Metrics for Lifetime Reliability

Pradeep Ramachandran[†], Sarita Adve[†], Pradip Bose[‡], Jude Rivers[‡] and Jayanth Srinivasan[†1]

[†]Department of Computer Science
University of Illionis at Urbana Champaign
{pramach2,sadve}@uiuc.edu

[‡]IBM T.J. Watson Research Center
Yorktown Heights, NY
{pbose,jarivers}@us.ibm.com

UIUC CS technical report UIUCDCS-R-2006-2762
August, 2006

Abstract

This work concerns appropriate metrics for evaluating microarchitectural enhancements to improve processor lifetime reliability. The most commonly used reliability metric is mean time to failure (MTTF). However, MTTF does not provide information on the reliability characteristics during the typical operational life of a processor, which is usually much shorter than the MTTF. An alternative to MTTF that provides more information to both the designer and the user is the time to failure of a small percentage, say n%, of the population, denoted by t_n. Determining t_n, however, requires knowledge of the distribution of processor failure times which is generally hard to obtain. In this paper, we show (1) how t_n can be obtained and incorporated within previous architecture-level lifetime reliability tools, (2) how t_n relates to MTTF using state-of-the-art reliability models, and (3) the impact of using MTTF instead of t_n on reliability-aware design.

We perform our evaluation using RAMP 2.0, a state-of-the-art architecture-level tool for lifetime reliability measurements. Our analysis shows that no clear relationship between t_n and MTTF is apparent across several architectures. Two populations with the same MTTF may have different t_n, resulting in a difference in the number of failures in the same operational period. MTTF fails to capture such behavior and can thus be misleading. Further, when designing reliability-aware systems, using improvements in MTTF as a proxy for improvements in t_n can lead to poor design choices. Depending on the application and the system, MTTF-driven designs may be over-designed (incurring unnecessary cost or performance overhead) or under-designed (failing to meet the required t_n reliability target).

1 Introduction

1.1 Motivation

An important goal for processor designers is to ensure long-term or "lifetime" reliability against hard failures. Unfortunately, aggressive CMOS scaling coupled with increased on-chip temperatures is accelerating the onset of wear-out or aging related hard failures due to effects such as electromigration, gate oxide breakdown, and negative bias temperature instability (NBTI) [4, 21].

Until recently, in the broad general-purpose processor market, lifetime reliability was largely addressed at the device level, without much help from microarchitects. Although such an

[1]Jayanth Srinivasan graduated from the University of Illinois at Urbana-Champaign in May, 2006

approach tackles the problem at its root, it generally cannot exploit high-level application and system behavior. Recently, several academic and industry researchers have suggested exploring microarchitecture level solutions for the lifetime reliability problem [6, 7, 8, 19, 20, 22, 25]. Such solutions can be application-aware and open up new cost-performance reliability points for general-purpose processors, which were previously unavailable.

Appropriate metrics and models are essential to enable effective microarchitecture-level lifetime reliability research. The most commonly used metric for reliability is mean time to failure (MTTF), which is the expected lifetime of the given population of processors. Srinivasan et al. recently proposed RAMP, the first generation of microarchitecture level models to calculate workload-specific MTTF of a processor for various wear-out failure mechanisms [20, 22].[2]

A key limitation of the MTTF metric is that it is a single summary statistic that does not provide insight on the failure behavior during the useful lifetime of most processors, as illustrated by the following example.

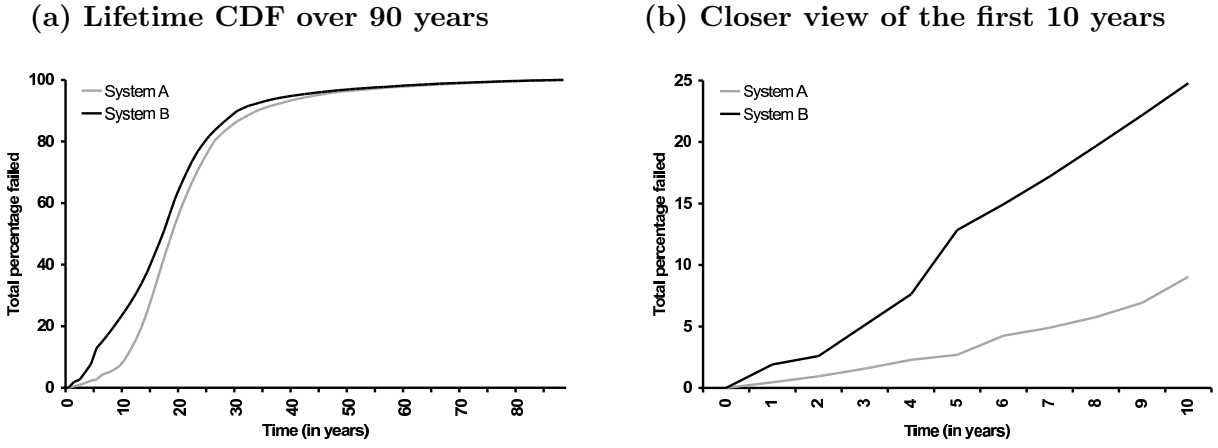


Figure 1: MTTF can be misleading. Part (a) shows the cumulative distribution function (CDF) of the lifetimes for two product lines that have the same MTTF of 20 years. Part (b) is a close view of the same graph for the first 10 years. The distributions are different although the MTTFs are identical. If the user expects to use the system for around 5 years, system A is preferable to system B as it sees fewer failures within 5 years. The MTTF metric hides such information.

Figure 1 shows the cumulative distribution function (CDF) of the lifetimes for two simulated product lines (the methodology for this simulation is explained in Section 4). The MTTF for

[2]An alternate industry metric is based on failure rate, measured in units of FITs or failures per billion hours of operation. It is common practice to quote a constant FIT rate and use the relationship $FITs = \frac{10^9}{MTTF}$. However, the constant failure rate assumption and the reciprocal relationship between MTTF and failure rate only hold for failure mechanisms with exponential lifetime distributions [23]. It is widely accepted that wearout based failures do not follow exponential lifetime distributions and have failure rates that change with time [2, 23]. We assume the more realistic non-exponential model and so do not use FITs.

both the product lines is 20 years, a typical MTTF target for processor designers [16]. Figure 1(a) shows the CDF of the lifetimes of the product lines over 90 years, while Figure 1(b) shows a closer view of the first 10 years. The figures show the onset of failures for system A is far slower than that for system B in the first 10 years. For example, in 5 years, only about 3% of the processors in system A have failed, while about 13% of the processors in system B have failed. For users who expect to upgrade within 5 years, system A is clearly a better choice than system B. The MTTF metric, however, fails to make this distinction.

Designers typically target a much higher MTTF than the expected operational life of a processor so that most customers do not see failures during the operational life. However, the MTTF metric itself does not reveal the probability of failure over the operational life of the processor, as Figure 1 illustrates. This limitation of the MTTF metric has an impact on both the customer and the vendor.

From the customer's perspective, paying a premium for a high MTTF is not useful if it does not translate to a commensurately lower probability of failure during the expected operational life. In Figure 1, the customer would rather choose system A, but has no way to do so if the vendor only supplies the metric of MTTF.

From the vendor's perspective, apart from the number of disgruntled customers, warranty and replacement costs also depend on the number of failures during the expected operational life. Again, as Figure 1 illustrates, two designs with the same MTTF may have different numbers of failures in the operational life, with significantly different maintenance costs. Reliability models that only provide MTTF cannot distinguish between such designs.

1.2 Beyond MTTF

This paper explores an alternate metric that addresses the above limitations of MTTF. We consider the metric of time to failure of n% of the population, denoted t_n, where n is a relatively small number [16, 24]. For example, $t_5 = 10$ years implies that 5% of the processor population will fail in 10 years; conversely, the probability of failure for a given processor over 10 years is 0.05. For consistency, we henceforth denote MTTF as t_{mean}. In some literature, the metric t_n is also denoted by L_n (where L stands for lifetime) [24] and L_{10} is a relatively common metric in many mechanical industry markets [18]. The function t_n or L_n is also referred to as the percent point function or the inverse distribution function (since it is the inverse of the

cumulative distribution function of processor lifetime) [1].

From the vendor's point of view, a t_n driven design must ensure that t_n exceeds the anticipated useful operational life for some acceptable failure probability ($0.01 \times n$). Both n and the desirable t_n depend on the market, including factors such as customer satisfaction, warranty costs, and anticipated useful operational life. For example, acceptable values of n (probability of failure) for the desktop market are likely to be higher than for the server market. Providing t_n data for multiple values of n would allow the customer to consider the value most appropriate for their use.

From the customer's point of view, t_n indicates that a given processor will fail with probability $0.01 \times n$ within t_n years. This gives the customer more information than with MTTF for comparing products. For example, it may not be worthwhile for a desktop user to pay a cost premium to obtain $t_{10} > 10$ years, but it may be worthwhile to pay a cost premium to obtain $t_5 > 5$ years.[3]

1.3 Contributions

This paper makes two contributions.

- **Computing t_n.** First, we show how the previous MTTF (t_{mean}) based RAMP model can be used in a straightforward way to measure t_n [22].

  In general, computing t_n (vs. t_{mean}) of a system is not straightforward since it requires information about the distribution of failures, which is inherently hard to obtain. A key relevant exception is for the sum-of-failure-rates (SOFR) model assumption widely used in industry [14]. SOFR assumes that the lifetime distribution of each system component is exponential and the system is a series failure system. With these assumptions, it can be shown that system t_{mean} is proportional to t_n. Previous studies, however, have shown that the exponential assumption is inaccurate for wearout failures where the failure rate is not constant over time [1]. Prior work has therefore suggested the use of more complex distributions (e.g., lognormal) for individual components [1, 21]. Further, reliability-aware systems use redundancy which also violates the series-failure assumption of SOFR.

[3] An alternative metric to t_n is its inverse – the percentage of processors failed (n) within a given time (t). Arguably, this metric may be more insightful for customers who may be able to better anticipate their useful operational period (t) and wish to compare the probability of failure during this time. This metric is the same as the CDF of the lifetimes, which is also equivalent to 1 - reliability. Our survey of the literature showed more usage of t_n than its inverse, and so we report results on t_n here.

Analytically computing the lifetime for a multicomponent processor with lognormally distributed component lifetimes in series-parallel-standby organizations is difficult. The state-of-the-art architecture-level model, RAMP 2.0, therefore uses Monte-Carlo simulations to determine t_{mean} using Min-Max computations on underlying lognormal lifetime distributions for individual components [22]. We make the observation that these same Monte-Carlo runs can be used to provide the distribution of the lifetimes and, consequently, values of t_n.

- **Implications of t_n vs. t_{mean} based design for architects.** Second, we provide empirical data showing the impact of the use of t_n vs. t_{mean} on reliability-aware microarchitecture design, assuming lognormal lifetime distributions for individual components. To the best of our knowledge, this is the first detailed quantitative comparison of the two metrics for microarchitectural design.

  We use RAMP 2.0 with 16 SPEC applications for our experiments. We find that the relationship between t_n and t_{mean} is complex for underlying lognormal distributions. Most significantly, when considering the benefits of reliability enhancing techniques, using the improvement in t_{mean} as an estimate of the improvement in t_n can lead to poor design choices. Depending on the application and reliability-enhancement technique used, the resulting system can be over-designed (i.e., it is unnecessarily expensive or low performance) or under-designed (i.e., it does not meet the intended reliability target).

  For example, our results show that for a system running a workload similar to the SpecInt application crafty, the lowest overhead in performance to achieve a 1.4X benefit in t_{mean} is 1.4X (for the reliability-enhancing techniques studied here). Although a system with this performance loss achieves a similar benefit in t_n, a performance overhead of 1.25X would have sufficed to obtain that benefit in t_n. Thus, the t_{mean} driven design incurs an unnecessary performance overhead. Our results show other cases where the t_{mean} driven design does not meet the reliability target for t_n. These examples clearly illustrate the significant limitations of using t_{mean} as the evaluation metric.

The difference between t_n and t_{mean} driven designs for underlying lognormal distributions is in marked contrast to the case of exponential distributions. In the latter case, the linear relationship between t_n and t_{mean} implies that a given reliability enhancing technique has the

same benefit in t_{mean} and t_n; therefore, results from a t_{mean} driven methodology can be used as a proxy for a t_n driven methodology when comparing different reliability techniques. Our results show that this is not the case for the more realistic complex non-exponential distributions that characterize wearout!

As lifetime reliability becomes important, architecture-level tools to measure lifetime reliability will become increasingly important. In prior work, RAMP 2.0 improved on RAMP 1.0 by incorporating lognormal lifetime distribution for processor components [22]. This paper takes the next step to enhance RAMP 2.0 with a metric that is more meaningful than MTTF for complex non-exponential lifetime distributions. Admittedly, current tools are still preliminary and make significant assumptions [22]. Nevertheless, this paper takes an important step forward to establish appropriate metrics. Our experimental results question the common industry practice of designs targeting t_{mean} for wearout failures that have complex lifetime distributions and show quantitatively that the combination of correct metric and underlying lifetime distribution has a significant impact on reliability-aware architecture.

2 Background

Section 2.1 provides background on RAMP [20, 22], a state-of-the-art architecture level reliability model and tool we use in this study. Section 2.2 provides background on the two reliability-enhancing techniques we study – structural duplication (SD) and graceful performance degradation (GPD) [22].

2.1 RAMP

RAMP [20] models the following wear-out failure mechanisms in processors: electromigration (EM), stress migration (SM), time dependent dielectric breakdown (TDDB), thermal cycling (TC) and negative bias temperature instability (NBTI) [5]. It works in conjunction with a timing simulator (in our case, Turandot [17]), a power model (PowerTimer [10]), and a temperature model (HotSpot [19]).

RAMP starts with state-of-the-art device level analytical models for each of the above failure mechanisms. These models compute MTTF as a function of various operating parameters such as temperature, voltage, and utilization, assuming steady state operating conditions. RAMP abstracts these models at the architecture level. Much like previous power and temperature

models [11, 19], RAMP divides the processor into a few structures - ALUs, register file, branch predictor, caches, load-store queue, and instruction window. Every few cycles of the timing simulator, RAMP applies the analytic models to each structure as an aggregate, to calculate the "instantaneous" MTTF for the current operating conditions (which are derived through the timing, power, and temperature simulators). It then averages the instantaneous MTTFs to give a net MTTF for a given structure and given failure mechanism for the specified workload.[4]

To obtain the overall reliability of the processor, RAMP needs to combine the MTTFs across different failure mechanisms across different structures. This requires knowledge of the lifetime distributions of the different failure mechanisms, which is generally difficult. RAMP 1.0 used a simple but commonly used and potentially inaccurate assumption while RAMP 2.0 used a potentially more accurate assumption. We describe both below since we use both in this paper.

Combining failures from different mechanisms and different structures with RAMP 1.0.

RAMP 1.0 uses the industry standard Sum-of-Failure-Rates (SOFR) model, which makes two assumptions: (1) the processor is a series failure system - in other words, the first instance of any structure failing due to any failure mechanism causes the entire processor to fail and (2) each individual failure mechanism has a failure rate that stays constant in time, or equivalently, each failure mechanism has an exponentially distributed lifetime.

The above two assumptions imply [23]: (1) the lifetime distribution of the processor is also exponential (i.e., constant failure rate) and the failure rate of the processor is the sum of the failure rates of the individual structures due to individual failure mechanisms, and (2) the MTTF of the processor, $MTTF_p$, is the inverse of the total (constant) failure rate of the processor, λ_p (true for exponentially distributed lifetimes). Hence,

$$MTTF_p = \frac{1}{\lambda_p} = \frac{1}{\sum_{i=1}^{n}\sum_{j=1}^{f}\lambda_{ij}} \quad (1)$$

where λ_{ij} is the failure rate of the i^{th} structure due to the j^{th} failure mechanism and n is the number of structures and f, the number of failure mechanisms.

Combining failures from different mechanisms and different structures with

[4]The assumption underlying the averaging over time is analogous to the SOFR assumption described for RAMP 1.0 below since SOFR averages over space.

RAMP 2.0.

The SOFR model used by RAMP 1.0 assumes a constant failure rate for a given failure mechanism and structure. This is clearly inaccurate as a typical wear-out based failure mechanism starts with a low failure rate at the beginning of the component's lifetime and has an increasing failure rate as the component ages. Further, the series failure assumption of the SOFR model is also inaccurate if the processor supports redundant structures (as with the two reliability enhancing techniques studied here). In these cases, the failure of a single component does not imply the failure of the processor since the redundant component takes over.

RAMP 2.0 addresses the above two limitations of the SOFR model used in RAMP 1.0. First, instead of the exponential distribution, RAMP 2.0 assumes lognormal lifetime distributions for individual structures and failure mechanisms. Lognormal distributions have shown to be more accurate for degradation processes common to semiconductor materials due to the multiplicative degeneration argument [1].

Second, since lognormal distributions are hard to deal with analytically, RAMP 2.0 uses the Monte Carlo simulation method to calculate the full processor MTTF from the lognormal lifetime distributions of individual structures and failure mechanisms. The mean of these individual distributions is provided by the timing simulator run similar to RAMP 1.0 and the variance σ is set at 0.5 (used for wear-out failures in prior work [1, 3]). This method does not require the series failure assumption. Series, parallel redundant, and/or standby redundant systems can be modeled using a MIN-MAX analysis on the individual component lifetimes in a given Monte Carlo trial.

2.2 Reliability-Enhancing Techniques

We examine two reliability-enhancing methods based on structure-level redundancy explored by Srinivasan et al. [22].

In the first method, referred to as **Structural Duplication** (SD), certain redundant microarchitectural structures are added to the processor, and these are designated as *"spares"*. Spare structures can be turned on during the processor's lifetime when the original structure fails. Hence, in a situation where the processor would have normally failed, the spare structure extends the processor's lifetime. With SD, the processor fails only when a structure with no spare fails, or if all available spares for a structure have also failed. The main function of the

spares is to increase the reliability and not to enhance the performance; therefore, they are power gated and not used in the beginning of the processor's life.

The second reliability-enhancing method is **Graceful Performance Degradation** (GPD). This method allows the processor to exploit existing microarchitectural redundancy for reliability. Modern processors have replicated structures so as to increase the performance for applications with heavy parallelism. The replicated structure, however, is not necessary for functional correctness. If the replicated structure fails in the course of a processor's lifetime, the processor can shut down the structure and can still maintain functionality, thereby increasing lifetime. With GPD, the processor fails only when a structure with no redundancy fails or when all redundant structures of a given type fail.

Both SD and GPD incur overheads while increasing processor reliability. In the case of SD, extra processor die area is required to introduce the spares incurring a cost overhead. In the case of GPD, a performance loss is incurred to improve the reliability. We will explore SD and GPD configurations of various overheads.

3 Incorporating t_n into RAMP

Mathematically, t_n is defined as the time t at which

$$F(t) = \frac{n}{100} \tag{2}$$

where $F(t)$ is the cumulative distribution function (CDF) for the processor lifetimes [23].

We calculate t_n with RAMP 2.0 as follows. As mentioned in Section 2, RAMP 2.0 uses a Monte Carlo simulation to generate the processor MTTF from the MTTFs of the individual structures and failure mechanisms generated from the timing simulator run. Each Monte Carlo trial generates a value for the time to failure for each individual structure for each failure mechanism. This time is generated from the corresponding lognormal distribution, which is fully specified by the mean (MTTF) provided by the timing simulator run and variance σ of 0.5 as mentioned earlier.

Using these lifetime values, RAMP 2.0 performs a MIN-MAX analysis across all structures to get the lifetime for the full processor. For example, the base processor with no redundancy is a series system. The lifetime of each structure is the minimum of the time to failure from

Technology Parameters	
Processor technology	65nm
V_{dd}	1.0V
Processor frequency	2.0GHz
Processor size (without L2)	3.6mm × 3.2mm
Leakage power density at 383K	0.60 W/mm^2
Base Processor Parameters	
Fetch/finish rate	8 per cycle
Retirement rate	1 dispatch group (=5,max) per cycle
Functional units	2 Int, 2 FP, 2 Load-Store, 1 Branch, 1 LCR
Integer FU latencies	1/7/35 add/multiply/divide (pipelined)
FP FU latencies	4 default,12 div (pipelined)
Reorder buffer size	150
Register file size	120 integer, 96 floating point
Memory queue size	32 entries
Base Memory Hierarchy Parameters	
Data L1	32KB
Instruction L1	32KB
L2 (Unified)	2MB

Table 1: Base processor simulated.

each failure mechanism. The lifetime of the full processor is the minimum of the lifetimes for each structure. Details on the analysis for the SD and GPD systems are provided in [22].

Thus, each Monte Carlo trial generates the lifetime for one processor sample. RAMP 2.0 averages these lifetimes over a large number of trials (typically 10^7 trials) to provide the MTTF of the processor.

The results of the Monte Carlo trials of RAMP 2.0 also directly provide information on the cumulative distribution function of the processor lifetimes. To calculate t_n, we therefore simply find the smallest processor lifetime value such that n% of the trials lie within that time.

4 Experimental Methodology

We perform our evaluations using RAMP 2.0 (extended to calculate t_n) coupled with performance, power, and temperature simulators and models, using a methodology similar to previous RAMP-based work [20, 21, 22].

4.1 Base Processor

The base processor we used for our simulation is a 65nm, out-of-order, 8-way super-scalar processor, conceptually similar to a single core POWER4 processor [15]. The 65nm processor

parameters were derived from scaling down parameters from the 180nm POWER4 processor [21]. Although we model the performance impact of the L2 cache, we don't model the reliability as its temperature is much lower than the processor core [15], resulting in very few L2 cache failures. Table 1 summarizes the base processor modeled.

4.2 Simulation Environment

For timing simulations, we use Turandot, a trace-driven research simulator developed at IBM's T.J.Watson Research Center [17]. Turandot was calibrated against a pre-RTL, detailed, latch-accurate processor model. Thus, despite its trace-driven nature, the extensive validation methodology provides high confidence in its results[17].

For the power model, we use PowerTimer, a tool-set built around Turandot [9]. The power values from PowerTimer are fed into HotSpot to evaluate the temperatures of various components on chip [19]. RAMP 2.0 uses the temperature estimates from HotSpot and utilizations from Turandot to calculate processor MTTF and t_n for the simulated workloads, as described in Section 3. Power, temperature, and reliability samples are computed at a granularity of 1 μs. The power, temperature, and reliability models all work at the granularity of architecture-level structures. For our experiments, the processor is divided into the following seven structures: floating point unit (FPU), fixed point unit (FXU), instruction decode unit (IDU), instruction scheduling unit (ISU), load store unit (LSU), instruction fetch unit (IFU) and the branch prediction unit (BXU).

4.3 Workloads

We evaluate 16 SPEC2000 benchmarks (8 SpecInt + 8 SpecFP). The SPEC2000 trace repository used in this study was generated using the Aria trace facility in the MET toolkit [17] using a full reference input set. Sampling was used to limit the trace length to 100 million instructions per program. The sampled traces have been validated with the original full traces for accuracy and correct representation [13].

4.4 Reliability Enhancements

As mentioned earlier, we study two reliability enhancements to the base processor - structural duplication (SD) and graceful performance degradation (GPD). Our methodology is identical

Group	Units in group	Area	Original configurations	Degraded Configuration
1	FPU	0.96	2 FP units + 96 FP regs	1 FP unit + 48 FP regs
2	FXU	0.96	2 Int units + 120 Int regs	1 Int unit + 60 Int regs
3	BXU + IFU	2.56	16K BHT entries + 32KB ICache	8K BHT entries + 16KB ICache
4	LSU	4.0	2 L/S queues + 32KB DCache	1 L/S queue + 16KB DCache
5	IDU + ISU	3.04	128 instruction window	N/A

Table 2: Configurations for SD and GPD. Groups 1-5 are replicated in SD. Groups 1-4 are allowed to degrade in GPD. IDU + ISU is not allowed to degrade in GPD.

to that used in [22].

For SD, the seven structures of the processor (Section 4.2) are grouped into five groups, each of which can be duplicated for standby or spare redundancy. The cost overhead of this area increase is evaluated using the Hennessey-Patterson die cost model [12], and reported as the ratio of the cost of the SD and base processors. For GPD, the structures are grouped into four groups, each of which is allowed to degrade to half performance without the entire processor failing. The net performance overhead of a GPD configuration is reported as the slowdown incurred by the fully degraded version of that configuration for the entire application (relative to the base configuration).[5] SD with a cost overhead of X and GPD with a performance overhead of Y are denoted by SD-X and GPD-Y respectively. In each case, we use the system configuration that gives the most benefit for the specified overhead for the given application (this turns out to be the same configuration for benefits in t_{mean} and all t_n for a given application).

Table 2 (from [22]) shows the area of each structure on chip, and the original and degraded configurations.

5 Results

5.1 Relationship between t_{mean} and t_n

Since t_{mean} (MTTF) is widely used to evaluate reliability, we first explore if t_{mean} can be used to predict t_n. As mentioned in Section 1, with the widely used (but incorrect) SOFR assumption,

[5]The performance of the fully degraded version of a given GPD configuration is the guaranteed performance. In reality, the early part of the lifetime will see better performance. Srinivasan et al. report both the guaranteed and the actual performance [22]. We chose to report only the former since the method of counting this overhead is orthogonal to the point of this paper.

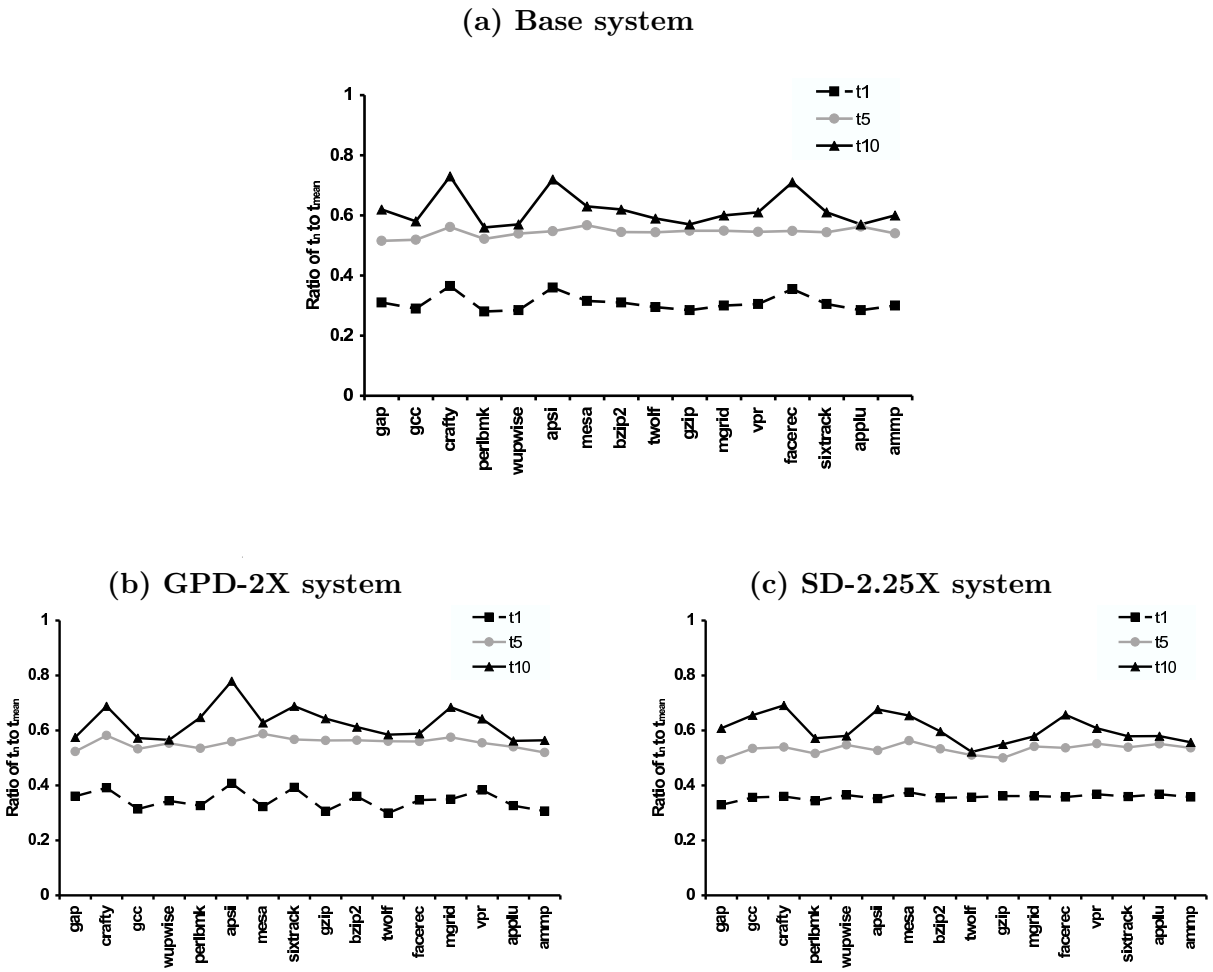


Figure 2: Ratio of t_n to t_{mean}. The figure shows the ratios of t_n to t_{mean} for different applications on (a) Base, (b) GPD-2X, and (c) SD-2.25X systems. The applications in each system are ordered in increasing order of t_{mean}. The ratio of t_n to t_{mean} varies and is not a constant for several cases.

system t_{mean} is proportional to t_n.[6] We investigate if such a proportionality relationship exists even for our systems, where the underlying structure lifetimes are lognormally distributed and the system employs redundancy. If such a relationship were to exist, then t_{mean} would be a reasonable proxy for t_n; e.g., when comparing the benefits of two reliability-enhancing techniques, the improvement in t_{mean} would be representative of the improvement in t_n.

Figure 2 shows the ratio of t_n to t_{mean} for different values of n for each application. Parts (a), (b), and (c) show this data for the base, SD-2.25X, and GPD-2X systems respectively. The figures show several cases where the ratio of t_n to t_{mean} is not a constant, and so assuming t_n proportional to t_{mean} could lead to erroneous results. To provide some quantitative measure of the error in estimating t_n as a constant factor times t_{mean}, we computed estimates for t_n as follows. For each n, we computed $t_n = \mathrm{k} \times t_{mean}$ where k is the average t_n/t_{mean} for the Base system for that n. We then determined the error in this approximation.

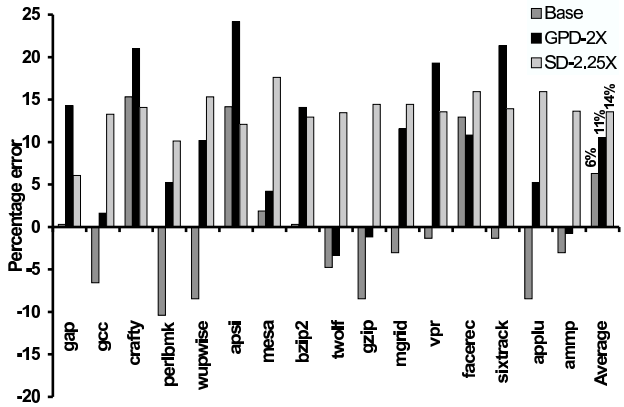
Figure 3 shows that the percentage error in the above approximation is significant for several cases for t_1 and t_{10}. Focusing on t_1, we see that four applications (crafty, apsi, vpr and sixtrack) show more than 19% (absolute) error, the errors across all the applications are both positive and negative, and the overall range of errors is large (from -10% to 24%). Further, there is no noticeable trend in the error across the different applications (the applications are ordered in increasing order of t_{mean}). For a given system, the variation in the errors across the applications is high for Base and GPD-2X (126% and 86% standard deviation as a fraction of the mean respectively). For SD-2.5X, the variation is relatively low, but the average error is relatively high (14%).

t_{10} shows similar data, although there are fewer cases of absolute errors $> 15\%$ and the average error is modest. Nevertheless, the range in errors is still quite large (-18% to 21%), the errors are again both positive and negative, and there is no clear significant trend among applications and systems.

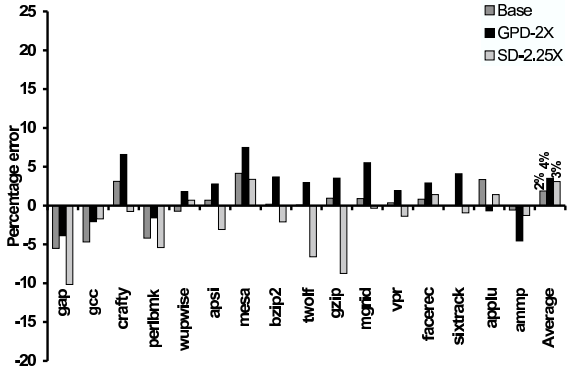
Overall, we conclude that there are several cases for which an approximation of t_n that assumes proportionality to t_{mean} could result in significant positive and negative errors, potentially resulting in under-designed or over-designed systems.

[6]SOFR assumes individual system components have an exponential lifetime distribution and the system is a series-failure system. These assumptions imply that the system itself has an exponential lifetime distribution. For exponential distributions, it is known that $t_n = -t_{mean} \times ln(1 - \frac{n}{100})$ [14].

(a) Error in t_1 estimation



(b) Error in t_5 estimation



(c) Error in t_{10} estimation

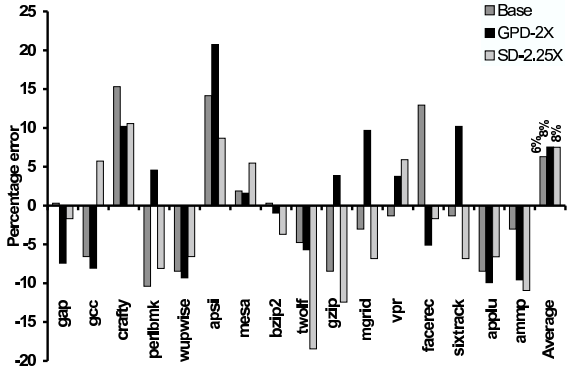


Figure 3: Percentage error in estimation of t_n. The figure shows the percentage error in estimating (a) t_1, (b) t_5, and (c) t_{10} for the Base, GPD-2X, and SD-2.25X systems. The applications are ordered in increasing order of t_{mean}. The t_n estimate assumes that for a given n, $t_n/\ t_{mean}$ is a constant equal to the average t_n/t_{mean} for the base system for that n. The error is significant for several cases and both positive and negative, indicating that in general, a constant factor approximation of t_n to t_{mean} could potentially lead to under- or over-designed systems.

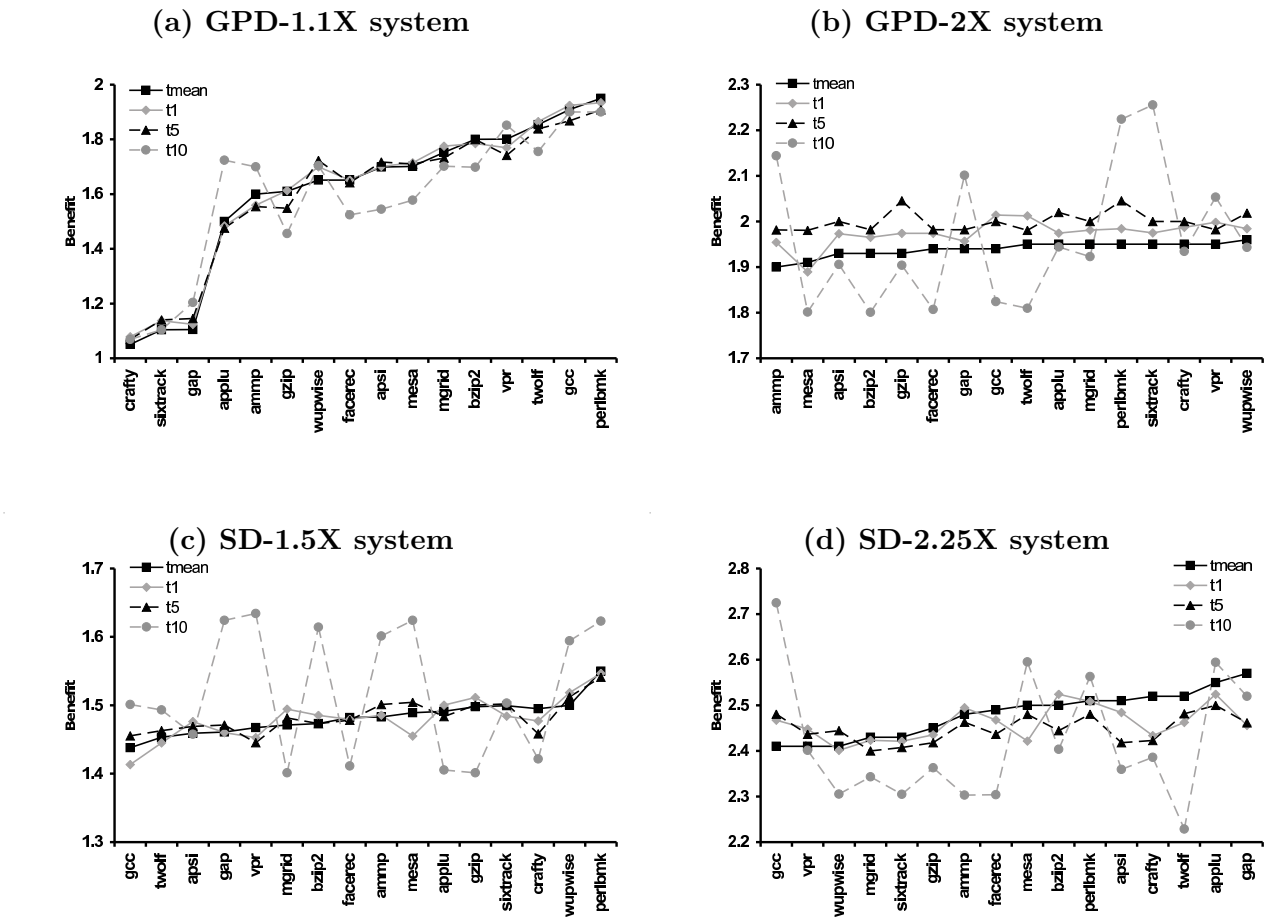


Figure 4: Benefit in t_{mean}, t_1, t_5, and t_{10} for different systems. The figure shows the benefits in t_{mean} and t_n for (a) GPD-1.1X, (b) GPD-2X, (c) SD-1.5X, and (d) SD 2.25X. The applications are ordered according to increasing benefit in t_{mean} for each system. Note that the axes are not identical. In many cases, a benefit in t_{mean} does not translate to a corresponding benefit in t_n. Further, no clear relationship between the benefits in t_n and t_{mean} appears to hold across all cases.

5.2 Improvement in t_{mean} versus t_n

The previous section showed that the absolute value of t_n cannot be reliably predicted from that of t_{mean}. However, designers and users are often concerned with relative comparisons among systems; therefore, we next ask the question whether the *improvement* in t_n from a reliability-enhancing technique can be predicted from the improvement in t_{mean}. We use the techniques of SD and GPD for this purpose.

Figure 4 shows the benefit in t_{mean} and t_n for different reliability-enhanced systems, for different applications. (The benefit in t_n (or t_{mean}) is computed as the ratio of the t_n (or t_{mean}) of the reliability enhanced system over that of the base system.) For each system, the applications are ordered in increasing order of improvement of t_{mean} over the base system.

The figure shows that a benefit in t_{mean} does not, in general, imply the same benefit in t_n and could be higher or lower than the benefit in t_n. For example, for t_{10}, GPD-2X shows a higher benefit for sixtrack (2.2 for t_{10} vs. 1.9 for t_{mean}) whereas SD-2.25X shows a lower benefit for twolf (2.2 for t_{10} vs. 2.5 for t_{mean}).

The figure also does not indicate a clear relationship between the benefit in t_{mean} and t_n that would hold across all systems, applications, and n. For example, with GPD-2X, the t_{mean} benefit for sixtrack and facerec is similar (roughly 1.9), but the t_{10} benefit is a higher 2.3 and a lower 1.8 respectively.

Thus, in addition to the inability to predict the values of t_n from t_{mean} (Section 5.1), one cannot reliably predict the benefits in t_n from the benefits in t_{mean} either. From this, we can infer that designing systems with t_{mean} as a proxy for t_n can potentially lead to poor design choices. Such systems could be over-designed with unnecessarily high cost or low performance (when t_n benefits are under-estimated), or the systems could be under-designed and not meet the required reliability target (when t_n benefits are over-estimated). The next section provides experimental data to show such scenarios.

5.3 Implications for Reliability-Aware Design

This section illustrates the impact of using t_{mean} as a proxy for t_n when considering alternative reliability-enhanced designs. Again, we use SD and GPD as our example reliability enhancements.

Figures 5 shows the benefit under different metrics (t_{mean}, t_1, t_5, t_{10}) for SD and GPD with various overheads for four SpecFP (left column) and four SpecInt (right column) applications. The solid and dotted lines respectively represent SD and GPD systems.

The figure shows several cases where the t_{mean}-optimal design choice is either too aggressive or too conservative from the point of view of t_n. For example, consider a designer targeting a 1.4X improvement in t_5 (or t_{10}) on a system running a workload similar to crafty. If the designer projects the 1.4X improvement in t_n as a 1.4X improvement in t_{mean}, the choices available are a cost increase of 1.5X or more for the SD configurations or a performance slowdown of 1.4X or more for the GPD configurations. These are fairly high overheads from both a cost and performance point of view. However, if the designer designed directly to t_5 (or t_{10}), then we see that a GPD configuration with a 1.25X slowdown also meets the target. Thus, a t_{mean} driven

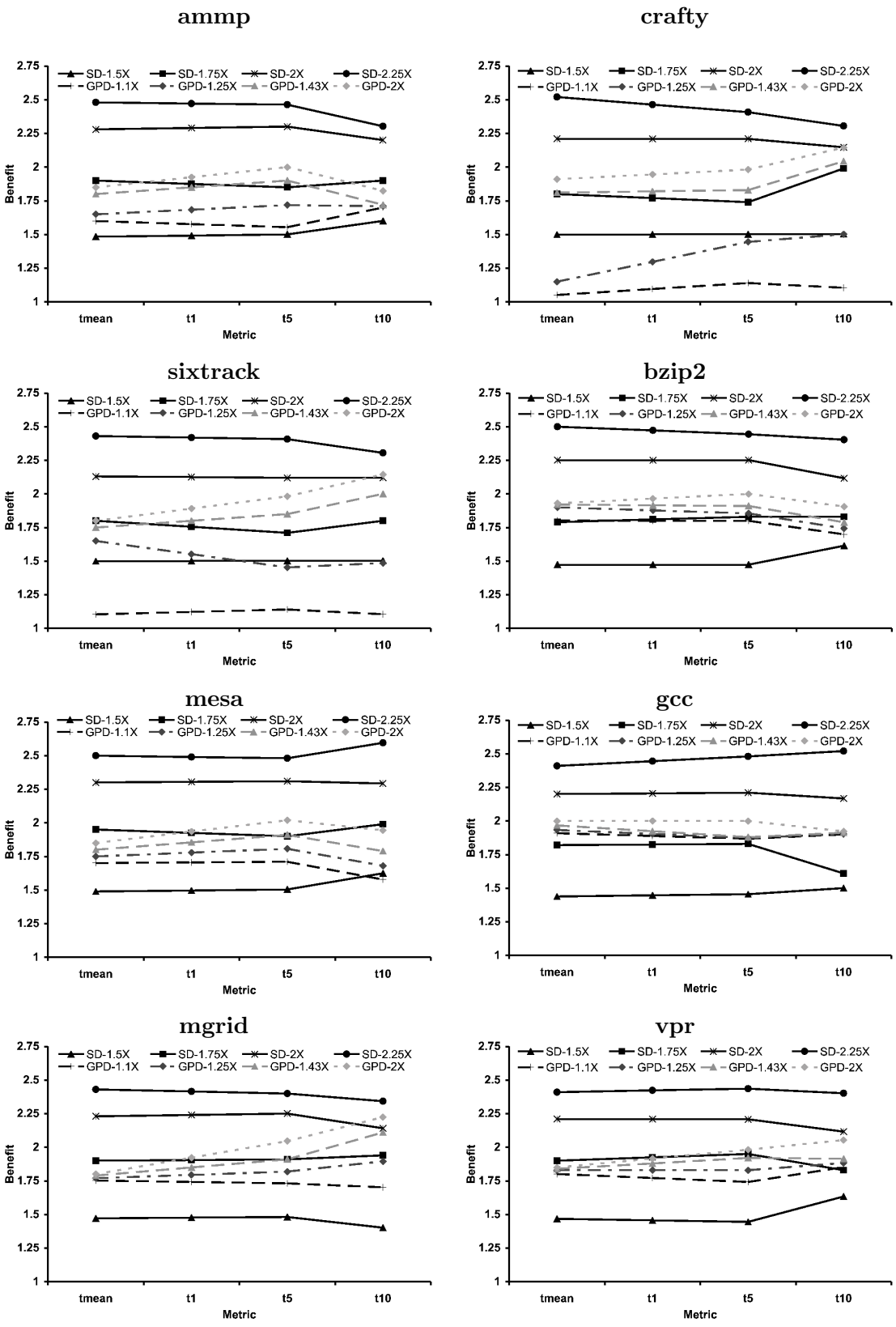


Figure 5: Impact of using t_{mean} instead of t_n on application specific reliability-aware design. Each figure shows the benefit under different metrics (t_{mean}, t_5, t_{10}) for reliability-aware systems with different overheads. The solid and dotted lines respectively represent SD and GPD systems. A system achieving the required reliability target under one metric may fail to do so under other metrics, leading to poor designs.

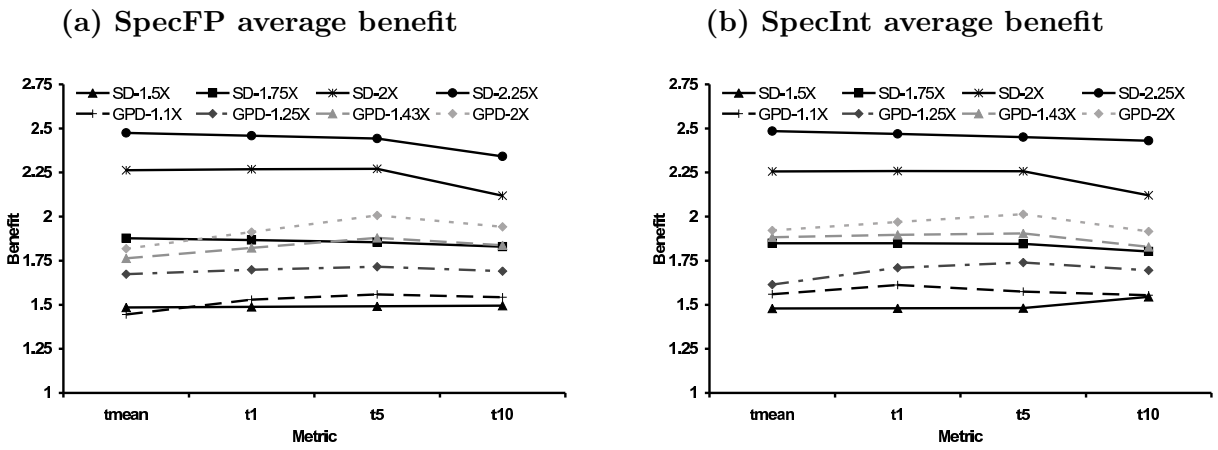


Figure 6: Impact of using t_{mean} instead of t_n on reliability-aware design for average (a) SpecFP and (b) SpecInt applications.

design would be unnecessarily expensive or slow.

Conversely, for sixtrack, a cost-bound designer targeting an improvement of 1.6X in some t_n would choose the GPD-1.25X configuration when using t_{mean} as the proxy metric. However, this does not meet the target for any of the t_n's and the system is under-designed for reliability. Instead, a designer employing t_n data would have chosen GPD-1.43X and achieved the target reliability (albeit at reduced performance).

Figure 6 shows the same data as Figure 5, but for the benefit averaged across all SpecFP (part (a)) and SpecInt (part (b)) applications. We see that the above effects are less pronounced in the average graphs, but nevertheless do exist.

Thus, using t_{mean} as a proxy for t_n to perform reliability-aware system design can lead to incorrect design decisions. The system may achieve the desired reliability under the proxy t_{mean} but may fail to do so for the real metric t_n. Further, systems that suffer an unnecessary dip in performance or increase in die area may be chosen to achieve the desired target while a system with a significantly lower overhead would have sufficed. Thus, designing with t_{mean} as a proxy for t_n can have a large undesirable effect.

5.4 Lognormal vs. Exponential Distributions

Our results so far show that using t_{mean} instead of t_n can have a significant implication for design. The reason for the discrepancy is that we use lognormal lifetime distributions for the underlying components of the system. Combining component-wise metrics to get a system-wide metric in this case is analytically hard and results in a complex relationship between t_n

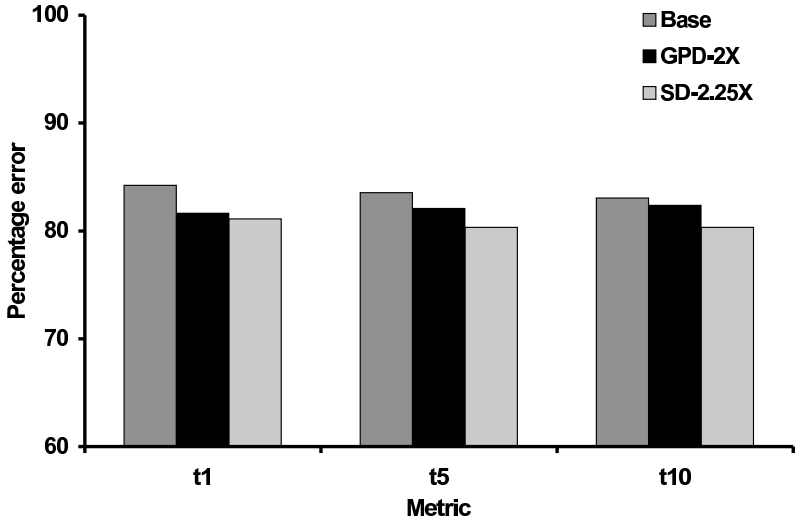


Figure 7: Percentage error in t_n when using exponential distribution. The figure shows the average error in t_n, for different n and different systems, when the underlying failure distribution is assumed to be exponential. The significant error margin in the estimation suggests that the exponential approximation, though simpler, is not valid even when using t_n as a design metric.

and t_{mean} that does not have a closed form [1]. However, as discussed earlier, this is not the case for exponential component-wide distributions, which result in exponential system-wide distributions and a closed form between t_{mean} and t_n [1]. Although exponential distributions are known to not adequately represent wear-out behavior, much work in industry makes the assumption of exponential distributions due to its simplicity [14].

We therefore next analyze the complexity vs. accuracy trade-off between the exponential and lognormal distributions. We estimate the exponential t_n using (1) the SOFR-derived t_{mean} from the simpler RAMP 1.0 and (2) the closed form relationship between t_{mean} and t_n for exponential distributions (Section 4). We then report the error in this t_n relative to the t_n derived from the lognormal assumption with RAMP 2.0. Figure 7 shows this percentage error averaged across all applications for each of t_1, t_5, and t_{10} for different systems. (In order to perform a fair comparison, the t_{mean} values were normalized such that the average t_{mean} across all applications is 30 years for both RAMP 1.0 and for RAMP 2.0.)

The average error in t_n when assuming an exponential distribution is significant in all systems, for all n. Thus, although the exponential approximation largely simplifies the computation of t_n, the simplicity comes at a large cost in accuracy.

This is consistent with our previous data. The exponential methodology computes t_n as a constant function of t_{mean}. As seen in section 5.1, this approximation is not valid. In addition,

the constant used for a particular n is $-ln(1 - \frac{n}{100})$ which is significantly smaller than the ratio of t_n to t_{mean}, for small n, as seen from Figure 2. Hence, the error in the estimation is significant.

Thus, our results show that t_n for different systems has to be computed using first principles – a constant factor approximation applied to t_{mean} and a simpler failure distribution assumption lead to erroneous results.

6 Conclusion

Aggressive CMOS scaling has lead to the onset of wear-out based failures at a rate not seen in the past. While high-end systems in niche high-reliability markets have always been concerned about lifetime reliability, recently this concern has spread to the broader general-purpose market as well. Concerned microarchitects have begun to propose reliability-aware microarchitectures to tackle problems caused by wearout failures. However, these designs are largely analyzed based on the benefits achieved in MTTF (t_{mean}). MTTF, being an average, does not capture sufficient information about the useful operational life of the processor, which is typically much smaller than the mean life. Thus, MTTF can be misleading to both customers and designers as systems with similar MTTFs can have different failure distributions and reliability characteristics for the most useful part of the product's life. The reliability literature provides an alternate metric that does not have this limitation of MTTF. In this paper, we have studied the metric of time to failure of n% of the population, denoted t_n.

This paper, for the first time to our knowledge, analyzes the implication of designing reliability-aware microarchitectures for MTTF as a proxy for potentially more accurate metrics like t_n. Distribution dependent metrics such as t_n are inherently significantly more complex to compute, but this paper presents a straightforward way of computing such a metric through the use of the state-of-the-art architecture level lifetime reliability tool RAMP 2.0.

Our analysis indicates that the use of MTTF as a proxy for t_n can lead to poor reliability-aware microarchitecture designs. Systems designed based on MTTF may fail to meet the intended reliability target owing to an incorrect projection of the benefit in t_n as a benefit in MTTF. In other cases, the systems may incur an unnecessary performance or cost overhead as a low benefit in MTTF may correspond to a higher benefit in t_n.

Admittedly, current architecture level tools for lifetime reliability are still preliminary and

make significant assumptions. Nevertheless, this paper takes an important step in understanding the right metrics that should be targeted by such tools, and the impact of instead using the widely used MTTF metric for reliability-aware processor design.

Additionally, t_n may not be the only metric that designers are concerned about. Other metrics such as mean time to repair, etc. are also important. Although this paper does not analyze all such metrics, it establishes the first important step to move beyond the widely used but potentially misleading MTTF for architects.

References

[1] Assessing Product Reliability, Chapter 8, NIST/SEMATECH e-Handbook of Statistical Methods. In *http://www.itl.nist.gov/div898/handbook/*.

[2] Limitations of the exponential distribution for reliability analysis. In *Reliasoft Newsletter, http://www.reliasoft.com/newsletter/4q2001/exponential.htm*, 2001.

[3] Methods for Calculating Failure Rates in Units of FITs. In *JEDEC Publication JESD85*, 2001.

[4] Critical Reliability Challenges for The Intl. Technology Roadmap for Semiconductors. In *Intl. Sematech Tech. Transfer 03024377A-TR*, 2003.

[5] Failure Mechanisms and Models for Semiconductor Devices. Technical report, March 2006.

[6] S. Borkar. Microarchitecture and design challenges for gigascale integration. In *Proceedings of the 37th Annual International. Symp. on Microarchitecture (Key note)*, 2004.

[7] F. A. Bower, P. G. Shealy, S. Ozev, and D. J. Sorin. Tolerating hard faults in microprocessor array structures. In *Proceedings of Intl. Conference on Dependable Systems and Networks*, 2004.

[8] F. A. Bower, D. J. Sorin, and S. Ozev. A mechanism for online diagnosis of hard faults in microprocessors. In *Proceedings of the 38th Annual Intl. Symp. on Microarchitecture*, 2005.

[9] D. Brooks, P. Bose, V. Srinivasan, M. K. Gschwind, P. G. Emma, and M. G. Rosenfield. New methodology for early-stage, microarchitecture-level power-performance analysis of microprocessors. *IBM J. Res. Dev.*, 47(5-6):653–670, 2003.

[10] D. M. Brooks, P. Bose, S. E. Schuster, H. Jacobson, P. N. Kudva, A. Buyuktosunoglu, J.-D. Wellman, V. Zyuban, M. Gupta, and P. W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6), 2000.

[11] A. Dasgupta et al. Electromigration Reliability Enhancement Via Bus Activity Distribution. In *Design Automation Conference*, 1996.

[12] J. L. Hennessy and D. A. Patterson. *Computer Architecture, A Quantitative Approach*. Morgan Kaufmann, 2003.

[13] V. S. Iyengar, L. H. Trevillyan, and P. Bose. Representative traces for processor models with infinite cache. In *Proceedings of the 2nd IEEE Symposium on High-Performance Computer Architecture*, 1996.

[14] W. Q. Meekar and L. A. Escobar. *Statistical Methods for Reliability Data*. 1998.

[15] C. Moore. The POWER4 System Microarchitecture. In *Microprocessor Forum*, 2000.

[16] Reliability in CMOS IC Design: Physical Failure Mechanisms and their Modeling. MOSIS Technical Notes, http://www.mosis.org/Faqs/tech_cmos_rel.pdf.

[17] M. Moudgill, P. Bose, and J. Moreno. Validation of Turandot, a Fast Processor Model for Microarchitecture Evaluation. In *Proceedings of Intl. Performance, Computing and Communication*, 1999.

[18] Sidharth and S. Sundaram. A methodology to assess microprocessor fan reliability. In *The 9th Intersociery conference on Thermal and Thermomechanical Phenomena in Electronic Systems*, 2004.

[19] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *Proceedings of the 30th Annual Intl. Symp. on Comp. Architecture*, 2003.

[20] J. Srinivasan, S. V.Adve, P. Bose, and J. A. Rivers. The Case for Microarchitectural Awareness of Lifetime Reliability. In *Proceedings of the 31st Annual Intl. Symp. on Comp. Architecture*, 2004.

[21] J. Srinivasan, S. V.Adve, P. Bose, and J. A. Rivers. The Impact of Scaling on Processor Lifetime Reliability. In *Proceedings of the Intl. Conf. on Dependable Systems and Networks*, 2004.

[22] J. Srinivasan, S. V.Adve, P. Bose, and J. A. Rivers. Exploiting Structural Duplication for Lifetime Reliability Enhancement. In *Proceedings of the 32nd Annual Intl. Symp. on Comp. Architecture*, 2005.

[23] K. Trivedi. *Probability and Statistics with Reliability, Queueing, and Computer Science Applications*. Prentice Hall, 1982.

[24] A. Wood. Reliability-metric varieties and their relationships. In *Proceedings of Annual Reliability and Maintainability Symposium*, 2001.

[25] D. Yen. Chip multithreading processors enable reliable high throughput computing. In *Proceedings of the International Reliability Physics Symposium (Key note)*, 2005.