Profiling and debugging

Mandes Schönherr

Compiler feedback

- **Compiler can print annotated intermediate source files**

source with line numbers

```
270.   + 1 b----------<      DO k = 2,kmax-1
271.   + 1 b b--------<       DO j = 2,jmax-1
272.     1 b b Vr3----<        DO i = 2,imax-1
273.     1 b b Vr3               S0 = a(i,j,k,1)* ...
```

loop handling annotations

additional messages below

- **Generate listing files**
  - CCE:

    `FLAGS+=-hlist=a`

    Produces commentary files <stem>.lst
  - PGI:

    `FLAGS+=-Minfo`

    Feedback to STDERR
  - Compiler teams work very hard to make feedback useful
  - advice: use it, it's free!
    (i.e. no impact on performance)

Example Loop mark

G = accelerated

g = partioned

b = blocked loops

+ = additional message below

```
269.  + 1 G----------< !$omp target teams distribute reduc...
270.    1 G g--------<     DO k = 2,kmax-1
271.  + 1 G g b------<      DO j = 2,jmax-1
272.    1 G g b gb---<       DO i = 2,imax-1
273.    1 G g b gb              S0 = a(i,j,k,1)*p(i+1,j,k) &
. . .
ftn-6405 ftn: ACCEL File = himeno_F_v01.F90, Line = 269
A region starting at line 269 and ending at line 293 was placed
  on the accelerator.
```

Runtime commentary

- **Cray runtime commentary**
  - Compile with CCE (no special options)
  - Set environment variable:
    `CRAY_ACC_DEBUG=2`
  - Run the executable
  - The commentary is written to STDERR

- **Advice:**
  - Don't set both:
    `CRAY_ACC_DEBUG`
    with
    `COMPUTE_PROFILE`

- **NVIDIA Compute Profiler**
  - Event-by-event timing information
  - Compile with CCE (no special options)
  - Set environment variable:
    `COMPUTE_PROFILE=1`
  - Run the executable
  - The commentary is written to file cuda_profile_0.log

```
ACC: Start transfer 16 items from himeno_F_v01a.F90:153
ACC:       allocate, copy to acc 'a' (136855584 bytes)
. . .
ACC:       allocate reusable <internal> (1008 bytes)
ACC: End transfer (to acc 444780680 bytes, to host 0 bytes)
ACC: Execute kernel jacobi_$ck_L153_1 blocks:126 threads:
            128 async(auto) from himeno_F_v01a.F90:153
ACC: Wait async(auto) from himeno_F_v01a.F90:153
```

```
$> cat cuda_profile_0.log
. . .
method=[ memcpyHtoD ] gputime=[ 22636.031 ] cputime=[ 22714.756 ]
method=[ memcpyHtoD ] gputime=[ 16801.504 ] cputime=[ 16814.359 ]
method=[ memcpyHtoD ] gputime=[ 1.088 ] cputime=[ 7.279 ]
. . .
method=[ jacobi_$ck_L153_1 ] gputime=[ 3737.728 ]
                cputime=[ 15.326 ] occupancy=[ 0.312 ]
method=[ memcpyDtoH ] gputime=[ 2.400 ] cputime=[ 19.159 ]
method=[ memcpyDtoH ] gputime=[ 15007.552 ] cputime=[ 15675.952 ]
```

CrayPAT

Evaluate program behavior on Cray supercomputer systems …

Sampling

Advantages

- Only need to instrument main routine
- Low Overhead – depends only on sampling frequency
- Smaller volumes of data produced

Disadvantages

- Only statistical averages available
- Limited information from performance counters

Tracing/Instrumentation

Advantages

- More accurate and more detailed information
- Data collected from every traced function call not statistical averages

Disadvantages

- Increased overheads as number of function calls increases
- Huge volumes of data generated

The best approach is *guided tracing*.
e.g. Only tracing functions that are not small (i.e. very few lines of code) and contribute a lot to application's run time.
APA is an automated way to do this.

Profiling the application

CRAY

Perftools

- **lite** — sampling
- **lite_events** — tracing
- **lite_loops** — tracing with loop measurment
- **lite_gpu** — tracing also on the accelerator
- **(full) craypat** — Full adjustable
  - pat_build (instrumentation)
  - pat_report (data collection filtering)

```
Experiment:     lite lite/event_profile

Table 1:  Profile by Function Group and Function
(top 4 functions shown)

  Time% |      Time | Imb. |  Imb. |    Calls |Group
```

```
Experiment:     lite lite/gpu

Table 1:  Accelerator Table by Function (top 10 functions shown)

   Host | Host | Acc | Acc Copy | Acc Copy | Events |Function=[max10]
  Time% | Time | Time |      In |      Out |        | PE=HIDE
        |      |      | (MBytes) | (MBytes) |        |  Thread=HIDE

 100.0% | 0.03 | 0.62 |   848.36 |    65.26 |  1,038 |Total
|-----------------------------------------------------------------------------
|  72.3% | 0.02 | 0.10 |   424.18 |    32.63 |    402 |main_.ACC_COPY@li.174
|  13.8% | 0.00 | 0.42 |       -- |       -- |    200 |main_.ACC_ASYNC_KERNEL@li.174
|   5.7% | 0.00 |   -- |       -- |       -- |    201 |main_.ACC_SYNC_WAIT@li.174
|   4.5% | 0.00 | 0.08 |   424.18 |    32.63 |     14 |main_.ACC_COPY@li.153
|   2.3% | 0.00 |   -- |       -- |       -- |    200 |main_.ACC_REGION@li.174
|=============================================================================
```

Profiling the application

- **More details of CrayPAT lite output**
  - Reanalyze the data using `pat_report` file and e.g. -T option

```
$> pat_report -T *.ap2
```

```
...
Experiment:    lite lite/gpu

Table 1:  Profile by Function Group and Function
  Time% |      Time | Imb. |  Imb. |    Calls |Group
        |           | Time | Time% |          | Function
        |           |      |       |          |  Thread=HIDE

 100.0% | 1.273765 |   -- |    -- | 9,763.0 |Total
|-----------------------------------------------------------------------------
|  86.2% | 1.098388 |   -- |    -- | 8,714.0 |ETC
||----------------------------------------------------------------------------
||  33.5% | 0.426703 |   -- |    -- |   208.0 |__cray_acc_hw_wait
||  20.2% | 0.257243 |   -- |    -- |     1.0 |_END
||  17.3% | 0.220493 |   -- |    -- |    11.0 |__cray_acc_hw_init
||  12.0% | 0.153406 |   -- |    -- |   837.0 |__cray_acc_hw_copy_host_to_acc
. . .
||============================================================================
|  13.8% | 0.175323 |   -- |    -- | 1,045.0 |USER
||----------------------------------------------------------------------------
||   9.9% | 0.126006 |   -- |    -- |     1.0 |initmt_.LOOP@li.212
...
```

```
...
Experiment:    lite lite/gpu

Table 3:  Time and Bytes Transferred for Accelerator Regions
  Host | Host |  Acc | Acc Copy | Acc Copy | Events |Calltree
 Time% | Time | Time |       In |      Out |        | Thread=HIDE
       |      |      | (MBytes) | (MBytes) |        |

 100.0% | 0.01 | 0.60 |   848.36 |    65.26 |  1,038 |Total
|-----------------------------------------------------------------------------
| 100.0% | 0.01 | 0.60 |   848.36 |    65.26 |  1,038 |main_
|        |      |      |          |          |        | jacobi_
|||---------------------------------------------------------------------------
3||  90.7% | 0.00 | 0.44 |     0.00 |     0.00 |  1,030 |jacobi_.REGION@li.279
4||        |      |      |          |          |        | jacobi_
3||   9.3% | 0.00 | 0.16 |   848.35 |    65.26 |      8 |jacobi_.ACC_DATA_REGION@li.276
||||--------------------------------------------------------------------------
4|||   8.6% | 0.00 | 0.15 |   848.35 |       -- |      2 |jacobi_.ACC_COPY@li.276
4|||   0.5% | 0.00 | 0.01 |       -- |    65.26 |      2 |jacobi_.ACC_COPY@li.329
4|||   0.2% | 0.00 |   -- |       -- |       -- |      2 |jacobi_.ACC_DATA_REGION@li.276..
4|||   0.1% | 0.00 |   -- |       -- |       -- |      2 |jacobi_.ACC_SYNC_WAIT@li.329
|=============================================================================
...
```

Profiling using (fully) CrayPAT

```
$> module load perftools-base
$> module load perftools
$> pat_build <exe>
```

- Instrumentation
- set what/how is measured
- Create new executable `<exe>+pat`

Option	Description
	Sampling profile
`-w`	Tracing is default experiment
`-u`	tracing of functions in user source files
`-T <func>`	Specifies a function which will be traced
`-t <file>`	Tracing all functions in the specified file
`-g <group>`	Instrument functions in group, e.g. omp, mpi, cuda, blas, io, netcdf, syscall

Run job with new executable

```
$> pat_report *.xf
```

- Measurement data analysis
- Generates ap2 (compressed) file and text report
- ap2 is input for pat_report, Apprentice², Reveal

Option	Description
`-O <table opt>`	Variety of table options
`-sfilter-input=...`	Defines filter, e.g. 'pe%2==0'
`-o`	Defines report output file name
`-f <format>`	Output format, e.g. plot, rpt, ap2, ap2-xml, ap2-txt, xf-xml, xf-txt, html

Interlude: A synchronous profile

- **GPU kernels launch asynchronously**
  - So the compute time shows up in the SYNC_WAIT events
- **We can switch off the automatic asynchronicity**
  - This can give a clearer profile, but it may be skewed
  - Recompile with CCE flag -hacc_model=auto_async_none
  - And do a new CrayPAT profile

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

  Time%  |     Time   | Imb.  |  Imb.   | Calls   |Group
         |            | Time  | Time%   |         | Function

 100.0% | 0.755941 |    -- |     -- | 1131.0 |Total
|-----------------------------------------------------------------------
| 100.0% | 0.755749 |    -- |     -- |  730.0 |USER
||----------------------------------------------------------------------
||  40.9% | 0.308847 |    -- |     -- |    1.0 |main_
||  28.4% | 0.214851 |    -- |     -- |  103.0 |jacobi_.ACC_SYNC_WAIT@li.320
||  24.1% | 0.182321 |    -- |     -- |    2.0 |jacobi_.ACC_COPY@li.280
||   4.3% | 0.032620 |    -- |     -- |  103.0 |jacobi_.ACC_COPY@li.293
||   1.0% | 0.007862 |    -- |     -- |    2.0 |jacobi_.ACC_COPY@li.340
|=======================================================================
```

avoid synchronization in profiling

Combining Sampling and Tracing: APA

- **Automatic Profiling Analysis:**
  - **Target:** large, long-running program (general a trace will inject considerable overhead)
  - **Goal:** limit tracing to those functions that consume the most time.
  - **Procedure:** use a preliminary sampling experiment to determine and instrument functions consuming the most time
  - Note: Accelerator performance data for sampling experiments is not supported

```
$> module load perftools-base
$> module load perftools
```

```
$> pat_build <exe>
```

- The APA is the default experiment. No option needed.
- A new executable is created with a name like <exe>**+pat**

```
$> qsub job.pbs #aprun ... <exe>+pat
$> pat_report -o myrep.txt <exe>+pat+*
```

- Applying pat_report to the *.xf generates an ***.apa** file in addition to the *.ap2 file.

```
$> vi *.apa
```

- The *.apa file contains instructions for the next instrumentation step.
- Modify it according to your needs.

```
$> pat_build -O *.apa
```

- Generates an instrumented binary <exe>**+apa** for tracing

```
$> qsub job.pbs #aprun ... <exe>+apa
$> pat_report -o myrep.txt <exe>+apa+*
```

- Applying `pat_report` to the *.xf generates a new*.ap2 file.

Loop work estimates

- **Assess suitability of loop nests for porting to the GPU**
  - Requires information on inclusive time spent in the loop nests and typical trip count of the loops.
  - CrayPAT can generate this information via a special kind of tracing experiment.

- **Load the performance tools**

```
$> module load perftools-base
$> module load perftools
```

- **Recompile your program for gathering loop statisitcs**

```
$> ftn -c -h profile_generate my_program.f
$> ftn -o my_program -h profile_generate my_program.o
```

- **Instrument the application for tracing (APA also possible)**

```
$> pat_build –w[-u] my_program
```

- **This generates a new binary named `my_program+pat`.**

- **Execute the instrumented program.**

```
> aprun –n #PE ./my_program+pat
```

- **This generates one or more raw data files(s) in .xf format. Process the raw data files(s) for use by Reveal.**

```
> pat_report -o report.txt my_porgram*.xf
```

  - This generates a performance data file `my_program.ap2` and text report `report.txt`.

- **Reveal can use the `my_program.ap2` to visualize time expensive loops.**

- **It is recommended to turn off OpenMP and OpenACC for the loop work estimates via `–h noomp –h noacc`**

Profiling accelerated codes

- CrayPAT tracing offers a powerful profiling for accelerated codes. (Sampling does not collect GPU performance data)
- Load the GPU module and the performance tools
  - `module load craype-accel-nvidia60`
  - `module load perftools-base perftools`

- Recompile your program
  - `ftn -c my_program.f`
  - `ftn -o my_program my_program.o`

- Instrument the application for tracing and execute
  - `pat_build -w my_program`
  - `aprun -n pes my_program`

- Generate a report out of the raw data file(s)
  - `pat_report -o report.txt my_porgram*.xf`

Contents of report.txt (Table 1 and 2)

```
Table 1:  Profile by Function Group and Function
  Time% |      Time | Imb. |  Imb. |    Calls |Group
        |           | Time | Time% |          | Function
        |           |      |       |          |  Thread=HIDE
 100.0% | 1.222670 |   -- |    -- | 1,046.0 |Total
|---------------------------------------------------------------------
|  79.0% | 0.965458 |   -- |    -- | 1,043.0 |USER
||--------------------------------------------------------------------
||  34.5% | 0.421250 |   -- |    -- |   201.0 |main_.ACC_SYNC_WAIT@li.177
||  24.0% | 0.293347 |   -- |    -- |     1.0 |main_.ACC_COPY@li.148
||  17.2% | 0.209976 |   -- |    -- |     1.0 |main_
||====================================================================
|  21.0% | 0.257213 |   -- |    -- |     3.0 |ETC
||------------------------------------------------------------------
```

```
Table 2:  Load Imbalance by Thread

     Max. | Imb. |  Imb. |Thread
     Time | Time | Time% |

 1.222704 |   -- |    -- |Total
|-----------------------------------
| 1.222704 |   -- |    -- |thread.0
|===================================
```

- ACC_COPY lines report data movements
- ACC_KERNEL is essentially zero as it is launched asynchronously.
- GPU time is measured in the ACC_SYNC_WAIT
- ACC_REGION measures internal ops (set up pointers for transfer.)

Contents of report.txt (Table 3)

```
Table 3:  Time and Bytes Transferred for Accelerator Regions

   Host | Host |  Acc | Acc Copy | Acc Copy | Events |Calltree
  Time% | Time | Time |       In |      Out |        | Thread=HIDE
        |      |      | (MBytes) | (MBytes) |        |
 100.0% | 0.76 | 0.52 |   424.18 |     0.00 |  1,042 |Total
|-----------------------------------------------------------------------------
| 100.0% | 0.76 | 0.52 |   424.18 |     0.00 |  1,042 |main_
|        |      |      |          |          |        | main_.ACC_DATA_REGION@li.148
|||---------------------------------------------------------------------------
3||  59.3% | 0.45 | 0.43 |     0.01 |     0.00 |  1,004 |main_.ACC_DATA_REGION@li.177
4||  59.3% | 0.45 | 0.43 |     0.01 |     0.00 |  1,000 | main_.ACC_REGION@li.177
|||||-------------------------------------------------------------------------
5|||  55.8% | 0.42 |   -- |       -- |       -- |    200 |main_.ACC_SYNC_WAIT@li.177
5|||   3.1% | 0.02 | 0.01 |     0.01 |     0.00 |    400 |main_.ACC_COPY@li.177
```

- Table shows details on data transfers and timings for CPU and GPU.
- ACC_SYNC_WAIT time on CPU includes kernel time on GPU.
- For MPI programs the statistics are averaged over the PE.

Profiling with GPU Hardware Counters

- CrayPAT supports a wide range of accelerator performance counter
- A predefined set of groups has been created for ease of use (combines events that can be counted together.)
  - `module load perftools-base perftools`
  - `man accpc`
  - `more $CRAYPAT_ROOT/share/CounterGroups.nvidia_k20x`
- Enable collection similarly to CPU counter collection
  - PAT_RT_PERFCTR=group or events
- Set the PAT_RT_ACCPC variable appropriately and run the instrumented (tracing) program.

Apprentice^2

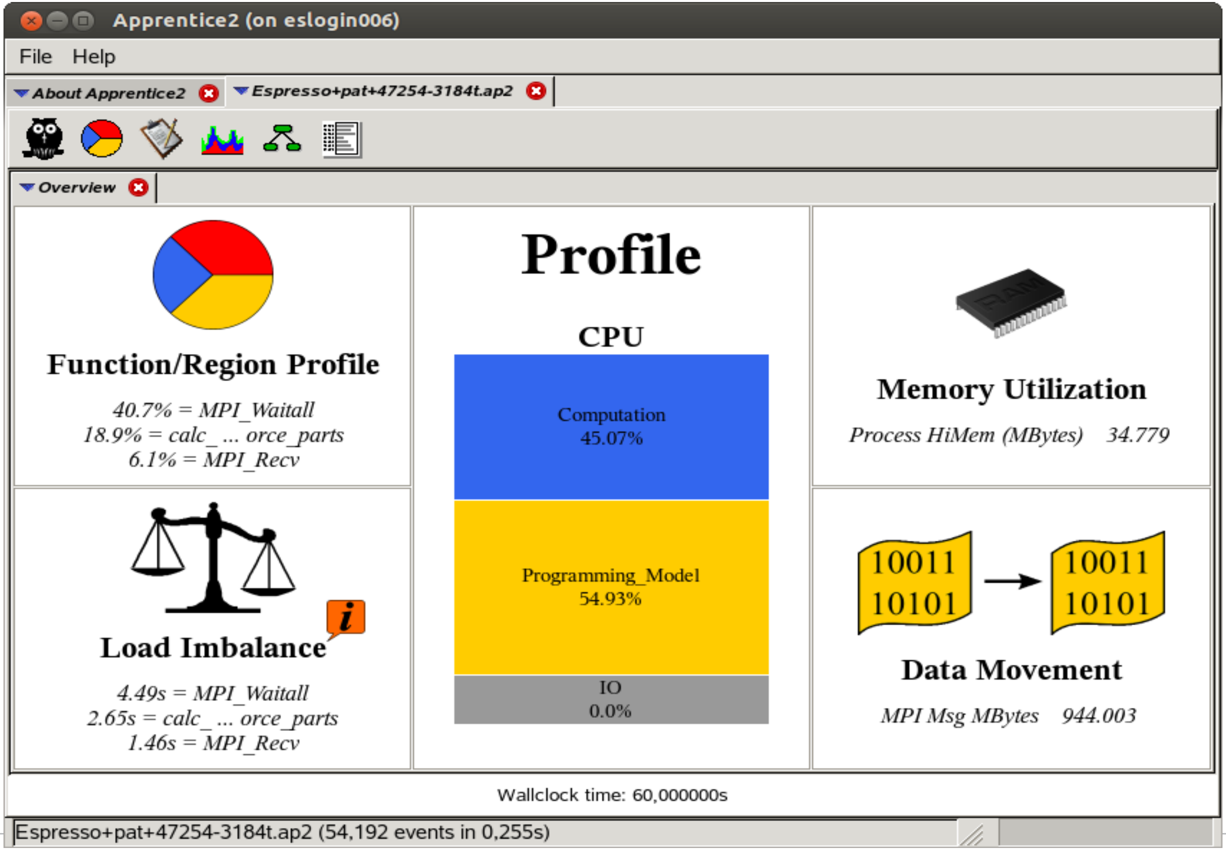
Graphical representation of performance data

Cray Apprentice2

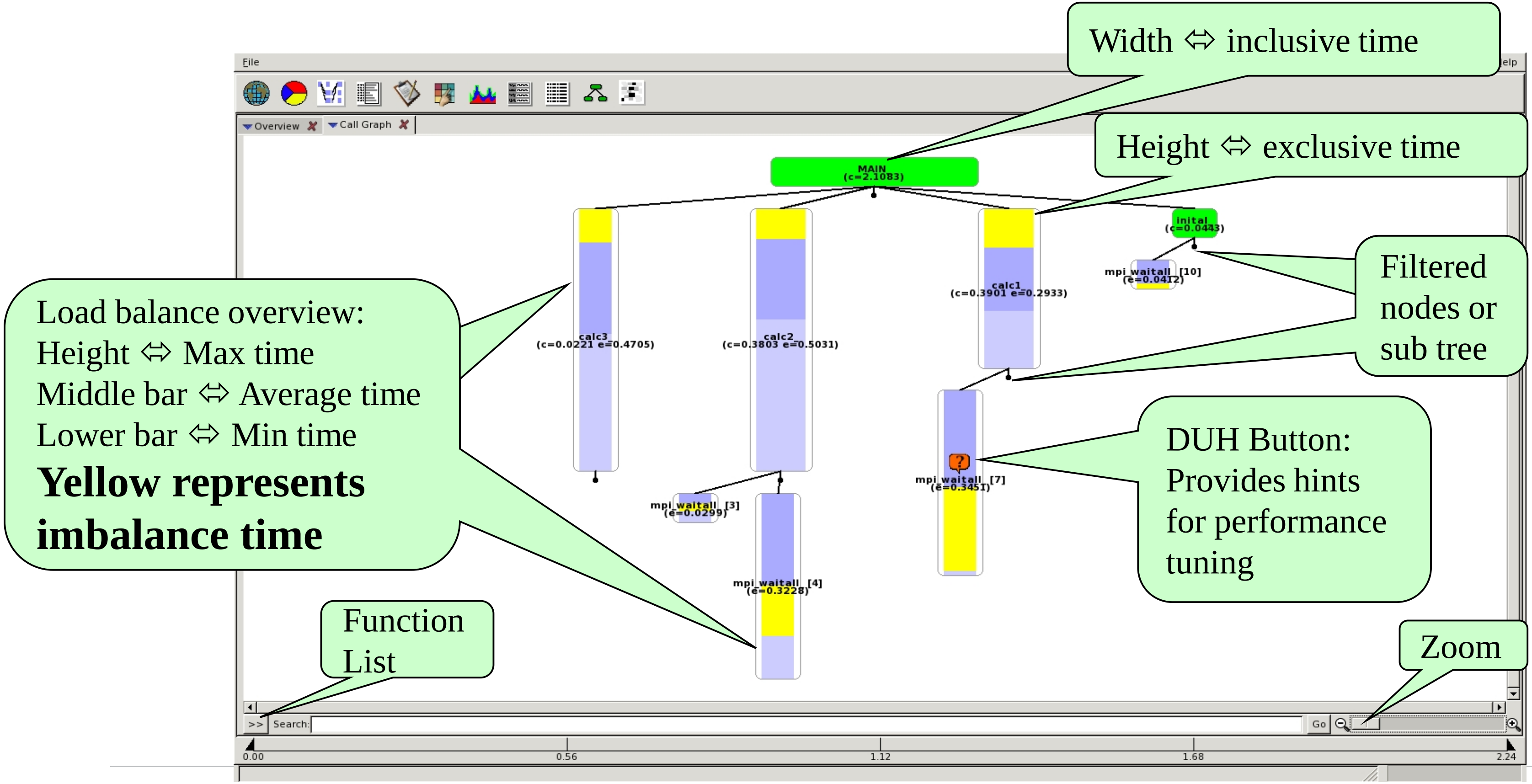
- **Cray Apprentice2 is a post-processing performance data visualization tool. Takes *.ap2 files as input.**
- **Main features are**
  - Call graph profile
  - Communication statistics
  - Time-line view for Communication and IO.
  - Activity view
  - Pair-wise communication statistics
  - Text reports
- **helps identify:**
  - Load imbalance
  - Excessive communication
  - Network contention
  - Excessive serialization
  - I/O Problems

```
$> module load perftools-base
$> app2 my_program.ap2 &
```

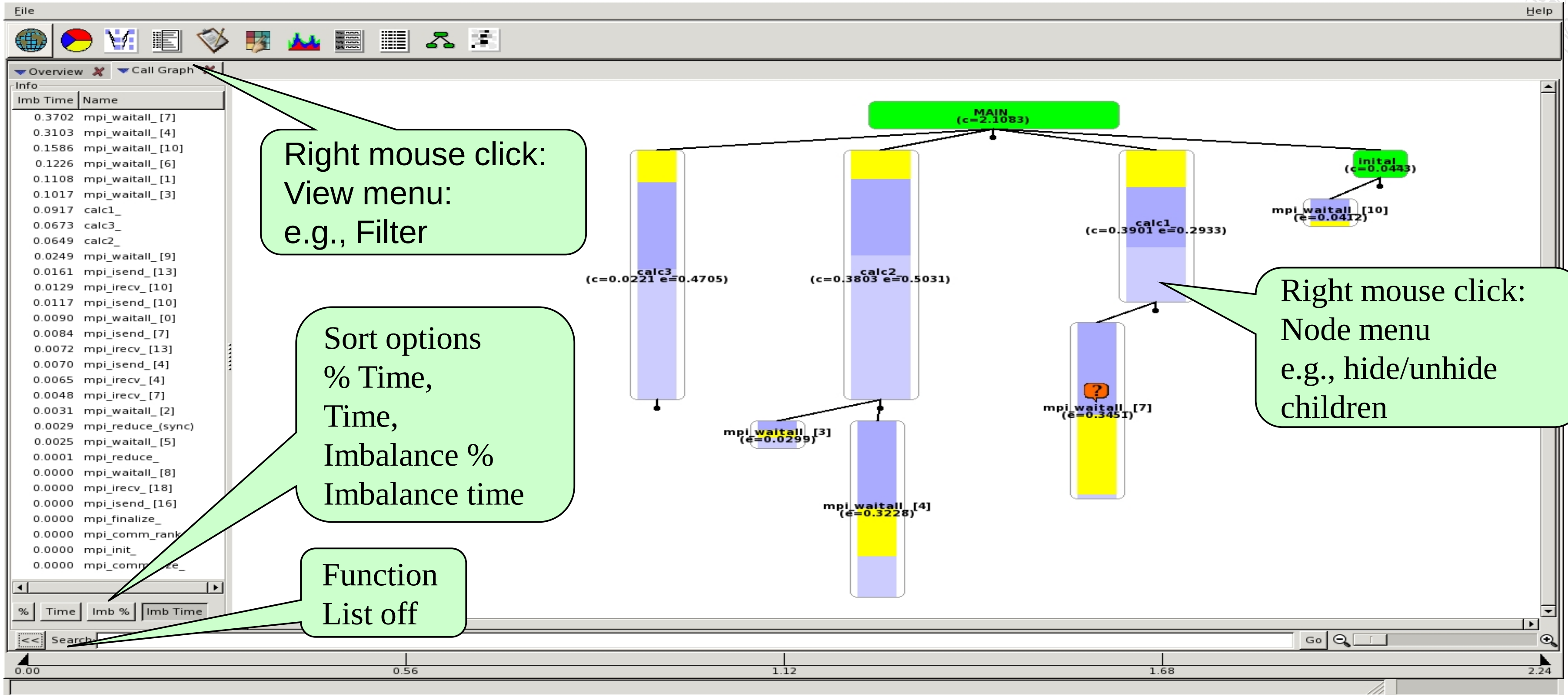
Cray Apprentice²



Call Tree View



Call Tree View – Function List

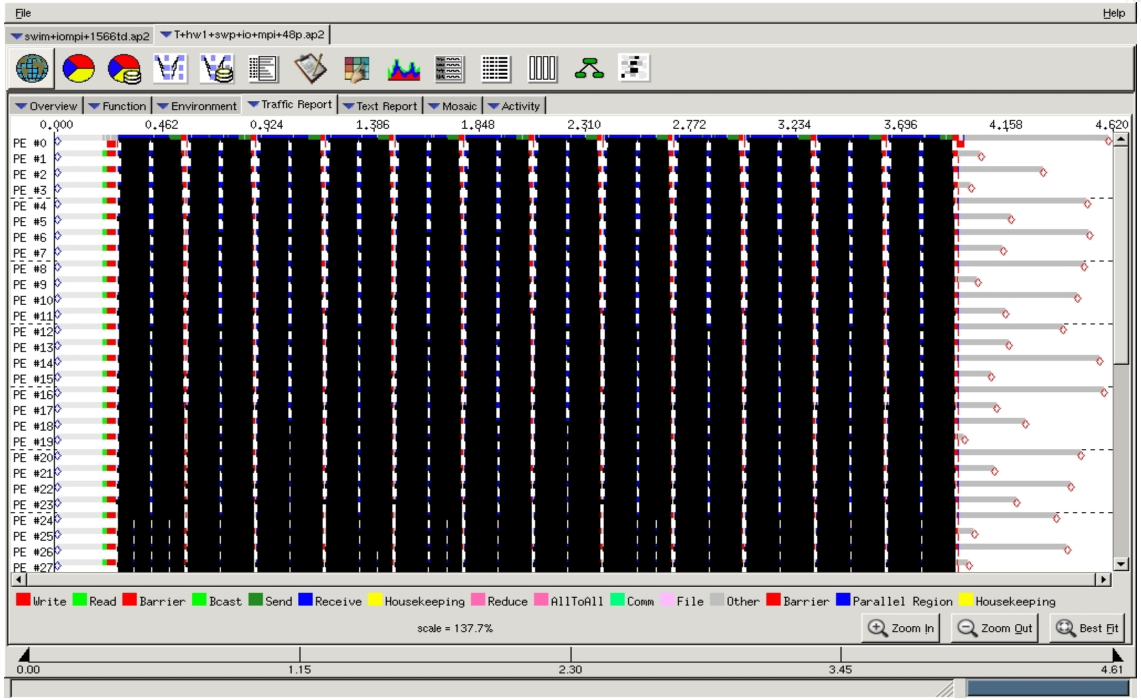


COMPUTE | STORE | ANALYZE

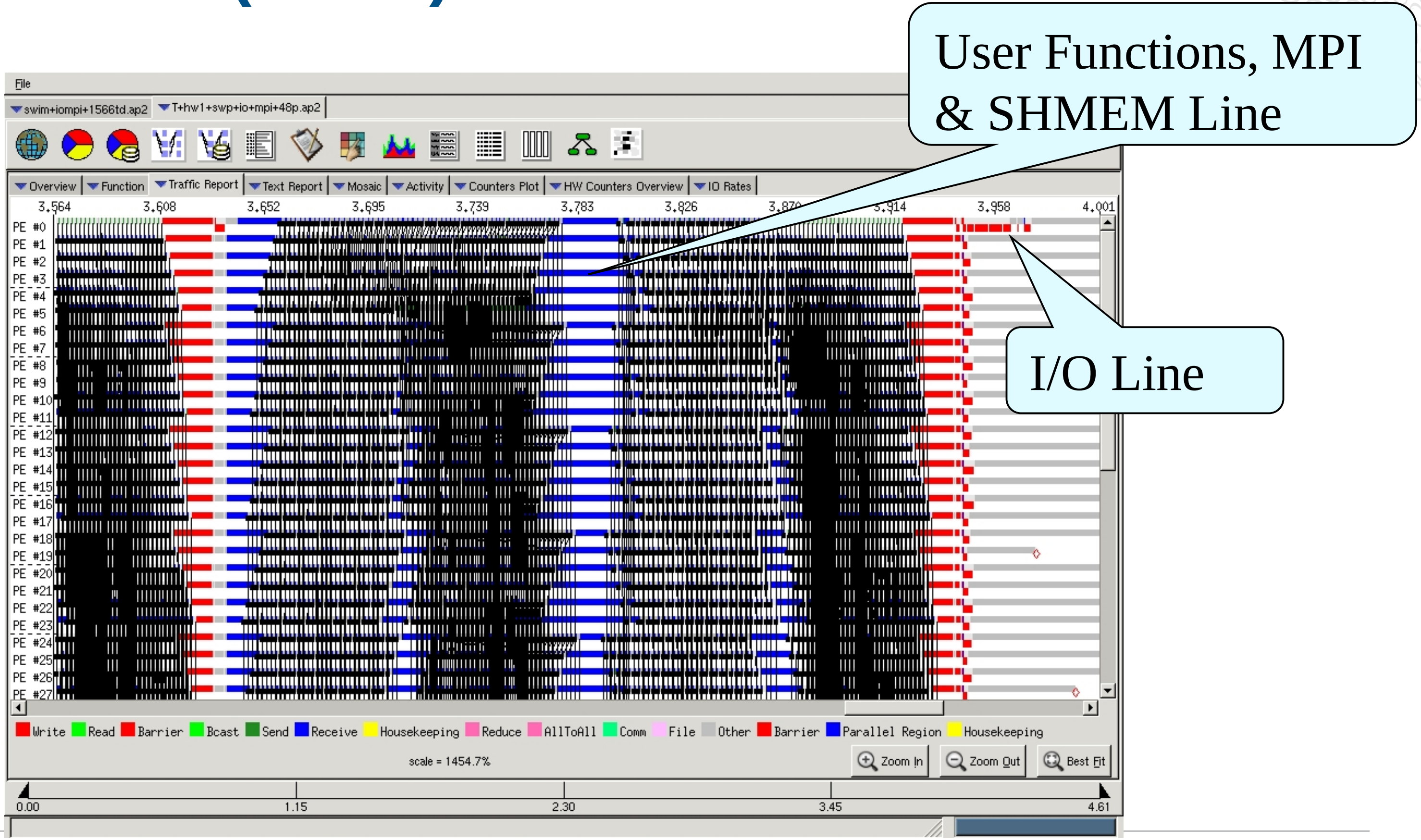
Load Balance View (from Call Tree)

Time Line View

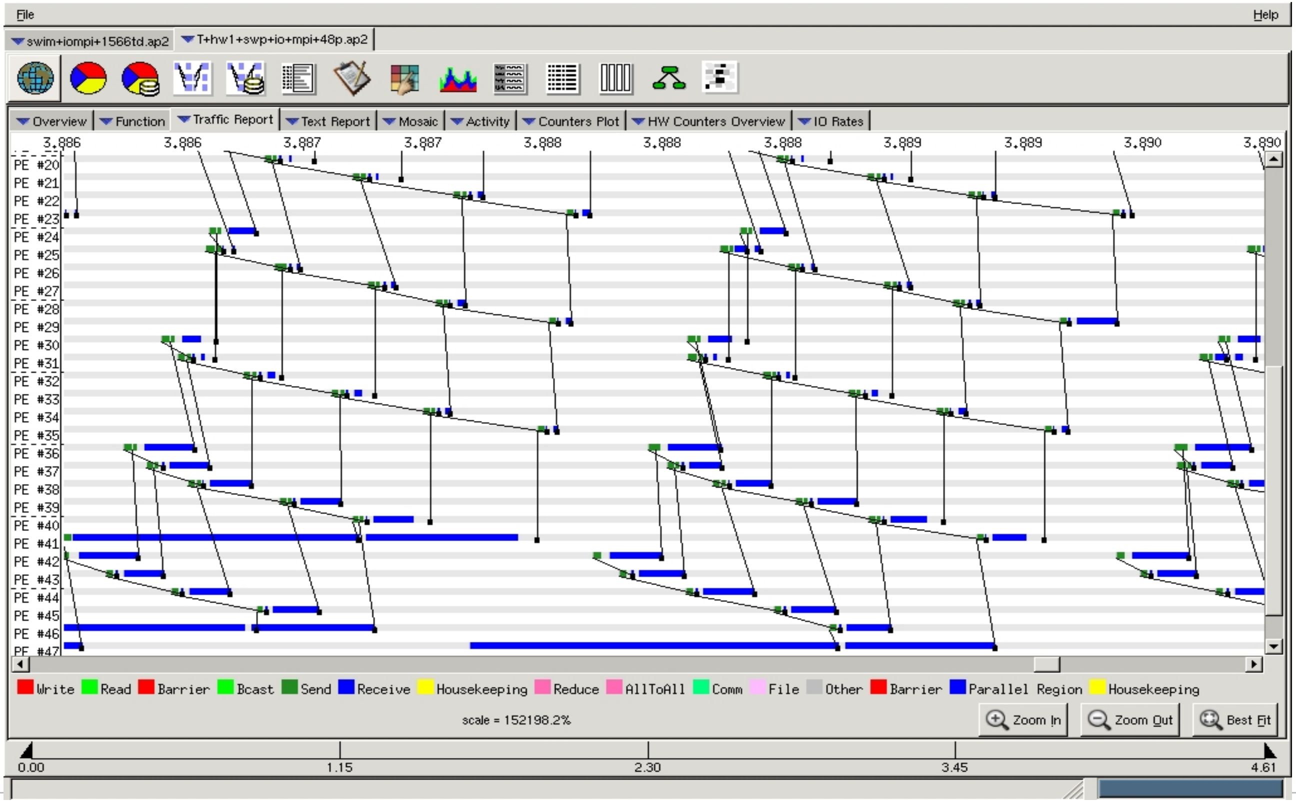
- Full trace (sequence of events) enabled by setting **PAT_RT_SUMMARY=0**
- Helpful to see communication bottlenecks.
- Use it only for small experiments !



Time Line View (Zoom)



Time Line View (Fine Grain Zoom)



Reveal

- Variable scoping and loopmark listing

Overview

- **As with OpenMP , in order to use the OpenACC directives one has to understand the scoping of the variables, i.e. whether variables are shared or private.**
- **Reveal is Cray's next-generation integrated performance analysis and code optimization tool.**
  - Source code navigation using whole program analysis (data provided by the Cray compilation environment.)
  - Coupling with performance data collected during execution by CrayPAT. Understand which high level serial loops could benefit from parallelism.
  - Enhanced loop mark listing functionality.
  - Dependency information for targeted loops
  - Assist users optimize code by providing variable scoping feedback and suggested compile directives.

How to use Reveal (generate program library)

- **Load the performance tools**

```
> module load perftools-base
```

- **Recompile only sources to generate program library**

```
> ftn -O3 -hpl=my_program.pl -c my_program_file1.f90
> ftn -O3 -hpl=my_program.pl -c my_program_file2.f90
```

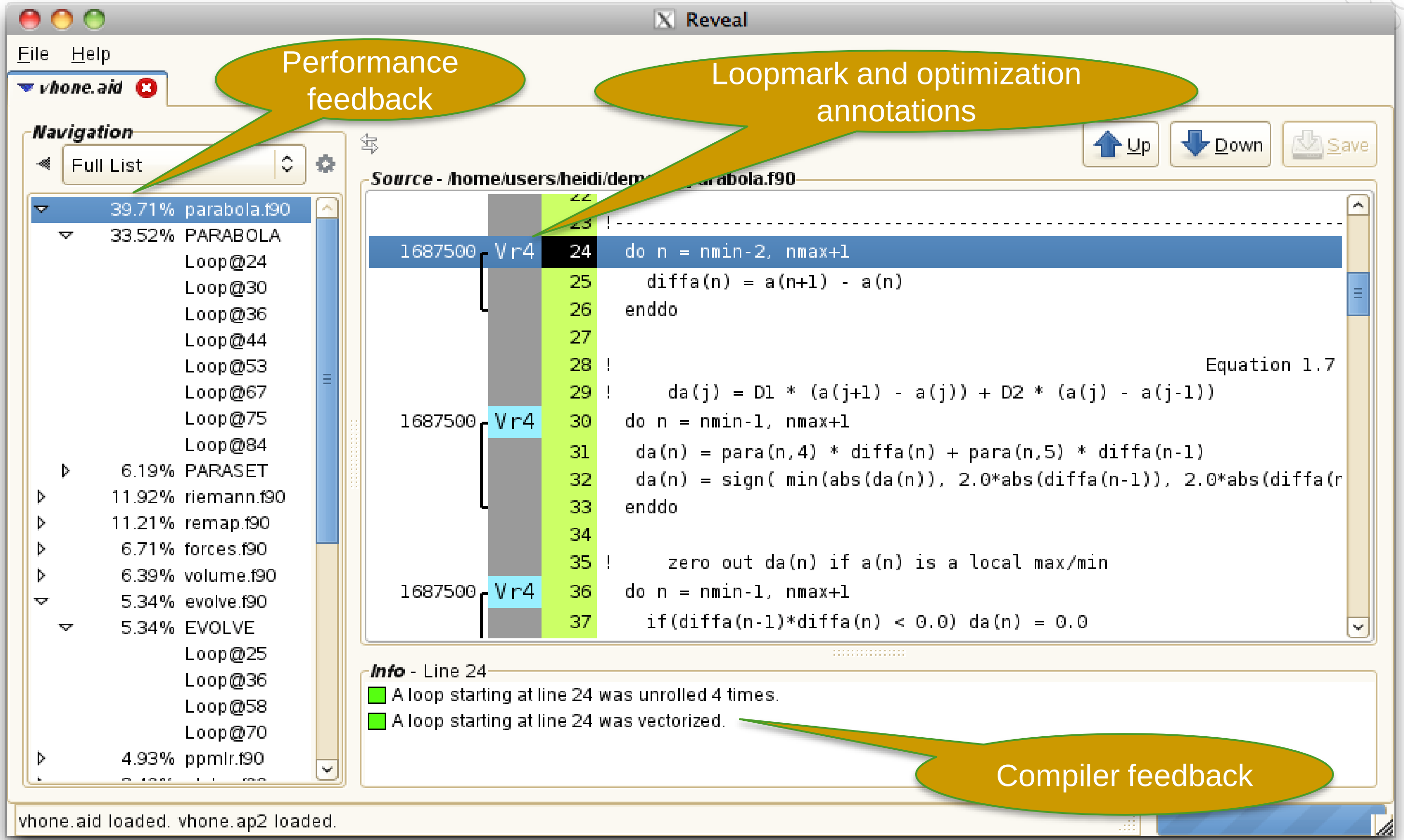
- **The program_library is most useful when generated from fully optimized code. Instrument the program for tracing.**

- **After the collection of performance data and generation of a program libaray you can launch Reveal**

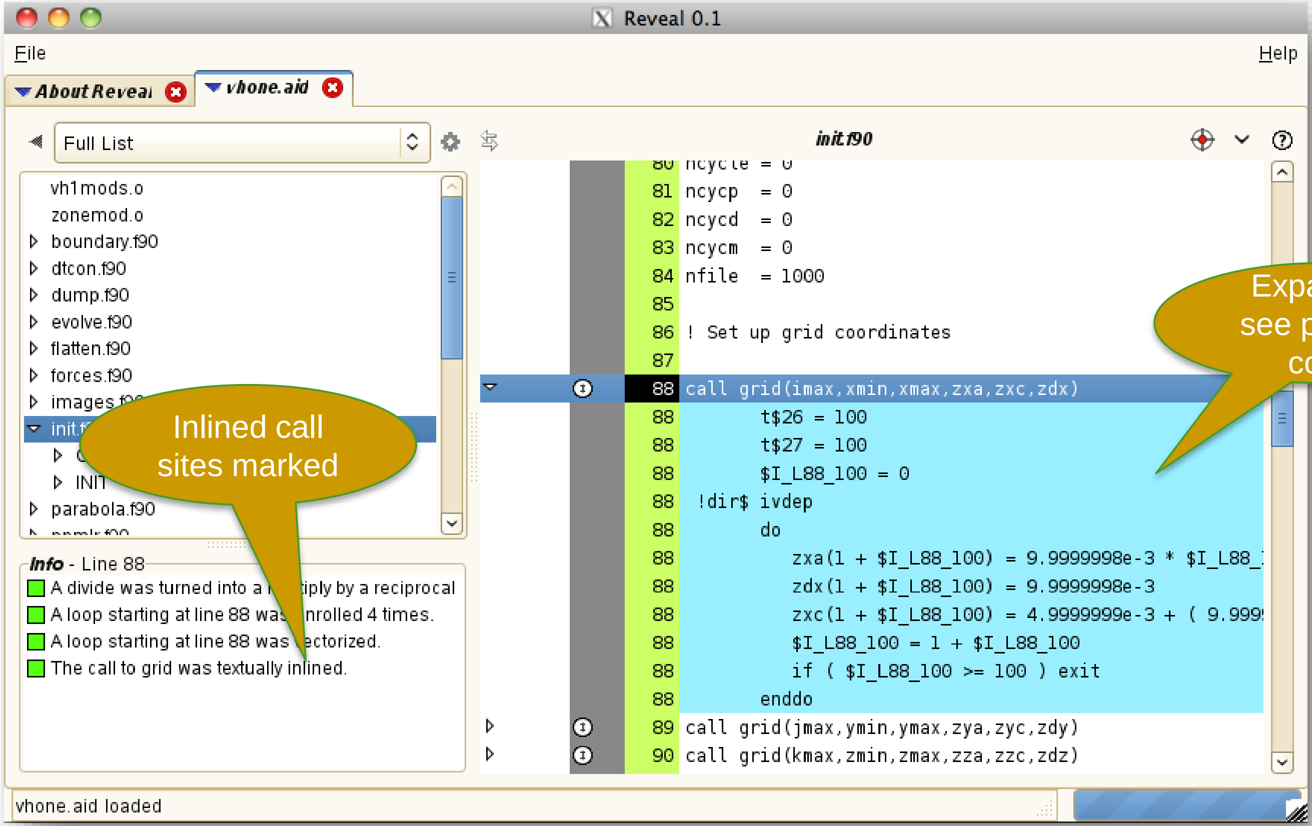
```
> reveal my_program.pl my_program.ap2
```

- **The *.ap2 is from a loop work estimate of my_program**
  - You can omit the *.ap2 and inspect only code listings.

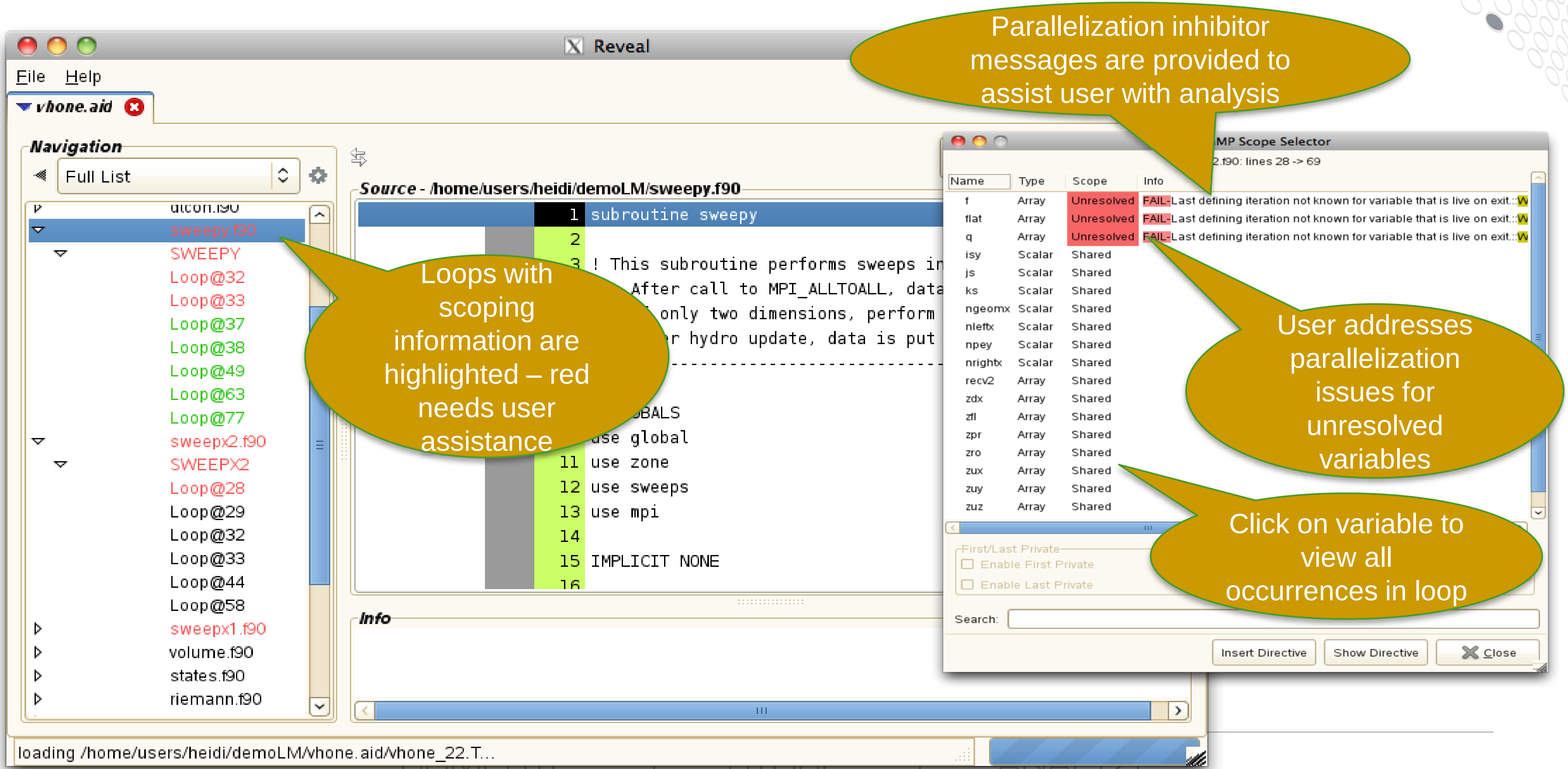
Visualize CCE's Loopmark with Performance Profile



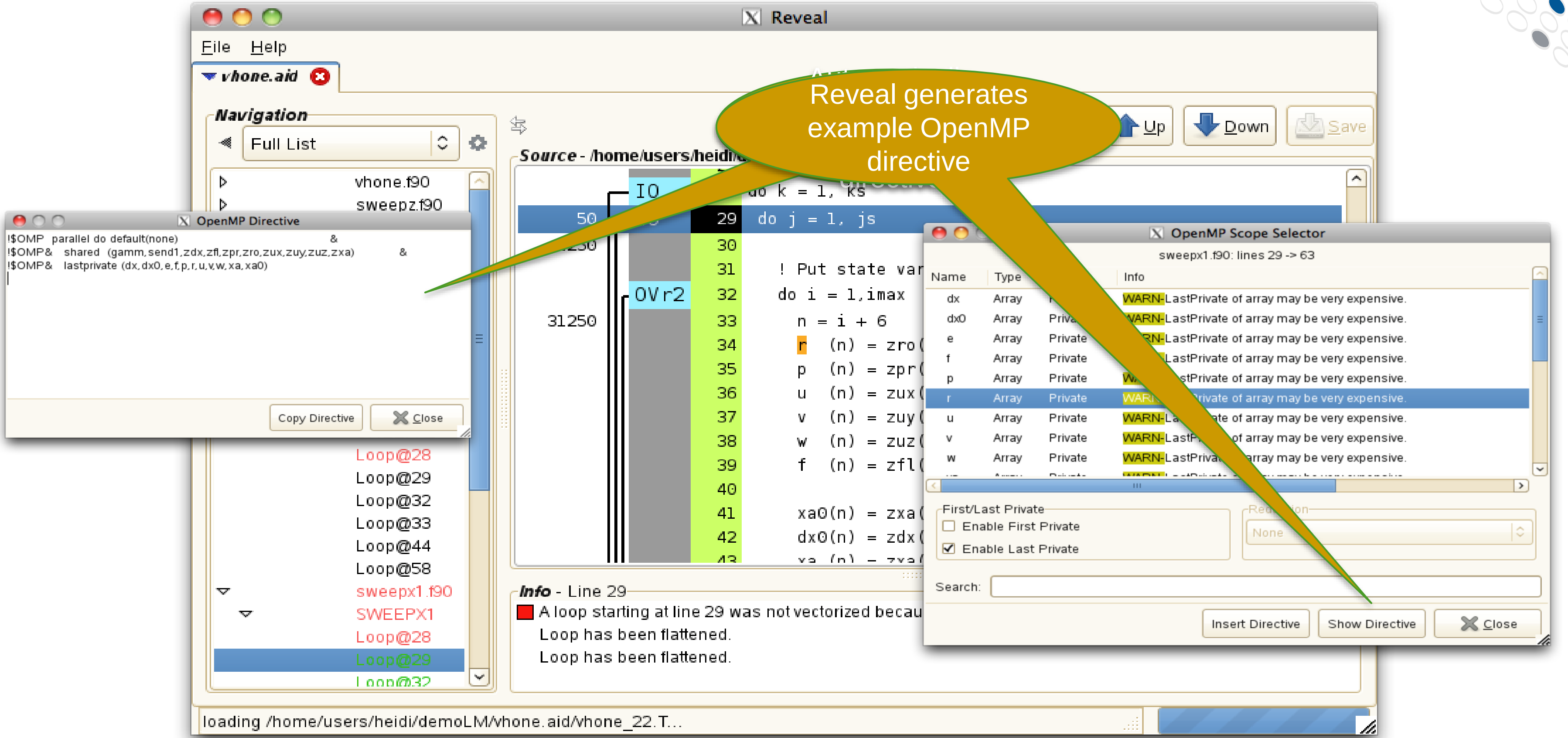
View Pseudo Code for Inlined Functions



Scoping Assistance – Review Scoping Results



Scoping Assistance – Generate Directive



Allinea DDT / Forge

Visual Debugging

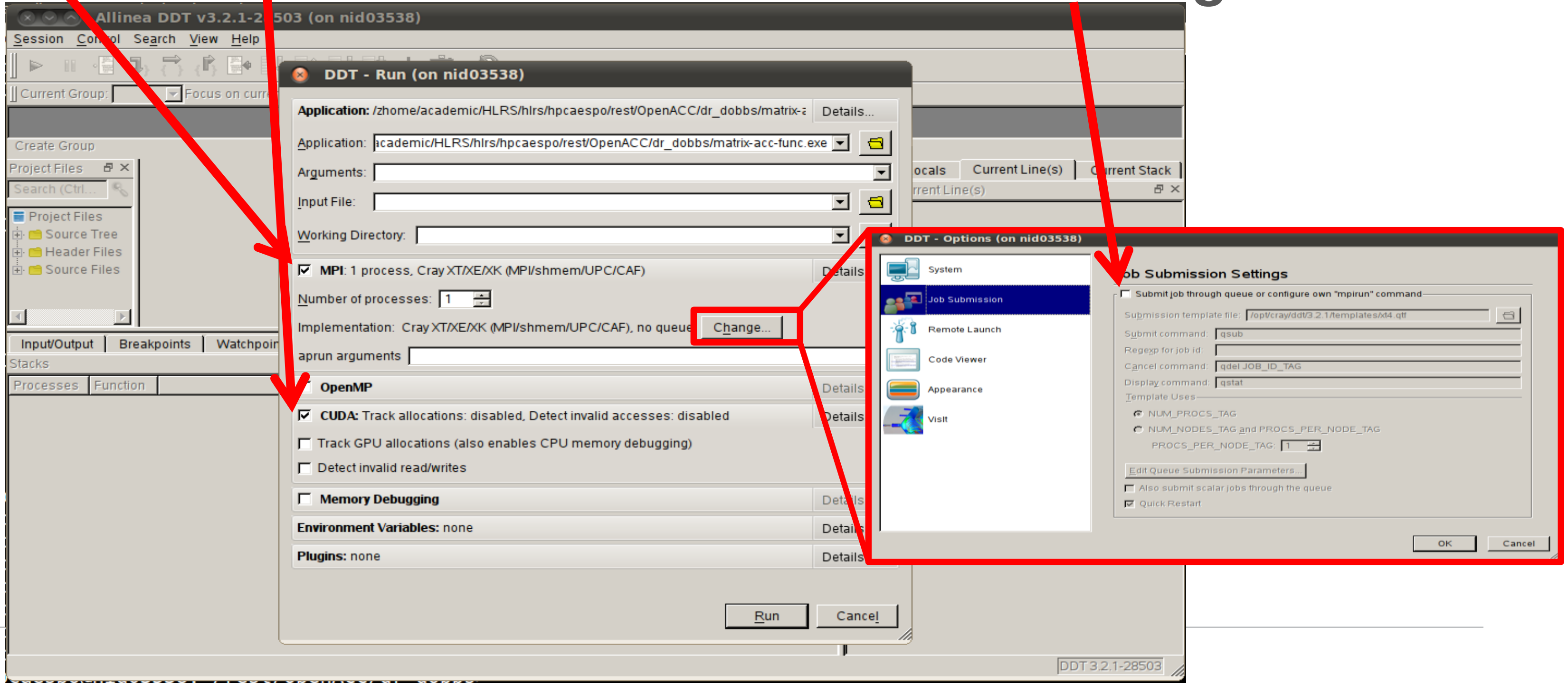
Overview

- **Graphical debugger designed for**
  - Distributed or shared memory parallelism or a combination.
  - Memory debugging, data analysis
  - C, C++, Fortran, UPC, CUDA, OpenACC
- **Recompile your application for DWARF support**
  - module load craype-accel-nvidia60
  - {CC,cc,ftn} -g <your_application>
- **Get an interactive session and load the debugger**
  - module load craype-accel-nvidia60
  - ddt [--connect] srun -C gpu -n1 -t15 ./exe

- **For sequential codes (no MPI) you need to add a dummy MPI_Init and MPI_Finalize.**

Using DDT for debugging OpenACC codes

- MPI and CUDA boxes should be already checked.
- Uncheck the box in the Job submission settings.



Summary and general remarks

- **Use CrayPAT to understand where your application is spending time.**
  - Automatic performance analysis based on tracing and sampling for large applications. Only tracing more efficient for smaller programs.
  - Loop work estimate to identify interesting loops to port to the GPU. Can also be done in the framework of the APA.
- **Use Reveal to better understand loop mark listings and do variable scoping for the interesting loops. Use the loop work estimates from the CrayPAT runs.**
- **Comparative debugging, e.g. comparing messages from different compiler (Cray, PGI, Nvidia, …) can be very helpful.**