Cray® XC™ Advanced Power Management Updates

Steven J. Martin, Greg J. Koprowski, Dr. Sean J. Wallace
Cray Inc.
{*stevem,gkoprowski,swallace*}@*cray.com*

***Abstract*—This paper will highlight the power management features of the newest blades for Cray® XC50™ and updates to the Cray PMDB (Power Management Database). The paper will first highlight the power monitoring and control features of the compute blades. The paper will then highlight power management changes in the SMW 8.0.UP06 release for the PMDB. These database implementation changes enhance the PMDB performance, configuration and management. This paper targets system administrators along with researchers involved in advanced power monitoring and management, power aware computing, and energy efficiency.**

***Keywords*-Cray Advanced Power Management; Power monitoring; power capping;**

I. Introduction

In previous work and Cray publications, we have shown details of, and documentation for power monitoring and management features of Cray XC systems [1]–[5]. In this paper we add to that work with details for Cray XC50 blades, PMDB updates in the SMW 8.0UP06 release, and new examples using power monitoring and control on Cray XC50 hardware.

This paper is organized as follows. In Section II we give a blade agnostic description of Cray XC compute blade power management features. In Section III some implementation details are given for the Intel Xeon Scalable Processors blade for Cray XC50. In Section IV we dive into PMDB enhancements delivered in the SMW 8.0.UP06 release. Last in Section V example experiments are shown to illustrate some of the power management features in use on Cray XC50 hardware in the Cray Lab in Chippewa Falls, WI.

II. Cray XC50 Compute Blade Power Management Features

In this section we describe Cray XC50 compute blade power monitoring and management features. The Cray XC50 supercomputer supports compute blades featuring the newest generation of CPU and GPU processors:

- NVIDIA® Tesla® P100 PCIe GPUs
- Intel® Xeon® Scalable processors
- Cavium ThunderX2™ processors

The following power management (power and energy monitoring and control) features will be discussed in some detail in this section:

- Total node power and energy monitoring
- Aggregate CPU power and energy monitoring
- Aggregate Memory power and energy monitoring
- Publishing of collected power and energy data:
  - Out-Of-Band (OOB) via the PMDB
  - In-band via **/sys/cray/pm_counters** sysfs files
- Power capping controls
- P-state and C-state controls

A. Total Node Power and Energy Monitoring

All blades developed for the Cray XC platform support OOB collection of total node power and energy at the blade level. Data collection is enabled by default on all blade types with a default collect rate of 1Hz. Collected data feeds into the data publishing paths described in Subsection II-C below.

B. Aggregate CPU and Memory; Power and Energy Monitoring

First introduced in [1], aggregate CPU-domain and memory-domain power telemetry is supported on blades with Intel Xeon Scalable processors, and on blades supporting the Cavium ThunderX2 processors. As with the design for the Cray® XC40™ blades supporting the Intel® Xeon Phi™ processor, XC50™ blade designs have multiple power rails. Some power rails support only the CPU-domain, some only the memory-domain, and others are shared. Power for shared rails is assigned to the dominant consumer. Collection of aggregate power and energy telemetry data is enabled by default on blades that support its collection with a default collection rate of 10Hz. Collected data is published using the same paths as the node-level telemetry data, which are described next.

C. Publishing of Collected Power and Energy Data

There are two primary paths for publishing power and energy telemetry data collected OOB at the blade level. First, we will describe the OOB path used to publish all the telemetry collected at the blade level into the Cray PMDB. This path supports the "High Speed" node-level, and aggregate CPU-domain and memory-domain data described in Subsections II-A and II-B, as well as blade-level HSS power and energy, and all System Environment Data Collection (SEDC) telemetry. The default rate for publishing collected power and energy data is 1Hz. A maximum rate of 5Hz is allowed on up-to 48 blades (192 nodes) in a system. The control is at the blade granularity, and can be changed at the SMW command line using the *xtpmaction* utility. By default, the PMDB is resident on the SMW, but Cray supports the ability to move the PMDB to an "External PMDB Node" [4]. Node level energy data, as well as cabinet and system level power data in the Cray PMDB are exposed via Cray Advanced Platform Monitoring and Control (CAPMC) REST interfaces to our workload manager (WLM) vendor partners. More about CAPMC in subsection II-D.

The second way that power and energy data collected OOB on Cray XC blades is published is the **CLE:/sys/cray/PM_counters/** "sysfs" path, which has been described and used in previous works. [6]–[8] Here are some details about this path, including some actual counter data collected from some Cray XC50 compute nodes.

Table I lists all the counter "filenames" you may find on a given compute node. The table is divided into three groupings for the purposes of this description: "Metadata Counters", "Base Telemetry Counters", and "Feature Specific Counters."

The "Metadata Counters" provide consumers of the raw sysfs data the ability to insure the counters are working as expected and to detect event and conditions that are not communicated by the basic and feature-specific telemetry counters. For example, a common use case is to read all of the counters, do some computational work, reread the counters, then compute values based on those readings, such as average power or total energy used. The change in the "freshness" counter can be used to validate the counters have updated between the first and second reads. If a change in the "generation" counter is observed, the caller can detect that the power cap was changed between the first and second read, even if the power cap was changed and restored to the original value. A change in the "startup" counter indicates that the OOB controller was restarted (an extremely rare edge case), and that no comparisons of other counters can be considered valid. The "raw_scan_hz" counter informs the user of the rate at which all counters should be updating. It can be used along with an accurate elapsed time when evaluating the "freshness" counter. The "Metadata Counters" are present on all Cray XC50 compute blade types.

The "Base Telemetry Counters" provide power, energy, and power cap data. These counters are present on all Cray XC50 compute blade types. As noted in the table, a power cap value of "0" watts is the special case for "NOT CAPPED".

The "Feature Specific Counters" are populated only where the OOB hardware collection of and/or the feature is present. For example, only nodes that support an Accelerator will populate the **CLE:/sys/cray/PM_counters/accel_power_cap** sysfs file. Likewise, only select blade types support the Aggregate CPU and Memory counters.

Next we show the use of *grep -v "not" /sys/cray/pm_counters/** to do the dirty work of printing the counter filenames and contents with minimal developer overhead. Listing 1 data was collected on prototype ThunderX2 hardware. The power readings from the prototype have been edited to show "XXX." Listing 2 data is from a node with Intel Xeon Scalable Processors. The dumps show that the two blades present the same data and functionality to the user even though they are built with different processor architectures.

Table I
CLE:/SYS/CRAY/PM_COUNTERS/ (SYSFS)

Filename		Description: Metadata Counters
freshness		Monotonically increasing counter
generation		Increments on OOB power cap changes
raw_scan_hz		Update Rate in Hz
startup		Changes on OOB controller restart
version		Protocol Version Number
Filename	**Units**	**Description: Base Telemetry Counters**
energy	Joules	Total Node Energy
power	Watts	Total Node Power
power_cap	Watts	Total Node Power Cap, 0 is uncapped
Filename	**Units**	**Description: Feature Specific Counters**
accel_energy	Joules	Accelerator Energy
accel_power	Watts	Accelerator Power
accel_power_cap	Watts	Accelerator Power Cap
cpu_energy	Joules	Aggregate CPU Energy
cpu_power	Watts	Aggregate CPU Power
memory_energy	Joules	Aggregate Memory Energy
memory_power	Watts	Aggregate Memory Power

Listing 1. PM Counters: (Prototype) Cavium ThunderX2 processors

```
/sys/cray/pm_counters/cpu_energy:4921323 J
/sys/cray/pm_counters/cpu_power:XXX W
/sys/cray/pm_counters/energy:85627470 J
/sys/cray/pm_counters/freshness:305224
/sys/cray/pm_counters/generation:27
/sys/cray/pm_counters/memory_energy:13529865 J
/sys/cray/pm_counters/memory_power:XXX W
/sys/cray/pm_counters/power:XXX W
/sys/cray/pm_counters/power_cap:0 W
/sys/cray/pm_counters/raw_scan_hz:10
/sys/cray/pm_counters/startup:1524167761885522
/sys/cray/pm_counters/version:2
```

Listing 2. PM Counters: Intel Xeon Scalable processors

```
/sys/cray/pm_counters/cpu_energy:1243834 J
/sys/cray/pm_counters/cpu_power:66 W
/sys/cray/pm_counters/energy:3186024 J
/sys/cray/pm_counters/freshness:326845
/sys/cray/pm_counters/generation:12
/sys/cray/pm_counters/memory_energy:409128 J
/sys/cray/pm_counters/memory_power:4 W
/sys/cray/pm_counters/power:72 W
/sys/cray/pm_counters/power_cap:0 W
/sys/cray/pm_counters/raw_scan_hz:10
/sys/cray/pm_counters/startup:1524658189863550
/sys/cray/pm_counters/version:2
```

The user is not expected to need to access the data published into the **CLE:/sys/cray/PM_counters/** directly, although they can. The interface is published so that both Cray provided, and third party, tools, software libraries, and workload managers can access the data. Cray provided tools include Resource Utilization Reporting (RUR), CrayPat, CrayPat-lite and Cray supported Performance Application Programming Interface (PAPI). [9]–[12]

D. *Power Capping Controls*

Cray supports power capping on all Cray XC blade types, including the latest Cray XC50 blades. The base capability for all blade types is OOB "Node-Level Power Capping." Higher level power capping constructs, such as "Job-Level" and "System-Level" power capping, are built on top of this basic capability. At the lowest level, the implementation of node-level power capping is hardware dependent. The controls available to manage power caps are common and consistent for all blade types.

The suggested interface for managing power capping on all Cray XC systems is CAPMC. CAPMC is a RESTful web service with a published API. Cray provides *capmc*, a command line utility that implements CAPMC's API, along with the CAPMC service to make it easier for administrators, operators, and processes to use CAPMC. Access to CAPMC is controlled using X.509 certificates. Setup and maintenance of the SSL infrastructure is facilitated on the SMW using the *xtmake_ca* utility. Access and distribution of certificates must be tightly controlled in accordance with site security policy. On systems with a properly setup SSL infrastructure, authorized CAPMC users can typically interact with CAPMC by first setting up their environment using *module load capmc*, then using the *capmc* utility. Third-party workload manager (WLM) vendors can use CAPMC to integrate with Cray XC's advanced platform capabilities, including power capping.

The other supported method to access power capping on Cray XC systems is using the *xtpmaction* utility. This is an older access method developed to manage static system power capping. Although this interface is still supported, sites should only use it to control power capping if CAPMC is not being used for power capping.

Official Cray documentation for **CAPMC**, *capmc*, and *xtpmaction* can be found on Cray's new online documentation server https://pubs.cray.com/ [4], [5], [13].

E. *P-state and C-state Controls*

Cray supports P-state and C-state controls via native Linux capabilities under Cray Linux Environment (CLE), and more specifically under Compute Node Linux (CNL) running on compute nodes. In addition to the expected native Linux support that requires escalated privileges, Cray supports job launch at fixed P-state with the *aprun –p-state=kHz* command line for Customers running a WLM that utilizes the Cray Application Level Placement Scheduler (ALPS) software suite.

Cray also enables P-state and C-state limiting with CAPMC. This enables WLM software to dynamically manage upper and lower P-state limits and C-state limits at node- and job-level granularity. This capability provides WLM software (and other authorized actors) additional tools to manage power and energy consumption.

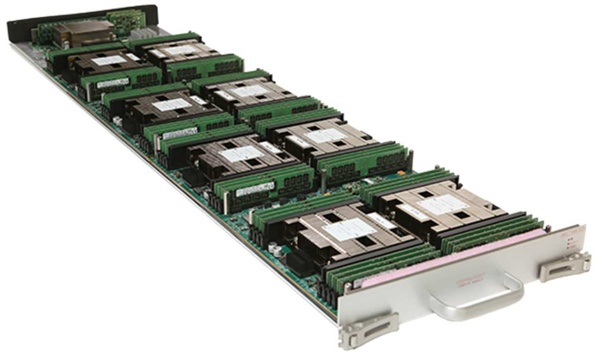
Figure 1: Cray XC50 Scalable Processors Blade

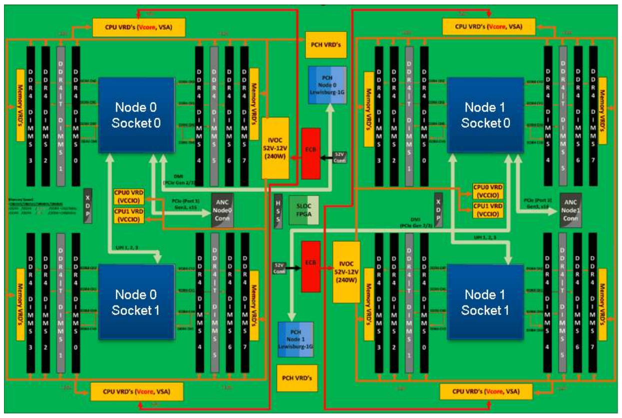

Figure 2: Cray XC50 SPDC Block Diagram

III. Cray XC50 Blade featuring Intel Xeon Scalable Processors

The Cray XC50 blade that features the Intel Xeon Scalable Processors shown in Figure 1 uses two SPDC boards and a Cray® Aries™ Network Card (ANC) to build a four-node compute blade. Each Cray XC50 cabinet can support 48 blades (192 Nodes) for a total of 384 processor sockets.

The Cray Scalable Processors Daughter Card (SPDC) is Cray's custom designed board to support the Intel Xeon Scalable Processors in the XC50 blade form factor. The SPDC block diagram is shown in Figure 2.

Each SPDC comprises two nodes, with two processor sockets each, and eight DDR4 memory slots in a 2-1-1 configuration across the six memory channels per socket. Cray supports processor SKUs with varying core counts and thermal design power (TDP) up to 165 watts.

The SPDC is the first Cray design to use a Vicor [14] Vcore [15] solution. With this design, the Vcore power rail is stepped down from the node input at 52 VDC to the socket input at point of load voltage in one stage. This solution is beneficial in several ways, but arguably the biggest benefit is the higher power density it offers with the lower board space it uses.

The following is a more complete list of benefits:

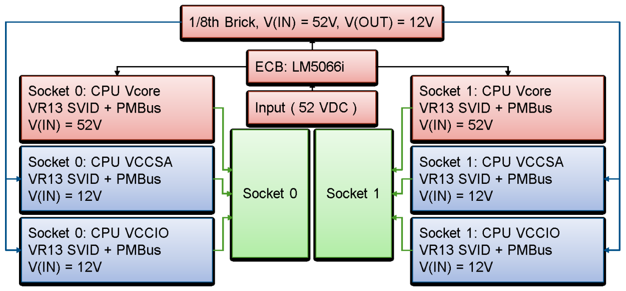
1/8th Brick, V(IN) = 52V, V(OUT) = 12V
ECB: LM5066i
Input (52 VDC)
Socket 0: CPU Vcore VR13 SVID + PMBus V(IN) = 52V
Socket 0: CPU VCCSA VR13 SVID + PMBus V(IN) = 12V
Socket 0: CPU VCCIO VR13 SVID + PMBus V(IN) = 12V
Socket 0
Socket 1
Socket 1: CPU Vcore VR13 SVID + PMBus V(IN) = 52V
Socket 1: CPU VCCSA VR13 SVID + PMBus V(IN) = 12V
Socket 1: CPU VCCIO VR13 SVID + PMBus V(IN) = 12V

Figure 3: Cray XC50 SPDC Socket Power Delivery

- Power density and circuit board real estate savings which are extremely valuable given the density of the Cray SPDC
- Improved current sense accuracy
- Support for full P_{MAX} power draw (at the maximum time duration) demands of the high-performance CPU sockets
- Reliability improvements with use of fewer power components that are extremely robust
- Power efficiency gains due to lower power distribution losses achieved by moving lower current at the node input voltage much closer to the point of load before stepping down to low voltage high current very close to the socket power pins

More SPDC socket power delivery detail is shown in Figure 3. In addition to the Vicor power solution discussed above, the figure also call out the use of the TI LM5066i [16] as the Electronic Circuit Breaker (ECB) in the SPDC design. The LM5066i is a “Hotswap Controller” with monitoring capabilities. The data sheet for the LM5066i indicates the following monitoring capabilities:

- 12-bit ADC with 1-kHz Sampling Rate
- Programmable I/V/P Averaging Interval
- Precision: V($\pm$1.25 %); I($\pm$1.75 %); P($\pm$2.5 %)

Cray blade-level firmware programs the LM5066i’s averaging interval to match the 10 Hz polling rate used for total node power and accumulated energy monitoring as describer in Section II of this paper. Blade-level firmware also monitors the Vcore, VCCSA, VCCIO to calculate aggregate CPU power and energy, as well as other voltage regulator modules to calculate aggregate memory power and energy at 10Hz. Note that all energy counters are 64-bit to enable tracking of total energy (and average power) utilization over large time intervals in joules.

Node-level power capping on Cray XC50 blade supporting Intel Xeon Scalable Processors utilizes Intel Node Manager firmware running on the Platform Controller Hub (PCH). Cray firmware communicates with the Intel firmware over an Intelligent Platform Management Bus (IPMB). The implemented power capping utilizes the Intel Running Average Power Limit (RAPL) [17].

IV. Cray Power Management Database (PMDB) Updates

The SMW 8.0.UP06 release delivers a significant update to the internal implementation for handling of time series power, energy, and System Environment Data Collection (SEDC) [4] data in the Cray PMDB. In the following subsections we will describe in detail the biggest changes with the administration of the PMDB as well as the new features available.

A. Changes to PMDB Administration

Beginning with SMW 8.0.UP06, a single new utility called *pmdb_util* exists for the purposes of configuring and, for the first time, checking the PMDB to ensure proper operation. This utility is documented in much greater detail in the *pmdb_util (8)* man page and also includes extensive online help, which can be accessed by entering *pmdb_util -h*. The following discussion focuses on the more commonly used subcommands. Please note that the output shown from the commands has been modified to better fit the presentation of this paper.

1) Checking the PMDB: To check the PMDB to make certain all of its features are working properly, use the *pmdb_util check –all* command, as shown in this example:

```
smw:~ # pmdb_util check --all
INFO: Data directory exists and matches installed version
    of PostgreSQL.
INFO: xtpgtune successfully tuned configuration. Output:
INFO:  >>> PostgreSQL configuration is already tuned.
INFO: User entry found.
INFO: pmdb database exists in PostgreSQL.
INFO: pmdbuser exists in PostgreSQL.
INFO: PMDB functions exist in PostgreSQL.
INFO: erfs schema exists in pmdb database in PostgreSQL.
INFO: sdbhwinv schema exists in pmdb database in
    PostgreSQL.
INFO: diags schema exists in pmdb database in PostgreSQL.
INFO: sm schema exists in pmdb database in PostgreSQL.
INFO: hsslocks schema exists in pmdb database in
    PostgreSQL.
INFO: pmdb schema exists in pmdb database in PostgreSQL.
INFO: ----------------------------------------------------
INFO: RESULTS:
INFO: --     pmdbuser_auth: SUCCESS --
INFO: --     pmdb_database: SUCCESS --
INFO: --         pmdb_user: SUCCESS --
INFO: --         functions: SUCCESS --
INFO: --       erfs_schema: SUCCESS --
INFO: --   sdbhwinv_schema: SUCCESS --
INFO: --      diags_schema: SUCCESS --
INFO: --         sm_schema: SUCCESS --
INFO: --   hsslocks_schema: SUCCESS --
INFO: --       pmdb_schema: SUCCESS --
INFO: PMDB passed all checks!
INFO: ----------------------------------------------------
```

By default, *pmdb_util* is set as a requirement for the PostgreSQL [18] service to start, and if any issues are found it will attempt to resolve them automatically in a non-destructive way (i.e., without removing or modifying any data in the database). If however it can not resolve an issue automatically, the PostgreSQL service will not be allowed to start:

```
smw:~ # systemctl start postgresql
A dependency job for postgresql.service failed. See '
    journalctl -xe' for details.
```

There are several ways to diagnose the problem, but perhaps the easiest is to investigate the *journalctl -u pmdb_util* output. The last start attempt will always be recorded at the bottom of that output and will include a list of problems that could not be resolved.

2) Configuring the PMDB: To start fresh and reinitialize the PMDB, run the *pmdb_util config –init* command. For example:

```
smw:~ # pmdb_util config --init
INFO: Configuring PostgreSQL data directory...
INFO: Old data directory removed.
INFO: New data directory successfully created.
INFO: xtpgtune successfully tuned configuration. Output:
INFO:  >>> Wrote backup of /var/lib/pgsql/data/postgresql.
    conf to /var/lib/pgsql/data/postgresql.conf.bak
INFO:  >>> Wrote configuration to /var/lib/pgsql/data/
    postgresql.conf
INFO: Configuring pmdbuser auth entry...
INFO: Adding pmdbuser entry to pg_hba.conf file...
INFO: User entry added.
INFO: Creating pmdbuser in PostgreSQL...
INFO: Created database user successfully.
INFO: Configuring the pmdb database in PostgreSQL...
INFO: Created the pmdb database successfully.
INFO: Installing the XTPMD schema...
INFO: XTPMD schema successfully installed.
INFO: Installing extensions...
INFO: xtpmd extensions installed.
INFO: erfs extensions installed.
INFO: Installation of dir_fdw extension and the diags
    schema succeeded.
INFO: sbdhwinv extension installed.
INFO: hsslocks extension installed.
INFO: ------------------------------------------------
INFO: PMDB Initialization SUCCEEDED
INFO: ------------------------------------------------
```

Any facet of the PMDB can be configured by using *pmdb_util config* with one or more of the specific options. For more information, enter the *pmdb_util config -h* command.

After configuration, there are additional "runtime" options that can be found in the file *pmdb_migration.conf* in the */var/adm/cray/pmdb_migration/* directory. Most are self explanatory, but there are a few that deserve special mention.

The first is *PMDB_MAX_SIZE*, which is the new method by which data retention is controlled. Rather than having to compute estimated storage requirements, the power management daemons now monitor the total size of the PMDB. If the size reaches this value or higher, data from the PMDB will be dropped in increments of the chunk time interval starting from the oldest data until the total size is less than this value.

In the previous generation of PMDB jobs data would be dropped when there was no remaining power data that it could be related to. A new setting, *PMDB_JOBS_NUM_DAYS*, changes this behavior so that jobs information is kept for the number of specified days (or forever is this is set to 0). If the system administrator sets *PMDB_JOBS_NUM_DAYS=0*, they must then take on the responsibility of pruning data from the *pmdb.job_info* and *pmdb.job_timing* tables to prevent job data from growing so large that disk space is not available for other telemetry. The default setting is to keep 30 days of job related data.

B. Integration of TimescaleDB

PostgreSQL is an object-relational database management system (ORDBMS) that is used by the PMDB to store data. It is best suited towards a high frequency update workload. Database engines of this type have poor write performance for large tables with indexes, and as these tables grow this problem worsens as table indexes no longer fit into memory. This results in every table insert performing many disk fetches swapping in and out portions of an index's B-Tree. Indexing is a necessity otherwise every query targeting any portion of the database would require a sequential scan of the entire database, a potentially very wasteful action when only a subset of the data is required.

Time-series databases are very regularly used as a solution to this problem. They are designed for fast ingest and fast analytical queries over almost always a single column (hence, key/value). This works very well for analytics on a single metric or some aggregate metric, but this leads to a lack of a rich query language or secondary index support. A lack of secondary index support means that queries targeting multiple fields are much less efficient and often result in entire tables being scanned.

One of the major functions of the PMDB is the storage of time-series power data, and one of the most common use cases is the joining of raw power data along with job info. As a result, we have been continually faced with the challenge of optimizing for the efficient ingest of potentially large collections of data as well as efficient querying of that data in multiple contexts.

Up to this point, our solution to the ingest problem was relatively simple; it's a non-trivial task to generate an index on a very large table, whereas smaller tables take much less effort to generate indexes. In this way, we set up the database to abstract the idea of a single large continuous table into smaller partitions that made index generation not only much faster, but also an action that could be performed in the background. This approach worked well to solve the scaling issues, however it still relied entirely on indexing for efficient querying.

TimescaleDB [19]–[21] is an open-source project that aims to approach this problem differently. Instead of trying to build yet another time-series database, they instead focused on extending PostgreSQL providing time-series features while retaining all of the PostgreSQL features and reliability that has been built up over the past 20 years. In a similar fashion as our previous implementation, Timescale partitions data into smaller chunks to abstract the idea of a single continuous table. Unlike our approach, however, this partitioning is done by time instead of a maximal number of rows.

This time based partitioning offers a number of potential advantages with one of the most immediately beneficial of them being aggressive query optimization. Since Timescale integrates with PostgreSQL at the extension level, it has the ability to better inform the query planner where data resides. Essentially, since Timescale knows exactly what time range each partition holds, any query that targets a specific range of time can be optimized to only scan those chunks that could possibly contain the data.

To demonstrate the potential of this approach, consider the following query which counts the number of samples per minute over a several day period for a specific sensor:

```
SELECT date_trunc('minute', ts) as time, COUNT(*)
    FROM pmdb.bc_data
    WHERE ts BETWEEN '2017-07-11 00:00:00'
                 AND '2017-07-13 00:00:00'
    AND id = 16
    GROUP BY time ORDER BY time DESC;
```

Because Timescale knows exactly where the data resides for this block of time and this sensor, it only has to do a sequential scan on 8 chunks. A system without Timescale would have to do an index scan on over 1,000 child table indexes and then sequentially scan each applicable partition to match on the id. For this query and data set, this optimization resulted in a savings of nearly 3.5 minutes or a speedup of 6.86x.

This optimization is possible with or without indexing, meaning most queries that target ranges of time can expect a 2x speedup in query execution time with certain queries seeing much more than that.

1) Determining Database Contents: With the previous partition method, it was rather simple to compute what data resided in the PMDB. With Timescale, it's still possible to figure this out, but the process is slightly less straight forward. It has always been possible to determine the total size of the database:

```
pmdb=> SELECT pg_size_pretty(pg_database_size('pmdb'));
pg_size_pretty
----------------
500 GB
(1 row)
```

It is also easy to check the size of each table:

```
pmdb=> select * from
pmdb-> hypertable_relation_size_pretty('pmdb.bc_data');
 table_size | index_size | toast_size | total_size
------------+------------+------------+------------
 234 GB     | 101 GB     |            | 335 GB
(1 row)
```

All of the metadata that Timescale uses to determine which chunks data belongs to is housed in the *_timescaledb_catalog* table schema. For example, one can use the information in these child tables to figure out exactly what range of time each chunk contains (full timestamps are abbreviated to save space):

```
smw:~> psql pmdb pmdbuser -c "
>   select hypertable.table_name as pmdb_table,
>    chunk.table_name as timescale_table,
>    to_char(to_timestamp(range_start/1000000),'HH:MI')
>      as start,
>    to_char(to_timestamp(range_end / 1000000),'HH:MI')
>      as end
>    from _timescaledb_catalog.chunk
>      inner join _timescaledb_catalog.chunk_constraint
>        on chunk.id = chunk_constraint.chunk_id
>      inner join _timescaledb_catalog.dimension_slice
>        on chunk_constraint.dimension_slice_id =
>          dimension_slice.id
>      inner join _timescaledb_catalog.hypertable
>        on chunk.hypertable_id = hypertable.id
>    order by range_start desc limit 8;
> ";
  pmdb_table   |    timescale_table     | start |  end
---------------+------------------------+-------+-------
 bc_data       | _hyper_1_16546_chunk | 05:30 | 06:00
 cc_sedc_data  | _hyper_4_16548_chunk | 05:30 | 06:00
 bc_sedc_data  | _hyper_3_16545_chunk | 05:30 | 06:00
 cc_data       | _hyper_2_16547_chunk | 05:30 | 06:00
 cc_sedc_data  | _hyper_4_16542_chunk | 05:00 | 05:30
 bc_sedc_data  | _hyper_3_16541_chunk | 05:00 | 05:30
 cc_data       | _hyper_2_16543_chunk | 05:00 | 05:30
 bc_data       | _hyper_1_16544_chunk | 05:00 | 05:30
(8 rows)
```

The above output shows the last two chunks of time (05:00 - 06:00 and 05:30 to 06:00) for each of the four tables that house power data. The actual data for each of these chunks resides in the *_timescaledb_internal* schema, and these tables can be treated as any other normal table. For example, to find out how many rows there are in the first chunk of the last result:

```
pmdb=# select count(*)
pmdb-# from _timescaledb_internal._hyper_1_16546_chunk;
count
--------
544842
(1 row)
```

V. Cray Power Management Usage Examples

The previous sections in this paper have described many of the power monitoring and control features of the Cray XC50 system. In this section we show some examples of using some of the monitoring and control features. In Subsection V-A we show and example with some experimentation running an unoptimized version of the industry standard stream memory benchmark over the range of supported frequencies of the Cray XC50 test system featuring Intel Xeon Scalable Processors. Then in Subsection V-B we use the same test system and nodes to show the use of power capping controls to modify the power ramp rate at job startup for nodes running a Cray diagnostics build of xhpl.

A. Stream examples using P-States

In this subsection, we show two simple examples of running stream with different "P-State" (or frequency) settings. The two examples will differ in how the frequency is set, and we also show some different ways of looking at power profiles collected for the test runs.

First, we show the use of selecting a "P-State" at job launch. The test system was configured to support the *qsub*.

We used the *qsub* command to launch a scrip that makes use of the Cray *aprun* command, which is part if the Cray Application Level Placement Scheduler (ALPS) software suite. Listing 3 shows the simple bash script used in this experiment. The script first uses *gcc* to compile stream.c as downloaded from https://github.com/jeffhammond/STREAM. The option *-DNTIMES=2048* is used extend the time of each test iteration to just under two minutes. The script then iterates over a list for supported processor frequencies for the nodes in the test system passing the requested frequency (in kHz) via the *aprun –p-state=kHz* option. The script then injects a 20 second delay between iterations.

This experiment could also be implemented on a system using SLURM job launch, using the *–cpu-freq* option.

Listing 3. do_run_stream

```
user:~/STREAM> cat do_run_stream
#!/bin/bash
## qsub -lnodes=12:ppn=1 ./do_run_stream

cd /home/users/user/STREAM

gcc -O -DSTREAM_ARRAY_SIZE=100000000 \
          -DNTIMES=2048 \
          -fopenmp stream.c \
          -o stream_omp_DNT_2048

for xxx in 2101000 2100000 2000000 \
                1900000 1800000 1700000 \
                1600000 1500000 1400000 \
                1300000 1200000 1100000 1000000
do
    aprun -n 12 -N 1 -cc none --p-state=$xxx \
       ./stream_omp_DNT_2048 | tee sweep.$$.$xxx.SK48_stream
    sleep 20
done
```

Figure 4 shows the power profile for stream on our test system, running on 12 Cray XC50 nodes with Intel Xeon Scalable Processors. The power profile was generated from data in the Cray PMDB. The blue lines in Figure 4 show (at 1Hz) total node power data average of 12 nodes, maximum node power, and minimum node power for each of the 13 stream iterations. The red lines are minimum-, average-, and maximum-CPU power, and the green lines show the same for memory power. One observation that stands out from this data is that the memory power is consistently high over all of the tested CPU frequencies. And unlike the memory power, the CPU and total node power decrease as the frequency is reduced.

This is not a performance paper, but when we look at the stream output files, the results are in line with the memory power, in that they are not highly correlated with the CPU frequency. Figure 5 shows that the "Best Rate MB/s" performance data output from the (unoptimized) stream test is flat over the supported CPU frequency range.

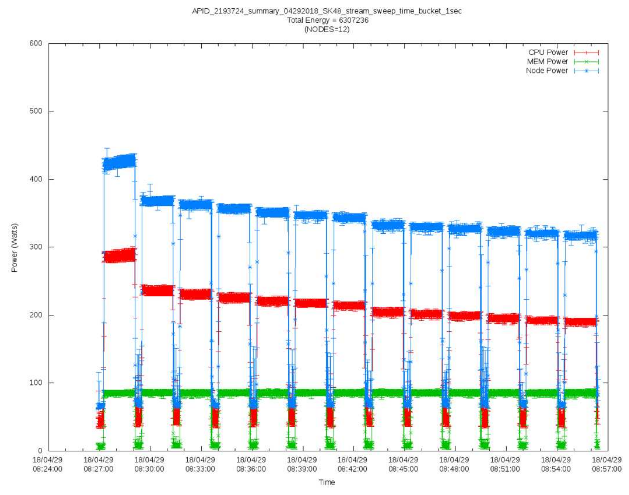

Figure 4: Stream Power Profile (ALPS)

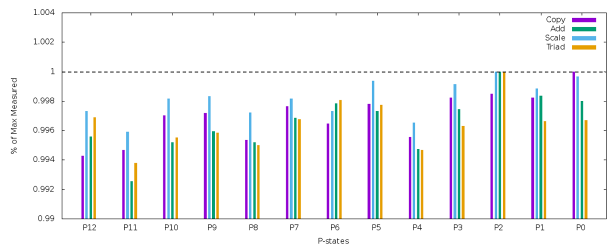

Figure 5: Relative (Best Rate MB/s) performance

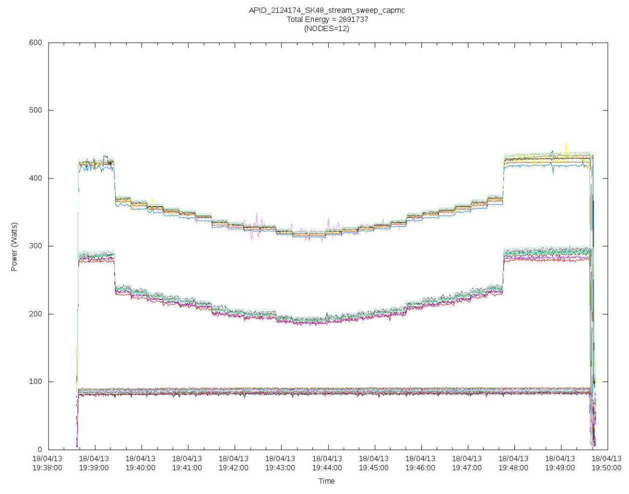

Figure 6: Example 1: Stream Power Profile (CAPMC)

In Listing 4 we show test script that supports the use of CAPMC to iterate over supported frequency settings. The script starts with the maximum *Turbo Frequency (2101000)*, lowers frequencies by iterating over the *$FREQ_H2L* list, and then from minimum frequency back to the maximum iterating over the *$FREQ_L2H* list. The underlying work is done by the call to *capmc set_freq_limits -n $NIDS -m*

1000000 -M $xxx. The script was run on the test system's SMW as *user:crayadm*, it could be run on properly configured service nodes by an authorized user. This CAPMC functional is targeted at third party WLM vendors, or researchers with escalated privileges.

Listing 4. CAPMC Frequency Walking Script
```
#!/bin/bash
SLEEP_TIME=20
FREQ_H2L="2101000 2100000 2000000 1900000 \
        1800000 1700000 1600000 1500000 1400000 \
        1300000 1201000 1100000 1000000";
FREQ_L2H="1000000 1100000 1200000 1300000 \
        1400000 1500000 1600000 1700000 1800000 \
        1900000 2000000 2100000 2101000";
NIDS="28,29,30,31,68,69,70,71,104,105,106,107"

do_it() {
  sleep $SLEEP_TIME;
  for xxx in $FREQ_H2L $FREQ_L2H ;
  do
    echo -n "$xxx :"; date;
    capmc set_freq_limits -n $NIDS -m 1000000 -M $xxx;
    sleep $SLEEP_TIME
  done
}
{lst:SK48_PCAP_walk}
module load capmc;
```

In Figure 6 we show the resulting power profile for a test run where the test script described above was run after the test started. As with the previous runs using "P-State" at job launch, the memory power and streams performance are not lowered as the frequency is changed, but the total node power and CPU power are reduced. Stream is designed to be memory bandwidth bound, so these findings are not unexpected. This example shows how power and frequency profiling can be done on the Cray XC systems.

B. Xhpl examples using Power Capping

Next, we show an example of using the CAPMC API to do power capping. The goal in this example is to show how power capping controls work in general by shaping the power consumption of a compute intensive application job launch. The test binary selected is the Cray xhpl binary distributed with the Cray diagnostics package. The script shown in Listing 5 was run on the test system's SMW as *user:crayadm*. The script first applies the minimum power cap and then prompts the user to "Enter to continue: ", then after the user hits enter it steps the power cap up in 5 watt increments, with a 5 second delay at each step.

We used the following test procedure:

1) **login-node:** Using *qsub -I* get an interactive allocation of test nodes that matches the "$NIDS" list in the power capping script.
2) **SMW-crayadm:** After the *qsub -I* completes, start the power capping script and wait for the "Enter to continue: " prompt. Do not hit enter yet.
3) **login-node:** Start the xhpl job from inside the allocation from the call to *qsub -I* in step 1.
4) **SMW-crayadm:** Now hit enter at the "Enter to continue: " prompt. The power capping script should complete about half way through the xhpl test run.

Listing 5. CAPMC Power Cap Ramp Up Script
```
#!/bin/bash
: ${SLEEP_TIME=5}; : ${PCAP_STEP=5};
: ${PCAP_MIN=240}; : ${PCAP_MAX=425};
: ${NIDS="28,29,30,31,68,69,70,71,104,105,106,107"};

set_cap() {
 echo -n "$1, $2 :"; date;
 capmc set_power_cap -n $1 --node $2
 sleep $SLEEP_TIME
}

do_it() {
 set_cap $NIDS $PCAP_MIN;
 echo -n "Enter to continue: "; read
 for ((xxx=PCAP_MIN; xxx <= PCAP_MAX; xxx += PCAP_STEP))
 do
  set_cap $NIDS $xxx
 done
 set_cap $NIDS 0;
}

module load capmc;
do_it
```

Figure 7 shows an aggregate view of data from all 12 nodes running xhpl with the power capping script modifying the the job power up ramp rate. The data from PMDB is averaged for all nodes in 6 second time buckets, and average power, maximum power, and minimum power are shown using *gnuplot "errorlines"*, the *group by time_bucket* function provided by the PostgreSQL extension is used by the SQL query from PMDB. We think this graphic representation shows interesting insight into data, especially when looking at jobs with much larger node counts.

Figure 8 gives a different view of the same data, this time displaying the 1Hz data directly out of PMDB but selectively showing only the data for three of the twelve nodes. The three selected nodes were selected by the script that generated the plots by first sorting the nodes by total energy used, then picking the the nodes from the top, bottom, and middle of the list.

The node-minimum and node-maximum data shown using the *gnuplot "errorlines"* in Figure 7 and again in the node-level data in Figure 8 both seem to show regions with more and less noisy behavior in the power ramp up. This might me an interesting subject for additional research on power capping.

Finally, Figure 9 shows the xhpl test program running on the same nodes of the test system, this time without power capping. This shows the near instantaneous per-node power ramp up from less then 50 watts per node to over 400 watts per node.

The examples shown in this paper are simple, and intended to show some Cray XC50 power management capabilities in a way that can be useful to a wide audience. We believe the real-world use cases that these capabilities support are very important to the Cray user community.

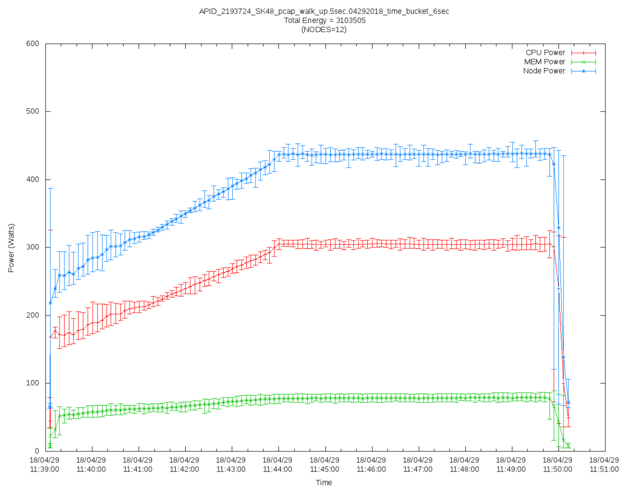

Figure 7: Xhpl Power Profile (aggregate view)

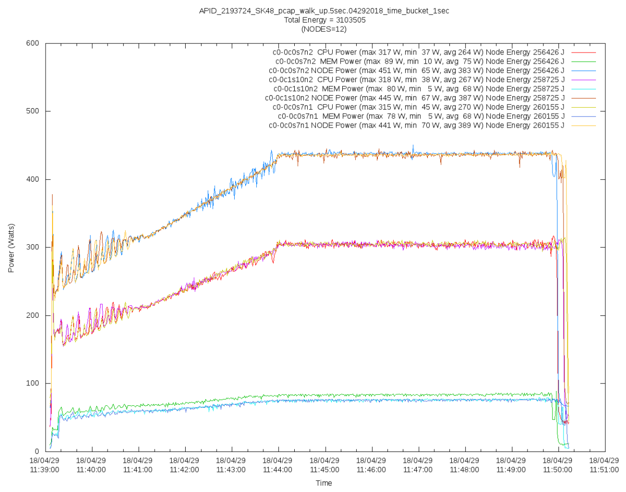

Figure 8: Xhpl Power Profile (node view)

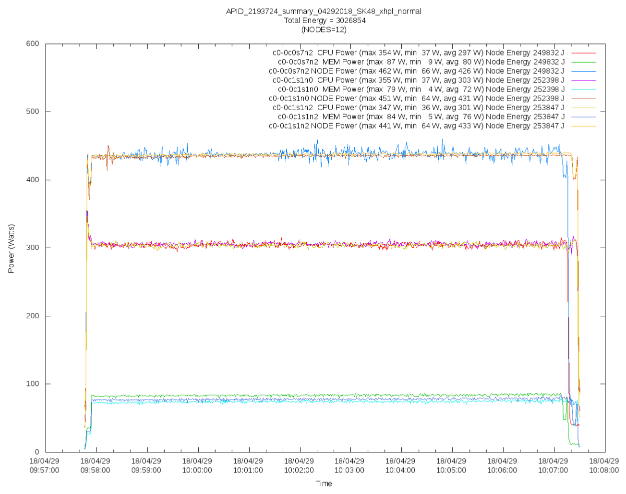

Figure 9: Xhpl Power Profile (no capping)

VI. Conclusion

This paper has provided an overview and updated description of the power management features of the Cray XC50 system, focusing mostly on the Cray XC50 compute blades. The paper started with a general description of features, then focusing in more detail on the features of the Cray XC50 blade that supports the Intel Xeon Scalable Processors. An update was then provided for the Cray PMDB with some details on how the design was changed to leverage the Time Scale extension for PostgreSQL, and related PMDB configuration management tool changes. Finally the paper provided two examples of using power management monitoring and control features on a Cray XC50 system. We are committed to providing the power and energy monitoring and control features needed for leadership class HPC systems. We encourage the user community to use these features and give us feedback on how we can improve, add new capabilities, and together solve new challenges.

References

[1] S. Martin, D. Rush, M. Kappel, M. Sandstedt, and J. Williams, "Cray XC40 Power Monitoring and Control for Knights Landing," *Proceedings of the Cray User Group (CUG)*, 2016, (Accessed 22.March.17). [Online]. Available: https://cug.org/proceedings/cug2016_proceedings/includes/files/pap112s2-file1.pdf

[2] S. Martin, D. Rush, and M. Kappel, "Cray Advanced Platform Monitoring and Control (CAPMC)," *Proceedings of the Cray User Group (CUG)*, 2015, (Accessed 24.Aprl.18). [Online]. Available: https://cug.org/proceedings/cug2015_proceedings/includes/files/pap132.pdf

[3] S. Martin, C. Whitney, D. Rush, and M. Kappel, "How to write a plugin to export job, power, energy, and system environmental data from your cray xc system," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 1, p. e4299. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4299

[4] "XC Series Power Management and SEDC Administration Guide (CLE 6.0.UP06) S-0043," (Accessed 6.April.18). [Online]. Available: https://pubs.cray.com/pdf-attachments/attachment?pubId=00529040-DB&attachmentId=pub_00529040-DB.pdf

[5] *CAPMC API Documentation, (CLE 6.0UP06) S-2553*, Cray Inc., (Accessed 9.May.18). [Online]. Available: https://pubs.cray.com/content/S-2553/CLE%206.0.UP06/capmc-api-documentation

[6] G. Fourestey, B. Cumming, L. Gilly, and T. C. Schulthess, "First experiences with validating and using the cray power management database tool," *arXiv preprint arXiv:1408.2657*, 2014, (Accessed 24.Aprl.18). [Online]. Available: http://arxiv.org/pdf/1408.2657v1.pdf

[7] A. Hart, H. Richardson, J. Doleschal, T. Ilsche, M. Bielert, and M. Kappel, "User-level power monitoring and application performance on cray xc30 supercomputers," *Proceedings of the Cray User Group (CUG)*, 2014, (Accessed 31.March.16). [Online]. Available: https://cug.org/proceedings/cug2014_proceedings/includes/files/pap136.pdf

[8] S. Martin and M. Kappel, "Cray XC30 Power Monitoring and Management," *Proceedings of the Cray User Group (CUG)*, 2014, (Accessed 24.Aprl.18). [Online]. Available: https://cug.org/proceedings/cug2014_proceedings/includes/files/pap130.pdf

[9] "CrayPAT: Cray Performance Measurement and Analysis," Cray Inc., (Accessed 29.Apr.18). [Online]. Available: https://pubs.cray.com/pdf-attachments/attachment?pubId=00505504-DA&attachmentId=pub_00505504-DA.pdf

[10] "PAPI: Performance Application Programming Interface," (Accessed 25.Apr.18). [Online]. Available: http://icl.cs.utk.edu/papi/

[11] M. Bareford, "Monitoring the Cray XC30 power management hardware counters," *Proceedings of the Cray User Group (CUG)*, 2015, (Accessed 29.Apr.18). [Online]. Available: https://cug.org/proceedings/cug2015_proceedings/includes/files/pap125.pdf

[12] S. P. Jammy, C. T. Jacobs, D. J. Lusher, and N. D. Sandham, "Energy efficiency of finite difference algorithms on multicore cpus, gpus, and intel xeon phi processors," *arXiv preprint arXiv:1709.09713*, 2017.

[13] "Cray technical documentation," (Accessed 26.April.18). [Online]. Available: https://pubs.cray.com/

[14] "Vicorpower," (Accessed 17.Apr.18). [Online]. Available: http://www.vicorpower.com

[15] "High Current, Low Voltage Solution For Microprocessor Applications from 48V Input," (Accessed 17.Apr.18). [Online]. Available: http://www.vicorpower.com/documents/whitepapers/whitepaper2.pdf

[16] "LM5066I 10 to 80 V Hotswap Controller With I/V/P Monitoring and PMBus Interface," (Accessed 28.April.18). [Online]. Available: http://www.ti.com/lit/ds/symlink/lm5066i.pdf

[17] "Intel 64 and IA-32 architectures software developer's manual, volume 3b: System programming guide, part 2, order number: 253669-047US," (Accessed 11.Apr.14). [Online]. Available: http://download.intel.com/products/processor/manual/253669.pdf

[18] "PostgreSQL," (Accessed 12.May.18). [Online]. Available: http://www.postgresql.org/

[19] "Timescaledb GitHub," (Accessed 12.Apr.18). [Online]. Available: https://github.com/timescale/timescaledb

[20] "TimescaleDB Documentation," (Accessed 12.Apr.18). [Online]. Available: https://docs.timescale.com/v0.9/main

[21] "Timescale," (Accessed 12.Apr.18). [Online]. Available: