Multi-Armed Bandits for Efficient Lifetime Estimation in MPSoC design

Calvin Ma, Aditya Mahajan, and Brett H. Meyer
Department of Electrical and Computer Engineering, McGill University
Montréal, Québec, Canada
calvin.ma@mail.mcgill.ca, {aditya.mahajan, brett.meyer}@mcgill.ca

***Abstract*—Reliability in integrated circuits is becoming a critical issue with the miniaturization of electronics. Smaller process technologies have led to higher power densities, resulting in higher temperatures and earlier device wear-out. One way to mitigate failure is by over-provisioning resources and remapping tasks from failed components to components with spare capacity, or slack. Since the slack allocation design space is large, finding the optimal is difficult, as brute-force approaches are impractical. During design space exploration, device lifetimes are typically evaluated using Monte-Carlo Simulation (MCS) by sampling each design equally; this method is inefficient since poor designs are evaluated as accurately as good designs. A better method will focus sampling time on the designs that are difficult to distinguish, reducing the time required to evaluate a set of designs; this can be accomplished using Multi-armed Bandit (MAB) Algorithms. This work demonstrates that MAB achieve the same level of accuracy as MCS in 1.45 to 5.26 times fewer samples.**

I. Introduction

In tightly integrated systems, such as Multi-Processor System on Chips (MPSoC), process innovations and miniaturization have resulted in faster device degradation and earlier wear-out related failures [1]–[3]. In one study, it was observed that scaling from 180nm to 65nm reduced the lifetime by 3× [3].

A common way to ensure reliability and extend the lifetime of a system is by adding redundant resources. Such redundancy can be provided by either adding spare components or adding extra capacity in the form of slack to existing components. Redundancy through the use of spare components has been widely studied, and although effective, it is more costly [4]. For cost-sensitive applications, allocating slack is a better alternative. Slack refers to over-provisioning by choosing components such that they are normally under-utilized; failed tasks can then be mapped onto these devices since it has additional computational capacity. Finding the optimal slack allocation is, however, a difficult problem [5].

The process of slack allocation involves selecting a component and then swapping it with one of higher capacity. In a typical MPSoC, a system has multiple components each with a number of upgrade options. For example, a base processor can be swapped with one having higher MIPS; memories can also be upgraded to have higher capacity. Since MPSoC have multiple processors and memories, the number of design permutations grow exponentially with the number of components and upgrade options.

It is not possible to evaluate the lifetime of all the designs due to the time requirements and the size of the design space. The current state-of-the-art optimization techniques, such as ant colony optimization [1], simulated annealing [2], evolutionary algorithms [6], and greedy algorithms [5], select the best design in an iterative manner. In particular, a subset of designs are selected, their performance is evaluated using Monte-Carlo simulation (MCS), the top few designs are selected, and a new set of designs are strategically selected. Such iterative procedures are tuned to converge to the best design.

At each step of these optimization techniques, the algorithms select the top few designs from a given subset using Monte-Carlo simulation. Traditionally, all designs lifetimes are evaluated with the same accuracy. For example, if 500 samples are available to evaluate four designs, then each design is evaluated using 125 samples.

For the aforementioned meta-heuristic algorithms, accurate lifetime estimation may not be necessary; instead, partially ordering designs by their quality is often sufficient. Since only the top few designs are of interest, ideally one should spend more samples evaluating the good designs than poor designs. Since it is not known a priori which designs are good and which are poor, the designs have to be sampled to determine their quality. This predicament is referred to as the *exploration-exploitation* trade-off. One of the most popular approaches to address exploration-exploitation trade-offs is the multi-armed bandits (MAB) framework [7]. MAB have been successfully used in various application domains including clinical trials, online advertising, sensor networks and economics. In this

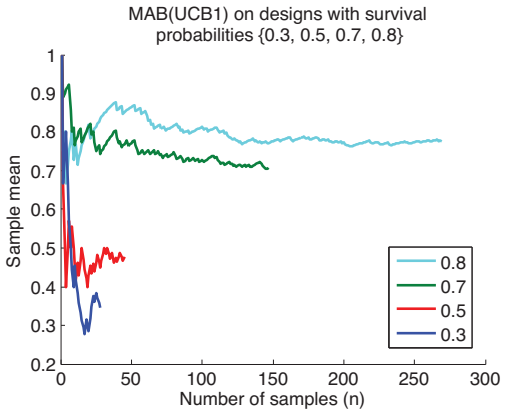


Fig. 1. The sample mean versus number of samples for UCB1 algorithm in a toy problem of identifying the design with the highest survival probability.

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

paper, we evaluate the effectiveness of the MAB framework for slack allocation in lifetime-aware design of MPSoC.

To illustrate why MAB framework may be useful for lifetime-aware design, consider the following toy problem. Fig. 1 shows a problem with four designs, each with independent survival probabilities; the objective is maximize the survival probability. Instead of distributing the sampling budget equally between all the designs, suppose we sample them sequentially using UCB1 (a simple MAB algorithm). In particular, suppose the survival probabilities are $\{0.3, 0.4, 0.7, 0.8\}$ and a total of 500 samples are available. The plot shows the sample mean versus the number of samples. Note that the good designs (those with survival rate 0.7 and 0.8) are sampled more often than the bad designs (those with survival rate 0.3 and 0.5). Such a balance between exploration and exploitation is a characteristic feature of MAB.

Our main contribution is to evaluate the effectiveness of the MAB framework in identifying top designs for slack allocations (out of a given set of designs) that maximize expected lifetimes in MPSoCs. To the best of our knowledge, this is the first attempt to use the MAB framework for lifetime-aware design. We evaluate two specific MAB algorithms—*Successive Accept Reject* (SAR) [8] and *Gap based Exploration with Variance* (GapE-V) [9]—on four benchmarks and find that the MAB framework requires 1.49–5.26× fewer samples to achieve the same level of accuracy as MCS.

II. Related Work

The earliest works in lifetime-aware design for MPSoCs were [4], [10], [11], which modeled each failure mechanism as a random variable and a corresponding distribution. Each of the failure distributions were fitted with the mean lifetime provided by the empirical estimates given by the Joint Electronic Device Engineering Council (JEDEC) [12].

In [10], exponential distributions were used but they did not provide a good estimate for wear-out failures since they are memoryless. In an extended work [4], log-normal distributions were used instead. This work showed that when resources were duplicated, lifetime could be improved 3.17× but with an area increase of 2.254×. The alternative is to operate circuits in a degraded state; the system continues to operate at a lower performance until requirements can no longer be met. This method showed that lifetime could be extended by 1.42× with a loss of performance of about 5%.

Other enhancements to lifetime modeling were made by [11], [13]. [11] corrected for temperature as workloads changed due to component failure. [13] explored using a combination of log-normal and Weibull distributions, and found that Weibull distributions were more accurate for systems with many components and log-normal distributions were more accurate for systems with fewer. In general, in a system with 500 components, the error for both log-normal and Weibull distribution cross over at 10% error when identifying the time where 1% of all devices fail [13].

One low-cost strategy for ensuring MPSoC can operate in the presence of device failure is to allocate slack. Finding the optimal slack (balancing increases in cost and lifetime) is the basis of the slack allocation problem. An exhaustive search of all the slack combinations can be avoided by skipping allocations which would not meet the requirements to survive certain types of failures. Using a greedy search of this restricted space, [5] is able to find designs near the Pareto-optimal front by only evaluating 1.4% of the designs.

The general approach to solving lifetime-aware problems has been to use MCS to estimate the true lifetime [1], [2], [5]. Since designers are interested in identifying the best designs, MCS is an inefficient evaluation method since the sampling time is spent uniformly on the bad as well as good designs; instead MAB algorithms are a better alternative.

III. Multi-Armed Bandits

The basic multi-armed bandit (MAB) formulation is as follows [14], [15]. There are D alternatives (often called arms or bandits in the MAB literature). For ease of notation, we use $\mathcal{D} = \{1, \ldots, D\}$ to denote the set of all alternatives. At each time instant, a decision maker picks one alternative, say $i \in \mathcal{D}$, and obtains a random reward R_i. Let $\mu_i = \mathbb{E}[R_i]$ denote the average reward of alternative i. For example, in slack allocation for lifetime-aware design, $\mathcal{D}$ corresponds to the set of designs that have to be evaluated, R_i corresponds to the time to system failure when we simulate the design, and μ_i therefore corresponds to the mean time to failure (MTTF).

If $\{\mu_i\}_{i \in \mathcal{D}}$ were known, it would be straightforward to pick the best or the top-m designs. However, in practice, $\{\mu_i\}_{i \in \mathcal{D}}$ are not known. Multi-armed bandits refers to algorithms that sequentially determine which alternatives to sample. These algorithms may be broadly classified into two categories depending on the type of performance guarantees that they provide. These are:

1) *Regret minimization*, where the objective is to minimize the rate of regret (i.e., how far is the average observed reward from the selected alternative from the reward of the best alternative), either asymptotically or over a finite interval of time. The motivation for looking at rate of regret comes from clinical trials where it is not desirable to give a bad treatment option to a patient just to get a better estimate of how the treatment works.
2) *Budgeted samples*, where the objective is to select the best alternative at the end of an exploration phase. In such models, we do not care about the performance of the alternatives selected during the exploration phase.

The problem at hand—slack allocation for lifetime-aware design—fits in the second category. Formally, the problem may be stated as: *given a set of D designs and a sampling budget of n samples, identify the top m designs.* We shortlisted two MAB algorithms: *Successive Accept Reject* (SAR) and *Gap based Exploration with Variance* (GapE-V).

A. Successive Accept Reject

SAR [8] is presented in Algorithm 1. First, the algorithm divides the sample budget into $D - 1$ phases. The algorithm maintains a list of active designs $\mathcal{A}$. Initially all designs are active, i.e., $\mathcal{A} = \mathcal{D}$. During each phase, the algorithm samples all active designs an equal number of times. At the end of each phase, the algorithm either accepts the arm (or, in our case,

Algorithm 1 SAR [8]

Notation: D = total number of designs
m = desired number of top designs
n = sample budget
Let $\overline{\log}(D) = \frac{1}{2} + \sum_{i=2}^{D} \frac{1}{i}$
Define $n_0 = 0$ and $n_k = \left\lceil \frac{1}{\overline{\log}(D)} \frac{n-D}{D+1-k} \right\rceil$ for $k \geq 1$.

Input: Vector $\mathcal{D}$ of designs to evaluate

Initialize: Set of active designs $\mathcal{A} = \mathcal{D}$
Set of selected designs $\mathcal{S} = \emptyset$
Set $m^\circ = m$

for each phase $k \in \{1, ..., D-1\}$ **do**
  for each active design $i \in \mathcal{A}$ **do**
    Get $n_k - n_{k-1}$ samples of design i
    Update the empirical mean $\hat{\mu}_i$ of design i
  end for
  Sort designs in $\mathcal{A}$ according to their empirical mean.
  Let i^* be the arm with the m°-th best empirical mean
  Let i_* be the arm with the $(m^\circ + 1)$-th best empirical mean.
  for each active design $i \in \mathcal{A}$ **do**
    if design i is among the top m° designs **then**
      Gap $\Delta_i = \hat{\mu}_i - \hat{\mu}_{i_*}$
    else
      Gap $\Delta_i = \hat{\mu}_{i^*} - \hat{\mu}_i$
    end if
  end for
  Let j be the active design with the highest gap
  if j was the best design in $\mathcal{A}$ **then**
    Include j in the set of selected arms, i.e., $\mathcal{S} \leftarrow \mathcal{S} \cup \{j\}$
    $m^\circ \leftarrow m^\circ - 1$
  end if
  Remove j from the set of active arms, i.e., $\mathcal{A} \leftarrow \mathcal{A} \setminus \{j\}$
end for

Output: The set $\mathcal{S}$ of selected designs

Algorithm 2 GapE-V [9]

Notation: D = total number of designs
m = desired number of top designs
n = sample budget

Input: Vector $\mathcal{D}$ of designs to evaluate

Initialize: Sample each design once
Set the empirical mean $\hat{\mu}_i$ as the observed value
Set empirical variance $\hat{\sigma}_i = 0$ for all designs
Set $T_i = 1$ for all designs

for each sample $t = D+1, \ldots, n$ **do**
  Sort designs $\mathcal{D}$ according to their empirical mean
  Let i^* be the arm with the m-th best empirical mean
  Let i_* be the arm with the $(m+1)$-th best empirical mean.
  for each design $i \in \mathcal{D}$ **do**
    if design i is among the top m designs **then**
      Gap $\Delta_i = \hat{\mu}_i - \hat{\mu}_{i_*}$
    else
      Gap $\Delta_i = \hat{\mu}_{i^*} - \hat{\mu}_i$
    end if
    Compute index$_i = -\Delta_i + \sqrt{\frac{2a\hat{\sigma}_i}{T_i}} + \frac{7ab}{3(T_i - 1)}$
  end for
  Let j be the design with the highest index. Sample design j
  Update the empirical mean $\hat{\mu}_j$ and empirical variance $\hat{\sigma}_i$.
  Update $T_j \leftarrow T_j + 1$
end for

Output: Return the designs with the top-m empirical means

design) with the highest empirical mean or rejects the arm with the lowest empirical mean. In both cases the corresponding arm is deactivated.

The critical part is to determine whether to accept or reject at the end of each phase. Suppose that the algorithm has already accepted $m - m^\circ$ designs, so there are m° designs left to find. To do so, the algorithm sorts the set of active designs according to their empirical mean. Let i^* denote the arm with the m°-th best empirical mean $\widehat{\mu}_{i^*}$ and i_* denote the arm with the $(m^\circ + 1)$-th best empirical mean $\widehat{\mu}_{i_*}$. Next, the algorithm computes the *gap* Δ_i for all active designs. For active designs that are among the top m° empirically best designs, the gap $\Delta_i = \hat{\mu}_i - \hat{\mu}_{i_*}$. For active designs that are not among the top m° designs, the gap $\Delta_i = \hat{\mu}_{i^*} - \hat{\mu}_i$.

Once these gaps have been computed, SAR deactivates the arm with the highest gap. If the deactivated arm is the empirically best arm, then it is added to the set $\mathcal{S}$ of selected arms and m° is set to $m^\circ - 1$.

It is shown in [8] that the probability that the set $\mathcal{S}$ selected by the SAR algorithm is *not* the set of top-m designs is upper bounded by $2D^2 \exp\left(-\frac{n-K}{8\overline{\log}(D) H_2^{(m)}}\right)$, where $H_2^{(m)}$ is a measure of the hardness of the problem that depends on the true means $\{\mu_i\}_{i \in \mathcal{D}}$ of the designs. Thus, when the number of samples n is $\mathcal{O}\left(H_2^{(m)}\right)$, the probability that the SAR algorithm chooses a wrong set is small. We refer the reader to [8] for an exact expression of $H_2^{(m)}$.

B. GapE-V (Gap-based Exploration with Variance)

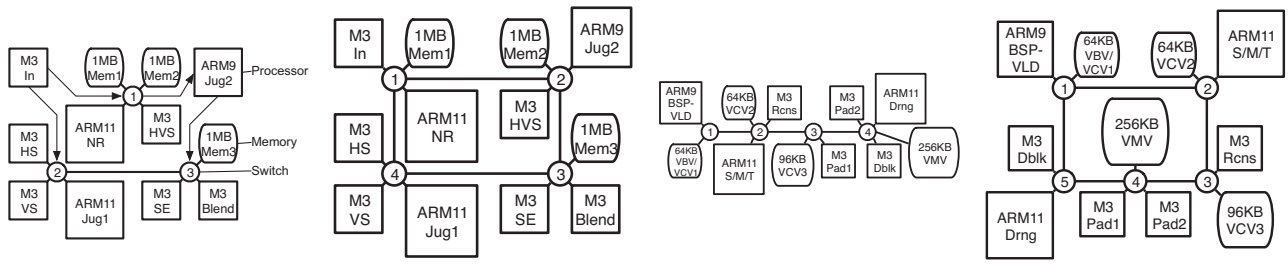
GapE-V [9] is presented in Algorithm 2. In contrast to SAR, GapE-V does not permanently remove any design from consideration. Designs are sampled sequentially. Let T_i denote the number of times design i has been sampled so far and $\hat{\mu}_i$ and $\hat{\sigma}_i$ denote the empirical mean and variance of design i. The designs are sorted according to their empirical mean. For each design, the gap Δ_i is calculated in similar fashion to how it was computed in the SAR algorithm. In particular, let i^* denote the m-th best arm and i_* denote the $m+1$-th best arm. For designs in the top-m empirically best designs, the gap $\Delta_i = \mu_i - \mu_{i_*}$. For designs that are not in the top-m designs, the gap $\Delta_i = \mu_{i^*} - \mu_i$.

Using the gap for each arm, GapE-V assigns an index

$$-\Delta_i + \sqrt{\frac{2a\hat{\sigma}_i}{T_i}} + \frac{7ab}{3(T_i - 1)},$$

where a and b are tunable parameters. b is selected to match the maximum value of the observed rewards. In our experiments, we set it to 15 years. GapE-V then samples the design with the highest index, reflecting a trade-off in exploitation (Δ_i) and exploration (the other terms). At the end of n samples, the designs with the top-m empirical means are selected.

For $m = 1$ and $a < \frac{8}{9}\frac{n-2D}{H^\sigma}$ (where H^σ is a measure of the complexity of the GapE-V algorithm), it is shown in [9] that the probability that the set of designs selected by GapE-V algorithm is *not* the set of top-m designs is upper bounded by $6nD \exp\left(-\frac{9a}{64 \times 64}\right)$. When $a = \frac{8}{9}\frac{n-2D}{H^\sigma}$, this bound equals



(a) MWD3S architecture (b) MWD4S architecture (c) MPEG4S architecture (d) MPEG5S architecture

Fig. 2. The task mapping and architectures for the various benchmarks. The squares represent the processing elements and the rounded squares the memory elements [5]; circles are switches. The text inside squares and rounded squares indicate tasks mapped to those elements.

$6nD\exp\left(-\frac{1}{8\times 64}\frac{n-2D}{H^{\sigma}}\right)$, which has is similar to the bound on the error probability of SAR algorithms. Ideally, we want to set a close to $\frac{8}{9}\frac{n-2D}{H^{\sigma}}$, but H^{σ}, like $H_2^{(m)}$ above, depends on the complexity of the distribution and hardness of the problem. [9] proposed an adaptive algorithm to iteratively estimate H^{σ} and tune a based on that estimate accordingly. We refer the reader to [9] for details.

C. Discussion

Both SAR and GapE-V algorithms may erroneously select a design that is not in top-m. The most likely designs to be misclassified are those whose means are closer to the mean of the m-th best design. Let's call this set $\mathcal{C}$. To avoid such a misclassification, both algorithms spend more samples for designs in $\mathcal{C}$ but they do so using different mechanisms.

In SAR, sampling time is split into phases. In each phase we spend an equal number of samples to improve the estimate of all active designs. At the end of the phase, the design with the largest gap relative to the m-th design is discarded. It is unlikely that the discarded design belongs to $\mathcal{C}$. Thus, the designs in $\mathcal{C}$ are sampled more because they are active for a larger number of phases. In contrast, in GapE-V the gap Δ_i for designs in $\mathcal{C}$ is close to zero, while the gap Δ_i for designs not in $\mathcal{C}$ is large. Since the gap Δ_i contributes negatively to the index, the designs in $\mathcal{C}$ will typically have a higher index than those not in $\mathcal{C}$ and will, therefore, be sampled more often.

IV. Methodology

We assume that when performing design space exploration for slack allocation, a number of designs are being evaluated concurrently; in this case, the objective is to identify the top designs in the population under consideration. This is the basic strategy of a number of meta-heuristics, including genetic algorithms, and ant colony optimization, which maintain a population of designs. These approaches have been previously employed to estimate lifetime [1], [6]. A greedy heuristic in the literature also identifies batches of designs to consider [5].

A. Lifetime Estimation

The lifetime of an MPSoC system depends on the failure time of its components and whether or not the failure is survivable. To determine component failure time, we model four dominant failure mechanisms, each as a separate random variable: electromigration (EM), time dependent dielectric breakdown (TDDB), stress migration (SM), and thermal cycling (TC) [4]. Each of these failure mechanisms are assumed to follow a log-normal distribution and can be parameterized with μ and σ. From a study in collaboration with IBM, it was determined that $\sigma = 0.5$ provides a good fit [4]; μ can be solved from the relationship $m = e^{\mu+\sigma^2/2}$ where m is the mean. The mean lifetime can estimated from empirical models from JEDEC [12]. Since the mean lifetime will vary between process technologies, a normalization constant is used. Each mechanism is initialized to have a failure time of 30 years at a temperature of $345K$, consistent with [4].

System lifetime is estimated by simulating the following for a large number of samples as determined by the algorithm:

1) For each component, sample from the failure mechanism distributions and find the minimum failure time among the working components
2) Remap the tasks from the failed component to the nearest working component with slack
3) Find the new operating temperature
4) Adjust time to failure
5) Repeat steps 2-4 until tasks cannot be remapped

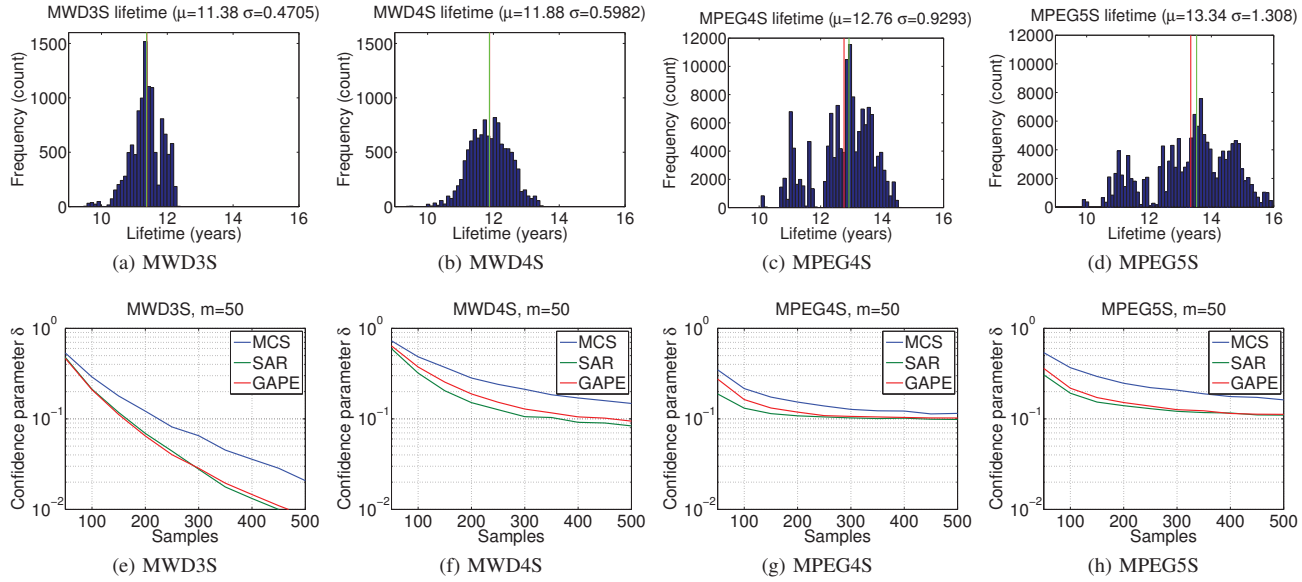
Temperature is determined using HotSpot [16], based on data sheet power values and floorplans generated by Parquet [17].

When a component fails, the system will continue to operate if the component workload can be remapped to one or more components with slack without violating workload and bandwidth requirements. If the component failure is not survivable then system failure has occurred, indicating the lifetime of that instance of the device. The mean time to failure (MTTF) is the average of all such failure times.

B. Workloads

The workloads we simulate for the slack allocation problem are multi-window display (MWD) and a MPEG decoder. MWD is an application which composites multiple video sequences into a single display [18]. MPEG is an implementation for the MPEG4 encoding standard [19]. For each of these workloads a number of processing components as well as memory components are allocated.

Each workload is configured on two possible system architectures. The MWD3S and MPEG4S (Figures 2a, 2c)



(a) MWD3S (b) MWD4S (c) MPEG4S (d) MPEG5S

(e) MWD3S (f) MWD4S (g) MPEG4S (h) MPEG5S

Fig. 3. The lifetime distribution of all benchmark and slack allocations are shown in (a)-(d). The red line indicates the mean, the green line, the median. (e)-(h) compare MAB with MCS choosing the best 50 out of 100 designs. The confidence parameter δ represents the probability of misclasification.

consist of the minimum number of switches to connect all the components together; in this case a switch failure is not survivable. In MWD4S and MPEG5S (Figures 2b, 2d) the switches are configured in a ring architecture so that one switch failure may be survivable [5].

The MWD system has six processors and three memories, and the MPEG system has seven processors and four memories. There are three types of processors and nine types of memories. The components can only be swapped with ones of higher capacity. In total there are 11,664 MWD configurations and 142,884 MPEG configurations. The size of this problem makes identifying the optimal design a challenge.

C. Algorithm Comparison Metric

For the purposes of testing MAB on each of the design spaces, 100 random designs were selected and the algorithms were used to identify the top set of designs. The true lifetimes were estimated by collecting 1 million samples, achieving a 95% confidence interval within 0.005 years. The true lifetime distributions of each benchmark are shown in Figures 3a-3d. We compare the sum of the lifetimes differences of the selected top m designs and the optimal set:

$$Pr\left[\sum_{i=1}^{m}\mu_i^* - \mathbf{E}\mu_{J(i)} > \epsilon\right] \leq \delta.$$

If the difference of the aggregated average lifetimes of the optimal set $\mu^*_{1..m}$ and the expected lifetime of the chosen set $\mu_{J(i)}$ using a policy $J(i)$ is greater than ϵ, then an identification error is said to have occurred. For this work, we use $\epsilon = 1$, which assumes an error tolerance of 1 year.

Using this metric, results are collected 100 times per set of 100 designs over 100 different sets of designs. The confidence parameter δ is the number of failures observed in the collected results, i.e., the probability of misidentification.

V. Results

The results for each of the four benchmarks are shown in Figures 3e-3h. As the number of samples increase, the confidence in identifying the top set of $m = 50$ designs out of 100 designs increases; $m = 50$ is chosen since identifying the top half of the designs is likely to be the most versatile result for use in other design space exploration algorithms (e.g. genetic algorithms). It is clear that both MAB algorithms outperform the baseline approach of uniform sampling (MCS): both algorithms reach a higher level of confidence compared to MCS for a given number of samples. The sensitivity of ϵ was tested over a range of $[0, 2]$ and MAB continued to reach higher confidence with fewer samples.

In Table I, the average savings obtained when reaching the same level of confidence as MCS at 500 samples per design is shown for selecting the top 20, 30, 40, and 50 designs out of 100. The obtained confidence parameter δ is also shown for an error of $\epsilon = 1$. In the worst case, when selecting the top 50 designs ($m = 50$), SAR requires 1.49-3.57× fewer samples and GapE-V 1.45-2.86× fewer. In the best case, when a smaller set of 20 designs is selected ($m = 20$), the improvements are more pronounced. SAR is able to achieve a reduction of 1.92-5.26× and GapE-V 1.72-3.57×.

There is a larger reduction in required samples when a smaller m is used; $m = 50$ is a more difficult problem than $m = 20$ for MAB algorithms. This is attributed to how the samples are distributed. When $m = 50$, roughly half of the samples would be spent on the top half and bottom half (assuming a symmetric distribution in the average case). When $m = 20$, the amount of samples near the boundary does not change significantly. Since there are fewer designs in the top set, the samples are redistributed toward bad designs, reducing variance and minimizing the probability of error.

It is worth noting that the δ that is achieved at 500 samples

TABLE I
SAMPLE SAVINGS FROM MAB ALGORITHMS FOR CONFIDENCE EQUIVALENT TO MCS WITH 500 SAMPLES.

	m=20			m=30			m=40			m=50		
Benchmark	δ	SAR	GapE-V	δ	SAR	GapE-V	δ	SAR	GapE-V	δ	SAR	GapE-V
MWD3S	0.002	1.92×	1.72×	0.003	1.72×	1.71×	0.009	1.79×	1.67×	0.021	1.49×	1.45×
MWD4S	0.071	3.33×	2.13×	0.112	2.96×	2.07×	0.180	2.54×	2.01×	0.148	2.44×	1.92×
MPEG4S	0.120	3.57×	2.70×	0.101	3.52×	2.48×	0.202	3.60×	2.43×	0.115	3.33×	2.27×
MPEG5S	0.052	5.26×	3.57×	0.083	4.07×	3.05×	0.292	3.70×	3.07×	0.162	3.57×	2.86×

is in general smaller for smaller m than larger m. One of the factors worth considering is that the chosen metric uses the sum of errors in the top m arms so when there are more arms, the compounded errors will be larger. There are a few exceptions in our results which break this trend, such as $m \in \{20, 40\}$, for the MPEG4S benchmark. This is likely attributed to the effects of the underlying design space distribution, since it will inherently affect the difficulty of the problem. Exploring the effects of different design spaces on MAB performance is subject of future work.

A. Discussion

Although SAR consistently outperforms GapE-V as an MAB algorithm for lifetime distributions, the choice of algorithm depends on the application. Recall that SAR discards designs from sampling so it cannot recover from earlier misidentification errors. Furthermore, SAR does not keep track of sample variance whereas GapE-V does; as such, SAR is not equipped to handle problems where new arms are introduced, e.g., in genetic algorithms.

The stated improvements are presented under the assumption that sample evaluation dominates the overhead of the MAB algorithms. However, this assumption may not be true if the number of arms to choose from increases. For this study, it was determined that the a set size of 100 was a fair comparison point where the overhead was no more than 20% for SAR and 25% for GapE-V.

The complexity of sampling for all the algorithms is $ND \times T_{sample}$ where N is the number of samples per design, D is the number of designs, and T_{sample} is the time required for sample evaluation. For MCS, there is no selection overhead. The overhead of SAR is $T_{sort}(D) + T_{sort}(D-1) + ... + T_{sort}(m) < D \times T_{sort}(D)$ where $T_{sort}(x)$ is the time to sort x designs and make a decision what to sample next. The overhead of GapE-V is $ND \times T_{sort}(D)$ since each sample requires a decision involving a sort on D designs. In general, the MAB algorithms are $\mathcal{O}(D^2)$ due to the selection process. Ultimately, the number of designs and samples to use will depend on the problem and will require a profiling of the exact implementation.

From this analysis, it is apparent that MAB does not scale well for a large number of designs; however, design space exploration techniques tend not to simultaneously evaluate a large numbers of designs. Further, GapE-V is not recommended when there are a large number of samples N since the decision process is made per sample.

VI. CONCLUSION

We have demonstrated an application of MAB to the slack allocation design space exploration problem. In the best case, when selecting the top 10 out of 100 designs on the MPEG5S benchmark, a 5.26× reduction in MCS samples was observed using SAR. In the worst case, when selecting the top 50 out of 100 designs using GapE-V on the MWD3S benchmark, a reduction of 1.45× is observed. Similar results are expected whenever sample evaluation time is much larger than the time to select the next design to sample. Overall, MAB show a consistent improvement over a uniform sampling strategy.

ACKNOWLEDGMENT

This research was supported by the McGill University Faculty of Engineering Chwang-Seto Faculty Scholar Research Award. Computing resources were provided by the Canada Foundation for Innovation and Calcul Québec.

REFERENCES

[1] A. S. Hartman *et al.*, "A case for lifetime-aware task mapping in embedded chip multiprocessors," *CODES+ISSS*, 2010.
[2] L. Huang *et al.*, "Lifetime reliability-aware task allocation and scheduling for MPSoC platforms," *DATE*, 2009.
[3] J. Srinivasan *et al.*, "Lifetime reliability: Toward an architectural solution," *IEEE Micro*, vol. 25, no. 3, 2005.
[4] J. Srinivasan *et al.*, "Exploiting structural duplication for lifetime reliability enhancement," *ISCA*, 2005.
[5] B. H. Meyer *et al.*, "Cost-effective lifetime and yield optimization for noc-based mpsocs," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 19, no. 2, pp. 1–33, Mar. 2014.
[6] A. Das *et al.*, "Combined DVFS and mapping exploration for lifetime and soft-error susceptibility improvement in MPSoCs," *DATE*, 2014.
[7] S. Bubeck *et al.*, "Regret analysis of stochastic and nonstochastic multi-armed bandit problems," *Foundations and Trends in Machine Learning*, vol. 5, no. 1, pp. 1–122, 2012.
[8] S. Bubeck *et al.*, "Multiple identifications in multi-armed bandits," *ICML*, vol. 28, 2013.
[9] V. Gabillon *et al.*, "Multi-Bandit Best Arm Identification," in *Advances in Neural Information Processing Systems 24*, 2011.
[10] J. Srinivasan *et al.*, "The case for lifetime reliability-aware microprocessors," *ISCA*, 2004.
[11] Z. Gu *et al.*, "Application-specific MPSoC reliability optimization," *VLSI Systems*, vol. 16, no. 5, 2008.
[12] Jedec Solid State Technology, "Failure mechanisms and models for semiconductor devices," *JEDEC Publication JEP122-B*, 2003.
[13] Y. Xiang *et al.*, "System-level reliability modeling for MPSoCs," in *CODES+ISSS*, 2010.
[14] T. L. Lai *et al.*, "Asymptotically Efficient Adaptive Allocation Rules," *Adv. Appl. Math.*, vol. 6, 1985.
[15] P. Auer *et al.*, "Finite-time Analysis of the Multiarmed Bandit Problem," *Machine Learning*, vol. 47, no. 2/3, 2002.
[16] K. Skadron *et al.*, "Temperature-aware microarchitecture," *ISCA*, 2003.
[17] S. Adya *et al.*, "Fixed-outline floorplanning: Enabling hierarchical design," *VLSI Systems*, 2003.
[18] E. G. T. Jaspers *et al.*, "Chip-set for video display of multimedia information," *IEEE Transactions on Consumer Electronics*, no. 3, 1999.
[19] J. K. Reissmann *et al.*, "The MPEG-4 video coding standard-a VLSI point of view," in *SIPS*, 1998.