CRAY
THE SUPERCOMPUTER COMPANY

Cray Performance Measurement and Analysis Tools for Cray XK Systems

Heidi Poxon
Manager & Technical Lead, Performance Tools
Cray Inc.

Outline

- Performance tools support for GPUs
- What statistics are available
  - How to collect them
  - How are they presented
- CUDA support

GPU Programming Models Supported

- Goal is to provide whole program analysis for programs written for x86 or x86 + accelerators
- Development focus for GPUs is on support of CCE with OpenACC directives
- Cray XK programming supported:
  - OpenACC, CUDA, PGI accelerator directives

GPU-specific Features in 2012

Perftools 5.3.0 (December 2011)

- Basic GPU support for codes with OpenACC directives
  - Performance statistics
  - GPU hardware performance counters

Perftools 5.3.1 (March 2012)

- Basic support for codes with CUDA
- GPU kernel statistics (grid, block, dynamic memory allocated)

GPU-specific Features in 2012

Perftools 6.0 (3Q2012)

- Complete support for calltree view that shows inclusive and exclusive time for functions, API regions, OpenMP regions, instrumented loops, and accelerated regions
- Reveal 1.0
- Update to CUPTI 4.1, PAPI 4.2.1 (update to CUPTI 5.0 if available)
- Kepler support
- Timing information for synchronous events

Post 6.0 Features

- Derived metrics for GPUs
  - Some taken from computeprof
  - Some for whole program
- GPU statistics in Cray Apprentice2
  - Show GPU time in call tree
  - GPU statistics in overview displays
  - Time line with CPU and GPU events to show overlap
  - GPU counters
  - Observations in summary display

Introduction to Performance Statistics

- Default statistics collected when accelerated directives are encountered with tracing
  - Host time for kernel launches, data copies and synchronization with the accelerator
  - Accelerator time for kernel execution and data copies
  - Data copy size to and from the accelerator
- Collection enabled by default for programs built with CCE
- Collection enabled with runtime environment variable for CUDA
- Sampling will not produce accelerator table in the report, but samples can show up in CUDA libraries

Collecting Accelerator Statistics

- Tracing produces GPU performance data:
  - `% pat_build –u my_program`
  - `% pat_build –w my_program`
- Sampling does not collect GPU performance data (although samples may be seen within CUDA libraries)
  - `% pat_build my_program`
  - `% pat_build –O apa my_program`
- The following disables compilation for OpenACC directives and therefore collection of accelerator statistics:
  - `% cce –h noacc`
  - `% cce –h profile_generate my_program.f`

Accelerator Table Column Definitions

- Host Time - wallclock time, in seconds, for the event
- Host Time% - percentage of wallclock time for events
- Acc Time - amount of time the event executed on the accelerator
- Acc Copy In - amount of data copied to the accelerator
- Acc Copy Out - amount of data copied from the accelerator
- Calls -the number of time the event occurred

All of the above are summed for regions and functions

Accelerator Table Layout

- Notes section at the beginning of the tables contains helpful information describing how the table was generated and suggestions on how to produce additional related tables.
- Data presented in default text report is organized as a calltree with functions/accelerated regions sorted in decreasing order by *Host Time*
- Called functions, regions and events are indented to the right
- Left-most column represents indentation in table
- By default, cells in accelerator tables that have no data are marked with '—'

Example Accelerator Statistics

```
Table 1:  Time and Bytes Transferred for Accelerator Regions
  Host  |    Host  |     Acc  | Acc Copy  | Acc Copy  | Calls  |Calltree
 Time%  |    Time  |    Time  |       In  |      Out  |        | PE=HIDE
        |          |          | (MBytes)  | (MBytes)  |        |
 100.0% |   2.750  |   2.015  |  2812.760 |    13.568 |    103 |Total
|-------------------------------------------------------------------------------------------------------------
| 100.0% |   2.750  |   2.015  |  2812.760 |    13.568 |    103 |lbm3d2p_d_
|        |          |          |           |           |        | lbm3d2p_d_.ACC_DATA_REGION@li.104
|||-----------------------------------------------------------------------------------------------------------
3||  63.5% |   1.747  |   1.747  |  2799.192 |        -- |      1 |lbm3d2p_d_.ACC_COPY@li.104
3||  22.1% |   0.609  |   0.088  |    12.304 |    12.304 |     36 |streaming_
||||---------------------------------------------------------------------------------------------------------
4|||  20.6% |   0.566  |   0.046  |    12.304 |    12.304 |     27 |streaming_exchange_
5|||        |          |          |           |           |        | streaming_exchange_.ACC_DATA_REGION@li.526
6|||  18.8% |   0.517  |      --  |       --  |       --  |      1 |  streaming_exchange_.ACC_DATA_REGION@li.526(exclusive)
4|||   1.6% |   0.043  |   0.042  |       --  |       --  |      9 |streaming_.ACC_DATA_REGION@li.907
5|||   1.1% |   0.031  |   0.031  |       --  |       --  |      4 | streaming_.ACC_REGION@li.909
6|||   1.1% |   0.031  |      --  |       --  |       --  |      1 |  streaming_.ACC_REGION@li.909(exclusive)
||||=========================================================================================================


...
```

Example Accelerator Statistics (2)

```
Table 1:  Time and Bytes Transferred for Accelerator Regions
  Host  |    Host  |     Acc  | Acc Copy  | Acc Copy  | Calls  |Calltree
 Time%  |    Time  |    Time  |       In  |      Out  |        | PE=HIDE
        |          |          | (MBytes)  | (MBytes)  |        |
 100.0% |   2.750  |   2.015  |  2812.760 |    13.568 |    103 |Total
...
3||   7.2% |    0.197 |    0.027 |     1.264 |     1.264 |     30 |collisionb_
4||        |          |          |           |           |        | collisionb_.ACC_DATA_REGION@li.594
|||||-------------------------------------------------------------------------------------------------------
5||||   4.0% |    0.111 |    0.006 |     1.264 |     1.264 |     21 |grad_exchange_
6||||        |          |          |           |           |        | grad_exchange_.ACC_DATA_REGION@li.440
7||||   3.7% |    0.103 |       -- |        -- |        -- |      1 |  grad_exchange_.ACC_DATA_REGION@li.440(exclusive)
5||||   2.4% |    0.065 |       -- |        -- |        -- |      1 |collisionb_.ACC_DATA_REGION@li.594(exclusive)
|||||=======================================================================================================
3||   1.6% |    0.044 |    0.044 |        -- |        -- |      1 |lbm3d2p_d_.ACC_COPY@li.167
3||   1.6% |    0.043 |    0.043 |        -- |        -- |      5 |recolor_
4||        |          |          |           |           |        | recolor_.ACC_DATA_REGION@li.822
5||   1.6% |    0.043 |    0.043 |        -- |        -- |      2 |  recolor_.ACC_REGION@li.823
6||   1.6% |    0.043 |       -- |        -- |        -- |      1 |   recolor_.ACC_REGION@li.823(exclusive)
3||   1.2% |    0.033 |       -- |        -- |        -- |      1 |lbm3d2p_d_.ACC_DATA_REGION@li.104(exclusive)
|=======================================================================================================
```

Kernel Statistics

- Statistics are averaged across all kernel launches on all PEs
- Current metrics supported
  - Kernel grid size
  - Block size
  - Amount of shared memory dynamically allocated for kernel
- Additional metrics will be available in perftools/6.0

Example Kernel Statistics

```
Table 2:  Kernel Stats for Accelerator Regions
  Avg   |  Avg   |  Avg   |   Avg   |   Avg   |   Avg   |Function
 Grid   | Grid   | Grid   | Block   | Block   | Block   |
    X   |    Y   |    Z   | X Dim   | Y Dim   | Z Dim   |
  Dim   |  Dim   |  Dim   |         |         |         |
|-------------------------------------------------------------------------------------------
| 62163 |      1 |      1 |    1024 |       1 |       1 |streaming_.ACC_KERNEL@li.909
|   402 |      1 |      1 |     128 |       1 |       1 |grad_exchange_.ACC_KERNEL@li.443
|   402 |      1 |      1 |     128 |       1 |       1 |grad_exchange_.ACC_KERNEL@li.467
|   402 |      1 |      1 |     128 |       1 |       1 |grad_exchange_.ACC_KERNEL@li.476
|   402 |      1 |      1 |     128 |       1 |       1 |grad_exchange_.ACC_KERNEL@li.500
|   400 |      1 |      1 |     512 |       1 |       1 |cal_velocity_.ACC_KERNEL@li.1126
|   400 |      1 |      1 |     512 |       1 |       1 |collisiona_.ACC_KERNEL@li.474
|   400 |      1 |      1 |     128 |       1 |       1 |collisionb_.ACC_KERNEL@li.597
|   400 |      1 |      1 |     128 |       1 |       1 |wall_boundary_.ACC_KERNEL@li.973
|   400 |      1 |      1 |     128 |       1 |       1 |collisionb_.ACC_KERNEL@li.629
|   400 |      1 |      1 |     512 |       1 |       1 |recolor_.ACC_KERNEL@li.823
|   128 |      1 |      1 |      64 |       1 |       1 |injection_.ACC_KERNEL@li.1281
|   128 |      1 |      1 |     128 |       1 |       1 |streaming_exchange_.ACC_KERNEL@li.829
|   128 |      1 |      1 |     128 |       1 |       1 |streaming_exchange_.ACC_KERNEL@li.729
|   128 |      1 |      1 |     128 |       1 |       1 |streaming_exchange_.ACC_KERNEL@li.641
|   128 |      1 |      1 |     128 |       1 |       1 |streaming_exchange_.ACC_KERNEL@li.538
|   101 |      1 |      1 |     128 |       1 |       1 |collisionb_.ACC_KERNEL@li.612
|   101 |      1 |      1 |     128 |       1 |       1 |set_boundary_micro_press_.ACC_KERNEL@li.299
|   101 |      1 |      1 |     128 |       1 |       1 |set_boundary_macro_press2_.ACC_KERNEL@li.259
|    14 |      1 |      1 |     256 |       1 |       1 |streaming_.ACC_KERNEL@li.919
|===========================================================================================
```

Accelerator Hardware Performance Counters

- Enable collection similarly to CPU counter collection:
  - CPU: PAT_RT_HWPC=*group or events*
  - GPU: PAT_RT_ACCPC=*group or events*
  - Can't mix collecting CPU and GPU counters in same run
- Enabling causes change in behavior of application:
  - Host needs to synchronize with the accelerator at each event (since accelerator executes asynchronously with the host)
  - Can be seen through accelerator table
    - No counters: time spent waiting for kernel to complete is shown with ACC_SYNC_WAIT (a synchronization created by the compiler)
    - Counters: perftools syncs with accelerator with each event so Host Time is exclusive time for the containing region (since waiting occurs within the event's trace point instead of in the compiler sync)

Accelerator HW Counter Groups

- A predefined set of groups has been created for ease of use
  - Combines events that can be counted together
- ACCPC groups start at 1000, and will be incremented by 100 as new families of accelerators are supported
- Specify group by number or name
  - PAT_RT_ACCPC=1000 *OR*
  - PAT_RT_ACCPC=inst_exec_gst
- See accpc(5) man page for list of groups and their descriptions

Examples of GPU Counter Groups

- 1002, div_branch_l1_shared_conf
  - `Divergent Branch` "Number of divergent branches within a warp."
  - `L1 Shared Bank Conflict` "Number of shared bank conflicts caused due to addresses for two or more shared memory requests fall in the same memory bank"
- 1012, warps_threads_launched
  - `Warps Launched` "Number of warps launched."
  - `Threads Launched` "Number of threads launched."

Example Performance Counter Data (Group 1000)

Table 3: ACC Performance Counter Data

global_store_transaction	inst_executed	Calltree
		PE=HIDE
2023220	80181186	Total
2023220	80181186	lbm3d2p_d_
		lbm3d2p_d_.ACC_DATA_REGION@li.104
847457	35826448	streaming_
817894	35071937	streaming_.ACC_DATA_REGION@li.907
497320	33071780	streaming_.ACC_REGION@li.909
		streaming_.ACC_KERNEL@li.909
320574	2000157	streaming_.ACC_REGION@li.919
		streaming_.ACC_KERNEL@li.919
29563	754511	streaming_exchange_
		streaming_exchange_.ACC_DATA_REGION@li.526
9854	198199	streaming_exchange_.ACC_REGION@li.641
		streaming_exchange_.ACC_KERNEL@li.641
8954	192431	streaming_exchange_.ACC_REGION@li.538
		streaming_exchange_.ACC_KERNEL@li.538
8944	176103	streaming_exchange_.ACC_REGION@li.729

…

Alternate Views of Accelerator Data

- Flat table of acc events (flat table of events sorted in decreasing order by Host Time)
  - % pat_report –O acc_fu
- Calltree by acc time (calltree with functions/regions sorted in decreasing order by `Acc Time`)
  - % pat_report –O acc_time
- Flat table of events sorted in decreasing order by Acc Time
  - % pat_report –O acc_time_fu

Performance Statistics for Programs with CUDA

- Cray PE supports CUDA that is contained within a function
- Accelerator performance and kernel level statistics available for code written using CUDA runtime API and CUDA driver API
- Implemented using Nvidia's CUPTI API which provides callbacks for CUDA functions
  - Perftools uses these callbacks to collect statistics when kernels are launched and data is copied to and from the GPU

CUDA (cont'd)

- Build program that contains CUDA:
  - Place CUDA code in separate function (compilation unit)
  - Compile functions containing CUDA with nvcc
    - Add –g option to nvcc command if you plan to use performance tools
  - Link object files using a PrgEnv-XXX programming environment
- Instrument program for data collection
  - `pat_build –u` (trace user functions, associates results with function that contains the CUDA code)
  - `pat_build –w` (trace MAIN, associates results with MAIN)
- Enable performance data collection:
  - Set runtime environment variable, `PAT_RT_ACC_STATS`, to '`all`'
- Performance statistics available in default text report

CUDA Example

- Access software

```
$ module load PrgEnv-cray craype-accel-nvidia20 perftools
```

- Build program with –g to get symbol information

```
$ nvcc -g -arch=sm_20 -c reduce0.cu  reduce6.cu
$ ftn -rm -o gpu_cuda gpu_reduce_int_cuda.F90 reduce0.o reduce6.o
```

- Enable tracing with '-u' or '-w' to instrument program

```
$ pat_build -u gpu_cuda
```

- Enable collection of statistics for CUDA, and run instrumented program

```
$ export PAT_RT_ACC_STATS=all
```

- Create report

```
$ pat_report gpu_cuda+pat+26300-0t.xf > cuda_report
```

CUDA Example (2)

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

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

 100.0% | 0.568974 |    -- |     -- |   42.0 |Total
|-----------------------------------------------------
|  99.9% | 0.568510 |    -- |     -- |   40.0 |USER
||----------------------------------------------------
||  97.6% | 0.555595 |    -- |     -- |    3.0 |reduce0_cuda_
||   1.8% | 0.009986 |    -- |     -- |    7.0 |wake_up_gpu_
||====================================================
|   0.1% | 0.000334 |    -- |     -- |    1.0 |DL
|   0.0% | 0.000131 |    -- |     -- |    1.0 |ETC
|=====================================================
```

CUDA Example (3)

```
Table 2:  Time and Bytes Transferred for Accelerator Regions

  Host   |    Host   |     Acc   | Acc Copy   | Acc Copy   | Calls  |Calltree
 Time%   |    Time   |    Time   |       In   |      Out   |        |
         |           |           | (MBytes)   | (MBytes)   |        |

 100.0% |    0.009 |    0.004 |     8.000 |     2.000 |     15 |Total
|-------------------------------------------------------------------------------
| 100.0% |    0.009 |    0.004 |     8.000 |     2.000 |     15 |reductions_
||------------------------------------------------------------------------------
||  88.3% |    0.008 |    0.004 |     6.000 |     2.000 |      6 |wake_up_gpu_
||   6.3% |    0.001 |    0.000 |     1.000 |     0.000 |      5 |reduce0_cuda_
3|   5.9% |    0.001 |    0.000 |     1.000 |     0.000 |      2 | reduce0_cuda_(exclusive)
||   5.4% |    0.001 |    0.000 |     1.000 |     0.000 |      4 |reduce6_cuda_
3|   5.2% |    0.000 |    0.000 |     1.000 |     0.000 |      2 | reduce6_cuda_(exclusive)
|===============================================================================
```

CUDA Performance Statistics Tips

- Limitations
  - Labels for events don't include line numbers or type of event (data copy, data region or kernel, etc.) like are available with CCE
  - Function association
- Multiple kernels within a single function wrapper will have statistics reporting back to wrapper
  - Create multiple functions for CUDA to separate results
- Statistics also available with pat_region API
  - Use of pat_region API associates statistics with a region (can separate statistics for different kernels)

PGI Accelerator Directive Support

- Use same method as with CUDA code
  - Instrument for tracing
  - Set PAT_RT_ACC_STATS to ‘all’

  OR

  - Use pat_region API around accelerated loop or region
- Performance statistics will show up under the first containing traced function, and not the directive (as with CCE)

Caveats

- Bug 782126 – ACC_SYNC_WAIT time is reported as exclusive time for the region when collecting GPU counters
  - Synchronization added by perftools - will be called out separately in a future release as is with MPI_Sync time
- Bug 782307 – perftools/5.3.0 is incompatible with papi/4.3.0 for GPUcounter collection
- GNU 4.6.X and perftools (incomplete .debug_pubnames which craypat relies on for global symbol names)
  - Alternative approach to gather global names will be released in perftools/6.0.0
  - Increases time pat_build needs to complete program instrumentation, no additional data collection overhead

CRAY
THE SUPERCOMPUTER COMPANY

Questions

??