ARCHIVIO ISTITUZIONALE
DELLA RICERCA

Alma Mater Studiorum Università di Bologna
Archivio istituzionale della ricerca

User-space APIs for dynamic power management in many-core ARMv8 computing nodes

This is the final peer-reviewed author's accepted manuscript (postprint) of the following publication:

Published Version:
Bortolotti, D., Tinti, S., Altoè, P., Bartolini, A. (2016). User-space APIs for dynamic power management in many-core ARMv8 computing nodes. NY : Institute of Electrical and Electronics Engineers Inc. [10.1109/HPCSim.2016.7568400].

Availability:
This version is available at: https://hdl.handle.net/11585/613812 since: 2019-04-23

Published:

DOI: http://doi.org/10.1109/HPCSim.2016.7568400

Terms of use:

Some rights reserved. The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (https://cris.unibo.it/).
When citing, please refer to the published version.

(Article begins on next page)

16 November 2026

This is the post peer-review accepted manuscript of:

D. Bortolotti, S. Tinti, P. Altoé and A. Bartolini, "User-space APIs for dynamic power management in many-core ARMv8 computing nodes," 2016 International Conference on High Performance Computing & Simulation (HPCS), Innsbruck, 2016, pp. 675-681.

The published version is available online at: https://doi.org/10.1109/HPCSim.2016.7568400

© 2018 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works

User-space APIs for Dynamic Power Management in Many-core ARMv8 Computing Nodes

Daniele Bortolotti
EEES - DEI
University of Bologna
Bologna, Italy
daniele.bortolotti@unibo.it

Simone Tinti, Piero Altoé
E4 Computer Engineering
Scandiano, Italy
simone.tinti@e4company.com
piero.altoe@e4company.com

Andrea Bartolini
Integrated Systems Laboratory
ETH Zurich
Zurich, Switzerland
barandre@iis.ee.ethz.ch

***Abstract*—The push for energy-efficient and energy-proportional computing nodes, together with the increasing number of cores integrated in the same silicon die has lead to computing nodes with fine grained power management capabilities. To unleash the potential of this HW design a novel user-space power management APIs is needed to bring fine-grain power management in the hands of the programmer. In this work we present a novel programming mechanism for energy efficiency which is build around novel user-space power management APIs suitable to be embedded in user-space applications. We evaluated its timing and power saving performance on a novel computing node based on Cavium ThunderX ARMv8-based many-cores SoC.**

I. Introduction

Supercomputers peak performance is expected to reach the ExaFlops (10^{18} Flops) scale in 2023 [1], as revealed by the exponential increase of the worldwide supercomputer installation [2]. With almost 100x more computational capabilities than today's most powerful supercomputers, the Exascale machine will bring supercomputer-assisted calculations into our daily life, revolutionizing many aspects of society, such as manufacturing, transportation, health and decision making, among others. However, Exascale supercomputers cannot be built by simply expanding the number of processing nodes and by leveraging technology scaling, as the power demand would increase unsustainably (in the order of hundreds of MW). To be sustainably powered at the Exascale level, current supercomputers must achieve a quantum leap in energy efficiency, pushing towards the goal of 50 GFlops/W.

Dynamic resource management and power management aim at reducing the energy consumption of computational systems by selecting at run-time the best operating point, which reduces the power consumption while preserving a given QoS for the user. The available operating points depend on mix of architectural features and physical low-power design strategies [3], [4]. According to the chip manufacturer and market segment, the configuration or run-time selection of these operating points are either handled at the firmware level or by the Operating System (OS) [4], [5], [6]. Today two main low-power mechanisms are available on general purpose processors to modulate the power consumption and to increase the energy-efficiency, namely dynamic voltage and frequency scaling (DVFS [7]) and power-gating. DVFS allows to dynamically scale the operating frequency and the voltage to reduce the power consumption and eventually save energy. Indeed, by reducing the voltage and the frequency of electronic devices the power consumption decreases super-linearly (with a theoretical cubic law). However, the application execution time may be affected by the reduced frequency up to a linearly proportional dependency with the clock frequency in case of a CPU-intensive load. These power states are referred in ACPI standard as P-states [4].

Power-gating instead aims at reducing the idle power by dynamically unplugging the cores voltage supply, thanks to a power gating switch. Differently from low-frequency DFVS states, the power-gated core achieves zero power consumption, also leakage power is removed at the cost of a context loss. Moreover according to the architecture, the power gated region can cover different logic areas (cores, cores + L1, individual cores + L1, cores + uncores, etc). Clearly, the more coarse the power-gated region is, the more costly the context loss will be. These power states are refereed in ACPI standard with C-states. Power-gating C-states are handled by the OS kernel by switching-off cores when their are unused (idle) for a time longer than a defined threshold. In this way it avoids to pay the context loss overhead for very short power-gated periods, which would lead to an overall performance and energy loss. Short idle periods are handled with HW mechanisms which do not induce context loss, however with reduced power savings (i.e. clock-gating and low-frequency DVFS states).

As a matter of fact, power-gated states which are capable of significantly reducing the power consumption are used only when cores stay idle for long periods. However, in current OSs several interrupts and kernel threads keep the cores active even when they do not execute any application [8], [9], [10]. On the contrary, applications running in computing nodes, usually empowered with accelerators, show workloads and communication phases not ideal for parallel scaling leading to potential idle-slack inside the application execution flow. This behavior cannot be captured by current OSs power management infrastructure as opportunistic and application-agnostic approaches would lead to an overall performance and energy loss. Even if the programmer could identify the most exploitable phase, it would lack mechanisms to insert from user-space power management directives directly inside

its code, while preserving a consistent environment.

Pushed by technology scaling, while searching for higher energy-efficient and cost-effective designs, chip manufacturers started to focus their attention on stand-alone many-core SoCs in additions to classical multi-core chips. Differently from their "big-brother" counter-parts, many-cores devices use simpler cores to achieve a higher cores count in the same silicon area and/or power budget, with the ultimate goal of providing a finer adaptation to workload phases and computational demand [11], [12], [13], [14], [15], [16], [17]. Recently the introduction of the 64-bit ARM architecture has created the opportunity for a wider range of chip manufacturers to propose solutions for the HPC market, trying to fill the embedded-to-HPC performance gap by increasing the cores count. In parallel several attempts have been made to integrate supercomputer machine with embedded processors [18], as these devices are characterized by stronger energy-efficiency constraints and are pushed by a more competitive and growing market [18].

As effect of the higher cores count and simpler cores logic, fine grained DVFS support would cause large overheads in many-cores devices making fine grained power-gating a more attractive low-power design solution. However to transform the latter as a base for the deployment of more energy-proportional computing nodes there is the need to offer such power management features to the programmer with novel APIs. This can happen only if computing nodes with software support become available to the wide public.

In this paper we aim for this by deploying a holistic co-design which includes (i) the design of a real computing node prototype based on a novel Cavium ThunderX SoC [16], (ii) the integration in the OS of the control mechanisms for fine grained power-gating, available on the target SoCs, (iii) the design of user-space power management APIs suitable to be added by the programmer in the scientific applications and finally (iv) an evaluation of the energy saving potential and the limiting factors for its achievement in current systems.

Summarizing, the main contribution of this work are the following:

- the integration of fine grained power-gating features in the Linux Kernel for switching off the unused cores when not used. We empirically demonstrate that the linux CPU hotplug system can be used as an enabling framework for exposing this low-level power management mechanisms to the user-space. Our results show that this mechanism can lead up to a $\approx 35\%$ of power reduction in the CPU logic and up to $\approx 24\%$ of idle power reduction at the node level.
- the creation of the user-space APIs based on the introduction of two novel system calls to be inserted directly in the application code. We quantified the time granularity at which this mechanism can operate, showing that even if in today's OSs this mechanism does not scale with the integrated number of cores, already presented solutions allow us to accurately estimate the final impact.

The rest of the paper is organized as follows. In Section III the state-of-the-art computing node is presented, while Section IV focuses on the the programming support for energy efficiency. In Section V we describe the experimental setup and the results of the evaluation in terms of energy consumption. Finally, the conclusions are presented in Section VI.

II. Related Work

The ACPI standard [4] defines several power management states which can be configured dynamically. P-states groups dynamic voltage and frequency scaling operating points. These can be selected at run-time to modulate the power consumption during active computation. Differently, C-states groups idle power management states which targets idle power reduction. C0 is the active idle state while C1 is the active low-power state and C3 is the power-gated idle state. Differently from C0 and C1, C3 leads to a loss of information and context in the logic area which is affected by the power gating. The Linux Kernel uses the ACPI power states by mean of two different kernel drivers, namely the "cpuidle" and "cpufreq".

The cpufreq driver modulates the P-states by mean of power governors which can be set to implement different policies. The most used one is the on-demand governor which modulates the current P-state to maximize the core utilization. It is effective with I/O intensive tasks for which the core is often in idle waiting for I/O operations. In a node of a supercomputer this condition is not frequent and thus system administrators prefer to keep the computing node always operating at the maximum frequency. The authors in [19] show that DVFS, if carefully used, can lead up to the 27% of energy saving considering real supercomputing workloads. In addition, the authors show that the optimal frequency level depends on the application, as more memory intensive applications achieve the most energy-efficient operating point at lower speed with respect to CPU-bound ones. Following this approach authors of [20] proposes to run the computing cluster at a lower but more energy-efficient frequency than the maximum for the general users, and then increase it to the maximum one only for running the applications which would obtain large benefits from executing in faster core. These applications are identified based on performance counters online characterization.

The cpuidle driver modulates the C-states according to the duration of an idle phase. In the Linux Kernel the idle task executes each time a core does not have useful tasks to be performed. Internally the idle task loops over architecture specific instructions which can selected to reduce the power consumption pushing the core to an idle. As deeper ACPI states incur in costly transition overheads, these transitions are only triggered when it is likely to stay in this state for a long time. This is done by setting a minimum time threshold before entering the deep low-power state [21]. However Operating Systems noise induced by periodic activities in the Linux Kernel reduces the maximum length of the idle phases limiting de-facto the usage of deep power states[9], [10].

In addition to cpuidle the linux kernel has a mechanisms to logically offline and online CPU cores. This is done by the CPU hotplug subsystem which exposes a virtual filesystem entry for each core which can configured dynamically to

logically switching on and off the operating system from that cores.

Authors in [22] present Barrelfish/DC, an extension to the Barrelfish OS which decouples physical cores from a native OS kernel. Thanks to this approach, the authors can replace and reboot the OS kernel in each core independently and migrate per-core OS state in between cores. As a result in Barrelfish/DC stopping cores costs only from $0.8\mu s$ (Haswell) to $3.5\mu s$ (Sandy-Bridge), while with the standard Linux the authors measured a core shutdown time ranging in between $46ms$ to $131ms$. As a matter of fact Barrelfish/DC achieves a 55x speedup. The work in [23] shows that by rethinking the Linux Kernel core hotplug mechanism, its performance can be drastically improved. The authors present Bolt which extends the existing hotplug infrastructure to reduce the transition latency betwee CPU states, as well as supports insertion and removal of multiple cores at once. Bolt can achieve over 20x speedup in the core shutdown routine and in between 13x to 20x for the online time. All the approaches mentioned above focus on the time taken by the OS Kernel to logically remove or restart a given core, without evaluating the real energy efficiency potentials when these mechanisms are connected to real power management mechanisms. In this paper we evaluate such potentials by inserting our novel low-level mechanisms in a scientific application to deploy an holistic co-design for energy efficiency.

III. Computing Node

The recent introduction of the many-cores ARMv8 SoC is a disruptive innovation for the market. Several chip designers have announced new silicon within very tight roadmaps, with early results already available. Lately, various companies have proposed low-power micro-servers (multi- or many-cores processors with a simple micro-architecture). Furthermore, most of these microservers have the potential to execute enterprise workloads, e.g. memory-intensive big data applications, the presentation layer of large-scale web applications, etc. Some examples of such products are Cavium 48 ARM cores ThunderX [16], Applied Micro X-Gene ARM server 2 [17], etc. These processors comes with interesting design trade-offs and power management features.

The considered computing node is composed by two Cavium ThunderX SoCs, 256GB of DDR3 memory and an Infiniband Mellanox Card. The node follows Openrack standard and is powered from a busbar and features a mixed of air and liquid cooling. The CPUs are liquid cooled while the rest of the components are air cooled. Figure 1 shows the computing node. Each node is equipped with a current sensor and a voltage sensor which measures the energy and power consumption with a ms of time granularity. These values are integrated over period of 1 *sec* and made available to the software stack.

Figure 2 shows a block scheme of the ThunderX SoC by Cavium. It features:

- Scales from 24 to 48 cores with up to 2.5GHz frequency;
- 78KB-I cache and 32KB-D cache per-core;

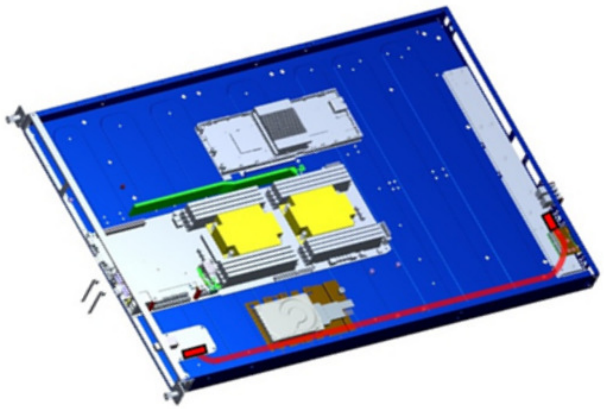

Fig. 1. Computing Node composed of 2 ThunderX SoCs, DDR memory, interconnection and mixed air/liquid cooling.

- 16MB shared L2 cache;
- Single and dual socket configuration support via (Cavium Coherent Processor Interconnect (CCPI);
- Up to 4 DDR3/4 memory controllers;
- Up to 1TB of memory capacity in dual socket configuration;
- Multiple 10/40GE ports;
- Multiple independent SATAv3 controllers;
- Multiple PCIe - x4, x8 support (24 lines total);

In addition the Cavium ThunderX SoC has a fine grained low-power management support with selective cores power gating, giving to the architecture peculiar power savings features, thanks to the many-core architecture and the fine grained cores shutdown possibilities. The per-core power gating can be configured online by means of specific machine registers, visible from all the cores and sockets. The potential of reducing the idle power by switching off the entire power consumption of the core region can be effective if (i) is integrated in the OS kernel and (ii) is provided to the final user in form of APIs to be inserted in the application.

Finally the computed node features a built-in power mea-

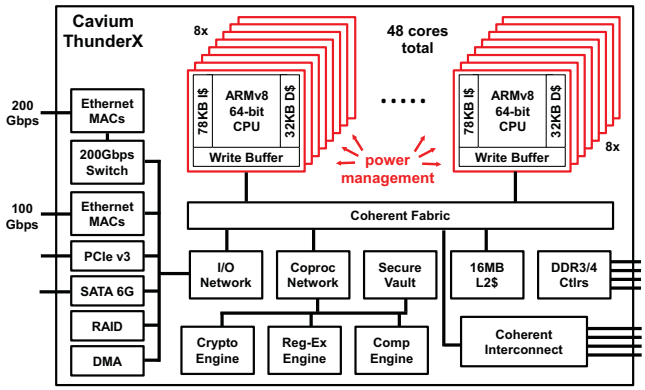

Fig. 2. Cavium ThunderX schematic view: highlighted in red are the power management knobs.

surement systems which measures through an hall-effect sensors the entire node power as well as the per-component (CPUs, DRAMs) power through the BMC. The node power measurement is measured with high-frequency sampling and averaged in periods of 1s.

IV. Programming Support for Energy-Efficiency

A. Linux Kernel

To integrate Cavium's specific low-level register calls inside the Hotplug mechanism, we modified the Linux Kernel choosing as starting point version $4.2.0-20.1$. By bridging the gap between OS and real HW knobs for power management, Hotplug can effectively zero the power consumption of a given core when triggered via the `sysfs` Linux interface, e.g.

```
echo 0 > /dev/sys/devices/system/cpu/cpuX/online
```

or alternatively resume a core, by switching it on with

```
echo 1 > /dev/sys/devices/system/cpu/cpuX/online
```

Clearly this mechanisms will be usable only for system administrators only as it requires root access, and thus will not lead to fine grained energy management while the application is running on the node. The block scheme in Figure 3 shows where we inserted our routines in the kernel code to implement low-level power gating.

Inside the `_cpu_down()` routine we added the initialization procedure, to set up and initialize data structures and remap virtual addresses to access low-level registers. As a last operation before leaving the inner lock (`hotplug.lock`), we inserted the procedure that accesses Cavium registers and power gate a given core. It is important to notice the double lock structure that guarantees a safe removal of the CPU from the OS kernel, but will cause transitions overheads.

B. User-space

To expose the power managements knobs from kernel mechanisms to the application programmer, we added two new architecture specific system calls to the Linux Kernel. The system call definitions, namely `pm_coredown` and `pm_coreup`, are reported in Listing 1 and 2 respectively.

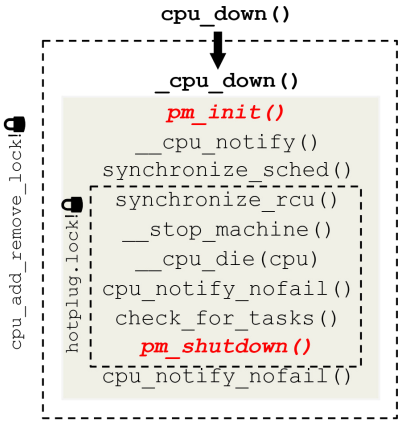

Fig. 3. Hotplug modification to add low-level power management features.

```
SYSCALL_DEFINE2( pm_coredown, unsigned int *,
                 cpu_v, unsigned int, N )
{
  int i,ret;
  unsigned int cpu_kv[_NC];

  if ( copy_from_user( cpu_kv, cpu_v,
                       sizeof(int) * N ) )
      return -EFAULT;

  for(i=0; i<N; i++) {
    /* shut down the cores */
    ret = cpu_down(cpu_kv[i]);
    /* error handling ... */
  }

  return 0;
}
```

Listing 1. System call definition for `pm_coredown()`.

```
SYSCALL_DEFINE2( pm_coreup, unsigned int *,
                 cpu_v, unsigned int, N )
{
  int i,ret;
  unsigned int cpu_kv[_NC];

  if ( copy_from_user( cpu_kv, cpu_v,
                       sizeof(int) * N ) )
      return -EFAULT;

  for(i=0; i<N; i++) {
    /* bring on the cores */
    ret = cpu_up(cpu_kv[i]);
    /* error handling ... */
  }

  return 0;
}
```

Listing 2. System call definition for `pm_coreup()`.

An example of the interaction between application (user-space) is schematized in Figure 4. The example considers an application that requires to switch off 6 cores and bring them back online after a kernel has executed.

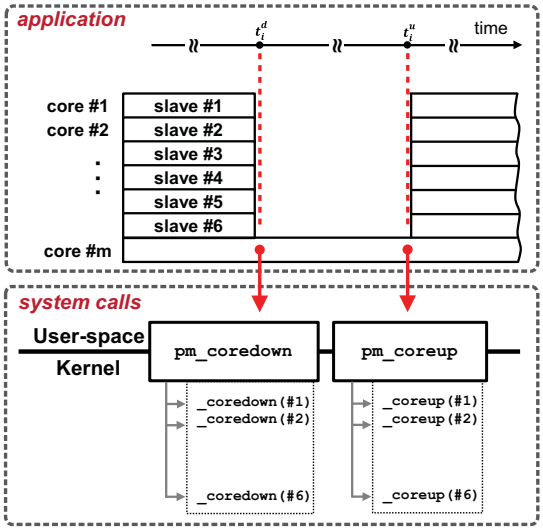

Fig. 4. APIs interaction between application (User-space) and OS Kernel.

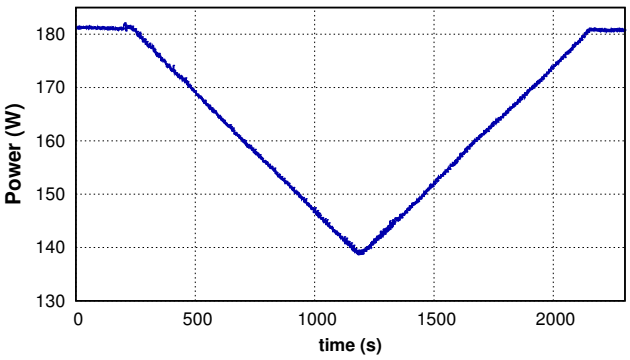

Fig. 5. Total power reduction through the power management APIs. Test: shutdown of 95 out of 96 cores, with a sleep of 10 seconds in between.

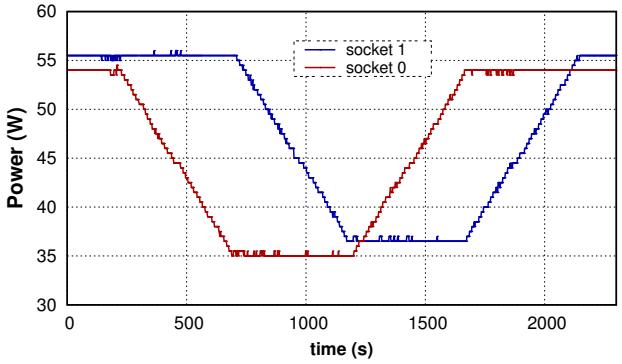

Fig. 6. Power reduction per socket through the power management APIs. Test: shutdown of 95 out of 96 cores, with a sleep of 10 seconds in between.

V. Experimental Results

In this section we show a characterization of the APIs in terms of power and timing.

A. APIs Power Characterization

To show the potential of our approach we created a simple user-space application that deploys our APIs to sequentially switch off all cores (except core0) and then later switch them back on. The code of the application is reported below.

```
#define _GNU_SOURCE
#include <unistd.h>
#include <stdio.h>
#include <sched.h>
#include <sys/syscall.h>

int main(int argc, char *argv[])
{
    unsigned int i, N = 95;
    long error;

    /* sched affinity on core0 ... */

    /* shutdown */
    for(i=1; i<=N; i++) {
      error = pm_coredown(&i,1);
      /* error handling ... */
      sleep(10);
    }

    sleep(20);

    /* wakeup */
    for(i=1; i<=N; i++) {
      error = pm_coreup(&i,1);
      /* error handling ...*/
      sleep(10);
    }

    return 0;
}
```

Listing 3. Synthetic benchmark with sequential cores shut-down for power characterization.

The application first loops over each core and shuts them down by means of the `pm_coredown` API. The API takes as input a pointer to a list of cores ID to be switched off (`i` in this case) and the length of the list (`N`). After switching off one core the application slepdf for 10 seconds. When all the 95 cores have been switched off, the application restarts one after the other through the `pm_coreup` API. By executing in the user-space, this mechanism allow the programmer to decide on-line which cores to use and which not. Figure 5 shows the execution of the synthetic benchmark in terms of power consumption of the entire node. As we can notice from Figure 5 the total power decreases from 181W to below 140W.

Figure 6 instead shows the separate power consumption of the ThunderX sockets.

The application first shuts down the cores in socket#0 and then in socket#1 (whose cores have a higher logical ID). From the same figure we can notice that both the two sockets consume similar power and that APIs are capable of saving $\approx$ 20W for each of them.

Table I reports the power measured for the node in idle, i.e. when all the core are in C1-state, and when they are progressively shut down. For each of the different cases we report the real measured node-level power consumption and the power consumption per-socket. It can be seen that the SoC accounts for the 60% of the total power consumption in idle. The table shows that this amount can be effectively reduced thanks to the introduced power management mechanisms. Indeed as the number of core per-socket gets switched off by our mechanism, each socket consumes up to the 35% less while the node power reduces of the 11% when 48 cores out of 95 are switched off and up to 23% when only one core for the entire node is kept on. We plan to improve these results in future works by extending the energy-efficiency APIs with system level power-gating techniques to enable the user to dynamically switch-off additional peripherals in each socket and parts of the memory subsystem.

B. APIs Time Characterization

In this section we characterize the performance of the energy-aware APIs in terms of transitions costs. As previously introduced, and supported by other works [23], the Linux Kernel Hotplug mechanism implements several operations to logically detach and reconnect a core from the OS. This is visible in terms of timing overheads for each `pm_coreup` and `pm_coredown` transition. In addition internals calls in the `cpu_up()` and `cpu_down()` mechanisms internal to the

	cores shut down								
	0 (idle)	1	2	4	8	16	32	48	95
Power Socket #0 (W)	**54.0**	53.60	53.21	52.42	50.83	47.67	41.33	**35.00**	-
savings (%)	-	*0.73%*	*1.47%*	*2.93%*	*5.86%*	*11.73%*	*23.46%*	*35.19%*	-
Power Socket #1 (W)	**55.50**	55.10	54.71	53.92	52.33	49.17	42.83	**36.50**	-
savings (%)	-	*0.71%*	*1.43%*	*2.85%*	*5.71%*	*11.41%*	*22.82%*	*34.23%*	-
Total Power (W)	**181.57**	181.12	180.66	179.76	177.95	174.34	167.12	159.89	**138.67**
savings (%)	-	*0.25%*	*0.50%*	*0.99%*	*1.99%*	*3.98%*	*7.96%*	*11.94%*	*23.63%*

TABLE I
POWER CHARACTERIZATION OF THE DUAL-SOCKET SOC BY MEANS OF THE POWER MANAGEMENT APIS.

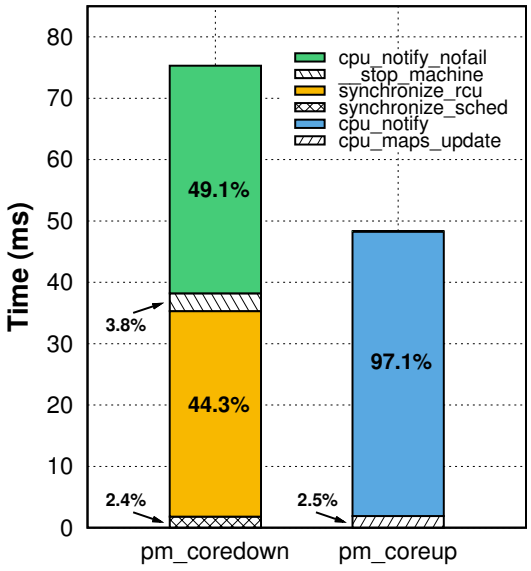

Fig. 7. Time breakdown for `pm_coredown` and `pm_coreup`.

Linux Kernel, the Hotplug mechanisms uses locks to serialize subsequent calls to the Linux Hotplug functionalities. This becomes severe when a large numbers of cores need to be switched on or off. A recent work from Panneerselvam et al. [23] investigates and proposes a lightweight reconfiguration mechanism for the Linux OS. For this reason, in this section we investigate the current performance of the proposed APIs with the standard Linux Hotplug mechanism and then we project the achievable performance using the results in [23].

Figure 7 shows with different bars the average time taken by the `pm_coreup` and `pm_coredown` system calls, when switching off 95 cores out of 96. The stacked bars report the time breakdown spent on the different CPU Hotplug calls. It can be noticed that `pm_coredown` takes significantly higher time to execute when compared to the `pm_coreup`, totaling in average 75.6ms compared to 47.8ms. In addition each of them shows a significant biased overheads towards specific subroutines. Namely the cpu_notify_nofail and synchronize_rcu which in sum accounts for the 89% of `pm_coredown` overheads and cpu_notify for the `pm_coreup` which accounts for its 92% of overhead. It must be noted that none of these three functions include the setting of the internal registers which is accounted in the remaining overheads. This results demystifies the myth that sees power management practice bounded by the HW transition time in between power management states.

The work from Panneerselvam et al. [23] introduces Bolt, a fast reconfiguration mechanism for OS. The functionality of Bolt is similar to hotplug in getting the system from one stable state to another stable after offlining the processors. However, Bolt aims at offering stability with a very low latency and is built by refactoring the existing hotplug infrastructure. Bolt achieves low latency by separating hotplug notifications into critical and non-critical operations. The former needs to be handled synchronously to ensure correctness of the system whereas the latter could be removed from the critical path and performed after the CPU goes online/offline. As we can see from the results for the ARM architecture, Bolt can achieve a 13x speedup for offline and almost 22x for offline. By projecting the scalability results to the number of cores in Cavium ThunderX, we can achieve with the Bolt Bulk Interface the results summarized in Table II.

From this projection the energy efficient APIs can obtain an extremely efficient scalability, being able to shut down the 95 out of 96 cores in $\approx$ 15ms. and bring them up online in $\approx$ 26ms. With these achievable transitions time we believe that the designed energy-efficiency APIs can be effectively coupled with parallel applications to reduce their energy consumption during weak scaling and serial phases as well as to reduce the cpu power when accelerators are empowered.

	NUMBER OF CORES SHUT DOWN **SYNCHRONOUSLY**								
	1	2	3	4	8	16	32	48	95
`pm_coredown` (ms)	3.32	3.63	3.75	3.87	4.38	5.37	7.38	9.38	15.06
`pm_coreup` (ms)	1.26	2.09	2.35	2.62	3.68	5.80	10.05	14.29	26.33

TABLE II
`PM_COREDOWN` AND `PM_COREUP` PROJECTION (TIME IN MILLISECONDS), ACCORDING TO [23].

VI. Conclusion

The increasing number of cores integrated in the same silicon die has led to computing nodes with fine grained power management capabilities. The Cavium ThunderX SoCs features 48 ARMv8 processors in the same silicon die with per-core power gating capabilities. In this paper we present the first computing node based on dual socket ARMv8 many-core SoCs together with a novel user-space power management APIs to unleash the potential of fine-grain power management allowing the user to dynamically switching on and off processing elements within its application. This has been done thanks to two novel system calls which extend the Linux Kernel CPU hotplug subsystem. Thanks to that we empirically demonstrate that the linux CPU hotplug system can be used as an enabling framework for exposing this low-level power management mechanisms to the user-space. Our results show that this mechanism can lead up to a $\approx 35\%$ of power reduction in the CPU logic and up to $\approx 24\%$ of idle power reduction at the node level. Moreover we quantified the time granularity at which this mechanism can operate, showing that even if in todays OSs this mechanism does not scale with the integrated number of cores, already presented solutions allow us to accurately estimate the final impact.

We believe that the proposed energy-efficient APIs can be effectively in increasing the energy-efficiency of today systems during serial workload phases and accelerators one. In future works we will evaluate their effectiveness in real scientific applications and relevant workload as well as we will improve their power-reduction performance by coupling them with system-level power saving mechanisms.

Acknowledgment

This work was commissioned by CINECA acting on its own behalf and on behalf of CSC - Tieeteen Tietotekniikan Keskus Oy, EPCC (University of Edinburgh), Forschungszentrum Jlich GmbH and GENCI. The views expressed in this publication are those of the author(s) and not necessarily those of the aforementioned entities. Additionally it was partially supported by the FP7 ERC Advance project MULTITHERMAN (g.a. 291125), by the EU H2020 FETHPC project Exanode (g.a. 671578) and by the YINS RTD project (no. 20NA21 150939), evaluated by the Swiss NSF and funded by Nano-Tera.ch with Swiss Confederation financing. Authors would like to acknowledge Cavium for the technical support.

References

[1] J. Hsu, "When will we have an exascale supercomputer? [news]," *IEEE Spectrum*, vol. 52, pp. 13–16, January 2015.

[2] J. J. Dongarra, H. W. Meuer, and E. Strohmaier, "Top500 supercomputer sites."

[3] I. Hewlett-Packard, "Microsoft, phoenix, and toshiba. advanced configuration and power interface specification," 2004.

[4] Unified EFI Inc., "Advanced configuration and power interface specification, version 6.1." http://www.uefi.org/sites/default/files/resources/ACPI_6_1.pdf, 2016.

[5] D. Hackenberg, R. Schone, T. Ilsche, D. Molka, J. Schuchart, and R. Geyer, "An energy efficiency feature survey of the intel haswell processor," in *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International*, pp. 896–904, IEEE, 2015.

[6] ARM Ltd., "Arm architecture reference manual armv8 (arm ddi 0487a.h)," 2015.

[7] P. Macken, M. Degrauwe, M. Van Paemel, and H. Oguey, "A voltage reduction technique for digital systems," in *Solid-State Circuits Conference, 1990. Digest of Technical Papers. 37th ISSCC., 1990 IEEE International*, pp. 238–239, IEEE, 1990.

[8] K. B. Ferreira, P. Bridges, and R. Brightwell, "Characterizing application sensitivity to os interference using kernel-level noise injection," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, p. 19, IEEE Press, 2008.

[9] T. Hoefler, T. Schneider, and A. Lumsdaine, "Characterizing the influence of system noise on large-scale applications by simulation," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, (Washington, DC, USA), pp. 1–11, IEEE Computer Society, 2010.

[10] H. Akkan, M. Lang, and L. Liebrock, "Understanding and isolating the noise in the linux kernel," *International Journal of High Performance Computing Applications*, p. 1094342013477892, 2013.

[11] J. Howard, S. Dighe, Y. Hoskote, S. Vangal, D. Finan, G. Ruhl, D. Jenkins, H. Wilson, N. Borkar, G. Schrom, *et al.*, "A 48-core ia-32 message-passing processor with dvfs in 45nm cmos," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, pp. 108–109, IEEE, 2010.

[12] J. Jeffers and J. Reinders, *Intel Xeon Phi Coprocessor High-performance Programming*. Newnes, 2013.

[13] M. Dehyadegari, A. Marongiu, M. R. Kakoee, L. Benini, S. Mohammadi, and N. Yazdani, "A tightly-coupled multi-core cluster with shared-memory hw accelerators," in *Embedded Computer Systems (SAMOS), 2012 International Conference on*, pp. 96–103, July 2012.

[14] S. Anthony, "Intel unveils 72-core x86 knights landing cpu for exascale supercomputing," 2013.

[15] P. Burgio, A. Marongiu, D. Heller, C. Chavet, P. Coussy, and L. Benini, "Openmp-based synergistic parallelization and hw acceleration for on-chip shared-memory clusters," in *Digital System Design (DSD), 2012 15th Euromicro Conference on*, pp. 751–758, Sept 2012.

[16] Cavium Inc., "Thunderx arm processors." https://www.cavium.com/ThunderX_ARM_Processors.html, 2015.

[17] Applied Micro Circuits Corp., "X-gene." https://www.apm.com/products/data-center/x-gene-family/x-gene, 2015.

[18] N. Rajovic, P. M. Carpenter, I. Gelado, N. Puzovic, A. Ramirez, and M. Valero, "Supercomputing with commodity cpus: Are mobile socs ready for hpc?," in *High Performance Computing, Networking, Storage and Analysis (SC), 2013 International Conference for*, pp. 1–12, IEEE, 2013.

[19] F. Fraternali, A. Bartolini, C. Cavazzoni, G. Tecchiolli, and L. Benini, "Quantifying the impact of variability on the energy efficiency for a next-generation ultra-green supercomputer," in *Low Power Electronics and Design (ISLPED), 2014 IEEE/ACM International Symposium on*, pp. 295–298, IEEE, 2014.

[20] A. Auweter, A. Bode, M. Brehm, L. Brochard, N. Hammer, H. Huber, R. Panda, F. Thomas, and T. Wilde, "A case study of energy aware scheduling on supermuc," in *Supercomputing*, pp. 394–409, Springer, 2014.

[21] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 8, no. 3, pp. 299–316, 2000.

[22] G. Zellweger, S. Gerber, K. Kourtis, and T. Roscoe, "Decoupling cores, kernels, and operating systems," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pp. 17–31, 2014.

[23] S. Panneerselvam, M. Swift, and N. S. Kim, "Bolt: Faster reconfiguration in operating systems," in *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pp. 511–516, 2015.