( "\nTime for 1D real FFT of length log2  ( %d ) = %d
              only is %4.4f secs.\n\n\n\n", n, log2n, time  );
        }

      // Free the allocated memory.
      vDSP_destroy_fftsetup ( setupReal );
      vec_free ( obtainedReal );
      vec_free ( originalReal );
      vec_free ( A.realp );
      vec_free ( A.imagp );
    }

/***********************************************************************

     File Name: FFTSample.c

     A sample program to illustrate the usage of complex, real,  1-d and 2-d
     FFT functions.  This program also times the functions using  the
     microsecond timer.

     Copyright © 2000 Apple Computer, Inc.  All rights reserved.

************************************************************************/
```

```
#include <stdio.h>
#include <stdlib.h>
#include <Types.h>
//#include <OSUtils.h>
#include <Timer.h>
#include <Gestalt.h>
//#include <Errors.h>
#include <fp.h>
#include "vDSP.h"

#define MAX_LOOP_NUM       10000  // Number of iterations used in  the timing
loop
#define kHasAltiVecMask    ( 1 << gestaltPowerPCHasVectorInstructions  )  //
used  in looking for a g4
#define MAX(a,b)           ( (a>b)?a:b )
#define N                  10     // This is a power of 2 defining  the length
 of the FFTs

static UnsignedWide StartTime, endTime;

Boolean HasAltivec     ( );
void TurnJavaModeOff   ( vector unsigned long *oldJavaMode );
void RestoreJavaMode   ( vector unsigned long *oldJavaMode );
void Start_Clock       ( void );
void Stop_Clock        ( float *call_time );
void Compare           ( float *original, float *computed, long  length );
void Dummy_fft_zip    ( FFTSetup setup, COMPLEX_SPLIT  *C, long  stride, unsigned
  long log2n, long flag );
void Dummy_fft2d_zip   ( FFTSetup setup, COMPLEX_SPLIT *A, long  rowStride, long
  columnStride, unsigned log2nr, unsigned log2nc, long flag );
void Dummy_fft_zrip    ( FFTSetup setup, COMPLEX_SPLIT  *C, long  stride, unsigned
  long log2n, long flag );
void Dummy_ctoz        ( COMPLEX *C, SInt32 strideC, COMPLEX_SPLIT  *Z, SInt32
  strideZ, SInt32 size );
void Dummy_ztoc        ( COMPLEX_SPLIT *A, SInt32 strideZ, COMPLEX  *C, SInt32
  strideC, SInt32 size );
void RealFFTUsageAndTiming     ( );
void ComplexFFTUsageAndTiming  ( );
void Complex2dFFTUsageAndTiming( );


main ( )
      {
        RealFFTUsageAndTiming ( );
        ComplexFFTUsageAndTiming ( );
        Complex2dFFTUsageAndTiming ( );

        return 0;
      }


/*************************************************************************
     Real FFT.

     The real FFT, unlike the complex FFT, may possibly have to  use two
     transformation functions, one before the FFT call and one after.
     This is if the input array is not in the even-odd split configuration.
     A real array A = {A[0],...,A[n]} has to be transformed into  an
     even-odd array

       AEvenOdd = {A[0],A[2],...,A[n-1],A[1],A[3],...A[n]}

     via thevDSP_ctoz call.

     The result of the real FFT of AEvenOdd of dimension n is a  complex
     array of dimension 2n, with a special format:

       {[DC,0],C[1],C[2],...,C[n/2],[NY,0],Cc[n/2],...,Cc[2],Cc[1]}

     where

     1. DC and NY are the dc and nyquist components (real valued),
     2. C is complex in a split representation,
     3. Cc is the complex conjugate of C in a split representation.

     For an n size real array A, the complex results require 2n
     spaces.  In order to fit the 2n size result into an n size  input and
     since the the complex conjugates are duplicate information,  the real
     FFT produces its results as follows:

       {[DC,NY],C[1],C[2],...,C[n/2]}.

     The time for a length 1024 real FFT with the transformation  functions
     on a 500mhz machine is 14.9s.  If the data structure of the  calling
     program renders the transformation functions unecessary, the  same
     signal is processed in 11.3s.

************************************************************************/

void RealFFTUsageAndTiming ( )
    {
      COMPLEX_SPLIT A;
      FFTSetup      setupReal;
      UInt32        log2n;
      UInt32        n, nOver2;
      SInt32        stride;
      UInt32        i;
      float         *originalReal, *obtainedReal;
      float         scale;

      // Set the size of FFT.
      log2n = N;
      n = 1 << log2n;

      stride = 1;
      nOver2 = n / 2;

      printf ( "1D real FFT of length log2 ( %d ) = %d\n\n",  n, log2n );

      // Allocate memory for the input operands and check its availability,
      // use the vector version to get 16-byte alignment.
      A.realp = ( float* ) vec_malloc ( nOver2 * sizeof ( float  ) );
      A.imagp = ( float* ) vec_malloc ( nOver2 * sizeof ( float  ) );
      originalReal = ( float* ) vec_malloc ( n * sizeof ( float  ) );
      obtainedReal = ( float* ) vec_malloc ( n * sizeof ( float  ) );
      if ( originalReal == NULL || A.realp == NULL || A.imagp ==  NULL  )
      {
            printf ( "\nmalloc failed to allocate memory for  the real FFT
              section of the sample.\n" );
            exit ( 0 );
      }

      // Generate an input signal in the real domain.
      for ( i = 0; i < n; i++ )
            originalReal[i] = ( float ) ( i+1 );

      // Look at the real signal as an interleaved complex vector  by casting
      // it.
      // Then call the transformation function vDSP_ctoz to get  a split
      // complex vector,
      // which for a real signal, divides into an even-odd configuration.
      vDSP_ctoz ( ( COMPLEX * ) originalReal, 2, &A, 1, nOver2  );

      // Set up the required memory for the FFT routines and check  its
      // availability.
      setupReal = vDSP_create_fftsetup ( log2n, FFT_RADIX2);
      if ( setupReal == NULL )
      {
            printf ( "\nFFT_Setup failed to allocate enough  memory for the
              real FFT.\n" );
            exit ( 0 );
      }

      // Carry out a Forward and Inverse FFT transform.
      vDSP_fft_zrip ( setupReal, &A, stride, log2n, FFT_FORWARD  );
      vDSP_fft_zrip ( setupReal, &A, stride, log2n, FFT_INVERSE  );

      // Verify correctness of the results, but first scale it by  2n.
      scale = (float)1.0/(2*n);

      vDSP_vsmul( A.realp, 1, &scale, A.realp, 1, nOver2 );
      vDSP_vsmul( A.imagp, 1, &scale, A.imagp, 1, nOver2 );

      // The output signal is now in a split real form.  Use the  function
      // vDSP_ztoc to get a split real vector.
      vDSP_ztoc ( &A, 1, ( COMPLEX * ) obtainedReal, 2, nOver2  );

      // Check for accuracy by looking at the inverse transform  results.
      Compare ( originalReal, obtainedReal, n );

      // Timing section for the real 1d FFT plus the translation  functions.
      {
            float time, overheadTime;
            vector unsigned long oldJavaMode;

     // Turn Java mode off.  Otherwise, an extra cycle is added  to the vfpu.
     // WARNING:  Java mode has to be treated with care.  Some algorithms
     // may be sensitive to flush to zero and need proper IEEE-754
     // denormal handling.
            TurnJavaModeOff ( &oldJavaMode );

            Start_Clock ( );
```

```
            for ( i = 0; i < MAX_LOOP_NUM; i++ )
            {
              vDSP_ctoz ( ( COMPLEX * ) originalReal, 2, &A,  1, nOver2 );
              vDSP_fft_zrip ( setupReal, &A, stride, log2n,  FFT_FORWARD );
              vDSP_ztoc ( &A, 1, ( COMPLEX * ) obtainedReal,  2, nOver2 );
            }
            Stop_Clock ( &time );

            // Restore Java mode.
            RestoreJavaMode( &oldJavaMode );

            // Measure and take off the calling overhead of the  FFT and the
            // translation functions vDSP_ctoz and vDSP_ztoc
            // (minimal impact).

            Start_Clock();
            for ( i = 0; i < MAX_LOOP_NUM; i++ )
            {
              Dummy_ctoz ( ( COMPLEX * ) originalReal, 2, &A,  1, nOver2 );
              Dummy_fft_zrip (setupReal, &A, stride, log2n,  FFT_FORWARD );
              Dummy_ztoc ( &A, 1, ( COMPLEX * ) obtainedReal,  2, nOver2 );
            }
            Stop_Clock ( &overheadTime );

            time -= overheadTime;
            time /= MAX_LOOP_NUM;

            printf ( "\nTime for 1D real FFT of length log2  ( %d ) = %d
              plus the translation functions is %4.4f secs.",  n,
              log2n, time );

       }

      // Timing section for the real 1d FFT.
      {
            float time, overheadTime;
            vector unsigned long oldJavaMode;

     // Turn Java mode off.  Otherwise, an extra cycle is added  to the vfpu.
     // WARNING:  Java mode has to be treated with care.  Some algorithms
     // may be sensitive to flush to zero and need proper IEEE-754
     // denormal handling.
            TurnJavaModeOff ( &oldJavaMode );

            Start_Clock ( );
            for ( i = 0; i < MAX_LOOP_NUM; i++ )
              vDSP_fft_zrip ( setupReal, &A, stride, log2n,  FFT_FORWARD );
            Stop_Clock ( &time );

            // Restore Java mode.
            RestoreJavaMode( &oldJavaMode );

           // Measure and take off the calling overhead of the FFT  and the
           // translation functions vDSP_ctoz and vDSP_ztoc
           // (minimal impact).
            Start_Clock ( );
            for ( i = 0; i < MAX_LOOP_NUM; i++ )
```

```
                  Dummy_fft_zrip ( setupReal, &A, stride, log2n,  FFT_FORWARD
);
            Stop_Clock ( &overheadTime );

            time -= overheadTime;
            time /= MAX_LOOP_NUM;

            printf ( "\nTime for 1D real FFT of length log2  ( %d ) = %d
              only is %4.4f secs.\n\n\n\n", n, log2n, time  );
      }

      // Free the allocated memory.
      vDSP_destroy_fftsetup ( setupReal );
      vec_free ( obtainedReal );
      vec_free ( originalReal );
      vec_free ( A.realp );
      vec_free ( A.imagp );
    }
```

Convolution Sample

This is a sample program showing how to use the convolution and correlation functions.

Listing A-3 Convolution sample code

```
/************************************************************************

     File Name: ConvolutionSample.c

     Sample program to illustrate the usage of convolution and correlation
     functions.  This also times the functions using the microsecond
     timer

     Copyright © 2000 Apple Computer, Inc.  All rights reserved.

*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <Types.h>
#include <Timer.h>
#include <Gestalt.h>
#include "vDSP.h"

#define MAX_LOOP_NUM       1000    // Number of iterations used  in the timing
 loop
#define kHasAltiVecMask    (1 << gestaltPowerPCHasVectorInstructions)   // used
 in  looking for a g4
#define MAX(a,b)           ( (a>b)?a:b )

static UnsignedWide startTime, endTime;

Boolean HasAltivec     ( );
void TurnJavaModeOff   ( vector unsigned long *oldJavaMode );
void RestoreJavaMode   ( vector unsigned long *oldJavaMode );
void Start_Clock       ( void );
void Stop_Clock        ( float *call_time );
void Compare           ( float *original, float *computed, long  length );
void Dummy_Conv        ( float *signal, SInt32 signalStride, float   *filter,
SInt32  filterStride, float  *result, SInt32 resultStride, SInt32 resultLength,
  SInt32  filterLength );

void ConvTiming ( );


int main()
      {
       ConvTiming();

       return 0;
      }

/************************************************************************
     Convolution.

     This function performs at 3.69 gigaflops for a convolution  of size
     ( 2048 x 256 ) on a 500mhz vector enabled processor.

************************************************************************/

void ConvTiming ( )
      {
      float  *signal, *filter, *result;
      SInt32 signalStride, filterStride, resultStride;
      UInt32 lenSignal, filterLength, resultLength;
      UInt32 i;

      filterLength = 256;
      resultLength = 2048;
      lenSignal = ( ( filterLength + 3 ) & 0xFFFFFFFC ) + resultLength;

      signalStride = filterStride = resultStride = 1;

      printf("\nConvolution ( resultLength = %d,
        filterLength = %d )\n\n",resultLength,filterLength);

      // Allocate memory for the input operands and check its availability.
      signal = ( float* ) vec_malloc ( lenSignal * sizeof ( float  ) );
      filter = ( float* ) vec_malloc ( filterLength * sizeof ( float  ) );
      result = ( float* ) vec_malloc ( resultLength * sizeof ( float  ) );

      if( signal == NULL || filter == NULL || result == NULL )
            {
            printf ( "\nmalloc failed to allocate memory for  the
              convolution sample.\n");
            exit ( 0 );
            }

      // Set the input signal of length "lenSignal" to  [1,...,1].
      for( i = 0; i < lenSignal; i++ )
            signal[i] = 1.0;

      // Set the filter of length "filterLength" to [1,...,1].
```

```
    for( i = 0; i < filterLength; i++ )
          filter[i] = 1.0;

    // Correlation.
    vDSP_conv ( signal, signalStride, filter, filterStride,
      result, resultStride, resultLength, filterLength );

    // Carry out a convolution.
    filterStride = -1;
    vDSP_conv ( signal, signalStride, filter + filterLength -  1,
      filterStride, result, resultStride, resultLength, filterLength  );

    // Timing section for the convolution.
          {
          float time, overheadTime;
          vector unsigned long oldJavaMode;
          float GFlops;

   // Turn Java mode off.  Otherwise, an extra cycle is added  to the vfpu.
   // WARNING:  Java mode has to be treated with care.  Some algorithms
   // may be sensitive to flush to zero and need proper IEEE-754
   // denormal handling.
          TurnJavaModeOff ( &oldJavaMode );

          Start_Clock ( );
          for ( i = 0; i < MAX_LOOP_NUM; i++ )
             vDSP_conv ( signal, signalStride, filter, filterStride,
               result, resultStride, resultLength, filterLength  );
          Stop_Clock ( &time );

          // Restore Java mode.
          RestoreJavaMode( &oldJavaMode );

          // Measure and take off the calling overhead of the  convolution
         // (minimal impact).
          Start_Clock ( );
          for ( i = 0; i < MAX_LOOP_NUM; i++ )
            Dummy_Conv ( signal, signalStride, filter, filterStride,
              result, resultStride, resultLength, filterLength  );
          Stop_Clock ( &overheadTime );

          time -= overheadTime;
          time /= MAX_LOOP_NUM;

          GFlops = ( 2.0f * filterLength - 1.0f ) * resultLength
            / ( time * 1000.0f );

          printf("Time for a ( %d x %d ) Convolution is %4.4f  secs
            or %4.4f GFlops\n\n", resultLength, filterLength,
            time, GFlops );
          }

    // Free allocated memory.
    vec_free ( signal );
    vec_free ( filter );
    vec_free ( result );
    }
```

```
void Start_Clock ( void )
      {
      Microseconds ( &startTime );
      }

void Stop_Clock ( float *call_time )
      {
      Microseconds ( &endTime );

      if ( endTime.hi == startTime.hi )
            *call_time = ( float ) ( endTime.lo - startTime.lo );
      else
            *call_time = -1.0f;
      }

void Dummy_Conv ( float *signal, SInt32 signalStride, float *filter,
  SInt32 filterStride, float *result, SInt32 resultStride,
  SInt32 resultLength, SInt32 filterLength )
      {
      #pragma unused( signal )
      #pragma unused( signalStride )
      #pragma unused( filter )
      #pragma unused( filterStride )
      #pragma unused( result )
      #pragma unused( resultStride )
      #pragma unused( resultLength )
      #pragma unused( filterLength )
      }

Boolean HasAltiVec ( )
      {
      Boolean hasAltiVec = false;
      OSErr   err;
      long    ppcFeatures;

      err = Gestalt ( gestaltPowerPCProcessorFeatures, &ppcFeatures  );
      if ( err == noErr)
            {
            if ( ( ppcFeatures & kHasAltiVecMask) != 0 )
                  hasAltiVec = true;
            }
      return hasAltiVec;
      }

void TurnJavaModeOff( vector unsigned long *oldJavaMode )
      {
      if ( HasAltiVec ( ) )
            {
            vector unsigned long javaOffMask = ( vector unsigned  long )
             ( 0x00010000 );
            vector unsigned long java;

            *oldJavaMode = ( vector unsigned long ) vec_mfvscr (   );

            java = vec_or ( *oldJavaMode, javaOffMask );
            vec_mtvscr ( java );
            }
      }
```

```
void RestoreJavaMode( vector unsigned long *oldJavaMode )
      {
      if ( HasAltiVec ( ) )
            vec_mtvscr ( *oldJavaMode );
      }
```

Document Revision History

This table describes the changes to *vDSP Library*.

Date	Notes
2005-09-08	Corrected an error in the description of the vDSP_fft2d_zript and vDSP_fft2d_zriptD functions.
2001-01-01	New document describing the vDSP framework.

Index