The Cray Programming Environment

Luiz DeRose & Heidi Poxon
Cray Inc.

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 publically 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.

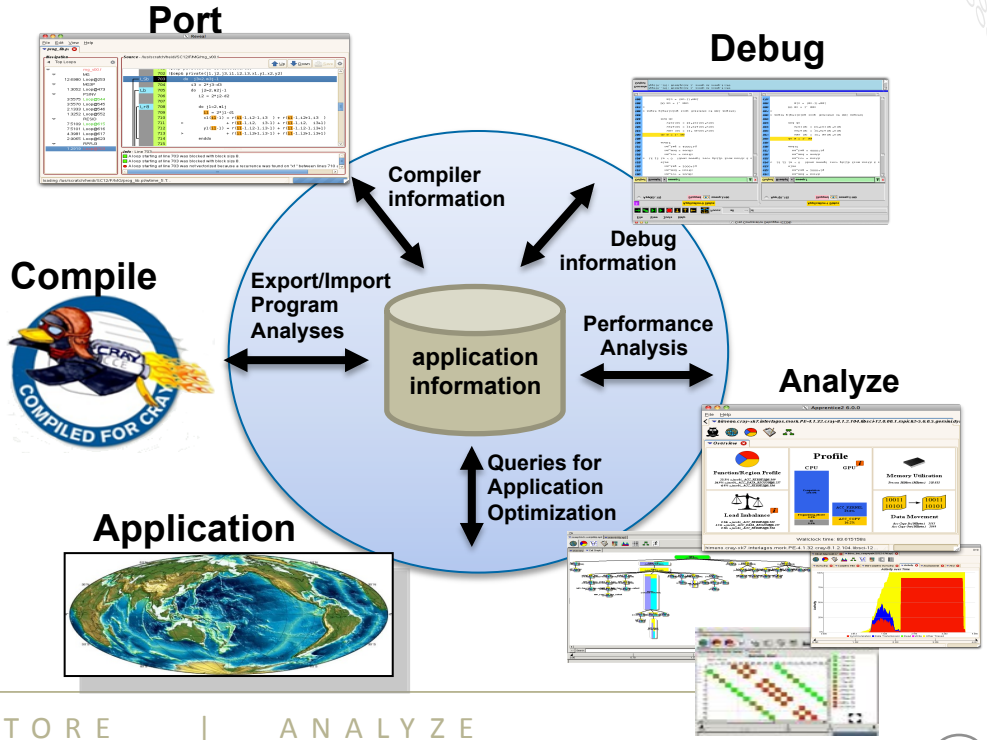
Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

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.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and 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 names and brands may be claimed as the property of others. Other product and service names mentioned herein are the trademarks of their respective owners.

The Programming Environment Mission

- Focus on **Performance** and **Programmability**
  - It is the role of the Programming Environment to **close the gap** between observed performance and achievable performance
- Support the **application development life cycle** by providing a **tightly coupled** environment with compilers, libraries, and tools that will **hide the complexity** of the system
  - Address issues of scale and complexity of HPC systems
  - Target **ease of use** with extended **functionality** and increased **automation**
  - Close **interaction with users**
    - For feedback targeting functionality enhancements



Cray Programming Environment Focus

- **Performance**
  - Help users **maximize the cycles to the application**
    - Address issues of scale and complexity of HPC systems
- **Programmability**
  - How do you get intuitive behavior and best performance with the least amount of effort
    - Provide the **best environment to develop, debug, analyze, and optimize** applications for production supercomputing
    - Provide programming environment **consistency across Cray platforms**

Cray Programming Environment for KNL

Programming Languages

- Fortran
- C
- C++
- Python

Programming models

- Distributed Memory (Cray MPT)
  - MPI
  - SHMEM
  - GA
- Shared Memory
  - OpenMP 4.0
  - OpenACC 2.0
- PGAS
  - UPC
  - CAF
  - CoArray C++

Compilers

- Cray Compiling Environment (CCE)
- GNU
- Intel compilers

Environment setup

- Modules

Tools

Debuggers

- TotalView
- DDT
- Igdb

Debugging Tools

- ATP
- STAT

Performance Analysis

- CrayPat & Cray Apprentice2

Porting Tools

- Reveal
- CCDB

Optimized Scientific Libraries

Dense

- BLAS
- LAPACK
- ScaLAPACK
- Iterative Refinement Toolkit

Sparse

- Cray PETSc (with CASK)
- Cray Trilinos (with CASK)

FFT

- FFTW

I/O Libraries

- NetCDF
- HDF5

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

Software Support

- **PE software is released on a monthly schedule and an annual "roll-up" release**
  - Cray Compiling Environment (CCE)
  - Cray Message Passing Toolkit (MPT)
  - Cray Performance Measurement and Analysis Tools (CPMAT)
  - Cray Scientific and Math Libraries (CSML)
  - Cray Debugger Support Tools (CDST)
  - Cray Environment Setup and Compiling support (CENV)
- **Each component can be released asynchronously**
- **Access to software (including 3rd party) is managed using the GNU module command**
  - Cray provides utility for sites to create modules for 3rd party software

Software Support (continued)

- **Software components go through multiple layers of testing**
  - Over 35,000 tests run nightly for CCE

- **Supported platforms:**
  - Cray XC series systems with HSW, IVB, SNB, with Kepler, Atlas, and/or KNC coprocessors
  - Cray XE and Cray XK7 Gemini systems with IL, AD, IL with Kepler
  - Cray CS300 systems

The Cray Compiling Environment

- **Cray technology focused on scientific applications**
  - Takes advantage of **automatic vectorization**
  - Takes advantage of **automatic shared memory parallelization**
- **Automatic optimizations for Cray architectures to deliver performance of a new target through simple recompile**
  - Hide system complexity
- **PGAS languages (UPC & Fortran Coarrays) fully optimized and integrated into the compiler**
  - No preprocessor involved
  - Target the network appropriately
  - Full debugger support with Allinea's DDT
- **Focus on standards for application portability and investment protection**
  - Fortran 2008 standard compliant
  - C++11 compliant (9/24/2015)
  - OpenMP 4.0 compliant (9/24/2015)
  - OpenACC 2.0
  - UPC 1.3

OpenMP

- **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**

UPC and Fortran Coarray Features on CCE

- **C-based UPC and Fortran Coarray are PGAS language extensions, not stand-alone languages**
- **A subset of Fortran coarray collectives were added for CCE**
  - Although they are not yet part of the official language – they are too useful to be delayed
- **Significant improvements were made to the automatic use of blocked network transfers, including:**
  - Automatic conversion of multiple single-word accesses into blocked accesses
  - Improved capabilities for pattern matching to hand-optimized library routines, including messages stating what might be inhibiting the conversion
- **UPC and Fortran coarrays support up to 2,147,483,647 threads within a single application**
  - We actually did hit the previous limit of 65,535!

Coarray C++

- **Coarray concept is programming language independent**
  - Fortran just happens to be the first example
- **Coarray C++ templates extend standard C++ without requiring compiler changes**
- **The Cray implementation uses the same runtime library as Cray UPC and coarray Fortran, optimized for Gemini and Aries networks**
- **Examples:**
  - `coarray<int[100]> x;` // Each image of program has 100 ints
  - `x(2)[3] = 4;` // Write x[3] = 4 on program image 2
  - `int y = x[1];` // Read x[1] from the local program image
- **Why not create UPC++ instead?**
  - Would require modifying C++ syntax, unlike most new C++11 features
    - UPC shared concept modifies array element type (internal to array) whereas a coarray adds a dimension (external to array)
  - Coarray data distribution used by many UPC programs for best locality
  - Coarrays withstood scrutiny of Fortran standardization process

The CCE Program Library (PL)

- **An application wide repository for compiler and tools information**
  - Allows the user to specify a repository of compiler information for an application build
- **Provides the framework for whole application analysis**
  - Whole application IPA information for optimization
  - Automatic whole application inlining and cloning
  - Various interprocedural optimizations
  - Whole application static error detection
- **Provides ability for tools to annotate loops with runtime feedback and other performance hints without source change**
  - Support for the Cray refactoring tool Reveal.

CCE Program Library

- **Two command line options control the Program Library functionality**
  - -h pl = <PL_path> specifies the repository
    - Ftn –hpl=./PL.1 tells the compiler to either update the Program Library "./PL.1" if it exists, or create it if it does not exist.
    - <PL_path> must specify a single location to be used for entire application build. If a makefile changes directories during a build, an absolute path might be necessary.
  - -h wp enables whole-program mode
    - All inlining, optimization and code generation is delayed until the link step of the build.
- **The PL can be moved as long as the relative paths to all application files remains unchanged**

Some CCE Usage Tips (1 of 2)

- **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 the most thoroughly tested configuration

- **Using –O3,fp3 (or –O3 –hfp3, or some variation)**
  - -O3 only gives you slightly more than –O2
  - -hfp3 gives you a lot more floating point optimization, esp. 32-bit
  - This is also thoroughly tested

Some CCE Usage Tips (2 of 2)

- **Optimizing for compile time rather than execution time**
  - Compile time can sometimes be improved by disabling certain features/ optimizations
  - Some common things to try: -hnodwarf, -hipa0, -hunroll0
  - With CCE 8.3, try the new –hdevelop option
  - -O0 may actually compile slower than –O2 and is not a good option to try

- **CCE only gives minimal information to stderr when compiling**
  - To see more information, you should request a compiler listing file
    - Flags **-ra** for ftn or **-hlist=a** for cc
    - Writes a file with extension .lst
    - Contains annotated source listing, followed by explanatory messages
  - Each message is tagged with an identifier, e.g.: **ftn-6430**
    - to get more information on this, type: **explain <identifier>**
  - Cray Reveal can display all this information (and more)

CCE Enhancements for KNL

- **Directive (pragma) to support data allocation in high bandwidth memory**
  - Support for Fortran, C, and C++
  - The directive can be used on both local and global variables to place the variables in high bandwidth memory
  - The directive can also be used on a statement to change any allocation routines on that statement (allocate, malloc, etc.) to use high bandwidth memory
  - Future direction for memory hierarchy control
    - Ideally will become part of a standard, possibly OpenMP

CCE Proposed API for KNL HBM

- **Directive (pragma) to control placement for high bandwidth memory**
  - Support for Fortran, C and C++
  - Proposed directive
    - !dir$ memory(*attributes*) *[list of variables]*
    - #pragma memory(*attributes*) *[list of variables or allocatable members]*
    - *Attributes* – list of desired memory attributes (bandwidth, capacity, nonvolatile, etc.).
    - KNL high bandwidth memory would be requested with:
      - #pragma memory(bandwidth) …
- **The directive is valid both on statements and variables**
- **Statements**
  - Changes explicit allocation routines in the next statement to use high bandwidth memory
  - Fortran: allocate
  - C: malloc, calloc, realloc
  - C++: new

CCE Proposed API for KNL HBM

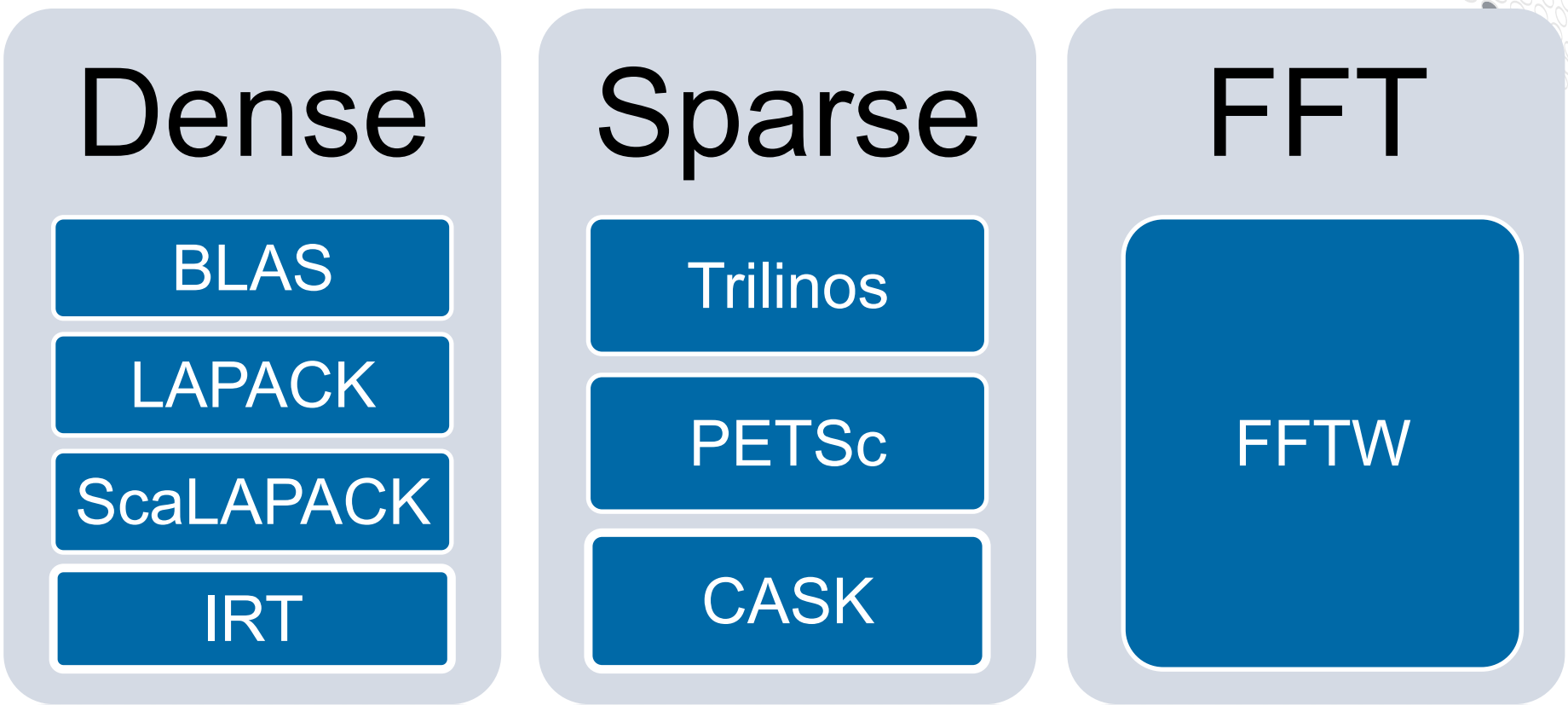
- **Variables**
  - Specified at declaration of variable
    - Within type for allocatable members
    - For global variables, directive must be visible for every use of global
  - **Allowed**
    - Local and global variables
    - Fixed size and variable length arrays
    - Fortran allocatables
      - Memory allocated will use high bandwidth memory
    - Allocatable members of derived types
      - Memory allocated will use high bandwidth memory
  - **Not allowed**
    - Arguments
    - Common blocks or variables within a common block
    - Fortran pointers
    - Variables involved in equivalences
    - Coarray or UPC shared variables

Cray MPI & Cray SHMEM

- **MPI**
  - Implementation based on MPICH3 from ANL
    - ANL does base MPI standard support, we add new functionality, improve performance both on-node, and all ranges of scale including at very high scale
  - Full MPI-3 support with the exception of
    - MPI-2 Dynamic process management (MPI_Comm_spawn)
  - MPI Forum active participant
  - Participated in the MPICH ABI Consortium
    - ANL MPICH, Intel MPI, IBM PE MPI and Cray MPI

- **Cray SHMEM**
  - Cray XE & XC implementation on top of the Distributed Memory Applications API (DMAPP)
  - Fully optimized Cray SHMEM library supported
  - Cray implementation close to the T3E model
  - Fully support the OpenSHMEM draft 1.0 standard
  - OpenSHMEM standard active participant

Cray Scientific Libraries – Functional View



CrayBLAS

- CrayBLAS is a Cray Proprietary library implementation for the Basic Linear Algebra Subprograms (BLAS) which is tuned for Cray systems

- Users of libsci who depend heavily on BLAS performance are encouraged to contact Cray at CrayBLAS@cray.com so that optimizations for their particular calling sequences can be added to an incremental enhancement of the CrayBLAS performance model

Libsci Dense Linear Algebra Libraries

- **LAPACK**
  - Cray optimized set of routines that support OpenMP multithreading for use on single nodes for several different compilers and architectures
- **ScaLAPACK**
  - Cray optimized set of distributed linear algebra for multinode use that makes use of both CrayBLAS and LibSci LAPACK routines
- **Iterative Refinement Toolkit (IRT)**
  - Custom set of Cray LU, Cholesky, and QR solver routines for well conditioned problems (low condition number systems) that use mixed-precision to offer significant speed-ups over full-precision solvers. LAPACK and ScaLAPACK wrappers as well as expert interfaces for greater control over the algorithms

Threading

- **LibSci is compatible with OpenMP**
  - Control the number of threads to be used in your program using OMP_NUM_THREADS
  - e.g. in job script
  - setenv OMP_NUM_THREADS 16
  - Then run with aprun –n1 –d16
- **What behavior you get from the library depends on your code**
  - No threading in code
    - The BLAS call will use OMP_NUM_THREADS threads
  - Threaded code, outside parallel region
    - The BLAS call will use OMP_NUM_THREADS threads
  - Threaded code, inside parallel region
    - The BLAS call will use a single thread
- **Threaded LAPACK works exactly the same as threaded BLAS**
  - Anywhere LAPACK uses BLAS, those BLAS can be threaded
  - Some LAPACK routines are threaded at the higher level
  - No special instructions

Iterative Refinement Toolkit

- Mixed precision can yield a big win on x86 machines
- SSE (and AVX) units issue double the number of single precision operations per cycle
- On CPU, single precision is always 2x as fast as double
- IRT is a suite of tools to help exploit single precision
  - A library for direct solvers
  - An automatic framework to use mixed precision under the covers

Iterative Refinement Toolkit - Library

- **Various tools for solves linear systems in mixed precision**
- **Obtaining solutions accurate to double precision**
  - For well conditioned problems
- **Serial and Parallel versions of LU, Cholesky, and QR**
- **2 usage methods**
  - **IRT Benchmark routines**
    - Uses IRT 'under-the-covers' without changing your code
      - Simply set an environment variable
      - Useful when you cannot alter source code
  - **Advanced IRT API**
    - If greater control of the iterative refinement process is required
      - Allows
        - condition number estimation
        - error bounds return
        - minimization of either forward or backward error
        - 'fall back' to full precision if the condition number is too high
        - max number of iterations can be altered by users

IRT library usage

Decide if you want to use advanced API or benchmark API

benchmark API :

setenv IRT_USE_SOLVERS 1

advanced API :

1. **locate the factor and solve in your code (LAPACK or ScaLAPACK)**

2. **Replace factor and solve with a call to IRT routine**
   - e.g. dgesv -> irt_lu_real_serial
   - e.g. pzgesv -> irt_lu_complex_parallel
   - e.g pzposv -> irt_po_complex_parallel

3. **Set advanced arguments**
   - Forward error convergence for most accurate solution
   - Condition number estimate
   - “fall-back” to full precision if condition number too high

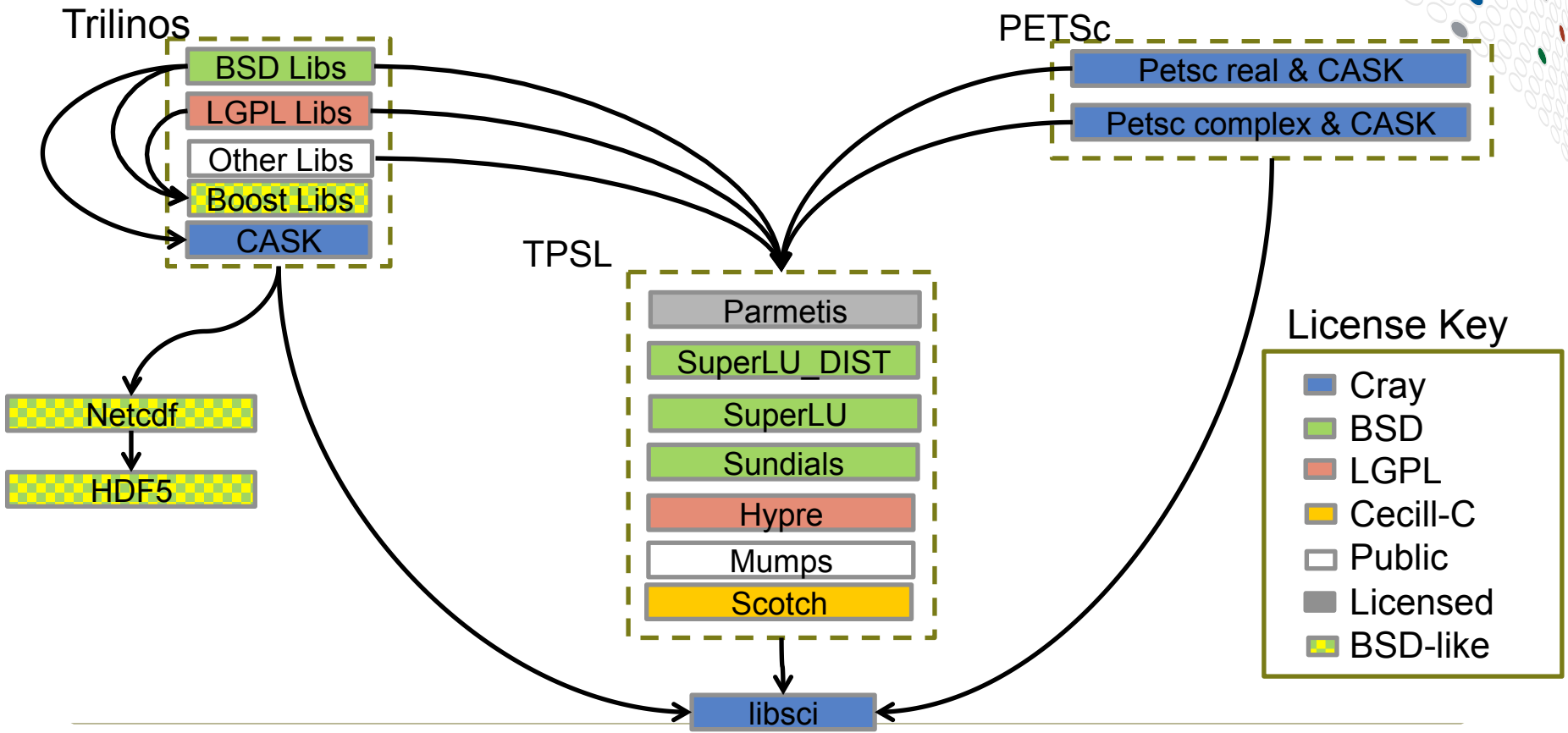
Note : “info” does not return zero when using IRT !!

Cray Adaptive Sparse Kernel (CASK)

- **Sparse matrix operations in PETSc and Trilinos on Cray systems are optimized via CASK**

- **CASK is a product developed at Cray using the Cray Auto-tuning Framework**
  - Offline :
    - ATF program builds many thousands of sparse kernels
    - Testing program defines matrix categories based on density, dimension etc
    - Each kernel variant is tested against each matrix class
    - Performance table is built and adaptive library constructed
  - Runtime
    - Scan matrix at very low cost
    - Map user's calling sequence to nearest table match
    - Assign best kernel to the calling sequence
    - Optimized kernel used in iterative solver execution

Cray Sparse Libraries



Cray Scientific Libraries Usage

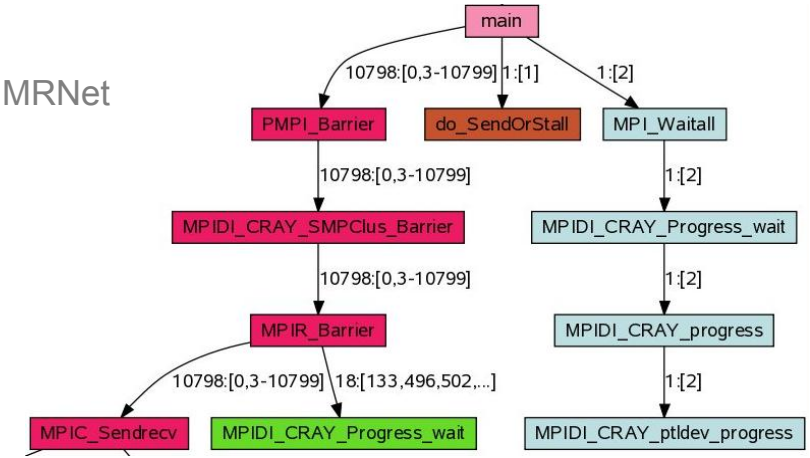
- **LIbSci**
  - The drivers should do it all for you. Don't explicitly link.
  - For threads, set OMP_NUM_THREADS
    - Threading is used within libsci.
    - If you call within parallel region, single thread used
    - -Wl, -ydgemm_ reveals where the link was resolved
- **FFTW**
  - Module load fftw (there are also wisdom files you can pick up)
- **PETSc**
  - Module load petsc (or module load petsc-complex)
  - Use as you would your normal petsc build
- **Trilinos**
  - Module load trilinos
- **CASK – no need to do anything you get free optimization**

Debugging on Cray Systems

- **Systems with thousands of threads of execution need a new debugging paradigm**
- **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 tre
    - **ATP - Abnormal Termination Processing**
      - Scalable analysis of a sick application, delivering a STAT tree and a minimal, comprehensive, core file set.
  - LGDB / CCDB
    - Ability to see data from multiple processors in the same instance of lgdb
      - without the need for multiple windows
    - **Comparative debugging**
      - A **data-centric paradigm** instead of the traditional control-centric paradigm
      - Collaboration with University of Queensland
- **Support for traditional debugging mechanism**
  - RogueWave TotalView and Allinea DDT

STAT & ATP

- **Stack trace sampling and analysis for large scale applications**
  - Sample application stack traces
  - Scalable generation of a single, merged, stack backtrace tree
    - A comprehensible view of the entire application
    - Discover equivalent process behavior
      - Group similar processes
      - Reduce number of tasks to debug
    - 128K processes analyzed in 2.7 seconds, using MRNet
- **Merge/analyze traces:**
  - Facilitate scalable analysis/data presentation
  - Multiple traces over space or time
  - Create call graph prefix tree
    - Compressed representation
    - Scalable visualization
    - Scalable analysis
- **Abnormal Termination Processing (ATP)**
  - **Provides merged stack backtrace tree** for STAT like analysis on any application process trap
    - Leaf nodes of tree define a modest set of processes to core dump (or to attach a debugger)

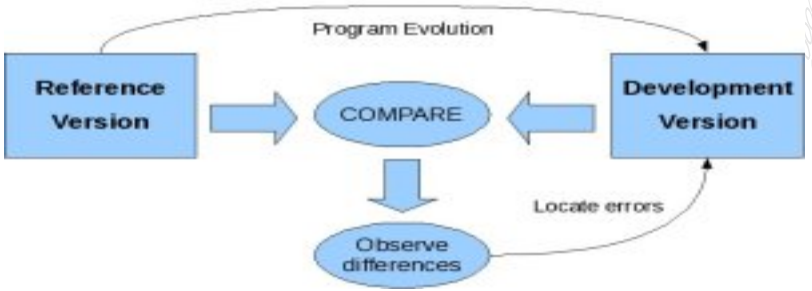


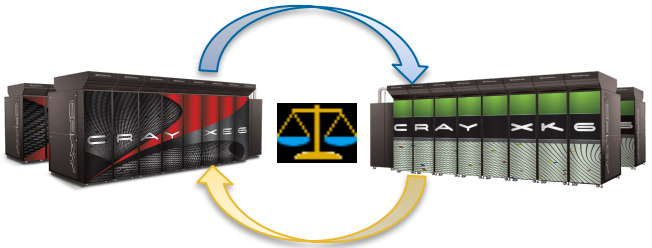
ATP Hold Time

- **ATP is able to hold a dying application in stasis in order to allow the user to attach to it with a debugger**
  - To do so, set the **ATP_HOLD_TIME** environment variable to the number of minutes desired
- **Once attached, the debugging session can last as long as the batch system allows**
  - Which in turn depends on the compute node resources you requested when you began your session
  - So use ATP_HOLD_TIME to define the time you need to attach to the application, not the total time needed for the debugging session.
- **If ATP_HOLD_TIME is set, core dumping is disabled**

CCDB Overview

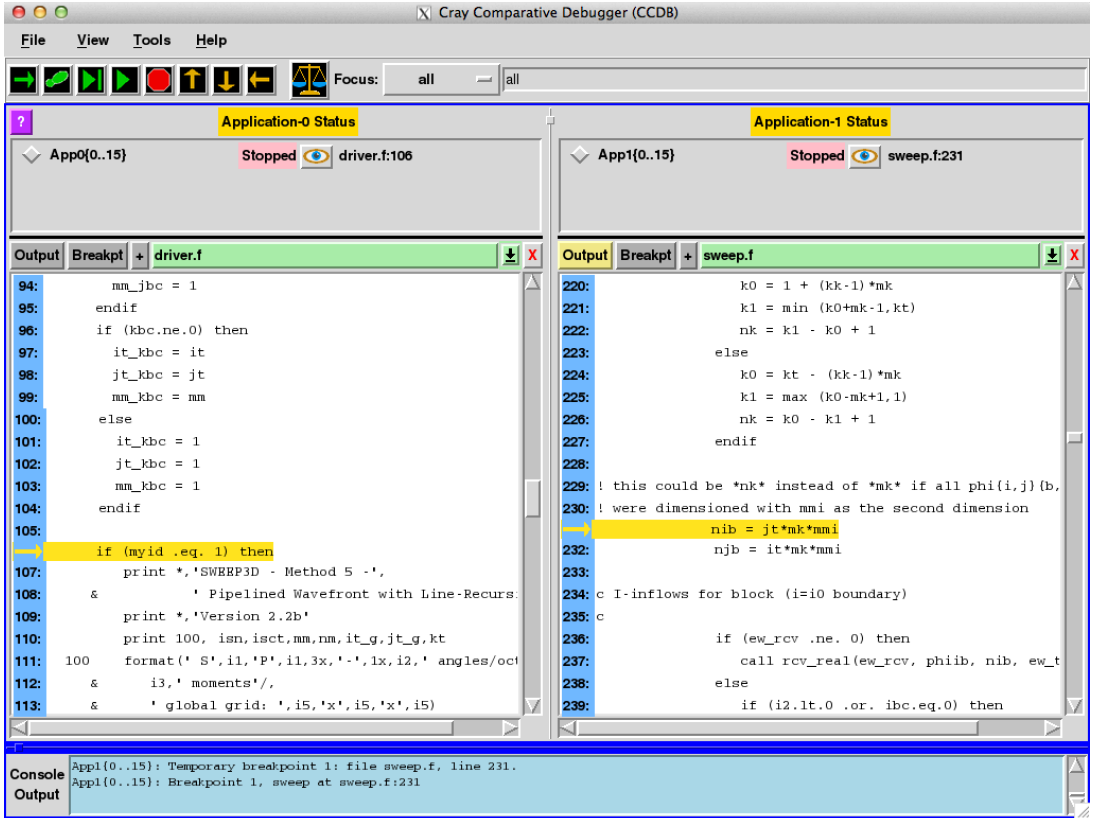
- **What is comparative debugging?**
  - Data centric approach instead of the traditional control-centric paradigm
  - Two applications, same data
  - Key idea: The data should match
  - Quickly isolate deviating variables
- **Comparative debugging tool**
  - NOT a traditional debugger!
  - Assists with 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
- **Collaboration with University of Queensland**



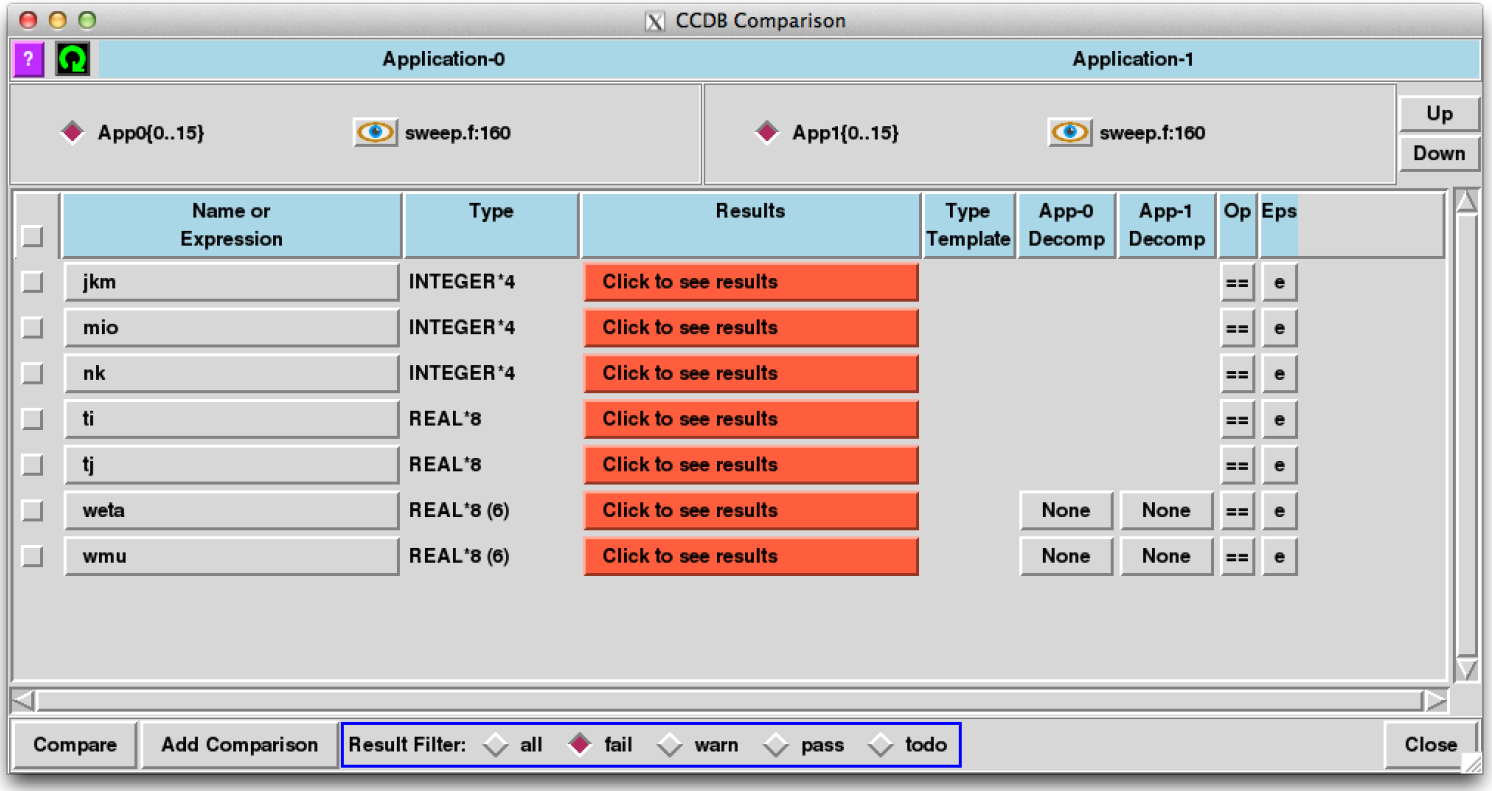


THE UNIVERSITY OF QUEENSLAND
AUSTRALIA

Cray Comparative Debugger



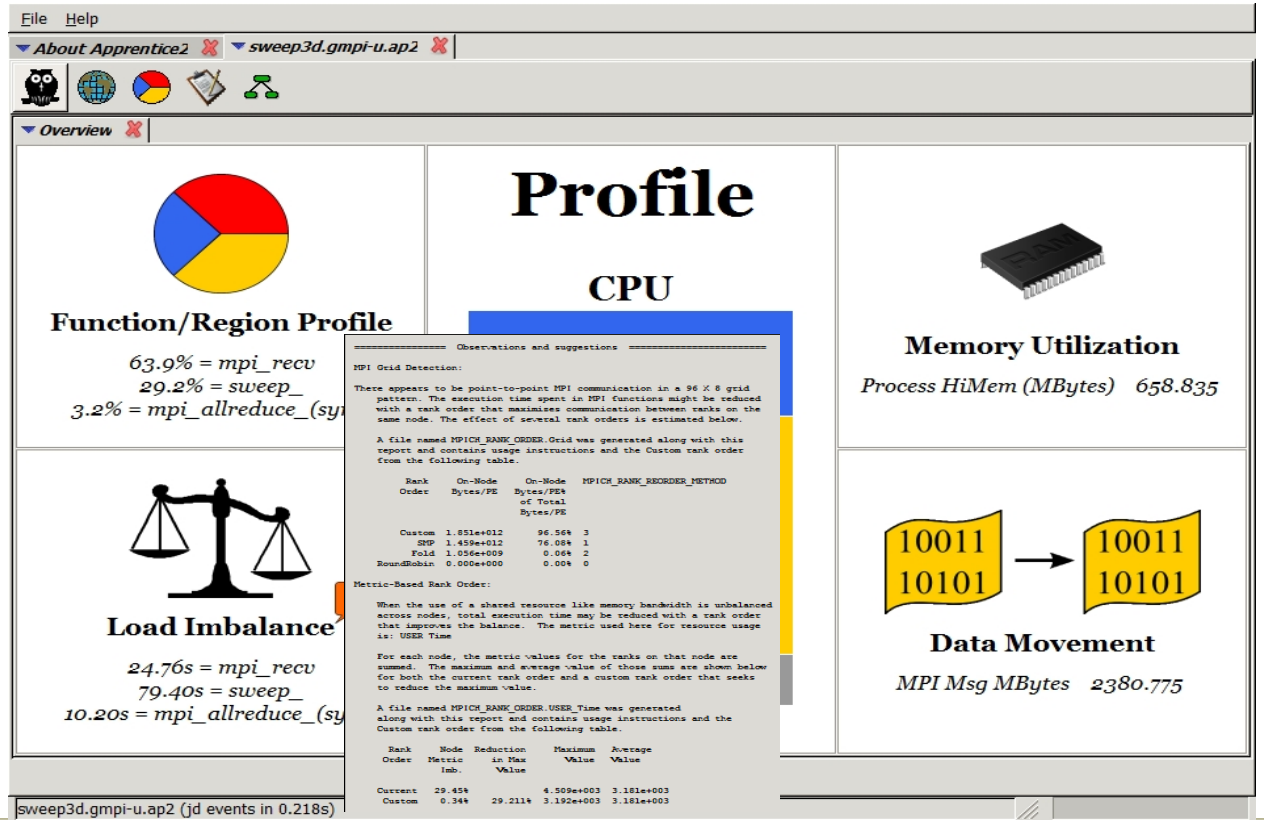
CCDB - Comparison



Cray Performance Analysis Tools

- From performance measurement to performance analysis
- Assist the user with application performance analysis and optimization
  - Help user identify important and meaningful information from potentially massive data sets
  - Help user identify problem areas instead of just reporting data
  - Bring optimization knowledge to a wider set of users
- Focus on ease of use and intuitive user interfaces
  - Automatic program instrumentation
  - Automatic analysis
- Target scalability issues in all areas of tool development

Application Performance Summary



Two Interfaces to the Performance Tools

- **CrayPat offers a wealth of performance measurement, analysis and presentation options for in-depth performance investigation and tuning assistance**

- **CrayPat-lite offers easy access to an application performance summary for users not familiar with the Cray performance tools or who may not be familiar with performance analysis**

- **CrayPat classic and CrayPat-lite are designed to compliment each other**
  - Produce files with same format
  - Users familiar with CrayPat can easily switch back and forth between the two interfaces
  - CrayPat-lite users become familiar with reporting style also used with CrayPat

CrayPat-lite Output - Job Information

- Number of MPI ranks, ranks per node, number of threads
- Wallclock
- High memory water mark
- Aggregate MFLOPS (if available on processor)
- I/O read and write rates
- Energy consumption
- Profile of top time consuming routines with load balance
- Observations
- Instructions on how to get more information

Example CrayPat-lite Output

```
CrayPat/X:  Version 6.1.4.12457 Revision 12457 (xf 12277)  02/26/14 13:58:24
Experiment:                   lite  lite/sample_profile
Number of PEs (MPI ranks):    8164
Numbers of PEs per Node:        16  PEs on each of  510  Nodes
                                 4  PEs on    1  Node
Numbers of Threads per PE:       1
Number of Cores per Socket:      8
Execution start time:  Fri Feb 28 23:06:31 2014
System name and speed:  hera2 2100 MHz

Wall Clock Time:     999.595275 secs
High Memory:             475.52 MBytes
MFLOPS (aggregate):  806112.33 M/sec
I/O Read Rate:            33.57 MBytes/Sec
I/O Write Rate:          215.40 MBytes/Sec
```

Example CrayPat-lite Output (2)

```
Table 1:  Profile by Function Group and Function (top 7 functions shown)
 Time%  |      Time  |    Imb.   |  Imb.   |    Calls  |Group
        |            |    Time   | Time%   |           | Function
        |            |           |         |           |  PE=HIDE

 100.0% | 101.961423 |       -- |     -- | 5315211.9 |Total
|-------------------------------------------------------------------------------------
|  92.5% |  94.267451 |       -- |     -- | 5272245.9 |USER
||------------------------------------------------------------------------------------
||  75.8% |  77.248585 | 2.356249 |   3.0% |    1001.0 |LAMMPS_NS::PairLJCut::compute
||   6.5% |   6.644545 | 0.105246 |   1.6% |      51.0 |LAMMPS_NS::Neighbor::half_bin_newton
||   4.1% |   4.131842 | 0.634032 |  13.5% |       1.0 |LAMMPS_NS::Verlet::run
||   3.8% |   3.841349 | 1.241434 |  24.8% | 5262868.9 |LAMMPS_NS::Pair::ev_tally
||   1.3% |   1.288463 | 0.181268 |  12.5% |    1000.0 |LAMMPS_NS::FixNVE::final_integrate
||====================================================================================
|   7.0% |   7.110931 |       -- |     -- |   42637.0 |MPI
||------------------------------------------------------------------------------------
||   4.8% |   4.851309 | 3.371093 |  41.6% |   12267.0 |MPI_Send
||   1.5% |   1.536106 | 2.592504 |  63.8% |   12267.0 |MPI_Wait
|====================================================================================
```

Example CrayPat-lite Output (3)

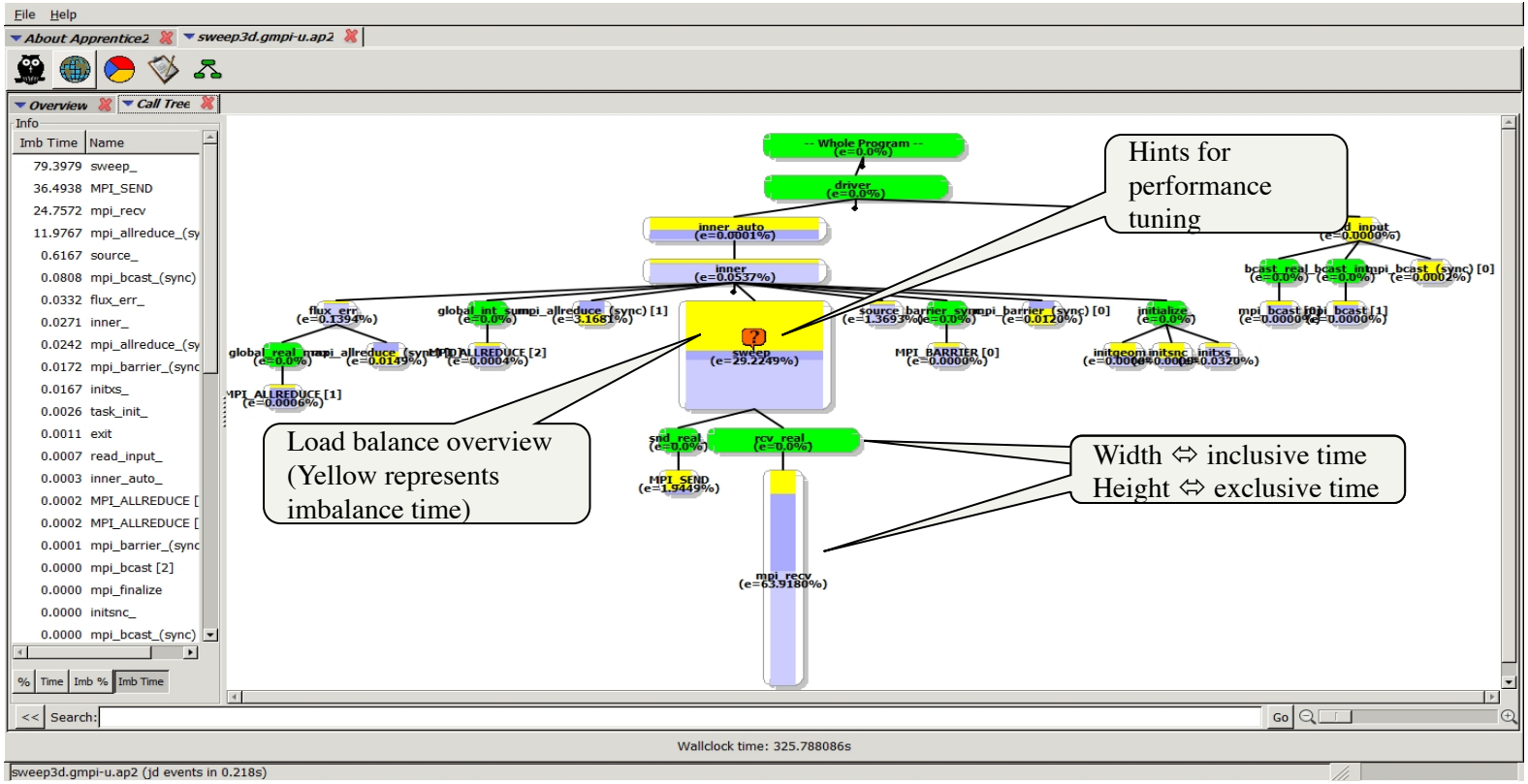
```
Table 2:  File Input Stats by Filename

  Read Time |  Read MBytes |  Read Rate |      Reads |  Bytes/ Call |File Name[max10]
            |              | MBytes/sec |            |              | PE=HIDE

 387.432937 | 13006.522781 |  33.571030 | 41596900.0 |       327.87 |Total
|-------------------------------------------------------------------------------------
| 331.691801 |  1395.829828 |   4.208213 | 13153931.0 |       111.27 |/proc/self/maps
|  13.129507 |  4075.682968 | 310.421627 |      868.0 |   4923575.28 |regional.grid.a
|  12.654338 |  2000.329418 | 158.074605 | 26892862.0 |        77.99 |./patch.input
|   3.924810 |   679.265625 | 173.069704 |        3.0 | 237420544.00 |./forcing.radflx.a
. . .
```

Sampling with Line Number information

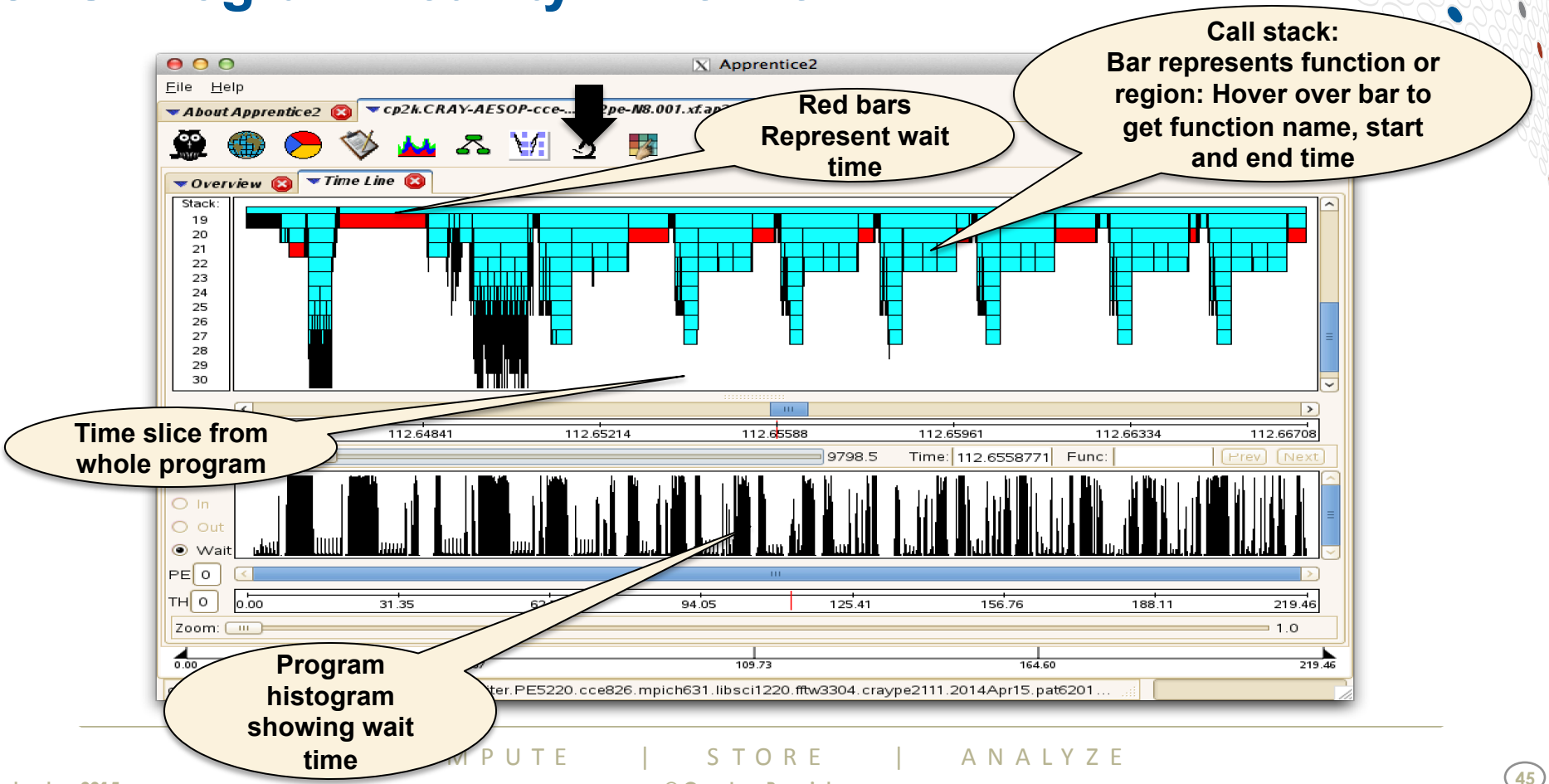
heidi@limited: /h/heidi — ssh — 81×26

```
Table 2:  Profile by Group, Function, and Line

 Samp%  |  Samp   | Imb.   |  Imb.   |Group
        |         | Samp   | Samp%   | Function
        |         |        |         |  Source
        |         |        |         |   Line
        |         |        |         |    PE=HIDE

 100.0% | 8376.9 |    -- |     -- |Total
|-------------------------------------------------------------------------------
|  93.2% | 7804.0 |    -- |     -- |USER
||------------------------------------------------------------------------------
||  51.7% | 4328.7 |    -- |     -- |calc3_
3|        |        |       |        | heidi/DARPA/cache_util/calc3.do300-ijswap.F
||||----------------------------------------------------------------------------
4|||  15.7% | 1314.4 |  93.6 |   6.8% |line.78
4|||  13.9% | 1167.7 |  98.3 |   7.9% |line.79
4|||  14.5% | 1211.6 |  97.4 |   7.6% |line.80
4|||   1.2% |  103.1 |  26.9 |  21.2% |line.93
4|||   1.1% |   88.4 |  22.6 |  20.8% |line.94
4|||   1.0% |   84.5 |  17.5 |  17.6% |line.95
4|||   1.0% |   86.8 |  33.2 |  28.2% |line.96
4|||   1.3% |  105.0 |  23.0 |  18.4% |line.97
4|||   1.4% |  116.5 |  24.5 |  17.7% |line.98
||||============================================================================
                                                              144,1         38%
```

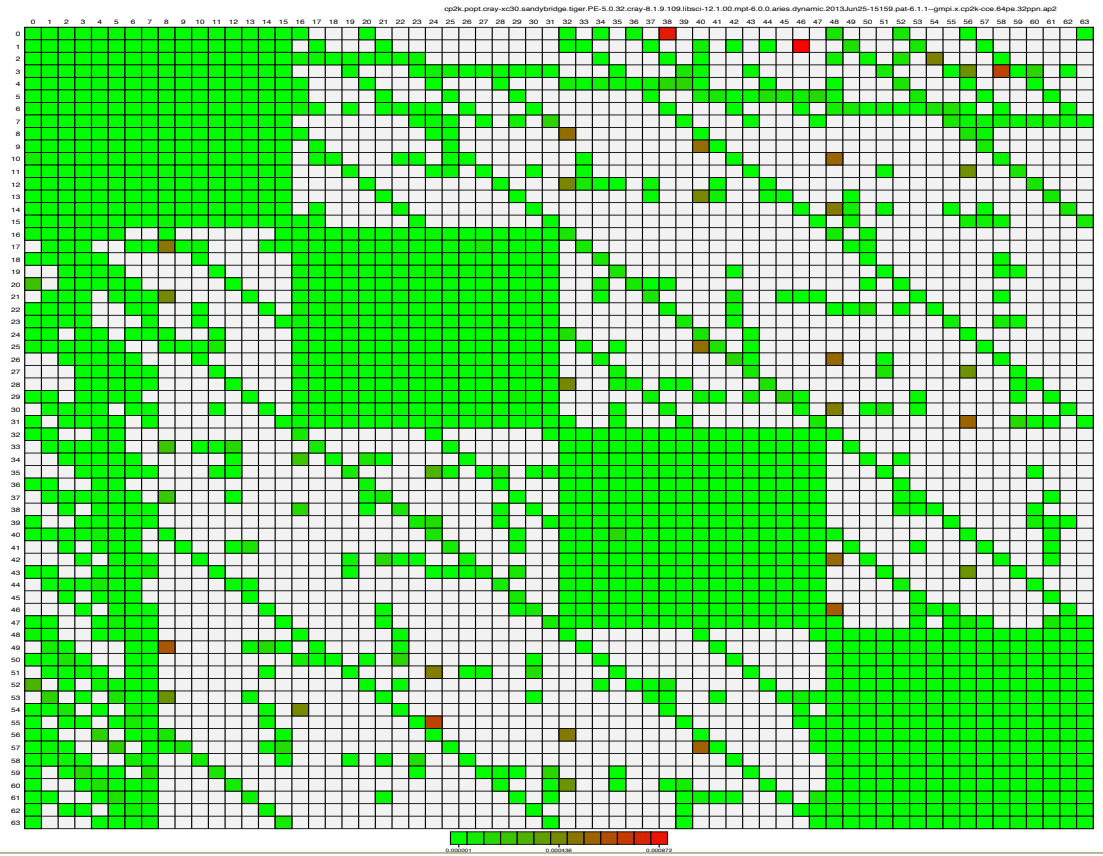
Find Load Imbalance within Program



CPU Program Activity Timeline



What is My Program's Communication Pattern?



MPI Rank Order Observations

```
Table 1:  Profile by Function Group and Function

 Time%  |      Time  |     Imb.   |  Imb.   |  Calls   |Group
        |            |     Time   | Time%   |          | Function
        |            |            |         |          |  PE=HIDE

 100.0% | 463.147240 |        -- |     -- | 21621.0 |Total
|-------------------------------------------------------------------------
|  52.0% | 240.974379 |        -- |     -- | 21523.0 |MPI
||------------------------------------------------------------------------
||  47.7% | 221.142266 | 36.214468 |  14.1% | 10740.0 |mpi_recv
||   4.3% |  19.829001 | 25.849906 |  56.7% | 10740.0 |MPI_SEND
||========================================================================
|  43.3% | 200.474690 |        -- |     -- |    32.0 |USER
||------------------------------------------------------------------------
||  41.0% | 189.897060 | 58.716197 |  23.6% |    12.0 |sweep_
||   1.6% |   7.579876 |  1.899097 |  20.1% |    12.0 |source_
||========================================================================
|   4.7% |  21.698147 |        -- |     -- |    39.0 |MPI_SYNC
||------------------------------------------------------------------------
|   4.3% |  20.091165 | 20.005424 |  99.6% |    32.0 | mpi_allreduce_(sync)
||========================================================================
|   0.0% |   0.000024 |        -- |     -- |    27.0 |SYSCALL
|=========================================================================
```

MPI Rank Order Observations (2)

```
MPI Grid Detection:

    There appears to be point-to-point MPI communication in a 96 X 8
    grid pattern. The 52% of the total execution time spent in MPI
    functions might be reduced with a rank order that maximizes
    communication between ranks on the same node. The effect of several
    rank orders is estimated below.

    A file named MPICH_RANK_ORDER.Grid was generated along with this
    report and contains usage instructions and the Custom rank order
    from the following table.

         Rank     On-Node      On-Node    MPICH_RANK_REORDER_METHOD
        Order    Bytes/PE    Bytes/PE%
                             of Total
                             Bytes/PE

       Custom  2.385e+09       95.55%   3
          SMP  1.880e+09       75.30%   1
         Fold  1.373e+06        0.06%   2
    RoundRobin 0.000e+00        0.00%   0
```

Auto-Generated MPI Rank Order File

```
# The 'Custom' rank order in this file targets nodes with multi-core
# processors, based on Sent Msg Total Bytes collected for:
#
# Program:      /lus/nid00030/heidi/sweep3d/mod/sweep3d.mpi
# Ap2 File:     sweep3d.mpi+pat+27054-89t.ap2
# Number PEs:   48
# Max PEs/Node: 4
#
# To use this file, make a copy named MPICH_RANK_ORDER, and set the
# environment variable MPICH_RANK_REORDER_METHOD to 3 prior to
# executing the program.
#
# The following table lists rank order alternatives and the grid_order
# command-line options that can be used to generate a new order.
…
```

Auto-Generated MPI Rank Order File (2)

```
# The 'USER_Time_hybrid' rank order in this file targets nodes with multi-core
# processors, based on Sent Msg Total Bytes collected for:
#
# Program:      /lus/nid00023/malice/craypat/WORKSHOP/bh2o-demo/Rank/sweep3d/src/sweep3d
# Ap2 File:     sweep3d.gmpi-u.ap2
# Number PEs:   768
# Max PEs/Node: 16
#
# To use this file, make a copy named MPICH_RANK_ORDER, and set the
# environment variable MPICH_RANK_REORDER_METHOD to 3 prior to
# executing the program.
#
0,532,64,564,32,572,96,540,8
,596,72,524,40,604,24,588
104,556,16,628,80,636,56,620
,48,516,112,580,88,548,120,6
12
1,403,65,435,33,411,97,443,9
,467,25,499,105,507,41,475
73,395,81,427,57,459,17,419,
113,491,49,387,89,451,121,48
3
6,436,102,468,70,404,38,412,
14,444,46,476,110,508,78,500
86,396,30,428,62,460,54,492,
118,420,22,452,94,388,126,48
4
129,563,193,531,161,571,225,
539,241,595,233,523,249,603,
185,555
153,587,169,627,137,635,201,
619,177,515,145,579,209,547,
217,611
7,405,71,469,39,437,103,413,
47,445,15,509,79,477,31,501
111,397,63,461,55,429,87,421
,23,493,119,389,95,453,127,4
85
134,402,198,434,166,410,230,
442,238,466,174,506,158,394,
246,474
190,498,254,426,142,458,150,
386,182,418,206,490,214,450,
222,482
128,533,192,541,160,565,232,
525,224,573,240,597,184,557,
248,605
168,589,200,517,152,629,136,
549,176,637,144,621,208,581,
216,613
5,439,37,407,69,447,101,415,15
13,471,45,503,29,479,77,511
53,399,85,431,21,463,61,391,
109,423,93,455,117,495,125,4
87
2,530,34,562,66,538,98,522,1
0,570,42,554,26,594,50,602
18,514,74,586,58,626,82,546,
106,634,90,578,114,618,122,6
10
135,315,167,339,199,347,259,
307,231,371,239,379,191,331,
247,299
175,363,159,323,143,355,255,
291,207,275,183,283,151,267,
215,223
133,406,197,438,165,470,229,
414,245,446,141,478,237,502,
253,398
157,510,189,462,173,430,205,
390,149,422,213,454,181,494,
221,486
130,316,260,340,194,372,162,
348,226,308,234,380,242,332,
250,300
202,364,186,324,154,356,138,
292,170,276,178,284,210,218,
268,146
4,535,36,543,68,567,100,527,
12,599,44,575,28,559,76,607
52,591,20,631,60,639,84,519,
108,623,92,551,116,583,124,6
3,440,35,432,67,400,99,408,1
1,464,43,496,27,472,51,504
19,392,75,424,59,456,83,384,
107,416,91,488,115,448,123,4
80
132,401,196,441,164,409,228,
433,236,465,204,473,244,393,
188,497
252,505,140,425,212,457,156,
385,172,417,180,449,148,489,
220,481
131,534,195,542,163,566,227,
526,235,574,203,598,243,558,
187,606
251,590,211,630,179,638,139,
622,155,550,171,518,219,582,
147,614
761,660,737,652,705,668,745,
692,673,700,641,684,713,644,
753,724
729,732,681,756,721,716,764,
676,697,748,689,657,740,665,
649,708
760,528,736,536,704,560,744,
520,672,568,712,592,752,552,
640,600
728,584,680,624,720,512,696,
632,688,616,664,544,608,656,
648,576
762,659,738,651,706,667,746,
643,714,691,674,699,754,683,
730,723
722,731,763,658,642,755,739,
675,707,650,682,715,698,666,
690,747
257,345,265,313,281,305,273,
337,609,369,577,377,617,329,
513,529
545,297,633,361,625,321,585,
537,601,289,553,353,593,521,
569,561
256,373,261,341,264,349,280,
317,272,381,269,309,285,333,
277,365
352,301,320,325,288,357,328,
304,360,312,376,293,296,368,
336,344
258,338,266,346,282,314,274,
370,766,306,710,378,742,330,
678,362
646,298,750,322,718,354,758,
290,734,662,686,670,726,702,
694,654
262,375,263,343,270,311,271,
351,286,319,278,342,287,350,
279,374
294,318,358,383,359,310,295,
382,326,303,327,367,366,335,
302,334
765,661,709,663,741,653,711,
669,767,655,743,671,749,695,
679,703
677,727,751,693,647,701,717,
687,757,685,733,725,719,735,
645,759
```

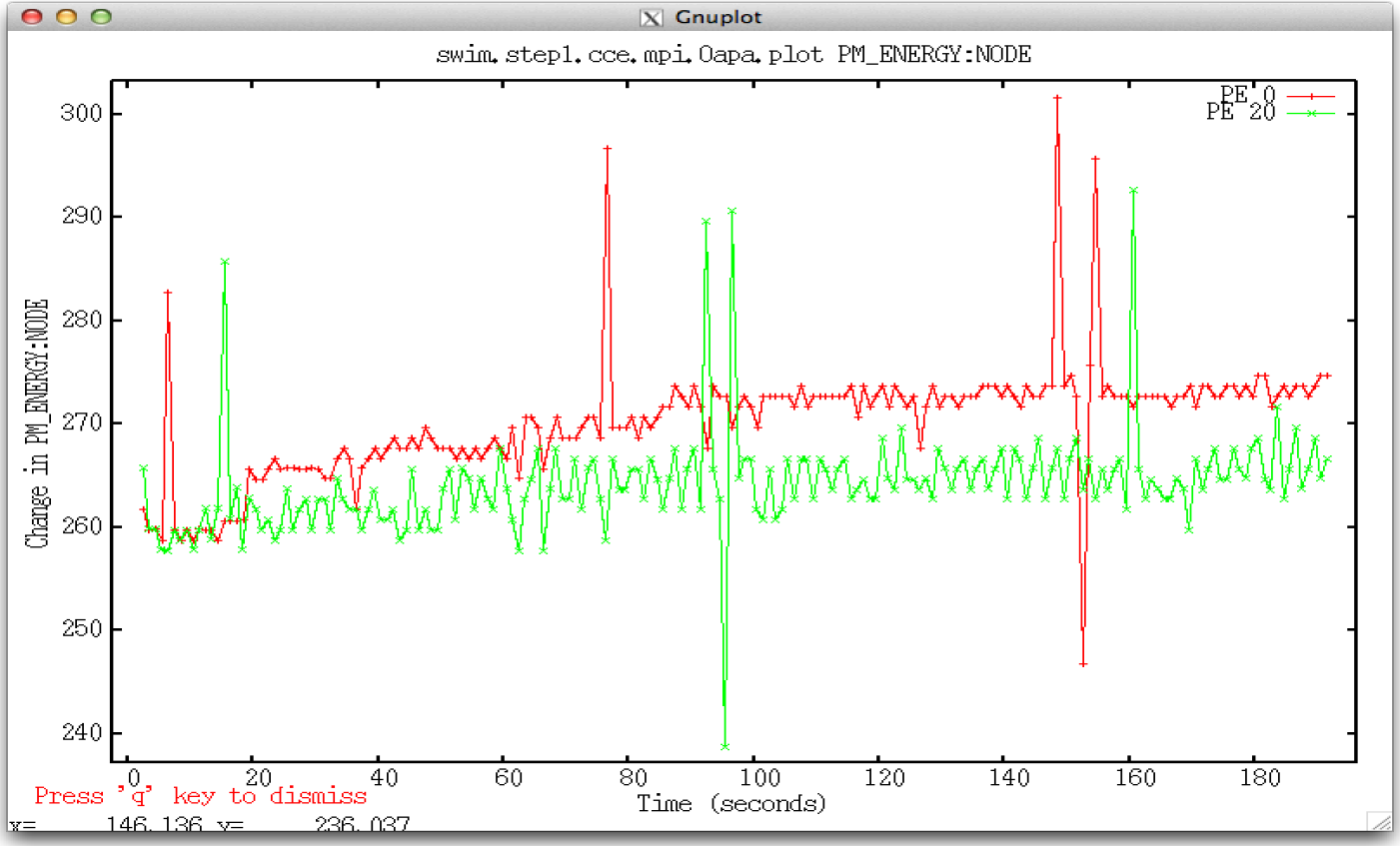
Monitoring Power on Cray XC Systems

- Feedback on performance and power consumption will be key to understanding the behavior of an applications on future systems.
- Cray performance tools support both Intel RAPL and Cray Power Management monitoring on Cray XC systems
- Provide total energy or power consumption for an application as well as energy/power over time

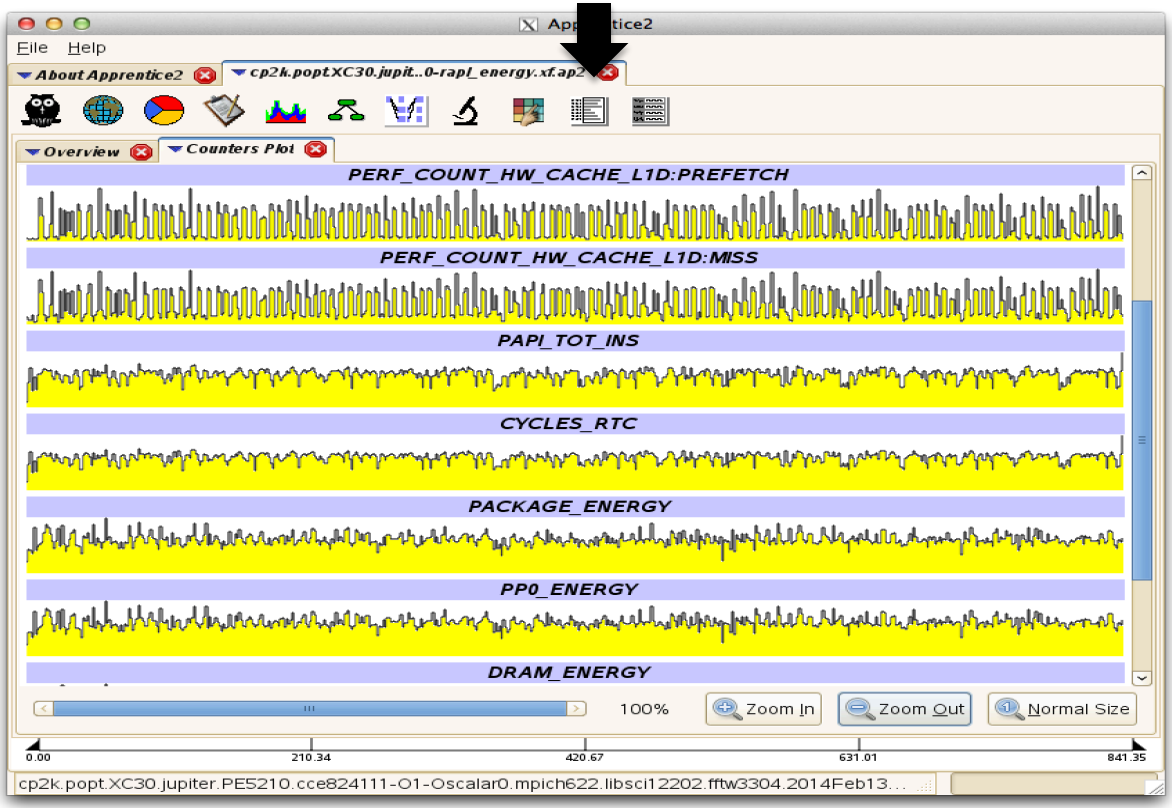
Energy / Power Statistics

```
CrayPat/X:  Version 6.2.1 Revision 13075 (xf 13045)  08/27/14 12:39:06
Experiment:                  lite  lite/sample_profile
Number of PEs (MPI ranks):      4
Numbers of PEs per Node:        1  PE on each of  4  Nodes
Numbers of Threads per PE:      4
Number of Cores per Socket:    10
Execution start time:  Tue Sep  9 10:53:59 2014
System name and speed:  kay-esl 3001 MHz

Process Time:       34.135  secs        8.534 secs per PE
High Memory:          2298 MBytes     574.423 MBytes per PE
MFLOPS (aggregate):  6.132 M/sec        1.533 M/sec per PE
I/O Read Rate:      61.494 MBytes/sec
I/O Write Rate:      1.526 MBytes/sec
Ave CPU Energy:       2471 joules     617.750 joules per node
Ave CPU Power:      72.389 watts       18.097 watts per node
Ave ACC Energy:       1392 joules        348 joules per node
Ave ACC Power:      40.779 watts       10.195 watts per node
. . .
```

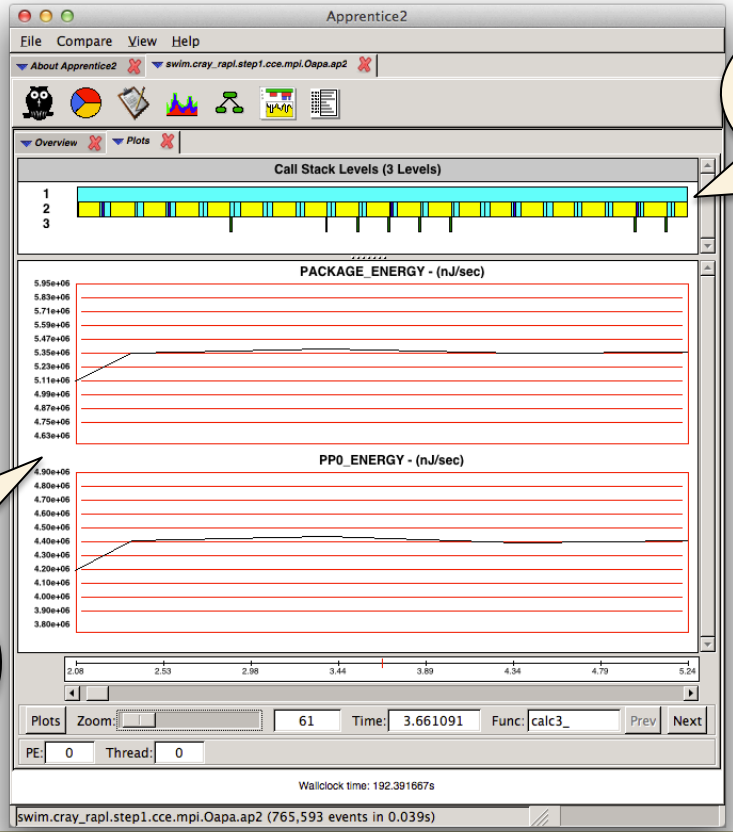
Change in Energy Per Node Over Time



Hardware Counters Plot with Energy



Energy Consumption Over Time



Call stack: Bar represents function or region: Hover over bar to get function name, start and end time

Plots of energy consumed by the socket and by the cores within a socket over time. Can also show memory high water mark, etc.

Creating Hybrid Codes, Why Should You Care?

- **For the next decade (at least) all HPC systems will have the same basic architecture:**
  - Communication between nodes
    - Application developers will continue to use MPI between nodes or sockets
      - Maybe SHMEM, UPC, Coarray
  - Shared memory programming paradigm (or PGAS) on the node
    - MPI only will not do
  - Vectorization at the low level looping structures
    - SSE, AVX, ...
    - GPU, MIC, ...
    - etc
- **Hybridizing a code gives many performance advantages**
  - Resource contention (network, node memory, ...)
- **While there is a potential acceptance of new languages for addressing multiple levels of parallelism, most developers cannot afford this approach until they are assured that the new language will be accepted and the generated code is within a reasonable performance range**

When to Move to a Hybrid Programming Model

- **When code is network bound**
  - Increased MPI collective and point-to-point wait times
- **When MPI starts leveling off**
  - Too much memory used, even if on-node shared communication is available
  - As the number of MPI ranks increases, more off-node communication can result, creating a network injection issue
- **When contention of shared resources increases**

Moving to a Hybrid Code: a Four-Task Approach

1. **Identify potential loops to accelerate**
   - **Determine where to add additional levels of parallelism**
     - Assumes the MPI application is functioning correctly
     - Find top work-intensive loops

2. **Perform parallel analysis and scoping on identified loops**
   - **Split loop work among threads**
     - Do parallel analysis and restructuring on targeted high level loops

3. **Add OpenMP**
   - **Add parallel directives and acceleration extensions**
     - Insert OpenMP directives
     - Verify application and check for performance improvements

4. **Analyse performance for further optimization, specifically vectorization of innermost loops**
   - **We want a performance-portable application at the end**

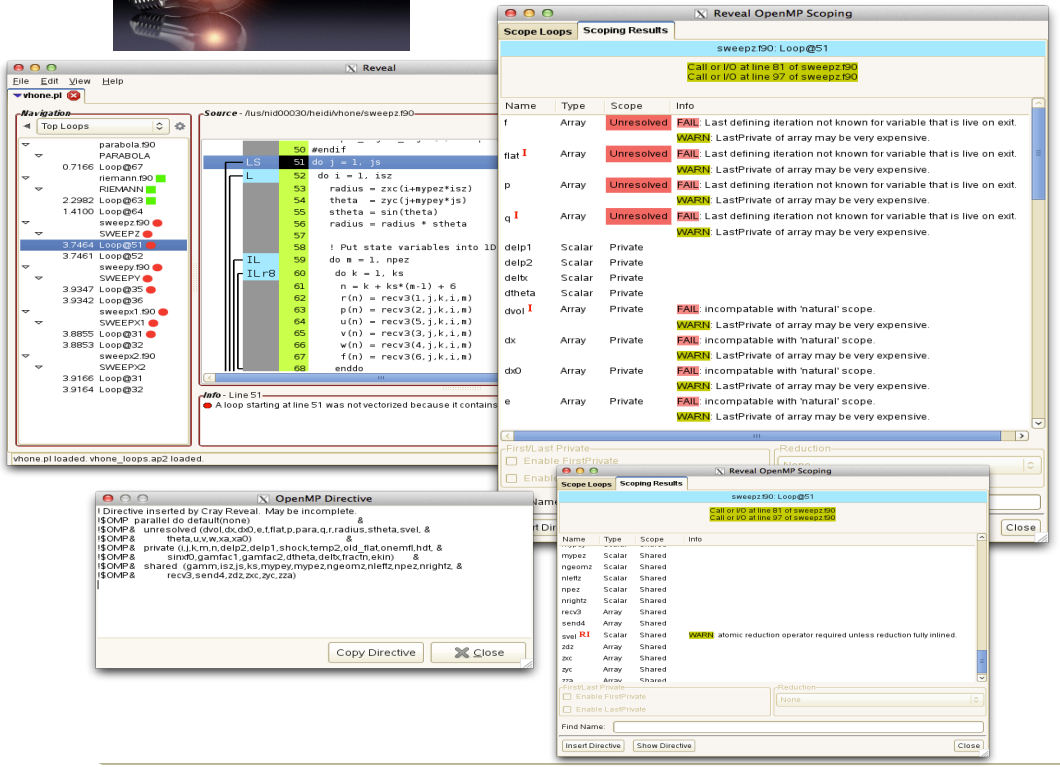
The Problem – How Do I Parallelize This Loop?

- How do I know this is a good loop to parallelize?
- What prevents me from parallelizing this loop?
- Can I get help building a directive?

```
subroutine sweepz
…
do j = 1, js
 do i = 1, isz
   radius = zxc(i+mypez*isz)
   theta  = zyc(j+mypey*js)
   do m = 1, npez
    do k = 1, ks
     n = k + ks*(m-1) + 6
     r(n) = recv3(1,j,k,i,m)
     p(n) = recv3(2,j,k,i,m)
     u(n) = recv3(5,j,k,i,m)
     v(n) = recv3(3,j,k,i,m)
     w(n) = recv3(4,j,k,i,m)
     f(n) = recv3(6,j,k,i,m)
    enddo
   enddo
   …
   call ppmlr
   do k = 1, kmax
     n = k + 6
     xa (n) = zza(k)
     dx (n) = zdz(k)
     xa0(n) = zza(k)
     dx0(n) = zdz(k)
     e  (n) = p(n)/(r(n)*gamm)+0.5 &
        *(u(n)**2+v(n)**2+w(n)**2)
   enddo
   call ppmlr
…
 enddo
enddo
```

```
subroutine ppmlr

call boundary
call flatten
call paraset(nmin-4, nmax+5, para, dx, xa)

call parabola(nmin-4,nmax+4,para,p,dp,p6,pl,flat)
call parabola(nmin-4,nmax+4, para,r,dr,r6,rl,flat)
call parabola(nmin-4,nmax+4,para,u,du,u6,ul,flat)

call states(pl,ul,rl,p6,u6,r6,dp,du,dr,plft,ulft,&
            rlft,prgh,urgh,rrgh)
call riemann(nmin-3,nmax+4,gam,prgh,urgh,rrgh,&
             plft,ulft,rlft pmid umid)
call evolve(umid, pmid)  ← contains more calls

call remap ← contains more calls

call volume(nmin,nmax,ngeom,radius,xa,dx,dvol)

call remap ← contains more calls

return
end
```

Simplifying the Task with Reveal



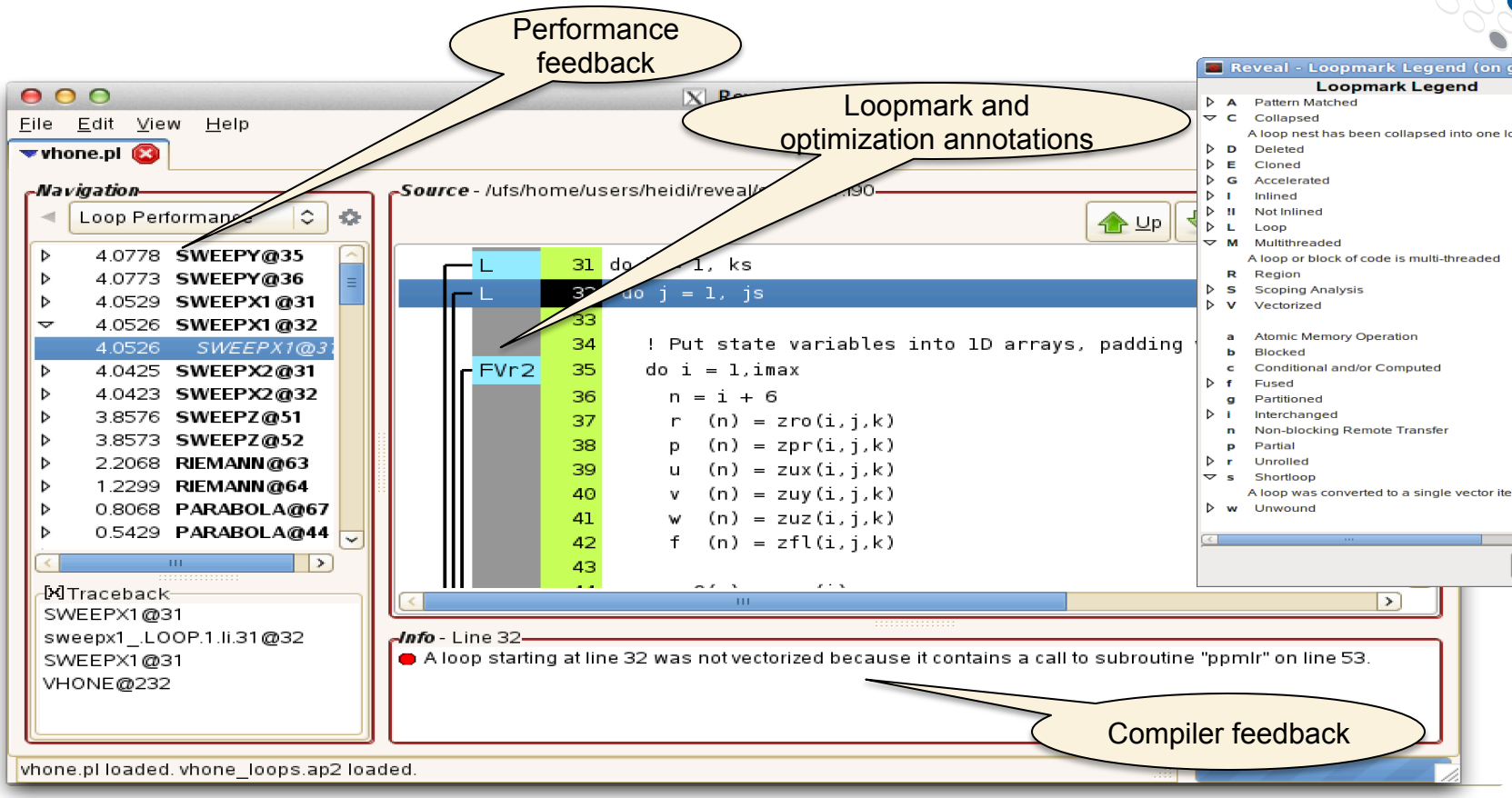
- Navigate to relevant loops to parallelize
- Identify parallelization and scoping issues
- Get feedback on issues down the call chain (shared reductions, etc.)
- Optionally insert parallel directives into source
- Validate scoping correctness on existing directives

Hybridization Step 1: Loop Work Estimates

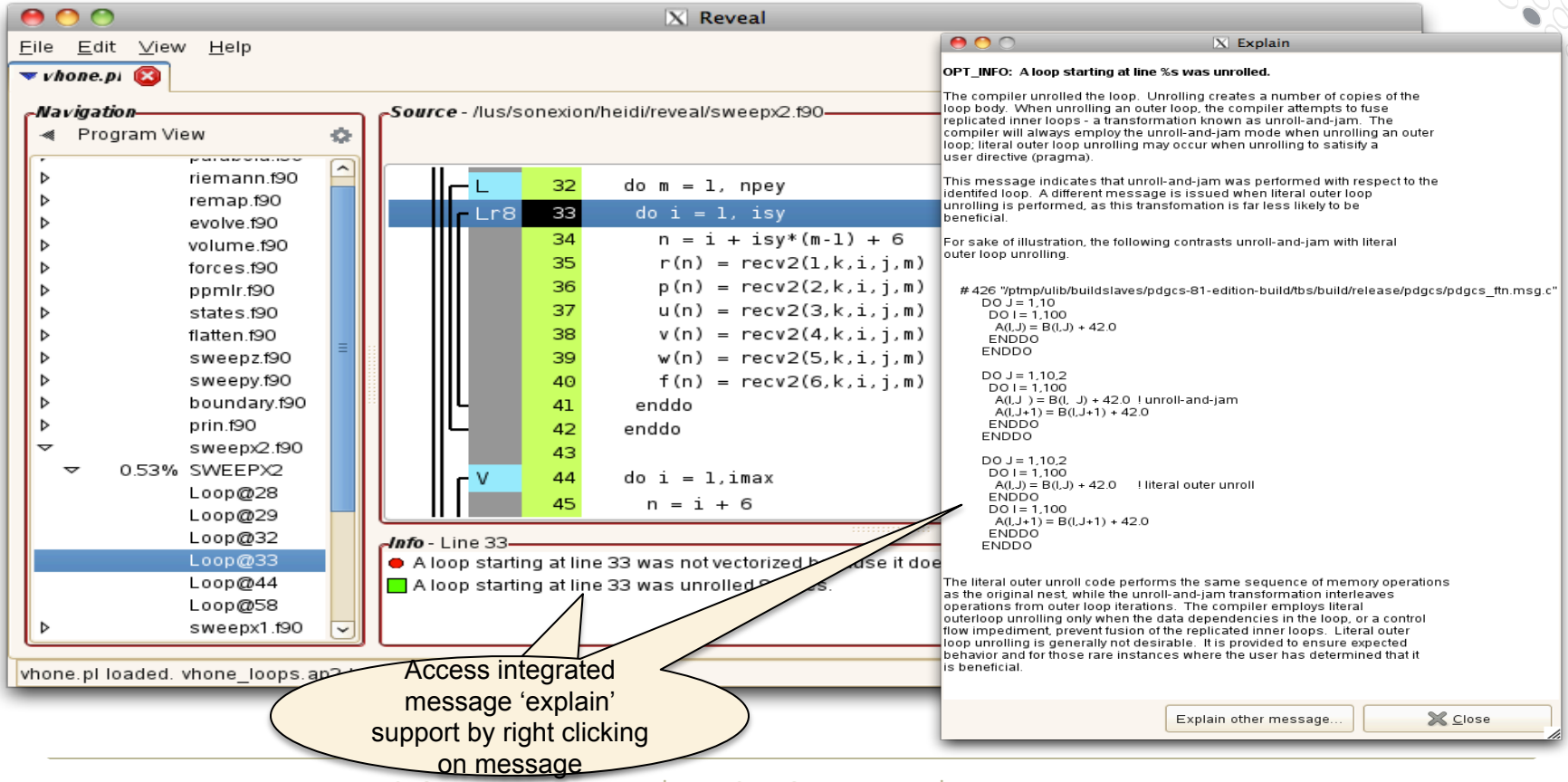
Gather loop statistics using CCE and the Cray performance tools to determine which loops have the most work

- **Helps identify high-level serial loops to parallelize**
  - Based on runtime analysis, approximates how much work exists within a loop
- **Provides the following statistics**
  - Min, max and average trip counts
  - Inclusive time spent in loops
  - Number of times a loop was executed

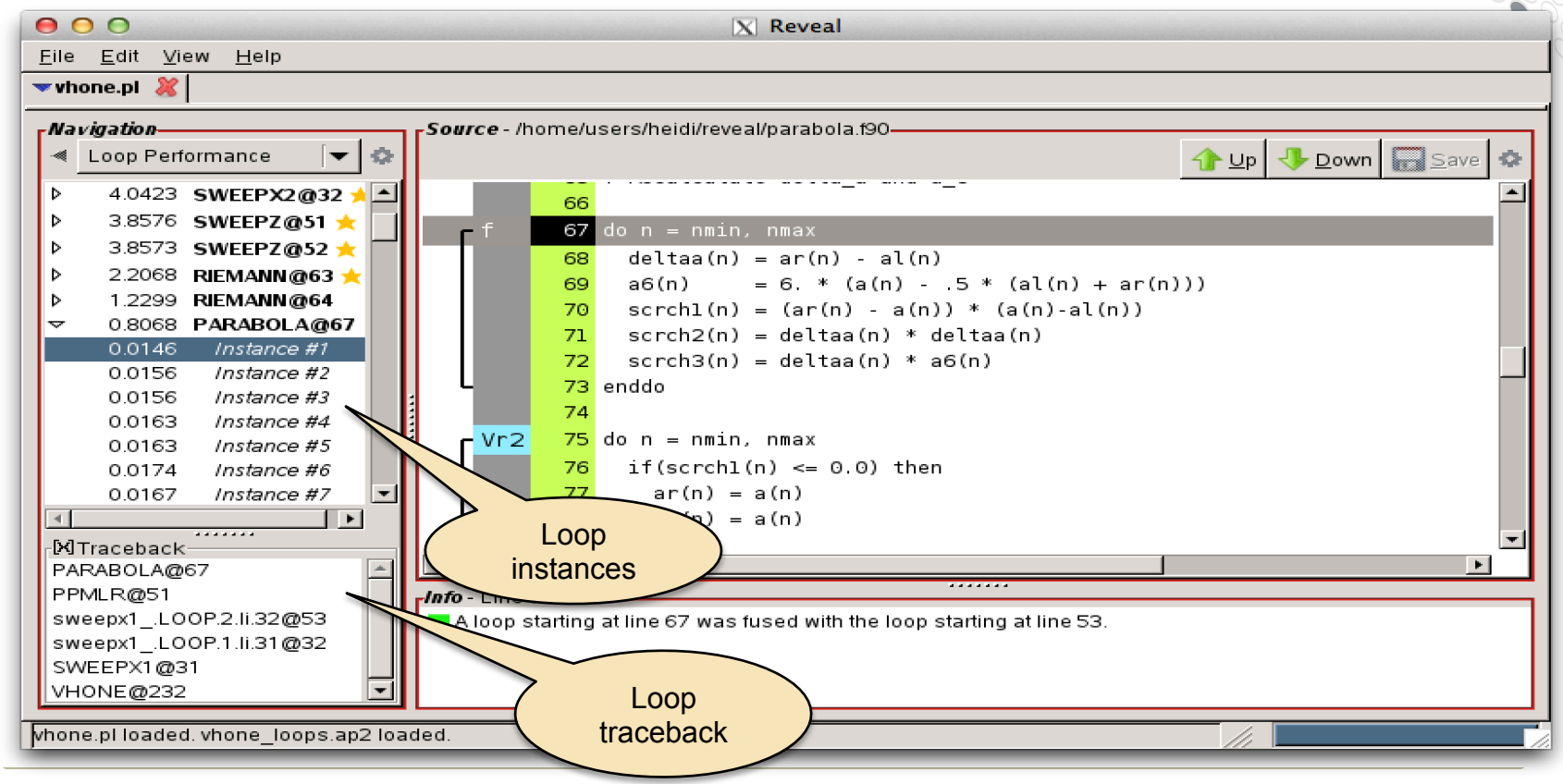
Visualize Compiler and Performance Information



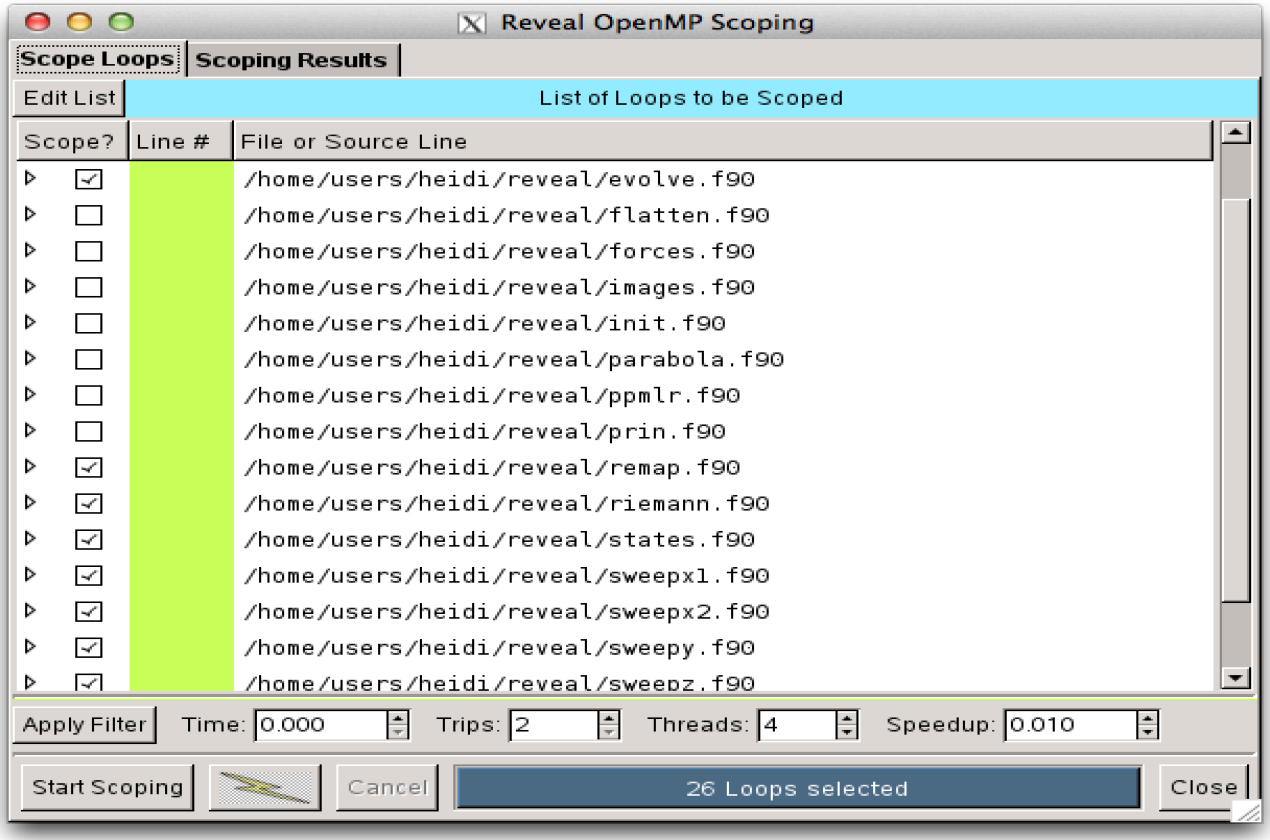
Access Compiler Message Information



View Loops through Call Chain



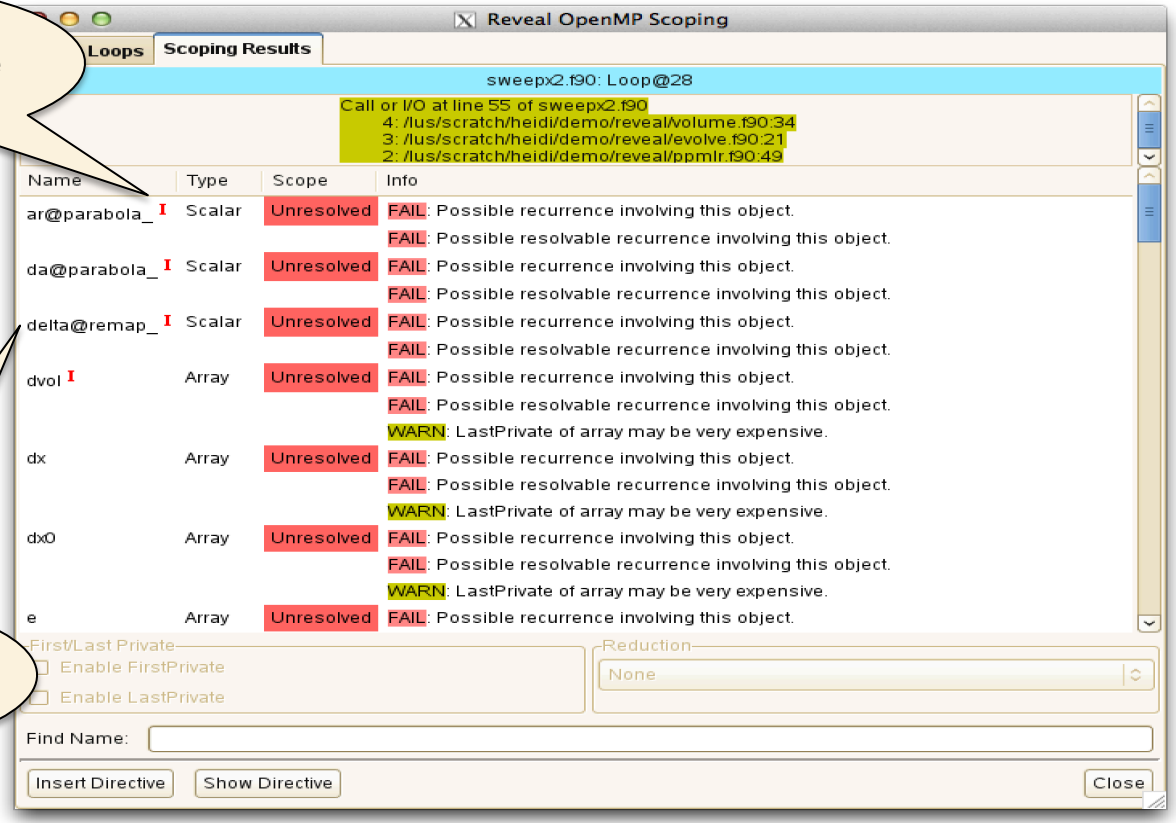
Hybridization Step 2: Scope Selected Loop(s)



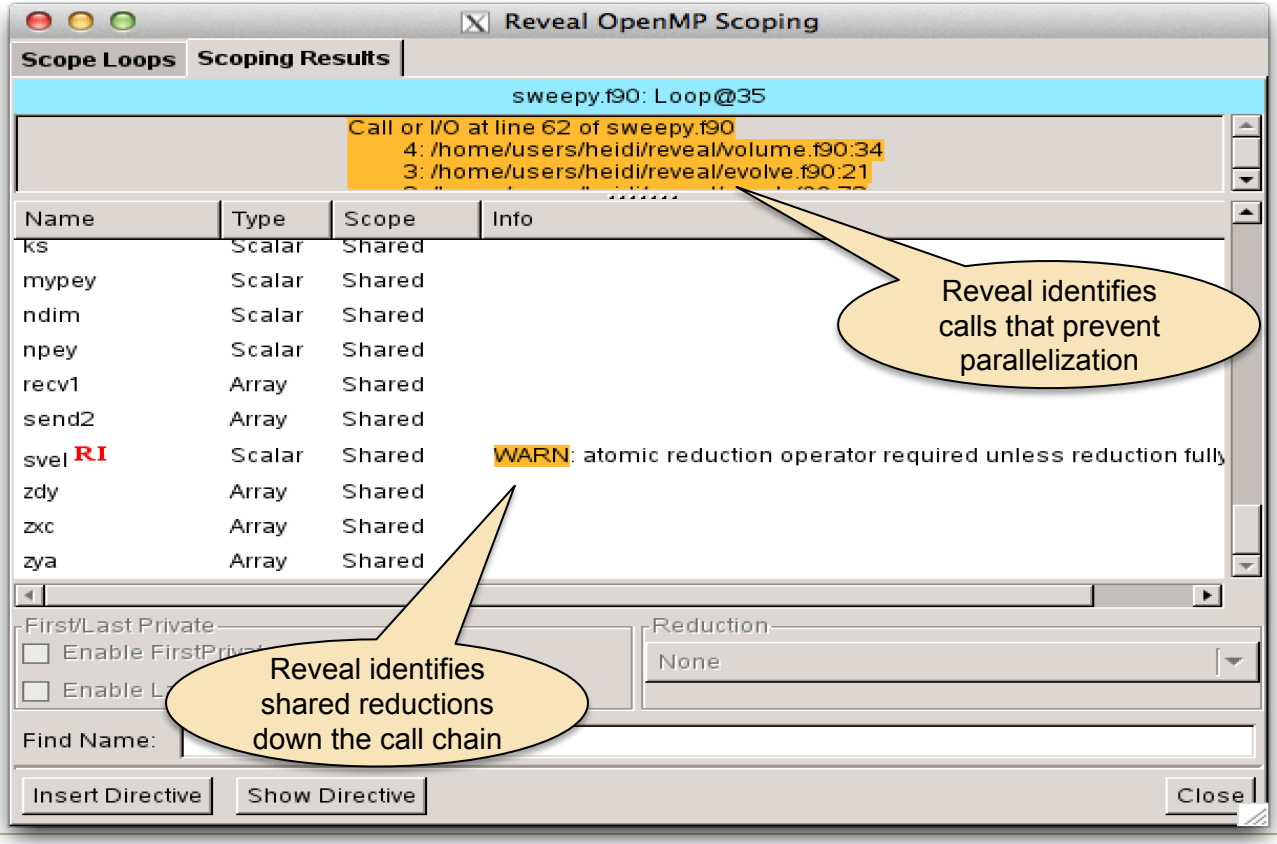
Review Feedback on Scoping Results

Variable from inlining – hover over 'I' to see what symbol means

See where variable came from (@function_name)



Review Parallelization Issues

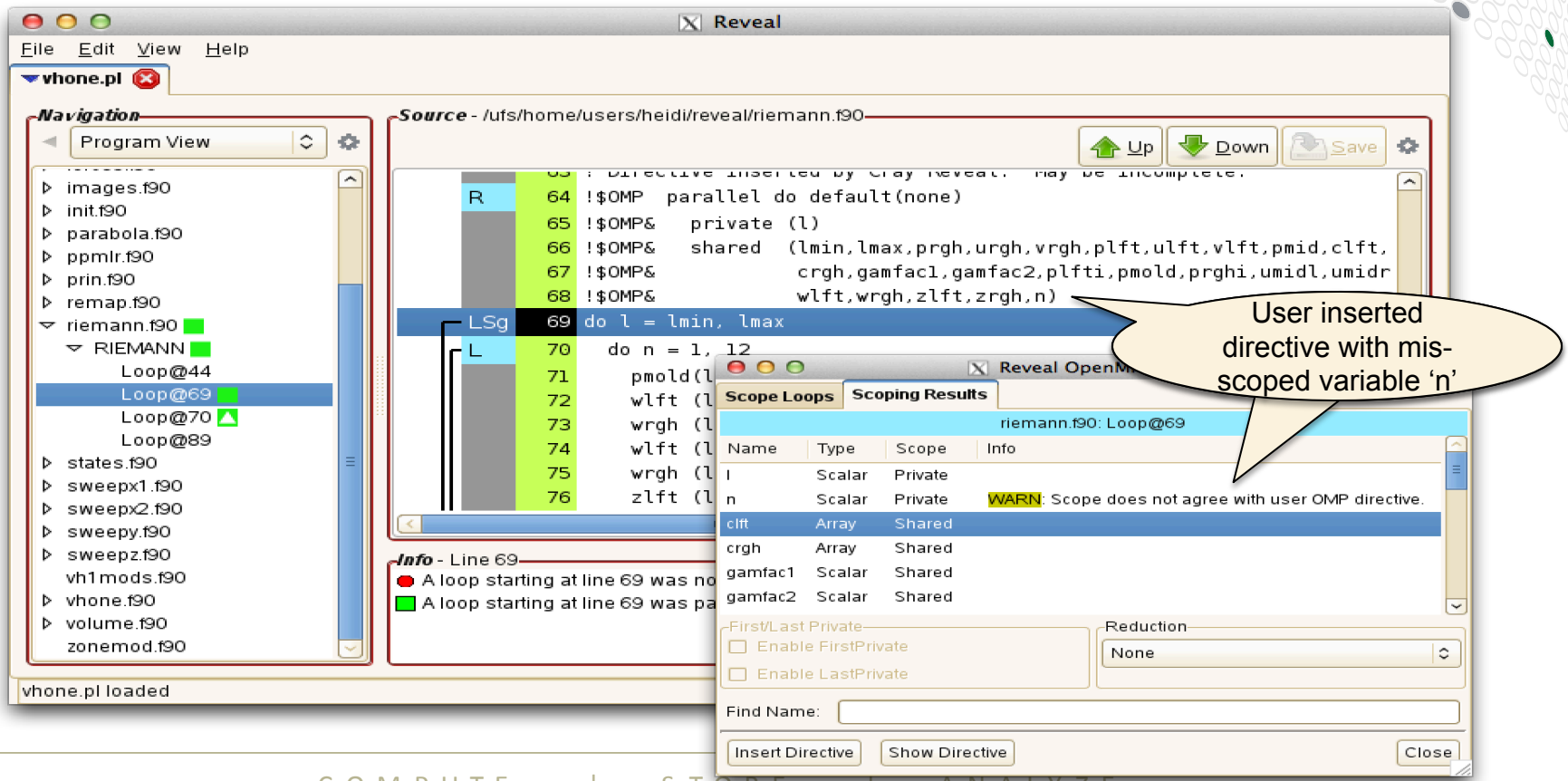


Hybridization Step 3: Generating OpenMP Directives

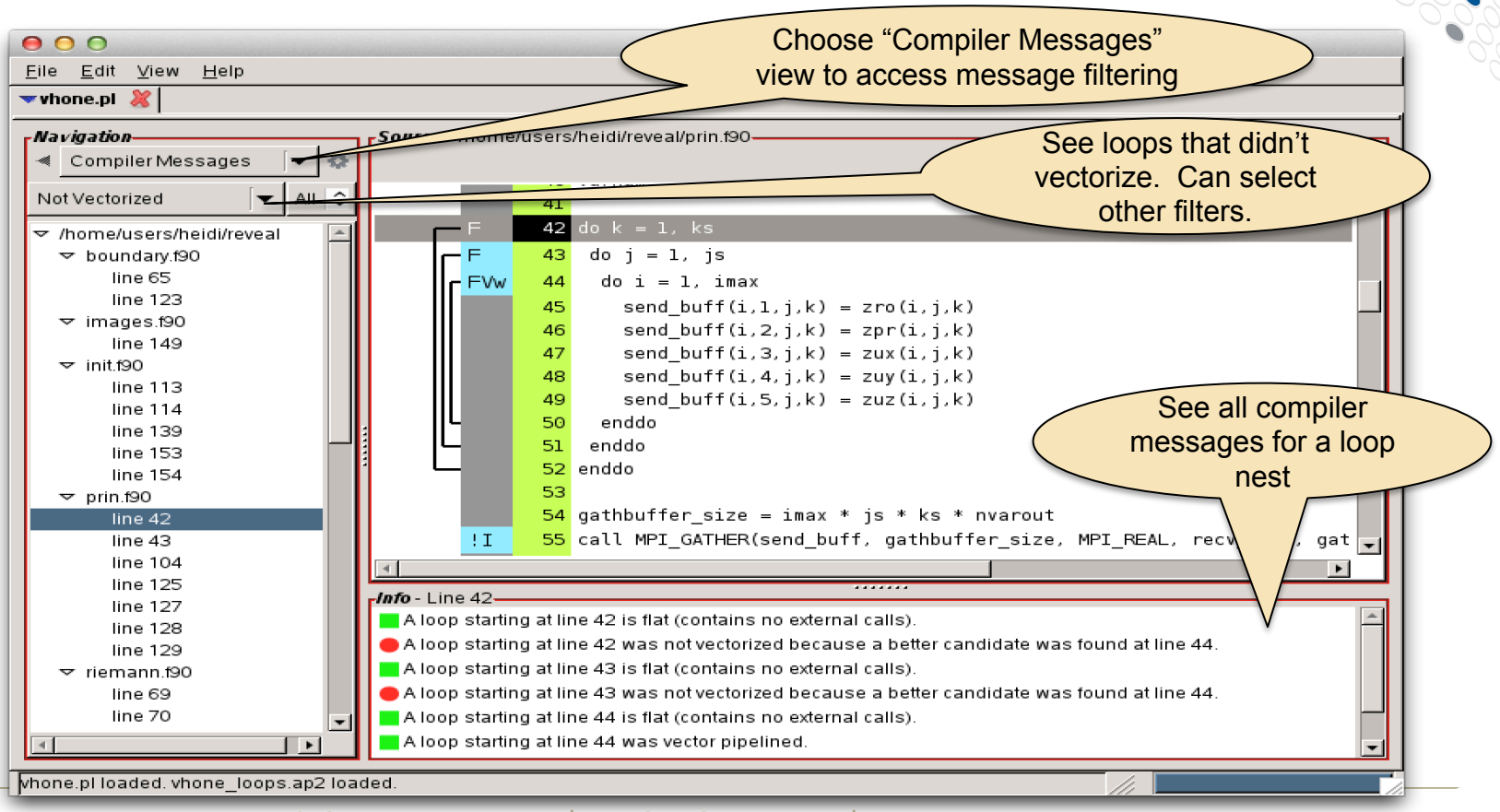
```
! Directive inserted by Cray Reveal.  May be incomplete.
!$OMP  parallel do default(none)                                    &
!$OMP&   unresolved (dvol,dx,dx0,e,f,flat,p,para,q,r,radius,svel,u,v,w,  &
!$OMP&          xa,xa0)                                             &
!$OMP&   private (i,j,k,m,n,$$_n,delp2,delp1,shock,temp2,old_flat,   &
!$OMP&          onemfl,hdt,sinxf0,gamfac1,gamfac2,dtheta,deltx,fractn,  &
!$OMP&          ekin)                                               &
!$OMP&   shared  (gamm,isy,js,ks,mypey,ndim,ngeomy,nlefty,npey,nrighty, &
!$OMP&          recv1,send2,zdy,zxc,zya)
do k = 1, ks
 do i = 1, isy
  radius = zxc(i+mypey*isy)

  ! Put state variables into 1D arrays, padding with 6 ghost zones
  do m = 1, npey
   do j = 1, js
    n = j + js*(m-1) + 6
    r(n) = recv1(1,k,j,i,m)
    p(n) = recv1(2,k,j,i,m)
    u(n) = recv1(4,k,j,i,m)
    v(n) = recv1(5,k,j,i,m)
    w(n) = recv1(3,k,j,i,m)
    f(n) = recv1(6,k,j,i,m)
   enddo
  enddo

  do j = 1, jmax
    n = j + 6
```

Reveal generates OpenMP directive with illegal clause marking variables that need addressing

Or Validate User Inserted Directives



Hybridization Step 4: Performance Analysis



Hybridization Step 4: Performance Analysis

```
================  Observations and suggestions  ========================
D1 cache utilization:
    61.7% of total execution time was spent in 1 functions with D1 cache
    hit ratios below the desirable minimum of 90.0%. Cache utilization
    might be improved by modifying the alignment or stride of references
    to data arrays in these functions.

        D1    Time%   Function
     cache
       hit
     ratio

       74.3%    61.7%  calc3_

D1 + D2 cache utilization:
    61.7% of total execution time was spent in 1 functions with combined
    D1 and D2 cache hit ratios below the desirable minimum of 97.0%.
    Cache utilization might be improved by modifying the alignment or
    stride of references to data arrays in these functions.

     D1+D2    Time%   Function
     cache
       hit
     ratio

       96.6%    61.7%  calc3_
…
```

Example: VH1 – Astrophysics Code

- VH1 is written with high level loops and complex decision processes.
- Ported to hybrid MPI + OpenMP using Reveal
- Reveal was able to identify
  - storage conflicts
  - private variables in modules
  - reductions down the call chain that require critical regions
- Scoping was performed in seconds where it would have taken weeks to get correct without Reveal

Questions ?