An In Depth Look at VOLK

The Vector-Optimized Library of Kernels

Nathan West

U.S. Naval Research Laboratory

26 August 2015

A brief look at VOLK organization

- VOLK is a sub-project of GNU Radio
- http://libvolk.org
- Sub-maintainer: Nathan West (that's me!)
- Code © Free Software Foundation licensed under GPLv3
- Submit issues and pull requests through http://github.com/gnuradio/volk

What's a VOLK?

- VOLK: Vector-Optimized Library of Kernels
- Collaboration, compatibility, and speed (in that order)
- A library of functions optimized for various CPU architectures– usually this means SIMD
- An API (and dispatcher) for calling the fastest code for your machine
- A build system for code targetting specific instruction sets/architectures

Outline

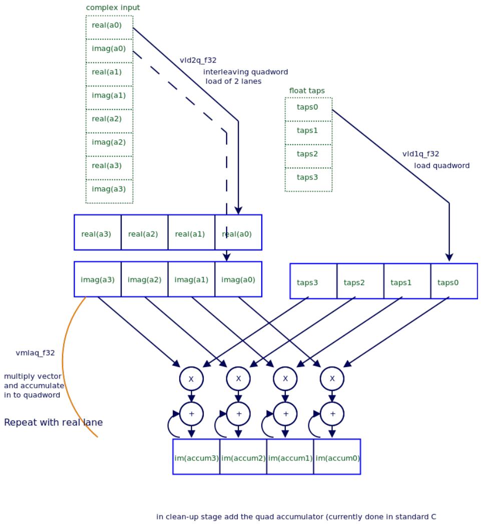
We will cover the priorities of VOLK in reverse order:

1. Speed
2. Compatibility
3. Collaboration

Goals:

- Become fluent in VOLK
- Know how to add a protokernel
- Know how to add a kernel
- Know how to make VOLK work for your project
- Know where to look to support new hardware

Motivation: a NEON register view of 32fc_32f_dot_prod_32fc


complex input
real(a0)
imag(a0)
real(a1)
imag(a1)
real(a2)
imag(a2)
real(a3)
imag(a3)
vld2q_f32
interleaving quadword load of 2 lanes
float taps
taps0
taps1
taps2
taps3
vld1q_f32
load quadword
real(a3) real(a2) real(a1) real(a0)
imag(a3) imag(a2) imag(a1) imag(a0)
taps3 taps2 taps1 taps0
vmlaq_f32
multiply vector and accumulate in to quadword
Repeat with real lane
x
+
im(accum3) im(accum2) im(accum1) im(accum0)
in clean-up stage add the quad accumulator (currently done in standard C

Motivation: Disassembled C Complex Dot Product excerpt

Code can be inefficient even when hardware features are used

```
.loop1
    mov r11, r8                   @ copy new address - aVec
    mov r10, r4                   @ copy new address - bVec
    vld4.32 {d16,d18,d20,d22}, [r11]!
    add r9, r9, #1                @ number += 1
    cmp r9, r12                   @ number < half_points?
    add r8, r8, #64               @ calculate next aVec address
    add r4, r4, #64               @ calculate next bVec address
    vld4.32 {d24,d26,d28,d30}, [r10]!
    vld4.32 {d17,d19,d21,d23}, [r11]
    vld4.32 {d25,d27,d29,d31}, [r10]

    <snipped NEON dot product ops>

    bcc .loop1                    @ repeat half_points times
```

Problem: Architecture-optimized code is not portable

- Historical solution is #ifdef fences
- Meta-compilers exist (ORC, Spiral, PEACHPy)

This brings us to VOLK's second concern: compatibility

Compatibility

"All problems in computer science can be solved by another layer of indirection." –David Wheeler

VOLK abstractions:

- arch (gen/archs.xml)
- machines (gen/machines.xml)

Compatibility: the arch abstraction

Architectures are hardware features, described in an xml file with

- Arch name
- Compiler flags
- Alignment restrictions
- check name (defined in tmpl/volk_cpu.tmpl.c)

Compatibility: the machine abstraction

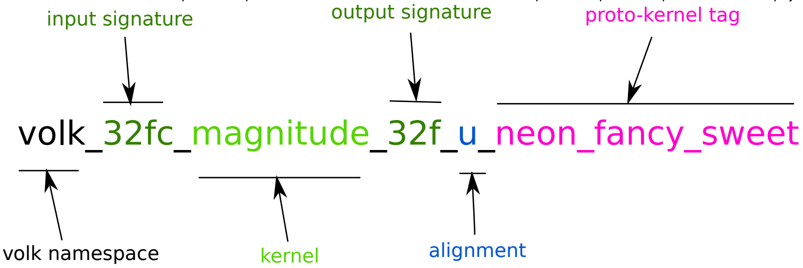
Machines are a collection of architectures that are found together in real hardware.

Example ("neon"):

- generic
- neon
- softfp OR hardfp
- ORC is optional

Compatibility: kernels and protokernels

Kernel: the body of a vectorized loop (kernels/volk/*.h)
Proto-kernel: a specific implementation of a kernel (functions in kernels/volk/*.h or .S files in kernels/volk/asm/<arch>/)



API: from your application call the dispatcher
`volk_32fc_magnitude_32f(...)`

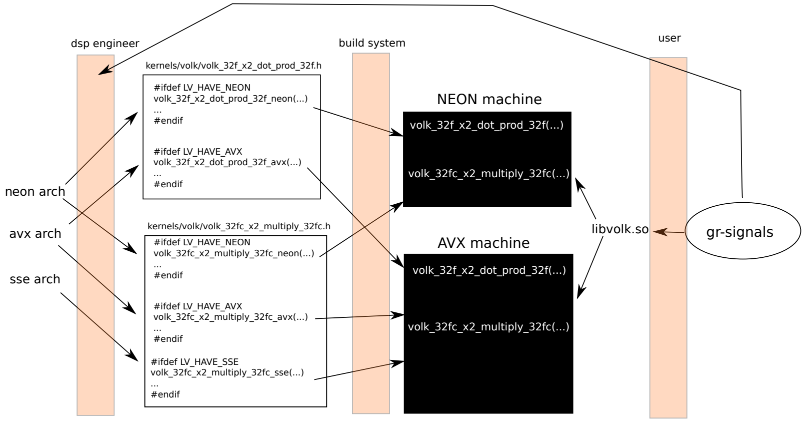
From: Tom Rondeau
Subject: Re: [Discuss-gnuradio] Talking about DSP and SDR [Was: On tunnel.py]
Date: Sat, 30 Mar 2013 15:09:09 -0400

```
I love the idea of taking those tutorials you pointed to and making
GNU Radio applications that showcase them. I suggested a similar
concept of using the "DSP Tips and Tricks" section of IEEE Sig. Proc.
Magazine a while ago:
http://www.trondeau.com/blog/2011/5/16/dsp-tips-and-tricks.html. They
are small, 2-page papers that show off some small DSP tip or trick (as
the name would imply) that is simple to express and therefore work up
in GNU Radio.

Students learning comms theory and DSP could benefit a lot from either
developing these or having them available for study.

Now the question is: who wants to start working on this concept?

I'd suggest that Brian start formatting the page based on the list he
put above, but I think we might be better served by having someone
come up with a full page for something, like the OFDM model you
mentioned, so we can get a feel for how this will go and how the pages
will eventually look.

Tom
```

Compatibility: the profiler and dispatcher

- `volk_profile` will run and time all proto-kernels for your machine
- Fastest proto-kernels per kernel are stored in `$HOME/.volk/volk_config`
- At run-time dispatcher loads volk_config and runs what volk_profile reported as fastest (lib/volk_rank_archs.c)
- Dispatcher handles selecting alignment and valid proto-kernels

Compatibility: putting it all together



dsp engineer
build system
user
kernels/volk/volk_32f_x2_dot_prod_32f.h
#ifdef LV_HAVE_NEON
volk_32f_x2_dot_prod_32f_neon(...)
...
#endif
#ifdef LV_HAVE_AVX
volk_32f_x2_dot_prod_32f_avx(...)
...
#endif
kernels/volk/volk_32fc_x2_multiply_32fc.h
#ifdef LV_HAVE_NEON
volk_32fc_x2_multiply_32fc_neon(...)
...
#endif
#ifdef LV_HAVE_AVX
volk_32fc_x2_multiply_32fc_avx(...)
...
#endif
#ifdef LV_HAVE_SSE
volk_32fc_x2_multiply_32fc_sse(...)
...
#endif
neon arch
avx arch
sse arch
NEON machine
volk_32f_x2_dot_prod_32f(...)
volk_32fc_x2_multiply_32fc(...)
AVX machine
volk_32f_x2_dot_prod_32f(...)
volk_32fc_x2_multiply_32fc(...)
libvolk.so
gr-signals

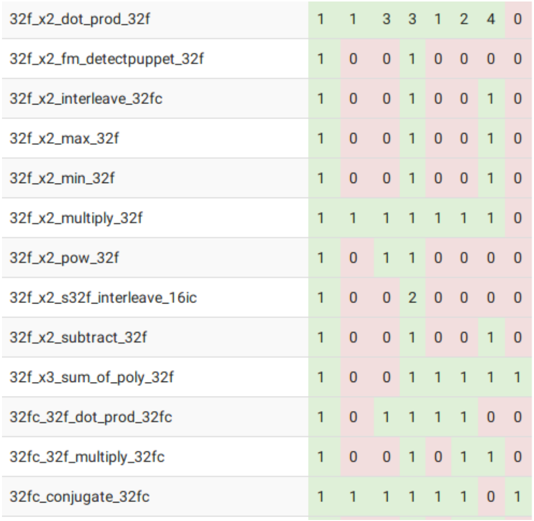
A final note on compatibility: QA and Puppets

VOLK QA reads function signatures to generate buffers and random input data

- Arch-specific proto-kernels are compared to generic for correctness
- If VOLK QA cannot generate buffers for your signature you need a puppet
- Puppets wrap your kernel in a VOLK QA-friendly form. See the rotatorpuppet for examples

Collaboration

VOLK is a repository for efficient code that people actually wrote (not the code we wish they wrote)



32f_x2_dot_prod_32f	1	1	3	3	1	2	4	0
32f_x2_fm_detectpuppet_32f	1	0	0	1	0	0	0	0
32f_x2_interleave_32fc	1	0	0	1	0	0	1	0
32f_x2_max_32f	1	0	0	1	0	0	1	0
32f_x2_min_32f	1	0	0	1	0	0	1	0
32f_x2_multiply_32f	1	1	1	1	1	1	1	0
32f_x2_pow_32f	1	0	1	1	0	0	0	0
32f_x2_s32f_interleave_16ic	1	0	0	2	0	0	0	0
32f_x2_subtract_32f	1	0	0	1	0	0	1	0
32f_x3_sum_of_poly_32f	1	0	0	1	1	1	1	1
32fc_32f_dot_prod_32fc	1	0	1	1	1	1	0	0
32fc_32f_multiply_32fc	1	0	0	1	0	1	1	0
32fc_conjugate_32fc	1	1	1	1	1	1	0	1

Collaboration: VOLK for you

- volk_modtool: like gr_modtool, but for VOLK
- experiment with new archs
- Add kernels (new header files)
- Hopefully contribute upstream

Parting Thoughts

- VOLK is a necessity caused by vectorization being very difficult for compilers
- DSP designers write code specific to hardware and wrap it inside VOLK
- Users call VOLK kernels when they are available
- After profiling your code, create a VOLK kernel where it is needed
- Contribute VOLK kernels upstream for collaboration

Working Group

Thursday at 3:00 PM

- Missing kernels/proto-kernels
- Arch-specific helper sub-libraries
- Build issues
- General feedback