Cray Programming Environment Workshop

Luiz DeRose John Levesque Heidi Poxon

Cray PE Workshop - Agenda

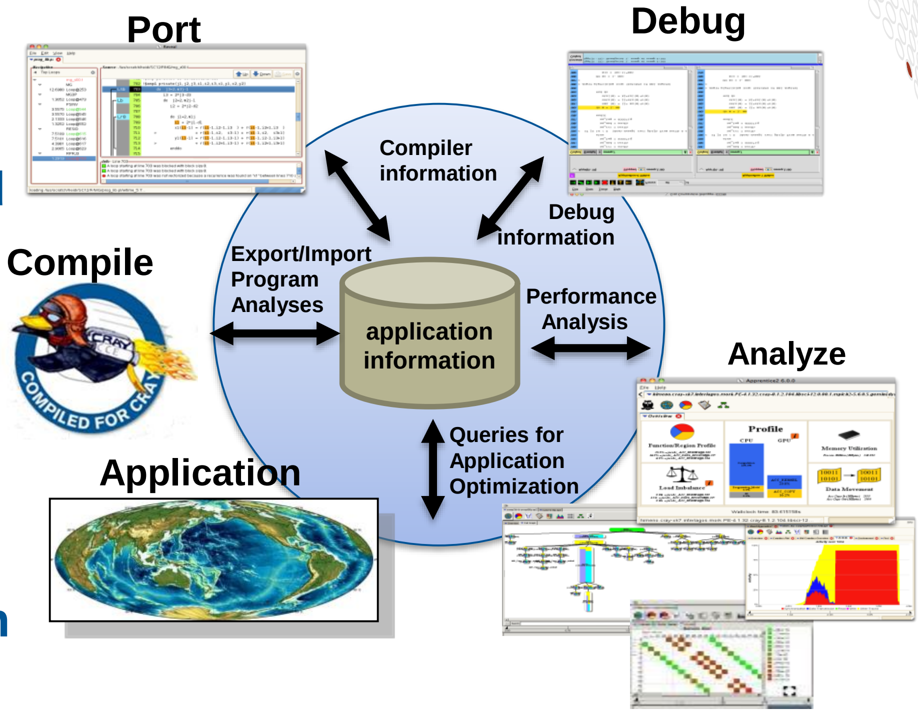
09:00 – 09:15	introduction
09:15 – 10:45	Applying a “Whack-a-mole” Method using Cray’s perftools to identify the Moles
10:45 – 11:00	Break
11:00 – 12:00	Continue with Applying a “Whack-a-mole” Method using Cray’s perftools to identify the Moles
12:00 – 13:00	Lunch
13:00 – 13:30	What is new in PE
13:30 – 14:30	Perftools tips and tricks, data interpretation
14:30 – 14:45	Break
14:45 – 15:15	Tips when using Cray MPI
15:15 – 15:45	Cray PE DL Scalability Plugin
15:45 – 16:00	Wrap-up & Questions

What is new in the Cray Programming Environment

Luiz DeRose
Sr. Principal Engineer
Programming Environments Director

The Cray Programming Environment Mission

- Provide **scalable performance, portability, and programmability** on homogeneous and heterogeneous Cray systems
- Provide the best environment to develop, debug, analyze, and optimize applications for **production supercomputing** with **tightly coupled compilers, libraries, and tools**
  - Address issues of scale and complexity of HPC systems
  - Intuitive behavior and best performance with the least amount of effort
  - Target **ease of use** with extended **functionality** and increased **automation**
  - Close **interaction with users**



COMPUTE | STORE | ANALYZE

Performance at Scale

- **Drive maximum computing performance** **while focusing on programmability and portability**
  - **Close the gap** between observed performance and achievable performance
  - **Maximize the cycles to the application**
  - **Address issues of scale** and complexity of HPC systems
  - A **performance portable programming environment**
    - Same look and feel, independently of processor architecture

Cray Performance Tools profile production applications with over 99,000 ranks

Cray MPI runs with 2,001,150 ranks

Programmability Focused Environment

- **GNU Modules simplify build environment**
  - Complexity of compile and link lines (-h –I –l –L) reduced
- **Multiple product versions, compilers, and compiler versions available on system at the same time offers more flexibility and convenience**
- **Product agnostic drivers (cc, CC, ftn) are used to compile for supported Programming Environments**
  - Customer-integrated and Cray libraries share the same driver interface
- **Support available to plug 3rd party software into Cray software environment (craypkg-gen)**

Cray Developer Environment on XC Systems

Programming Languages
- Fortran
- C
- C++
- Chapel
- Python
- R

Programming Models
Distributed Memory
- Cray MPI
- SHMEM

Shared Memory / GPU
- OpenMP
  OpenACC
- Global Arrays

PGAS & Global View
- UPC
  Fortran coarrays
  Coarray C++
  Chapel

Programming Environments
PrgEnv-
- Cray Compiling Environment
  PrgEnv-cray
- GNU
  PrgEnv-gnu
- 3rd Party compilers
  PrgEnv-xxx

Libraries
Scientific Libraries
- LAPACK
- ScaLAPACK
- BLAS
- Iterative Refinement Toolkit
- FFTW

I/O Libraries
- NetCDF
- HDF5

Development Tools
Environment setup
- Modules

Debuggers
- TotalView
- DDT
- gdb4hpc

Debugging Support
- Abnormal Termination Processing (ATP)
- STAT

Development Tools
Performance Analysis
- CrayPAT
- Cray Apprentice^2

Porting
- Reveal
- CCDB

ML Frameworks
- Cray PE ML Plugin

Cray Developed
Cray added value to 3rd party
3rd party packaging
Licensed ISV SW

Using the Compiler

- **Cray Systems come with compiler wrappers to simplify building parallel applications**
  - Fortran Compiler: **ftn**
  - C Compiler: **cc**
  - C++ Compiler: **CC**

- **Using these wrappers ensures that your code is built for the compute nodes and linked against important libraries**
  - Cray MPT (MPI, Shmem, etc.)
  - Cray LibSci (BLAS, LAPACK, etc.)
  - …

- **Do not call the Cray compilers directly**

- **Cray Compiler wrappers try to hide the complexities of using the proper header files and libraries**

Compiler man Pages

- **The cc(1), CC(1), and ftn(1) man pages contain information about the compiler driver commands**

- **The *craycc(1), crayCC(1), and crayftn(1) man pages contain descriptions of the Cray compiler command options***

- **To verify that you are using the correct version of a compiler, use:**
  - -V option on a cc, CC, or ftn command with CCE

The Cray Compiling Environment (CCE)

- Cray technology **designed for real scientific applications**, not just for benchmarks
- Fully integrated **heterogeneous optimization** capability
- Focus on standards compliance for **application portability** and **investment protection**

C++ 14 | Fortran 2008 | OpenMP 4.5

C11 | UPC 1.3

Some Cray Compiling Optimization Basics

- **Start with default options, then add options as desired/needed for performance tuning or optional features**
  - Optimization: **-O2** is the default and you should usually use this
  - It's the equivalent of most other compilers –O3 or –fast
  - It is also our most thoroughly tested configuration

- **Use the restrict keyword on all of a function's pointer parameters, provided that they do not alias**

- **Limit functions to a reasonable size**
  - **Thousands of lines of code in a single function will drive up compile time and memory usage**

Recommended CCE Compilation Options

- **Using –O3,fp3 (or –O3 –hfp3, or some variation)**
  - -O3 only gives you slightly more than –O2 **(but more compilation time)**
  - We also test this thoroughly
    - Notice that higher numbers are not always correlated with better performance
  - **-hfp3 gives you a lot more floating point optimization, esp. 32-bit**

- **With C++:**
  - **Use -hipa4**
    - This is now the default in CCE 8.7
  - **Use the predefined complex type** rather than the standard template implementation

- **Avoid using –h aggress and –h ipa5**
  - Few codes actually see a performance benefit (these options are available for rare cases)

- **Optimizing for compile time rather than execution time**
  - Compile time can sometimes be improved by disabling certain optimizations
    - Some common things to try: -hnodwarf, -hipa0, -hunroll0
    - **-h dev**elop reduces compile time at the expense of optimization, by omitting optimizations that are known to increase compile time. This option is intended to be used when a program is under development and being recompiled frequently

Helpful Directives

- See ‘man intro_directives’ for a summary
  - Many directives also have their own man page
- *Use the “optimize”* directive to apply an optimization to a function
  - Overrides the command line for that function
    - See ‘man optimize’.
- Use the *“safe_address”* directive when possible for loops to improve performance
  - See 'man safe_address'

Tips and Hints on using CCE - Vectorization

- **Compiler options:**
  - -h cpu={x86-skylake,mic-knl,...}
  - -h preferred_vector_width={64,128,256,512}
    - For Xeon Skylake, using AVX512VL at 256 bits can sometimes be faster than the full 512 bit vector width
  - -h nofp_trap (the default) allows the compiler to optimize more aggressively

- **Directives:**
  - OpenMP SIMD is a portable way to identify a loop nest for vectorization
  - #pragma concurrent. See ‘man concurrent’.
  - #pragma ivdep. See ‘man ivdep’.

- **Other tips:**
  - **CCE is not limited to vectorizing innermost loops**
    - **Entire loop nests are candidates for vectorization**
  - Based on target hardware characteristics, CCE may decide to leave a loop as scalar if that is expected to be faster than the vectorized counterpart
    - This can be overridden by #pragma prefervector or #pragma omp simd

OpenMP

- **On CCE OpenMP is ON by default**
  - Optimizations controlled by **–hthread#**

- **Autothreading is NOT on by default;**
  - -hautothread to turn on
  - Modernized version of Cray X1 streaming capability
  - **Interacts with OpenMP directives**

- **If you do not want to use OpenMP and have OMP directives in the code, make sure to shut off OpenMP at compile time**
  - **To shut off** use **–hthread0** or **–xomp** or **–hnoomp**

Communications From CCE

- **“Positive” and “negative” optimization messages**
  - “**Positive**” messages report
    - Key optimizations that were performed
    - Various optimization concerns (like overuse of registers)
    - Possible functional issues
      - like potential numeric differences and use-before-definition problems)
    - … and a few other situations
  - “**Negative**” messages report
    - Most important reason why a key optimization was not performed
    - Compiler tries very hard to report only the most critical optimization inhibitors – otherwise basically looking at a lot of “noise”
  - “**explain**” utility to obtain detailed information on each message

- **Loopmark**
  - Annotated listing
  - Generally easier to use than raw message output

- **Assembly language output**

Loopmark: Compiler Feedback

- **Compiler can generate an filename.lst file**
  - Contains annotated listing of your source code with letter indicating important optimizations

```
%%%    L o o p m a r k   L e g e n d    %%%
     Primary Loop Type         Modifiers
     ------- ---- ----         ---------
     A - Pattern matched       a - atomic memory operation
                               b - blocked
     C - Collapsed             c - conditional and/or computed
     D - Deleted
     E - Cloned
     F - Flat - No calls       f - fused
     G - Accelerated           g - partitioned
     I - Inlined               i - interchanged
     M - Multithreaded         m - partitioned
                               n - non-blocking remote transfer
                               p - partial
     R - Rerolling             r - unrolled
                               s - shortloop
     V - Vectorized            w - unwound
```

Example: Cray loopmark Messages

- –hlist=a ...

```
b-------<  do i3=2,n3-1
b b-----<     do i2=2,n2-1
b b Vr--<        do i1=1,n1
b b Vr            u1(i1) = u(i1,i2-1,i3) + u(i1,i2+1,i3)
b b Vr     *        + u(i1,i2,i3-1) + u(i1,i2,i3+1)
b b Vr            u2(i1) = u(i1,i2-1,i3-1) + u(i1,i2+1,i3-1)
b b Vr     *        + u(i1,i2-1,i3+1) + u(i1,i2+1,i3+1)
b b Vr-->        enddo
b b Vr--<        do i1=2,n1-1
b b Vr            r(i1,i2,i3) = v(i1,i2,i3)
b b Vr     *       - a(0) * u(i1,i2,i3)
b b Vr     *       - a(2) * ( u2(i1) + u1(i1-1) + u1(i1+1) )
b b Vr     *       - a(3) * ( u2(i1-1) + u2(i1+1) )
b b Vr-->        enddo
b b----->     enddo
b------->  enddo
```

Outer loops were blocked

Inner-loops wa vectorized and unrolled

Example: Cray loopmark messages (cont)

ftn-6289 ftn: VECTOR File = resid.f, Line = 29
A loop starting at line 29 was not vectorized because a recurrence was found on "U1" between lines 32 and 38.
ftn-6049 ftn: SCALAR File = resid.f, Line = 29
A loop starting at line 29 was blocked with block size 4.
ftn-6289 ftn: VECTOR File = resid.f, Line = 30
A loop starting at line 30 was not vectorized because a recurrence was found on "U1" between lines 32 and 38.
ftn-6049 ftn: SCALAR File = resid.f, Line = 30
A loop starting at line 30 was blocked with block size 4.
ftn-6005 ftn: SCALAR File = resid.f, Line = 31
A loop starting at line 31 was unrolled 4 times.
ftn-6204 ftn: VECTOR File = resid.f, Line = 31
A loop starting at line 31 was vectorized.
ftn-6005 ftn: SCALAR File = resid.f, Line = 37
A loop starting at line 37 was unrolled 4 times.
ftn-6204 ftn: VECTOR File = resid.f, Line = 37
A loop starting at line 37 was vectorized.

Example of Explain Utility

users/ldr> explain ftn-6289

VECTOR: A loop starting at line %s was not vectorized because a recurrence

was found on "<u>var</u>" between lines <u>num</u> and <u>num</u>.

Scalar code was generated for the loop because it contains a linear recurrence. The following loop would cause this message to be issued:

```
DO I = 2,100
   B(I) = A(I-1)
   A(I) = B(I)
ENDDO
```

CCE 8.7 Key New Features

- More aggressive C++ inlining at default
  - **Default inlining level for C++ is now -hipa4**
    - The Fortran and C default inlining level remains -hipa3 (unchanged)
- The **-hfp3** option is now **enabled when -O3 is specified**
  - Previously, -O3 did not modify the -hfp level
- New **-hnofma** option disables the use of fused-multiply-add instructions
  - Applications sensitive to rounding differences may benefit from disabling fused multiply add (FMA) instructions.
    - This switch is **intended for application debugging purposes**
- Fortran 2018 features: SELECT RANK, COSHAPE, GENERIC
- Enhancements for better automatic **OpenMP affinity and wait policy** settings
  - The cray-specific extension, **AUTO**, is our new default value for OMP_PROC_BIND and OMP_WAIT_POLICY
    - our default used to be FALSE and ACTIVE respectively
  - See the intro_openmp(7) manpage for details
- C11 atomics
  - Atomic operations were an optional part of the C11 standard
- PGAS support for large memory nodes
  - Automatic node partitioning to accommodate address space limitations
  - Transparent to users

Cray MPT Highlights – Cray PMI

- **New topology and placement-aware rank reordering option**
  - This option determines an optimized rank placement based on the hardware resources available to the job at the time of job launching
    - This option has no effect on the resources selected by the workload manager
  - Initial results have shown some applications have improved by as much as 35% using this option

- **PMI_LABEL_ERROUT now supports rank reordering and user-defined labels**
  - The user-defined labels may be set using PMI_LABEL_ERROUT_FORMAT environment variable

Cray MPT Highlights - Dynamic Process Management

- **Cray MPI now supports a subset of Dynamic Process Management (DPM) from the MPI-2 and MPI-3 standards**
  - This support is available as a separate version of the Cray MPICH library, invoked using the new "-craympich-dpm" compiler driver option
    - Full DPM support is targeted for June 2018

- **Cray MPI now supports a larger MPI_TAG_UB value**
  - This feature is linked to the DPM support

- **Cray MPI now supports optimized message matching**
  - Since this feature was needed for the feature to increase the max tag size it is also only available when using the new "-craympich-dpm" option
    - Initial results have shown improvements of as much as 16% in some micro-benchmarks

Additional Cray MPT Highlights

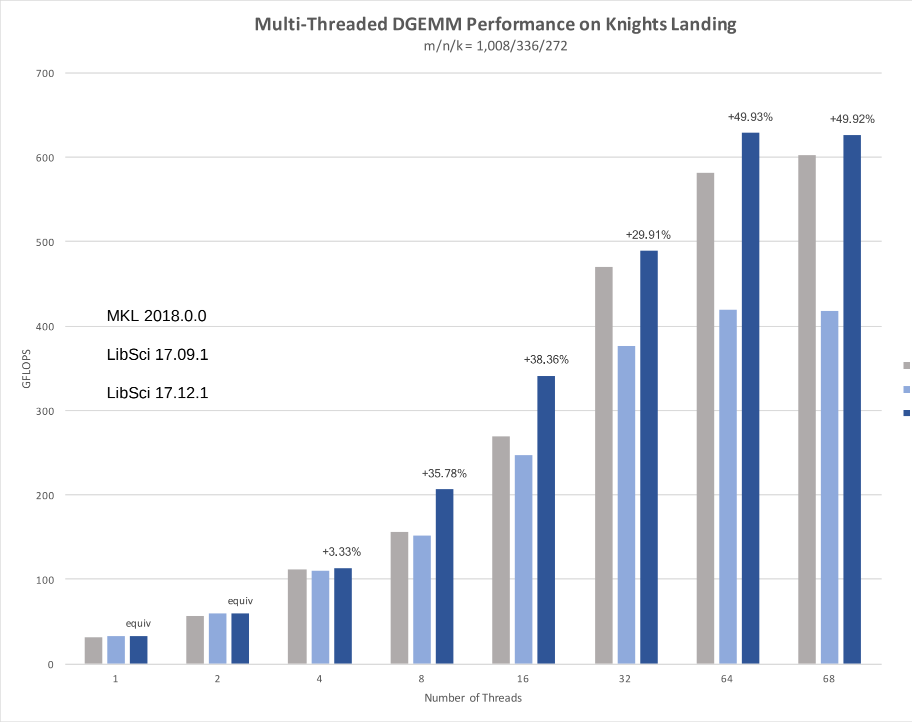
- **Improved support for hugepages in Cray MPICH and Cray SHMEM**
- **Support has been added to improve the default Cray MPI one-sided performance on XC systems**
  - Initial performance improvement over the previous default MPI one-sided version has been observed to be over 4X for both latency and bandwidth
- **Improvements have been made to the Cray MPI async-progress algorithms**
- **MPI_Reduce_scatter and MPI_Reduce_scatter_block has been modified to scale better on high process counts by using much less memory**
- **Cray MPI has been enhanced so that MPI-IO will recognize the new DataWarp Cache FileSystem feature**
- **Cray MPI has been optimized to improve network communication bandwidth performance for Intel Skylake**
  - Performance improvements of up to 22% have been seen when running with more than 8 ranks per node.

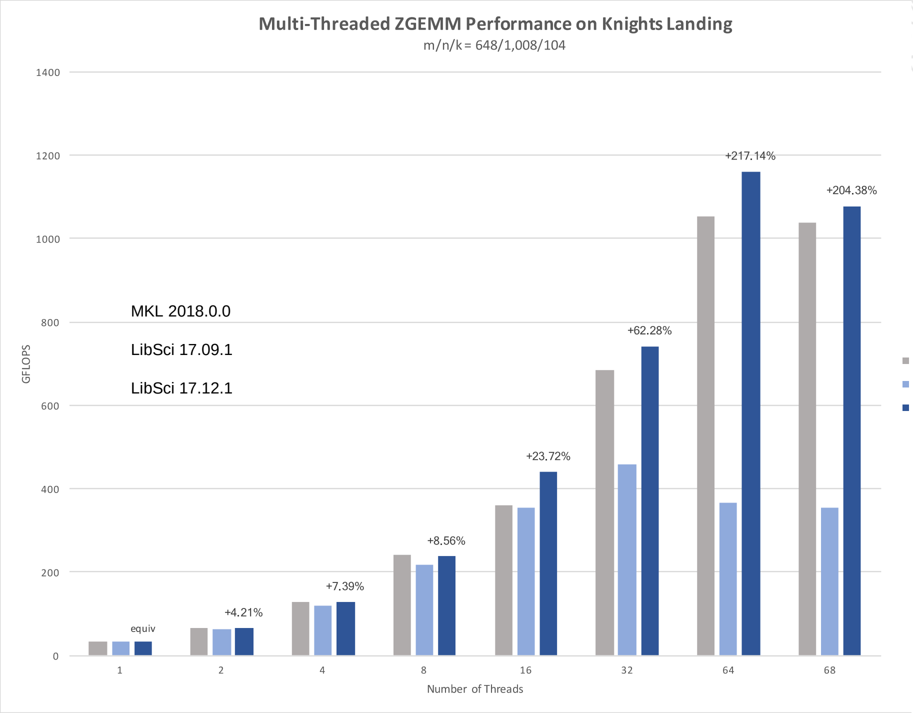
CSML Highlights

- **Cray LibSci**
  - New QDWH, KSVD, and ELPA backends for Scalapack eigensolvers
  - Backend integration with recent releases of cray-R
  - NumPy and SciPy integration with cray-python

- **Cray FFTW3**
  - FASTPLAN optimizations for Intel Skylake CPU targets
  - Default threading model is now OpenMP for all targets
  - Continued support for arbitrary dimension and size for real/complex

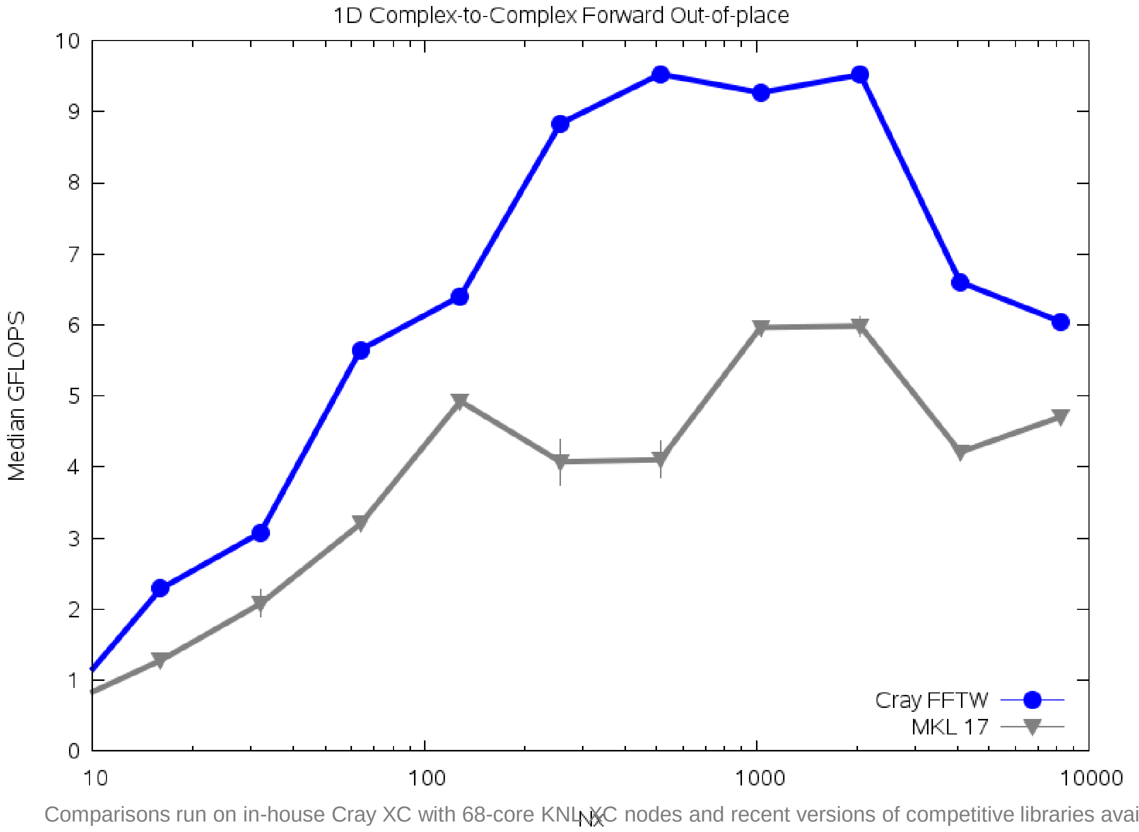
Cray LibSci DGEMM and ZGEMM for KNL





Comparisons run on in-house Cray XC with 68-core Knight's Landing XC nodes and recent versions of competitive libraries available

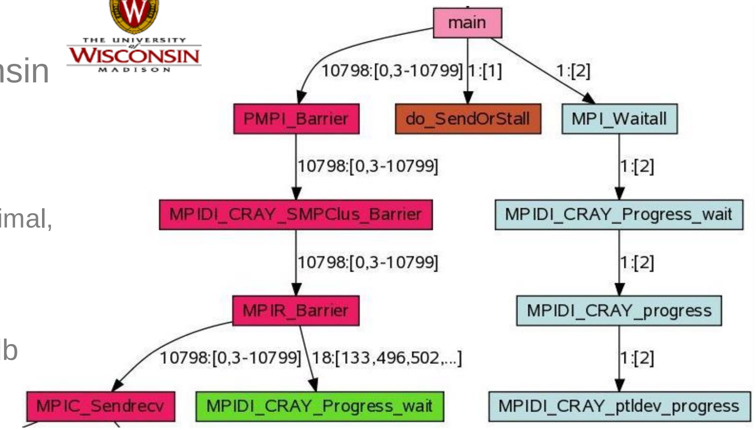
Cray FFTW Performance on KNL



Comparisons run on in-house Cray XC with 68-core KNL XC nodes and recent versions of competitive libraries available

Debugging on Cray Systems

- **Systems with thousands of threads of execution need a new debugging paradigm**
- **Support for traditional debugging mechanism**
  - RogueWave TotalView and Allinea DDT
- **Cray's focus is to build tools around traditional debuggers with innovative techniques for productivity and scalability**
  - **Scalable** Solutions based on MRNet from University of Wisconsin
    - **STAT - Stack Trace Analysis Tool**
      - Scalable generation of a single, merged, stack backtrace tree
    - **ATP - Abnormal Termination Processing**
      - Scalable analysis of a sick application, delivering a STAT tree and a minimal, comprehensive, core file set.
  - gdb4hpc / CCDB
    - Ability to see data from multiple processors in the same instance of gdb
      - without the need for multiple windows
    - **Comparative debugging**
      - A **data-centric paradigm** instead of the traditional control-centric paradigm
      - Collaboration with University of Queensland



gdb4hpc

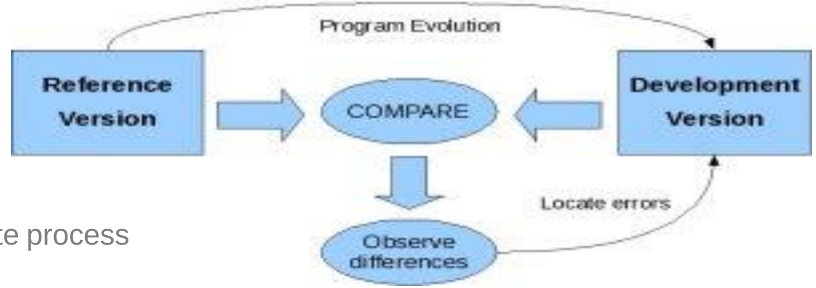
```
$ lgdb
lgdb 3.0 - Cray Line Mode Parallel Debugger
With Cray Comparative Debugging Technology.
Copyright 2007-2016 Cray Inc. All Rights Reserved.
Copyright 1996-2016 University of Queensland. All Rights
Reserved.

Type "help" for a list of commands.
Type "help <cmd>" for detailed help about a command.
dbg all> launch --launcher-args="-N 4 --tasks-per-node=32 --
cpus-per-task=1 --exclusive --partition=hsw16" $App1{128} himeno
Starting application, please wait...
Creating MRNet communication network...
184871928.475738: UNKNOWN_THREAD(0x7f34b9fe0840): Network.C[840]
init_FrontEnd - WARNING: Topology Root (falcon) is not local
host (falcon.cray.com)
SLURM PID FILE: /tmp/cray_cti-ldr/slurmLMKIoe/slurm_pid
Waiting for debug servers to attach to MRNet communications
network...
Timeout in 400 seconds. Please wait for the attach to complete.
Number of dbgsrvs connected: [1];  Timeout Counter: [0]
Number of dbgsrvs connected: [34];  Timeout Counter: [0]
Number of dbgsrvs connected: [44];  Timeout Counter: [0]
Number of dbgsrvs connected: [92];  Timeout Counter: [0]
Number of dbgsrvs connected: [128];  Timeout Counter: [0]
Finalizing setup...
Launch complete.
App1{0..127}: Initial breakpoint, initcomm at himeno.f:381
dbg all> break jacobi
App1{0..127}: Breakpoint 1: file himeno.f, line 209.
dbg all> c
App1{0..127}: Breakpoint 1, jacobi at himeno.f:209
```

```
dbg all> l
App1{0..127}: 209            subroutine jacobi(nn,gosa)
App1{0..127}: 210
C*******************************************************************
App1{0..127}: 211            IMPLICIT NONE
App1{0..127}: 212      C
App1{0..127}: 213            include 'mpif.h'
App1{0..127}: 214            include 'param.h'
App1{0..127}: 215      C
App1{0..127}: 216            integer :: nn,i,j,k,loop,ierr
App1{0..127}: 217            real (kind=4) ::
gosa,wgosa,s0,ss
App1{0..127}: 218
dbg all> backtrace
App1{0..127}: #0  0x0000000000402020 in jacobi at
himeno.f:209
App1{0..127}: #1  0x00000000004012de in himenobmtxp at
himeno.f:91
dbg all> print npe
App1{0..127}: 128
dbg all> p jmax
App1{0..3,12..19,28..35,44..51,60..67,76..83,92..99,108..11
5,124..127}: 129
App1{4..11,20..27,36..43,52..59,68..75,84..91,100..107,116.
.123}: 130
dbg all>
```

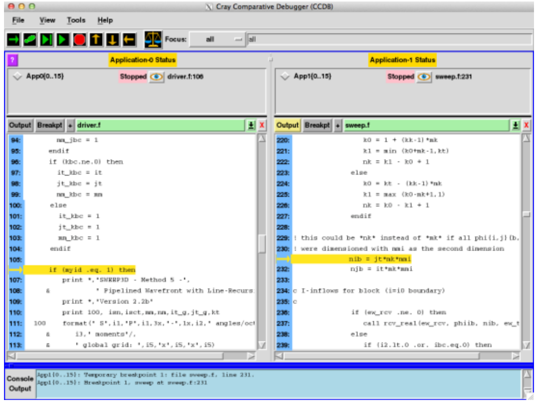
COMPUTE | STORE | ANALYZE

Comparative Debugger

- **What is comparative debugging?**
  - Data centric approach instead of the traditional control-centric paradigm
  - Two applications, same data
  - Key idea: The data should match
  - Quickly isolate deviating variables



- **Comparative debugging tool**
  - NOT a traditional debugger!
  - Assists with comparative debugging
  - CCDB GUI hides the complexity and helps automate process
    - Creates automatic comparisons
    - Based on symbol name and type
    - Allows user to create own comparisons
    - Error and warning epsilon tolerance
    - Scalable



- **How does this help me?**
  - Algorithm re-writes
  - Language ports
  - Different libraries/compilers
  - New architectures

```
assert P1::T1[0..99]@"file.c":240
       = P2::Y2(1,100)@"prog.f":300
```

- **Collaboration with University of Queensland**

THE UNIVERSITY OF QUEENSLAND
AUSTRALIA

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publicly announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, URIKA and YARCDATA. The following are trademarks of Cray Inc.: CHAPEL, CLUSTER CONNECT, CLUSTERSTOR, CRAYDOC, CRAYPAT, CRAYPORT, DATAWARP, ECOPHLEX, LIBSCI, NODEKARE, REVEAL. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used on this website are the property of their respective owners.

CRAY

COMPUTE | STORE | ANALYZE

Questions?

Thank You!