Addressing Performance and Programmability Challenges in Current and Future Supercomputers

Luiz DeRose
Sr. Principal Engineer
Programming Environments Director
Cray Inc.

Outline

- Computer architecture and applications trends
- Programming Environment mission
- Tools focus for extreme scale computing
- Open issues
- Conclusions

Future Architectural Directions

- **Nodes are becoming more parallel**
  - More processors per node
  - More threads per processor
  - Vector lengths are getting longer
  - Memory hierarchy is becoming more complex
  - Scalar performance is not increasing and will start decreasing

- **As processors get faster, memory bandwidth cannot keep up, resulting in:**
  - More complex caches
  - Non-uniform memory architecture (NUMA) for shared memory on node
  - Operand alignment is becoming more important

Application Developers: Are you ready for the future?

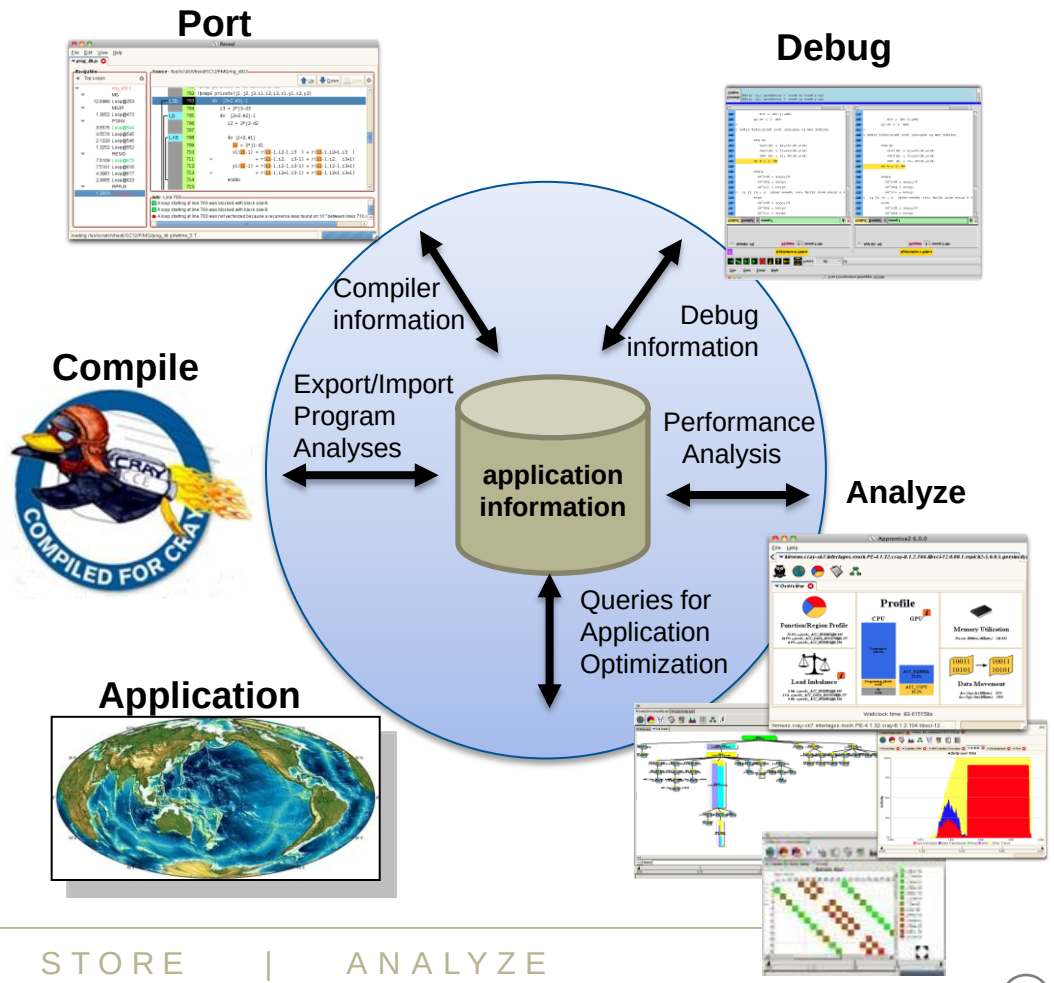
- **Today's petascale applications are not structured to take advantage of these architectures**
  - Currently 80-90% of applications use a single level of parallelism
    - Message passing between cores of the MPP system

- **Looking forward, application developers are faced with a significant task in preparing their applications for the future**
  - You will have to **move to a hybrid** (MPI, threading, & vector) architecture

  - You must start considering how to **manage your arrays** to have them close to the computational engine when you need them
    - We are moving to a more complex memory hierarchy that will require user intervention to achieve good performance.

The Programming Environment Mission

- 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**



Extreme Scale Tools Focus

- **The current and future generation of tools should focus on**
  - 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**
  - Power
    - One of the biggest challenges facing extreme scale systems
      - Need to develop power-aware tools and techniques to aid the user to properly direct the system's use of power

What is needed to support the Performance, Programmability, and Power?

- **Transitioning tools** to assist the user in assessing and re-balancing an application for performance and power
- An **environment** that allows the developer **to change applications a little at a time**
- Provide support for **user-directed tuning guidance**
  - for both performance and power
- Provide **analysis and automation** as we learn more

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 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**

Moving to a Hybrid Code: a Three-Task Approach

- **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
- **Parallelize and vectorize identified loops**
  - **Split loop work among threads**
    - Do parallel analysis and restructuring on targeted high level loops
- **Add OpenMP (and then OpenACC directives)**
  - **Add parallel directives and acceleration extensions**
    - Insert OpenMP directives
    - Verify application and check for performance improvements
      - Convert desired OpenMP directives to OpenACC
- **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?

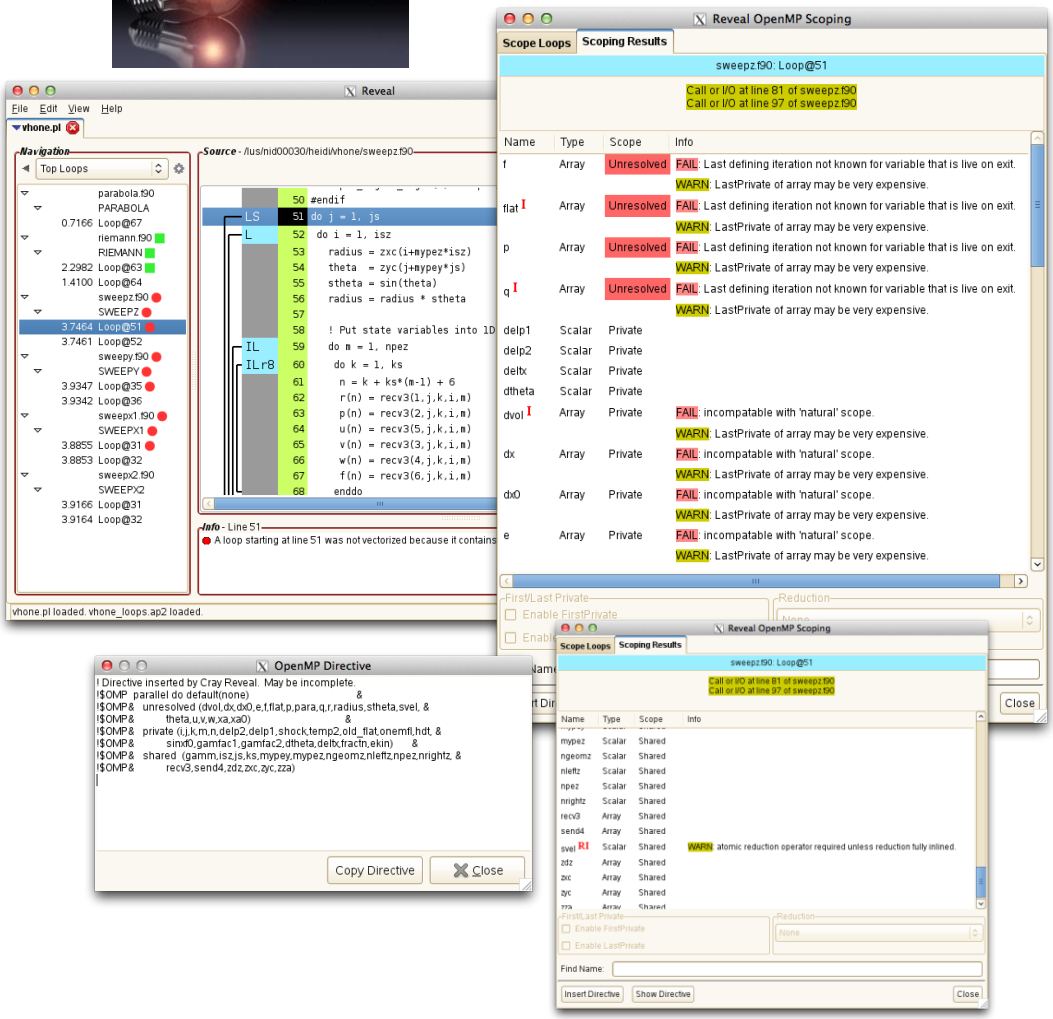
```
subroutine sweepz
…
do j = 1, js
 do i = 1, isz
   radius = zxc(i+mypez*isz)
   theta  = zyc(j+mypey*js)
   do m = 1, npez
    do k = 1, ks
     n = k + ks*(m-1) + 6
     r(n) = recv3(1,j,k,i,m)
     p(n) = recv3(2,j,k,i,m)
     u(n) = recv3(5,j,k,i,m)
     v(n) = recv3(3,j,k,i,m)
     w(n) = recv3(4,j,k,i,m)
     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
```

Tools Needed When Creating Hybrid Codes

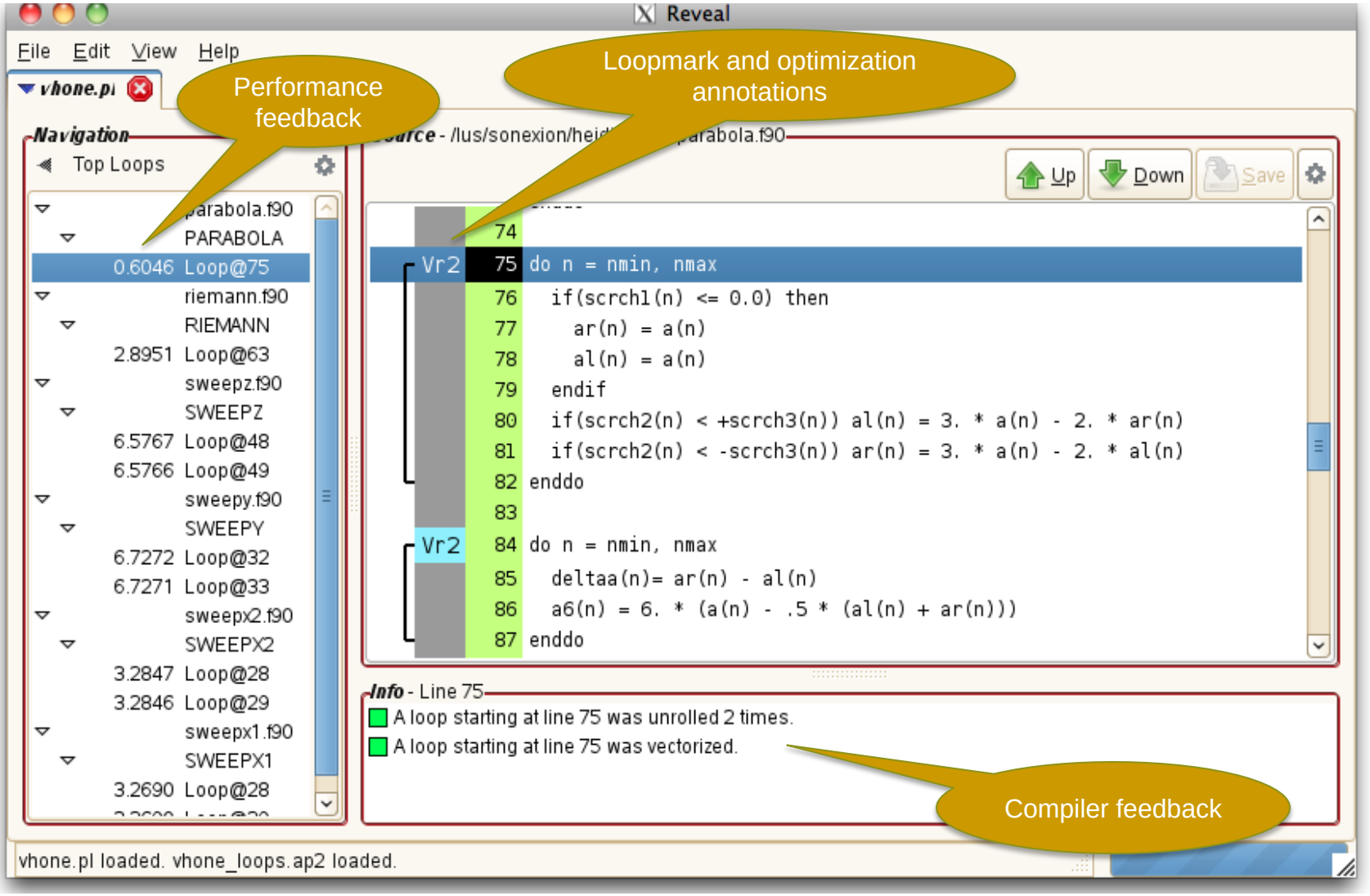
- **Tools to design your application to be performance-portable across a wide range of systems**
  - Application developers want to develop a single code that can run efficiently on multi-core nodes with or without an accelerator
- **Tools to understand your application**
  - Where is the time spent?
  - Where are the key parallel structures (loops)?
  - Where are the important arrays used?
  - What prohibits parallelizing these structures?
- **Tools to help with scoping analysis**
  - Identify shared, private and ambiguous variables

Simplifying the Task with Reveal

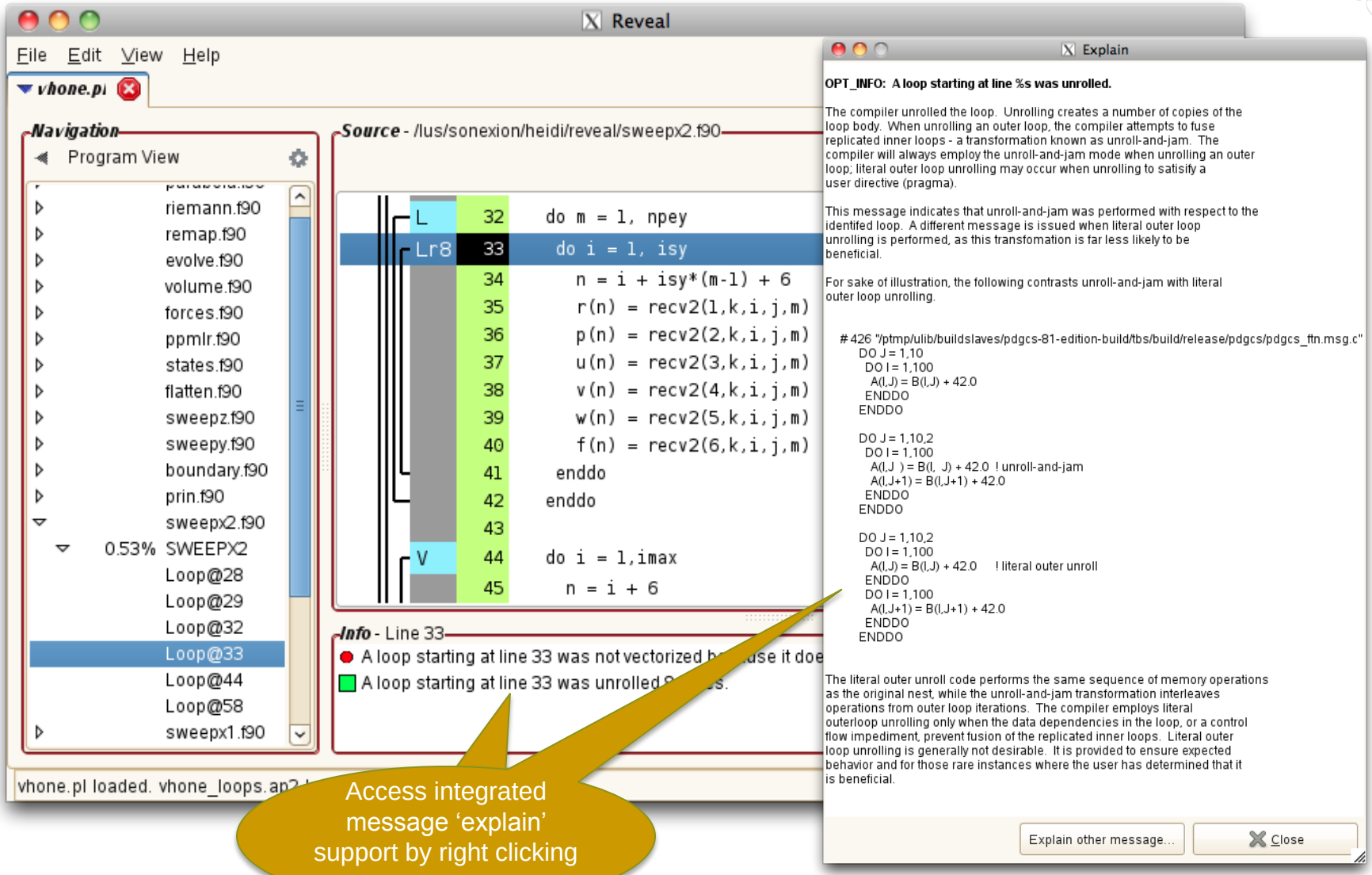


- Navigate to relevant loops to parallelize
- Identify parallelization and scoping issues
- Get feedback on issues down the call chain (shared reductions, etc.)
- Optionally insert parallel directives into source
- Validate scoping correctness on existing directives

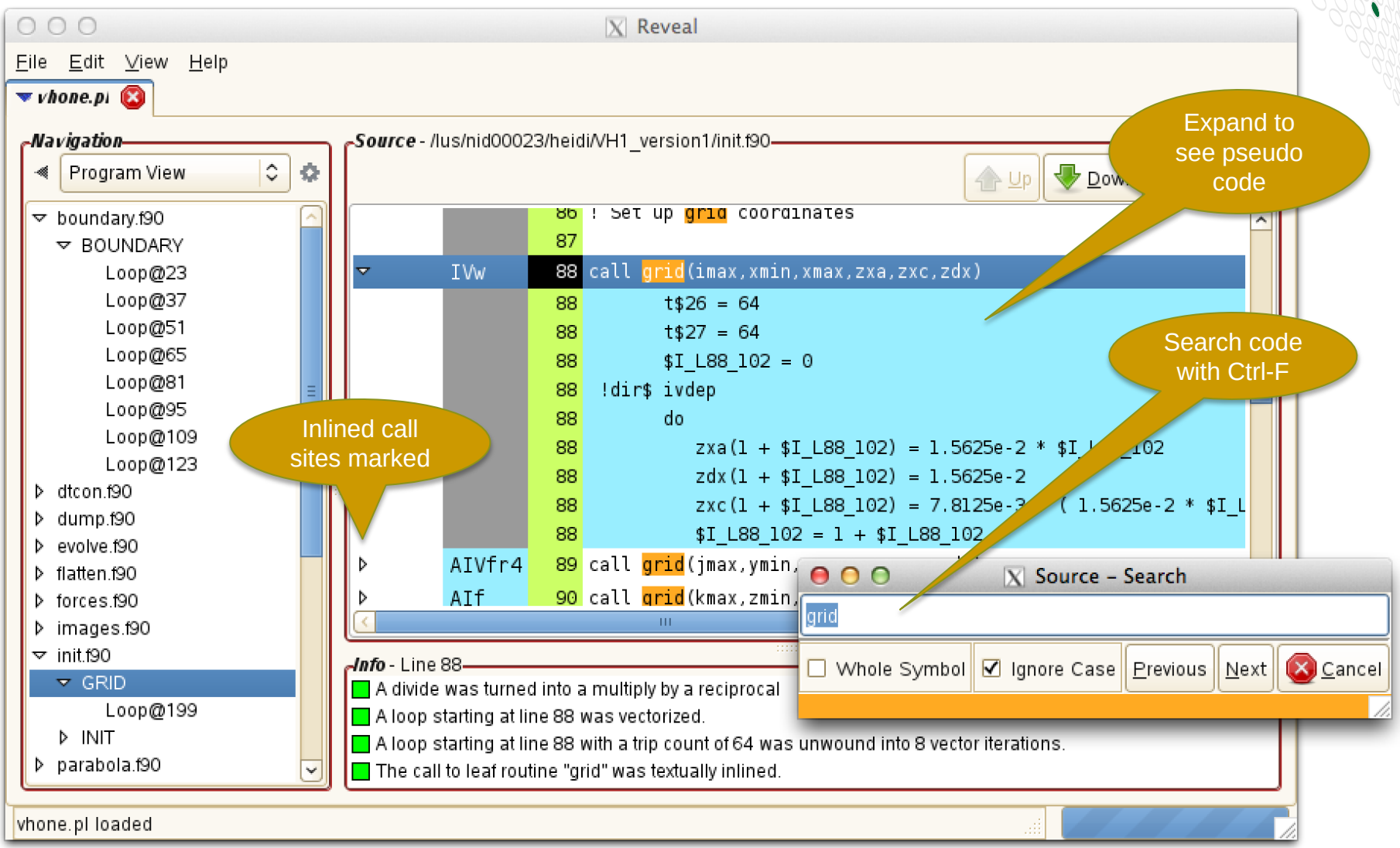
Visualize Loopmark with Performance Information



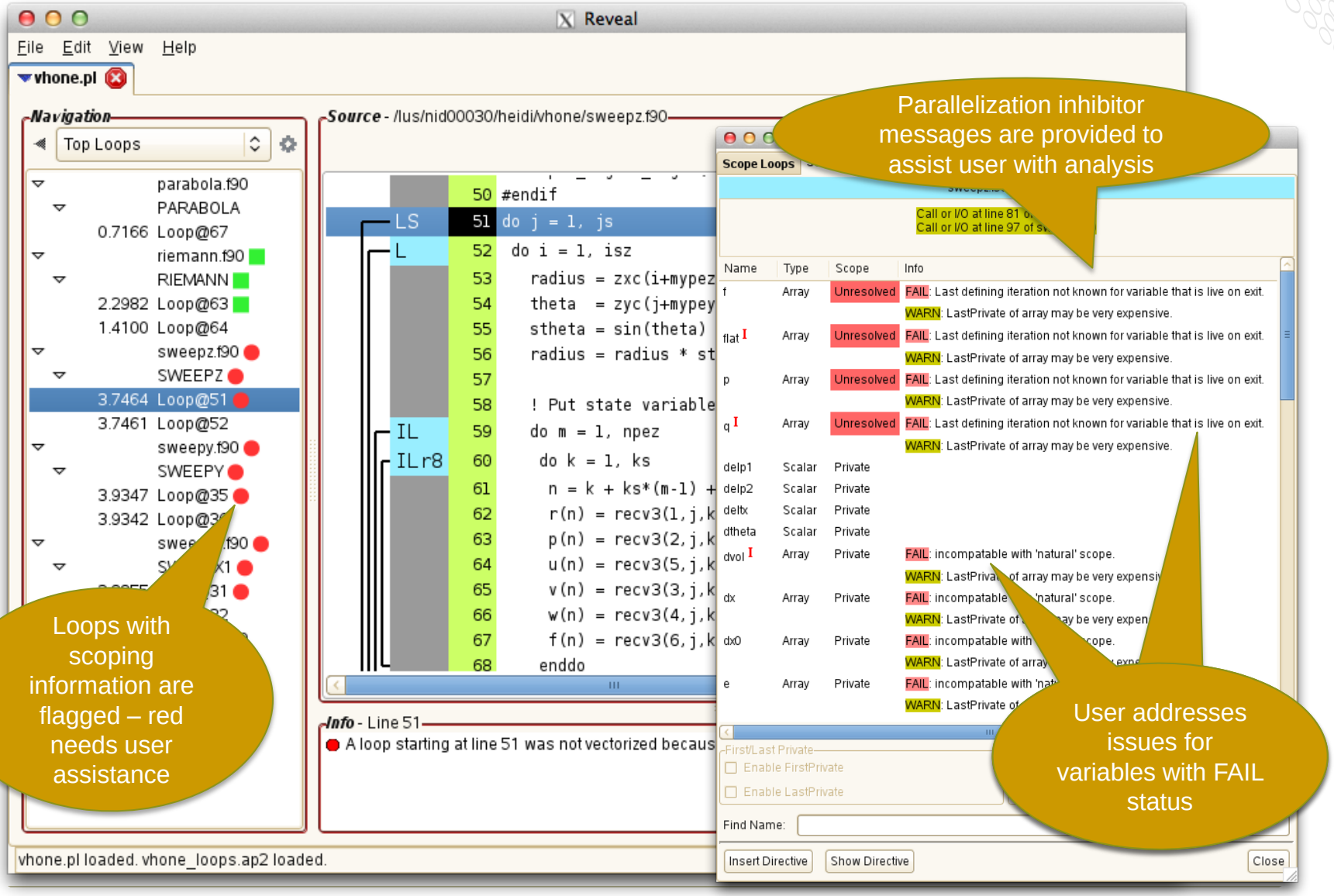
Access Cray Compiler Message Information



View Pseudo Code for Inlined Functions



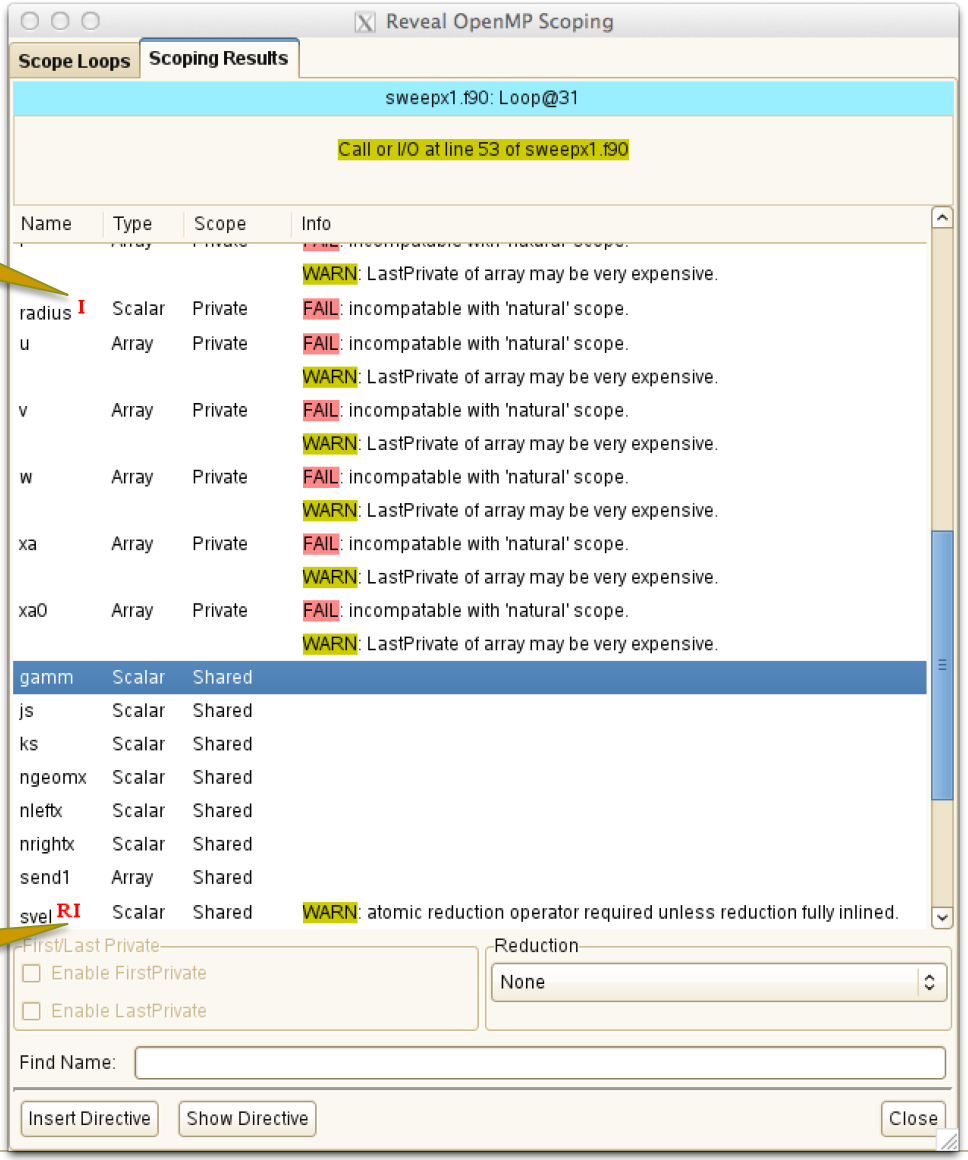
Scoping Assistance – Review Scoping Results



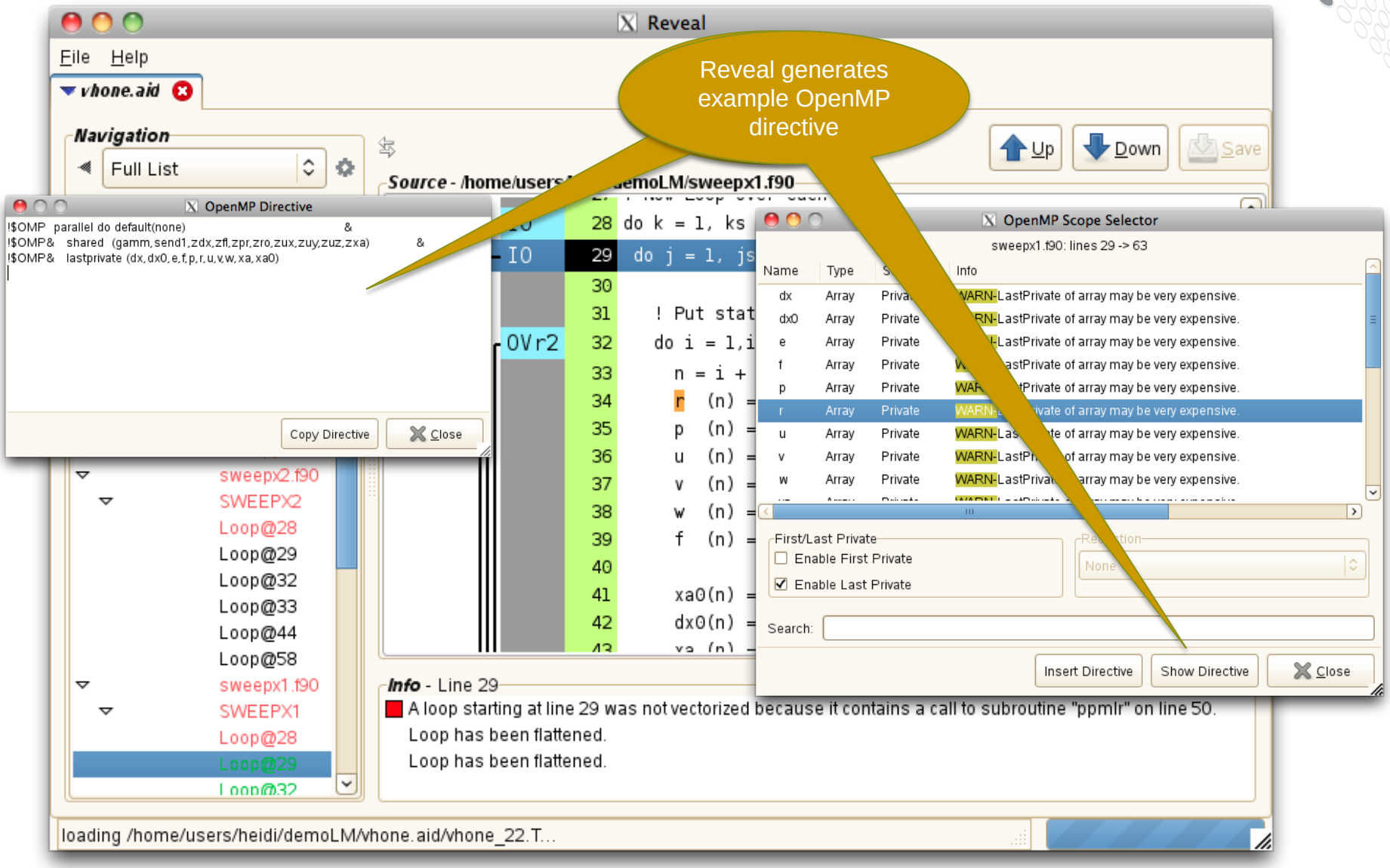
Reveal Gives Feedback on Scoping Results

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

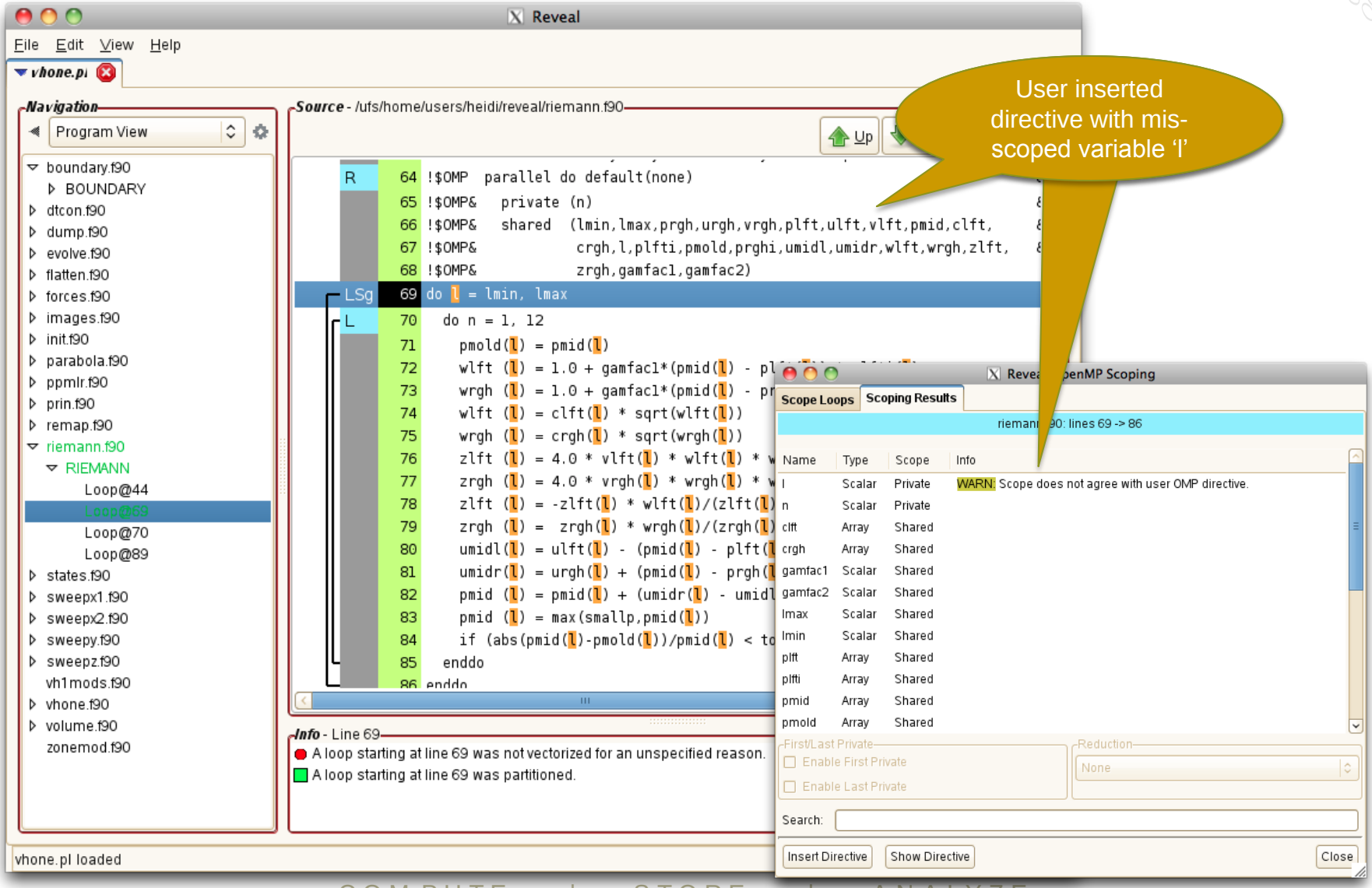
Reduction variable from down the call stack - hover over ‘RI’ to see what symbol means



Scoping Assistance – Reveal Generates Directive



Use Reveal to Validate User Inserted Directives



Moving to a Hybrid Code: a Three-Task Approach

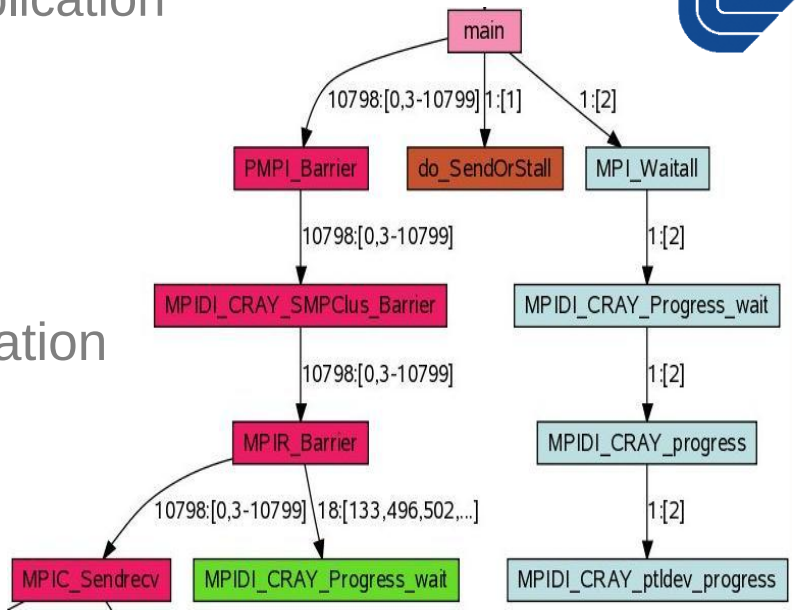
- **Identify potential loops to accelerate**
  - **Determine where to add additional levels of parallelism**
    - Assumes the MPI application is functioning correctly
    - Find top work-intensive loops
      - **Performance tools + compiler loop work estimates**
- **Parallelize and vectorize identified loops**
  - **Split loop work among threads**
    - Do parallel analysis and restructuring on targeted high level loops
      - **Reveal with compiler feedback and source code browsing**
- **Add OpenMP (and then OpenACC directives)**
  - **Add parallel directives and acceleration extensions**
    - Insert OpenMP directives
      - **Reveal with compiler scoping assistance**
    - Verify application and check for performance improvements
      - Convert desired OpenMP directives to OpenACC
- **We want a performance-portable application at the end**

Debugging on Extreme Scale Systems

- **Systems with thousands of threads of execution need a new debugging paradigm**
- **Need to build tools around traditional debuggers with innovative techniques for productivity and scalability**
  - Support for traditional control-centric debugging mechanism
  - **STAT - Stack Trace Analysis Tool**
    - MRNet based 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.
  - **Comparative debugging**
    - Collaboration with University Queensland
    - A **data-centric paradigm**
      - Ability to see data from multiple program executions in the same instance

Stack Trace Analysis Tool (STAT)

- **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
- **Merge/analyze traces:**
  - Facilitate scalable analysis/data presentation
  - Multiple traces over space or time
  - Create call graph prefix tree
    - Compressed representation
    - Scalable visualization & analysis



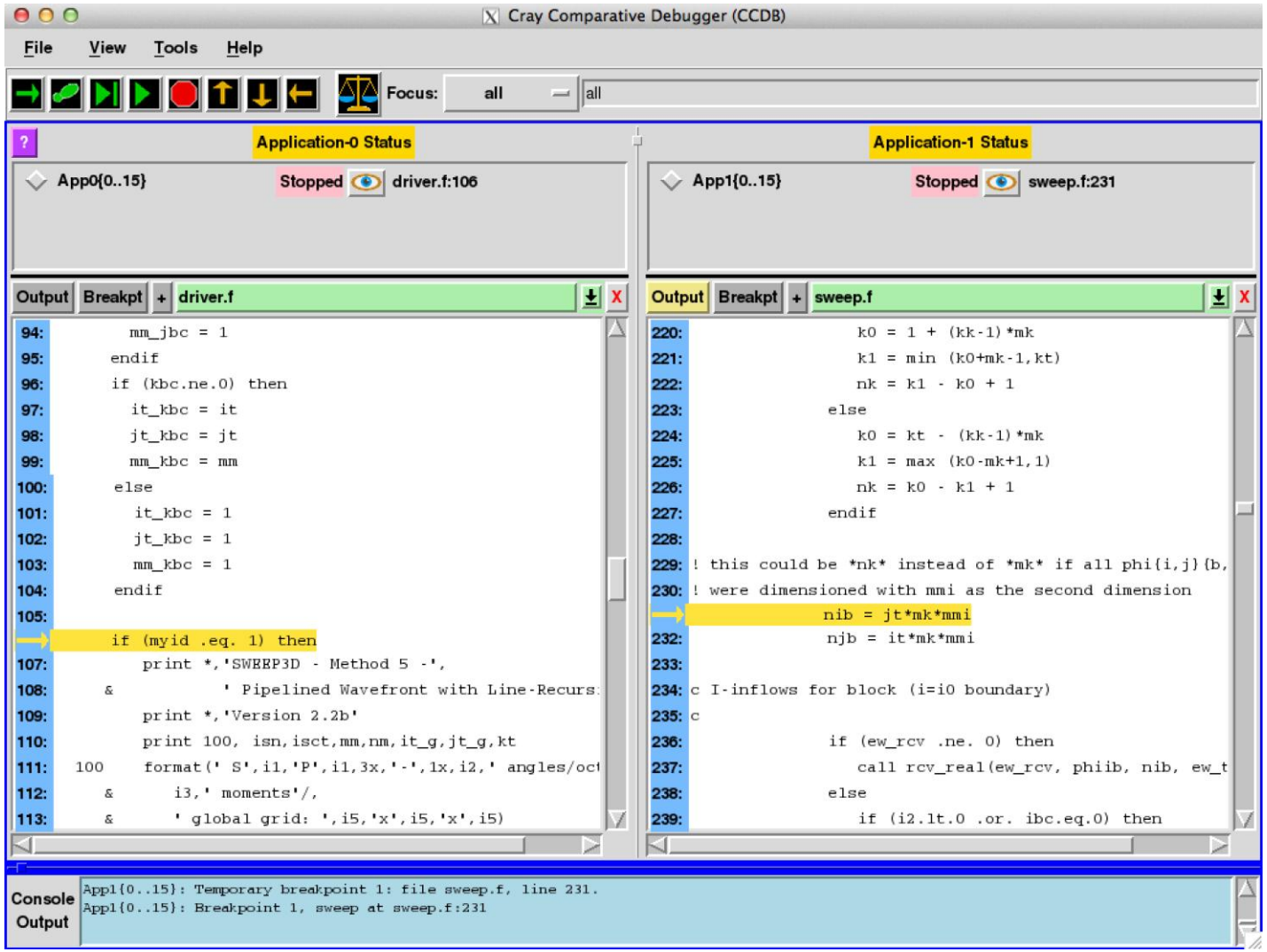
- **ATP - Automatic Termination Processing**
  - System of light weight back-end monitor processes on compute nodes
  - Leap into action on any application process trapping

Comparative Debugger

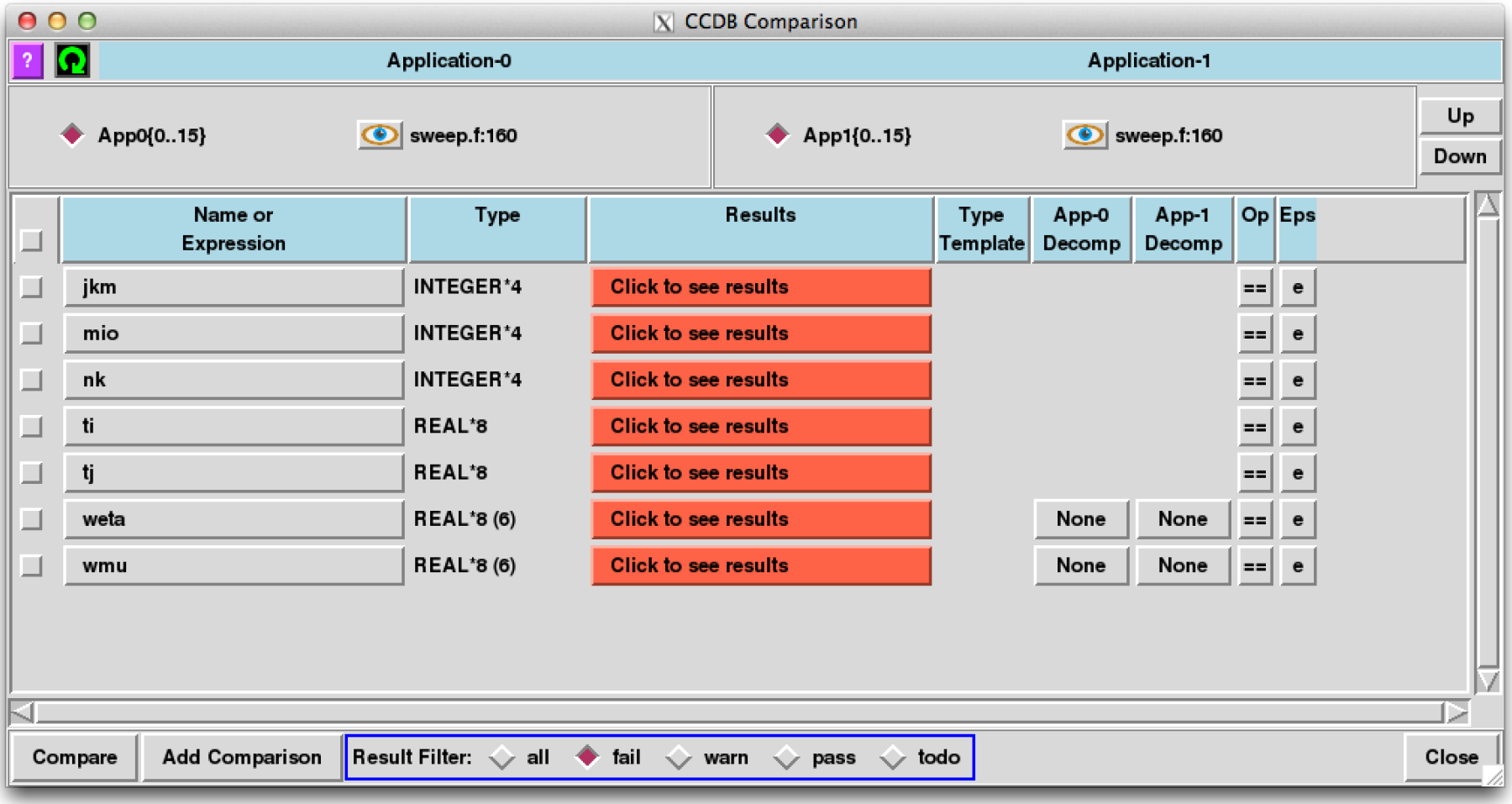
- **Helps the programmer locate errors in the program by observing the divergence in key data structures as the programs are executing**

  - Allows comparison of a “suspect” program against a “reference” code using assertions
    - Simultaneous execution of both
    - Ability to assert the match of data at given points in execution
    - Focus on data – not state and internal operations
    - Narrow down problem without massive thread study
  - Data comparison
    - Tolerance control – nobody expect it to be perfect
    - Array subsets – correlate serial to parallel bits
    - Array index permutation – loops rearranged
    - Automated asserts – let it run until a problem is found
    - Forcing correct values – continue on with correct data

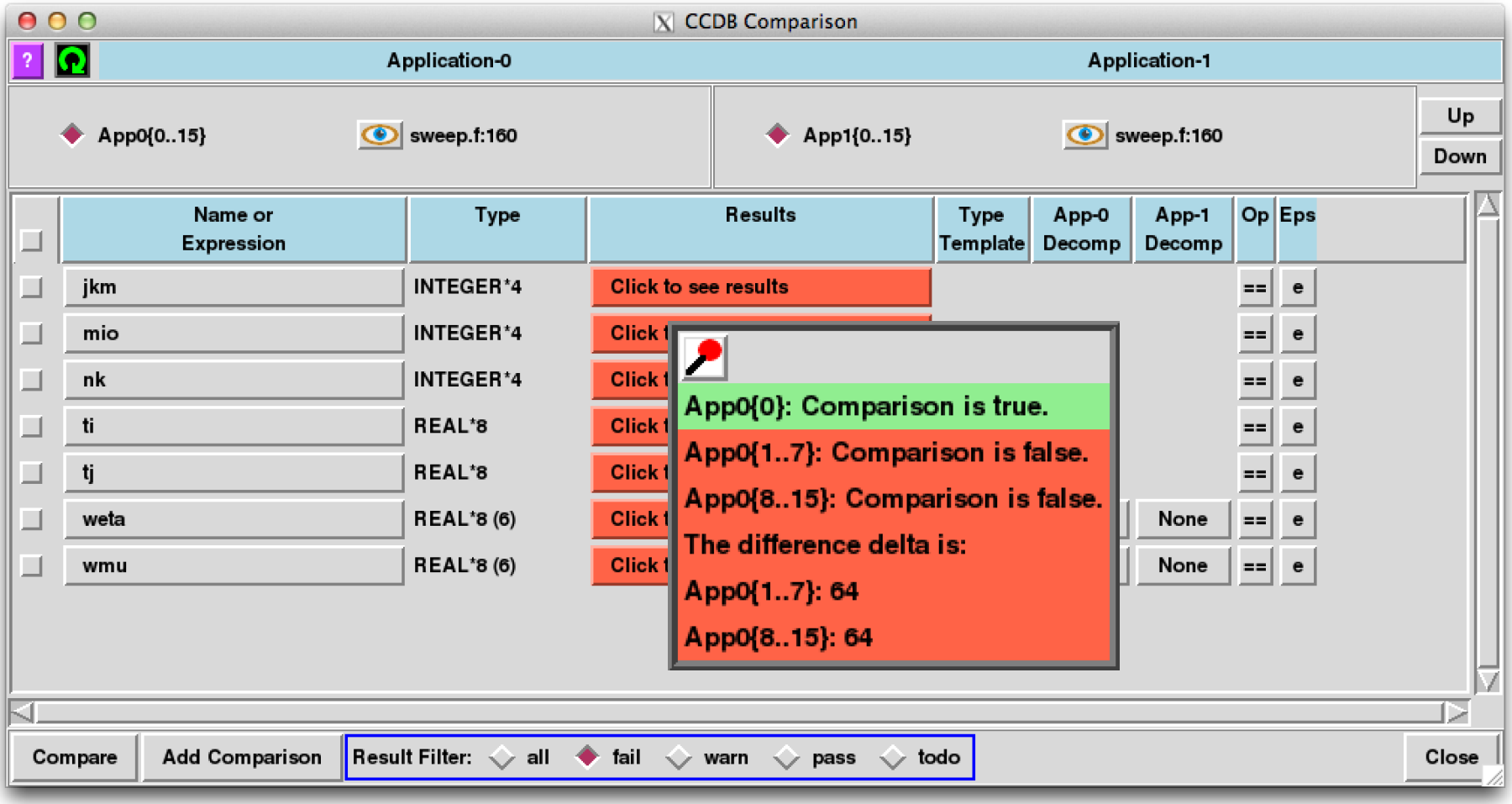
Cray Comparative Debugger



CCDB - Comparison



CCDB – Comparison



Why Application Level Power Management?

- **Power consumption is one of the biggest challenges facing extreme scale systems**
  - Scaling up from today's requirements for a petaflop computer is not a viable solution
  - Sites are already increasingly constrained by power & cooling limitations as well as cost of system power and cooling
- **Hardware could attempt to manage power automatically**
  - This strategy would most likely be reactive and certainly be sub-optimal
- **Power management must be proactive**
  - Software can help by monitoring and proactively managing how power is being used in the system
    - The programmer must understand the tradeoffs between power and performance
- **More research and development is needed for**
  - Power measurement and monitoring tools
  - Power analysis
  - Management of power consumption

Power Analysis

- **Ultimate goal is to develop power-aware tools and techniques**
  - Provide suggestions of possible transformations or implementations of the application with different performance/power profiles
  - Aid the user to properly direct the system's use of power
    - Using transparent automation where possible
    - Transparently adopting environmental settings to achieve a desired power reduction and energy savings with a predetermined impact to performance
  - Can leverage the work done in automatic performance analysis
- **Challenge**
  - Ability to identify and correctly attribute the power weights to the individual “operation” components and map back to the application

Power Measurement Wish List

- **Per H/W core or memory component**
  - How much does each H/W attribute (L1 cache, L2, cache, TLB, etc) draw?
  - Get relative power usage of components
  - Get information on joules per operator or per access
    - Example: how many joules does a DCACHE hit or miss use per reference?
  - Provide core utilization metrics
- **Overall program power measurement**
  - Similar to wallclock (total watts or joules)
  - Multiple power meters
    - 1 for core
    - 1 for memory system
- **Correlating HWPCs to power usage in performance tools**
  - Can be directly or indirectly
    - Roughly measured with existing H/W performance counters

Adaptive Power Management

- **Statically managed**
  - Compiler support
    - Compile-time options to direct optimization for maximum or minimum
      - e.g., -P 1, 2, 3 like current -O 1, 2, 3
    - Compiler directives
      - Put this data there and keep it there until I tell you to release it
  - Application-selected power modes/algorithms

- **Runtime managed**
  - Based on compiler hints
    - Which components are about to be utilized more or less heavily
      - Integer algorithm coming up
      - Intensive floating point region coming up
      - Poor locality coming up, turn off cache use for this next segment
  - Auto-tuning
    - Auto-tuning for performance and for power (and a combination of both)
  - User Guided
    - Runtime assertions/options/hints

Outline

- Computer architecture and applications trends
- Programming Environment mission
- Tools focus for extreme scale computing
- **Open Issues**
- **Conclusions**

How About Resilience?

- **On extreme scale systems the probability of a node failure is substantial**
- **Extreme-scale applications will have built-in resilience support**
  - There will be support for detection of failure and information about which images failed
    - e.g., error return for synchronization and collective operations can indicated that an image has failed
- **Will tools be ready for resilient applications?**
- **Will tools be resilient themselves?**

Other Issues to Think About

- **Running on an extreme-scale environment**
  - Target **scalability** issues in all areas of tool development
  - **Volume of data**
    - Need to rethink the performance measurement approach
      - Can we focus only on things that are different?
      - What kind of runtime aggregation will be possible?
      - How about the “Heisenbug” effect?
  - What can you do to prep the tools?
    - Availability of extreme-scale systems will be limited!
  - Cost/benefit estimates (porting apps)
    - Can you provide an estimate to the user of the improvements that certain transformations will bring?
  - Ease of use
    - Automatic program instrumentation
    - Automatic analysis

CrayPat-lite

Access light version of performance tools software

```
> module load perftools-lite
```

Build program

```
> make
```

a.out (instrumented program)

Run program (no modification to batch script)

```
aprun a.out
```

Condensed report to stdout
a.out*.rpt (same as stdout)
a.out*.ap2
MPICH_RANK_XXX files

Conclusions

- **Extreme scale systems will be more parallel, with more processors per node, more threads per processor, longer vector lengths, more complex memory hierarchies, and …**
- **Application developers will need sophisticated tools and adaptive runtime systems to help them address issues of scale and complexity of these systems**
- **Extreme scale tools should address issues of**
  - Performance
    - Help users maximize the cycles to the application
  - Programmability
    - Ease developing, porting, and tuning efforts
  - Power
    - Help the user understand the tradeoffs between power and performance, and tune for both