Hewlett Packard Enterprise

Advanced Performance Analysis

Comprehensive General LUMI Course

April 23–26, 2024

Agenda

- In-depth performance analysis with `perftools`
  - Sampling, tracing, and loop work estimates
  - Automatic Performance Analysis
- Reveal
  - Compiler Feedback and Variable Scoping
- OpenMP profiling
- Perftools API
  - Customized performance analysis
- Hardware performance counters
- Load imbalance analysis
- pat_run
  - Profile existing dynamically linked executables
- Concluding remarks

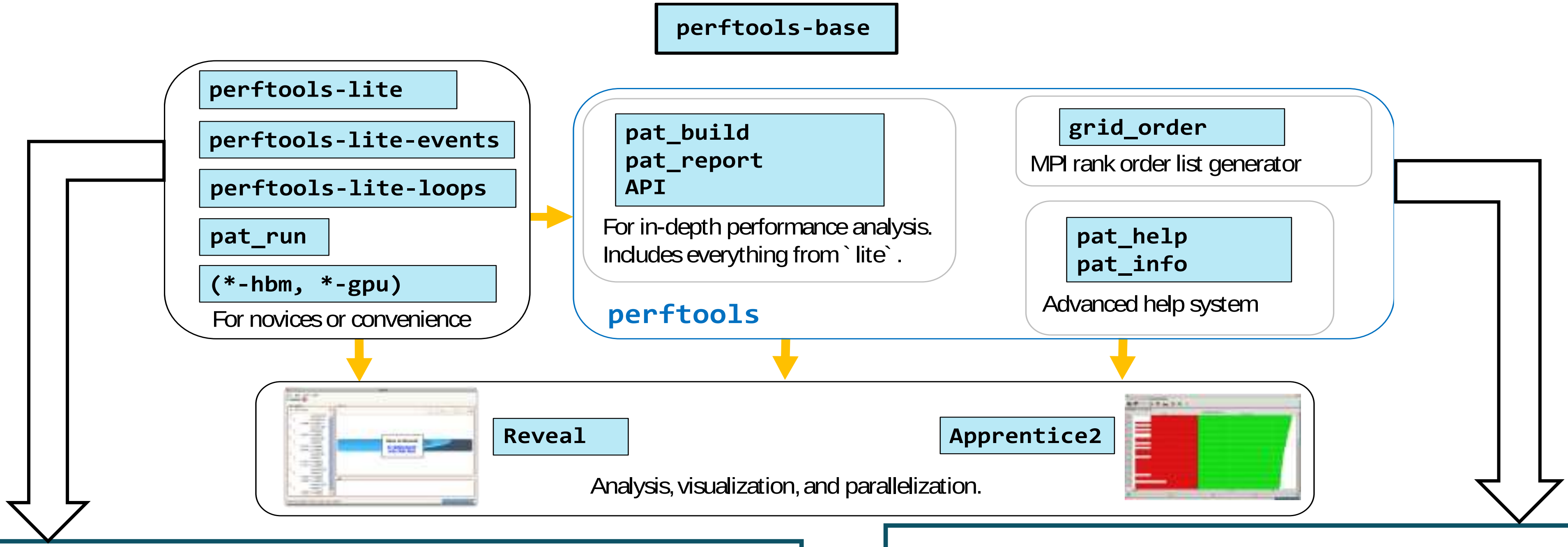
© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

Perftools

In-depth Perfomance Analysis

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

Overview



Previous talk:

1. Load the **perftools-lite*** module **wrt desired profiling**
2. Generate the binary
3. Run the binary
4. Look at the **automatically** generated report

This talk:

1. Load the **perftools** module
2. Generate the binary
3. **Manually** generate an **instrumented binary wrt desired profiling**
4. Run the **instrumented binary**
5. **Manually** generate the report
6. Look at the **manually** generated report

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

Sampling with Perftools

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

```
> make clean; make
> pat_build -S app.exe
```

- This generates a new executable `app.exe+pat` and preserves `app.exe`.
- Object files must be present during this stage.

```
> srun -n 8 ./app.exe+pat
> pat_report -o myrep.trace.rpt app.exe+pat+*/
```

- Running the `app.exe+pat` creates an `app.exe+pat+*/` directory.
- `pat_report` reads that directory and prints a lot of human-readable performance data.
- If worthwhile, a MPI rank reordering file will be produced.

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

Event Tracing using Perftools

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

```
> make clean; make
> pat_build -u -g mpi app.exe
```

- This generates a new executable `app.exe+pat` and preserves `app.exe`.
- If object files and user libraries have already been compiled with perftools enabled, just relink the application with `rm app.exe; make` instead.
- Traces MPI functions calls and functions defined in the program source files.

```
> srun -n 8 ./app.exe+pat
> pat_report -o myrep.trace.rpt app.exe+pat+*/
```

- Running the `app.exe+pat` creates a `app.exe+pat+*/` directory.
- `pat_report` reads that directory and prints human-readable performance data.

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

Output: Event Tracing with Perftools

General job information

```
CrayPat/X:  Version 6.4.1 Revision 6a6694f  06/27/16
17:24:11

Number of PEs (MPI ranks):    8
Numbers of PEs per Node:      8
Numbers of Threads per PE:    1
Number of Cores per Socket:  16
Execution start time:  Tue Mar  7 21:28:21 2017
System name and speed:  nid00036  2301 MHz (approx)
Intel haswell CPU  Family:  6  Model: 63  Stepping:  2
```

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


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

100.0% | 36.703706 |       -- |    -- | 3,036.0 |Total
|----------------------------------------------------------------
| 97.2% | 35.666406 |      -- |    -- |   457.0 |USER
||---------------------------------------------------------------
|| 93.8% | 34.432069 | 0.683771 |  2.2% |   2.0 |jacobi
||  3.4% |  1.232095 | 0.012880 |  1.2% |   1.0 |initmt
||  0.0% |  0.000941 | 0.001077 | 61.0% |   1.0 |main
||  0.0% |  0.000486 | 0.000232 | 36.9% | 150.0 |sendp
||  0.0% |  0.000356 | 0.000445 | 63.5% | 150.0 |sendp3
||  0.0% |  0.000266 | 0.000024 |  9.3% |   1.0 |initcomm
||  0.0% |  0.000172 | 0.000192 | 60.3% | 150.0 |sendp2
||  0.0% |  0.000020 | 0.000008 | 31.8% |   1.0 |exit
||  0.0% |  0.000002 | 0.000001 | 29.9% |   1.0 |initmaxclone_12853_1
||===============================================================
==========
```

All user defined routines traced and shown (-T option).

```
||----------------------------------------------------------------
|  2.8% |  1.019139 |       -- |    -- | 2,423.0 |MPI
||----------------------------------------------------------------
||  2.3% | 0.833848 | 0.688832 | 51.7% |  450.0 |MPI_Waitall
||  0.5% | 0.169196 | 0.025124 | 14.8% |  900.0 |MPI_Isend
||  0.0% | 0.012631 | 0.016372 | 64.5% |  900.0 |MPI_Irecv
||  0.0% | 0.001162 | 0.000014 |  1.3% |  152.0 |MPI_Allreduce
||  0.0% | 0.001077 | 0.000043 |  4.4% |    1.0 |MPI_Cart_create
||  0.0% | 0.000703 | 0.000038 |  5.8% |    3.0 |MPI_Type_commit
...
||================================================================|
0.0% | 0.018161 |       -- |    -- |  156.0 |MPI_SYNC
||----------------------------------------------------------------
||  0.0% | 0.014154 | 0.014143 | 99.9% |    2.0 |MPI_Barrier(sync)
||  0.0% | 0.003969 | 0.003266 | 82.3% |  152.0 |MPI_Allreduce(sync)
||  0.0% | 0.000029 | 0.000027 | 92.5% |    1.0 |MPI_Init(sync)
||  0.0% | 0.000009 | 0.000009 | 99.6% |    1.0 |MPI_Finalize(sync)
|=================================================================
```

MPI calls traced

```
Table 2:  Load Balance with MPI Message Stats (limited entries
shown)
 Time% |      Time |   MPI | MPI Msg Bytes |    Avg MPI |Group
       |           |  Msg  |               |   Msg Size | PE=[mmm]
       |           | Count |               |            |

100.0% | 36.703700 | 602.0 | 197,839,216.0 | 328,636.57 |Total
|-------------------------------------------------------------
| 97.2% | 35.666406 |   0.0 |           0.0 |         -- |USER
||------------------------------------------------------------
||  99.1% | 36.361896 |  0.0 |           0.0 |         -- |pe.3
||  96.9% | 35.581600 |  0.0 |           0.0 |         -- |pe.7
||  95.3% | 34.961048 |  0.0 |           0.0 |         -- |pe.5
||=============================================================
==========
|  2.8% |  1.019132 | 602.0 | 197,839,216.0 | 328,636.57 |MPI
||------------------------------------------------------------
||   4.7% | 1.722054 | 602.0 | 197,839,216.0 | 328,636.57 |pe.5
||   2.4% | 0.881184 | 602.0 | 197,839,216.0 | 328,636.57 |pe.1
||   0.9% | 0.332689 | 602.0 | 197,839,216.0 | 328,636.57 |pe.3
||=============================================================
==========
|  0.0% |  0.018161 |   0.0 |           0.0 |         -- |MPI_SYNC
||------------------------------------------------------------
||   0.1% | 0.029523 |  0.0 |           0.0 |         -- |pe.6
||   0.1% | 0.019500 |  0.0 |           0.0 |         -- |pe.1
||   0.0% | 0.001622 |  0.0 |           0.0 |         -- |pe.0
|==============================================================
===========
```

Load balance with MPI message statistics

```
Table 3:  MPI Message Stats by Caller (limited entries shown)


   MPI | MPI Msg Bytes |   MPI | MsgSz | 64KiB<= |Function
   Msg |               |   Msg |  <16  |   MsgSz | Caller
Bytes% |               | Count | Count |   <1MiB |  PE=[mmm]
       |               |       |       |   Count |

100.0% | 197,839,216.0 | 602.0 | 152.0 |   450.0 |Total
|-----------------------------------------------------------
| 100.0% | 197,838,600.0 | 450.0 |   0.0 |   450.0 |MPI_Isend
||----------------------------------------------------------
||  40.0% |  79,104,600.0 | 150.0 |   0.0 |   150.0 |sendp2
3|        |               |       |       |         | sendp
4|        |               |       |       |         |  jacobi
5|        |               |       |       |         |   main
||||||-----------------------------------------------------
6|||||  40.0% |  79,104,600.0 | 150.0 |   0.0 |   150.0 |pe.0
6|||||  40.0% |  79,104,600.0 | 150.0 |   0.0 |   150.0 |pe.4
6|||||  40.0% |  79,104,600.0 | 150.0 |   0.0 |   150.0 |pe.7
||||||=====================================================
||  40.0% |  79,104,600.0 | 150.0 |   0.0 |   150.0 |sendp1
3|        |               |       |       |         | jacobi
4|        |               |       |       |         |  main
|||||------------------------------------------------------
5||||  40.0% |  79,104,600.0 | 150.0 |   0.0 |   150.0 |pe.0
5||||  40.0% |  79,104,600.0 | 150.0 |   0.0 |   150.0 |pe.4
5||||  40.0% |  79,104,600.0 | 150.0 |   0.0 |   150.0 |pe.7
|||||======================================================
||  20.0% |  39,629,400.0 | 150.0 |   0.0 |   150.0 |sendp3
```

Load balance with MPI message statistics

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

Remarks for Tracing

- More information is given in the `pat_build` man page
- Only true function calls can be traced. Functions that are inlined by the compiler or that have local scope in a compilation unit cannot be traced
- -g tracegroup Instrument the program to trace all function references belonging to the trace function group tracegroup. Only those functions actually executed by the program at runtime are traced. A selection of tracegroup values is:
  - **blas** Basic Linear Algebra subprograms
  - **netcdf** Network Common Data Form
  - **hdf5** HDF5 I/O library
  - **heap** dynamic heap
  - **io** includes stdio and sysio groups
  - **lapack** Linear Algebra Package
  - **mpi** MPI
  - **omp** OpenMP API
  - **sysio** I/O system calls
  - **syscall** system calls
  - **hip** hip calls

  - More information on the various tracegroup values is given in `$CRAYPAT_ROOT/share/traces` after loading the `perftools-base` module

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

Event Tracing using Perftools for GPU (HIP)

```
> module load PrgEnv-cray
> module load craype-x86-trento craype-accel-amd-gfx90a rocm
> module load perftools-base
> module load perftools
```

```
> make clean; make
> pat_build -u -g hip app.exe
```

- This generates a new executable `app.exe+pat` and preserves `app.exe`.
- If object files and user libraries have already been compiled with perftools enabled, just relink the application with `rm app.exe; make` instead.
- Traces MPI functions calls, openMP offload directives and functions defined in the program source files.

```
> srun -n 8 ./app.exe+pat
> pat_report -T -o myrep.trace.rpt app.exe+pat+*/
```

- Running the `app.exe+pat` creates a `app.exe+pat+*/` directory.
- `pat_report` reads that directory and prints human-readable performance data.

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

Output: Event Tracing with Perftools for GPU (HIP)

General job information

```
CrayPat/X:  Version 22.06.0 Revision 4b5ab6256  05/21/22 02:03:49
Number of PEs (MPI ranks):    1
Numbers of PEs per Node:      1
Numbers of Threads per PE:    2
Number of Cores per Socket:  64
Execution start time:  Sun Feb 12 10:34:24 2024
System name and speed:  nid007295  2.010 GHz (nominal)
AMD   Trento              CPU  Family: 25  Model: 48  Stepping:  1
```

```
Table 1:  Profile by Function Group and Function
  Time% |     Time | Imb. |  Imb. |   Calls | Group
        |          | Time | Time% |         |  Function
        |          |      |       |         |   Thread=HIDE
 100.0% | 1.055954 |   -- |    -- | 1,255.0 | Total
|-----------------------------------------------------------------------
|  99.9% | 1.054614 |   -- |    -- |   808.0 | HIP
||----------------------------------------------------------------------
||  88.1% | 0.930584 |   -- |    -- |   110.0 | hipDeviceSynchronize
||  11.7% | 0.123213 |   -- |    -- |   644.0 | hipMalloc
||   0.1% | 0.000684 |   -- |    -- |    18.0 | hipLaunchKernel
||   0.0% | 0.000057 |   -- |    -- |    10.0 | hipMemcpy
||   0.0% | 0.000039 |   -- |    -- |    10.0 | hipMemsetAsync
||   0.0% | 0.000012 |   -- |    -- |     4.0 | hipFree
||   0.0% | 0.000011 |   -- |    -- |     1.0 | hipKernel.count_kernel
||   0.0% | 0.000008 |   -- |    -- |     3.0 | hipGetDeviceProperties
||   0.0% | 0.000004 |   -- |    -- |     4.0 | __hipPushCallConfiguration
||   0.0% | 0.000003 |   -- |    -- |     4.0 | __hipPopCallConfiguration
||======================================================================
```

Hip routines

```
|   0.1% | 0.001300 |   -- |    -- |   446.0 | ETC
||----------------------------------------------------------------------
||   0.1% | 0.001225 |   -- |    -- |     3.0 | __hip_module_dtor
||   0.0% | 0.000044 |   -- |    -- |    41.0 | hiprtcCreateProgram
||   0.0% | 0.000025 |   -- |    -- |   400.0 | call_init.part.0
||   0.0% | 0.000004 |   -- |    -- |     1.0 | __hip_module_ctor
||   0.0% | 0.000002 |   -- |    -- |     1.0 | __hip_register_globals
||======================================================================
|   0.0% | 0.000040 |   -- |    -- |     1.0 | USER
||----------------------------------------------------------------------
||   0.0% | 0.000040 |   -- |    -- |     1.0 | main
|=======================================================================
|=======================================================================
```

```
Table 2:  Profile of maximum function times
  Time% |     Time | Imb. |  Imb. | Function
        |          | Time | Time% |  Thread=HIDE
|-----------------------------------------------------------------
| 100.0% | 0.930584 |   -- |    -- | hipDeviceSynchronize
|  13.2% | 0.123213 |   -- |    -- | hipMalloc
|   0.1% | 0.001225 |   -- |    -- | __hip_module_dtor
|   0.1% | 0.000684 |   -- |    -- | hipLaunchKernel
|   0.0% | 0.000057 |   -- |    -- | hipMemcpy
|   0.0% | 0.000044 |   -- |    -- | hiprtcCreateProgram
|   0.0% | 0.000040 |   -- |    -- | main
|   0.0% | 0.000039 |   -- |    -- | hipMemsetAsync
|   0.0% | 0.000025 |   -- |    -- | call_init.part.0
|   0.0% | 0.000012 |   -- |    -- | hipFree
|   0.0% | 0.000011 |   -- |    -- | hipKernel.count_kernel
|   0.0% | 0.000008 |   -- |    -- | hipGetDeviceProperties
|   0.0% | 0.000004 |   -- |    -- | __hip_module_ctor
|   0.0% | 0.000004 |   -- |    -- | __hipPushCallConfiguration
|   0.0% | 0.000003 |   -- |    -- | __hipPopCallConfiguration
|   0.0% | 0.000002 |   -- |    -- | __hip_register_globals
|=================================================================
```

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

Event Tracing using Perftools for GPU (MPI+OpenMP offload)

```
> module load PrgEnv-cray
> module load craype-x86-trento craype-accel-amd-gfx90a rocm
> module load perftools-base
> module load perftools
```

```
> make clean; make
> pat_build -u -g mpi,omp app.exe
```

- This generates a new executable `app.exe+pat` and preserves `app.exe`.
- If object files and user libraries have already been compiled with perftools enabled, just relink the application with `rm app.exe; make` instead.
- Traces MPI functions calls, openMP offload directives and functions defined in the program source files.

```
> srun -n 8 ./app.exe+pat
> pat_report -T -o myrep.trace.rpt app.exe+pat+*/
```

- Running the `app.exe+pat` creates a `app.exe+pat+*/` directory.
- `pat_report` reads that directory and prints human-readable performance data.

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

Output: Event Tracing with Perftools for GPU (MPI+openMP offload)

```
CrayPat/X:  Version 22.06.0 Revision 4b5ab6256  05/21/22 02:03:49
Number of PEs (MPI ranks):    8
Numbers of PEs per Node:     4  PEs on each of  2  Nodes
Numbers of Threads per PE:    1
Number of Cores per Socket:  64
Execution start time:  Sat Feb 11 10:37:06 2024
System name and speed:  nid007370  2.009 GHz (nominal)
AMD   Trento             CPU  Family: 25  Model: 48  Stepping:  1
```

General job information

```
Table 1:  Profile by Function Group and Function
  Time% |      Time |   Imb. |  Imb. |   Calls | Group
        |           |   Time | Time% |         |  Function
        |           |        |       |         |   PE=HIDE
 100.0% | 24.551284 |     -- |    -- | 3,449.0 | Total
|-----------------------------------------------------------------------------
|  90.3% | 22.160000 |     -- |    -- | 1,733.0 | OACC
||----------------------------------------------------------------------------
||  83.2% | 20.418715 | 0.373699 |  2.1% | 1,094.0 | jacobi.ACC_COPY@li.242
||   4.7% |  1.164765 | 0.021092 |  2.0% |    90.0 | jacobi.ACC_COPY@li.236
||   2.3% |  0.576451 | 0.038592 |  7.2% |   429.0 | jacobi.ACC_COPY@li.268
||   0.0% |  0.000038 | 0.000013 | 28.6% |    60.0 | jacobi.ACC_KERNEL@li.242
||   0.0% |  0.000030 | 0.000020 | 45.6% |    60.0 | jacobi.ACC_KERNEL@li.268
||============================================================================
```

openMP offload regions

```
|   6.2% |  1.515424 |     -- |    -- |   130.0 | USER
||----------------------------------------------------------------------------
||   6.1% |  1.505413 | 0.002802 |  0.2% |     1.0 | initmt
||   0.0% |  0.009833 | 0.000202 |  2.3% |     4.0 | jacobi
||   0.0% |  0.000090 | 0.000026 | 25.7% |     1.0 | main
||   0.0% |  0.000049 | 0.000018 | 30.9% |    60.0 | jacobi.ACC_REGION@li.242
||   0.0% |  0.000032 | 0.000020 | 44.9% |    60.0 | jacobi.ACC_REGION@li.268
||   0.0% |  0.000008 | 0.000001 | 15.6% |     4.0 | jacobi.ACC_DATA_REGION@li.236
||============================================================================
```

User traced routines

```
|   3.5% |  0.862093 |     -- |    -- |   983.0 | MPI
||----------------------------------------------------------------------------
||   3.4% |  0.825582 | 0.313861 | 31.5% |   180.0 | MPI_Waitall
||   0.1% |  0.028702 | 0.002903 | 10.5% |   360.0 | MPI_Isend
||   0.0% |  0.006636 | 0.016262 | 81.2% |   360.0 | MPI_Irecv
||   0.0% |  0.001061 | 0.000024 |  2.5% |    62.0 | MPI_Allreduce
||   0.0% |  0.000075 | 0.000010 | 12.9% |     1.0 | MPI_Cart_create
||   0.0% |  0.000021 | 0.000004 | 19.6% |     2.0 | MPI_Barrier
||   0.0% |  0.000007 | 0.000002 | 21.2% |     4.0 | MPI_Wtime
||   0.0% |  0.000003 | 0.000003 | 54.7% |     3.0 | MPI_Type_commit
||   0.0% |  0.000002 | 0.000002 | 54.4% |     3.0 | MPI_Type_vector
||   0.0% |  0.000002 | 0.000000 | 24.1% |     1.0 | MPI_Cart_get
||   0.0% |  0.000001 | 0.000000 | 20.9% |     3.0 | MPI_Cart_shift
||   0.0% |  0.000000 | 0.000000 | 21.7% |     1.0 | MPI_Finalize
||   0.0% |  0.000000 | 0.000000 | 20.0% |     1.0 | MPI_Init
||   0.0% |  0.000000 | 0.000000 | 14.0% |     1.0 | MPI_Comm_size
||   0.0% |  0.000000 | 0.000000 | 25.1% |     1.0 | MPI_Comm_rank
||============================================================================
|   0.0% |  0.005111 |     -- |    -- |    66.0 | MPI_SYNC
||----------------------------------------------------------------------------
||   0.0% |  0.002822 | 0.002792 | 98.9% |     2.0 | MPI_Barrier(sync)
||   0.0% |  0.002211 | 0.000429 | 19.4% |    62.0 | MPI_Allreduce(sync)
||   0.0% |  0.000066 | 0.000062 | 92.8% |     1.0 | MPI_Init(sync)
||   0.0% |  0.000012 | 0.000010 | 80.8% |     1.0 | MPI_Finalize(sync)
||============================================================================
```

MPI routines

```
|   0.0% |  0.000606 |     -- |    -- |   121.0 | HIP
||----------------------------------------------------------------------------
||   0.0% |  0.000357 | 0.000092 | 23.3% |    60.0 |
hipKernel.__omp_offloading_43b2fce4_1f016d09_jacobi_l242
||   0.0% |  0.000204 | 0.000084 | 33.1% |    60.0 |
hipKernel.__omp_offloading_43b2fce4_1f016d09_jacobi_l268
||   0.0% |  0.000044 | 0.000000 |  0.9% |     1.0 | hipGraphicsUnmapResources
||============================================================================
==
|   0.0% |  0.000041 | 0.000008 | 17.9% |   400.0 | ETC
||----------------------------------------------------------------------------
||   0.0% |  0.000041 | 0.000008 | 17.9% |   400.0 | __libc_csu_init
|=============================================================================
```

HIP kernels calls during openMP offload

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

Loop Work Estimates using Perftools

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

- Use the compiler flag `-h profile_generate` for Fortran or `-finstrument-loops` for C for your build (Cray Compiler only.). This flag turns off OpenMP and significant compiler loop restructuring optimizations except for vectorization.

```
> make clean; make
> pat_build -w app.exe
```

- This generates a new executable `app.exe+pat` and preserves `app.exe`.
- Loop profiling requires a rebuild of object files and user libraries.
- Automatically traces functions defined in the program source files despite `-w`

```
> srun -n 8 ./app.exe+pat
> pat_report -T -o myrep.trace.rpt app.exe+pat+*/
```

- Running the `app.exe+pat` creates a directory `app.exe+pat+*/` experiment directory.
- `pat_report` reads that directory and prints a lot of human-readable performance data.

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

Automatic Profiling Analysis (1/2)

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

```
> make clean; make
> pat_build app.exe
```

- The APA (`-O apa`) is the default experiment. No option needed.
- The `pat_build` generates a binary instrumented for sampling (different from the pure sampling shown before with –S option)

```
> srun -n 8 app.exe+pat
> pat_report -o myrep.txt app+pat+*/
```

- Running the "`+pat`" binary creates an experiment directory.
- Applying `pat_report` to the `app+pat+*/` directory generates an `*.apa` file therein.

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

Automatic Profiling Analysis (2/2)

```
> vi app.exe+pat+*/*.apa
```

- The `*.apa` file contains instructions for the next step, i.e. tracing. Modify it according to your needs.

```
> pat_build -O app.exe+pat+*/*.apa
```

- Generates an instrumented binary `app.exe+apa` for tracing according to the instructions in the `app.exe+pat+*/*.apa` file.

```
> srun -n 8 ./app.exe+apa
> pat_report -o myrep.txt app.exe+apa+*/
```

- Running the `app.exe+apa` binary creates a new data file or directory.
- Applying `pat_report` to the `app+apa+*/` directory generates a new report.

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

Output: APA with perftools

Sampling profiling

```
CrayPat/X:  Version 6.4.1 Revision 6a6694f  06/27/16 17:24:11

Number of PEs (MPI ranks):   8
Numbers of PEs per Node:     8
Numbers of Threads per PE:   1
Number of Cores per Socket:  16

Execution start time:  Wed Mar  8 14:24:42 2017
System name and speed:  nid00036  2301 MHz (approx)
Intel haswell CPU  Family:  6  Model: 63  Stepping:  2
...

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

 100.0% | 3,685.0 |  -- |   -- |Total
|--------------------------------------------------
|  84.2% | 3,104.5 |  -- |   -- |USER
||-------------------------------------------------
||  81.9% | 3,018.1 | 57.9 |  2.2% |jacobi
||   2.3% |    86.1 |  2.9 |  3.7% |initmt
||=================================================
|  13.3% |   489.9 |  -- |   -- |ETC
||-------------------------------------------------
||  12.2% |   451.2 |  6.8 |  1.7% |_cray_scopy_HSW
||   1.0% |    38.6 |  2.4 |  6.6% |_cray_sset_HSW
||=================================================
|   2.5% |    90.6 |  -- |   -- |MPI
||-------------------------------------------------
||   1.9% |    69.6 | 46.4 | 45.7% |MPI_Waitall
|==================================================
...
 Total
------------------------------------------------------------------------
  CPU_CLK_THREAD_UNHALTED:THREAD_P                 109,408,212,031
  CPU_CLK_THREAD_UNHALTED:REF_XCLK                   3,588,344,079
  DTLB_LOAD_MISSES:MISS_CAUSES_A_WALK                   53,188,056
  PM_ENERGY:NODE                 30.382 /sec             1,112 J
  CPU_CLK                       3.05GHz
  TLB utilization            268.83 refs/miss      0.53 avg uses
  D1 cache hit,miss ratios      70.8% hits         29.2% misses
  D1 cache utilization (misses)   3.43 refs/miss    0.43 avg hits
  D2 cache hit,miss ratio       67.8% hits         32.2% misses
  D1+D2 cache hit,miss ratio    90.6% hits          9.4% misses
  D1+D2 cache utilization      10.65 refs/miss      1.33 avg hits
  D2 to D1 bandwidth      6,149.405MiB/sec  236,055,684,984 bytes
```

APA file

```
# ------------------------------------------------------------------------
# Collect the default PERFCTR group.

  -Drtenv=PAT_RT_PERFCTR=default

...

# Libraries to trace.

  -g mpi

# ------------------------------------------------------------------------

# User-defined functions to trace, sorted by % of samples.
...

  -w  # Enable tracing of user-defined functions.
      # Note: -u should NOT be specified as an additional option.

# 81.90% 3735 bytes
        -T jacobi

# 2.34% 2450 bytes
        -T initmt

# ------------------------------------------------------------------------

-o himeno.exe+apa # New instrumented program.
...
```

Suggestion to collect Performance counters

Augment list if needed, i.e. `-g mpi,io`

Create the binary for tracing

Add or remove functions as needed.

Event Tracing tuned profiling

```
CrayPat/X:  Version 6.4.1 Revision 6a6694f  06/27/16 17:24:11

Number of PEs (MPI ranks):   8
Numbers of PEs per Node:     8
Numbers of Threads per PE:   1
Number of Cores per Socket:  16
...
Table 1:  Profile by Function Group and Function

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

 100.0% | 37.748701 |       -- |    -- | 2,584.0 |Total
|--------------------------------------------------
|  95.9% | 36.184768 |       -- |    -- |     5.0 |USER
||-------------------------------------------------
||  92.5% | 34.935081 | 1.178364 |  3.7% |     2.0 |jacobi
||   3.3% |  1.248950 | 0.016487 |  1.5% |     1.0 |initmt
||=================================================
|   4.1% |  1.537996 |       -- |    -- | 2,423.0 |MPI
||-------------------------------------------------
||   3.6% |  1.344349 | 1.096798 | 51.3% |   450.0 |MPI_Waitall
...
====================================================================
 USER / jacobi
--------------------------------------------------------------------
 Time%                                        92.5%
 Time                                     34.935081 secs
 Imb. Time                                 1.178364 secs
 Imb. Time%                                    3.7%
 Calls                         0.057 /sec       2.0 calls
 CPU_CLK_THREAD_UNHALTED:THREAD_P                  104,712,083,433

 CPU_CLK_THREAD_UNHALTED:REF_XCLK                    3,465,232,627
 DTLB_LOAD_MISSES:MISS_CAUSES_A_WALK                    44,033,261
 DTLB_STORE_MISSES:MISS_CAUSES_A_WALK                   26,116,487

 L1D:REPLACEMENT                              6,955,645,277
 L2_RQSTS:ALL_DEMAND_DATA_RD                          3,675,034,522
 L2_RQSTS:DEMAND_DATA_RD_HIT                          1,408,925,969
 MEM_UOPS_RETIRED:ALL_LOADS                          22,406,881,264
 CPU_CLK                           3.02GHz
 TLB utilization              319.41 refs/miss      0.62 avg uses
 D1 cache hit,miss ratios        69.0% hits         31.0% misses
 D1 cache utilization (misses)   3.22 refs/miss      0.40 avg hits
```

Using `pat_report`

- Always need to run `pat_report` at least once to perform data conversion
  - Experiment directory contains *.xf files in the xf-files/ subdirectory
  - Perftools combines information from `xf` output and executable to produce `ap2` stored in the ap2-files/ subdirectory : instrumented binary must still exist when data is converted!
  - Resulting `ap2` files are the input for subsequent `pat_report` calls and Reveal or Apprentice[2]
  - Always use the entire experiment directory as input and not single ap2 files
  - `xf` files and instrumented binary files could be removed once `ap2` file is generated.

- Generates a text report of performance results
  - Data laid out in tables
  - Many options for sorting, slicing or dicing data in the tables.

```
> pat_report -O <table option> app.exe+pat+*/
> pat_report -O help (list of available profiles)
```

  - Volume and type of information depends upon sampling vs tracing

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

Using `pat_report`

- The performance numbers reported are in general an average over all tasks (also explains non-integer values)

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

100.0% | 20.643909 |       -- |    -- | 1149.0 |Total
|-----------------------------------------------------------
|  98.8% | 20.395989 |       -- |    -- |  219.0 |USER
||----------------------------------------------------------
||  91.1% | 18.797060 | 0.115535 |  0.7% |    2.0 |jacobi
||   7.7% |  1.597866 | 0.006647 |  0.5% |    1.0 |initmt
||   0.0% |  0.000402 | 0.000167 | 33.5% |   53.0 |sendp3
```

- Not always meaningful
  - controller/worker schemes
  - MPMD
- To solve this you can filter the *.ap2 file

```
> pat_report -sfilter_input='condition' …
```

  - The 'condition' should be an expression involving 'pe' such as 'pe<1024' or 'pe%2==0'.
  - This option is also useful when the size of the full data file makes a report incorporating data from all PEs take too long or exceed the available memory
- More help:
  - `pat_report -h` => usage
  - `pat_report -O -h` => available report tables
  - `pat_report -s -h` => options for content and format
  - `pat_report -d -h` => options for data columns

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

General Remarks

- Always check that the instrumenting binary has not affected the run time significantly compared to the original executable
- Collecting event traces on large numbers of frequently called functions, or setting the sampling interval very low can introduce a lot of overhead (check `trace-text-size` option to `pat_build`)
- Highly recommended to run on Lustre !
- The runtime analysis can be modified using environment variables of the form `PAT_RT_*`
  - Check the `PAT_LD_OBJECT_TMPDIR` variable if you cannot preserve the original build tree
- `pat_build` may recognize for instance MPI in your application and trace `MPI_Init` and `MPI_Finalize` when adding –g mpi trace all MPI calls
- Processing the `app+*/` directory from `perftools-lite` yields all the options of `pat_build` to reproduce the experiment with regular `perftools`.

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

Reveal

Performance analysis and code restructuring assistant

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

Motivation: Compiler Listings (CCE)

- Much information produced by compiler.
  - Can be listed together with source code for better clarity.
  - Use `-fsave-loopmark` for C and `–hlist=m` (or `a`) for Fortran.
  - It generates an `*.lst` file for every source code file.
  - A + sign indicates that more information can be found after the routine definition.
  - Can also be inspected in Reveal with a corresponding program library.

```
%%%    L o o p m a r k   L e g e n d    %%%

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

+ - More messages listed at end of listing
------------------------------------------
```

```
210.  + 1---------------<   for(i=0 ; i<imax ; ++i)
211.  + 1 2-------------<     for(j=0 ; j<jmax ; ++j)
212.    1 2 Vr2---------<       for(k=0 ; k<kmax ; ++k){
213.    1 2 Vr2                   a[0][i][j][k]=1.0;
214.    1 2 Vr2                   a[1][i][j][k]=1.0;
215.    1 2 Vr2                   a[2][i][j][k]=1.0;
216.    1 2 Vr2                   a[3][i][j][k]=1.0/6.0;
217.    1 2 Vr2                   b[0][i][j][k]=0.0;
218.    1 2 Vr2                   b[1][i][j][k]=0.0;
219.    1 2 Vr2                   b[2][i][j][k]=0.0;
220.    1 2 Vr2                   c[0][i][j][k]=1.0;
221.    1 2 Vr2                   c[1][i][j][k]=1.0;
222.    1 2 Vr2                   c[2][i][j][k]=1.0;
223.    1 2 Vr2 A-------<>        p[i][j][k]=(float)((i+it)*(i+it))/(float)((mx-1)*(mx-1));
224.    1 2 Vr2 A-------<>        wrk1[i][j][k]=0.0;
225.    1 2 Vr2 A-------<>        wrk2[i][j][k]=0.0;
226.    1 2 Vr2 A-------<>        bnd[i][j][k]=1.0;
227.    1 2 Vr2--------->>>     }
228.                          }
```

```
CC-6294 CC: VECTOR initmt, File = himeno.c, Line = 210
  A loop was not vectorized because a better candidate was found at line 212.

CC-6294 CC: VECTOR initmt, File = himeno.c, Line = 211
  A loop was not vectorized because a better candidate was found at line 212.

CC-6005 CC: SCALAR initmt, File = himeno.c, Line = 212
  A loop was unrolled 2 times.

CC-6204 CC: VECTOR initmt, File = himeno.c, Line = 212
  A loop was vectorized.
```

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

Input to Reveal: Program Library

- Mandatory

```
> ftn -O3 -hpl=my_program.pl -c my_program_file1.f90
> cc -O3 -fcray-program-library-path=my_program.pl -c my_program_file2.c
> reveal my_program.pl &
```

  - Recompile only sources to generate program library `my_program.pl`

- Optional but highly recommended

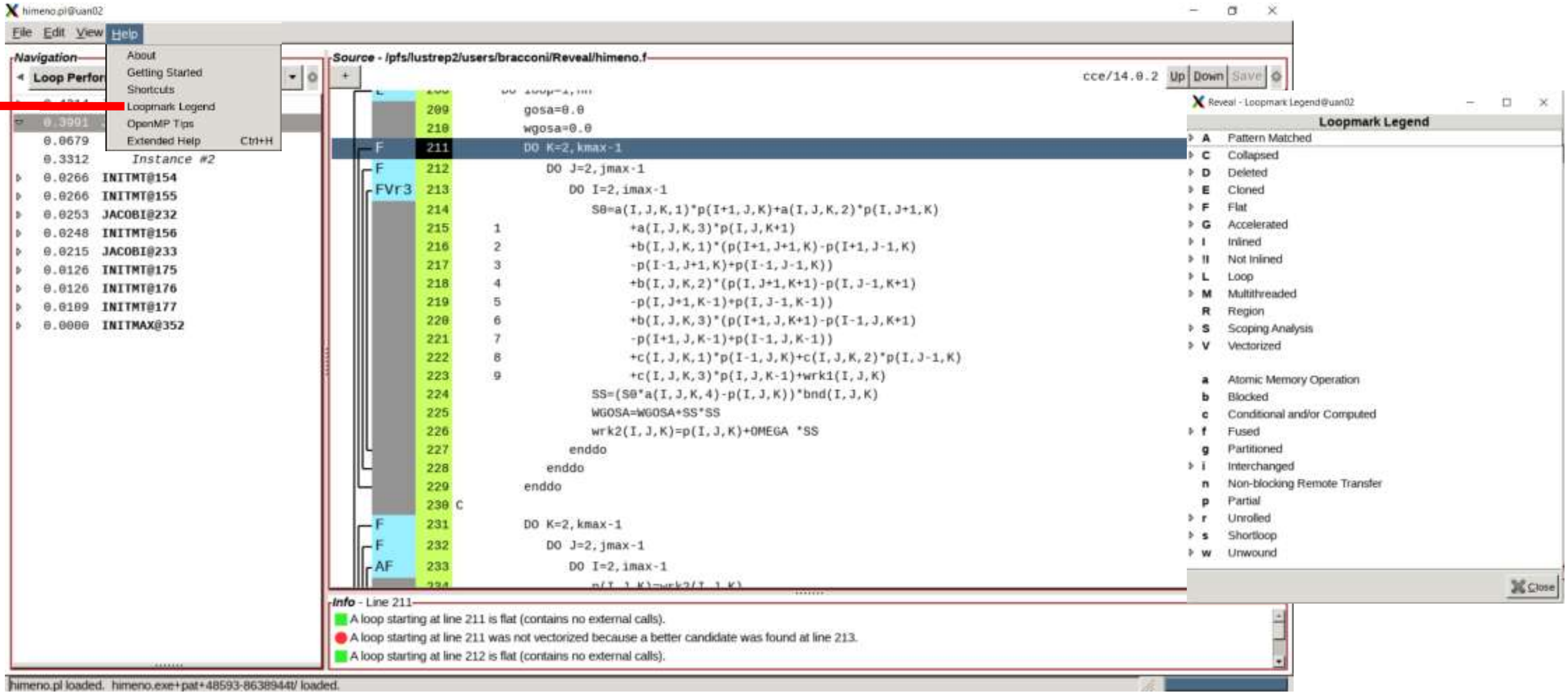
```
> ftn -O3 -hpl=my_program.pl -c my_program_file1.f90
> cc -O3 -fcray-program-library-path=my_program.pl -c my_program_file2.c
> reveal my_program.pl my_program_exe_*/ &
```

  - Collect loop work estimates in a separate experiment and load it too with `my_program.pl`
  - Note that `-hprofile_generate` option disables OpenMP and significant compiler loop restructuring optimizations except for vectorization

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

Reveal

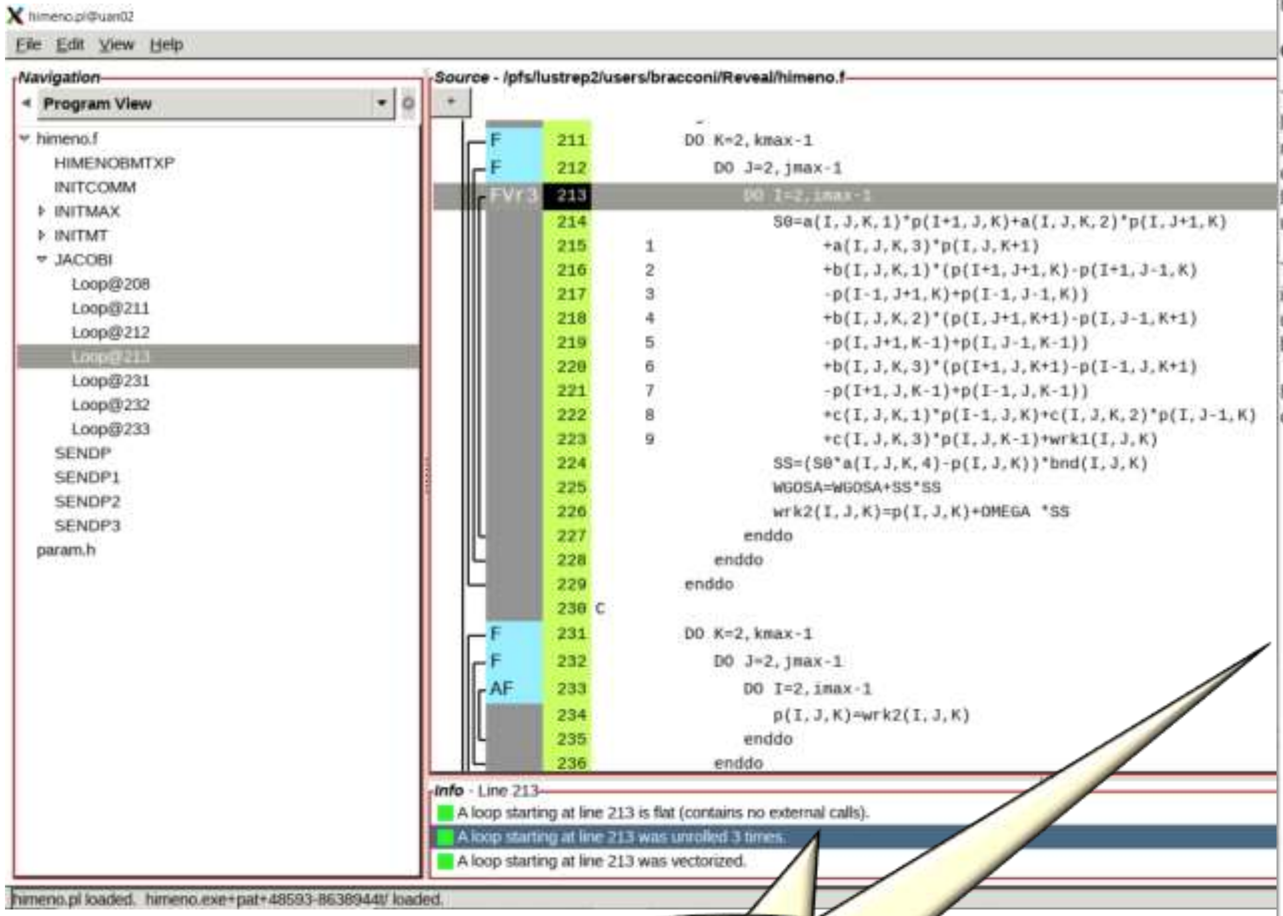
- View source, performance, and optimization information at the same time.

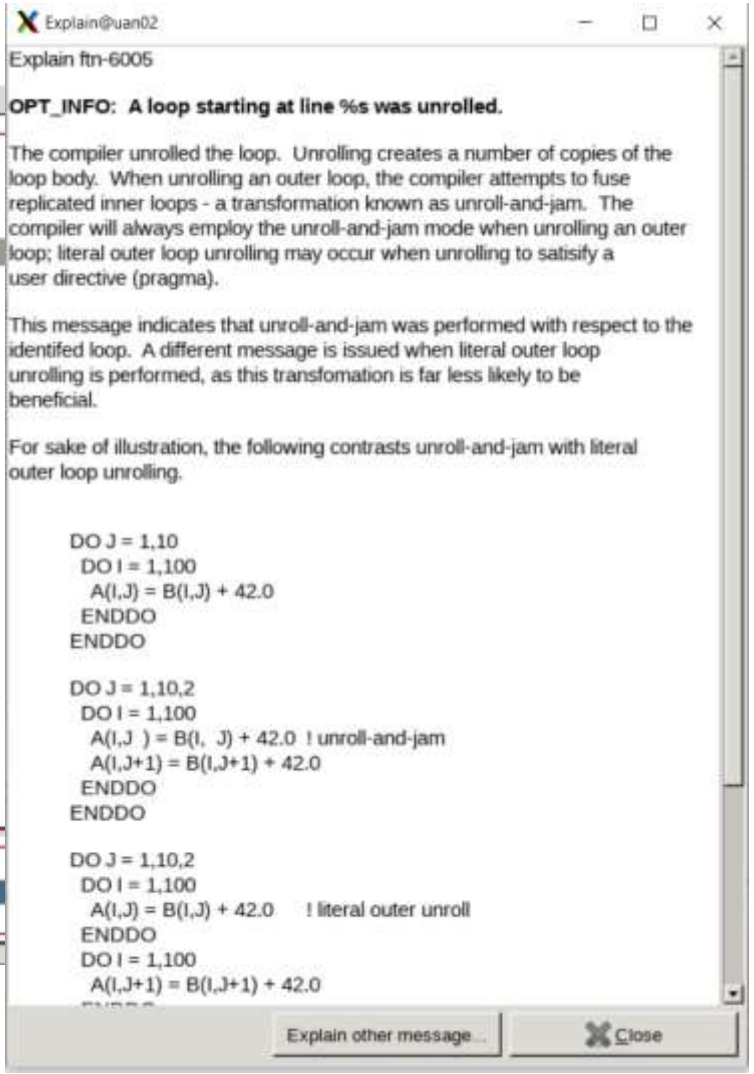


© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

Reveal

- Access Cray compiler message information

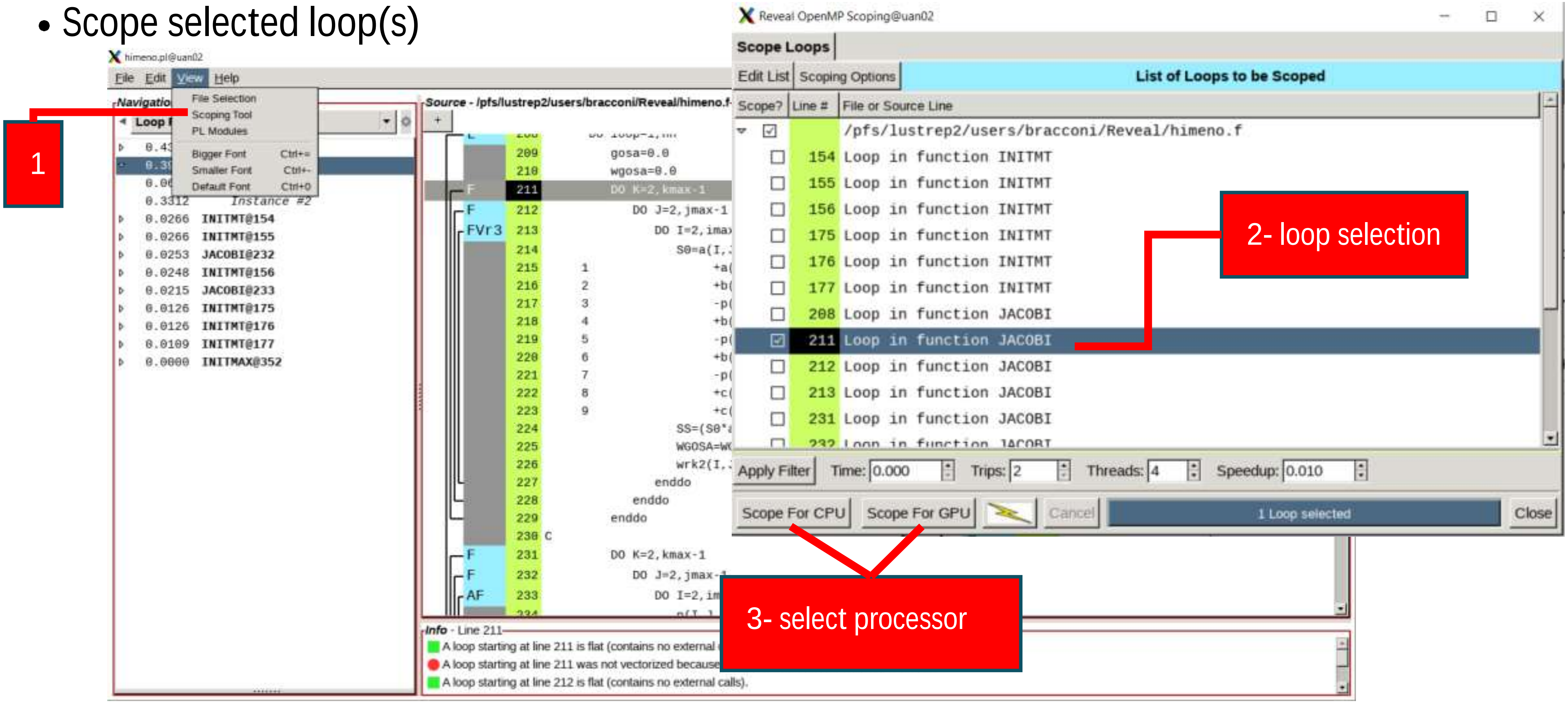




Double click on optimization message for more detailed information

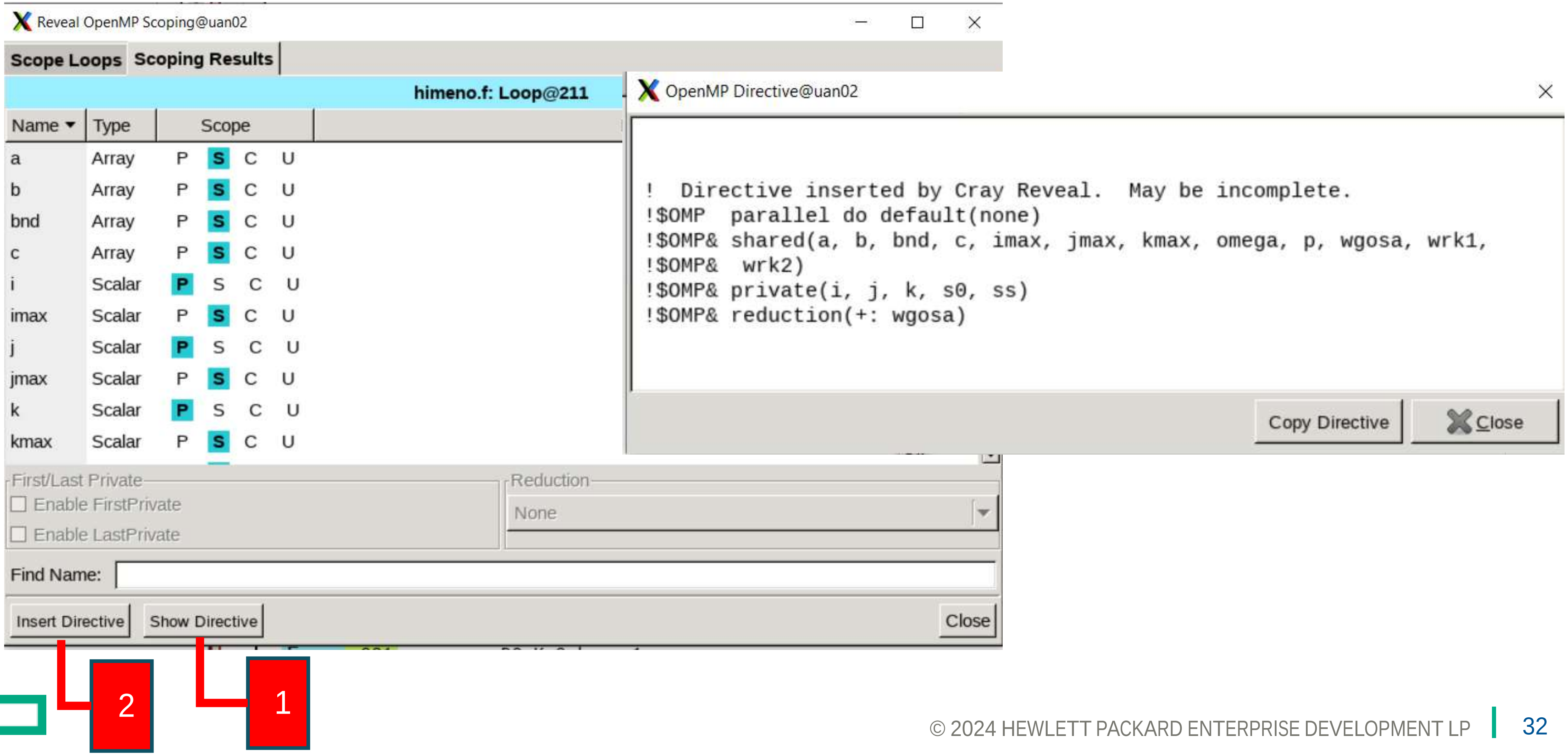
Reveal

- Scope selected loop(s)



Reveal

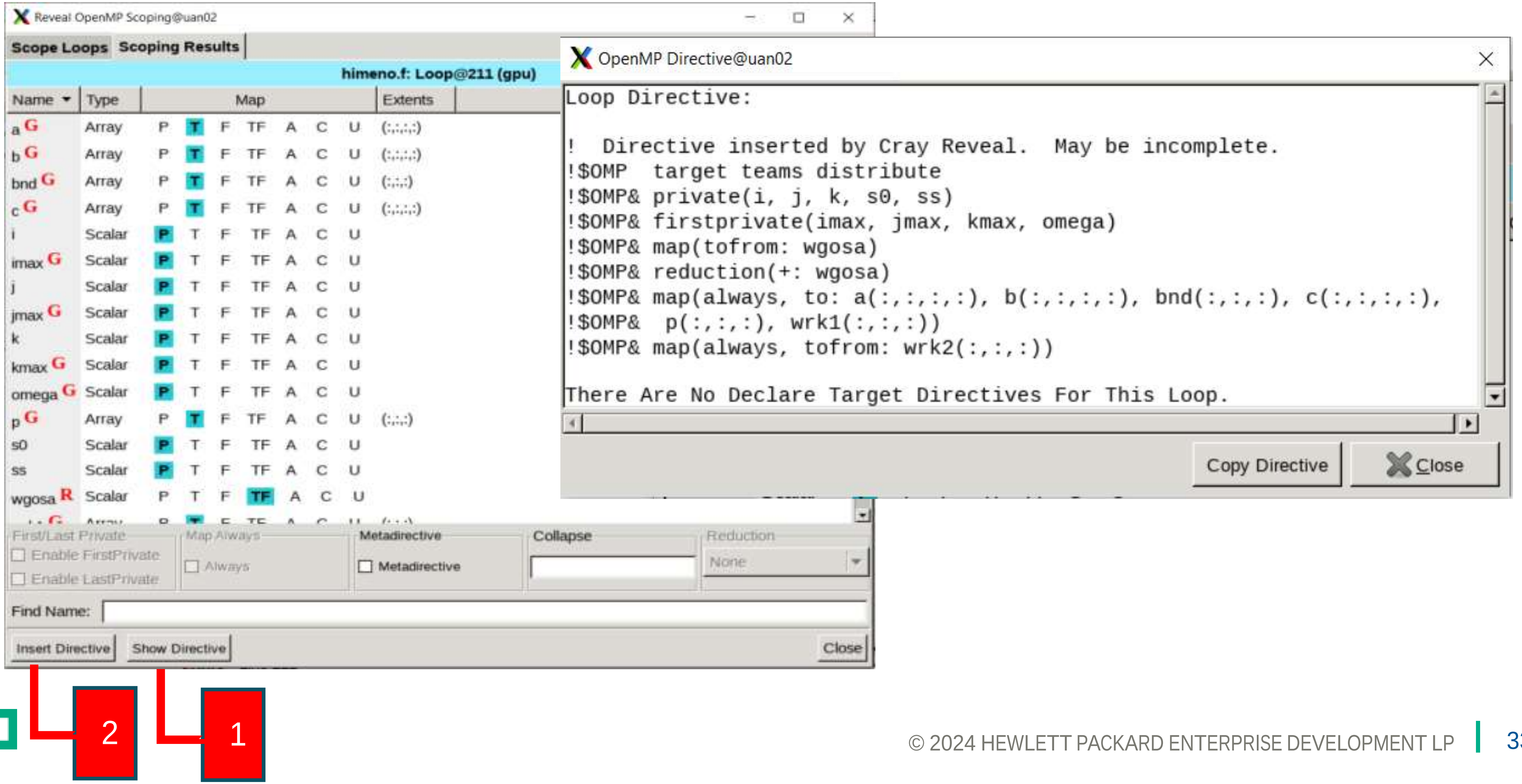
- Scope selected loop(s) for CPU



© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

Reveal

- Scope selected loop(s) for GPU



© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

Reveal

- Generate OpenMP directives.

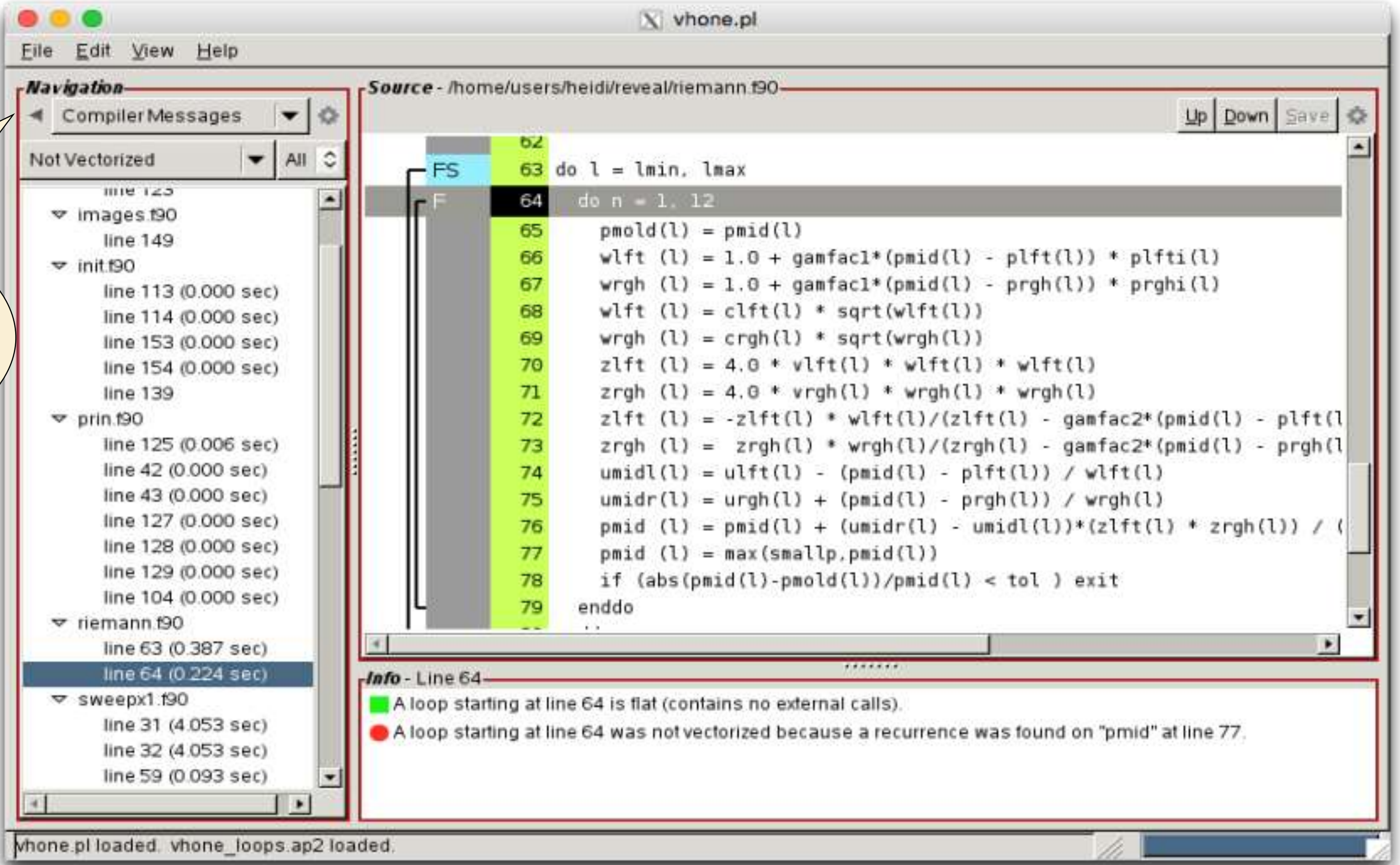
```
! Directive inserted by Cray Reveal.  May be incomplete.
!$OMP  parallel do default(none)                                      &
!$OMP&   unresolved (dvol,dx,dx0,e,f,flat,p,para,q,r,radius,svel,u,v,w,   &
!$OMP&            xa,xa0)                                                &
!$OMP&   private (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

Reveal

- Look for vectorization opportunities.

Choose "Compiler Messages" view to access message filtering, then select desired type of message



OpenMP profiling

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

OpenMP Data Collection and Reporting

- For OpenMP programs
  - Measure the overhead incurred by entering and leaving parallel regions and work-sharing constructs within parallel regions
  - Show per-thread timings in addition to other data.
  - Calculate the load balance across threads for such constructs.
- For programs that use both MPI and OpenMP
  - Profiles by default show the load balance over PEs of the average time in the threads for each PE
  - But you can also see load balances for each programming model separately.
- Options for `pat_report`
  - `profile_pe.th`
    - Imbalance based on the set of all threads in the program
  - `profile_pe_th` (default view)
    - Highlights imbalance across MPI ranks
    - Uses max for thread aggregation to avoid showing under-performers
    - Aggregated thread data merged into MPI rank data
  - `profile_th_pe`
    - For each thread, show imbalance over MPI ranks
    - Example: Load imbalance shown where thread 4 in each MPI rank didn't get much work

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

OpenMP Data Collection and Reporting

- OpenMP support needs to be enabled during compilation
- OpenMP tracing calls inserted by default when perftools is loaded

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

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

 100.0% | 2.452453 |        -- |    -- | 1426.8 |Total
|--------------------------------------------------------------------------
|  96.9% | 2.377154 |        -- |    -- |  309.8 |USER
||-------------------------------------------------------------------------
||  82.1% | 2.013394 | 0.027282 |  1.8% |  100.0 |work.LOOP@li.533
||  10.6% | 0.259470 | 0.000282 |  0.1% |    1.0 |exit
||   2.4% | 0.057711 | 0.000562 |  1.3% |    1.0 |initializeMatrix
||   1.0% | 0.024130 | 0.000313 |  1.7% |    1.0 |setPEsParams.SINGLE@li.355
||=========================================================================
|   1.6% | 0.039963 |        -- |    -- |  909.0 |MPI
||-------------------------------------------------------------------------
||   1.6% | 0.039247 | 0.079519 | 89.3% |  301.5 |MPI_Wait
||=========================================================================
|   1.2% | 0.029108 |        -- |    -- |  101.0 |OMP
||-------------------------------------------------------------------------
||   1.2% | 0.029058 | 0.012000 | 39.0% |  100.0 |work.REGION@li.492(ovhd)
|==========================================================================
```

Work sharing construct

Region

Overhead

```
Table 2:  Load Imbalance by Thread

    Max. |      Imb. |  Imb. |Thread
    Time |      Time | Time% | PE=HIDE

 2.452470 | 0.316486 | 17.2% |Total
|-----------------------------------
| 2.453287 | 0.000817 |  0.0% |thread.0
| 2.078727 | 0.036293 |  2.3% |thread.2
| 2.074969 | 0.048712 |  3.1% |thread.1
| 2.066243 | 0.043468 |  2.8% |thread.3
|===================================
```

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

Perftools API

Customized performance analysis.

API for Adding User Instrumentation

The Perftools API calls enable you to insert functions into your source code that write special tracing records into the experiment data file at runtime. Useful for large routines

- API calls are supported in both Fortran and C
- API works for sampling, tracing, and loop profiling as well as with the `perftools-lite*` modules
- When `perftools` module is loaded
  - -I`$CRAYPAT_ROOT/include` is added to compiling flag (C header `pat_api.h`, Fortran header `pat_apif.h` and Fortran 77 header files, `pat_apif.h` and `pat_apif77.h` are available)
  - Macro `CRAYPAT`and –DCRAYPAT is added to compiling flag

- `int PAT_region_begin (int id, char *label)`
  - `id` is a unique identifier for the region
  - label is the description that will appear in profiling output
- `int PAT_region_end (int id)`
  - `id` must match begin call

- Fortran subroutines with extra final integer argument for return value similar to MPI.
- Data collection through API can be controlled with `PAT_RT_TRACE_API`
- For more information, see `pat_build` man page in the section APPLICATION PROGRAM INTERFACE

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

PAT Regions Example in C and Fortran

C:

```
#ifdef CRAYPAT
#include "pat_api.h“
#endif
...
#ifdef CRAYPAT
PAT_region_begin( 1, ”jacobi_part1");
#endif
// the execution of this code segment will
// appear in CrayPAT output as “jacobi_part1”
 ...
#ifdef CRAYPAT
PAT_region_end( 1);
#endif
 ...
#ifdef CRAYPAT
PAT_region_begin( 2, ”jacobi_part2”);
#endif
// the execution of this code segment will
// appear in CrayPAT output as “jacobi_part2”
 ...
#ifdef CRAYPAT
PAT_region_end( 2);
#endif
 ...
```

Fortran:

```
#ifdef CRAYPAT
#include "pat_apif.h“
#endif
...
#ifdef CRAYPAT
Call PAT_region_begin( 1, ”jacobi_part1“, istat)
#endif
// the execution of this code segment will
// appear in CrayPAT output as “jacobi_part1”
 ...
#ifdef CRAYPAT
Call PAT_region_end( 1, istat)
#endif
 ...
#ifdef CRAYPAT
Call PAT_region_begin( 2, ”jacobi_part2”, istat)
#endif
// the execution of this code segment will
// appear in CrayPAT output as “jacobi_part2”
 ...
#ifdef CRAYPAT
Call PAT_region_end( 2, istat)
#endif
```

Event Tracing w/ and w/o Perftools API (C)

Regions

```
 Time% |      Time |     Imb. |  Imb. |    Calls |Group
       |           |     Time | Time% |          | Function
       |           |          |       |          |  PE=HIDE
100.0% | 45.429778 |       -- |    -- |  3,336.0 |Total
|-----------------------------------------------------------------------
| 82.6% | 37.513233 |      -- |    -- |    757.0 |USER
||----------------------------------------------------------------------
||  69.9% | 31.777804 | 7.855317 | 22.7% |   150.0 |#1.jacobi_part1
||   9.9% |  4.480465 | 0.484320 | 11.1% |   150.0 |#2.jacobi_part2
||   2.8% |  1.252451 | 0.066733 |  5.8% |     1.0 |initmt
||   0.0% |  0.001015 | 0.002053 | 76.5% |     1.0 |main
||   0.0% |  0.000479 | 0.000493 | 58.0% |   150.0 |sendp
||   0.0% |  0.000298 | 0.000075 | 22.9% |     2.0 |jacobi
||   0.0% |  0.000297 | 0.000026 |  9.2% |     1.0 |initcomm
||   0.0% |  0.000271 | 0.000211 | 50.0% |   150.0 |sendp3
||   0.0% |  0.000132 | 0.000255 | 75.2% |   150.0 |sendp2
||   0.0% |  0.000019 | 0.000000 |  2.8% |     1.0 |exit
||======================================================================
| 17.3% |  7.843884 |      -- |    -- |  2,423.0 |MPI
||----------------------------------------------------------------------
||  16.8% |  7.652373 | 2.858017 | 31.1% |   450.0 |MPI_Waitall
||   0.4% |  0.170341 | 0.033417 | 18.7% |   900.0 |MPI_Isend
||   0.0% |  0.016676 | 0.018946 | 60.8% |   900.0 |MPI_Irecv
||   0.0% |  0.001239 | 0.000039 |  3.4% |     1.0 |MPI_Cart_create
||   0.0% |  0.001214 | 0.000015 |  1.4% |   152.0 |MPI_Allreduce
||   0.0% |  0.000933 | 0.000022 |  2.6% |     3.0 |MPI_Type_commit
||   0.0% |  0.000644 | 0.000031 |  5.2% |     3.0 |MPI_Cart_shift
||   0.0% |  0.000459 | 0.000027 |  6.5% |     3.0 |MPI_Type_vector
||   0.0% |  0.000004 | 0.000002 | 34.5% |     2.0 |MPI_Barrier
```

With API

```
 Time% |      Time |     Imb. |  Imb. |    Calls |Group
       |           |     Time | Time% |          | Function
       |           |          |       |          |  PE=HIDE
100.0% | 45.649773 |       -- |    -- |  3,036.0 |Total
|-----------------------------------------------------------------------
| 83.4% | 38.065287 |      -- |    -- |    457.0 |USER
||----------------------------------------------------------------------
||  80.6% | 36.771025 | 7.132552 | 18.6% |     2.0 |jacobi
||   2.8% |  1.291914 | 0.077825 |  6.5% |     1.0 |initmt
||   0.0% |  0.001031 | 0.000933 | 54.3% |     1.0 |main
||   0.0% |  0.000520 | 0.000342 | 45.3% |   150.0 |sendp
||   0.0% |  0.000337 | 0.000028 |  8.7% |     1.0 |initcomm
||   0.0% |  0.000320 | 0.000264 | 51.6% |   150.0 |sendp3
||   0.0% |  0.000121 | 0.000322 | 83.1% |   150.0 |sendp2
||   0.0% |  0.000017 | 0.000001 |  3.9% |     1.0 |exit
||======================================================================
| 16.4% |  7.500291 |      -- |    -- |  2,423.0 |MPI
||----------------------------------------------------------------------
||  16.0% |  7.298155 | 4.192142 | 41.7% |   450.0 |MPI_Waitall
||   0.4% |  0.183874 | 0.048775 | 24.0% |   900.0 |MPI_Isend
||   0.0% |  0.014285 | 0.020122 | 66.8% |   900.0 |MPI_Irecv
||   0.0% |  0.001146 | 0.000013 |  1.3% |     1.0 |MPI_Cart_create
||   0.0% |  0.001003 | 0.000012 |  1.4% |   152.0 |MPI_Allreduce
||   0.0% |  0.000756 | 0.000012 |  1.7% |     3.0 |MPI_Type_commit
||   0.0% |  0.000566 | 0.000009 |  1.8% |     3.0 |MPI_Type_vector
||   0.0% |  0.000501 | 0.000015 |  3.3% |     3.0 |MPI_Cart_shift
||   0.0% |  0.000005 | 0.000002 | 34.1% |     2.0 |MPI_Barrier
||   0.0% |  0.000000 | 0.000000 | 15.4% |     4.0 |MPI_Wtime
||   0.0% |  0.000000 | 0.000000 | 18.7% |     1.0 |MPI_Cart_get
```

Without API

Load Imbalance Analysis

Load Imbalance

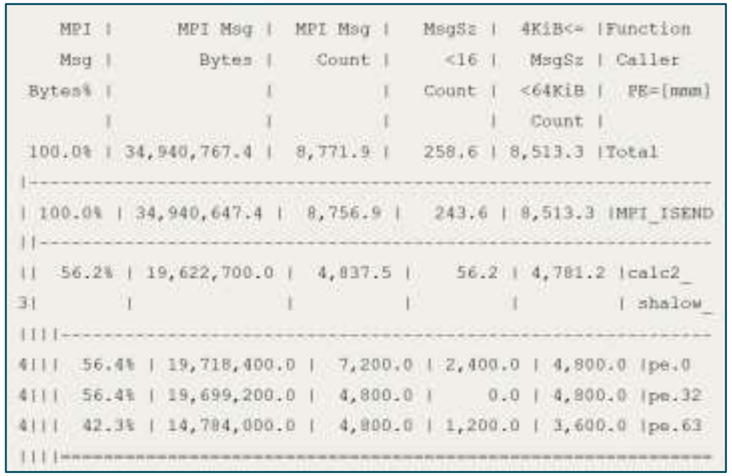
Common cause for performance bottlenecks when running applications at scale

- Look for high imbalance time and percentage
  - User functions
    Imbalance time = Maximum time – Average time
  - Synchronization (Collective communication and barriers)
    Imbalance time = Average time – Minimum time
  - Imbalance percentage of time that the rest of the team is not engaged in useful work on the given function
- Also look for MPI (SYNC) time.
  - Measure for time spent waiting in collectives
  - Only available with event tracing experiments
- Guidance on rank reordering for better load balance might appear in report
  (MPI Utilization: …)

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

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

 100.0% | 1.957703 |      -- |     -- |  42,970.8 |Total
|-----------------------------------------------------------------
|  60.0% | 1.174021 |      -- |     -- |   3,602.0 |USER
||----------------------------------------------------------------
||  30.8% | 0.603850 | 0.176924 |  23.0% |   1,198.0 |function3_
||  19.2% | 0.375117 | 0.128748 |  26.0% |   1,200.0 |function2_
||   9.1% | 0.178111 | 0.081880 |  32.0% |   1,200.0 |function1_
||================================================================
|  36.0% | 0.704928 |      -- |     -- |   9,613.0 |MPI_SYNC
||----------------------------------------------------------------
||  25.8% | 0.505174 | 0.385130 |  76.2% |   9,596.0 |mpi_barrier_(sync)
||  10.2% | 0.199537 | 0.199518 | 100.0% |       1.0 |mpi_init_(sync)
||================================================================
|   4.0% | 0.078736 |      -- |     -- |  29,754.8 |MPI
||----------------------------------------------------------------
||   2.3% | 0.045351 | 0.003531 |   7.3% |   9,596.0 |MPI_BARRIER
||   1.1% | 0.021520 | 0.051295 |  71.6% |   8,756.9 |MPI_ISEND
|=================================================================
```

Communication Bottlenecks

- Sort messages by caller
  - Available in default report for tracing (lite-events) experiments
- Analyze message sizes
  - Useful when tuning MPI env vars (eager, rendez-vous, …)
  - Found in default report
- Put barrier in front of collectives to filter sync times
- Use rank reordering for max. on-node communication
  - Look for guidance and instructions in report
  - `grid_order` utility
- Visualizing data
  - Apprentice2 (`app2 app.exe+*/`) imbalance or activity report
  - `pat_report –s pe=ALL` shows data by MPI rank



```
   MPI |       MPI Msg |  MPI Msg |   MsgSz |  4KiB<= |Function
   Msg |         Bytes |    Count |     <16 |   MsgSz | Caller
 Bytes% |              |          |   Count |  <64KiB |  PE=[mmm]
        |              |          |         |   Count |
 100.0% | 34,940,767.4 |  8,771.9 |   258.6 | 8,513.3 |Total
|-----------------------------------------------------------------------
| 100.0% | 34,940,647.4 |  8,756.9 |   243.6 | 8,513.3 |MPI_ISEND
||----------------------------------------------------------------------
||  56.2% | 19,622,700.0 |  4,837.5 |    56.2 | 4,781.2 |calc2_
3|        |              |          |         |         | shalow_
|||---------------------------------------------------------------------
4|||  56.4% | 19,718,400.0 |  7,200.0 | 2,400.0 | 4,800.0 |pe.0
4|||  56.4% | 19,699,200.0 |  4,800.0 |     0.0 | 4,800.0 |pe.32
4|||  42.3% | 14,784,000.0 |  4,800.0 | 1,200.0 | 3,600.0 |pe.63
|||=====================================================================
```

```
Total
-------------------------------------------------------
  MPI Msg Bytes%                         100.0%
  MPI Msg Bytes                 4,465,684,125.8
  MPI Msg Count                      13,057.0 msgs
  MsgSz <16 Count                       719.0 msgs
  16<= MsgSz <256 Count                  28.0 msgs
  256<= MsgSz <4KiB Count                 0.7 msgs
  4KiB<= MsgSz <64KiB Count             279.8 msgs
  64KiB<= MsgSz <1MiB Count          12,029.6 msgs
=======================================================
  MPI_Send
-------------------------------------------------------
  MPI Msg Bytes%                         100.0%
  MPI Msg Bytes                 4,465,680,353.8
  MPI Msg Count                      12,318.0 msgs
  MsgSz <16 Count                         8.0 msgs
  16<= MsgSz <256 Count                   0.0 msgs
  256<= MsgSz <4KiB Count                 0.7 msgs
  4KiB<= MsgSz <64KiB Count             279.8 msgs
  64KiB<= MsgSz <1MiB Count          12,029.6 msgs
```

```
MPI_Send / LAMMPS_NS::Comm::reverse_comm
-------------------------------------------------------
  MPI Msg Bytes%                          48.6%
  MPI Msg Bytes                 2,171,466,158.3
  MPI Msg Count                       6,006.0 msgs
  MsgSz <16 Count                         0.0 msgs
  16<= MsgSz <256 Count                   0.0 msgs
  256<= MsgSz <4KiB Count                 0.0 msgs
  4KiB<= MsgSz <64KiB Count               0.0 msgs
  64KiB<= MsgSz <1MiB Count           6,006.0 msgs
=======================================================
  MPI_Send / LAMMPS_NS::Comm::reverse_comm /
LAMMPS_NS::Verlet::run
-------------------------------------------------------
  MPI Msg Bytes%                          48.6%
  MPI Msg Bytes                 2,169,218,110.3
  MPI Msg Count                       6,000.0 msgs
  MsgSz <16 Count                         0.0 msgs
  16<= MsgSz <256 Count                   0.0 msgs
  256<= MsgSz <4KiB Count                 0.0 msgs
  4KiB<= MsgSz <64KiB Count               0.0 msgs
  64KiB<= MsgSz <1MiB Count           6,000.0 msgs
```

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

Hardware Performance Counters (HWPC)

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

Hardware Performance Counters

- Perftools supports the use of hardware counters to collect hardware events
  - All counters accessed through the PAPI interface.
  - Predefined sets of hardware counters are specified that can be instrumented for performance analysis experiment.
  - Number of simultaneous counters limited by hardware.
- Perftools provides information at the function call level on hardware features like caches, vectorization and memory bandwidth.
  - Very useful feature for deep understanding of application performance bottlenecks.
  - Impact of compiler options and code optimization.
- HWPC collection can slow down the execution notably.
  - Should be used within a tracing experiment only for a small set of functions or ideally through an automatic performance analysis.
- Only LUMI, only a limited number of hardware counters are available:
  - Use `papi_avail` on a compute node to get a list

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

When To Collect Hardware Performance Counters

- Use to understand the “why” of a bottleneck.
- Default set of CPU counters are already collected for whole program
  - Used to present memory and vector summary metrics
- To collect performance counters
  - Set `PAT_RT_PERFCTR` environment variable to list of events or group prior to execution. (Or use `-Drtenv=PAT_RT_PERFCTR=<event list> | <group>` for `pat_build`. Environment variable has priority.)

```
==================================================================
  Total
------------------------------------------------------------------
  Time%                                                 100.0%
  Time                                               70.507153 secs
  Imb. Time                                                 -- secs
  Imb. Time%                                                --
  Calls                       5.744 /sec                405.0 calls
  PAPI_BR_TKN                0.064G/sec    4,506,524,243.125 branch
  PAPI_TOT_INS              12.154G/sec  856,922,640,377.375 instr
  PAPI_BR_INS                0.124G/sec    8,718,726,704.750 branch
  PAPI_TOT_CYC                           244,793,361,310.500 cycles
  Instr per cycle                                       3.50 inst/cycle
  MIPS                    97,229.58M/sec
  Average Time per Call                             0.174092 secs
  CrayPat Overhead : Time  0.0%
==================================================================
```

- Run the following utility on a compute node to get list of events for a processor:
  - `papi_native_avail`
  - `papi_avail`
- Use pat_help to see counter groups and derived metrics
  - `$> pat_help counters processor_type deriv`
  - Example for LUMI-C: `$> pat_help counters trento deriv`

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

pat_run

Launch a dynamically-linked program instrumented for performance analysis

Profile Existing Dynamically Linked Binaries

```
$> srun -n 16 pat_run ./app.exe
$> pat_report app.exe+*/ > my_report
```

- Insert `pat_run` before executable. No instrumentation needed
- Useful if source code is not accessible but program is dynamically linked

```
$> export PAT_RT_PERFCTR=1
$> srun -n 8 pat_run ./app.exe
```

- Use existing perftools capability
- Optionally collect a different group of performance counters

```
$> pat_report -P -O callers+src app.exe+*/ > my_callers_report
```

- Create additional views of the data with `pat_report` options
- If at least object files and libraries are available
  - load the `perftools-preload` module before relinking
  - more information like line number hot spots are given

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

Final notes and Recap on Perftools

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

Perftools-lite modules vs Perftools module

- Perftools-lite modules : "novice" approach
  - Little/no knowledge of the application and do not want to dive deeper into the source code or profiling tools
  - Load the –lite module, make, run, analysis of automatically generated report
    – Sampling perftools-lite , tracing perftools-lite-events , loop work estimate perftools-lite-loops, GPU perftools-lite-gpu
  - Everything is automatic
    – Compilation and run commands are the same : instrumented binary has the same name as the original one
    – Report automatically generated
- Perftools module : "advanced" approach
  - Good knowledge of the application and the used libraries (MPI, openMP, gpu, io and more)
  - module load perftools, make, pat_build opt, run, pat_report, analysis of generated report
    – Sampling opt= ,-S; tracing opt=-u, loop work estimate opt=-w, GPU opt=-g omp/cuda/hip
    – AND MUCH MORE opt=-g mpi,io,blas,lapack,netcdf…
  - Everything is generated by the user
    – The original binary must be instrumented with pat_build and the options corresponding to the analysis to be performed
    – Run command must use the name of the new instrumented binary generated by the pat_build command
    – Report must be generated with pat_report

Summary

<table>
<tr><th></th><th>Sampling</th><th>Event Tracing</th><th>Loop Work Estimates</th><th>APA</th><th>Step</th></tr>
<tr><td rowspan="2"><code>perftools-lite</code></td><td>Is the default experiment.</td><td>Available through <code>perftools-lite-events</code></td><td>Available through <code>perftools-lite-loops</code></td><td>N/A</td><td>1</td></tr>
<tr><td colspan="4">Rebuild for all except <code>pat_run</code>. Rerun for all.</td><td>2</td></tr>
<tr><td rowspan="7"><code>perftools</code></td><td></td><td></td><td>Rebuild with <code>-h profile_generate</code> for Fortran or <code>-finstrument-loops</code> for C. Only for Cray Compiler</td><td>Is the default experiment</td><td>0</td></tr>
<tr><td><code>pat_build …</code></td><td><code>pat_build [-w|-u] [-g]</code></td><td><code>pat_build –w …</code></td><td><code>pat_build …</code></td><td>1</td></tr>
<tr><td colspan="4">For all: Run <code>app.exe+pat</code></td><td>2</td></tr>
<tr><td colspan="4">For all: <code>pat_report app+pat+*/</code></td><td>3</td></tr>
<tr><td colspan="3" rowspan="3"></td><td><code>pat_build –O *.apa</code></td><td>4</td></tr>
<tr><td>Run <code>app.exe+apa</code></td><td>5</td></tr>
<tr><td><code>pat_report <apa>.xf</code></td><td>6</td></tr>
</table>

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

Summary

	Sampling	Event Tracing	Loop Work Estimates	APA
Reveal	N/A	N/A	Need a program library obtained with the -hpl=<app>.pl compiler option for Fortran or -fcray-program-library-path=<app.pl> for C and optionally an <app>+*/ directory from a loop profiling experiment (Cray compiler only)	N/A
Hardware performance counters	Can be enabled via -Drtenv=PAT_RT_PERFCTR=<event list> | <group> option for **pat_build** or the environment variable PAT_RT_PERFCTR. For **perftools-lite*** only the environment variable is applicable. Note that for **perftools-lite** some counters are already collected.			Is typically specified in the **<app>+*/*.apa** file. Note that during the sampling phase, some counters are already collected.
Apprentice2	Can be applied to all **<app>+*/** directories			
pat_run	When you can't compile (only dynamically linked binary or objects and libraries are available).			
pat_info	Can be applied to all **<app>+*/** directories			

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

What does perftools support?

Cray Performance Tools Components
- Reveal
  - compiler optimization presentation
  - OpenMP
  - memory activity tracking
- Apprentice2 (app2)
- CrayPat runtime
- perftools / perftools-lite experiments
  - pre-defined vs custom instrumentation
  - sampling
  - tracing with runtime summarization
  - predefined trace groups
  - full trace
- pat_build
- pat_report
- pat_run
- pat_region API
- grid_order
- pat_view
- pat_help
- Performance counters
  - PAPI
  - CPU, GPU, network, energy
  - default, predefined groups, individual events

Programming Models
- MPI
- OpenMP
- CUDA
- OpenACC / OpenMP 4.5
- PGAS (upc, Forttran coarrays)
- SHMEM
- OpenCL
- HIP

Distros
- CLE
- Mac
- Windows
- RH/Centos
- SLES

Languages
- Fortran
- C
- C++
- Chapel

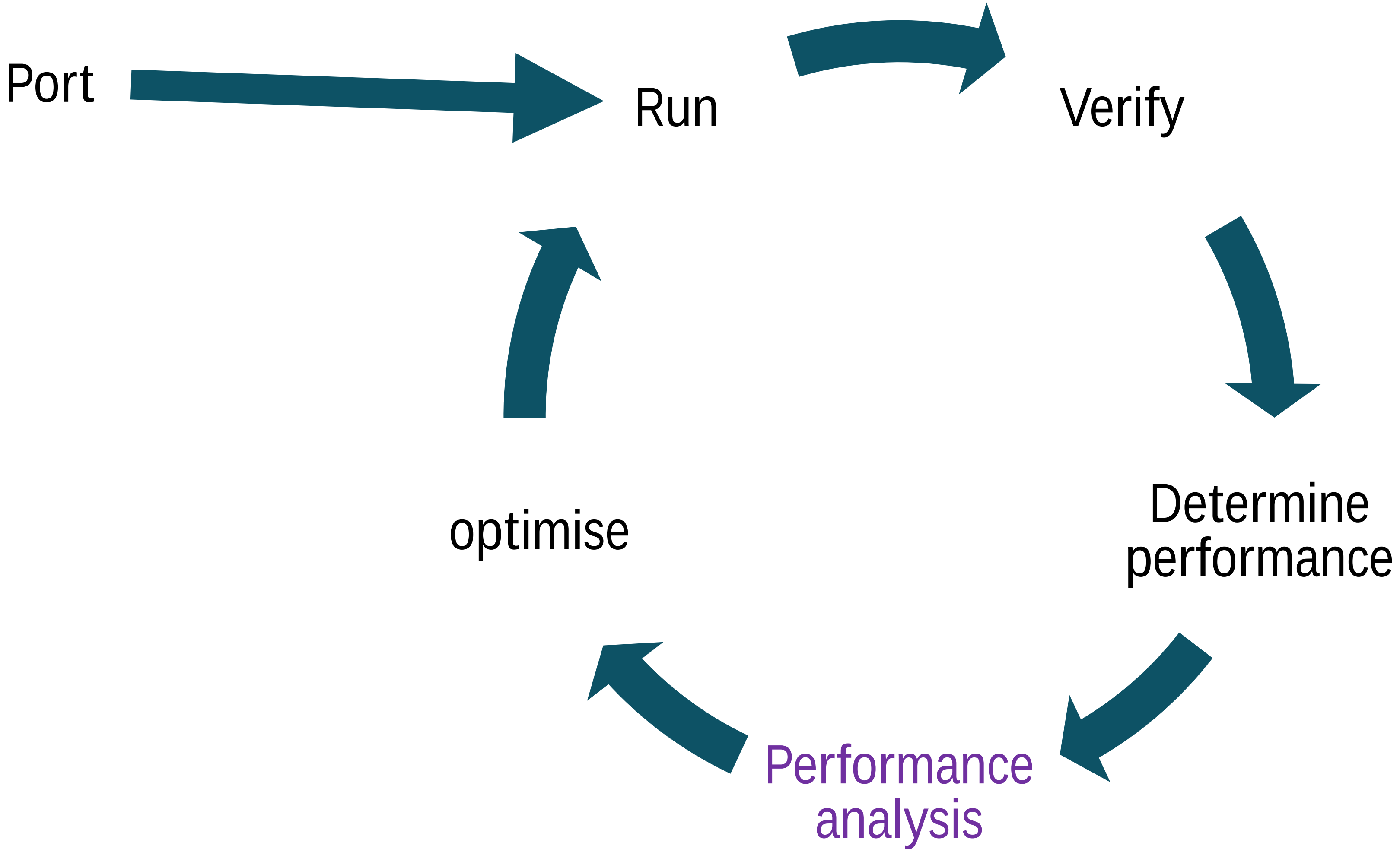
Platforms
- Cray XC, CS, Shasta
- HPE Apollo 2000 Gen10Plus, Apollo 80*
- Mac (app2, reveal), Windows (app2)

Compilers
- CCE
- GCC
- Intel
- PGI
- AMD (aocc, hipclang)

Processors
- Intel (SNB, IVB, HSW, KNL, BDW, SKL)
- ARM (Cavium TX2)
- AMD (Rome, Milan, MI60/MI100,MI200)
- NVIDIA (Pascal, Volta)

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP

Optimisation Cycle



Questions?

© 2024 HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP