

# 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.*

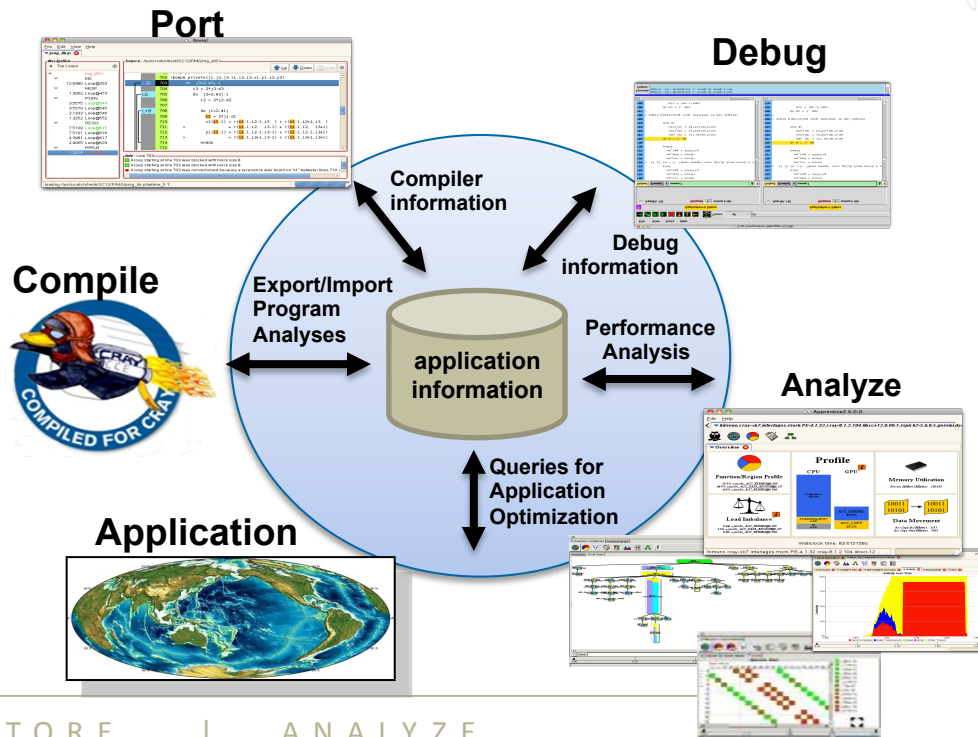
*The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, URIKA and YARCDATA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The 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.*

*Copyright 2015 Cray Inc. ©*

# 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

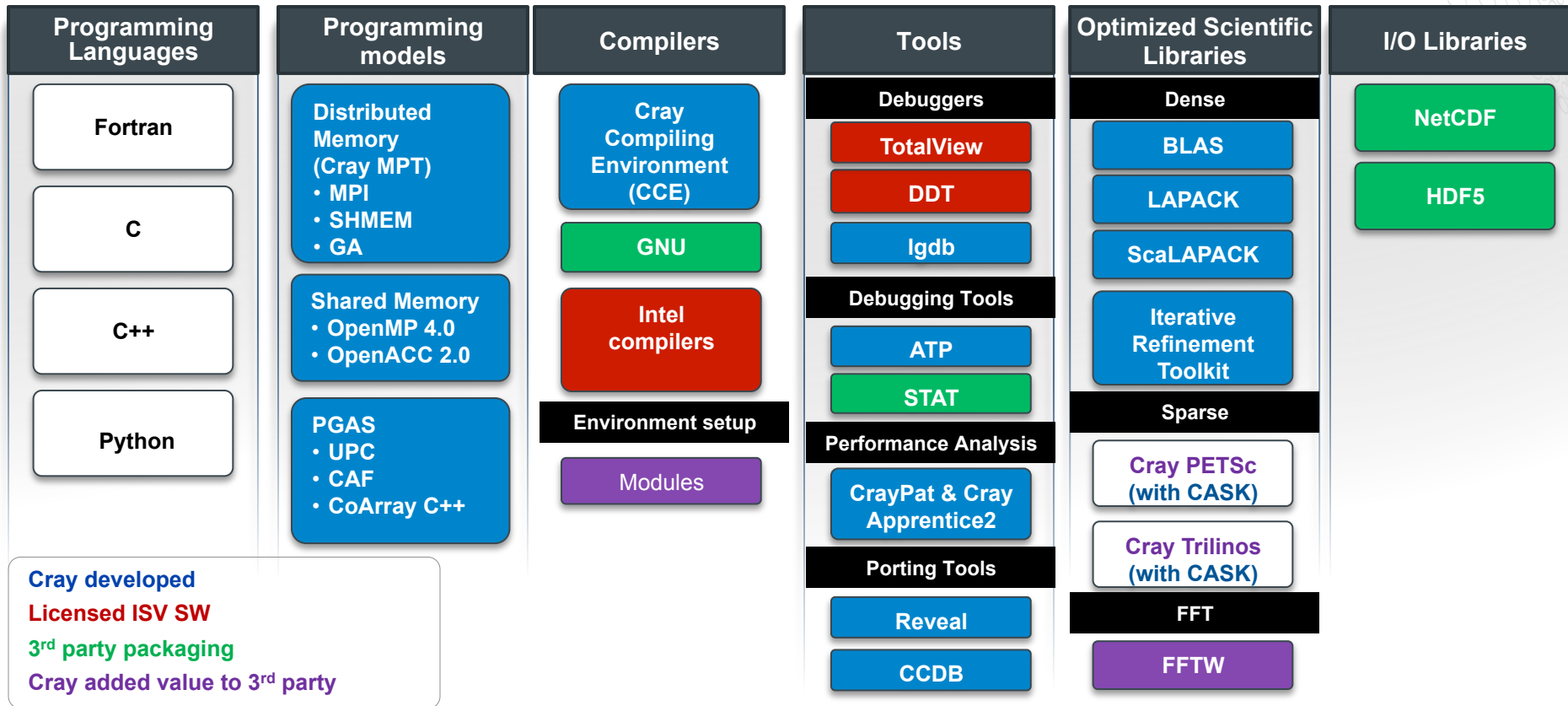
- **Performance**

- 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



COMPUTE | STORE | ANALYZE

# 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 3<sup>rd</sup> party) is managed using the GNU module command**
  - Cray provides utility for sites to create modules for 3<sup>rd</sup> 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.

- **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



## Dense

BLAS

LAPACK

ScaLAPACK

IRT

## Sparse

Trilinos

PETSc

CASK

## FFT

FFTW

COMPUTE | STORE | ANALYZE

# 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](mailto: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 !!

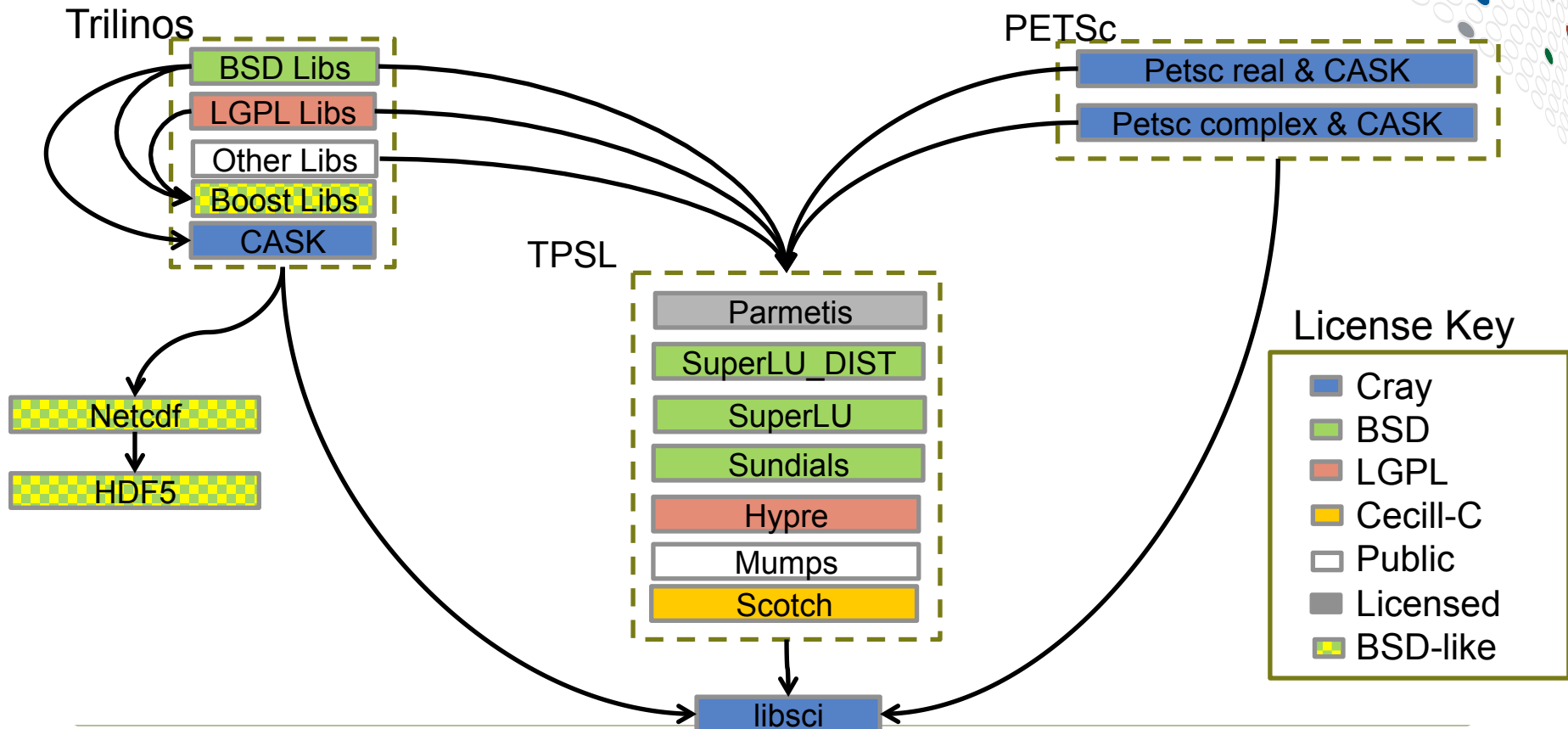
# Cray Adaptive Sparse Kernel (CASK)



CRAY®

- 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



COMPUTE | STORE | ANALYZE

# 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
  - `-WI, -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 tree
    - **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
      - 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



- **Stack trace sampling and analysis for large scale applications**

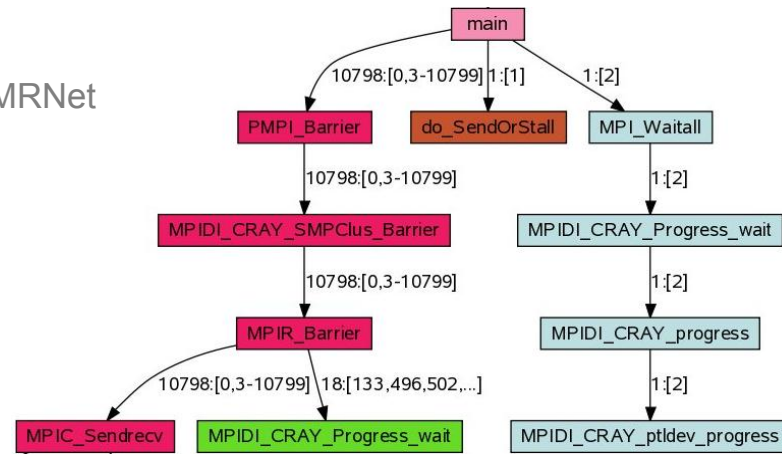
- Sample application stack traces
- Scalable generation of a single, merged, stack backtrace tree
  - A comprehensible view of the entire application
  - Discover equivalent process behavior
    - Group similar processes
    - Reduce number of tasks to debug
  - 128K processes analyzed in 2.7 seconds, 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

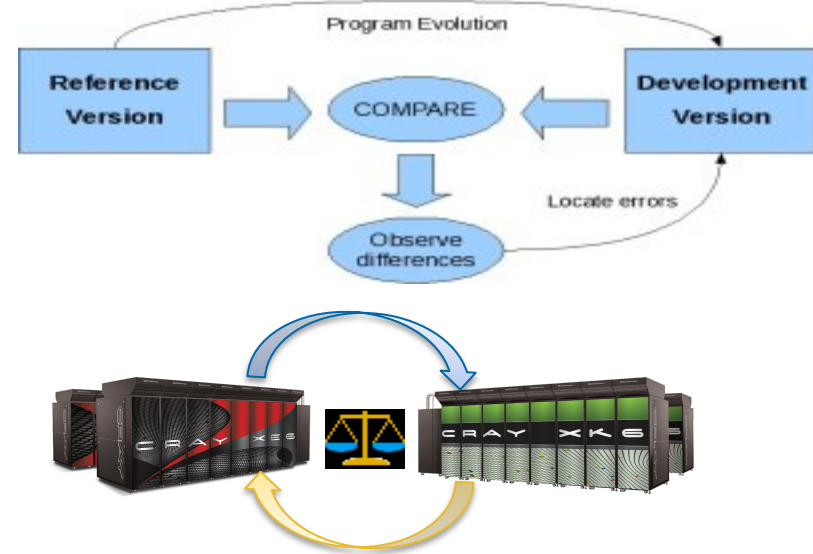
- **ATP is able to hold a dying application in stasis in order to allow the user to attach to it with a debugger**
  - To do so, set the **ATP\_HOLD\_TIME** environment variable to the number of minutes desired
- **Once attached, the debugging session can last as long as the batch system allows**
  - 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**



# Cray Comparative Debugger



Cray Comparative Debugger (CCDB)

File View Tools Help

Focus: all

### Application-0 Status

App0{0..15} Stopped driver.f:106

### Application-1 Status

App1{0..15} Stopped sweep.f:231

#### Output Breakpt + driver.f

```
94:      mm_jbc = 1
95:      endif
96:      if (kbc.ne.0) then
97:        it_kbc = it
98:        jt_kbc = jt
99:        mm_kbc = mm
100:     else
101:       it_kbc = 1
102:       jt_kbc = 1
103:       mm_kbc = 1
104:     endif
105:     if (myid .eq. 1) then
107:       print *, 'SWERP3D - Method 5 - ',
108:         & ' Pipelined Wavefront with Line-Recurs.
109:       print *, 'Version 2.2b'
110:       print 100, isn, isct, mm, nm, it_g, jt_g, kt
111:     100  format(' S', i1, 'P', i1, 3x, '-', i1, i2, ' angles/oct
112:     & i3, ' moments' /,
113:     & ' global grid: ', i5, 'x', i5, 'x', i5)
```

#### Output Breakpt + sweep.f

```
220:      k0 = 1 + (kk-1)*mk
221:      k1 = min (k0+mk-1, kt)
222:      nk = k1 - k0 + 1
223:     else
224:      k0 = kt - (kk-1)*mk
225:      k1 = max (k0-mk+1, 1)
226:      nk = k0 - k1 + 1
227:     endif
228:
229: ! this could be *nk* instead of *mk* if all phi(i,j) (b,
230: ! were dimensioned with nmi as the second dimension
231:     nib = jt*mk*nmi
232:     njb = it*mk*nmi
233:
234: c I-inflows for block (i=i0 boundary)
235: c
236:     if (ew_rcv .ne. 0) then
237:       call rcv_real(ew_rcv, phiib, nib, ew_t
238:     else
239:     if (i2.lt.0 .or. ibc.eq.0) then
```

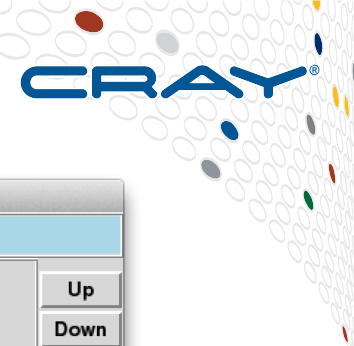
#### Console Output

```
App1(0..15): Temporary breakpoint 1: file sweep.f, line 231.
App1(0..15): Breakpoint 1, sweep at sweep.f:231
```

COMPUTE | STORE | ANALYZE

© Cray Inc. Proprietary

# CCDB - Comparison



CCDB Comparison

Application-0      Application-1

App0{0..15}      sweep.f:160      App1{0..15}      sweep.f:160      Up      Down

	Name or Expression	Type	Results	Type Template	App-0 Decomp	App-1 Decomp	Op	Eps
<input type="checkbox"/>	jkm	INTEGER*4	Click to see results				==	e
<input type="checkbox"/>	mio	INTEGER*4	Click to see results				==	e
<input type="checkbox"/>	nk	INTEGER*4	Click to see results				==	e
<input type="checkbox"/>	ti	REAL*8	Click to see results				==	e
<input type="checkbox"/>	tj	REAL*8	Click to see results				==	e
<input type="checkbox"/>	weta	REAL*8 (6)	Click to see results		None	None	==	e
<input type="checkbox"/>	wmu	REAL*8 (6)	Click to see results		None	None	==	e

Compare    Add Comparison    Result Filter:  all  fail  warn  pass  todo    Close

# 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



File Help
▼ About Apprentice2 x
▼ sweep3d.gmpi-u.ap2 x

▼ Overview x

### Function/Region Profile

63.9% = mpi\_recv  
29.2% = sweep\_  
3.2% = mpi\_allreduce\_(sy

## Profile

### CPU

### Memory Utilization

Process HiMem (MBytes) 658.835

### Load Imbalance

24.76s = mpi\_recv  
79.40s = sweep\_  
10.20s = mpi\_allreduce\_(sy

Observations and suggestions

**NPI Grid Detection:**

There appears to be point-to-point MPI communication in a 96 X 8 grid pattern. The execution time spent in MPI functions might be reduced with a rank order that maximises 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 Order	On-Node Bytes/FE	On-Node Bytes/FE% of Total Bytes/FE	MPICH_RANK_ORDER_METHOD
Custom	1.851e+012	96.56% 3	
SDF	1.455e+012	76.08% 1	
Fold	1.056e+009	0.06% 2	
RoundRobin	0.000e+000	0.00% 0	

**Metric-Based Rank Order:**

When the use of a shared resource like memory bandwidth is unbalanced across nodes, total execution time may be reduced with a rank order that improves the balance. The metric used here for resource usage is: USER Time

For each node, the metric values for the ranks on that node are summed. The maximum and average value of those sums are shown below for both the current rank order and a custom rank order that seeks to reduce the maximum value.

A file named MPICH\_RANK\_ORDER.USER\_Time was generated along with this report and contains usage instructions and the Custom rank order from the following table.

Rank Order	Node Metric Imb.	Reduction In Max Value	Maximum Value	Average Value
Current	29.45%		4.509e+003	3.181e+003
Custom	0.34%	29.211%	3.192e+003	3.181e+003

### Data Movement

MPI Msg MBytes 2380.775

sweep3d.gmpi-u.ap2 (jd events in 0.218s)

# 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. Time	Imb. Time%	Calls	Group 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

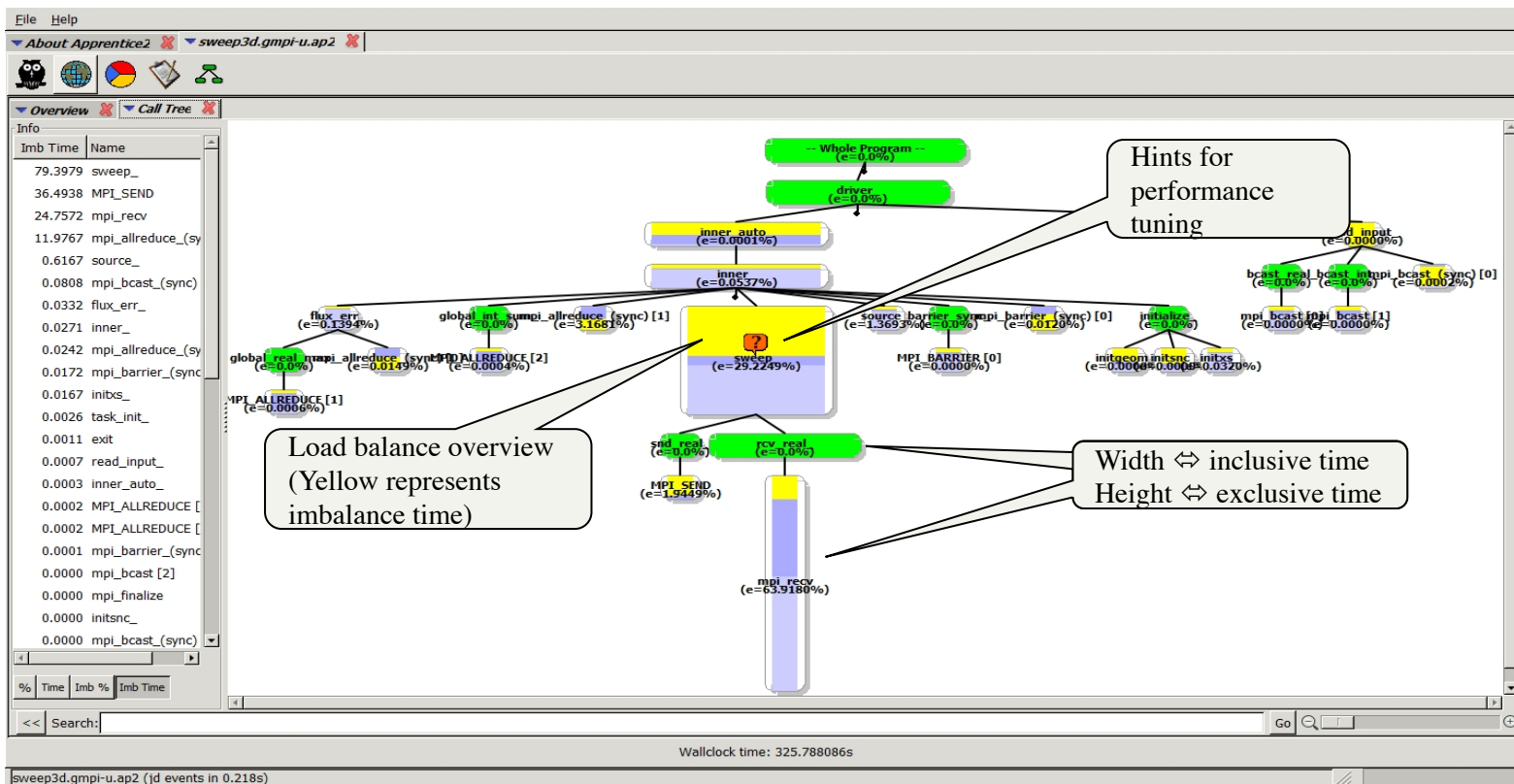


```
heidi@limited: /h/heidi — ssh — 81x26
Table 2: Profile by Group, Function, and Line

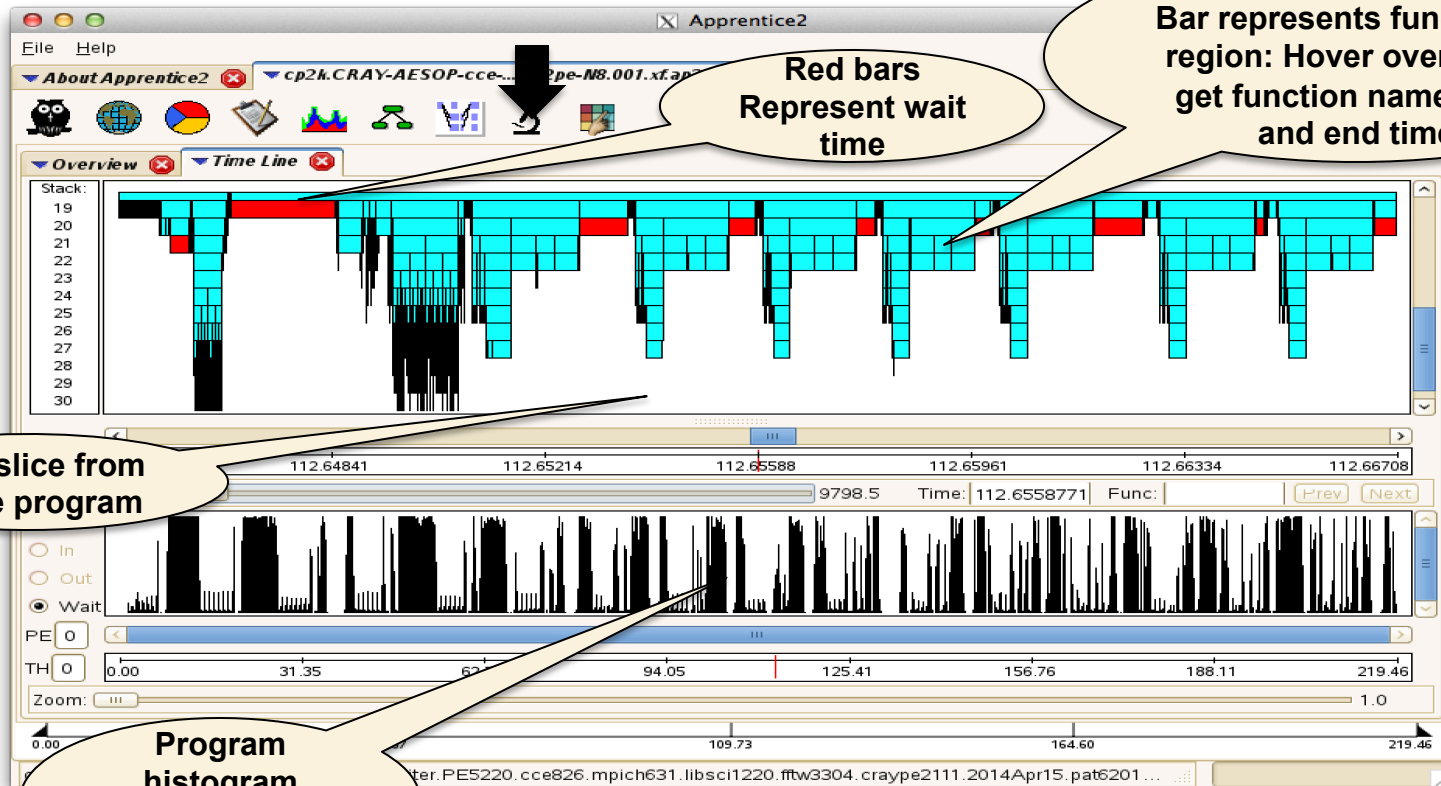
Samp% | Samp | Imb. | Imb. | Group
      |      | Samp | Samp% | Function
      |      |      |      | Source
      |      |      |      | Line
      |      |      |      | PE=HIDE

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



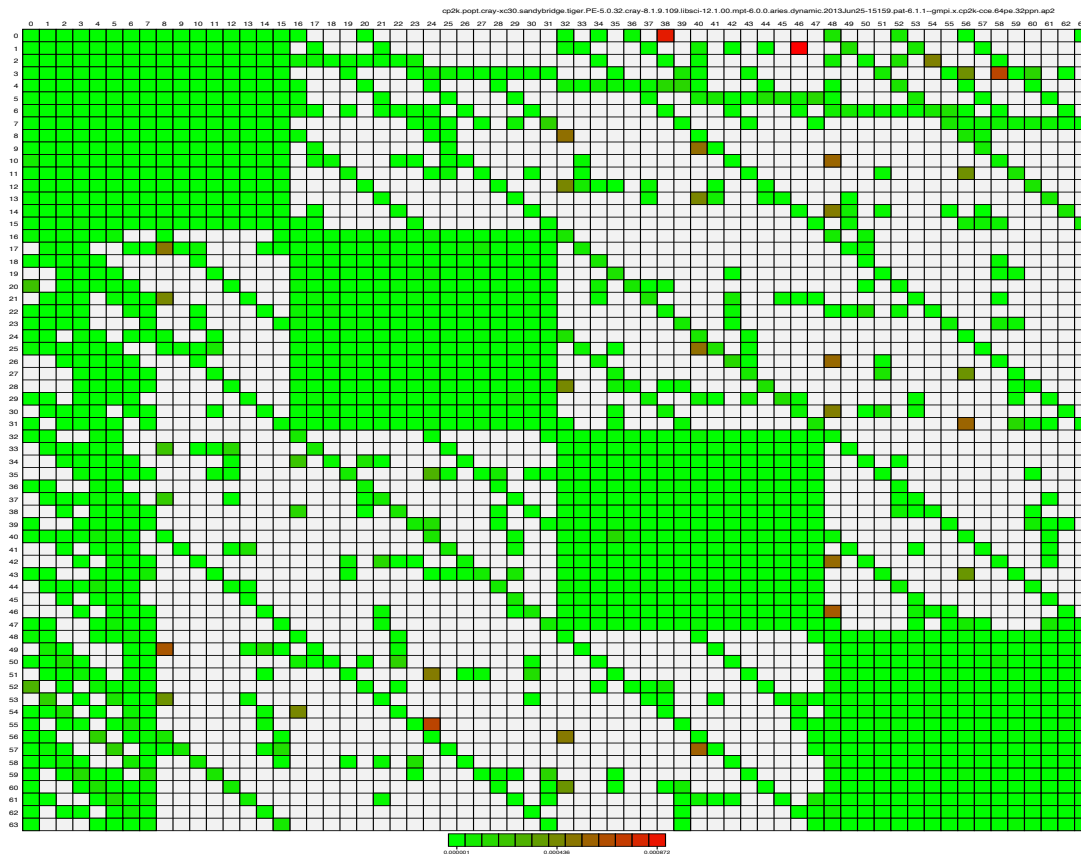
Red bars  
Represent wait  
time

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

Time slice from  
whole program

Program  
histogram  
showing wait  
time

# What is My Program's Communication Pattern?



COMPUTE | STORE | ANALYZE

© Cray Inc. Proprietary

# MPI Rank Order Observations



Table 1: Profile by Function Group and Function

Time%	Time	Imb. Time	Imb. Time%	Calls	Group Function PE=HIDE
100.0%	463.147240	--	--	21621.0	Total
<b>52.0%</b>	<b>240.974379</b>	--	--	<b>21523.0</b>	<b>MPI</b>
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 Order	On-Node Bytes/PE	On-Node Bytes/PE% of Total Bytes/PE	MPICH_RANK_REORDER_METHOD
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
,467,25,499,105,507,41,475 5,439,37,407,69,447,101,415,15 730,723
73,395,81,427,57,459,17,419,13,471,45,503,29,479,77,511 3,440,35,432,67,400,99,408,1722,731,763,658,642,755,739,
113,491,49,387,89,451,121,485,399,85,431,21,463,61,391,1,464,43,496,27,472,51,504 675,707,650,682,715,698,666,
3 109,423,93,455,117,495,125,419,392,75,424,59,456,83,384, 690,747
# processors, based on Sent Msg Total Bytes collected for:
6,436,102,468,70,404,38,412,87 107,416,91,488,115,448,123,4257,345,265,313,281,305,273,
14,444,46,476,110,508,78,5002,530,34,562,66,538,98,522,180 337,609,369,577,377,617,329,
86,396,30,428,62,460,54,492,0,570,42,554,26,594,50,602 132,401,196,441,164,409,228, 513,529
#
118,420,22,452,94,388,126,4818,514,74,586,58,626,82,546,433,236,465,204,473,244,393, 545,297,633,361,625,321,585,
4 106,634,90,578,114,618,122,6188,497 537,601,289,553,353,593,521,
# Program: /lus/
nid00023/malice/craypat/ 129,563,193,531,161,571,225,10 252,505,140,425,212,457,156, 569,561
WORKSHOP/bh2o-demo/Rank/ 539,241,595,233,523,249,603,135,315,167,339,199,347,259,385,172,417,180,449,148,489,256,373,261,341,264,349,280,
sweep3d/src/sweep3d 185,555 307,231,371,239,379,191,331,220,481 317,272,381,269,309,285,333,
# Ap2 File: sweep3d.gmpi-u.ap2 153,587,169,627,137,635,201,247,299 131,534,195,542,163,566,227, 277,365
# Number PEs: 768 619,177,515,145,579,209,547,175,363,159,323,143,355,255,526,235,574,203,598,243,558, 352,301,320,325,288,357,328,
# Max PEs/Node: 16 217,611 291,207,275,183,283,151,267,187,606 304,360,312,376,293,296,368,
# 7,405,71,469,39,437,103,413,215,223 251,590,211,630,179,638,139, 336,344
# 47,445,15,509,79,477,31,501 133,406,197,438,165,470,229,622,155,550,171,518,219,582,258,338,266,346,282,314,274,
# 111,397,63,461,55,429,87,421 414,245,446,141,478,237,502,147,614 370,766,306,710,378,742,330,
# To use this file, make a copy named MPICH_RANK_ORDER, and set the 253,398 761,660,737,652,705,668,745, 678,362
85 157,510,189,462,173,430,205,692,673,700,641,684,713,644, 646,298,750,322,718,354,758,
# environment variable 134,402,198,434,166,410,230,390,149,422,213,454,181,494,753,724 290,734,662,686,670,726,702,
MPICH_RANK_REORDER_METHOD to 442,238,466,174,506,158,394,221,486 729,732,681,756,721,716,764, 694,654
3 prior to 246,474 130,316,260,340,194,372,162,676,697,748,689,657,740,665, 262,375,263,343,270,311,271,
# executing the program. 190,498,254,426,142,458,150,348,226,308,234,380,242,332,649,708 351,286,319,278,342,287,350,
386,182,418,206,490,214,450,250,300 760,528,736,536,704,560,744, 279,374
# 222,482 202,364,186,324,154,356,138,520,672,568,712,592,752,552,294,318,358,383,359,310,295,
0,532,64,564,32,572,96,540,8128,533,192,541,160,565,232,292,170,276,178,284,210,218,640,600 382,326,303,327,367,366,335,
,596,72,524,40,604,24,588 525,224,573,240,597,184,557, 268,146 728,584,680,624,720,512,696, 302,334
104,556,16,628,80,636,56,620248,605 4,535,36,543,68,567,100,527,632,688,616,664,544,608,656, 765,661,709,663,741,653,711,
,48,516,112,580,88,548,120,6168,589,200,517,152,629,136, 12,599,44,575,28,559,76,607 648,576 669,767,655,743,671,749,695,
12 549,176,637,144,621,208,581,52,591,20,631,60,639,84,519,762,659,738,651,706,667,746, 679,703
1,403,65,435,33,411,97,443,9216,613 108,623,92,551,116,583,124,6643,714,691,674,699,754,683, 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**

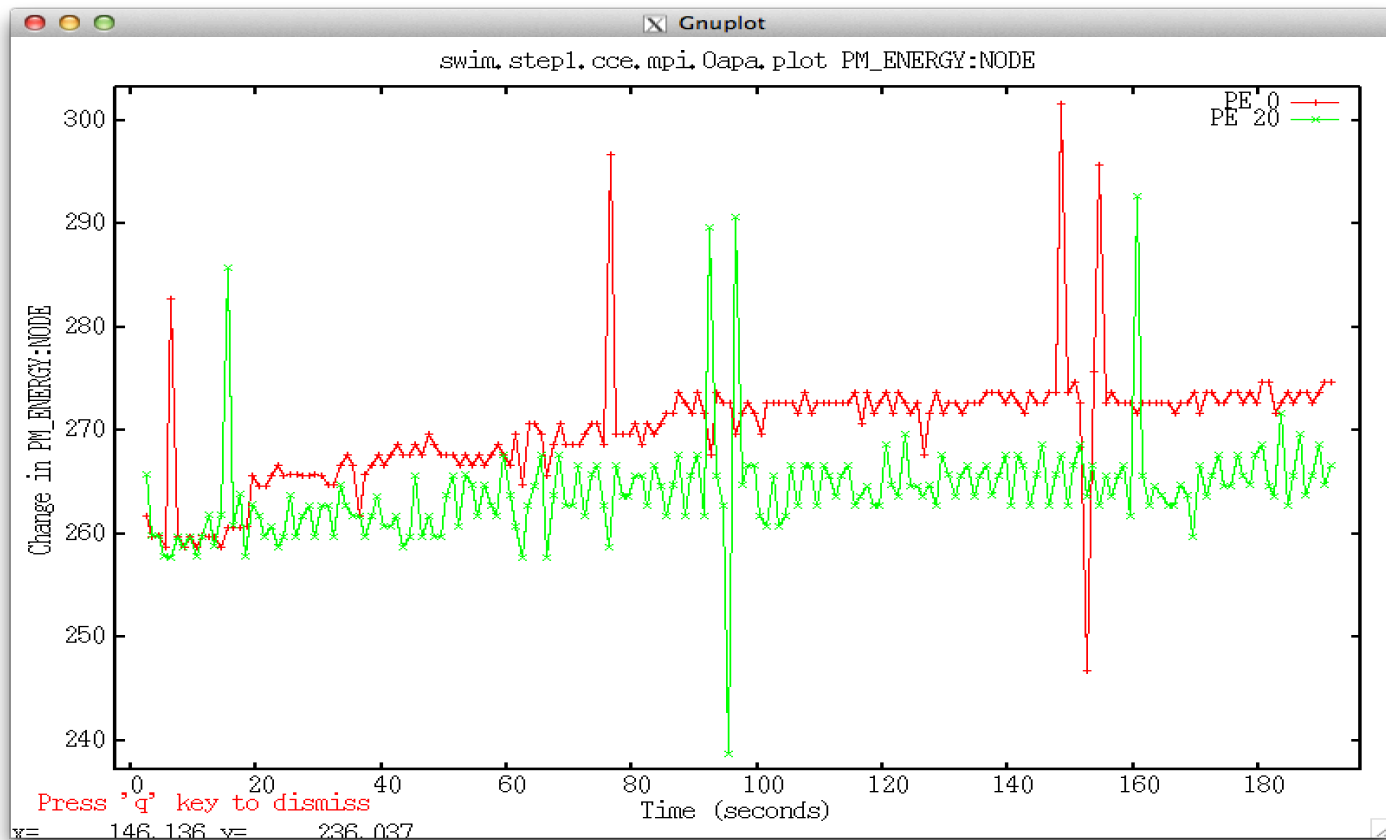
# Energy / Power Statistics

```
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-es1 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
```

• • •

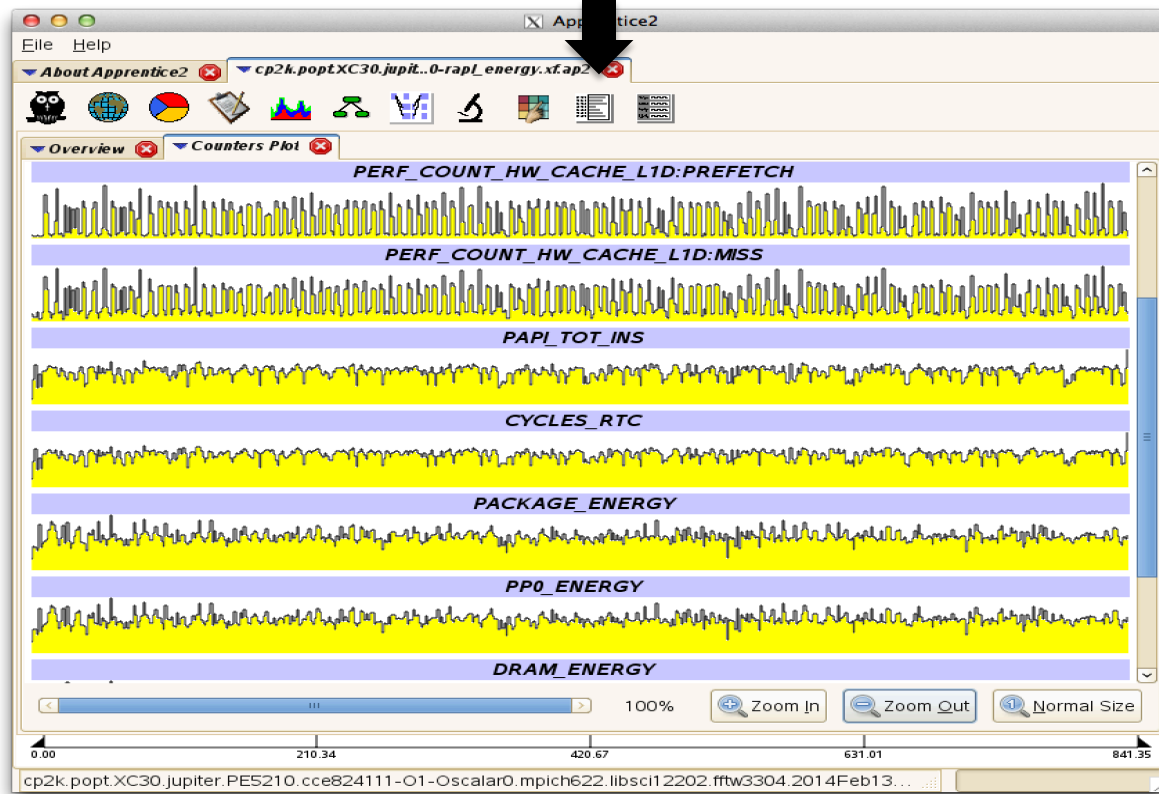
# Change in Energy Per Node Over Time



COMPUTE | STORE | ANALYZE

© Cray Inc. Proprietary

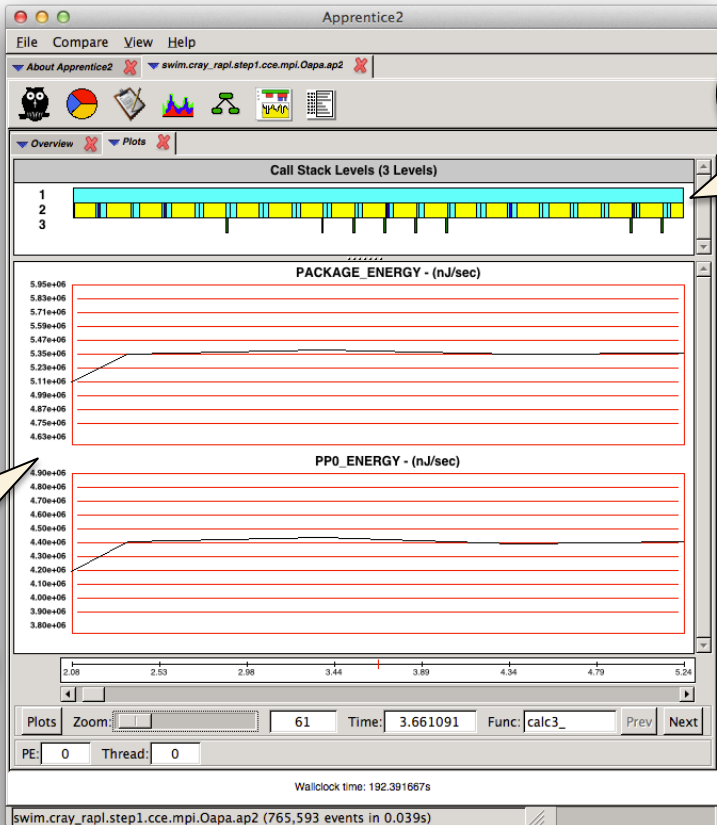
# Hardware Counters Plot with Energy



# 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

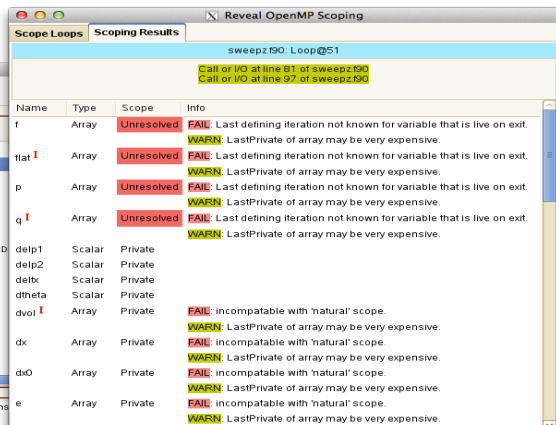
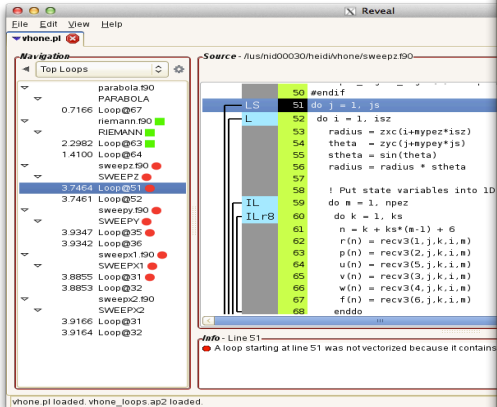
call remap ← contains more calls

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

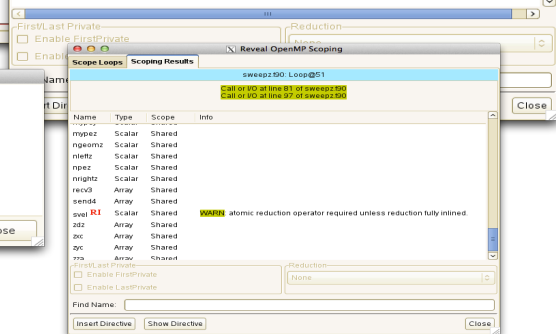
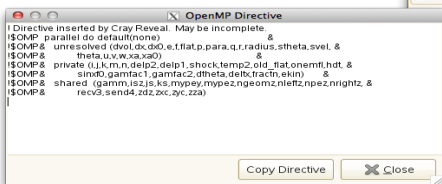
call remap ← contains more calls

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



The screenshot displays the Reveal Loopmark Legend tool interface. It features a main window with a source code editor, a navigation pane on the left, and a legend pane on the right. A status bar at the bottom provides compiler feedback.

**Performance feedback:** A callout bubble points to the 'Loop Performance' list in the navigation pane, which shows execution times for various loop iterations such as SWEEPY@35, SWEEPX1@31, and PARABOLA@67.

**Loopmark and optimization annotations:** A callout bubble points to the source code editor, which shows a code snippet with colored annotations (L, FVr2) and a comment: '! Put state variables into 1D arrays, padding'. The code includes a loop structure: `do i = 1, imax` followed by several assignment statements.

**Compiler feedback:** A callout bubble points to the 'Info' pane at the bottom, which displays a message: 'A loop starting at line 32 was not vectorized because it contains a call to subroutine "ppmlr" on line 53.'

**Loopmark Legend:** A legend pane on the right lists various optimization annotations and their meanings, such as 'A Pattern Matched', 'Collapsed', 'Deleted', 'Accelerated', 'Inlined', 'Not Inlined', 'Loop', 'Multithreaded', 'Region', 'Scoping Analysis', and 'Vectorized'.

# Access Compiler Message Information



The screenshot shows the 'Reveal' IDE with a file named 'vhone.pl' open. The 'Navigation' pane on the left shows a tree view of the program, with 'Loop@33' selected. The 'Source' pane displays the following code:

```
32 do m = 1, npxy
33 do i = 1, isy
34   n = i + isy*(m-1) + 6
35   r(n) = recv2(1,k,i,j,m)
36   p(n) = recv2(2,k,i,j,m)
37   u(n) = recv2(3,k,i,j,m)
38   v(n) = recv2(4,k,i,j,m)
39   w(n) = recv2(5,k,i,j,m)
40   f(n) = recv2(6,k,i,j,m)
41 enddo
42 enddo
43
44 do i = 1,imax
45   n = i + 6
```

The 'Info' pane below the source code shows two messages:

- A loop starting at line 33 was not vectorized because it does not have a constant stride.
- A loop starting at line 33 was unrolled 8 times.

The 'Explain' dialog box is open, displaying the following text:

**OPT\_INFO: A loop starting at line %s was unrolled.**

The compiler unrolled the loop. Unrolling creates a number of copies of the loop body. When unrolling an outer loop, the compiler attempts to fuse replicated inner loops - a transformation known as unroll-and-jam. The compiler will always employ the unroll-and-jam mode when unrolling an outer loop; literal outer loop unrolling may occur when unrolling to satisfy a user directive (pragma).

This message indicates that unroll-and-jam was performed with respect to the identified loop. A different message is issued when literal outer loop unrolling is performed, as this transformation is far less likely to be beneficial.

For sake of illustration, the following contrasts unroll-and-jam with literal outer loop unrolling.

```
# 426 "/ptmp/ulib/buildslaves/pdgc8-81-edition-build/tbs/build/release/pdgc8/pdgc8_ftn.msg.c"
DO J = 1,10
DO I = 1,100
A(I,J) = B(I,J) + 42.0
ENDDO
ENDDO

DO J = 1,10,2
DO I = 1,100
A(I,J) = B(I,J) + 42.0 ! unroll-and-jam
A(I,J+1) = B(I,J+1) + 42.0
ENDDO
ENDDO

DO J = 1,10,2
DO I = 1,100
A(I,J) = B(I,J) + 42.0 ! literal outer unroll
ENDDO
DO I = 1,100
A(I,J+1) = B(I,J+1) + 42.0
ENDDO
ENDDO
```

The dialog box also contains the following text:

The literal outer unroll code performs the same sequence of memory operations as the original nest, while the unroll-and-jam transformation interleaves operations from outer loop iterations. The compiler employs literal outerloop unrolling only when the data dependencies in the loop, or a control flow impediment, prevent fusion of the replicated inner loops. Literal outer loop unrolling is generally not desirable. It is provided to ensure expected behavior and for those rare instances where the user has determined that it is beneficial.

Buttons at the bottom of the dialog box: 'Explain other message...', 'Close'.

Access integrated message 'explain' support by right clicking on message

# View Loops through Call Chain



The screenshot shows the Reveal IDE interface. On the left, the 'Navigation' pane displays a 'Loop Performance' table with columns for time, loop name, and star rating. The 'PARABOLA@67' loop is selected, showing seven instances with times ranging from 0.0146 to 0.0167. Below this is a 'Traceback' pane showing the call chain: PARABOLA@67, PPMLR@51, sweep1\_LOOP2.li.32@53, sweep1\_LOOP1.li.31@32, SWEEPX1@31, and VHONE@232. The main 'Source' pane shows the code for 'parabola.f90', with lines 66-74 highlighted in green and lines 75-77 in black. A blue box labeled 'Vr2' is positioned above line 75. A yellow callout bubble points to the 'Instance #1' row in the performance table, labeled 'Loop instances'. Another yellow callout bubble points to the 'Traceback' pane, labeled 'Loop traceback'. The status bar at the bottom indicates 'vhone.pl loaded. vhone\_loops.ap2 loaded.'

Time	Loop Name	Rating
4.0423	SWEEPX2@32	★
3.8576	SWEEPZ@51	★★
3.8573	SWEEPZ@52	★★
2.2068	RIEMANN@63	★★
1.2299	RIEMANN@64	★★
0.8068	PARABOLA@67	★★
0.0146	Instance #1	
0.0156	Instance #2	
0.0156	Instance #3	
0.0163	Instance #4	
0.0163	Instance #5	
0.0174	Instance #6	
0.0167	Instance #7	

```
66  
67 do n = nmin, nmax  
68   deltaa(n) = ar(n) - al(n)  
69   a6(n)     = 6. * (a(n) - .5 * (al(n) + ar(n)))  
70   scrch1(n) = (ar(n) - a(n)) * (a(n)-al(n))  
71   scrch2(n) = deltaa(n) * deltaa(n)  
72   scrch3(n) = deltaa(n) * a6(n)  
73 enddo  
74  
75 do n = nmin, nmax  
76   if(scrch1(n) <= 0.0) then  
77     ar(n) = a(n)  
78     a(n) = a(n)
```



# Hybridization Step 2: Scope Selected Loop(s)



Reveal OpenMP Scoping

Scope Loops | Scoping Results

Edit List | List of Loops to be Scoped

Scope?	Line #	File or Source Line
<input checked="" type="checkbox"/>		/home/users/heidi/reveal/evolve.f90
<input type="checkbox"/>		/home/users/heidi/reveal/flatten.f90
<input type="checkbox"/>		/home/users/heidi/reveal/forces.f90
<input type="checkbox"/>		/home/users/heidi/reveal/images.f90
<input type="checkbox"/>		/home/users/heidi/reveal/init.f90
<input type="checkbox"/>		/home/users/heidi/reveal/parabola.f90
<input type="checkbox"/>		/home/users/heidi/reveal/ppmlr.f90
<input type="checkbox"/>		/home/users/heidi/reveal/prin.f90
<input checked="" type="checkbox"/>		/home/users/heidi/reveal/remap.f90
<input checked="" type="checkbox"/>		/home/users/heidi/reveal/riemann.f90
<input checked="" type="checkbox"/>		/home/users/heidi/reveal/states.f90
<input checked="" type="checkbox"/>		/home/users/heidi/reveal/sweepx1.f90
<input checked="" type="checkbox"/>		/home/users/heidi/reveal/sweepx2.f90
<input checked="" type="checkbox"/>		/home/users/heidi/reveal/sweepy.f90
<input checked="" type="checkbox"/>		/home/users/heidi/reveal/sweepz.f90

Apply Filter | Time: 0.000 | Trips: 2 | Threads: 4 | Speedup: 0.010

Start Scoping | Cancel | 26 Loops selected | Close

# Review Feedback on Scoping Results



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

Name	Type	Scope	Info
ar@parabola_ <b>I</b>	Scalar	Unresolved	<b>FAIL:</b> Possible recurrence involving this object. <b>FAIL:</b> Possible resolvable recurrence involving this object.
da@parabola_ <b>I</b>	Scalar	Unresolved	<b>FAIL:</b> Possible recurrence involving this object. <b>FAIL:</b> Possible resolvable recurrence involving this object.
delta@remap_ <b>I</b>	Scalar	Unresolved	<b>FAIL:</b> Possible recurrence involving this object. <b>FAIL:</b> Possible resolvable recurrence involving this object.
dvol <b>I</b>	Array	Unresolved	<b>FAIL:</b> Possible recurrence involving this object. <b>FAIL:</b> Possible resolvable recurrence involving this object. <b>WARN:</b> LastPrivate of array may be very expensive.
dx	Array	Unresolved	<b>FAIL:</b> Possible recurrence involving this object. <b>FAIL:</b> Possible resolvable recurrence involving this object. <b>WARN:</b> LastPrivate of array may be very expensive.
dx0	Array	Unresolved	<b>FAIL:</b> Possible recurrence involving this object. <b>FAIL:</b> Possible resolvable recurrence involving this object. <b>WARN:</b> LastPrivate of array may be very expensive.
e	Array	Unresolved	<b>FAIL:</b> Possible recurrence involving this object.

See where variable  
came from  
(@function\_name)

# Review Parallelization Issues



The screenshot shows the 'Reveal OpenMP Scoping' application window. The title bar reads 'Reveal OpenMP Scoping'. There are two tabs: 'Scope Loops' and 'Scoping Results', with 'Scoping Results' selected. The main content area is titled 'sweepy.f90: Loop@35'. It contains a list of calls or I/O operations, with the following lines highlighted in yellow:

```
Call or I/O at line 62 of sweepy.f90
4: /home/users/heidi/reveal/volume.f90:34
3: /home/users/heidi/reveal/evolve.f90:21
```

Below this list is a table with the following columns: Name, Type, Scope, and Info.

Name	Type	Scope	Info
ks	Scalar	Shared	
mypey	Scalar	Shared	
ndim	Scalar	Shared	
npey	Scalar	Shared	
recv1	Array	Shared	
send2	Array	Shared	
svel <b>RI</b>	Scalar	Shared	<b>WARN:</b> atomic reduction operator required unless reduction fully
zdy	Array	Shared	
zxc	Array	Shared	
zya	Array	Shared	

At the bottom of the window, there are several controls: 'First/Last Private' with checkboxes for 'Enable FirstPrivate' and 'Enable LastPrivate'; a 'Reduction' dropdown menu set to 'None'; a 'Find Name:' text input field; and three buttons: 'Insert Directive', 'Show Directive', and 'Close'.

Two callout boxes provide additional context:

- A callout box pointing to the highlighted call lines says: "Reveal identifies calls that prevent parallelization".
- A callout box pointing to the 'svel RI' row in the table says: "Reveal identifies shared reductions down the call chain".

# 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



The screenshot shows the Cray Reveal IDE interface. On the left is a navigation pane with a tree view of files under 'vhone.pl'. The main window displays a code editor for 'riemann.f90' with the following code:

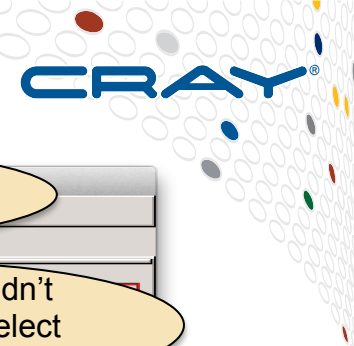
```
64 !$OMP parallel do default(none)
65 !$OMP& private (l)
66 !$OMP& shared (lmin,lmax,prgh,urgh,vrgh,plft,ulft,vlft,pmid,clft,
67 !$OMP& crgh,gamfac1,gamfac2,plfti,pmold,prghi,umidl,umidr
68 !$OMP& wlft,wrgh,zlft,zrgh,n)
69 do l = lmin, lmax
70 do n = 1, 12
71 pmold(l
72 wlft(l
73 wrgh(l
74 wlft(l
75 wrgh(l
76 zlft(l
```

A dialog box titled 'Scope Loops' is open, showing 'Scoping Results' for 'riemann.f90: Loop@69'. It contains a table with the following data:

Name	Type	Scope	Info
l	Scalar	Private	
n	Scalar	Private	<b>WARN:</b> Scope does not agree with user OMP directive.
clft	Array	Shared	
crgh	Array	Shared	
gamfac1	Scalar	Shared	
gamfac2	Scalar	Shared	

The dialog also has checkboxes for 'Enable FirstPrivate' and 'Enable LastPrivate', a 'Reduction' dropdown set to 'None', a 'Find Name' input field, and buttons for 'Insert Directive', 'Show Directive', and 'Close'. A speech bubble points to the 'WARN' message in the table, containing the text: 'User inserted directive with mis-scoped variable 'n''.

# Hybridization Step 4: Performance Analysis



The screenshot shows a compiler interface with a file named `vhone.pl`. The **Navigation** pane on the left shows a tree view of source files, with `prin.f90` selected and line 42 highlighted. The main editor displays Fortran code with annotations: a blue 'F' at line 42, a blue 'FVw' at line 44, and a blue '!I' at line 55. A yellow callout bubble points to the 'Compiler Messages' view, stating: "Choose 'Compiler Messages' view to access message filtering". Another yellow callout bubble points to the code, stating: "See loops that didn't vectorize. Can select other filters." A third yellow callout bubble points to the 'Info' pane, stating: "See all compiler messages for a loop nest".

```
41  
42 do k = 1, ks  
43   do j = 1, js  
44     do i = 1, imax  
45       send_buff(i,1,j,k) = zro(i,j,k)  
46       send_buff(i,2,j,k) = zpr(i,j,k)  
47       send_buff(i,3,j,k) = zux(i,j,k)  
48       send_buff(i,4,j,k) = zuy(i,j,k)  
49       send_buff(i,5,j,k) = zuz(i,j,k)  
50     enddo  
51   enddo  
52 enddo  
53  
54 gathbuffer_size = imax * js * ks * nvarout  
55 call MPI_GATHER(send_buff, gathbuffer_size, MPI_REAL, recv, gat
```

**Info - Line 42**

- A loop starting at line 42 is flat (contains no external calls).
- A loop starting at line 42 was not vectorized because a better candidate was found at line 44.
- A loop starting at line 43 is flat (contains no external calls).
- A loop starting at line 43 was not vectorized because a better candidate was found at line 44.
- A loop starting at line 44 is flat (contains no external calls).
- A loop starting at line 44 was vector pipelined.

# 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 ?