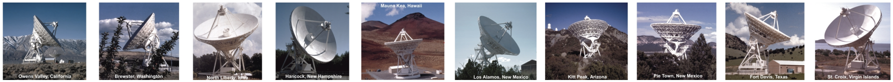
DiFX Performance Primitives (difxpp)

W. Brisken

National Radio Astronomy Observatory (Socorro, NM)

Max Planck Institüt für Radioastronomy (Bonn, Germany)

2015 Nov 17



Intel Integrated Performance Primitives

The good

* Wide variety of vectorized routines
* Optimized for many generations of Intel CPUs
* Generally a good match for DiFX
* Has served DiFX fairly well over a decade and 4 IPP versions

The missed opportunities

* Non-trivial inner loops handled in multiple passes
* Array granularity assumptions not possible to convey
* Some functionality just not there

The bad

* Intel IPP costs increasingly more
* Distribution not simple to install on all OSes and clusters
* Distribution directory structure constantly changing

difxpp

Goals

* Abstract existing vector routines used by DiFX (and possibly SFXC)
  * `architecture.h` disappears
* Provide clean autotool-based build and discovery mechanisms
* Implement generic code realizations (mostly done already by Dodson & others)
* Provide some higher-level routines specific to correlator needs
* Provide thin wrappers for existing libraries (e.g., Intel IPP)

Inspired by . . .

* Chris Phillips's talk at last year's meeting (`http://www.ira.inaf.it/difx-2014/presentations/DIFX_2014-SIMD.pdf`)

Some possbile architectures to be supported

* Generic (with FFTW providing FFT/DFT code)
* Thin Intel IPP wrapper
* AMD equivalents of Intel IPP
* SSE4 (128-bit vector registers)
* AVX (256-bit vector registers)
* AVX-512 (512-bit vector registers w/ FMA)
* CG (GPUs)

Basic parts

* Function to select architecture (default is generic)
    - Called functions will actually be function pointers that are set throug this setup function
    - Note: support for all compiler- and library-supported architectures is to be generated at compile time
* Separate memory allocation and deallocation functions per architecture
    - Needed for alignment purposed
    - It is possible that arrays will always be allocated a bit longer than need be to allow more loop unrolling and hence minimize cost of bounds checking
* Separate functions for each architecture for each routine
* It is possible for multiple architectures to share implementations for some routines

Suggested development path

1. First write complete set of Generic functions
2. Then add complete set of IPP wrappers
3. At this point difxpp could already become provider for DiFX
4. Then add stubs for new architecture that either points to IPP or Generic function depending on compile-time availability and appropriateness
5. Replace stubs with custom code
6. Develop high level functions as aggregates of low level functions
7. Develop as appropriate compound low-level functions

Status

* Overall structure largely implemented
* Generic code and Intel IPP mostly complete
* Most SSE4 simple vector functions in place
* Some AVX code in place
* Unit testing: benchmarks and correctness-tests of many of the vector functions
* Not yet in DiFX SVN (could be though)
* Looking anyone interested to take lead on development!