QoS-Aware Stochastic Power Management for Many-Cores

[*]Anuj Pathania, [*]Heba Khdr, [+]Muhammad Shafique, [†]Tulika Mitra, [*]Jörg Henkel

[*]Chair of Embedded System, Karlsruhe Institute of Technology, Germany
[+]Institute of Computer Engineering, Vienna University of Technology (TU Wien), Austria
[†]School of Computing, National University of Singapore, Singapore
Corresponding Author: anuj.pathania@kit.edu

ABSTRACT

A many-core processor can execute hundreds of multi-threaded tasks in parallel on its 100s - 1000s of processing cores. When deployed in a Quality of Service (QoS)-based system, the many-core must execute a task at a target QoS. The amount of processing required by the task for the QoS varies over the task's lifetime. Accordingly, Dynamic Voltage and Frequency Scaling (DVFS) allows the many-core to deliver precise amount of processing required to meet the task QoS guarantee while conserving power. Still, a global control is necessitated to ensure that the many-core overall does not exceed its power budget.

Previously, only non-stochastic controls have been proposed for the problem of QoS-aware power budgeting in many-cores. We propose the first stochastic control for the problem, which has a computational complexity less than the non-stochastic control by a factor of O ($\ln n$) but with equivalent performance. The proposed stochastic control can operate with 6.4x less overhead than the non-stochastic control for a 256-task workload.

CCS CONCEPTS

• **Computer systems organization → Embedded systems**;

KEYWORDS

Many-Core, Power Budgeting, Probabilistic Control

1 INTRODUCTION

A many-core processor comprises of 100s - 1000s of processing cores and can execute hundreds of multi-threaded tasks in parallel [8]. The many-core is expected to execute a task at a user-defined target Quality of Service (QoS) when deployed in a QoS-aware system. We choose to measure QoS of the task with the number of Instructions per Second (IPS) executed corresponding to the task.

Figure 1 denotes the changes in IPS of a single-threaded *ferret* benchmark (task) on a given frequency. Figure 1 shows that the fixed frequency cannot keep the task's QoS consistent. This problem can be abated with the help of Dynamic Voltage and Frequency Scaling (DVFS). DVFS allows change in frequency of the cores executing the task to deliver variable amount of processing. DVFS can, therefore, be used for keeping QoS of the task close to the target QoS as shown in Figure 2. As the number of DVFS frequencies are limited, not every target QoS can be precisely attained. Power consumption of a core increases with the increase in its frequency. This results in the task consuming variable amount of power over its lifetime as also shown in Figure 2. Achieving QoS more than the task's target QoS is an unnecessary waste of power.

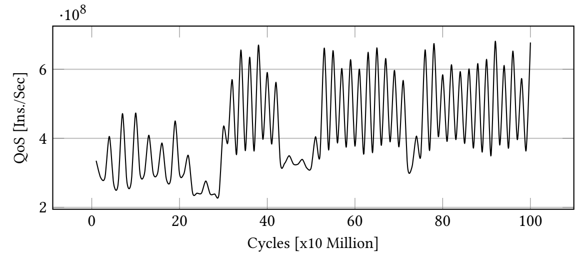

Figure 1: Execution profile of single-threaded *ferret* benchmark showing variation in its IPS during execution over a single fixed frequency.

Limited heat dissipation capacity of the many-core forces it to operate under a power budget called Thermal Design Power (TDP) [15]. Continuous operation beyond TDP leads to a thermal emergency wherein a hardware-triggered Dynamic Thermal Management (DTM) reduces all core frequencies to the minimum. Frequent triggering of DTM leads to deterioration in the many-core's performance. When executing in parallel, individual tasks executing within TDP can violate the TDP in totality. Therefore, it is mandated to carefully budget the TDP between tasks while keeping their QoS requirements under consideration. An Operating System (OS) sub-routine called a Governor is tasked to manage the TDP.

Previously, only non-stochastic controls – centralized [11] or distributed [5] – have been employed in Governors for QoS-aware power budgeting in many-cores. A non-stochastic Governor involves monitoring of executing tasks for their QoS, power consumption and other similar parameters to make power budgeting decisions. Decisions dictate to each individual task the frequency to be used. As the number of tasks executing on the many-core increase, the non-stochastic Governor struggles to keep up due to increased computational overhead and thereby does not scale.

A stochastic Governor in contrast centrally optimizes distribution of the many-core's total power consumption over time by manipulating executing tasks' target QoS. Under the stochastic Governor, task decides locally which core frequency to use itself

DAC '18, June 24–29, 2018, San Francisco, CA, USA

ACM ISBN 978-1-4503-5700-5/18/06…$15.00
https://doi.org/10.1145/3195970.3196097

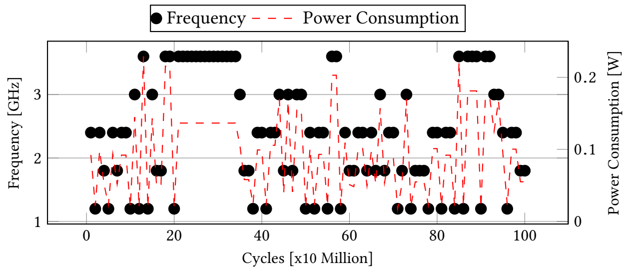

Figure 2: Execution profile of single-threaded *ferret* benchmark showing need for different DVFS levels to maintain a consistent QoS resulting in variable power consumption.

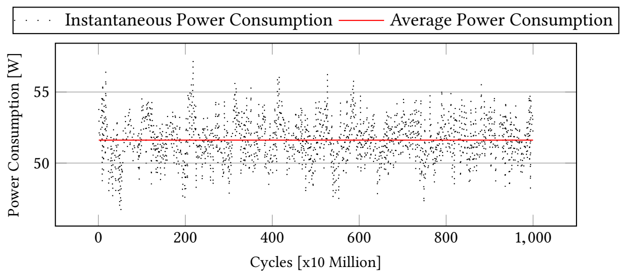

Figure 3: Total power consumption of a many-core when executing a 1024-thread workload from 256 tasks.

based on its current and set target QoS. As the stochastic Governor operates on distributions, it is required to change QoS of tasks only when some task arrives or leaves the many-core. Decisions of the stochastic Governor also have a lower computational overhead than similar decisions of the non-stochastic Governor. As the tasks perform DVFS independent of the stochastic Governor, DVFS could be performed at fine granularity to save more power without additional governor-induced scheduling overheads.

The stochastic Governor works on an observation that power consumption of the many-core when executing hundreds of independent tasks in parallel is quite stable. Figure 3 shows instantaneous total power consumption of the many-core when running a 1024-thread workload comprising of 256 independent tasks each with its own QoS and performing independent DVFS stays very close to the average total power consumption. This observation can be attributed to the fact that even though the power consumption of an individual task to maintain its QoS can vary over time, it has no correlation with the power consumptions of the other executing tasks. At any given time, some of the tasks transit from a high-power consumption phase to a low-power consumption phase and vice versa without any synchronization. This lack of synchronization results in a predictable total power consumption behavior that can be optimized by the stochastic Governor.

Our Novel Contributions: We propose the first stochastic DVFS-based QoS-aware power budgeting Governor for many-cores called *StoGov*. *StoGov* has a computational complexity $O\ (\ln n)$ factor less than a non-stochastic Governor, while providing equivalent performance. Therefore, *StoGov* can scale up much better with the increase in the number of cores in many-cores.

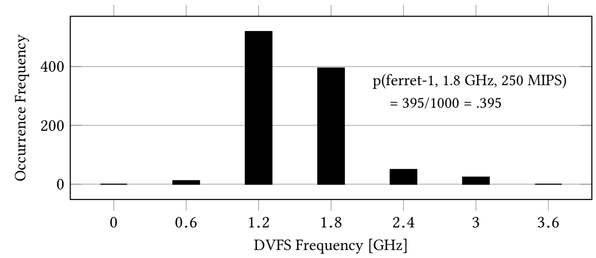

Figure 4: Histogram of DVFS frequency used in *ferret* (single-threaded) benchmark for a given QoS (250 MIPS) along with a sample calculation of probability for a frequency.

2 RELATED WORK

The problem of QoS-aware power budgeting for multi/many-cores has been previously studied only from a non-stochastic perspective [16]. A non-stochastic governor can operate directly on feedback from power sensors often available at a core-, cluster- or chip-level granularity depending upon the hardware [11]. Feedback can be processed to make power budgeting decisions using techniques like online learning [5] or greedy-search [11].

Many works opt to use a PID (Proportional Integrate Derivative) controller as the non-stochastic control in their QoS-aware Governors [15]. The gains in the PID controller of the non-stochastic Governor need to be tuned properly for it to work. As gains tuned for one workload may not hold for another workload, it makes the Governor based on the PID controller impractical. The stochastic Governor on the other hand requires no such fine tuning.

We were the first to develop a stochastic power budgeting Governor for many-cores with the goal of maximizing speedup [14]. However, the introduced Governor – due to its inherent mathematical constructs – cannot be applied to QoS tasks and can operate only with two frequency levels – *High* and *Low*. *StoGov* Governor introduced in this work addresses both these shortcomings.

3 STOCHASTIC POWER BUDGETING

Task Model: Let T be a set of $|T|$ multi-threaded tasks executing on the many-core, indexed by the symbol t_i. Let I_{t_i} be a target IPS (QoS) for the task t_i measured in MIPS (Millions of Instruction per Second). We assume a rigid task model [6], which means the task's threads-to-cores mapping is immutable once it starts execution. We also assume that the task executes with one thread per-core model well-suited for the many-core [3].

Core Model: Let F be a set of $|F|$ discrete frequencies a core in the many-core can operate using DVFS, indexed by the symbol f_j. We assume all the cores in the many-core can perform independent DVFS like *Intel Haswell* processor [7]. Still by design, the cores assigned to a given task operate at the same frequency. The unused cores are always power-gated to save power.

Probabilistic DVFS Model: Let $p(t_i, f_j, I_{t_i})$ represent a probability that under an isolated execution without any power budget constraints the task t_i is using the frequency f_j when its QoS target is set at I_{t_i}. Mathematically $p(t_i, f_j, I_{t_i})$ represents fraction of total execution time spent by the task t_i in the frequency f_j to achieve the QoS I_{t_i} and can be obtained using profiling. Figure 4 shows an exemplary calculation of the probability $p(t_i, f_j, I_{t_i})$. The probability is also dependent upon the input given to the task t_i.

The task t_i acts as an independent Bernoulli trial that uses the frequency f_j with the probability $p(t_i, f_j, I_{t_i})$ and uses frequencies other than the frequency f_j with the probability $1 - p(t_i, f_j, I_{t_i})$. As the tasks in the many-core have different probabilities of using the frequency f_j, the total usage of the frequency f_j shows a Poisson binomial distribution. Let μ_{f_j} and σ_{f_j} be the mean and the standard deviation of the Poisson binomial distribution, respectively.

$$\mu_{f_j} = \sum_{i=1}^{|T|} p(t_i, f_j, I_{t_i}) \quad (1)$$

$$\sigma_{f_j} = \sqrt{\sum_{i=1}^{|T|} (1 - p(t_i, f_j, I_{t_i})) p(t_i, f_j, I_{t_i})} \quad (2)$$

The probability that $K \leq |T|$ tasks would be using the frequency f_j is given by a Probability Mass Function (PMF) $Pr_{f_j}(K)$ [17].

$$Pr_{f_j}(K) = \sum_{A \in F_K} \prod_{t_x \in A} p(t_x, f_j, I_{t_x}) \prod_{t_y \in A^C} (1 - p(t_y, f_j, I_{t_y}))$$

where F_K is a set of all combinations of K tasks selected from the set of T tasks. Set A^C is a complement set of A. The complexity of obtaining the PMF $Pr_{f_j}(K)$ directly has a factorial complexity of $O(|T|!)$. Hence, it is infeasible to directly obtain PMF $Pr_{f_j}(K)$ at runtime when $|T| >> 1$.

We overcome the complexity using central limit theorem [12], which applied here states that the PMF Pr_{f_j} will approximately exhibit a normal distribution if the following two conditions are met. First condition: all the tasks in the many-core should run independent of each other and hence their usage of the frequency f_j should exhibit no correlation. The condition holds very well on system paradigms such as *InvasIC* computing [8] which support predictable execution [18] where shared-resource contentions do not manifest. This condition is not mandatory for the threads of a given task, which are inherently correlated. Second condition: there is large number of tasks executing in parallel that use the frequency f_j substantially, which holds on the many-core. Under these conditions, we assume that the discrete PMF $Pr_{f_j}(K)$ can be approximated by a continuous Probability Density Function (PDF) of a normal distribution with the mean μ_{f_j} and standard deviation σ_{f_j}.

$$Pr_{f_j}(K) = \frac{1}{\sqrt{2(\sigma_{f_j})^2 \pi}} e^{-\frac{(K-\mu_{f_j})^2}{2(\sigma_{f_j})^2}} \quad (3)$$

Figure 5 shows an observed PMF and corresponding approximated PDF $Pr_{1.8\ GHz}(K)$ for a 256-task (1024-thread) workload. Figure 5 shows that the approximation works well in practice.

Probabilistic Power Model: We now need to translate the PDF $Pr_{f_j}(K)$ that represents the distribution of usage of the frequency f_j to the contribution of that usage to the many-core's total power consumption. Using the normal approximation of PDF $Pr_{f_j}(K)$, we can find the probability that $K \leq |T|$ tasks would be using the frequency f_j but it does not tell us the composition of those K tasks. This makes translation of the PDF $Pr_{f_j}(K)$ to the power consumption distribution difficult because different tasks can have different

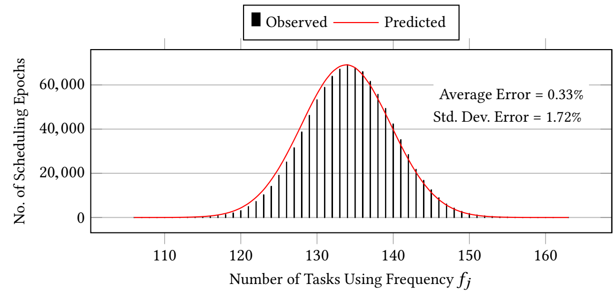

Figure 5: Observed and predicted distribution for usage by tasks of frequency $f_j = 1.8\ GHz$ for a 256-task (1024-thread) workload on a many-core.

power consumption at the same frequency. Furthermore, a task can also have a different power consumption at a given frequency depending upon its current execution phase. An approximation would be to work out the expected power consumption of a task at the frequency f_j and assume all the K tasks in the PDF $Pr_{f_j}(K)$ have the same expected power consumption. Due to the law of large numbers [12], the error introduced by this approximation will reduce with the increase in the number of independently executing tasks provided many scheduling epochs are observed.

Let $\overline{W}(t_i, f_j, I_{t_i})$ be an average power consumption of the task t_i at the frequency f_j with the target QoS set at I_{t_i}. We use probability weighted power consumption of the task set T at the frequency f_j to obtain the expected power consumption of all tasks at that frequency. We then combine it with Equations (1) and (2) to calculate the mean $\mu^W_{f_j}$ and standard deviation $\sigma^W_{f_j}$ of the power consumption distribution due the use of the frequency f_j, respectively.

$$\mu^W_{f_j} = \mu_{f_j} \frac{\sum_{t_i=1}^{|T|} \overline{W}(t_i, f_j, I_{t_i})\, p(t_i, f_j, I_{t_i})}{\sum_{t_i=1}^{|T|} p(t_i, f_j, I_{t_i})} \quad (4)$$

$$\sigma^W_{f_j} = \sigma_{f_j} \frac{\sum_{t_i=1}^{|T|} \overline{W}(t_i, f_j, I_{t_i})\, p(t_i, f_j, I_{t_i})}{\sum_{t_i=1}^{|T|} p(t_i, f_j, I_{t_i})} \quad (5)$$

The probability that a scheduling epoch will have a power consumption of X Watts due to the usage of the frequency f_j is given by a PDF $Pr^W_{f_j}(X)$.

$$Pr^W_{f_j}(X) = \frac{1}{\sqrt{2(\sigma^W_{f_j})^2 \pi}} e^{-\frac{(X-\mu^W_{f_j})^2}{2(\sigma^W_{f_j})^2}} \quad (6)$$

Figure 6 shows an observed PMF and approximated PDF of power consumption distribution at a frequency $Pr^W_{1.8\ GHz}(X)$ for a 256-task (1024-thread) workload. Figure 6 shows that the predicted PDF $Pr^W_{f_j}(X)$ is very close to the observed PMF.

We assume the power consumption due to an individual frequency is a linear combination of power consumptions of a same set of independent tasks. Therefore, all the power consumption distributions due to the use of frequencies are jointly normal with each other. This implies a distribution of their sum which is the many-core's total power consumption distribution is also normally

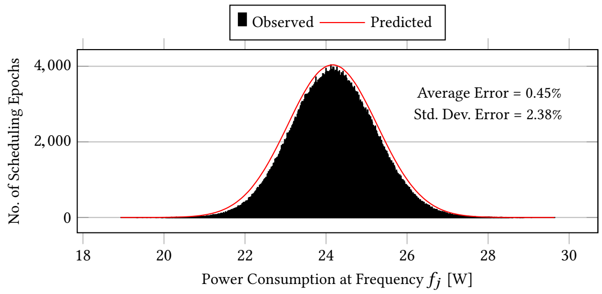

Figure 6: Observed and predicted power consumption distribution due to the frequency $f_j = 1.8\,GHz$ for a 256-task (1024-thread) workload on a many-core.

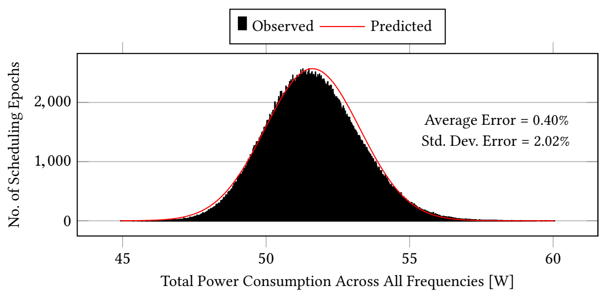

Figure 7: Observed and predicted total power consumption distribution for 256-task (1024-thread) workload.

distributed. Therefore, the mean μ^W of the total power consumption distribution can be obtained by adding the means of power consumption distributions due to the use of all frequencies.

$$\mu^W = \sum_{j=1}^{|F|} \mu_{f_j}^W \tag{7}$$

By design, when a task switches from one frequency to another, it leads to decrease in power consumption due to the former frequency with simultaneous increase in power consumption due to the latter frequency. Therefore, all power consumption distributions due to the use of different frequencies are negatively correlated with each other. Therefore, standard deviation σ^W of the total power consumption distribution can be obtained by adding the variance of power distributions at the individual discrete frequencies adjusted with their covariance.

$$\sigma^W = \sqrt{\sum_{j=1}^{|F|} (\sigma_{f_j}^W)^2 + 2\sum_{j<j'} \rho_{f_j,f_{j'}}^W \sigma_{f_j}^W \sigma_{f_{j'}}^W} \tag{8}$$

where $\rho_{f_j,f_{j'}}^W$ is a correlation coefficient between power consumption at the frequencies f_j and $f_{j'}$. The correlation coefficient $\rho_{f_j,f_{j'}}^W$ can be learned by taking power samples online.

The probability that in a scheduling epoch the many-core will have a power consumption X Watts is given by PDF $Pr^W(X)$.

$$Pr^W(X) = \frac{1}{\sqrt{2(\sigma^W)^2\pi}} e^{-\frac{(X-\mu^W)^2}{2(\sigma^W)^2}} \tag{9}$$

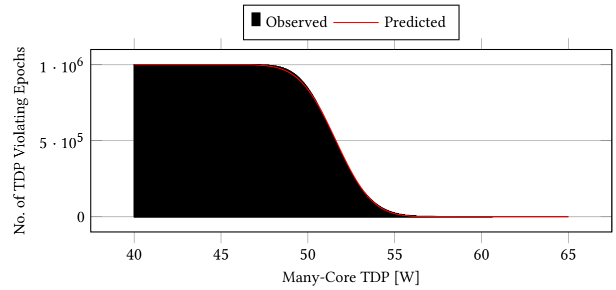

Figure 8: Observed and predicted TDP violation distribution for 256-task (1024-thread) workload on a many-core.

Algorithm 1 Stochastic power budgeting used in *StoGov*.

Input: T, $\hat{W}$, $\delta_{\hat{W}}$;
Output: Δ_I;
$\forall t_i \in T$ Read Profiled Data;
for $\Delta_I = 1.0$ to 0.0 **do**
  for $j = 1$ to $|F|$ **do**
    $I_{t_i} = I_{t_i} * \Delta_I \forall t_i \in T$ ▷ Discount QoS.
    Calculate μ_{f_j} using Equation (1);
    Calculate σ_{f_j} using Equation (2);
    Calculate $\mu_{f_j}^W$ using Equation (4);
    Calculate $\sigma_{f_j}^W$ using Equation (5);
  end for
  Calculate μ^W using Equation (7);
  Calculate σ^W using Equation (8);
  Predict $Q(\hat{W})$ using Equation (10);
  $I_{t_i} = I_{t_i}/\Delta_I \forall t_i \in T$ ▷ Reset Discounted QoS.
  if $Q(\hat{W}) < \delta_{\hat{W}}$ **then**
    break;
  end if
  $\Delta_I = \Delta_I$ - .01;
end for
return Δ_I;

Figure 7 shows the observed PMF and predicted PDF of total power consumption distribution $Pr^W(X)$ for a 256-task (1024-thread) workload with covariance between frequencies considered. Figure 7 shows the error is minimal.

Probabilistic TDP Model: Let TDP of the many-core be symbolized by $\hat{W}$. The probability that a scheduling epoch will violate the TDP is given by a Q-function $Q(\hat{W})$.

$$Q(\hat{W}) = 1 - \int_0^{\hat{W}} Pr^W(X)\, dx \tag{10}$$

Figure 8 shows the observed and predicted distribution of TDP violating epochs for a 256-task (1024-thread) workload. Figure 8 shows that our predicted distribution is quite accurate.

Power Budgeting Algorithm: The stochastic power budgeting used in *StoGov* is shown in Algorithm 1. *StoGov* cannot give a deterministic guarantee that TDP violation will never happen but it can reduce the probability of TDP violation to such low value that it never occurs in the lifetime of the many-core. Furthermore, TDP is a soft-constraint and a thermal emergency only occurs when TDP is violated for prolonged durations. A few TDP violating epochs spread out over time are benign. Hardware-triggered frequency throttling via DTM can act as backup if TDP violations under *StoGov* pushes chip temperature dangerously high. *StoGov* also allows for

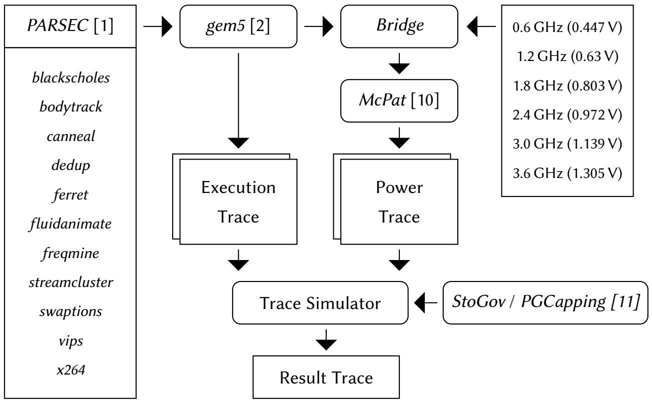

Figure 9: Experimental Setup.

a tradeoff between the TDP violation risk with the performance using a TDP risk threshold $\delta_{\hat{W}}$.

Algorithm 1 takes as input the set of tasks executing on the many-core T, the TDP $\hat{W}$ and the TDP risk threshold $\delta_{\hat{W}}$. It then calculates the risk of TDP violation $Q(\hat{W})$. If the risk is higher than the TDP risk threshold $\delta_{\hat{W}}$ then all the tasks are forced to pay equal performance penalty by discounting their target IPS (QoS) by a factor of Δ_I. Higher the value of Δ_I, higher is the performance. Algorithm 1 is executed only when some task enters or leaves the many-core. Note that DVFS is performed locally and independently by the tasks themselves without any relation to Algorithm 1.

Computational Complexity: In Algorithm 1, as QoS discount factor Δ_I always take value between 1.0 and 0.0, the computational complexity of the loop in Step 2 is constant. The worst-case computational complexity of any step in the frequency loop at Step 3 is O($|T|$), so the loop's complexity is O($|F||T|$). As all the other steps have computational complexity less than the frequency loop, worst-case computational complexity of *StoGov* algorithm is O($|F||T|$). Use of centrally available stochastic profiles in *StoGov* result in worst-case space complexity of O($|T|$). The need to propagate decisions to tasks and make online observation to learn covariance result in worst-case communication complexity of $O(|T|)$. Note that both the worst-case computation and communication complexity is $O(1)$ in scheduling epochs where no task enters or leaves the many-core, or any online observation is performed.

4 EXPERIMENTAL RESULTS

We must observe millions of scheduling epochs for a many-core processor executing hundreds of tasks in parallel to obtain various distributions shown in this work. As any many-core large enough to test stochastic power budgeting does not exist at present, we must rely on simulations. We are also bound to use trace-based simulations as neither cycle-accurate [2] nor even interval simulations [4] of such a large many-core is time-wise feasible.

Therefore, similar to prominent existing works [9, 13], we also use a trace-based simulator as shown in Figure 9. The simulator takes in isolated execution and power traces from cycle-accurate simulations of a multi-core system as input. Cycle-accurate traces are obtained from *gem5* [2] cycle-accurate simulator bridged with *McPat* [10] power simulator. Time constraints in obtaining cycle-accurate traces forces the simulated shared-memory system in *gem5* to be limited to a maximum of eight cores. Each core possesses 16 KB L1 data and instruction cache, and supports *Alpha* ISA. The cores share a 32 KB L2 cache. A core can choose to run at one of six frequencies listed in Figure 9. The 22 nm planar CMOS cores have an in-order pipeline with a low-power design. The core's maximum power consumption is around 0.0065 W (or 0.25 W) at the lowest (or highest) frequency of 0.6 GHz (or 3.6 GHz). The simulations are performed in *gem5*'s Full System (FS) mode. The cycle-accurate simulation traces are then combined with assumption of composable execution [18] by the trace simulator to simulate a 1024-core many-core. The ambient temperature is set at 40 °C whereas DTM is triggered when many-core temperature exceeds 85 °C. The thermal modeling parameters are set such that DTM will never be triggered if many-core always operates within a TDP of 45 W.

Random mixture of multi-threaded benchmarks (processing *simsmall* input) from *PARSEC* benchmark suite [1] as listed in Figure 9 are used as tasks. To simulate independent execution of large number of tasks with a limited set of available benchmark types, tasks are executed with a random initial skew in the trace simulator. Granularity of scheduling epoch is set at 10 ms.

Comparative Non-Stochastic Governor: We choose to compare *StoGov* against the *PGCapping* Governor [11]; both being centralized Governors. *PGCapping* uses an effective non-stochastic *Quicksearch* greedy algorithm to perform DVFS-based QoS-aware power budgeting for multi-/many-cores. *Quicksearch* similar to *StoGov* assumes availability of per-core DVFS for power budgeting.

Quicksearch operates on basis of power/performance ratios. Depending upon whether current power consumption of the many-core is above or below the TDP, *Quicksearch* calculate a ratio of power decrease to performance loss $D_{power-perf}$ or a ratio of performance gain to power increase $D_{perf-power}$ for all the cores, respectively. The frequency of the core with the highest power decrease to performance loss ratio $D_{power-perf}$ (or the highest performance gain to power increase ratio $D_{perf-power}$) is decreased (or increased) if the power is expected to be above (or below) the TDP. $D_{power-perf}$ (or $D_{perf-power}$) is then recalculated for the task whose frequency has been changed. *Quicksearch* algorithm is iteratively repeated till power consumption is just below the TDP.

As we operate with multi-threaded benchmarks with an assumption that all the cores assigned to a task operates at same frequency, we calculate ratios $D_{power-perf}$ and $D_{perf-power}$ for *PGCapping* at the task granularity rather than core granularity. Furthermore, *PGCapping* originally used product of core utilization and core frequency as a measure of performance (QoS), which we replace with IPS in this work for a fair comparison.

When the *Quicksearch* algorithm is implemented with help of *quicksort* and *binary search* algorithms, the worst-case computationally complexity of *Quicksearch* works out to be $O(|F||T|\ln|T|)$, which theoretically is a factor of $O\ (ln|T|)$ more than *StoGov*.

Stochastic vs. Non-Stochastic Performance: We simulate a many-core operating in a closed system [6] to compare efficacy of different Governors. Many-core attains peak performance (100%) when all QoS tasks executing on it achieve their target QoS at all times. Figure 10 shows how the many-core's performance measured in percentage of target QoS sustained for a task on average for a 256-task (1024-thread) workload changes with different values

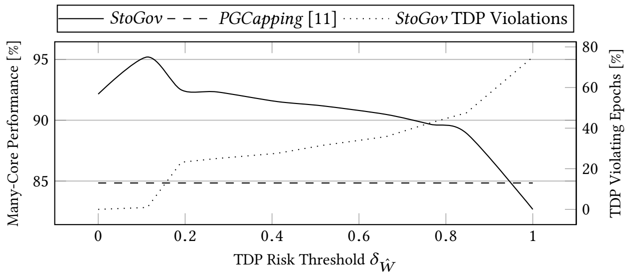

Figure 10: System performance comparison between *StoGov* and *PGCapping* for different values of TDP risk threshold $\delta_{\hat{W}}$ when executing 256-task (1024-thread) workload with TDP $\hat{W}$ set at 45 W.

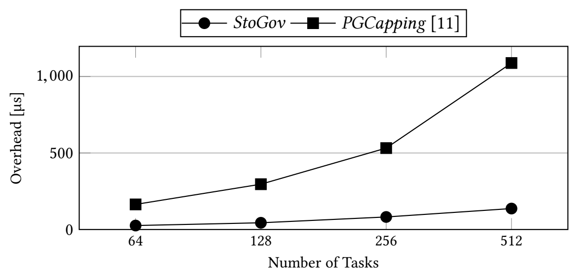

Figure 11: Measured worst-case scheduling overheads for *StoGov* for varisized workloads.

of the TDP risk threshold $\delta_{\hat{W}}$. As *PGCapping* does not consider $\delta_{\hat{W}}$, produces the same performance for all values of $\delta_{\hat{W}}$ whereas *StoGov* allows a tradeoff between the performance and TDP risk threshold $\delta_{\hat{W}}$. Increase in the TDP risk threshold $\delta_{\hat{W}}$ leads to increase in percentage of TDP violating epochs under *StoGov* as also shown in Figure 10. Ignoring the TDP beyond a certain level can lead to performance loss instead of gain as hardware-triggered thermal throttling on TDP violations can substantially deteriorate performance. This effect on many-core's performance can be seen in Figure 10 for higher values of the TDP risk threshold $\delta_{\hat{W}}$.

It can be seen from Figure 10 that *StoGov* results in superior performance compared to *PGCapping* even when the TDP risk threshold $\delta_{\hat{W}}$ is set to 0. *PGCapping* penalizes tasks asymmetrically resulting in several tasks operating far above their target QoS at the cost of other tasks. *StoGov* on the contrary, penalizes all tasks fairly in equal proportions which also results in better performance.

Stochastic vs Non-Stochastic Scalability: Figure 11 shows the worst-case scheduling overheads of *StoGov* and *PGCapping* for varisized workloads obtained using representative cycle-accurate simulations performed on *gem5*. Our proof-of-concept simulations show *StoGov* is highly scalable and has nearly 6.48x less worst-case scheduling overhead than *PGCapping* for a 256-task workload.

5 CONCLUSION

We introduced a QoS-aware stochastic power budgeting Governor for many-cores called *StoGov* in this work. *StoGov* provides strong stochastic guarantees on the risk of TDP violation while allowing trade-off of that risk with performance. Compared to a non-stochastic Governor, *StoGov* provides equivalent performance but with a computational complexity reduced by a factor O (ln n). Therefore, *StoGov* can scale up more efficiently with the increase in number of cores in many-cores.

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 National Research Foundation, Prime Minister's Office, Singapore under its Industry-IHL Partnership Grant and Huawei International Pte. Ltd. NRF2015-IIP003.

REFERENCES

[1] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. 2008. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *Parallel Architectures and Compilation Techniques (PACT)*.

[2] Nathan Binkert et al. 2011. The gem5 Simulator. In *SIGARCH Computer Architecture News (CAN)*.

[3] Silas Boyd-Wickizer, Haibo Chen, Rong Chen, Yandong Mao, M Frans Kaashoek, Robert Morris, Aleksey Pesterev, Lex Stein, Ming Wu, Yue-hua Dai, et al. 2008. Corey: An Operating System for Many Cores. In *Operating Systems Design and Implementation (OSDI)*.

[4] Trevor E Carlson, Wim Heirman, and Lieven Eeckhout. 2011. Sniper: Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-Core Simulation. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*.

[5] Zhuo Chen and Diana Marculescu. 2015. Distributed Reinforcement Learning for Power Limited Many-core System Performance Optimization. In *Design, Automation & Test in Europe Conference (DATE)*.

[6] Dror G Feitelson and Larry Rudolph. 1998. Metrics and Benchmarking for Parallel Job Scheduling. *Job Scheduling Strategies* (1998).

[7] Daniel Hackenberg, Robert Schöne, Thomas Ilsche, Daniel Molka, Joseph Schuchart, and Robin Geyer. 2015. An Energy Efficiency Feature Survey of the Intel Haswell Processor. In *International Parallel and Distributed Processing Symposium Workshop (IPDPSW)*.

[8] Jörg Henkel, Andreas Herkersdorf, Lars Bauer, Thomas Wild, Michael Hübner, Ravi Kumar Pujari, Artjom Grudnitsky, Jan Heisswolf, Aurang Zaib, Benjamin Vogel, Vahid Lari, and Sebastian Kobbe. 2012. Invasive Manycore Architectures. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*.

[9] Canturk Isci, Alper Buyuktosunoglu, Chen-Yong Cher, Pradip Bose, and Margaret Martonosi. 2006. An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget. In *International Symposium on Microarchitecture (MICRO)*.

[10] Sheng Li, Jung Ho Ahn, Richard D Strong, Jay B Brockman, Dean M Tullsen, and Norman P Jouppi. 2009. McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. In *International Symposium on Microarchitecture (MICRO)*.

[11] Kai Ma and Xiaorui Wang. 2012. PGCapping: Exploiting Power Gating for Power Capping and Core Lifetime Balancing in CMPs. In *Parallel Architectures and Compilation Techniques (PACT)*.

[12] Prem S Mann. 2007. *Introductory Statistics*. John Wiley & Sons.

[13] Santiago Pagani, Heba Khdr, Waqaas Munawar, Jian-Jia Chen, Muhammad Shafique, Minming Li, and Jörg Henkel. 2014. TSP: Thermal Safe Power: Efficient Power Budgeting for Many-Core Systems in Dark Silicon. In *Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*.

[14] Anuj Pathania, Heba Khdr, Muhammad Shafique, Tulika Mitra, and Jörg Henkel. 2017. Scalable Probabilistic Power Budgeting for Many-Cores. In *Design, Automation & Test in Europe (DATE)*.

[15] Amir-Mohammad Rahmani, Mohammad-Hashem Haghbayan, Anil Kanduri, Awet Yemane Weldezion, Pasi Liljeberg, Juha Plosila, Axel Jantsch, and Hannu Tenhunen. 2015. Dynamic Power Management for Many-Core Platforms in the Dark Silicon Era: A Multi-Objective Control Approach. In *International Symposium on Low Power Electronics and Design (ISLPED)*.

[16] Amit Kumar Singh, Muhammad Shafique, Akash Kumar, and Jörg Henkel. 2013. Mapping on Multi/Many-Core Systems: Survey of Current and Emerging Trends. In *Design Automation Conference (DAC)*.

[17] Yuan H Wang. 1993. On the Number of Successes in Independent Trials. *Statistica Sinica* (1993).

[18] Andreas Weichslgartner, Deepak Gangadharan, Stefan Wildermann, Michael Glaß, and Jürgen Teich. 2014. DAARM: Design-Time Application Analysis and Run-Time Mapping for Predictable Execution in Many-Core Systems. In *Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*.