Software-defined PMC for Runtime Power Management of a Many-core Neuromorphic Platform

Software-defined PMC for Runtime Power Management of a Many-core Neuromorphic Platform

Indar Sugiarto*, Delong Shang†, Amit Kumar Singh‡, Bassem Ouni§,
Geoff Merrett¶, Bashir Al-Hashimi‖ and Steve Furber**

*†**School of Computer Science, University of Manchester, United Kingdom
‡School of Computer Science and Electronic Engineering, University of Essex, United Kingdom
§¶‖School of Electronics and Computer Science, University of Southampton, United Kingdom
Email: {*indar.sugiarto, †delong.shang, **steve.furber}@manchester.ac.uk,
‡a.k.singh@essex.ac.uk,§Bassem.Ouni@soton.ac.uk, {¶gvm, ‖bmah}@ecs.soton.ac.uk

***Abstract*—This paper presents an approach to provide a Run-time Management (RTM) system for a many-core neuromorphic platform. RTM frameworks are commonly used to achieve an energy saving while satisfying application performance requirements. In commodity processors, the RTM can be implemented by utilizing the output of Performance Monitoring Counters (PMCs) to control the frequency of the processor's clock. However, many neuromorphic platforms such as SpiNNaker do not have PMC units; thus, we propose a software-defined PMC that can be implemented using standard programming tool-chains in such platforms. In this paper, we evaluate several control strategies for RTM in SpiNNaker. These control programs are equivalent with governors in standard operating systems such as Linux. For evaluation, we use the RTM with several image processing applications. The results show that our proposed method, called Improved-Conservative, produces the lowest thermal risk and energy consumption while achieving the same performance as other adaptive governors.**

***Index Terms*—PMC, RTM, many-core, SpiNNaker.**

I. Introduction

Run-time Management (RTM) is a feature commonly found in modern Operating Systems (OS), and its importance becomes prominent in the era of mobile computing [1]. It is a task of the OS that maximises performance whilst trying to maintain the overall reliability for long-term usage. In Unix/Linux based OSes, this RTM can be implemented using a framework called a governor. Even though the concept of the RTM has been developed for some time, it is still being actively explored for multi-core processors [2], [3].

The complexity of an RTM design increases when the system is expanded from a multi-core to a many-core system. In a many-core system, the RTM is required to work seamlessly accross the network of distributed processors. Several models of RTM for many-core systems have been proposed, and to our knowledge, those RTMs rely on the presence of a standard/mainstream OS that provides access to processor's performance monitoring hardware. However, not all many-core systems are equipped with such an OS; in this circumstance, the design of an RTM must be done from scratch, and this paper presents a study of an RTM design for a many-core system without an OS.

As a target platform, a many-core neuromorphic called SpiNNaker is used. SpiNNaker, which stands for Spiking Neural Network Architecture, is a many-core neuromorphic platform developed for simulating a massive spiking neural network (SNN) in biological real time [4]. SpiNNaker is built on a standard low-power ARM processor architecture; hence, it is possible to use the SpiNNaker for general purpose computing beyond SNN simulation [5].

The SpiNNaker system does not use a standard OS commonly found in computers. Instead, it uses a special kernel program known as SARK (SpiNNaker Application Runtime Kernel) that manages the entire operation of an application program running on a SpiNNaker machine. Currently, the SARK does not have any RTM, and in this paper we explore the possibility of deploying several RTM programs and evaluate their performance. The presence of an RTM in SpiNNaker is useful to maintain reliability over long-term operation of a SpiNNaker machine as well as to optimize the performance of application programs running on the machine.

The challenges of developing an RTM in SpiNNaker come not only in the absence of a standard OS, but also from the hardware itself; the SpiNNaker chips do not have any Performance Monitoring Counters (PMCs) that are usually used in an RTM program [6], [7]. Hence, we propose a method which we refer to as software-defined PMC as an alternative to provide performance metrics needed by RTMs. The aims of this paper are as follows.

1) To give readers an understanding of mechanisms for developing and using an RTM for many-core system.
2) To describe methods for developing PMCs in software for SpiNNaker that can be extended for general purpose many-core system.

3) To demonstrate the possiblity of implementing an RTM for SpiNNaker.
4) To provide a measurement baseline for further complex RTM designs in the future.

The structure of this paper is organized as follows. Section II shortly describes the platform and the basic concept of RTM that is relevant to its development for SpiNNaker. Section III describes several PMCs that are defined and used for control algorithms in the RTM framework. Section IV contains the experiment results and also discusses the current state of the implemented RTMs. Finally, the paper is closed with a conclusion in Section V.

II. Background

A. The SpiNNaker System

SpiNNaker is a novel massively parallel computer architecture, inspired by the fundamental structure and function of the human brain, which itself is composed of billions of simple computing elements, communicating using unreliable spikes. It is a power efficient heterogeneous system intended for modeling spiking neurons in real-time. Each SpiNNaker chip comprises of 18 identical ARM968 cores, each with its own local tightly-coupled memory (TCM) for storing data (64KB) and instructions (32KB). All cores have access to a shared off-die 128MB SDRAM through a self-timed system network-on-chip (NoC). In terms of the number of cores, there are several different SpiNNaker machines, including a 4-node board (72 cores), 48-node board (864 cores), and 24-board frame (20,736 cores). The final version of the SpiNNaker machine will contain 1,036,800 cores, which will be hosted in ten 19-inch cabinets. An example of a 48-node board is shown in Fig.1, which is used in this paper.

The communication infrastructure of SpiNNaker relies mainly on small packet protocols. A router is placed at the center of the chip. The router is capable of handling one-to-many communications efficiently, while its novel interconnection fabric allows it to cope with very large numbers of SpiNNaker data packets.

Each chip has 2-phased (phase-locked loop) PLL circuits that can be controlled for providing correct clock frequency. The PLLs provide clock to the following component in SpiNNaker chip: ARM cores, SDRAM, router, and system bus. In this paper, we only modify the clock frequency of ARM cores.

B. RTM for System Reliability

Running a compute-intense application on a multi-core system almost always raises issues. On one side, this type of applications consumes more timing-related resources from the processor, leaving the other applications shorter execution periods and thus reducing the overall performance. On the other side, due to the elevated operating temperature, the lifetime operability of the system is threatened by the acceleration of device wear-out. Also, in the era of mobile and green computing, the requirement for power/energy optimization is increasing. These are the main reasons for the existence of RTM in many modern OSes.

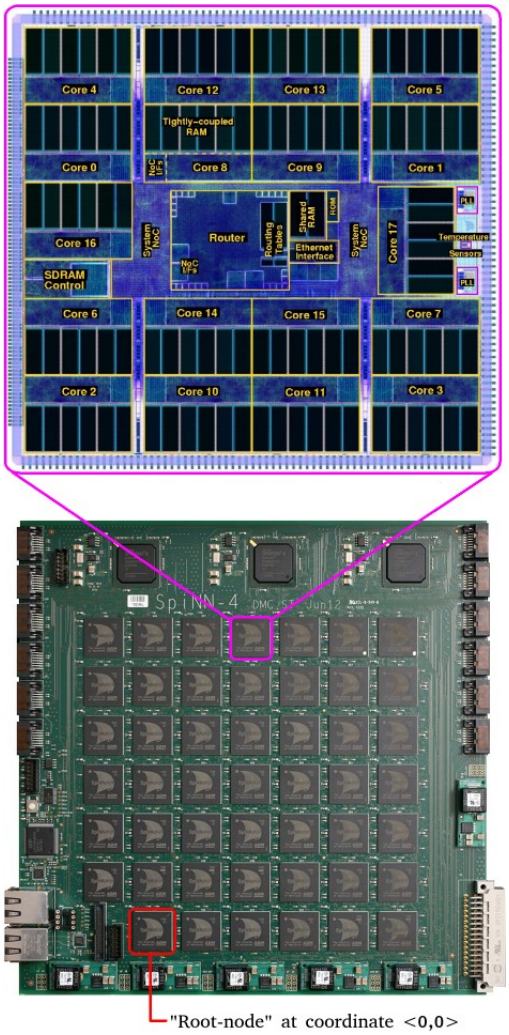

Fig. 1. 48-node SpiNNaker board. Each chip contains 18 ARM968 cores (shown on top figure) and 128MB SDRAM mounted on top of the processor die. A special chip labeled as "root-node" contains an RTM supervisor that coordinates all other RTMs in each chip on the board.

Many RTMs that address both system performance and the thermal awareness have been proposed in the literature. Dynamic Voltage and Frequency Scaling (DVFS) and Dynamic Power Management (DPM) are two hardware techniques for reducing power consumption commonly used in multi-core embedded systems [1], [8]–[10]. Fundamental to the approaches is a run-time system that collects various metrics using components such as on-board thermal sensors, and uses a controller program that maps the relationship between the frequency of processor cores and its temperature. The controller, which is also called governor, aims to control the average temperature and the thermal cycling to achieve an extended uptime of the system (i.e., mean time to failure or MTTF). Through this parameter, the lifetime reliability of the system can be determined as follows [11]:

$$R(t) = e^{-(t \cdot A)^{\beta}}$$

where A is the thermal aging of the system that depends on

the execution time of the application, the average temperature during the execution, and the fault density. The lifetime of the system is thus modeled by integrating *R(t)* overtime:

$$MTTF = \int_0^\infty R(t)dt = \int_0^\infty e^{-(t \cdot A)^\beta} dt$$

Hence, maximizing the MTTF is equivalent to minimizing the aging of the system. It can be done by reducing the thermal stress, which is closely related to the maximum temperature and its variation in thermal cycles.

Regarding the control mechanism, unfortunately the SpiNNaker hardware does not have a dedicated module that provides a fully compliant DVFS mechanism. It does, however, have programmable PLL units inside the chip that can be controlled for providing correct frequency settings. The RTM for SpiNNaker relies on these PLLs, and we have developed a program that emulates performance counters.

C. PMC for RTM

The use of PMCs for RTM is of paramount importance as they can be monitored at regular fine grained intervals, e.g. 100 ms, and the RTM can take decisions based on the monitored values to perform optimization for one or several objectives, such as energy consumption and/or performance [12], [13]. The decision can be mapping of an application to a different set of cores [14] or voltage/frequency levels of the cores [13]. In case of unavailability of PMCs, the design options related to different decisions need to be explored offline for all the possible run-time scenarios, which are non-traceable for dynamic and large scale systems [15].

Since usage of PMCs for RTM has tremendous potential, it has been well exploited. While most of the efforts have exploited PMCs during runtime [7], [14], [16], [17], some have also used them to build profiling information to facilitate efficient RTM [18], [19]. Runtime optimizations have replied on one or several PMCs, such as CPU cycles, L2 cache read refills, total amount of executed instructions, active cycles, L1 and L2 cache misses per instruction, and branch mispredictions per instruction. The values of PMCs at regular intervals provide information to optimize for one or several metrics, e.g. a high value of L2 cache read refills indicates that the program needs data from the main memory and thus the core executing the program can be run at low frequency.

In aforementioned works, the PMCs are made available by the OS to the scheduler/RTM to take appropriate decisions. However, the OS has significant memory and power overheads. Such OS overheads in SpiNNaker are significantly reduced by using SARK, but this imposes challenges to make PMCs available to the RTM.

III. Software-defined PMC for SpiNNaker

In this section, the core algorithm for RTM in SpiNNaker that makes use of software-defined PMCs is presented. One of the main elements of an RTM is the governor: the program that manages the clock frequency of the system. In our work, the three standard governors are implemented and tested: user defined, on-demand, and conservative. In addition to these, we propose a new algorithm to improve to performance of the standard conservative governor.

A. PMC Design

As described in Section II-A, SpiNNaker is a neuromorphic system that is used to mimic brain operation at the neuron level. The main characteristic of such an operation is the massive network of small computational units (i.e., the neurons) with very low power consumption. With this paradigm, the SpiNNaker chip was designed with focus on providing a reliable communication infrastructure. Hence, standard PMCs commonly found in conventional processor systems are not available in SpiNNaker chips. Furthermore, the SpiNNaker chip's core uses ARM968, which does not have any performance monitoring unit (PMU). The SpiNNaker hardware, however, provides PMCs that are directly related with the communication infrastructure, such as the router diagnostic counter, packet delay histogram, etc. The SpiNNaker kernel (SARK) uses these PMCs to maintain the communication reliability, e.g., by managing the emergency routing when a link fails (temporarily, due to congestion, or permanently, due to component failure).

In this paper, we introduce and define the following PMCs: CPU Idle Counter (CIC), DMA Full Counter (DFC), and Thermal Violation Counter (TVC).

The CIC is implemented by reading the CPU sleep status register (register 25 in the System Controller at address 0xe2000064). When a core is in idle state (awaiting an interrupt), it raises a flag in this register. By counting how many times the flag is raised, we can measure the load of the corresponding core. To facilitate counting, we utilize the system-wide slow counter of the SpiNNaker machine that runs at 32kHz.

The DFC is implemented by counting how many times the DMA module is in a full state. DMA is the most important memory access feature of SpiNNaker, since a core in a SpiNNaker chip has a very limited internal memory (32KB for instruction, and 64KB for data). By using DMA, the SpiNNaker core can access the external SDRAM (up to 128MB per chip) at high speed. However, this DMA module is shared among 18 cores in the chip. Hence, the DFC is very important in a multi-core processing access; by providing DFC, an application program in a core can adaptively adjust its performance. The DFC is related to register 5 of the DMA module in a SpiNNaker chip at address 0x40000014. When DMA is full, a core is prohibited to request a DMA access, hence increasing the processing latency of the core.

The TVC is implemented by counting how many times the SpiNNaker chip's temperature is above the predefined threshold value. As described in Section II-B, thermal stress is one important parameter to maintain to achieve longer lifetime of a SpiNNaker chip. For measuring this thermal stress, TVC is developed by utilizing internal temperature sensors of a

TABLE I
THE FOUR PMC-BASED GOVERNORS DEVELOPED FOR SPINNAKER RTM.

Name	Label	Description
User Defined	G1	Static/constant frequency defined by user.
On-demand	G2	Runs at highest frequency when cpu loads are high, and at lowest frequency when cpu loads are low.
Conservative	G3	Increase or decrease frequency at fixed frequency-step interval according to the cpu load levels.
Proposed	G4	Similar to G3, but the step is computed from the half of difference between the current frequency and the highest/lowest frequency.

SpiNNaker chip, which is read periodically using the system-wide slow timer alongside the CIC. In our proposed governor (see section III-B), we used the value of TVC to set the maximum frequency that can be selected during frequency-step calculation in order to avoid thermal violations.

B. Governor Design

Four governors for our RTM are implemented and evaluated. Three of these are standard Linux governors: User-defined, On-demand, and Conservative, whereas our proposed method is an enhancement to the Conservative governor. Table I shows the governors and their symbols used this paper.

The governor program runs exclusively on core-1 in every SpiNNaker chip, and each governor is responsible for managing only the PLLs inside the corresponding chip. A SpiNNaker chip might have a different frequency controlling scheme than the other chips. However, they can also run synchronously. In this synchronous mode, the governor in the chip with coordinate <0,0> on the 48-node board (labeled as "root-node" in Fig.1), behaves as the main governor that coordinates all governors in other chips. This scenario is useful for applications that run on several chips in the SpiNNaker machine. For communication among governors, the SpiNNaker Datagram Protocol (SDP) is utilized. Even though SDP is a slow mechanism, it can contain a larger payload than any other communication protocol available in SpiNNaker [20].

1) User-defined Governor: When this governor is selected, a fixed frequency defined by the user is applied to SpiNNaker cores in a chip. The minimum and maximum clock frequency for SpiNNaker cores selectable by the user are 10MHz and 255MHz respectively. The frequency can be incremented or decremented at 1MHz step. For our experiments, we set the User-defined frequency at 200MHz, which is the normal operation of SpiNNaker for spiking neural network applications.

2) On-demand Governor: In this paper, we developed the on-demand governor as follows. By using CIC, we defined a THRESHOLD utilization value. During application run-time, the CIC will increase and/or decrease dynamically. When the CIC value is higher than the THRESHOLD value, the clock frequency is set to the maximum (255MHz). Otherwise, it decreases the clock frequency at a fixed-step of 50MHz. When it reaches a frequency smaller than 100MHz, the frequency will be set at the minimum value (100MHz). This differs slightly from the standard implementation of the On-demand governor in the Linux kernel [21], where the decreasing-step is set to be 20% of the current frequency.

3) Conservative Governor: The difference between the conservative and the on-demand governor is in the mechanism of increasing and decreasing the frequency. In the conservative governor, the frequency is gracefully increased and/or decreased rather than jumping to the maximum value. In our work, the conservative governor is implemented in the same way as on-demand. By using the value from CIC, the clock frequency is increased or decreased by 5% until it reaches the maximum or minimum frequency respectively.

4) Improved Conservative Governor: The conservative governor described above only takes consideration of CPU load from CIC. In order to make it thermal aware whilst maintaining high responsiveness, we propose an improved version of the conservative algorithm (see Algorithm 1). Here, we include information provided by the TVC to control the maximum clock frequency that can be selected by the algorithm. We defined a TVC_THRESHOLD value that limits the number of violation of the maximum thermal heat when running a program. Everytime the TVC_THRESHOLD is reached, the maximum frequency that the governor can choose is reduced by 5MHz. However, when the TVC_THRESHOLD is not reached again after a specified period, then the maximum frequency can be increased again by factor of 5MHz. Another improvement we added in our proposed governor is the scaling factor for the increment and decrement steps. The original conservative governor uses a fixed size step; therefore, it is relatively slow to respond. Our algorithm, on the other hand, uses successive approximation where a step-size of the half between the maximum (or minimum) frequency and the current frequency is used. This makes the proposed governor more responsive without going into the extreme condition as experienced by the on-demand governor.

IV. EXPERIMENTAL SETUP AND EVALUATION

To evaluate the performance of our proposed governor, we developed three non-SNN applications and run the governors alongside the applications. We use non-SNN applications to demonstrate that our methods are applicable for general purpose applications even though they are implemented on a neuromorphic platform. Those applications are: JPEG image encoding (A1), JPEG image decoding (A2), and edge detection (A3). Application A3 has been used in our previous research to demonstrate graceful degradation and amelioration concept on SpiNNaker [5].

Application A1 and A2 are the first applications that are developed by considering the impact of DFC for performance improvement. Both applications retrieve/store data from/to SDRAM using DMA. Hence, it is crucial to detect the level of the DMA buffer before they request a direct memory access. Otherwise, the SpiNNaker core might be trapped in a live-lock waiting for a slot. This is a new mechanism introduced in our applications, whereas the application A3 still uses the

Algorithm 1 Improved Conservative Governor

```
for every thermal tick do
  if TVC < TVC_THRESHOLD and maxFreq < 255 then
    maxFreq += 5
  end if
end for
for every governing tick do
  if TVC > TVC_THRESHOLD then
    maxFreq -= 5
  end if
  get cputil {utilization since last check}
  get cf {current frequency}
  if cputil < CPU_THRESHOLD then
    step = (maxFreq - cf)/2
    if cf < maxFreq then
      cf += step
    else
      cf = maxFreq
    end if
  else
    step = (cf - minFreq)/2
    if cf > minFreq then
      cf -= step
    else
      cf = minFreq
    end if
  end if
end for
return cf
```

TABLE II
TIMING MEASUREMENT RESULT (IN MILLISECONDS).

App.	Res.	Governor G1	G2	G3	G4
A1	vga	955	976	976	975
	svga	1490	1522	1522	1523
	xga	2444	2498	2498	2498
A2	vga	2670	3080	3080	3080
	svga	4408	4737	4737	4737
	xga	7114	7342	7342	7342
A3	vga	437	454	454	451
	svga	674	696	696	697
	xga	1111	1150	1150	1150

old mechanism, in which a master core is assigned with a task for coordinating DMA among cores in a SpiNNaker chip.

We run A1, A2, and A3 alternately whilst changing the governor. During application execution, we measure the energy consumption and temperature of SpiNNaker chips using a SpiNNaker profiler program [22]. Table II shows the execution time for each application controlled by each governor. Table III and Table IV show the measured energy consumption and the temperature fluctuation when SpiNNaker runs the program and the governor respectively.

From Table II and Table III, one can see the close relationship between task execution time and the consumed power. In general, governor G1 runs faster than the other governors; however, it also consumes higher power than the others.

TABLE III
ENERGY CONSUMPTION (IN JOULE).

App.	Res.	Governor G1	G2	G3	G4
A1	vga	2.76	1.98	2.11	2.27
	svga	6.40	5.06	5.05	5.12
	xga	17.79	13.74	13.82	13.74
A2	vga	8.24	6.84	7.16	7.06
	svga	22.20	16.46	17.02	16.13
	xga	58.72	39.29	40.95	39.29
A3	vga	9.41	7.16	7.17	6.62
	svga	17.07	13.01	13.08	11.90
	xga	48.30	37.19	36.87	33.92

TABLE IV
TEMPERATURE INCREASE (IN DEGREE CELCIUS) DURING A PROGRAM EXECUTION.

App.	Res.	Governor G1	G2	G3	G4
A1	vga	0.19	0.72	0.75	0.71
	svga	0.3	0.95	0.93	0.87
	xga	0.22	1.08	1.05	1.05
A2	vga	0.36	1.39	1.21	1.3
	svga	0.34	1.39	1.34	1.43
	xga	0.5	1.72	1.66	1.55
A3	vga	1.07	1.2	1.08	1.15
	svga	1.2	1.36	1.34	1.27
	xga	1.34	1.54	1.52	1.41

Table IV shows that, in general, governor G1 produces a lower temperature variation than the other governors. It does not mean that G1 works better than the others in terms of heat production, because the SpiNNaker chip already has high temperature when running G1. SpiNNaker has been running at a high constant frequency of 200MHz, even when there is no user application loaded into the chip. On the other hand, governors G2, G3, and G4 run at low frequency of 100MHz when there is no running user application. Fig.2 shows the behavior of those governors with respect to this temperature anomaly. It also shows that our proposed algorithm works better than its original version; it produces lower temperature variation.

V. CONCLUSION

A basic run-time management framework for SpiNNaker has been developed, and RTM evaluation is presented in this paper. The RTM has several governors that control the clock frequency of SpiNNaker cores. The governors utilize performance monitoring counters (PMCs) developed on software using the existing registers and sensors inside the SpiNNaker chip. The performance of those governors with regard to application speed impact, energy consumption, and thermal dissipation are evaluated by running the governors alongside three non-SNN applications. From the experiment, we observe that setting the clock at fix and relatively high frequency makes the program run faster but with higher thermal risk and energy

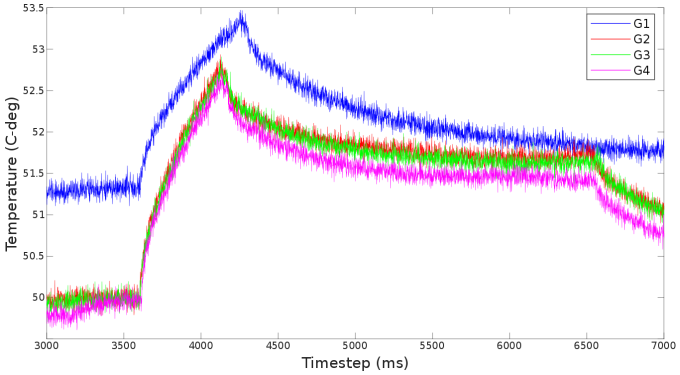

Fig. 2. Temperature measurement when running application A3 showing similar pattern in G2, G3, and G4, but distinguishable from G1. This shows that with G1 manages the system operation, it already produces higher termal dissipation even when there is no user application running on the machine.

consumption. On the other hand, applications that are under supervision of the three other adaptive governor, namely On-demand, Conservative, and Improved-Conservative, run a bit slower but with lower thermal risk and energy consumption. Especially for the Improved-Conservative, which implements our proposed method, the thermal risk and energy consumption are at the lowest while impacting the same speed comparing to the other governors.

ACKNOWLEDGMENT

This work was supported primarily by the Engineering and Physical Sciences Research Council (EPSRC) through the Graceful project (grant EP/L000563/1) and the PRiME project (grant EP/K034448/1). The design and construction of the SpiNNaker machine was also supported by EPSRC under grants EP/D07908X/1 and EP/G015740/1. Ongoing development of the software is supported by the EU ICT Flagship Human Brain Project (FP7-604102). Steve Furber receives support from the European Research Council under the European Unions Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement 320689. The dataset of this paper can be accessed from https://data.mendeley.com/datasets/37wctb99rw/draft?a=b43c5f80-4ada-45b2-a57a-66b3b397435c

REFERENCES

[1] K. Bhatti, C. Belleudy, and M. Auguin, "Power management in real time embedded systems through online and adaptive interplay of dpm and dvfs policies," in *2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, Dec 2010, pp. 184–191.

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

[3] H. Sasaki, S. Imamura, and K. Inoue, "Coordinated power-performance optimization in manycores," in *Proceedings of the International conference on Parallel architectures and compilation techniques*. IEEE, 2013, pp. 51–61.

[4] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The spinnaker project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, May 2014.

[5] I. Sugiarto, G. Liu, S. Davidson, L. A. Plana, and S. B. Furber, "High performance computing on spinnaker neuromorphic platform: A case study for energy efficient image processing," in *2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC)*, Dec 2016, pp. 1–8.

[6] L. C. Singleton, C. Poellabauer, and K. Schwan, "Monitoring of cache miss rates for accurate dynamic voltage and frequency scaling," in *Electronic Imaging*. International Society for Optics and Photonics, 2005, pp. 121–125.

[7] B. K. Reddy, A. K. Singh, G. V. Merrett, and B. M. Al-Hashimi, "Itmd: Run-time management of concurrent multi-threaded applications on heterogeneous multi-cores," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2017.

[8] C. Isci, G. Contreras, and M. Martonosi, "Live, runtime phase monitoring and prediction on real systems with application to dynamic power management," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 39. Washington, DC, USA: IEEE Computer Society, 2006, pp. 359–370.

[9] A. Das, A. Kumar, B. Veeravalli, C. Bolchini, and A. Miele, "Combined dvfs and mapping exploration for lifetime and soft-error susceptibility improvement in mpsocs," in *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2014, pp. 1–6.

[10] I. Yeo, C. C. Liu, and E. J. Kim, "Predictive dynamic thermal management for multicore systems," in *Proceedings of the 45th Annual Design Automation Conference*, ser. DAC '08. New York, NY, USA: ACM, 2008, pp. 734–739. [Online]. Available: http://doi.acm.org/10.1145/1391469.1391658

[11] A. Das, R. A. Shafik, G. V. Merrett, B. M. Al-Hashimi, A. Kumar, and B. Veeravalli, "Reinforcement learning-based inter- and intra-application thermal optimization for lifetime improvement of multicore systems," in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2014, pp. 1–6.

[12] A. Weissel and F. Bellosa, "Process cruise control: event-driven clock scaling for dynamic power management," in *Proc. of international conference on Compilers, architecture, and synthesis for embedded systems*. ACM, 2002, pp. 238–246.

[13] A. K. Singh, K. R. Basireddy, B. Al-Hashimi, G. Merrett *et al.*, "Learning-based run-time power and energy management of multi/many-core systems: current and future trends," *Journal of Low Power Electronics*, 2017.

[14] J. Ma, G. Yan, Y. Han, and X. Li, "An analytical framework for estimating scale-out and scale-up power efficiency of heterogeneous manycores," *IEEE Transactions on Computers*, vol. 65, no. 2, pp. 367–381, 2016.

[15] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: survey of current and emerging trends," in *Proceedings of the 50th Annual Design Automation Conference*. ACM, 2013, pp. 1–10.

[16] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer, "Scheduling heterogeneous multi-cores through performance impact estimation (pie)," in *ACM SIGARCH Computer Architecture News*, vol. 40, no. 3. IEEE Computer Society, 2012, pp. 213–224.

[17] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda, "Pack & cap: adaptive dvfs and thread packing under power caps," in *Proceedings of the 44th annual IEEE/ACM international symposium on microarchitecture*. ACM, 2011, pp. 175–185.

[18] B. Donyanavard, T. Mück, S. Sarma, and N. Dutt, "Sparta: run-time task allocation for energy efficient heterogeneous many-cores," in *Proceedings of the IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. ACM, 2016, p. 27.

[19] A. Aalsaud, R. Shafik, A. Rafiev, F. Xia, S. Yang, and A. Yakovlev, "Power–aware performance adaptation of concurrent applications in heterogeneous many-core systems," in *Proceedings of the International Symposium on Low Power Electronics and Design*. ACM, 2016, pp. 368–373.

[20] APT Group, *SpiNNaker datasheet version 2.03*, School of Computer Science, The University of Manchester, Kilburn Building, Oxford Road, Manchester, M13 9PL, United Kingdom, Jan 2015.

[21] V. Pallipadi and A. Starikovskiy, "The ondemand governor," in *Proc. of the Linux Symposium*, vol. 2. sn, 2006, pp. 215–230.

[22] I. Sugiarto, L. Plana, S. Temple, B. Sen Bhattacharya, S. Furber, and P. Camilleri, "Profiling a many-core neuromorphic platform," in *2017 11th IEEE International Conference on Application of Information and Communication Technologies (AICT2017)*, Sept 2017, pp. 1–6.