Performance Measurement and Analysis Tools for Cray XE/XK Systems

Heidi Poxon
Cray Inc.

Topics

- GPU support in the Cray performance tools
- CUDA proxy
- MPI support for GPUs (GPU-to-GPU)

GPU Support

Programming Models Supported for the GPU

- Goal is to provide **whole program analysis** for programs written for x86 or hybrid x86 + GPUs

- Development focus is on support of CCE with OpenACC directives

- Cray XK programming models supported
  - OpenACC, CUDA, PGI acc (or OpenACC) directives

Collecting GPU Statistics for OpenACC

- **Load PrgEnv-cray module**
- **Load perftools module**

- **To enable OpenACC**
  - module load craype-accel-nvidia35

- **Instrument binary for tracing and collecting GPU statistics (must be tracing, not sampling)**
  - pat_build –u –g mpi,blas my_program

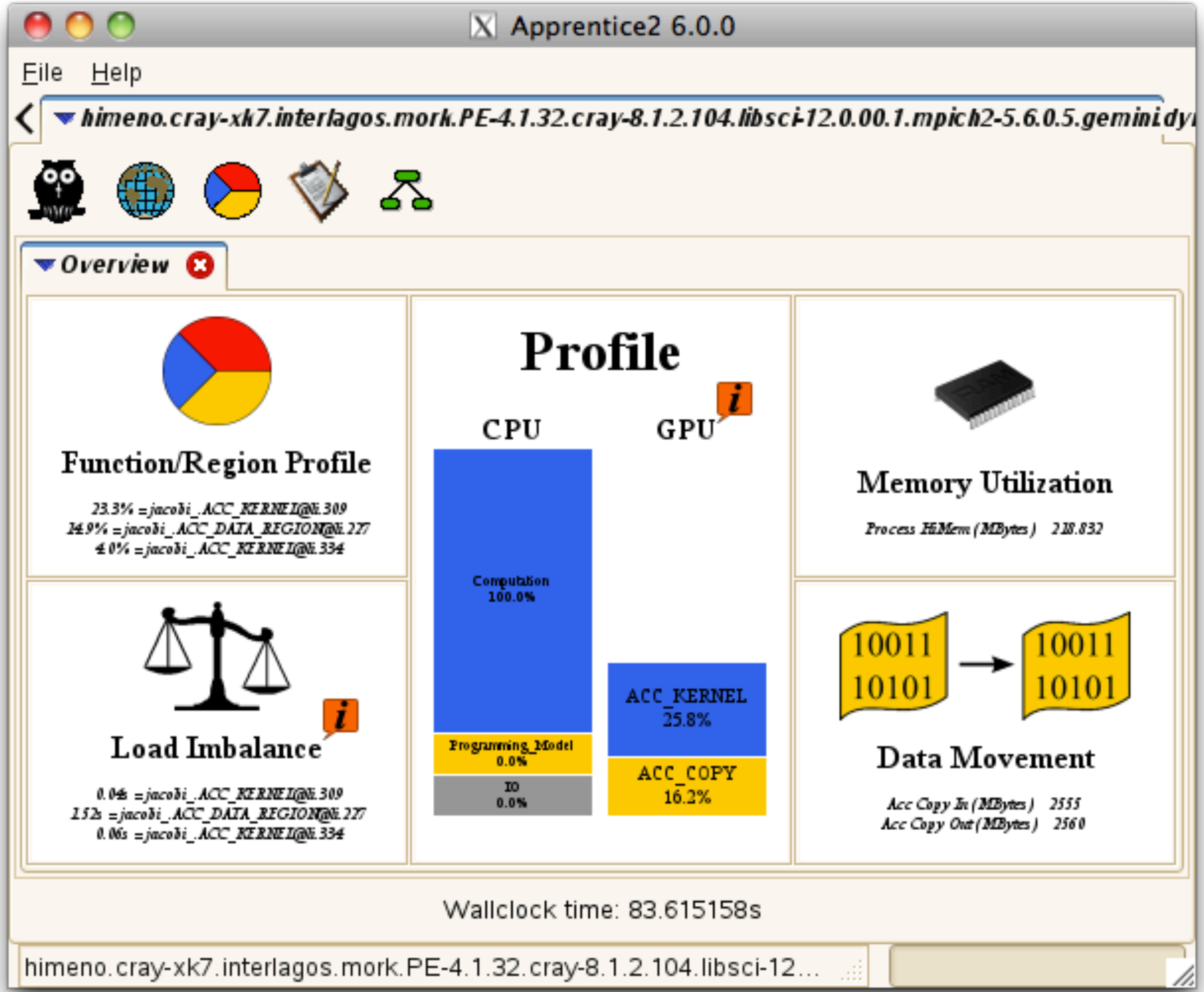
- **Run application**

- **Create report with GPU statistics**
  - pat_report my_program.xf > GPU_stats_report

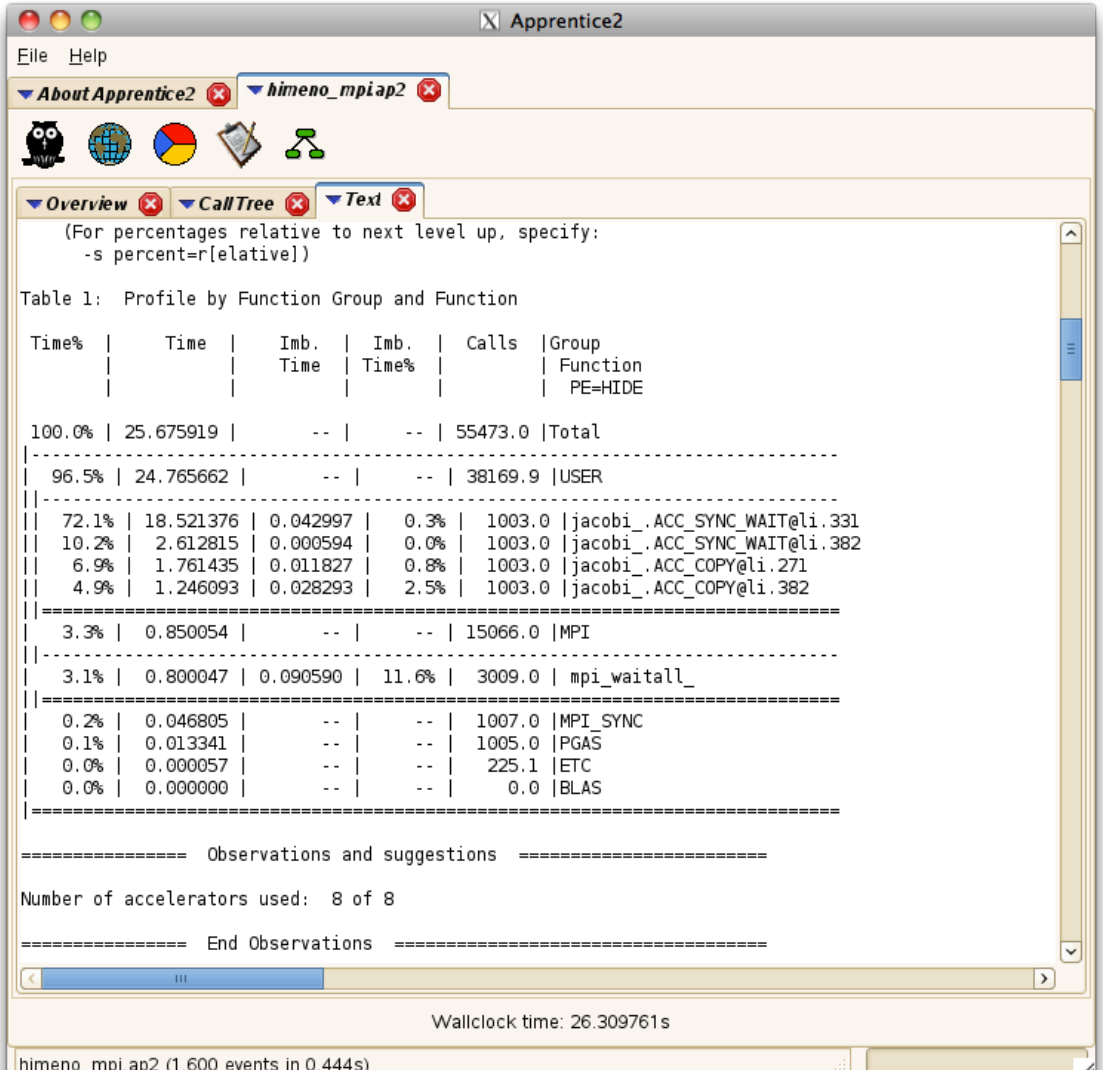
Analyze Performance of Accelerated Program

- **Statistics collected for programs with OpenACC directives**
  - Number of GPUs used in the job
  - 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
  - Kernel grid size
  - Block size
  - Amount of shared memory dynamically allocated for kernel
  - GPU performance counters
  - Derived metrics based on performance counters

Apprentice2 Overview

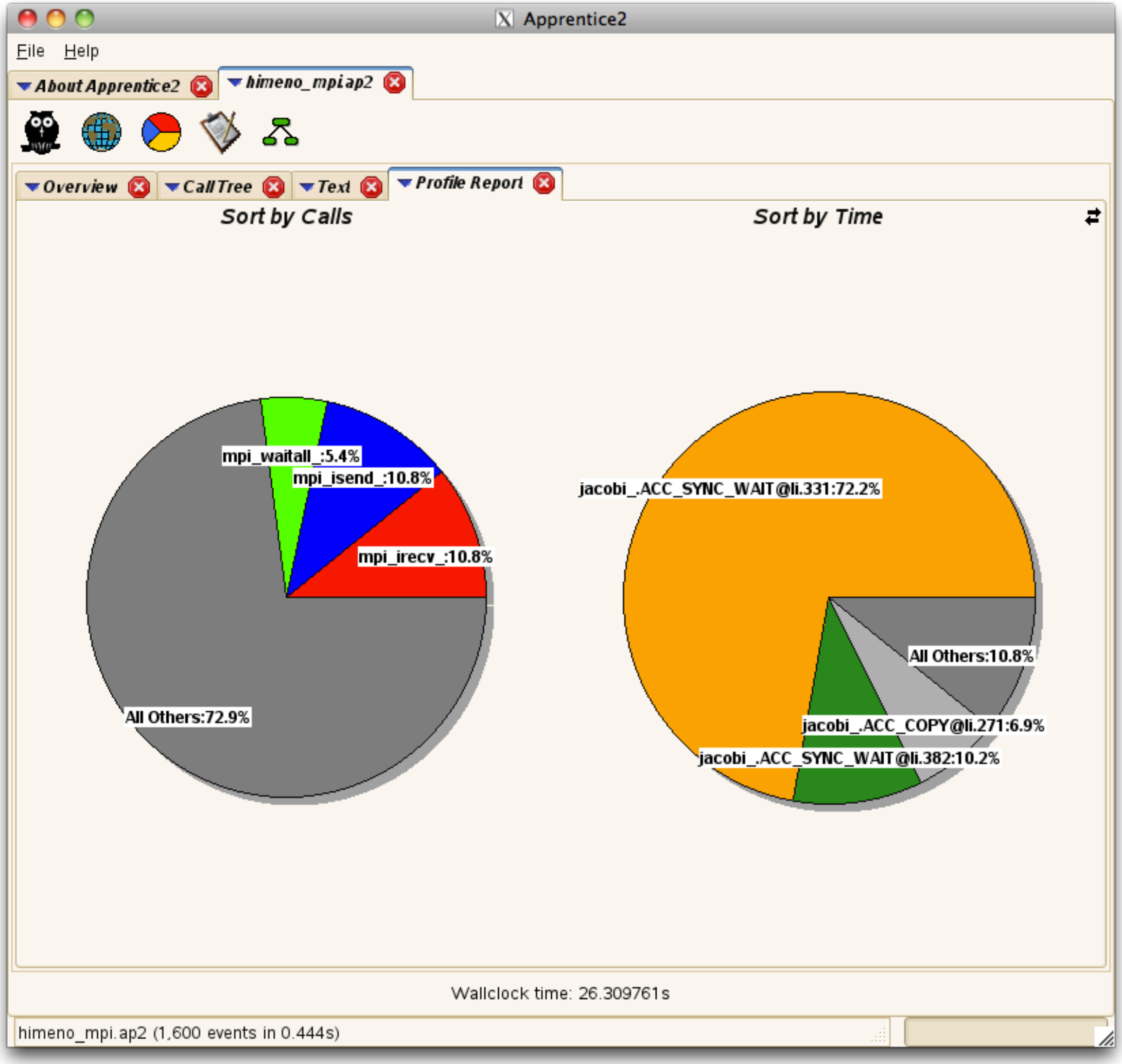


Profile with GPU Information

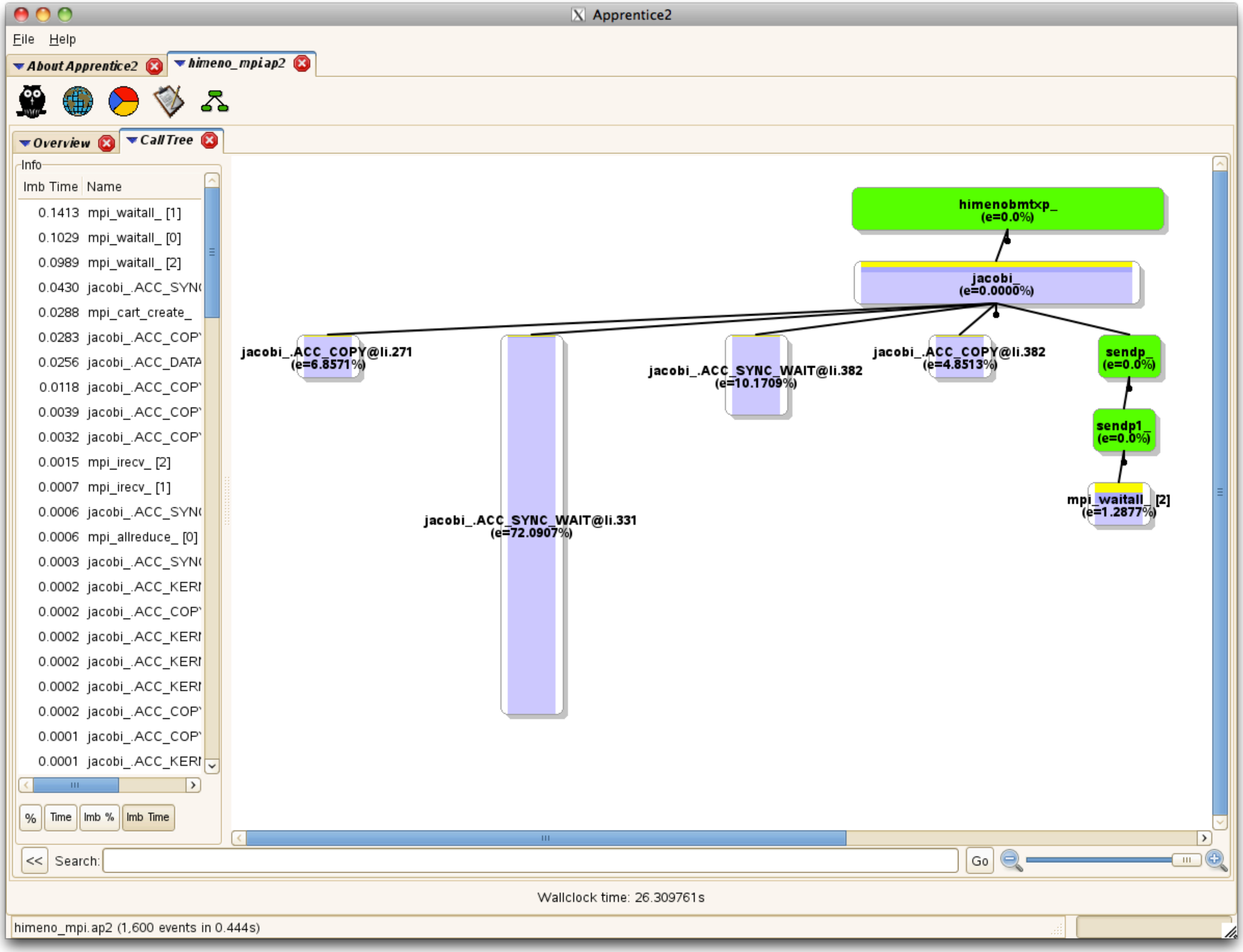



```
    (For percentages relative to next level up, specify:
      -s percent=r[elative])

Table 1:  Profile by Function Group and Function

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

 100.0% | 25.675919 |       -- |     -- | 55473.0 |Total
|-------------------------------------------------------------------------------
|  96.5% | 24.765662 |       -- |     -- | 38169.9 |USER
||------------------------------------------------------------------------------
||  72.1% | 18.521376 | 0.042997 |   0.3% |  1003.0 |jacobi_.ACC_SYNC_WAIT@li.331
||  10.2% |  2.612815 | 0.000594 |   0.0% |  1003.0 |jacobi_.ACC_SYNC_WAIT@li.382
||   6.9% |  1.761435 | 0.011827 |   0.8% |  1003.0 |jacobi_.ACC_COPY@li.271
||   4.9% |  1.246093 | 0.028293 |   2.5% |  1003.0 |jacobi_.ACC_COPY@li.382
||==============================================================================
|   3.3% |  0.850054 |       -- |     -- | 15066.0 |MPI
||------------------------------------------------------------------------------
|   3.1% |  0.800047 | 0.090590 |  11.6% |  3009.0 | mpi_waitall_
||==============================================================================
|   0.2% |  0.046805 |       -- |     -- |  1007.0 |MPI_SYNC
|   0.1% |  0.013341 |       -- |     -- |  1005.0 |PGAS
|   0.0% |  0.000057 |       -- |     -- |   225.1 |ETC
|   0.0% |  0.000000 |       -- |     -- |     0.0 |BLAS
|===============================================================================

================  Observations and suggestions  ========================

Number of accelerators used:  8 of 8

================  End Observations  ====================================
```

Top Time Consuming Routines or Regions



Call Tree with GPU regions



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 Kernel Statistics – Grid, Block

Table 2: Kernel Stats for Accelerator Regions

Avg Grid X Dim	Avg Grid Y Dim	Avg Grid Z Dim	Avg Block X Dim	Avg Block Y Dim	Avg Block Z Dim	Function
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:**
  - GPU: PAT_RT_ACCPC=*group or events*
  - CPU: PAT_RT_HWPC=*group or events*
  - NPU: PAT_RT_NWPC=*group or events*

- **Enabling GPU counters 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). Note "(exclusive)" in report.

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) and accpc_k20(5) man pages for list of groups and their descriptions**

Groups and Derived Metrics

- Groups 1000 and 1001 generate derived metrics
- Example

```
Group 1000, sm_eff_ach_occ

active_warps
"Accumulated number of active warps per cycle. For
every cycle it increments by the number of active
warps in the cycle which can be in the range 0 to
64."

active_cycles
"Number of cycles a multiprocessor has at least one
active warp."

warps_launched
"Number of warps launched."
```

Cray CUDA Proxy Support

CUDA Proxy (NVIDIA's Hyper-Q)

- Allows multiple processes to share a single GPU context (context can be thought of as a single view (shared virtual memory space) of the device
- Allows for overlap of kernels with memcpys without explicit use of streams (useful if you don't want to put kernels into streams yourself)
- Disabled by default
- The proxy server creates the shared GPU context, manages its clients (MPI ranks), and issues work to the GPU on behalf of its clients.

CUDA Proxy (cont'd)

- **How to use**
  - > setenv CRAY_CUDA_PROXY 1
  - Run with more than 1 MPI ranks per node
- **Caveats**
  - Cannot debug or profile applications when the CUDA proxy is enabled
  - NVIDIA is working on lifting this restriction
    - Need to work out how to associated GPU requests to the correct MPI rank
  - Since CUPTI doesn't support Hyper-Q, data collection by CrayPat through CUPTI is not available (GPU counters, kernel statistics)

Cray MPI GPU-to-GPU SUpport

GPU-to-GPU Optimization Feature

- Coming in February 2013
- Set MPICH_RDMA_ENABLED_CUDA=1
- Pass GPU pointer directly to MPI point-to-point or collectives

Example without GPU-to-GPU...

```
if (rank == 0) {
    // Copy from device to host, then send.
    cudaMemcpy(host_buf, device_buf, …);
    MPI_Send(host_buf, …);
} else if (rank == 1) {
    // Receive, then copy from host to device.
    MPI_Recv(host_buf,...);
    cudaMemcpy(device_buf, host_buf,...);
}
```

Example with GPU-to-GPU...

```
if (rank == 0) {
    // Send device buffer.
    MPI_Send(device_buf, …);
} else if (rank == 1) {
    // Receive device buffer.
    MPI_Recv(device_buf,...);
}
```

GPU-to-GPU Optimization Specifics

- Under the hood (i.e., in the GNI netmod), GPU-to-GPU messages are pipelined to improve performance (only applies to long message transfer aka rendezvous messages)
- The goal is to overlap communication between the GPU and the host, and the host and the NIC
- Ideally, this would hide one of the two memcpy's
- We see up to a 50% performance gain.

GPU-to-GPU optimization (Cont'd)

- On the send side (similar for recv. side)...
- Data is prefetched from the GPU using cudaMemcpyAsync.
- Data that has already been transferred to the host is sent over the network (this is off-loaded to the BTE engine).
- This allows for overlap between communication and computation.

Example GPU-to-GPU overlap

Since asynchronous cudaMemcpy's are used internally, it makes sense to do something like this...

```
if (rank == 0) {
    MPI_Isend(device_buf, …, &sreq);
    while (work_to_do) [do some work]
    MPI_Wait(&sreq, MPI_STATUS_IGNORE);
} else if (rank == 1)
    MPI_Irecv(device_buf,..., &rreq);
    while (nothing_better_to_do) [do some work]
    MPI_Wait(&rreq, MPI_STATUS_IGNORE);
}
```

Questions ?