Scalable Probabilistic Power Budgeting for Many-Cores

[*]Anuj Pathania, [*]Heba Khdr, [+]Muhammad Shafique, [†]Tulika Mitra, [*]Jörg Henkel
[*]Chair of Embedded System (CES), Karlsruhe Institute of Technology, Germany
[+]Institute of Computer Engineering, Vienna University of Technology (TU Wien), Austria
[†]School of Computing (SoC), National University of Singapore, Singapore
Corresponding Author: anuj.pathania@kit.edu

***Abstract*—Many-core processors exhibit hundreds to thousands of cores, which can execute lots of multi-threaded tasks in parallel. Restrictive power dissipation capacity of a many-core prevents all its executing tasks from operating at their peak performance together. Furthermore, the ability of a task to exploit part of the power budget allocated to it depends upon its current execution phase. This mandates careful rationing of the power budget amongst the tasks for full exploitation of the many-core.**

Past research proposed power budgeting techniques that redistribute power budget amongst tasks based on up-to-date information about their current phases. This phase information needs to be constantly propagated throughout the system and processed, inhibiting scalability. In this work, we propose a novel probabilistic technique for power budgeting which requires no exchange of phase information yet provides mathematical guarantees on judicial use of the TDP. The proposed probabilistic technique reduces the power budgeting overheads by 97.13% in comparison to a non-probabilistic approach, while providing almost equal performance on simulated thousand-core system.

I. Introduction

Many-cores are upcoming hundred-/thousand-core processors capable of executing scads of multi-thread tasks in parallel [1]. Unfortunately, the limited power dissipation capacity of the many-core requires adherence to a strict power budget called Thermal Design Power (TDP) determined by the manufacturer [2]. TDP restricts the many-core from executing all its tasks at peak performance simultaneously. Continuous operation at a power consumption beyond TDP is not recommended as it may cause serious damage to the processor.

Furthermore, executing tasks go through various execution phases during their lifetime that determines how well they can exploit the part of the TDP allocated to them [3]. Therefore, it is necessary to ensure proper rationing of TDP amongst the tasks. Governors are the Operating System (OS) sub-routines responsible for judicious and safe use of TDP.

Dynamic Voltage and Frequency Scaling (DVFS) is the knob available to the Governors for performing phase-aware power budgeting between the tasks [4]. DVFS allows different cores of the many-core to operate at different frequencies and voltages. When operating at a higher DVFS level, processing cores execute threads of an assigned task faster provided the task is in a processing intensive phase but at the cost of more power consumption. Increasing core frequency using DVFS when a task is in memory intensive phase result in no performance gain and only leads to power wastage.

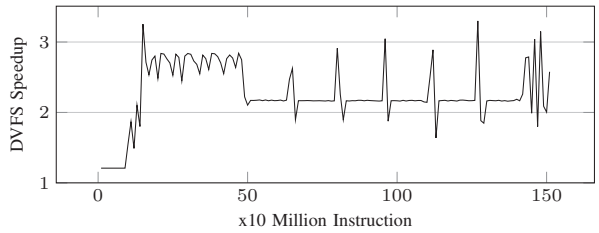


Fig. 1: Execution profile of *bodytrack* (single-threaded) benchmark showing DVFS speedup variation during execution.

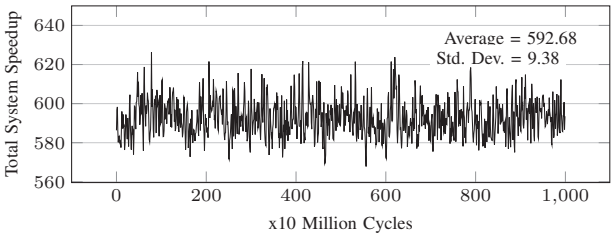


Fig. 2: Total system speedup over time when 256 tasks (1024 threads) are using higher DVFS level on a 1024-core chip.

Speedup is the metric used for measuring performance gain obtained from DVFS. Speedup is the ratio of Instruction per Second (IPS) of a task when its assigned cores are operating at a higher DVFS level to its IPS when they are operating at a lower DVFS level. Figure 1 shows how speedup of a single-threaded version of *bodytrack* benchmark changes as it goes through different phases of its execution. In a many-core restricted by TDP, a task should operate at a higher DVFS level only when it can derive considerable speedup from the level as it may deprive other tasks operating in parallel from raising their performance.

Power budgeting under a TDP in multi/many-core is a well-studied problem. However, all previously proposed Governors are non-probabilistic and take power budgeting decisions based upon phase information collected from tasks all over the many-core requiring substantial communication. We make an argument in this work that constant monitoring of task phases is not required when large number of independent tasks are running on the many-core as the total speedup that can be obtained is autonomously self-stabilizing. This can be seen Figure 2 where instantaneous total system speedup of

978-3-9815370-8-6/17/$31.00 ©2017 IEEE

256 tasks (1024 threads) running using higher DVFS level on 1024-core many-core has a very low standard deviation from the average. Reason being that even though locally all tasks are going through different execution phases, there is no synchronization among their phases. While some tasks transition from low- to high speedup phase at any given time, a near-equal number of tasks perform a reverse transition; resulting in a stable global behavior. This behavior can be exploited by a probabilistic power budgeting governor for many-cores to provide near-equal performance in comparison to non-probabilistic Governors but with significantly lower computational and communication overheads.

Our Novel Contribution: In this work, we introduce an alternative probabilistic Governor called *ProGov* for power budgeting under TDP on many-cores. Our alternative approach for power budgeting has potential to significantly reduce the associated overheads. Further, mathematical foundations of *ProGov* also allows for concrete guarantees on TDP violations.

Operations of *ProGov* is quite different from a non-probabilistic Governor. While a non-probabilistic Governor uses dynamic phase information obtained online to make decisions, *ProGov* uses static probabilistic phase profiles collected offline to make similar decisions. The decisions made by a non-probabilistic governor constantly change as executing tasks go through different phases. On the other hand, decisions made by *ProGov* change only when executing task population changes its constitution by arrival or departure of a task. A non-probabilistic Governor is well-suited when number of tasks is small whereas probabilistic Governor works well when number of tasks is large. In fact, due to the law of large numbers [5] results provided by *ProGov* become more accurate as number of executing tasks increase. Therefore, *ProGov* is particularly well suited for many-core paradigm.

Limitations: Results guaranteed by *ProGov* are also probabilistic in nature. While a non-probabilistic governor can always guarantee non-violation of TDP while extracting high performance, *ProGov* only provides a high probability that many-core will operate similarly for any given population of executing tasks. It is important to note that for a sufficiently loaded many-core the probability of TDP violation is always non-zero under *ProGov*, if DVFS is to be used for boosting performance substantially. Hence, *ProGov* is also not suitable for hard real-time or mission-critical systems.

II. Related Work

Power has always been a prime design constraint in processors [6]. Use of power budgeting Governors for keeping a processor safely operating close to TDP has been well-studied for many-cores. Still, the continuous trend of adding more cores to processors [1] warrants more scalable techniques for power budgeting compared to ones already proposed.

Centralized bounded state-search power budgeting techniques for multi-cores such as *MaxBIPS*, proposed in [7] are too slow when applied on many-cores. Authors in [8] proposed multiple light-weight power budgeting heuristics for many-cores amongst which a greedy algorithm called *SortedWS* provided high performance with low overhead. Orthogonally, authors in [9] proposed distributed approach for power budgeting for improved scalability. Distributed Governors reduce the per-core processing overhead by disbursing the calculations over multiple cores but this reduction comes at a cost of increased communication overhead. Still, all the previous proposed techniques remain non-probabilistic in nature inherently limiting their scalability.

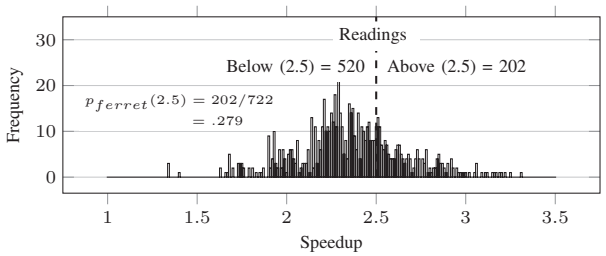


Fig. 3: Histogram of DVFS speedup in *ferret* (single-threaded) benchmark along with calculation of $p_{ferret}(2.5)$.

To best of our knowledge, probabilistic power budgeting remains unexplored vis-a-vis multi/many-cores. Reason being on multi-cores (small number of cores) scalability gains from a probabilistic Governor were insignificant in comparison to non-probabilistic Governors and on many-cores they are yet to be applied. In other domains, probabilistic models for power budgeting have been applied to solve large size problems in datacenters [10] and wireless sensor networks [11].

III. Power Budgeting with *ProGov*

System Overview: We begin with introduction of the notation used in this work to represent a system with a many-core processor. We assume N independent multi-threaded tasks are running in the system, indexed using symbol i. Tasks are assumed to be *rigid*, which means they do not support thread migration once they start execution [12]. Tasks execute using one thread per-core model well-suited for many-cores [13]. In this introductory work, we assume the cores of the many-core have only two DVFS level namely *Low* and *High*. This limits the ability of this work to target energy-efficient execution on many-cores where multiple DVFS level are available. We also assume per-core DVFS similar to *Intel SCC* [14].

Each Task i is assigned a strategy S_i by *ProGov*, which represents a speedup threshold. Task i chooses to operate at *Low*- or *High* DVFS level depending upon whether the instantaneous speedup is above or below S_i, respectively. It is important to note that once S_i is assigned to the Task i by *ProGov* there is no further communication between *ProGov* and Task i. Decision to boost performance using DVFS is taken by Task i independently and locally based on expected speedup boost. Expected instantaneous speedup boost can be calculated using locally available- profiles or DVFS performance prediction models [15]. We also assume that by design all cores assigned to a task always operate at same DVFS level. Let $S = \cup_{i=1}^{N} S_i$ represent the combined strategic profile of all executing tasks determined by *ProGov*.

Probabilistic Performance Model: Given an S_i for a Task i, let $p_i(S_i)$ be the probability that Task i is using a *High* DVFS level. Probability $p_i(S_i) \in [0,1]$ represents the fraction of time Task i under its execution exhibits speedup higher than S_i. Figure 3 shows histogram of DVFS speedup corresponding to single-threaded version of *ferret* benchmark and also shows as a numerical example how $p_{ferret}(2.5)$ is calculated.

$p_i(S_i)$ is a monotonically non-increasing function of speedup threshold S_i because as we increase the S_i a lesser or equal fraction of Task i would remain above the threshold. We obtain this information for discrete values of S_i for each unique task type and store it in a lookup table available with *ProGov*. The granularity of speedup data discretization is .01.

While executing in parallel with a given system strategy profile S, each Task i acts as an independent Bernoulli trial which uses *High* DVFS level with probability $p_i(S_i)$ and uses *Low* DVFS level with probability $1 - p_i(S_i)$. Therefore, our system exhibits Poisson binomial distribution and probability that $K \leq N$ tasks will be in *High* DVFS level simultaneously is given by following Probability Mass Function (PMF) [16].

$$Pr(K) = \sum_{A \in F_K} \prod_{i \in A} p_i(S_i) \prod_{j \in A^C} (1 - p_j(S_j)) \tag{1}$$

where F_K is set of combinations of K tasks selected from N tasks. A^C is complement set of A.

Probabilistic Performance Metric: Scheduling epoch is the time granularity at which Governors operate in an OS. We set it to 10 ms; same as the default *Linux* Governors [17]. We choose the maximum number of tasks in a scheduling epoch that can accelerate without violation of TDP as the metric for optimization. This metric has positive correlation with standard non-probabilistic performance metrics like throughput. Therefore, we want to optimize Equation (1) to peak at the highest value for K feasible under the TDP.

Binomial Simplification: Even obtaining a PMF distribution for any given S using Equation (1) has $O(n!)$ complexity, making direct optimization computationally infeasible when $N >> 1$. To make problem tractable, we propose a simplification that converts the Poisson binomial distribution into a binomial distribution; a well-studied and much more mathematically tractable discrete probability distribution.

By design instead of choosing a unique S_i and thereby a unique $p_i(S_i)$ for every Task i, *ProGov* instead selects a single global *High* DVFS level probability p. Each Task i is then assigned a strategy S_i such that $p_i(S_i) \approxeq p$. *This also introduces fairness into the system because each task has now equal probability of using DVFS to boost its performance*. On the other hand, in a Quality of Service (QoS) based system this simplification can result in wastage of energy as the heterogeneity in speedup behaviors then remains unexploited.

Under the above simplification, each task executing in parallel acts as an independent Bernoulli trial which uses *High* DVFS level with probability p and uses *Low* DVFS level with probability $1 - p$. Our system now exhibits binomial distribution and probability that $K \leq N$ tasks will be using *High* DVFS level together is given by the following PMF.

$$Pr(K) = \binom{N}{K} p^K (1-p)^{N-K} \tag{2}$$

We aim to maximize Equation (2) with respect to a given K using p. Since natural logarithm is a positive function, maximizing log of Equation (2) is same as maximizing the equation itself. By taking log of Equation (2) we get

$$\log(Pr(K)) = \log \binom{N}{K} + K \log(p) + (N-K) \log(1-p)$$

Derivating with respect to p and equating to zero we get

$$\frac{K}{p} - \frac{(N-K)}{1-p} = 0 \implies p = \frac{K}{N} \tag{3}$$

Therefore, if we know the target K then we can use Equation (3) to find a p which maximizes probability of many-core boosting K tasks using DVFS in a given scheduling epoch. The strategy S can be then determined by *ProGov* for a target p using probabilistic speedup profiles.

The mean (μ) and standard deviation (σ) of the targeted binomial distribution is given by following equations.

$$\mu = \sum_{K=1}^{N} K Pr(K) = Np \tag{4}$$

$$\sigma = \sqrt{\sum_{K=1}^{N} K^2 Pr(K) - \mu^2} = \sqrt{Np(1-p)} \tag{5}$$

Probabilistic Power Consumption Model: We now attempt to predict the probabilistic power consumption of the system for a given high DVFS level probability p. For each Task i, let W_i^L and W_i^H be its expected power consumption in Low- and High DVFS level, respectively. This include both static and dynamic power consumption of the task at those levels. Let W^L and W^H be the average power consumption of all tasks in the system at Low- and High DVFS level, respectively. Assuming an additive power consumption of tasks, quick rough estimate for their values at run-time can be made using the following equations.

$$W^L = \frac{\sum_{i=1}^{N} W_i^L}{N} \qquad W^H = \frac{\sum_{i=1}^{N} W_i^H}{N} \tag{6}$$

Since the number of tasks that boost up using DVFS follow a binomial distribution, system's power consumption will also thereby exhibit a equivalent normal distribution [18]. Let $W(x)$ represent the normal distribution of the power consumption given by the following equation.

$$W(x) = \frac{1}{\sqrt{2(\sigma_W)^2 \pi}} e^{-\frac{(x-\mu_W)^2}{2(\sigma_W)^2}} \tag{7}$$

where μ_W and σ_W represent the mean and standard deviation of the normal distribution.

To obtain the total power consumption distribution we reason that at for any given value of p, Np number of tasks are expected to use *High* DVFS level consuming W^H power

each. Similarly, we can expect $N(1-p)$ number of tasks to use *Low* DVFS level consuming W^L power each. Therefore, following relation between K and x holds.

$$\forall_{K\in[0,N]}\exists x = W_H K + W_L(N-K) \mid Pr(K) = W(x) \quad (8)$$

Using above equation, we can derive the following equations for μ_W and σ_W.

$$\mu_W = N(1-p)W^L + NpW^H \quad (9)$$

$$\sigma_W = \sqrt{Np(1-p)}(W^H - W^L) \quad (10)$$

The full proofs of Equation (9) and Equation (10) from Equation (4) and Equation (5), respectively using Equation (8) are extensive and not shown here due to space limitations.

Assumption of normality in distribution is a strong assumption but is necessary to provide a formal mathematical analysis. Fortunately, the error introduced by this assumption become less severe as the number of independently executing tasks on many-core increase due to central limit theorem [5]. Central limit theorem applied in our context states that distribution of the arithmetic sum of power consumption of independently executing tasks approaches a normal distribution as the number of tasks approaches infinity irrespective of power consumption distribution of individual tasks. Therefore, *ProGov* feasibility and accuracy increases with the size of the problem making it especially suitable for many-cores.

It is important to note that if tasks are not executing independently or the tasks in a workload are not diverse enough then system would not exhibit any normal distribution. For system with interdependent tasks, covariance between them is not insignificant and needs to be consider using a much more complex probability model than presented here.

Probabilistic TDP Model: Provided system is not severely under loaded, the probability of TDP violation under *ProGov* can be non-zero if DVFS has to be used aggressively. Therefore, it is important to quantize the risk the many-core is taking for a given *High* DVFS level probability p and a given task population N. Let $\hat{W}$ symbolically represent the TDP of the many-core. The probability that many-core will stay within the TDP $\hat{W}$ is given using cumulative distribution function $F(\hat{W})$ of normal distribution defined by the following equation.

$$F(\hat{W}) = \int_0^{\hat{W}} W(x)dx = \int_0^{\hat{W}} \frac{1}{\sqrt{2(\sigma_W)^2\pi}} e^{-\frac{(x-\mu_W)^2}{2(\sigma_W)^2}} dx \quad (11)$$

No closed form solution exists for calculation of $F(\hat{W})$. However, it can be numerically approximated with high accuracy using Chebyshev fitting [19] in constant time. The probability that TDP is violated is given by Q-function symbolically represented by $Q\ (\hat{W})$ and is given by the following equation.

$$Q\ (\hat{W}) = 1 - F(\hat{W}) = 1 - \int_0^{\hat{W}} \frac{1}{\sqrt{2(\sigma_W)^2\pi}} e^{-\frac{(x-\mu_W)}{2(\sigma_W)^2}} dx \quad (12)$$

Power Budgeting: Based on mathematical foundation laid above, we now present the power budgeting algorithm used by *ProGov*. Unlike a non-probabilistic Governor, *ProGov* cannot guarantee that TDP is never violated. However, it provides formal guarantees on the risk of TDP violations. Let TDP threshold $\hat{Q}$ represent the fraction of TDP violating scheduling epochs, a system designer is willing to tolerate. Accordingly, expected number of tasks to boost using DVFS K needs to be determined based on current task population N. Value of K should be as high as possible for performance.

Algorithm 1 Proposed probabilistic power budgeting technique named *ProGov*.

Input: $N, \hat{W}, \hat{Q}$;
Output: S;
1: $\forall i \in N$ read profiled (or estimate) W_i^L and W_i^H;
2: Calculate W^L and W^H using Equations (6)
3: **for** $K = 1$ to N **do**
4: Calculate p for a given K using Equation (3);
5: Calculate $W(x)$ using Equation (7);
6: Calculate $F(\hat{W})$ using Equation (11);
7: Calculate $Q(\hat{W})$ using Equation (12);
8: **if** $Q(\hat{W}) \geq \hat{Q}$ **then**
9: break;
10: **end if**
11: **end for**
12: Calculate p for $K-1$
13: $\forall i \in N$ set S_i such that $p_i(S_i) \cong p$;
14: return S;

For a given task population N, $K \propto p$ based on Equation (3). Therefore, maximizing *High* DVFS level probability p is same as maximizing K. It is a common observation that all tasks consume less or equal power while using *Low* DVFS level than when using *High* DVFS level. Hence, for any given task population N we know $W^H \geq W^L$. Based on this knowledge and Equation (9), we can state $\mu_W \propto p$.

Being a normal distribution, $W(x)$ is unimodal with peak around μ_W. Since TDP $\hat{W}$ is fixed, increase in μ_W will push more cumulative distribution beyond $\hat{W}$. Therefore, we can conclude $Q\ (\hat{W})$ is monotonically non-decreasing with μ_W. Based on transitivity of above proportionality argumentation, risk of TDP violation $Q\ (\hat{W})$ is monotonically non-decreasing with task boost target K.

Determination of optimum value of K by *ProGov* is thereby simplified to a search in discretized domain $K \in [0, N]$ such that $Q\ (\hat{W}) \cong \hat{Q}$. Algorithm 1 summarizes the technique used in *ProGov* using a simple linear search. Since $Q\ (\hat{W})$ is inherently sorted with value of K, for improved efficiency a binary search can also be used.

Computational Complexity: Since *ProGov* under any step requires not more than one iteration over the executing tasks, it has a linear time computational complexity of $O(N)$. Use of centralized lookup tables leads to a communication complexity of $O(1)$ for *ProGov*. In comparison, a probabilistic greedy Governor *SortedWs* [8] has a computation complexity of $O(N \lg N)$ and communication complexity of $O(N)$. Both techniques will have a $O(N)$ space complexity. Furthermore, the greedy algorithm needs to be invoked in every scheduling epoch to operate. On the other hand, *ProGov* is executed only when a task arrives or leaves the system.

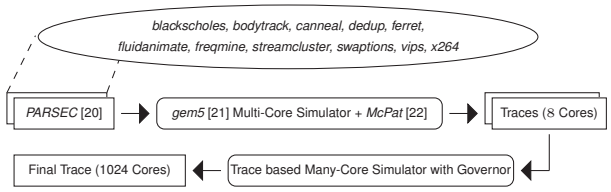


Fig. 4: Experimental Setup.

IV. Experimental Evaluation

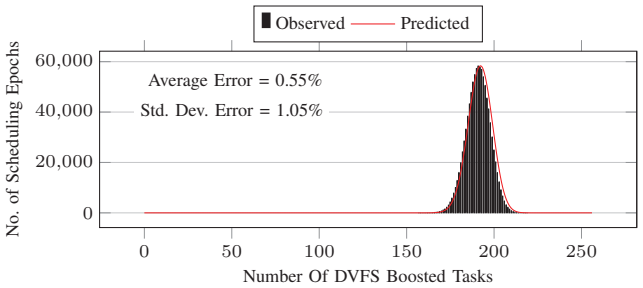
We use a two-stage simulator for empirical evaluation of *ProGov* as shown in Figure 4. In stage one, we use *gem5* [21] cycle-accurate simulator bridged with *McPat* [22] power simulator. Due to limitations on execution time, cycle-accurate simulations are limited to maximum eight cores. Each core uses *Alpha* ISA and holds a 16 KB L1 data- and instruction cache, along with a 32 KB private L2 cache. Cores can run at two DVFS frequencies 1 GHz and 3 GHz representing *Low* and *High* DVFS level, respectively. Unused cores are aggressively power-gated. We believe at 22 nm low-power in-order cores with small caches are the most representative cores for a real-world thousand core chip.

Cycle-accurate isolated execution traces of tasks with up to eight cores allocated from stage one is piped into a trace-based simulator in stage two. Stage two simulator then combined traces together for a final many-core trace with up to a thousand cores assuming a 2D mesh Network on Chip (NoC) between the cores. NoC has a concentration of 1 router per-core and operates at 1 GHz using 256-bits flit. NoC links have bandwidth of 1 flit per cycle and latency of 4 cycle per hop. Program logic for Governors operating at granularity of 10 ms is also implemented in this simulator.

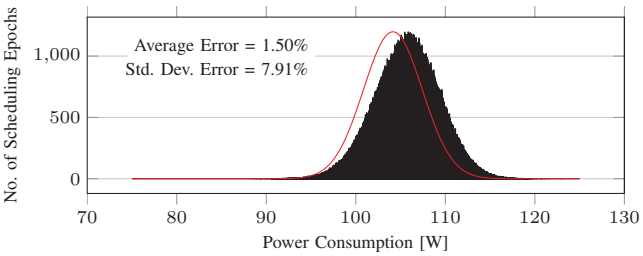
Evaluation of *ProGov* requires large number of scheduling epochs to be observed. Even though our trace-based simulator lacks the sophistication of cycle-accurate simulators, we believe our setup is the only thousand-core many-core setup fast enough to evaluate a probabilistic Governor. For each experiment, around 3 hours of system time is simulated. Systems simulated are closed [12], which means all tasks in the system immediately start execution once they finish. To simulate independent task execution in a closed system each task is initiated with a random instructional skew.

For software, eleven multi-threaded benchmarks as enumerated in Figure 4 from *PARSEC* benchmark suite [20] are used as tasks. Multi-program workloads are formed from random composition of these tasks with each task randomly spawning between one to eight threads. The benchmarks are run in Full System (FS) mode of *gem5* with *sim-small* input. Out of thirteen *PARSEC* benchmarks, only two benchmarks *facesim* and *raytrace* were not used due to lack of *sim-small* input.

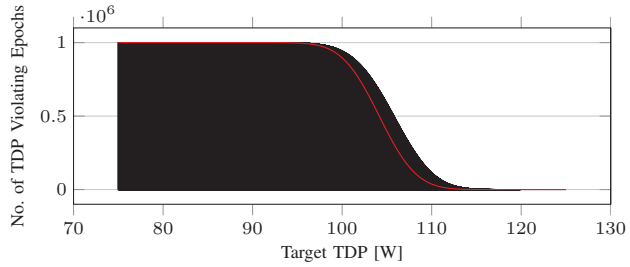
Comparative Baseline: In this work, we choose to compare against a scalable non-probabilistic greedy algorithm called *SortedWS* [8] designed for many-cores. *SortedWS* allocates power budget to tasks in decreasing order of instantaneous speedup without TDP violation for maximizing throughput. Throughput is measured as aggregate of speedups of all tasks.



(a) Task Boost Distribution



(b) Power Consumption Distribution



(c) TDP Violation (Q-Function) Distribution

Fig. 5: Observed- and predicted distributions for a 256 tasks (1024 threads) workload with boost target of 192 tasks ($p = .75$) on a 1024-core many-core.

Probabilistic Modeling Accuracy: We ran 256 tasks (1024 threads) workload on a 1024-core many-core with target of 192 tasks (75 %) to boost using DVFS in a given scheduling epoch. Figure 5a plots the predicted- and observed distribution of DVFS boosted tasks. Figure 5b shows the corresponding predicted- and observed total system power consumption distribution. Results show that our model can predict distribution of both speedup and power consumption with high accuracy. Figure 5c plots the predicted- and observed distribution of TDP violating scheduling epochs with different targeted TDP values. Results show that our model can also predict TDP violation distribution accurately.

Performance Comparison: Figure 6 compares the throughput for a 256 tasks (1024 threads) workload under *ProGov* and *SortedWS* [8] Governor for different values of TDP Threshold $\hat{Q}$ with TDP $\hat{W}$ set at 100 W. Average total system speedup per scheduling epoch is selected as the throughput metric. Since *SortedWS* does not allow for any TDP violation, it ignores $\hat{Q}$ and produces only one fixed result in Figure 6. On the other hand, *ProGov* allows for an increase in throughput with increase in the value of $\hat{Q}$. When TDP Threshold $\hat{Q}$

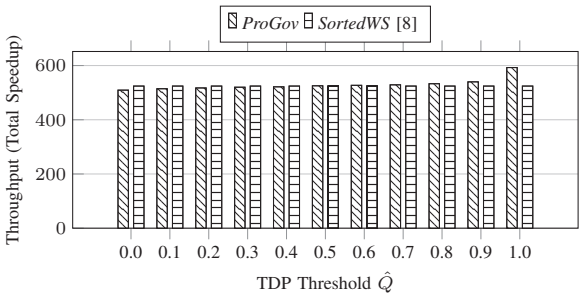


Fig. 6: Throughput comparison between *ProGov* and *SortedWS* [8] for different values of TDP Threshold $\hat{Q}$ with TDP $\hat{W}$ set at 100 W while executing 256 tasks (1024 threads) workload on 1024-core many-core.

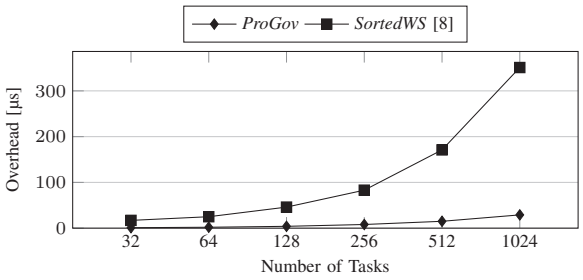


Fig. 7: Expected overhead for power budgeting varisized workloads with *ProGov*.

is set to its lowest value zero, there is practically no risk of TDP violation but at this setting it also has 2.85% lower performance than *SortedWS*. When TDP Threshold $\hat{Q}$ is set to one, TDP is completely ignored and system runs all tasks using high DVFS level resulting in maximum system performance. Intermediate values of $\hat{Q}$ allows for trade-off between performance and TDP violation risk (Figure 6).

Scalability: The real motivation behind using a probabilistic Governor is its ability to scale up much more gracefully with increase in problem size. We run *ProGov* and *SortedWs* cycle-accurately on *gem5* with representative input and report its worst case problem solving time in Figure 7 with workloads of different sizes. Results show that *ProGov* can solve a problem of allocating power budgets to 1024 tasks many-core in only 29 µs. For a Governor operating at default 10 ms scheduling granularity of current multi-core OS [17], this translates into an overhead of 0.29%. In comparison, *SortedWS* takes 351 µs to perform power budgeting for same number of tasks. Therefore, *ProGov* provides a 97.13% (or 12x) reduction in overhead compared to *SortedWS*.

V. Conclusion

In this work, we proposed a Governor called *ProGov* based on a probabilistic technique for power budgeting on many-cores. *ProGov* provides superior scalability in comparison to existing non-probabilistic power budgeting techniques, while providing mathematical guarantees on risk of TDP violations. Proof of concept cycle-accurate simulations show that *ProGov* results in 97.13% less overhead than non-probabilistic greedy Governor on a thousand-core many-core.

Acknowledgment

This work was supported in parts by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Centre “Invasive Computing” (SFB/TR 89) and in parts by Singapore Ministry of Education Academic Research Fund Tier 2 MOE2014-T2-2-129.

References

[1] J. Henkel, A. Herkersdorf, L. Bauer, T. Wild, M. Hübner, R. K. Pujari, A. Grudnitsky, J. Heisswolf, A. Zaib, B. Vogel, V. Lari, and S. Kobbe, “Invasive Manycore Architectures,” in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2012.

[2] T. S. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra, and S. Vishin, “Hierarchical Power Management for Asymmetric Multi-Core in Dark Silicon Era,” in *Design Automation Conference*, 2013.

[3] A. Das, A. Kumar, B. Veeravalli, R. Shafik, G. Merrett, and B. Al-Hashimi, “Workload Uncertainty Characterization and Adaptive Frequency Scaling for Energy Minimization of Embedded Systems,” in *Design, Automation & Test in Europe (DATE)*, 2015.

[4] K. Kang, J. Kim, S. Yoo, and C.-M. Kyung, “Temperature-Aware Integrated DVFS and Power Gating for Executing Tasks with Runtime Distribution,” *Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2010.

[5] P. S. Mann, *Introductory Statistics*. John Wiley & Sons, 2007.

[6] T. Mudge, “Power: A First-Class Architectural Design Constraint,” *Computer*, 2001.

[7] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi, “An Analysis of Efficient Multi-core Global Power Management Policies: Maximizing Performance for a Given Power Budget,” in *International Symposium on Microarchitecture (MICRO)*, 2006.

[8] J. Sartori and R. Kumar, “Three Scalable Approaches to Improving Many-Core Throughput for a Given Peak Power Budget,” in *International Conference on High Performance Computing (HiPC)*, 2009.

[9] T. Somu Muthukaruppan, A. Pathania, and T. Mitra, “Price Theory based Power Management for Heterogeneous Multi-Cores,” in *Architectural Support for Programming Languages and Operating Systems*, 2016.

[10] S. Fan, S. M. Zahedi, and B. C. Lee, “The Computational Sprinting Game,” in *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2016.

[11] Z. Li and B. Li, “Probabilistic Power Management for Wireless Ad Hoc Networks,” *Mobile Networks and Applications*, 2005.

[12] D. G. Feitelson and L. Rudolph, “Metrics and Benchmarking for Parallel Job Scheduling,” *Job Scheduling Strategies for Parallel Processing*, 1998.

[13] S. Boyd-Wickizer, H. Chen, R. Chen, Y. Mao, M. F. Kaashoek, R. Morris, A. Pesterev, L. Stein, M. Wu, Y.-h. Dai *et al.*, “Corey: An Operating System for Many Cores,” in *Operating Systems Design and Implementation (OSDI)*, 2008.

[14] J. Held, “Single-chip Cloud Computer,” in *Euro-Par-Workshop*, 2010.

[15] B. Rountree, D. K. Lowenthal, M. Schulz, and B. R. De Supinski, “Practical Performance Prediction under Dynamic Voltage Frequency Scaling,” in *Green Computing Conference and Workshops*, 2011.

[16] Y. H. Wang, “On the Number of Successes in Independent Trials,” *Statistica Sinica*, 1993.

[17] V. Pallipadi and A. Starikovskiy, “The Ondemand Governor,” in *The Linux Symposium*, 2006.

[18] I. Ukhov, P. Eles, and Z. Peng, “Probabilistic Analysis of Power and Temperature under Process Variation for Electronic System Design,” *Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2014.

[19] W. J. Cody, “Rational Chebyshev Approximations for the Error Function,” *Mathematics of Computation*, 1969.

[20] C. Bienia, S. Kumar, J. P. Singh, and K. Li, “The PARSEC Benchmark Suite: Characterization and Architectural Implications,” in *Parallel Architectures and Compilation Techniques (PACT)*, 2008.

[21] N. Binkert *et al.*, “The gem5 Simulator,” in *SIGARCH Computer Architecture News*, 2011.

[22] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, “McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures,” in *International Symposium on Microarchitecture (MICRO)*, 2009.