Resilience and Real-Time Constrained Energy Optimization in Embedded Processor Systems

Liang Wang[*] [†], Jude A. Rivers[*], Meeta S. Gupta[*], Augusto J. Vega[*], Alper Buyuktosunoglu[*], Pradip Bose[*], and Kevin Skadron[†]
[*]IBM T. J. Watson Research Center and [†]University of Virginia

***Abstract*—Low-power embedded processing relies on dynamic voltage-frequency scaling (DVFS) in order to optimize energy usage and therefore battery life. DVFS allows the processor to continuously adapt voltage and frequency to the minimum that still meets a program's current performance requirements. However, low-voltage operation exacerbates the incidence of soft errors. Similarly, high-voltage operation (to meet real-time deadlines) is constrained by power dissipation (and associated thermal) maxima - as dictated by aging limits. In this paper, we introduce a novel modeling framework called PEARL to examine a class of embedded system applications relevant to mobile, airborne vehicles. Using PEARL, we investigate the problem of assigning optimal voltage-frequency settings to individual segments within example workflows. The goal of this study is to understand the limits of achievable energy efficiency (performance per watt) under varying levels of system-resilience targets. The analysis results show that: (a) the resilience constraints limit achievable efficiency; and (b) higher variability in power dissipation across workflow segments provides more opportunities to boost efficiency, despite stringent resilience constraints.**

***Keywords*-low power, embedded systems, soft error resilience, real-time constraints**

I. Introduction

A popular optimization used in power management of embedded processor systems is dynamic voltage-frequency scaling (DVFS). However, soft error rates (SER) are known to increase sharply as the supply voltage is scaled downward. Hence, in order to preserve system resilience levels, it is important to apply voltage scaling rather carefully, keeping in mind the varying levels of vulnerability to SER within an application's execution profile. On the other hand, overclocking or turbo-boosting (with higher voltages applied if/as necessary) to meet real-time deadlines comes at the cost of higher power density and temperature, which results in higher fail rates due to hardware aging.

In this paper, we consider a class of embedded systems[1] that require high levels of power or energy efficiency while meeting mission-critical reliability specifications and real-time performance targets. Such systems require an energy optimization protocol that is cognizant of the variable resiliency needs and properties of the executed application. This paper consists of the following three contributions:

- We first describe PEARL, a novel software modeling framework that enables users to: (a) *statically* prepare application workflows for energy-optimized resilience; and (b) explore *run-time* deployment options in targeted embedded systems. The overall goal is to maximize performance per watt, while meeting real-time deadlines and resilience-related constraints (dictated by effective SER and maximum power dissipation). PEARL stands for Power Efficient And Resilient embedded processing with reaL-time constraints.
- We then describe an efficient technique (AFI) to evaluate the application-level derating (AD) [1] of SER. The application-level fault injection (AFI) facility is a resilience characterization tool within PEARL. We present results to show the wide range of SER vulnerability (2% to 55%) across the set of PERFECT suite applications that have been analyzed so far in this work.
- Finally, we provide illustrative experimental analysis of potential optimizations using PEARL. Our key conclusions are: (a) for workflows that exhibit little variation in power consumption across component application segments, the resilience-related constraints leave little room (5% at best) in improving energy efficiency over a simple baseline; (b) if the maximum-power constraints are relaxed, there is a significant opportunity (17%) in optimizing the efficiency, despite stringent SER related constraints; (c) if the inter-application power variation across the workflow is large, then the potential improvement opportunity increases dramatically (40%)

[1]In particular, we refer to systems targeted by the ongoing DARPA MTO sponsored PERFECT (Power Efficiency Revolution for Embedded Computing Technologies) program.

II. PEARL: A User Perspective

We consider a scenario (see Figure 1) in which one or more unmanned airborne vehicles (UAVs) are engaged in remote sensing of ground images, for the purposes of recognizing specific targets of interest (and then actuating tactical response mechanisms, if applicable) in national defense and homeland security applications. The UAVs carry embedded processor systems and are wirelessly interconnected with each other, while being supported by a secure ground-based (cloud) server.

As indicated in Figure 1, the dynamic computing environment can be rather harsh, with many run-time parameters that are hard to anticipate during ground-based application development. Although the basic compute kernels are known, the dynamic uncertainties can be encountered and dealt with only at run-time. The boxed inset at the bottom right-hand side of Figure 1 depicts the layered hardware-software embedded system that is actually modeled in

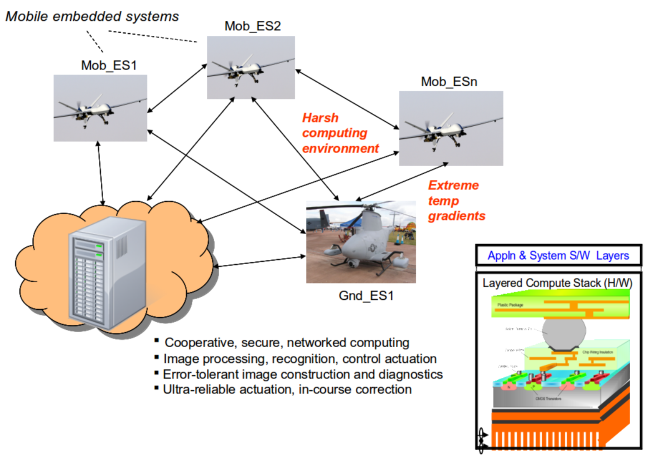
Mobile embedded systems
Mob_ES1
Mob_ES2
Mob_ESn
Harsh computing environment
Extreme temp gradients
Gnd_ES1
Appln & System S/W Layers
Layered Compute Stack (H/W)
▪ Cooperative, secure, networked computing
▪ Image processing, recognition, control actuation
▪ Error-tolerant image construction and diagnostics
▪ Ultra-reliable actuation, in-course correction

Figure 1. Cloud-backed airborne embedded system network, which operates under harsh conditions

PEARL.

Figure 2 depicts the high-level view of the PEARL modeling framework. The R-API smart graphical user interface (GUI) is the portal through which the user interacts with the rest of the software infrastructure. There are two modes in which R-API is invoked by the user/developer of the embedded system application. The first mode is that of *application preparation*: a static, offline step in which the applications in the workflow are characterized in terms of power, performance and SER resilience, and the workflow is annotated accordingly. In the second mode, the user is able to simulate the *dynamic deployment* of the workflow by factoring in run-time uncertainties associated with the environment depicted in Figure 1.

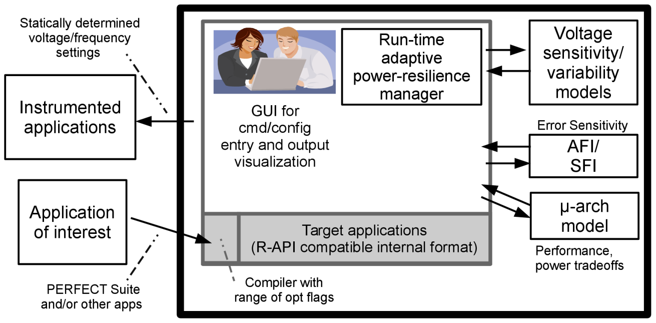
Statically determined voltage/frequency settings
Instrumented applications
Application of interest
PERFECT Suite and/or other apps
GUI for cmd/config entry and output visualization
Run-time adaptive power-resilience manager
Voltage sensitivity/ variability models
Error Sensitivity
AFI/ SFI
µ-arch model
Performance, power tradeoffs
Target applications (R-API compatible internal format)
Compiler with range of opt flags

Figure 2. PEARL Framework with Resilience-Aware Application Programing Interface (R-API): High-Level Overview

A. *Voltage Sensitivity Models*

In Figure 2, a key input to the R-API GUI is the box marked “voltage sensitivity” model. This captures empirically derived sensitivity of per-device SER to voltage changes. In PEARL, we use IBM-published voltage sensitivity gradients [2] to model the change in raw SER (unmasked) caused by the scaling of operating supply voltage in a DVFS setting:

$$\mathrm{SER}_{\mathrm{raw}} = e^{\alpha\cdot\mathrm{Vdd}+\beta}$$

where α,β are fitting parameters derived from the empirical data shown by Cannon [2].

B. *Fault Injection Facility*

As indicated in Figure 2, the user of PEARL can invoke fault injection facilities (labeled as AFI/SFI) to profile a given application component in terms of its inherent resilience to application state bit corruptions. The AFI facility internals are described in more detail in section IV.

C. *Static Optimizer Module*

After power-performance and resilience characterization of component application segments, the static optimizer module can be invoked by the user to set optimal DVFS settings to each segment within a given workflow of interest. Details of how this optimization algorithm works are described in section V.

D. *Cross-Layer Dynamic Optimizer*

The R-API smart GUI is architected to invoke dynamic resource and resilience managers to provide an emulation facility for run-time deployment of statically-prepared application workflows. These modules are still under development, but an example dynamic optimization analysis is provided in section VI.

III. PEARL-Based Analysis Methodology

A. *Applications*

Six representative applications were selected from the benchmark suite announced for targeted research in the DARPA PERFECT project: 2dconv, dwt53, hist, oprod, syssol, and iprod. These particular applications were selected with care as representative components from which to synthesize composite workflows in our experiments (see section V). dwt53 represents a well-known image compression algorithm; and the histogram-equalization (hist) code is an integral part of many sort routines. 2D-Convolution (2dconv) is another frequently used image-processing kernel. Outer Product (oprod), Inner Product (iprod) and SystemSolver (syssol) are well known matrix-oriented data processing algorithms that are used in both embedded computing and high performance scientific computing domains.

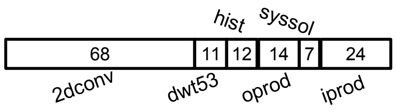

Figure 3. Workflow constructed by stitching together six applications selected from the PERFECT suite. Numbers in the block indicate execution times of each application. The execution times are measured based on single-core, single-threaded measured runs on a POWER7+ machine at a nominal operation point of 4.1 GHz

In the section V and VI, we consider a stitched kernel workflow, composed of six separate kernels within the PERFECT suite as depicted in Figure 3.

We stipulate that the primary power-performance-resilience control knob in our optimization process is dynamic voltage and frequency scaling (DVFS). For this initial experimental analysis, in order to develop basic understanding, we focus only on a single-core, single-thread execution mode.

The experimental system has two IBM POWER7+ processors [3] running at 4.1 GHz (nominal).The performance metrics are accessed via the performance monitoring unit (PMU) facilities within the processor. The power metrics are measured via the AMESTER framework[4], which is a specially developed firmware that is capable of monitoring the power and temperature measurements tracked by the EnergyScale micro-controller [5]. We choose the DVFS settings with frequency from 4.3 GHz through 2.4 GHz with a step size of 0.1 GHz. Based on the machine measurements, we further extrapolate the characterizations to 1 GHz assuming power scales as the k-th power of supply voltage (V): $P \propto V^k$. In this study, we varied the value of k from 2.5 to 4 across all applications. As shown in Figure 9 section V, we obtain the best energy improvement with $k = 2.5$. Therefore we use $k = 2.5$ for the rest of our analysis. This is purely for the purposes of illustrating the optimization capabilities of PEARL. The real value of k is usually larger, depending on the particular system and the executing application. Further, we assume that voltage and frequency are linearly co-related and performance scales linearly with frequency. The resilience of an application is the inverse of the effective failure rate, measured in failure in one billion hours (FITs). For the purposes of this paper, we model the FIT of an application as $\text{SER}_{\text{raw}} * \text{SDC\_rate}$. The SDC_rate is obtained by applying AFI on the application running on the same POWER7+ platform. (See also the next section for details on AFI).

The efficiency metric of optimization in this paper is performance per watt, where performance is measured as giga-operations per second (GOPS).

IV. Application-Level Resilience Characterization

To accurately predict system FIT values for a specific application workload (or workflow) in a pre-silicon setting, the associated microarchitectural and application derating (i.e. masking) factors (MD and AD) [1], [6] must be estimated. Pre-silicon system-level methods for estimating the MD factor are found in prior works describing techniques such as AVF [7] and Phaser [8]. The Application Fault Injection (AFI) tool (with a built-in programmable statistical fault injection routine, SFI) that we describe in this section provides an estimate for the application-level derating (AD). The application-specific sensitivity of MD is known to be small when it comes to net SDC FITs estimation [1], [6]; in this paper, we focus on the AD factor, combined with basic SER sensitivity to voltage described in section II.

AFI makes use of the ptrace debugging facility in Unix to create a framework in which any target application can be compiled and run, under user-controllable fault injection directives. The resulting AFI facility provides a mechanism by which a parent process can observe and control the execution of another process by examining or changing the architected register and memory state of the monitored process. Since AFI runs natively on actual hardware, the speed of fault injection is orders of magnitude faster than conventional full-system, simulator-driven fault injection studies.

A statistically saturating total (in this case, 1000) of single-bit, pseudo-random fault injections are made into the architected register space in each controlled experiment[2]. Each such injection leads to one of the following outcomes

- The injected bit-flip has no effect on the program execution profile, including the final program output or final data memory state. These fault injections are categorized as being fully masked.
- The injected bit-flip results in an SDC (silent data corruption) in the final program output state, when compared to the fault-free, golden run of the program.
- The injected bit-flip results in a program crash, where the operating system terminates the program due to a detected runtime error (e.g. divide-by-zero exception, segmentation fault, illegal memory reference, etc.)
- The injected bit-flip results in a hung state, where there is no forward progress of the program execution; in practice such a situation would require a user-initiated program termination or even a machine reboot.

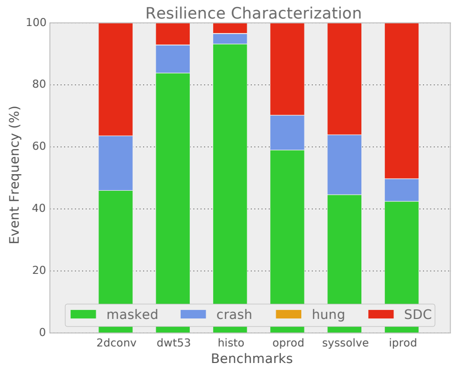

Figure 4. AFI-deduced fault distribution GPR injections across selected PERFECT applications.

Figure 4 shows the distribution of faults across the six applications chosen from the announced PERFECT suite. Here we only show the effect of faults injected into the general purpose architected registers (GPR). The observed SER (especially SDC) sensitivity for floating point register (FPR) injections was much smaller, hence we omit that here.

[2]We studied the masking saturation rate of the PERFECT suite workloads. 1000 injections indicated statistical sufficiency for our experiments.

We see significant difference in SDC vulnerability across the various applications (ranging from 2% to as much as 55%). The SDC sensitivity in the AD component of SER is largest in the inner product kernel (iprod) because of the large number of general-purpose registers used with the unrolling optimization applied by the compiler in this case.

It is important to note that Figure 4 deals only with the AD component of SER masking. The net SER FITs, capturing both MD and AD, would show different vulnerability percentages than shown in Figure 4. Yet, in the PEARL-based analysis reported in this paper (see section V), our primary focus is on SDC; and, when it comes to SDC FITs evaluation, our prior experience [1], [6] dictates that we need to focus mainly on the AD masking component.

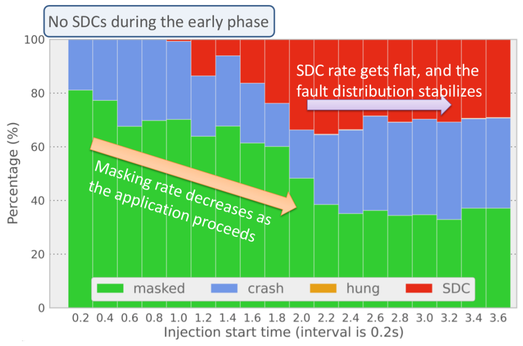

Figure 5. SER resilience (AD factor) as a function of execution time (i.e. interval of 0.2s) for one of the FFT routines analyzed.

The dynamic resilience profile for a particular FFT routine (Figure 5) shows that the SDC related vulnerability is near-zero in the initial phases, where data is being read or initialized from protected memory or storage space; but it increases and stabilizes to around 30% as the kernel enters the actual iterative data computational phase. This is because the fault injection is limited to GPRs, which are hardly used during the initialization phase.

V. Static Compile-time Optimization

We now formulate the following static real-time-constrained DVFS-mode allocation problem in our PEARL framework. The basic goal is to maximize delivered performance per watt, subject to minimum resilience and maximum power constraints.

Given:

- A workflow composed of N applications, $W = \{A_1, A_2, \dots, A_N\}$;
- A list of M possible processor DVFS settings (or modes), $F = \{f_1, f_2, \dots, f_M\}$, each associated with M corresponding DVFS settings;
- The power, execution time and resilience characteristics of each application under all available DVFS settings, $P[i] = \{p_{i1}, p_{i2}, \dots, p_{iM}\}$, $T[i] = \{t_{i1}, t_{i2}, \dots, t_{iM}\}$, $R[i] = \{r_{i1}, r_{i2}, \dots, r_{iM}\}$, $i = 1, 2, \dots, N$;
- Constraints on maximum power (C_p), maximum execution time (C_t), and minimum resilience (C_r). C_r is defined as the maximum failure rate (also referred to here as *inverse resilience*, measured in units of FITs). In this paper, we use SDC-vulnerability at the application level as a proxy for failure rate, and the time-weighted FIT rates of all applications as the resilience of a workflow:
$$\frac{\sum FIT_i * T_i}{\sum T_i} \leq C_r$$
where FIT_i and T_i are the FITs and run time of the ith application.

Minimize: Total energy of execution by picking the "best" combination of DVFS settings indicated by frequencies $(f_1, f_2, \dots, f_N)$ across the N applications while adhering to workflow execution time, power, and resilience constraints.

We first demonstrate a simple optimization algorithm (SimpleOpt) as the baseline, then we map the above problem to an integer linear programming problem and solve it with a standard algorithm (LinOpt).

A. SimpleOpt

The low-power literature has established that, in order to achieve the best GOPs/Watt for any given application, it is generally necessary to operates at the lowest supported DVFS level. So, for the segmented workflow, *SimpleOpt* tries to choose the lowest feasible single DVFS level that meets all constraints for the workflow. These constraints are that the real-time execution deadline must be met and the power and SER resilience targets must also be obeyed. The algorithm proceeds as follows:

1) Start by picking the lowest DVFS level that meets the run-time constraint of the workflow.
2) If the current DVFS setting does not meet the resilience constraint, raise the DVFS level from the one picked in the last step, until the resilience constraint is met.
3) Check whether the selected DVFS level from the last step violates the power constraint. If yes, then there is no feasible solution from the SimpleOpt, otherwise the selected DVFS level is the solution.

This algorithm is sub-optimal, because per-application DVFS operating points are not explored in achieving a globally optimal solution.

B. LinOpt

We now describe an improved heuristic called LinOpt that is an option within PEARL. LinOpt is formulated as a static, real-time-constrained DVFS mode allocation problem, and solved by a linear programming algorithm. To formulate it as an integer linear programming problem, for each application A_i, we define a frequency mask bit-vector (FM) of length M. Only one bit j within the FM_i bit-vector is set to 1, indicating a particular DVFS mode setting. This is illustrated in Figure 6.

A_i's power, performance and resilience can be expressed as the dot product of the characteristics and the associated frequency mask FM_i:

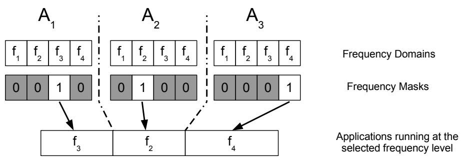

Figure 6. Using frequency masks to denote the selected frequency of each application.

- Power = $P[i] \cdot FM_i$
- (Inverse) Resilience = $R[i] \cdot FM_i$
- Execution time = $T[i] \cdot FM_i$

Constraints can be expressed as:

- Power: $P[i] \cdot FM_i \leq C_p$ for each application A_i
- (Inverse) Resilience: $(\sum R[i] \cdot FM_i)/C_t \leq C_r$ for each application A_i
- Execution time: $\sum T[i] \cdot FM_i \leq C_t$

The goal of the original problem is equivalent to minimizing the total energy of execution by finding the "best" combination of FM bit-vectors $(FM_1, FM_2, \dots, FM_N)$ across the N applications, while adhering to workflow execution time, power, and resilience constraints. It is an integer linear programming problem. We use a standard branch-and-cut algorithm [9] to solve the problem.

C. *LinOpt vs. SimpleOpt*

Figure 7 shows the improvement offered by *LinOpt* over the baseline *SimpleOpt* heuristic, as a function of the operating frequency across the various constraints. Various constraint scenarios (SC, URC, UPC and UPRC), as explained in the figure legend, are shown.

In the case of the strict constraint (SC), the best benefit we observe is only about 5%. The reason is that the power variations across the six application segments within the workflow were minimal. Therefore, *LinOpt* was not able to exploit the variable DVFS settings across the workflow. In order to demonstrate the benefit of *LinOpt*, therefore, one needs to relax the power constraint (e.g. UPC or UPRC).

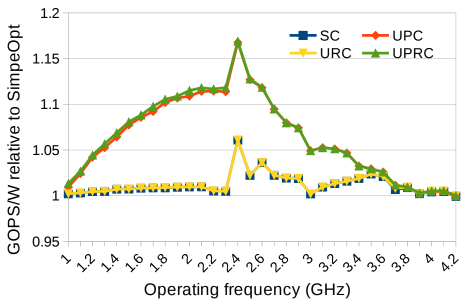

Figure 7. LinOpt improvement in performance per watt (GOPS/W) over SimpleOpt, across a range of modeled DVFS levels. The legend SC refers to a strict constraint, where stipulated power, resilience and execution time constraints must all be met. This case is almost completely overlapped with the URC case, in which the SER resilience constraint is removed; UPC removes just the power constraint; and UPRC removes both the power and SER resilience constraints.

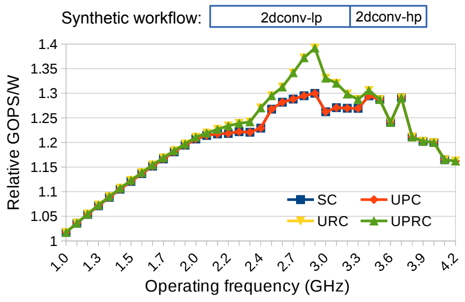

Figure 8. *LinOpt* benefit over *SimpleOpt* for a synthetic workflow.

We see that *LinOpt* achieves the maximum benefit (17% better GOPS/W than *SimpleOpt*) at around the mid-point (2.5 GHz) of the covered DVFS settings. This can be explained by noting the fact that at this mid-point, the *LinOpt* optimizer has the maximum opportunity of adjusting DVFS settings (up or down) across the workflow segments relative to the nominal point.

We examine the SC case in more detail by experimenting with workflows in which the power variation is a lot greater than the case we have studied above. For example, if we consider a two-segment workflow, consisting of 2dconv-hp (a synthetically augmented 2dconv in which the power is 2x of the nominal) and 2dconv-lp (a synthetically augmented 2dconv in which the power is 0.1x of the nominal), the *LinOpt* benefit over *SimpleOpt* is as large as 40%, as shown in Figure 8.

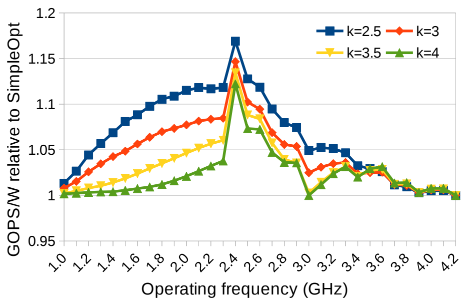

Figure 9. Sensitivity study on k in the power scaling mechanism of $P \propto V^k$. We only show the unlimited power and resilience constraints (UPRC) here.

VI. Dynamic, Run-time Efficiency Optimization

In this section, we present a view of run-time DVFS control options to provide a more robust system that can override statically-set options (if/as needed). We show an illustrative analysis of dynamic (run-time) optimization performed using PEARL for the particular 6-segment application workflow that was considered before for the static optimization experiments. The dynamic experiment starts with statically allocated DVFS settings, derived using *LinOpt*. These settings are used to start up the dynamic emulation run of the workflow. Once started, each experiment works as follows where we start with $i = 1$:

1) A random slack (positive and negative) is added to the nominal execution time of the A_i within the workflow (see Figure 3).
2) At the end of A_i, the elapsed execution time is compared with the statically measured nominal execution time. Then, the remaining segments constituting the residual workflow are re-optimized using *LinOpt* to deduce new DVFS settings.
3) We go back to step 1) and iterate until the workflow is fully processed.
4) If it is feasible to finish the workflow by the stipulated deadline, then the experiment is marked as being a feasible run; else, it is marked as an infeasible trial run.

Such an experiment is repeated 1,000 times to compute the average GOPS/W achieved by the feasible trials. The whole exercise is then repeated by making the *LinOpt* optimizer's deadline constraints stricter by 5%, 10%, 15% and 20%. Figure 10 summarizes the results of the above experiments.

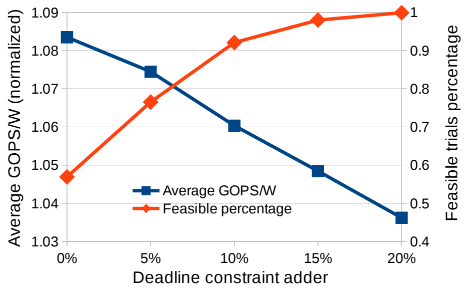

Figure 10. The left Y-axis plots GOPS/W boost from dynamic optimization in cases where real-time deadline could be met, normalized to static *LinOpt* measured at 2.5 GHz. While the right Y-axis shows the percentage of feasible runs (i.e. those that meet the stipulated real-time deadline) in the dynamic experiments, as a function of the percentage slack added to the original deadline.

The left axis of Figure 10 shows the measured GOPS/W boost over the best case of *LinOpt* (i.e. at 2.5 GHz as shown in Figure 7). The benefit is plotted as a function of progressively stricter deadline constraint presented to the optimizer. Only the feasible trials within the dynamic experiments are used to calculate the GOPS/W boost. The benefit decreases with stricter deadline constraint, because a stricter deadline generally imposes higher DVFS points. As such, the efficiency decreases.

The right axis of Figure 10 reports the percentage of feasible dynamic experimental runs. As we make the deadline constraint stricter, the infeasible trials dwindle to zero. This is because setting a conservative deadline target leaves more slack even in the presence of large timing variations.

The possible range of practical run-time heuristics to dynamically adjust DVFS settings is obviously quite large, and we address this whole topic area in a separate paper in submission to a subsequent conference. Here, we only provide an illustrative example of the run-time optimization capability in PEARL.

VII. Related Work

Dynamic voltage and frequency scaling (DVFS) has been widely used in the system architecture area for managing workload-driven power in a processor or system context. For example, Isci et al. [10] propose multi-core DVFS algorithms with the objective of maximizing chip throughput performance for a given power budget. In the real-time embedded systems domain, early work by Pillai et al. [11] and more recent work by Devadas et al. [12] and Qi et al. [13] are but a few representatives from a large body of work. These works all utilize DVS (or DVFS) in order to meet low power constraints, while meeting execution time deadlines.

Combining reliability considerations with energy savings in a unified dynamic resource management framework is a topic area that has not been explored as widely as dynamic power management alone. Zhang et al. [14] address a reliability-aware power management problem, but in this case the reliability focus is on checkpoint-restart based systems only. Zhao et al. [15] address the mutually opposing issues of energy efficiency and transient error probability in the context of applying DVS or DVFS control. However, Zhao et al. use a set voltage dependence equation to model system resilience, without factoring in the application-level masking effects (as in our work). Perhaps the work that is closest to ours is the one by Shafik et al. [16]. As in our static optimization work, the authors of [16] propose a similar linear optimization based approach, while emphasizing the impact of application level correctness to the system resilience. However, in their work, the method of characterizing the application level resilience is SoC-specific, and does not apply to general-purpose processors, as studied in our work. Furthermore, in our work, the emphasis is on developing a pre-deployment application preparation and dynamic emulation framework (PEARL), driven by an interactive user interface (R-API). This framework is targeted for use in the specific context of airborne embedded systems, in which targeted applications of interest in the DARPA PERFECT program are used.

PEARL emphasizes an application-centric treatment of the energy-efficient resilience problem in a real-time embedded systems context. As such, application-level masking effects in response to single-event upsets at the physical level are considered in determining the regions within an application workflow that are most vulnerable to soft errors. We believe this approach enables the use of more realistic constraints in the resilience dimension of this real-time optimization problem.

VIII. Conclusions

Future ultra-efficient embedded systems with mission-critical resilience requirements in the deep-submicron design era will require careful balance in static preparation and run-time deployment of applications. In this paper, we first provide a vision of an interactive facility (PEARL) for application preparation and runtime steering in the context of a class of embedded systems that is relevant to the domain of interest in the R&D effort launched under the DARPA PERFECT program. The basic trade-offs across the energy, performance and resilience dimensions as depicted in this paper are of course a strong function of the particular applications chosen for the workflow. The goal of the experimental analysis for the particular application sequence is to demonstrate the function and capabilities of our PEARL emulation framework, driven by the R-API smart user interface.

Our key conclusions are: (a) under tight power constraints, for workflows that exhibit little variation in power consumption across component application segments, the resilience-related constraints leave little room (5% at best) in improving energy efficiency over a simplistic baseline; (b) if the max-power constraints are relaxed, there is a significant opportunity (17%) in optimizing the efficiency, despite stringent SER related constraints; (c) if the inter-application power variation across the workflow is large, then the potential improvement opportunity can be large (40%).

IX. Acknowledgement

This work is sponsored by Defense Advanced Research Projects Agency, Microsystems Technology Office (MTO), under contract no. HR0011-13-C-0022. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government. This document is: Approved for Public Release, Distribution Unlimited.

References

[1] P. N. Sanda *et al.*, "Soft-error resilience of the IBM POWER6 processor." *IBM Jour. Res. and Dev.*, vol. 52, no. 3, pp. 275–284, 2008.

[2] E. Cannon, "SER Evaluation and Modeling," http://www.sematech.org/meetings/archives/reliability/20041025/SER/03SER_Cannon_TRC2004.pdf, Oct 2004.

[3] S. Taylor, "POWER7+: IBM's Next Generation POWER Microprocessor," in *Proc. of Hot Chips 24*, 2012.

[4] C. Lefurgy *et al.*, "Server-Level Power Control," in *Proc. of Autonomic Computing*, 2007.

[5] M. Floyd *et al.*, "Introducing the Adaptive Energy Management Features of the Power7 Chip," *IEEE Micro*, vol. 31, no. 2, pp. 60–75, March 2011.

[6] P. Kudva *et al.*, "Fault Injection Verification of IBM POWER6 Soft Error Resilience," in *Proc. of the Workshop on Architectural Support for Gigascale Integration*, 2007.

[7] S. S. Mukherjee *et al.*, "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor," in *Proc. of Int'l Symp. on Microarchitecture*, 2003.

[8] J. A. Rivers *et al.*, "Phaser: Phased methodology for modeling the system-level effects of soft errors," *IBM Jour. Res. and Dev.*, vol. 52, no. 3, pp. 293–306, 2008.

[9] J. E. Mitchell, "Branch-and-cut algorithms for combinatonial optimization problems," *Handbook of Applied Optimization*, pp. 65–77, 2002.

[10] C. Isci *et al.*, "An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget," in *Proc. of Int'l Symp. on Microarchitecture*, 2006.

[11] P. Pillai *et al.*, "Real-time Dynamic Voltage Scaling for Low-power Embedded Operating Systems," in *Proc. of Symp. on Operating Systems Principles*, 2001.

[12] V. Devadas *et al.*, "Real-Time Dynamic Power Management Through Device Forbidden Regions," in *Proc. of IEEE Real-Time and Embedded Technology and Applications Symp.*, 2008.

[13] X. Qi *et al.*, "Power Management for Real-Time Embedded Systems on Block-Partitioned Multicore Platforms," in *Proc. of Int'l Conf. on Embedded Software and Systems*, 2008.

[14] Y. Zhang *et al.*, "A Unied Approach for Fault Tolerance and Dynamic Power Management in Fixed-Priority Real-Time Embedded Systems," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 1, pp. 111–125, 2006.

[15] B. Zhao *et al.*, "Reliability-aware Dynamic Voltage Scaling for Energy-constrained Real-time Embedded Systems," in *Proc. of Int'l Conf. on Computer Design*, 2008.

[16] R. A. Shafik *et al.*, "Soft Error-Aware Voltage Scaling Technique for Power Minimization in Application-Specific Multiprocessor System-on-Chip," *Journal of Low Power Electronics*, vol. 5, no. 2, pp. 145–156, 2009.