9978 Granite Point Ct.
Granite Bay, CA 95746
www.codesourcery.com

Using FFTW3 with the VSIPL++ Reference Implementation

Prepared for U.S. Air Force under contract FA8750-05-C-0004.

CODESOURCERY, INC.

1 Introduction

By default, the VSIPL++ reference implementation uses a C-VSIPL implementation to perform FFT operations. This document describes how to use the FFTW3 library to perform FFT operations instead.

Note: these instructions should only be used with the reference implementation. The optimized implementation is already capable of using the FFTW3 library to perform FFT operations when configured with '`--enable-fft=fftw3`' or '`--enable-fft=builtin`'.

2 Instructions

2.1 Prerequisites

First, you will need the following files:

- sourceryvsipl++-1.3.tar.bz2 – the Sourcery VSIPL++ source package.
- 1.3-ri-fftw3.diff – patch for using FFTW3 with the reference implementation.

You will also need a working installation of FFTW3, with libraries and header files installed in your C++ compiler's search paths. The vendor FFTW3 package for systems such as Debian, Ubuntu, and Redhat, meet this criteria.

2.2 Source Tree Setup

1. Untar the source package

```
% tar xfj sourceryvsipl++-1.3.tar.bz2
```

This will create a directory `sourceryvsipl++-1.3` in your current directory. Cd (change directory) into it.

```
% cd sourceryvsipl++-1.3
```

2. Apply the patch

```
% patch –p1 < 1.3-ri.fftw3.diff
```

2.3 Configure

3. Configure the library. Using the following options:

- `--enable-ref-impl`
- `--with-lapack=no`
- `LDFLAGS="-lfftw3 –lfftw3f"`

It may be necessary to specify the location of your C-VSIPL implementation with `--with-cvsip-prefix` if it is not in a default location.

```
% configure --enable-ref-impl \
            --with-lapack=no  \
            LDFLAGS="-lfftw3 –lfftw3f"
```

2.4 Build

4. Build the library with make:

```
% make
```

5. Optionally, install the library:

```
% make install
```

2.5 Test

6. Finally, test the library by building and running the FFT test:

```
% make tests/fft
% tests/fft
```

A return value of 0 indicates success.

3 Appendix: Patch

The following patch modifies Sourcery VSIPL++ 1.3 to use FFTW3 with the reference implementation.

```
diff -x 'autom4te*' -Nur 1.3/configure 1.3-ri-fftw3/configure
--- 1.3/configure      2007-02-12 08:17:03.000000000 -0800
+++ 1.3-ri-fftw3/configure    2007-06-25 14:10:23.213713000 -0700
@@ -12383,6 +12383,7 @@
 mkdir -p lib
 mkdir -p lib/python/site-packages/vsip
 mkdir -p src/vsip/core/cvsip
+mkdir -p src/vsip/core/fftw3_ri
 mkdir -p src/vsip/core/expr
 mkdir -p src/vsip/core/fft
 mkdir -p src/vsip/core/parallel
diff -x 'autom4te*' -Nur 1.3/configure.ac 1.3-ri-fftw3/configure.ac
--- 1.3/configure.ac   2007-02-07 12:15:52.000000000 -0800
+++ 1.3-ri-fftw3/configure.ac 2007-06-24 18:45:04.994048000 -0700
@@ -2468,6 +2468,7 @@
 mkdir -p lib
 mkdir -p lib/python/site-packages/vsip
 mkdir -p src/vsip/core/cvsip
+mkdir -p src/vsip/core/fftw3_ri
 mkdir -p src/vsip/core/expr
 mkdir -p src/vsip/core/fft
 mkdir -p src/vsip/core/parallel
diff -x 'autom4te*' -Nur 1.3/src/vsip/core/fft.hpp 1.3-ri-
fftw3/src/vsip/core/fft.hpp
--- 1.3/src/vsip/core/fft.hpp 2007-02-02 06:01:50.000000000 -0800
+++ 1.3-ri-fftw3/src/vsip/core/fft.hpp       2007-06-24 20:41:38.498854000
-0700
@@ -42,6 +42,7 @@
 # endif
 #endif // VSIP_IMPL_REF_IMPL

+#include <vsip/core/fftw3_ri/fft.hpp>
 #if VSIP_IMPL_CVSIP_FFT
 # include <vsip/core/cvsip/fft.hpp>
 #endif
@@ -190,7 +191,7 @@
     VSIP_THROW((std::bad_alloc))
     : base(dom, scale, false, S, by_value),
 #ifdef VSIP_IMPL_REF_IMPL
-      backend_(cvsip::create<fft::backend<D, I, O, axis, exponent> >
+      backend_(fftw3_ri::create<fft::backend<D, I, O, axis, exponent> >
                (dom, scale, N)),
 #else
       backend_(factory::create(dom, scale)),
@@ -242,7 +243,7 @@
     VSIP_THROW((std::bad_alloc))
     : base(dom, scale, false, S, by_reference),
 #ifdef VSIP_IMPL_REF_IMPL
-      backend_(cvsip::create<fft::backend<D, I, O, axis, exponent> >
+      backend_(fftw3_ri::create<fft::backend<D, I, O, axis, exponent> >
                (dom, scale, N)),
 #else
       backend_(factory::create(dom, scale)),
diff -x 'autom4te*' -Nur 1.3/src/vsip/core/fftw3_ri/fft.cpp 1.3-ri-
fftw3/src/vsip/core/fftw3_ri/fft.cpp
```

```
--- 1.3/src/vsip/core/fftw3_ri/fft.cpp       1969-12-31 16:00:00.000000000
-0800
+++ 1.3-ri-fftw3/src/vsip/core/fftw3_ri/fft.cpp     2007-06-25
14:24:20.135798000 -0700
@@ -0,0 +1,542 @@
+/* Copyright (c) 2007 by CodeSourcery.  All rights reserved. */
+
+/** @file    vsip/core/fftw3_ri/fft.cpp
+    @author  Jules Bergmann
+    @date    2006-06-22
+    @brief   VSIPL++ Library: FFT wrappers and traits to bridge with
+             FFTW3 in the ref-impl.
+*/
+
+/***********************************************************************
+  Included Files
+***********************************************************************/
+
+#include <fftw3.h>
+
+#include <vsip/core/config.hpp>
+#include <vsip/core/fftw3_ri/fft.hpp>
+#include <vsip/core/cvsip/block.hpp>
+#include <vsip/core/cvsip/view.hpp>
+
+extern "C" {
+#include <vsip.h>
+}
+
+
+
+/***********************************************************************
+  Declarations
+***********************************************************************/
+
+namespace vsip
+{
+namespace impl
+{
+namespace fftw3_ri
+{
+
+template <typename T> struct Fftw_traits;
+
+#if VSIP_IMPL_FFTW3_RI_HAVE_FLOAT
+template <>
+struct Fftw_traits<complex<float> >
+{
+  typedef fftwf_plan plan_type;
+
+  // Plan 1-D interleaved CC FFT
+  static plan_type
+  plan(
+    length_type           size,
+    complex<float> const* in,
+    complex<float>*       out,
+    int                   dir,
+    int                   flags)
+  {
+    return fftwf_plan_dft_1d(size,
+                         (fftwf_complex*)in,
+                         (fftwf_complex*)out,
+                         dir, flags);
```

```
+  }
+
+  // Plan 1-D interleaved RC FFT
+  static plan_type
+  plan_rc(
+    length_type      size,
+    float const*     in,
+    complex<float>*  out)
+  {
+    dimension_type const dim = 1;
+    int size_array[dim];
+
+    // for (dimension_type i = 0; i < dim; ++i) size_array[i] =
dom[i].size();
+    size_array[0] = size;
+
+    return fftwf_plan_dft_r2c(dim, size_array,
+                              (float*)in,
+                              (fftwf_complex*)out,
+                              FFTW_PRESERVE_INPUT);
+  }
+
+  // Plan 1-D interleaved CR FFT
+  static plan_type
+  plan_cr(
+    length_type           size,
+    complex<float> const* out,
+    float*                in)
+  {
+    dimension_type const dim = 1;
+    int size_array[dim];
+
+    // for (dimension_type i = 0; i < dim; ++i) size_array[i] =
dom[i].size();
+    size_array[0] = size;
+
+    return fftwf_plan_dft_c2r(dim, size_array,
+                              (fftwf_complex*)in,
+                              (float*)out,
+                              FFTW_PRESERVE_INPUT);
+  }
+
+  // Plan N-D interleaved-complex FFT
+  static plan_type
+  plan_nd(
+    dimension_type        D,
+    int const*            size,
+    complex<float> const* in,
+    complex<float>*       out,
+    int                   dir,
+    int                   flags)
+  {
+    return fftwf_plan_dft(D, size,
+                          (fftwf_complex*)in,
+                          (fftwf_complex*)out,
+                          dir, flags);
+  }
+
+  static void
+  execute(
+    plan_type             p,
+    complex<float> const* in,
+    complex<float>*       out)
```

```
+  {
+    fftwf_execute_dft(p, (fftwf_complex*)in, (fftwf_complex*)out);
+  }
+
+  static void
+  execute_rc(
+    plan_type       p,
+    float const*    in,
+    complex<float>* out)
+  {
+    fftwf_execute_dft_r2c(p, (float*)in, (fftwf_complex*)out);
+  }
+
+  static void
+  execute_cr(
+    plan_type             p,
+    complex<float> const* in,
+    float*                out)
+  {
+    fftwf_execute_dft_c2r(p, (fftwf_complex*)in, (float*)out);
+  }
+
+  static void
+  destroy_plan(plan_type p)
+  {
+    fftwf_destroy_plan(p);
+  }
+};
+#endif
+
+
+
+#if VSIP_IMPL_FFTW3_RI_HAVE_DOUBLE
+template <>
+struct Fftw_traits<complex<double> >
+{
+  typedef fftw_plan plan_type;
+
+  // Plan 1-D interleaved-complex FFT
+  static plan_type
+  plan(
+    length_type           size,
+    complex<double> const* in,
+    complex<double>*       out,
+    int                   dir,
+    int                   flags)
+  {
+    return fftw_plan_dft_1d(size,
+                        (fftw_complex*)in,
+                        (fftw_complex*)out,
+                        dir, flags);
+  }
+
+  static plan_type
+  plan_rc(
+    length_type      size,
+    double const*     in,
+    complex<double>*  out)
+  {
+    dimension_type const dim = 1;
+    int size_array[dim];
+
```

```
+    // for (dimension_type i = 0; i < dim; ++i) size_array[i] =
dom[i].size();
+    size_array[0] = size;
+
+    return fftw_plan_dft_r2c(dim, size_array,
+                             (double*)in,
+                             (fftw_complex*)out,
+                             FFTW_PRESERVE_INPUT);
+  }
+
+  // Plan 1-D interleaved CR FFT
+  static plan_type
+  plan_cr(
+    length_type            size,
+    complex<double> const* out,
+    double*                in)
+  {
+    dimension_type const dim = 1;
+    int size_array[dim];
+
+    // for (dimension_type i = 0; i < dim; ++i) size_array[i] =
dom[i].size();
+    size_array[0] = size;
+
+    return fftw_plan_dft_c2r(dim, size_array,
+                             (fftw_complex*)in,
+                             (double*)out,
+                              FFTW_PRESERVE_INPUT);
+  }
+
+  // Plan N-D interleaved-complex FFT
+  static plan_type
+  plan_nd(
+    dimension_type        D,
+    int const*            size,
+    complex<double> const* in,
+    complex<double>*       out,
+    int                   dir,
+    int                   flags)
+  {
+    return fftw_plan_dft(D, size,
+                         (fftw_complex*)in,
+                         (fftw_complex*)out,
+                         dir, flags);
+  }
+
+  static void
+  execute(
+    plan_type             p,
+    complex<double> const* in,
+    complex<double>*       out)
+  {
+    fftw_execute_dft(p, (fftw_complex*)in, (fftw_complex*)out);
+  }
+
+  static void
+  execute_rc(
+    plan_type       p,
+    double const*    in,
+    complex<double>* out)
+  {
+    fftw_execute_dft_r2c(p, (double*)in, (fftw_complex*)out);
+  }
```

```
+
+  static void
+  execute_cr(
+    plan_type             p,
+    complex<double> const* in,
+    double*                out)
+  {
+    fftw_execute_dft_c2r(p, (fftw_complex*)in, (double*)out);
+  }
+
+  static void
+  destroy_plan(plan_type p)
+  {
+    fftw_destroy_plan(p);
+  }
+};
+#endif
+
+
+
+/*****************************************************************************
+  Fft_impl
+*****************************************************************************/
+
+template <dimension_type D,  // Dimension
+          typename       I,    // In type
+          typename       O,
+          int            A,
+          int            E>
+class Fft_impl;
+
+
+
+/*****************************************************************************
+  Fft_impl: Complex -> Complex
+*****************************************************************************/
+
+template <typename T, int A, int E>
+class Fft_impl<1, std::complex<T>, std::complex<T>, A, E>
+  : public fft::backend<1, std::complex<T>, std::complex<T>, A, E>
+{
+  typedef T rtype;
+  typedef std::complex<rtype>       ctype;
+  typedef std::pair<rtype*, rtype*> ztype;
+  typedef Fftw_traits<std::complex<T> > traits;
+  typedef typename traits::plan_type    plan_type;
+
+public:
+  Fft_impl(Domain<1> const& dom, rtype /*scale*/, unsigned int /*n*/, int
/*h*/)
+    : in_buffer_ (32, dom.size()),
+      out_buffer_(32, dom.size()),
+      plan_ip_   (traits::plan(dom.size(),
+                               in_buffer_.get(), in_buffer_.get(),
+                               E, 0)),
+      plan_op_   (traits::plan(dom.size(),
+                               in_buffer_.get(), out_buffer_.get(),
+                               E, 0))
+  {}
+  ~Fft_impl()
+  {
+    traits::destroy_plan(plan_op_);
+    traits::destroy_plan(plan_ip_);
```

```
+  }
+
+  virtual bool supports_scale() { return false; }
+
+  virtual void in_place(ctype* inout, stride_type stride, length_type
length)
+  {
+    ctype* use_inout;
+    if (stride != 1)
+    {
+      for (index_type i=0; i<length; ++i)
+       in_buffer_.get()[i] = inout[i*stride];
+      use_inout = in_buffer_.get();
+    }
+    else use_inout = inout;
+    traits::execute(plan_ip_, use_inout, use_inout);
+    if (stride != 1)
+    {
+      for (index_type i=0; i<length; ++i)
+       inout[i*stride] = in_buffer_.get()[i];
+    }
+  }
+
+  virtual void in_place(ztype inout, stride_type stride, length_type
length)
+  {
+    for (index_type i=0; i<length; ++i)
+      in_buffer_.get()[i] = complex<T>(inout.first[i*stride],
+                                       inout.second[i*stride]);
+    traits::execute(plan_ip_, in_buffer_.get(), in_buffer_.get());
+    for (index_type i=0; i<length; ++i)
+    {
+      inout.first[i*stride]  = in_buffer_.get()[i].real();
+      inout.second[i*stride] = in_buffer_.get()[i].imag();
+    }
+  }
+
+  virtual void by_reference(ctype *in, stride_type in_stride,
+                            ctype *out, stride_type out_stride,
+                            length_type length)
+  {
+    if (in_stride != 1)
+    {
+      for (index_type i=0; i<length; ++i)
+       in_buffer_.get()[i] = in[i*in_stride];
+      in = in_buffer_.get();
+    }
+    ctype *use_out = (out_stride == 1) ? out : out_buffer_.get();
+    traits::execute(plan_op_, in, use_out);
+    if (out_stride != 1)
+    {
+      for (index_type i=0; i<length; ++i)
+       out[i*out_stride] = out_buffer_.get()[i];
+    }
+  }
+
+  virtual void by_reference(ztype in, stride_type in_stride,
+                            ztype out, stride_type out_stride,
+                            length_type length)
+  {
+    for (index_type i=0; i<length; ++i)
+      in_buffer_.get()[i] = complex<T>(in.first[i*in_stride],
+                                       in.second[i*in_stride]);
```

```
+    traits::execute(plan_op_, in_buffer_.get(), out_buffer_.get());
+    for (index_type i=0; i<length; ++i)
+    {
+      out.first[i*out_stride]  = out_buffer_.get()[i].real();
+      out.second[i*out_stride] = out_buffer_.get()[i].imag();
+    }
+  }
+
+private:
+  aligned_array<complex<T> > in_buffer_;
+  aligned_array<complex<T> > out_buffer_;
+  plan_type                  plan_ip_;
+  plan_type                  plan_op_;
+};
+
+
+
+/*************************************************************************
+  Fft_impl: Real -> Complex
+*************************************************************************/
+
+template <typename T, int A>
+class Fft_impl<1, T, std::complex<T>, A, -1>
+  : public fft::backend<1, T, std::complex<T>, A, -1>
+{
+  typedef T rtype;
+  typedef std::complex<rtype>       ctype;
+  typedef std::pair<rtype*, rtype*> ztype;
+  typedef Fftw_traits<std::complex<T> > traits;
+  typedef typename traits::plan_type    plan_type;
+
+public:
+  Fft_impl(Domain<1> const& dom, rtype /*scale*/, unsigned int /*n*/, int
/*h*/)
+    : in_buffer_ (32, dom.size()),
+      out_buffer_(32, dom.size()),
+      plan_      (traits::plan_rc(dom.size(),
+                                  in_buffer_.get(), out_buffer_.get()))
+  {}
+  ~Fft_impl()
+  {
+    traits::destroy_plan(plan_);
+  }
+
+  virtual bool supports_scale() { return false; }
+
+  virtual void by_reference(rtype *in, stride_type in_stride,
+                            ctype *out, stride_type out_stride,
+                            length_type length)
+  {
+    if (in_stride != 1)
+    {
+      for (index_type i=0; i<length; ++i)
+       in_buffer_.get()[i] = in[i*in_stride];
+      in = in_buffer_.get();
+    }
+    ctype *use_out = (out_stride == 1) ? out : out_buffer_.get();
+    traits::execute_rc(plan_, in, use_out);
+    if (out_stride != 1)
+    {
+      for (index_type i=0; i<length/2+1; ++i)
+       out[i*out_stride] = out_buffer_.get()[i];
+    }
```

```
+  }
+
+  virtual void by_reference(rtype *in, stride_type in_stride,
+                            ztype out, stride_type out_stride,
+                            length_type length)
+  {
+    for (index_type i=0; i<length; ++i)
+      in_buffer_.get()[i] = in[i*in_stride];
+    traits::execute_rc(plan_, in_buffer_.get(), out_buffer_.get());
+    for (index_type i=0; i<length/2+1; ++i)
+    {
+      out.first[i*out_stride]  = out_buffer_.get()[i].real();
+      out.second[i*out_stride] = out_buffer_.get()[i].imag();
+    }
+  }
+
+private:
+  aligned_array<T>           in_buffer_;
+  aligned_array<complex<T> > out_buffer_;
+  plan_type                  plan_;
+};
+
+
+
+template <typename T, int A>
+class Fft_impl<1, std::complex<T>, T, A, 1>
+  : public fft::backend<1, std::complex<T>, T, A, 1>
+{
+  typedef T                              rtype;
+  typedef std::complex<rtype>            ctype;
+  typedef std::pair<rtype*, rtype*>      ztype;
+  typedef Fftw_traits<std::complex<T> > traits;
+  typedef typename traits::plan_type     plan_type;
+
+public:
+  Fft_impl(Domain<1> const& dom, rtype /*scale*/, unsigned int /*n*/, int
/*h*/)
+    : in_buffer_ (32, dom.size()),
+      out_buffer_(32, dom.size()),
+      plan_      (traits::plan_cr(dom.size(),
+                                  in_buffer_.get(), out_buffer_.get()))
+  {}
+  ~Fft_impl()
+  {
+    traits::destroy_plan(plan_);
+  }
+
+  virtual bool supports_scale() { return false; }
+
+  virtual void by_reference(ctype *in, stride_type in_stride,
+                            rtype *out, stride_type out_stride,
+                            length_type length)
+  {
+    if (in_stride != 1)
+    {
+      for (index_type i=0; i<length/2+1; ++i)
+       in_buffer_.get()[i] = in[i*in_stride];
+      in = in_buffer_.get();
+    }
+    rtype* use_out = (out_stride == 1) ? out : out_buffer_.get();
+    traits::execute_cr(plan_, in, use_out);
+    if (out_stride != 1)
+    {
```

```
+      for (index_type i=0; i<length; ++i)
+       out[i*out_stride]  = out_buffer_.get()[i];
+    }
+  }
+
+  virtual void by_reference(ztype in, stride_type in_stride,
+                            rtype *out, stride_type out_stride,
+                            length_type length)
+  {
+    for (index_type i=0; i<length/2+1; ++i)
+      in_buffer_.get()[i] = complex<T>(in.first[i*in_stride],
+                                       in.second[i*in_stride]);
+    traits::execute_cr(plan_, in_buffer_.get(), out_buffer_.get());
+    for (index_type i=0; i<length; ++i)
+      out[i*out_stride]  = out_buffer_.get()[i];
+  }
+
+private:
+  aligned_array<complex<T> > in_buffer_;
+  aligned_array<T>           out_buffer_;
+  plan_type                  plan_;
+};
+
+
+
+
+#define VSIPL_IMPL_PROVIDE(D, I, O, A, E)            \
+template <>                                            \
+std::auto_ptr<fft::backend<D, I, O, A, E> >          \
+create(Domain<D> const &dom, Scalar_of<I>::type scale,   \
+       unsigned int n)                                 \
+{                                                      \
+  return std::auto_ptr<fft::backend<D, I, O, A, E> >     \
+    (new Fft_impl<D, I, O, A, E>(dom, scale, n, 0));     \
+}
+
+#if defined VSIP_IMPL_FFT_USE_FLOAT && VSIP_IMPL_FFTW3_RI_HAVE_FLOAT
+VSIPL_IMPL_PROVIDE(1, std::complex<float>, std::complex<float>, 0, -1)
+VSIPL_IMPL_PROVIDE(1, std::complex<float>, std::complex<float>, 0, 1)
+VSIPL_IMPL_PROVIDE(1, float, std::complex<float>, 0, -1)
+VSIPL_IMPL_PROVIDE(1, std::complex<float>, float, 0, 1)
+#endif
+#if defined VSIP_IMPL_FFT_USE_DOUBLE && VSIP_IMPL_FFTW3_RI_HAVE_DOUBLE
+VSIPL_IMPL_PROVIDE(1, std::complex<double>, std::complex<double>, 0, -1)
+VSIPL_IMPL_PROVIDE(1, std::complex<double>, std::complex<double>, 0, 1)
+VSIPL_IMPL_PROVIDE(1, double, std::complex<double>, 0, -1)
+VSIPL_IMPL_PROVIDE(1, std::complex<double>, double, 0, 1)
+#endif
+#undef VSIPL_IMPL_PROVIDE
+
+} // namespace vsip::impl::fftw3_ri
+} // namespace vsip::impl
+} // namespace vsip
diff -x 'autom4te*' -Nur 1.3/src/vsip/core/fftw3_ri/fft.hpp 1.3-ri-
fftw3/src/vsip/core/fftw3_ri/fft.hpp
--- 1.3/src/vsip/core/fftw3_ri/fft.hpp       1969-12-31 16:00:00.000000000
-0800
+++ 1.3-ri-fftw3/src/vsip/core/fftw3_ri/fft.hpp     2007-06-24
19:48:23.184755000 -0700
@@ -0,0 +1,138 @@
+/* Copyright (c) 2007 by CodeSourcery, LLC.  All rights reserved. */
+
+/** @file    vsip/core/fftw3_ri/fft.hpp
```

```
+    @author  Jules Bergmann
+    @date    2006-10-16
+    @brief   VSIPL++ Library: FFT wrappers and traits to bridge with
+             FFTW3 in the ref-impl.
+*/
+
+#ifndef VSIP_CORE_FFTW3_RI_FFT_HPP
+#define VSIP_CORE_FFTW3_RI_FFT_HPP
+
+#define VSIP_IMPL_FFTW3_RI_HAVE_FLOAT 1
+#define VSIP_IMPL_FFTW3_RI_HAVE_DOUBLE 1
+
+/***********************************************************************
+  Included Files
+***********************************************************************/
+
+#include <vsip/core/config.hpp>
+#include <vsip/support.hpp>
+#include <vsip/domain.hpp>
+#include <vsip/core/fft/factory.hpp>
+#include <vsip/core/fft/util.hpp>
+
+/***********************************************************************
+  Declarations
+***********************************************************************/
+
+namespace vsip
+{
+namespace impl
+{
+namespace fftw3_ri
+{
+
+template <typename I, dimension_type D, typename S>
+std::auto_ptr<I>
+create(Domain<D> const &dom, S scale, unsigned int);
+
+template <>
+std::auto_ptr<fft::backend<1, float, std::complex<float>, 0, -1> >
+create(Domain<1> const &, float, unsigned int);
+template <>
+std::auto_ptr<fft::backend<1, std::complex<float>, float, 0, 1> >
+create(Domain<1> const &, float, unsigned int);
+template <>
+std::auto_ptr<fft::backend<1, std::complex<float>, std::complex<float>,
0, -1> >
+create(Domain<1> const &, float, unsigned int);
+template <>
+std::auto_ptr<fft::backend<1, std::complex<float>, std::complex<float>,
0, 1> >
+create(Domain<1> const &, float, unsigned int);
+template <>
+std::auto_ptr<fft::backend<1, double, std::complex<double>, 0, -1> >
+create(Domain<1> const &, double, unsigned int);
+template <>
+std::auto_ptr<fft::backend<1, std::complex<double>, double, 0, 1> >
+create(Domain<1> const &, double, unsigned int);
+template <>
+std::auto_ptr<fft::backend<1, std::complex<double>, std::complex<double>,
0, -1> >
+create(Domain<1> const &, double, unsigned int);
+template <>
```

```
+std::auto_ptr<fft::backend<1, std::complex<double>, std::complex<double>,
0, 1> >
+create(Domain<1> const &, double, unsigned int);
+} // namespace vsip::impl::fftw3_ri
+
+
+namespace fft
+{
+struct Fftw3_ri_tag;
+
+template <typename             I,
+          typename             O,
+          int                  S,
+          return_mechanism_type R,
+          unsigned             N> // Number of Times
+struct evaluator<1, I, O, S, R, N, Fftw3_ri_tag>
+{
+  static bool const has_float =
+#if VSIP_IMPL_FFTW3_RI_HAVE_FLOAT
+    true
+#else
+    false
+#endif
+    ;
+  static bool const has_double =
+#if VSIP_IMPL_FFTW3_RI_HAVE_DOUBLE
+    true
+#else
+    false
+#endif
+    ;
+  static bool const ct_valid = (has_float &&
+                                Type_equal<typename Scalar_of<I>::type,
+                                           float>::value) ||
+                               (has_double &&
+                                Type_equal<typename Scalar_of<I>::type,
+                                           double>::value);
+  static bool rt_valid(Domain<1> const &/*dom*/) { return true;}
+  static std::auto_ptr<backend<1, I, O,
+                               axis<I, O, S>::value,
+                               exponent<I, O, S>::value> >
+  create(Domain<1> const &dom, typename Scalar_of<I>::type scale)
+  {
+    return fftw3_ri::create<backend<1, I, O,
+                                    axis<I, O, S>::value,
+                                    exponent<I, O, S>::value> >
+      (dom, scale, N);
+  }
+};
+
+} // namespace vsip::impl::fft
+
+namespace fftm
+{
+template <typename I,
+          typename O,
+          int A,
+          int E,
+          return_mechanism_type R,
+          unsigned N>
+struct evaluator<I, O, A, E, R, N, fft::Fftw3_ri_tag>
+{
+  static bool const ct_valid = !Type_equal<typename Scalar_of<I>::type,
```

```
+                                          long double>::value;
+  static bool rt_valid(Domain<2> const& /*dom*/) { return true;}
+  static std::auto_ptr<fft::fftm<I, O, A, E> >
+  create(Domain<2> const &dom, typename Scalar_of<I>::type scale)
+  {
+    return fftw3_ri::create<fft::fftm<I, O, A, E> >(dom, scale, N);
+  }
+};
+
+} // namespace vsip::impl::fftm
+} // namespace vsip::impl
+} // namespace vsip
+
+#endif
diff -x 'autom4te*' -Nur 1.3/src/vsip/GNUmakefile.inc.in 1.3-ri-
fftw3/src/vsip/GNUmakefile.inc.in
--- 1.3/src/vsip/GNUmakefile.inc.in   2007-02-02 06:01:50.000000000 -0800
+++ 1.3-ri-fftw3/src/vsip/GNUmakefile.inc.in 2007-06-24 18:42:39.907861000
-0700
@@ -22,6 +22,7 @@
 ifdef VSIP_IMPL_CVSIP_FFT
 src_vsip_cxx_sources += $(srcdir)/src/vsip/core/cvsip/fft.cpp
 endif
+src_vsip_cxx_sources += $(srcdir)/src/vsip/core/fftw3_ri/fft.cpp

 ifndef VSIP_IMPL_REF_IMPL
 src_vsip_cxx_sources += $(wildcard $(srcdir)/src/vsip/opt/*.cpp)
diff -x 'autom4te*' -Nur 1.3/tests/GNUmakefile.inc.in 1.3-ri-
fftw3/tests/GNUmakefile.inc.in
--- 1.3/tests/GNUmakefile.inc.in      2006-10-27 15:36:27.000000000 -0700
+++ 1.3-ri-fftw3/tests/GNUmakefile.inc.in    2007-06-24 18:42:09.619878000
-0700
@@ -27,7 +27,8 @@
 tests_run_ident :=-a run_id=$(tests_run_id)
 endif

-tests_cxx_sources := $(wildcard $(srcdir)/tests/*.cpp)
+tests_cxx_sources := $(wildcard $(srcdir)/tests/*.cpp) \
+                     $(wildcard $(srcdir)/tests/ref-impl/*.cpp)

 tests_cxx_exes := \
       $(patsubst $(srcdir)/%.cpp, %$(EXEEXT), $(tests_cxx_sources))
```