Understanding the Power-Performance Tradeoff through Pareto Analysis of Live Performance Data

Junya Michanan, Rinku Dewri, Matthew J. Rutherford
Department of Computer Science, University of Denver
Denver, Colorado U.S.A.
jmichana@du.edu, {rdewri, mjr}@cs.du.edu

***Abstract*—Optimizing the power-performance tradeoff of a software system is challenging as the design space is large and live data is difficult to obtain. As a result, many power reduction techniques are based on power models which may not represent the full complexity of the system being analyzed. In this paper, in contrast, we propose a process for performing a tradeoff analysis using live power/performance data. As a case study, we conduct an empirical evaluation of the power/performance impact of cache configuration on embedded systems. We gather live power consumption and execution time data for the programs in the CHStone benchmark suite on an embedded processor with configurable cache parameters and perform a Pareto analysis on these data to identify the optimal cache configurations. We observe that the optimal configurations are sparse in the design space, are inconsistent across the benchmark, and are counterintuitive in some cases. Our results reveal interesting, unexpected insights motivating the need for tools and methodologies that automate this process and operate directly on data gathered from the systems.**

Keywords—Energy; Power; Efficiency; Performance; Tradeoff; Optimization; Cache; FPGA; Pareto

I. INTRODUCTION

Power-performance optimization is challenging and becoming increasingly important among modern computer systems, especially for those that rely on battery power. The sophistication of software applications and the increasing needs of rich media and big data have made today's computer systems power-hungry, while battery standards are not keeping pace with the demand [14]. Therefore, many researchers have been developing optimization techniques to extend battery life and reduce power consumption while maintaining other performance characteristics at acceptable levels.

Many power reduction techniques are based on power models which might not represent the full complexity of the system being analyzed. Most computer systems are not originally designed to support power optimization so the onboard power monitoring systems are not included, or if included, they are not explicitly designed to measure the power consumption of software applications [7, 15]. Many power models have been developed to support power optimization [4, 10, 13, 15, 16]. They are mostly intended to evaluate a specific platform or specific technology [16]. As with all models, if there are errors with calibration or inaccuracies in the models, or if they are used incorrectly, the results can be skewed or different from those based on analysis of live power consumption data [7]. In order to avoid the use of power models, we focus on the use of live data.

Optimization with live data is difficult: the process of gathering and analyzing these data is tedious and understanding conflicting performance attributes is challenging. In software engineering, performance and power consumption are viewed as non-functional properties. They are considered conflicting attributes and are often traded off, making them difficult and time consuming to optimize [6]. Many researchers point out that high-level strategies can help in trading off the conflicting properties and solving the multi-objective optimization problem [6, 9]. Although their results are intuitive and feasible, there are still many open challenges and the strategies are far from being adopted into practice.

In this paper, as a case study, we conduct an experiment on an embedded hardware platform that can run a wide variety of software applications while providing live power consumption data. We investigate one aspect of the system, the cache system, because it has a major impact on both power consumption and execution time, and virtually all computer systems use caches [10]. Several other studies have shown that the cache has a large effect on the overall system performance and also accounts for a large amount of total energy consumption in embedded systems; up to 50% of total energy usage in some cases [13]. Also, there are many tunable parameters in most cache designs [11, 13].

We select Pareto optimality as the main principle to solve the bi-objective optimization problem because it is well-known and has been applied in many fields, including engineering and economics where optimal decisions need to be made in the presence of tradeoffs between two or more conflicting objectives [1]. Our goals are to demonstrate a detailed manual optimization process and to convey the basic concept of power-performance tradeoff in an energy-aware system and to understand the impact that different cache parameters have (or do not have) on the power/time tradeoff in order to better understand how an automated optimization methodology for performing this analysis might work.

We consider power consumption data (watts) for the analysis instead of energy (joules) because we want to look at the system's power consumption and performance as a whole not the specific software being executed. We consider these properties to be independent from each other. Power consumption and execution time are just a few of the many performance and non-functional properties of a system [7]. Our

978-1-4799-6177-1/14/$31.00 ©2014 IEEE

goal is to observe the interactions between the system's power consumption and execution time as effectuated by different cache system parameters when executing different benchmark programs.

The contributions of this paper are threefold: (1) the demonstration of a detailed manual process for power-performance tradeoff analysis using Pareto optimality and how some unexpected insights can be discovered and categorized. (2) To provide evidence that some optimal configurations might not be as expected when analyzing the live power consumption data. Our test results show that the optimal configurations can be sparse, inconsistent and in many cases counterintuitive, making automated optimization processes hard to implement without analysis from actual data. (3) To provide some useful test results of FPGA cache configurations and to demonstrate that the optimal cache configurations do exist in the selected CHStone benchmarks.

II. BACKGROUND

In the existing literature on power/performance tradeoffs, proposed techniques target improvements over the base system without using Pareto optimality. They often fail to address the overall space of possible solutions without knowing whether their chosen solution is optimal (where they are on the Pareto front). Much of the research is conducted without the understanding of the power-performance interactions at the system level. As stated in [8], observation of a lack of Pareto optimality is an alert to an opportunity to improve the design that might be missed, especially when no single engineer understands all the design dependencies. By applying the Pareto optimality principle with all possible solutions for the development of an efficient energy-aware system, we come up with the following hypothesis for the experiment:

- There exists a Pareto optimal curve on a solution space so that power and performance can be traded off at different weights.
- If the curve is sparse, the development of the efficient energy-aware system is difficult.

Based on our hypotheses, without a Pareto optimal analysis, it is hard to demonstrate that an improved result is optimal. It is possible that the reported result might in fact be suboptimal, far from a Pareto optimal curve. In that case, the work done could be wasted as the solutions do not encompass all the necessary elements.

A. The FPGA Cache System

To study the cache parameters of an embedded system in our experiment, we select an Atlys development board, a complete, ready-to-use digital circuit development platform based on a Xilinx Spartan-6 LX45 FPGA [2]. All of the hardware platforms configured for the experiments are based on Xilinx's MicroBlaze, a FPGA soft processor core that includes advanced architecture options like AXI or PLB interface, Memory Management Unit (MMU), Floating-Point Unit (FPU), instruction and data cache among other capabilities [11]. For MicroBlaze, the AXI System Cache soft-peripheral system used for the study is viewed as a direct-mapped L2 Cache and is highly configurable. The available cache configuration options in the Atlys board include cache size, cache line length, number of stream buffers, number of victims and write-back storage policy (all options are listed in Figure 3). Note that the MicroBlaze Cache configuration parameters are preset with some default values and the data cache write-back storage policy is disabled by default.

B. Pareto Optimality for a Typical Power-Performance Tradeoff

As an example of power-performance tradeoff analysis, the design process we consider can be viewed as solving a bi-objective optimization problem, where we seek a cache configuration that minimizes two objectives, namely the execution time of applications in the system, and the power consumption of the system. Choices in configuring the system are generated by varying multiple cache-related parameters. In Pareto optimality, all objectives are treated equally. The "optimal" solutions found in a Pareto analysis together form the Pareto set or the Pareto front [17]. Solutions in the Pareto set reflect tradeoffs in the achievement of the different objectives. The selection of these solutions is based on the concept of dominance—a solution is worse than another only if it is so in all the objectives in the problem [1].

A scatter plot of the objective values corresponding to Pareto optimal configurations (also called a Pareto curve) can give system designers and software developers an overview of how power and performance interact in the system. It can help them design optimization algorithms for an efficient energy-aware system that can handle a wide variety of power-performance requirements. With these algorithms, an energy-aware software system can have the ability to adapt its power consumption behavior at different stages during program execution. For example, when the battery level in a system is low, applications may be forced to run at a degraded performance level in order to induce a lower power consumption rate. Algorithms could navigate possible choices on the Pareto curve so that the performance of the applications is minimally affected even with reduced power availability.

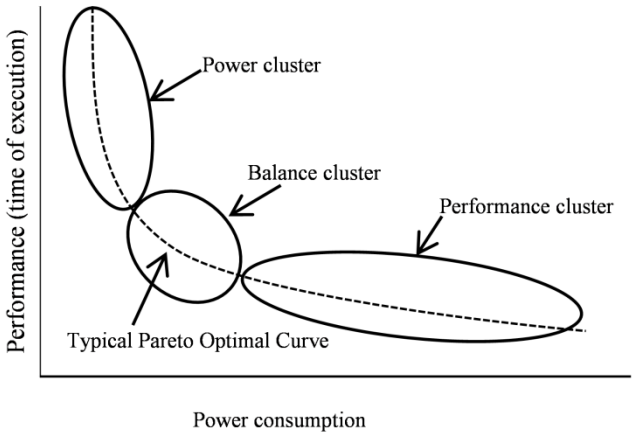

Fig. 1. A Pareto Optimal curve and clusters for typical power-performance tradeoffs

We generally categorize the power-performance requirements of a system into three types—performance-favored, power-favored and balanced. The performance-favored type is a system that demands fast execution time over

power consumption, while the power-favored type is a system that demands low power consumption over faster execution time. The balanced type is sought in a software system where both power consumption and performance are deemed equally important. Similarly, on a typical Pareto optimal curve, we can categorize the solutions into three clusters—performance, balanced and power (Figure 1). As can be seen in the figure, configurations in the power cluster allow flexibility in adjusting the performance of the applications, with no significant impact on the power consumption. While we hypothesize that a Pareto optimal curve will conform to this typical picture, the existence (or non-existence) of one or more cluster types is a characteristic of the application(s) under test. Further, a cluster may be dense, including a large number of configurations to choose from, while another may be sparse, with a significantly fewer number of choices.

III. Related Work

Most related work either does not include the Pareto optimality principle or analyzes power data derived from power estimation models. For example, the research in [9] focuses on high-level strategies through an approach for mapping software design to power consumption and exploring how high-level design decisions affect an application's energy usage. The results from this study show that applying design patterns can both increase and decrease the amount of energy used by an application; and design patterns within a category do not impact energy usage in similar ways. While this is just one study, the results imply that it is unlikely that impacts on the energy usage can be precisely estimated by only considering design-level artifacts. The research uses live power data but focuses on reducing energy consumption based on different software designs and does not address other non-functional properties, the tradeoff process nor include Pareto optimal analysis which are likely to be important in general.

Another interesting project related to our work is GISMOE [6]. This project sets out an alternative vision for a software development environment that can automatically generate a set of candidate program implementations, called Pareto program surface, with different non-functional attributes. At present, GISMOE is a proposed high-level architecture and set of principal features of the development environment. Although their concept is related to our research, unlike our study, their research is speculative rather than based on empirical evidence. Their high-level abstractions of Pareto analysis might hide some unexpected insights producing an inaccurate Pareto surface. Unlike the approach proposed in GISMOE, we do not transform or change any of the benchmark program code. We instead focus on the cache system as that has been shown to have impact on both the software performance and power consumption [13]. Although our manual process of gathering these data is tedious, the analysis results from CHStone benchmark suite yield useful information and provide a foundation toward efficient energy-aware systems and GISMOE.

There is also research related to energy efficiency and power-performance tradeoffs on the system cache [4, 10, 13]. However, the tradeoff techniques are either hardware/software specific or require additional hardware or features built in. The analyses are based on estimated power data using a power model of memory access. Their main focus is for designing optimal cache architecture and developing energy-efficient cache hardware, not for the whole system in general. In particular, the study in [10] is similar to ours but its main objective is for designing power efficient cache hardware systems. The study also does not include the Pareto optimality in their tradeoffs and their power data for the analysis are based on power models.

IV. Experiment

A. Experimental Setup

To perform our experiment, we developed a custom power monitoring and profiling tool for the Xilinx Atlys FPGA board [2] using the APIs and drivers provided by the manufacturer. The board is equipped with four on-board power-supply monitors with accuracy within 1%. The tool also annotates power consumption data with time stamps at a fine-grained resolution with an average of about one sample every two milliseconds. The monitoring tool simultaneously records power data in real-time for all four power-supply rails. In our experiments, we are primarily interested in the 1.2V and 1.8V rails which correspond to CPU and memory operations.

The experiment uses the CHStone benchmark suite version 1.6 as standard workloads for creating power consumption profiles. The suite is designed for C-based high-level synthesis, and is easy to use since the programs are self-contained and require no external libraries [5]. The CHStone suite consists of 12 programs taken from widely-used applications in the real world from various application domains—four arithmetic programs, four media applications, three cryptography programs, and one processor. We are able to compile and run 11 out of the 12 programs on the Atlys board (presented in Table I and Figure 3). For each hardware system with different cache configurations, the 11 programs were executed and profiled.

B. Experimental Method

There are three main steps in conducting the experiment—implementing the hardware platforms, profiling the software's power and performance, and analyzing the resulting data.

1) Implementing the Hardware Platforms

In the first step, we use the Xilinx EDK tool to design and implement each individual hardware platform with different cache parameters for the experiments. The system design and specifications of the hardware platforms are based on the Atlys Base System specifications provided by the manufacturer [3]. The only non-default parameter in each platform is the cache configuration parameter. The idea is not to build all hardware platforms for every possible combination of cache configurations but to vary only the controlled parameters from the base system, while leaving the other parameter values at the defaults. The purpose is to see how the dependent variable impacts the power consumption and performance of the base system. Based on the FPGA cache properties, we implement 36 hardware platforms, each labeled with a code—for example, ICS-64B for Instruction Cache Size of 64B (the rest of the parameters are set to the default values), DCS-512KB for Data Cache Size of 512KB, DC-WB-VIC4 for Data Cache Write-

Back Storage Policy Enabled with 4 victims, DC-LL-4W for Data Cache Line Length of 4 words (see Figure 3).

2) Software Power/Performance Profiling

The second step is to create a software project for each CHStone benchmark program on each hardware platform using the Xilinx SDK. With the combination of all 36 hardware platforms and 11 benchmark programs, a total of 396 software projects are built for the experiment. Since most of the benchmark programs are small and have short execution time, we modify the main programs to execute each benchmark multiple times. This provides for longer execution time so that the power monitoring tool can capture enough power data for the calculations in the next step. Also, during the data collection, all the print commands have been commented to minimize the CPU overhead caused by the commands.

3) Analyzing the Result Data

Power/performance profile data collected from each benchmark execution contain raw sample readings of power and execution times. The average power of each power rail is calculated as the sum of all power readings divided by the number of samples read from the start to the end of the program execution. The time of execution per iteration is calculated by the total time of execution from start to end divided by the number of times a benchmark was executed. For 396 projects, we have collected pairs of average power and time of execution per iteration for each program execution. These values are used in the Pareto optimal analysis of each benchmark described next.

V. Results and Discussion

With the data gathered we identify the set of Pareto optimal configurations for each benchmark. Note that the difference in power/time values may be very small across certain configurations; accordingly some points may look identical on the plots. By using the K-Means clustering algorithm [12], the Pareto front is divided into three clusters. These clusters may not exactly signify the three cluster types as discussed in Section II B; we manually merged one or more clusters generated by the K-Means algorithm in order to retain the underlying meaning of the three cluster types. A total of 11 such scatter plots are produced, one for each benchmark program. Figure 2 shows plots and clusters for three programs (ADPCM, AES and DFDIV) in the CHStone benchmark suite. The result data is also translated into a profile table (Figure 3) to be used in the power-performance tradeoff analysis.

A. Power-Performance Tradeoff Result

For each of the plots shown in Figure 2, the red dots (solid) represent the Pareto optimal cache configurations, while the blue points (hollow) represent suboptimal cache configurations. The groupings depict the three clusters—power, balanced and performance. Across all 11 benchmarks, we summarize the experimental result as follows:

(1) All graphs reveal promising Pareto optimal curves showing that the cache is a good candidate for power-performance optimization in an efficient energy-aware embedded system development.

(2) The results reveal that only a small numbers of choices of cache configurations (as low as 3 choices from the total of 36 cache configurations) are optimal for power-performance tradeoffs (see Table I).

TABLE I. Number of Pareto optimal solutions and negligible ones by benchmark program

Benchmark Programs	Number of Pareto optimal solutions	Percentage of all cache configurations (out of 36)	Number of negligible Pareto optimal solutions
ADPCM	11	30.56%	2
GSM	5	13.89%	2
MOTION	3	8.33%	1
AES	9	25.00%	4
BLOWFISH	11	30.56%	0
SHA	8	22.22%	2
DFADD	4	11.11%	3
DFDIV	4	11.11%	1
DFMUL	3	8.33%	0
DFSIN	11	30.56%	1
MIPS	4	11.11%	1
Total	**73**		**17**

(3) The clusters show that a small set of configurations optimally satisfies all three types of system requirements (performance-favored, power-favored and balanced).

B. Optimal Cache Configuration Result

With some post-Pareto analysis, power/execution time data from the benchmarks are put into a tradeoff profile table, as shown in Figure 3. This Figure demonstrates how tradeoff analysis result data can be displayed, providing the power/performance visibility and initial guidelines for tradeoff purposes. Other information can be added depending on the requirements and selection criteria. In Figure 3, in addition to the Pareto analysis results, there are also data from using the K-Means and Normalized Distance to Ideal Point [12] methods as the post-Pareto analysis for clustering the optimal points in the scatter plots (e.g., Figure 2). As an example, our tradeoff profile table contains 36 rows representing 36 cache configurations and 11 columns for 11 benchmark programs. The table cells highlighted with red (darkest in grayscale) are the Pareto optimal cache configurations. The green (gray in grayscale) and light gray cells are the 2nd and 3rd iterations of Pareto analysis respectively. The second iteration of Pareto analysis is done after removing the Pareto optimal configurations from the design space. The 3rd iteration is done after removing the configurations found optimal the 1st and 2nd iterations.

Multiple iterations of Pareto analysis is useful when the power/performance data exhibit low variability and points on the scatter plot are close together (as seen in Figure 2(a) inside

the balanced cluster). Some configurations result in similar power-performance outputs—we found that some values are equal up to the 3rd decimal place. The purpose of this analysis is to provide additional choices from the suboptimal solutions, which might be meaningful to developers or system designers. It also provides a better picture of how each cache property impacts power and performance on different benchmarks.

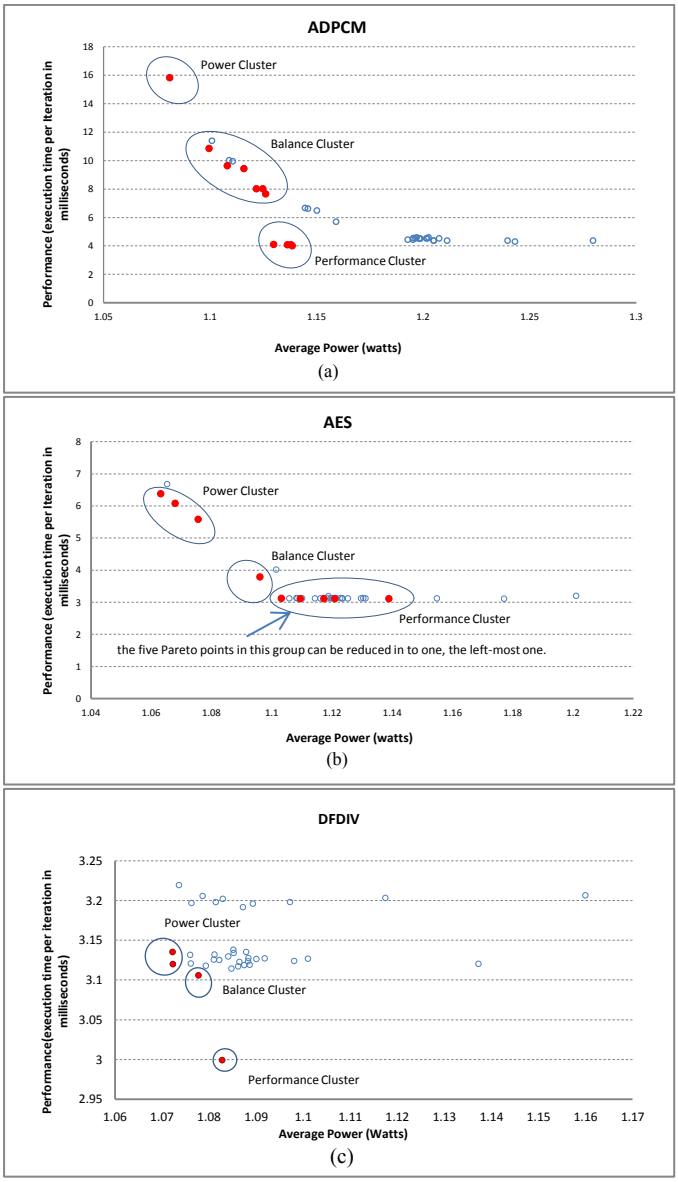

Fig. 2. Examples of Pareto optimal cache configurations and clusters of four programs in the CHStone benchmark.

In Figure 3, all decimal numbers in the cells are the Normalized Distance to the Ideal Point *(0, 0)* in normalized scatter plots—the modified version of scatter plots in Figure 2 but with both scales of execution time and power axis normalized to the scale from 0-10 units. The ideal point *(0, 0)* signifies a system that can run a program at the fastest speed (lowest execution time) and consume the lowest possible power, both ideally zeroes. The bold, italic and underlined numbers indicate a Pareto optimal configuration with the lowest normalized distance to the ideal point, among all optimal cache configurations for the same benchmark program. For example, in the ADPCM column, data cache with write-back policy enabled and number of victim 0 (row DC-WB-VIC0) is one of the optimal cache configurations. In addition, the bold italic and underlined notation ***CT(8.98525)*** implies that 8.98525 is also the lowest distance to the ideal point (0, 0) among the 11 optimal choices (red or darkest cells in ADPCM column)—meaning that when running program ADPCM on the system we just need to enable write back policy for the data cache and set the number of victim of the cache to 0 to get the optimal power-performance result.

In the red-highlighted cells (or darkest cells in grayscale), there are also the letters CT, CB or CP. The letters indicate clusters in the scatter plots or categories of the Pareto cache configurations. CT stands for performance cluster, CB for balance cluster and CP for power cluster. This means that, for our selected example, DC-WB-VIC0 is an optimal cache configuration categorized in the performance cluster (CT)—meaning that it is one of the optimal choices suitable for running ADPCM benchmark program in a system state where performance is more important than power, or execution time is more important than the power consumption.

By looking horizontally across all benchmarks in the power-performance tradeoff profile (Figure 3) to count the number of the red (or darkest) cells, and the data in Table II, we can additionally summarize the optimal cache configuration result related to cache properties as follows:

(1) Data and instruction cache size, and write-back policy are the most influential cache properties in the power-performance tradeoff analysis. They account for most of the Pareto optimal configurations. In Figure 3, I-Cache, D-Cache size and D-Cache Write-Back rows have the highest combined number of red cells or darkest cells across all benchmark programs. Also in Table II, the first three rows cover more than 95% of all optimal configurations. This means that when trying to configure the cache parameters for the optimal power/performance, we just look at these three cache properties as the main starting points for power-performance optimization.

(2) Data and instruction cache with larger size (2K and above), line length, instruction cache's string buffer and number of victims did not contribute significantly in the optimal achievement of a power-performance tradeoff. In Figure 3, minor rows of cache size 2K and larger and the last three major rows of the table do not have as many red cells as the previous three cache properties. Table II shows that the last three major rows only cover 4.1% of the optimal cache configurations. It seems that they can be ignored for most applications and are not the first candidate for power-performance optimization. This is counterintuitive.

(3) Manipulating the write-back policy on the data cache is recommended for performance-favored and balanced systems, while the instruction and data cache size are recommended for power-favored and balanced interactions. In Figure 3 and Table II, the write-back policy row has the highest number of red cells with CT (performance cluster) letters.

(4) Smaller instruction cache size (smaller than 2K) is recommended for power-favored optimization since this property produces the lowest power consumption with the graceful degradation in performance. In Table II, the instruction cache size row has the highest number of optimal configurations categorized in power cluster (CP). Also, in Figure 3, minor rows of I-Cache size from 64 to 2K bytes cover the majority of the optimal cache configurations. While the I-Cache with larger size than 2K does not produce many optimal cache configurations.

C. Insight Summary

Based on our observations from the experimental results, we summarize our insights as follows:

1) Only a few cache configurations are optimal for power-performance tradeoffs

All 11 benchmarks reveal only a small number of optimal (non-dominated) cache configurations from a relatively large cache configuration space. For example, as summarized in Table I, the smallest number of optimal cache configurations are from the MOTION and DFMUL benchmarks, with only 3 Pareto points from 36. These account for about 8% of the cache configuration space that we evaluated and therefore the other 92% could have been ignored or removed from the solution space. On average, the benchmark programs produce 6.6 Pareto points or about 18% of the solution space. The number is relatively small and more than 80% of solution space can be disregarded or avoided on average. Smaller optimal choices can contribute to less complexity in the calculations of an optimization algorithm. In this case, there are fewer options for the system to switch the cache configuration parameters and adapt itself to its current state of power/performance requirements.

2) Optimal cache configurations are sparse, inconsistent and sometimes counterintutive

The results are sparse and inconsistent because not all Pareto curves are in the expected Pareto shape. As seen in the examples, Figure 2(a), 2(b) and 2(c), some sections of the curves form vertical or horizontal lines. This means that by changing from one optimal configuration to another on the Pareto curve, there is no significant improvement on either power consumption or system performance. In this case, some optimal solutions can be ignored and one solution on each section can be selected as the representative optimal solution in the group. We call the ignored optimal solutions "negligible" because, by definition, they are insignificant or unimportant as to be not worth considering.

For example, in Figure 2(a) and 2(b) in the performance clusters, most of the optimal cache configuration points are flat and form horizontal lines on the performance axis (time of program execution). By changing from one optimal configuration to another, there is not a significant gain in performance. Therefore, only one optimal solution should be retained in this case, the one with the lowest power consumption (best solution in term of power consumption), which is the left-most one in the cluster.

		Media			Cryptography			Arithmetic				Processor
		ADPCM	GSM	MOTION	AES	BLOWFISH	SHA	DFADD	DFDIV	DFMUL	DFSIN	MIPS
I-Cache Size	ICS-64B	CP(12.10407)	CP(12.6106)	CP(13.29683)	CP(11.66424	CP(12.034389)	CP(12.49671	13.28304	CP(13.25121)	12.97542	CP(11.86953)	13.4829083
	ICS-128B	10.57308	CB(11.5302)	13.4135	11.9359	10.5664502	CP(11.28324	CP(13.30432)	13.45107	12.97074	CP11.62412)	CP(13.318208
	ICS-256B	10.18704	***CB(11.3650)***	13.47432	CP(11.4419)	CB(10.505671)	9.479168		CP(13.21924)	***CB(12.8345)***	CP(11.58594)	CP(13.336081
	ICS-512B	10.18041	11.45963		CP(11.07766)	10.53867825	9.51649	13.51082	***CT(13.0144)***		CP(11.48474)	13.3917808
	ICS-1KB	CB(10.05921)	11.47495		10.14644	CB(10.450580)					CP(11.22462)	13.4487865
	ICS-2KB	CB(9.733527)	11.4697		9.73237	CB(9.4483732)				12.96558	CP(10.94869)	
	ICS-4KB	9.555242	11.4835	13.50794								
	ICS-8KB	9.50268			9.779774				13.29399			
	ICS-16KB	9.582617		13.44362	9.843192					13.0386	CT(9.766825)	
	ICS-32KB	9.629181	11.59119		CT(9.906587)						9.801238	13.3215064
	ICS-64KB	9.85685			10.19131						10.04601	
D-Cache Size	DCS-64B	CB(10.39008)		13.33218	CB(9.968029)	CB(10.018033)	CB(9.44259)	***CT(12.91722)***	CB(13.2192)	12.99141		***CT(13.271123)***
	DCS-128B	CB(10.06734)	11.47932		9.670209	CB(9.8440253)	CB(9.458805		13.26656	CT(12.8473)		13.4113525
	DCS-256B	CB(9.715121)			9.692222	CB(9.7426903)	CB(9.448334		13.25611	12.98704		13.432871
	DCS-512B	CB(9.650008)	11.51831		9.696039	CB(9.6289988)	9.469237	13.31465	13.24286	CP(12.9463)		13.4231817
	DCS-1KB	9.554246				CB(9.4749122)		CB(13.15027)				
	DCS-2KB	9.463525	CT(11.4631)		9.745117	CB(9.4095792)	9.489417		13.27934			
	DCS-4KB	9.553731			CT(9.750549)				13.28821			
	DCS-8KB	9.518849	CT(11.4893)	CT(13.40506)								
	DCS-16KB	9.58346	11.5489									
	DCS-32KB	9.839793										13.404553
	DCS-64KB	10.14061										
D-Cache Write-Back Policy Enabled with different Number of Victim	DC-WB-VIC0	***CT(8.98525)***	11.46333	13.34831	***CT(9.648599)***	***CB(8.9756610)***	***CT(9.294251***	CP(13.26862)	13.41464	13.12777	***CB(9.26888)***	CB(13.277194
	DC-WB-VIC2	CT(9.031878)		13.37705	CT(9.693676)	9.000216265	9.337504		13.28224		9.31699	13.3225436
	DC-WB-VIC4	CT(9.044664)		13.40549	9.698613	9.055877747	CT(9.372639				CB(9.30665)	13.3281005
	DC-WB-VIC8	CT(9.037831)	11.47831	***CB(13.25647)***	9.689596	9.048775599	CT(9.355299)	13.32929			9.333707	13.3147547
Line Length	DC-LL-4W	9.554812										
	DC-LL-8W							13.37339				
	IC-LL-4W			13.49192	CT(9.776797)		9.508951					
	IC-LL-8W	9.555049									CT(9.62897)	
I-Cache String Buffer	IC-NUM-STR0	9.5362						13.36616				13.4715134
	IC-NUM-STR1			13.43333							CT(9.68828)	
I-Cache Number of Victim	IC-VICTIM-0	9.543207										
	IC-VICTIM-2											
	IC-VICTIM-4			13.43096				13.37133				
	IC-VICTIM-8									13.02388		

Optimal Cache Configuration Secondary Optimal Cache Configuration Tertiary Optimal Cache Configuration

Fig. 3. A power-performance tradeoff profile of the Pareto analysis result (CT = Performance cluster, CB = Balance cluster and CP = Power cluster; bold, italic and underlined numbers indicate the minimum normalized distance to ideal point of a benchmark)

Similarly, in Figure 2(b), the five optimal configurations in the performance cluster can be collapsed into one optimal solution; the other four can be ignored because they do not create any significant performance improvement. In Figure 2(c) inside the power cluster, the two cache configurations on the DFDIV plot graph form a straight vertical line on the power axis. There is one additional negligible Pareto point (the top one) that can also be removed from the optimal cache configurations. All negligible Pareto points in the benchmark programs are presented in Table I. Not all plot graphs are shown in the paper due to limited space.

TABLE II. COUNTS OF FIRST ITERATION PARETO POINTS BY CACHE PROPERTY AND CLUSTER

	Power Cluster (CP)	**Performance Cluster (CT)**	**Balance Cluster (CB)**	**Total**
I-Cache Size	20	3	8	31(42.5%)
D-Cache Size	1	7	16	24 (32.9%)
Write-Back Policy	1	9	5	15 (20.5%)
Line Length	0	2	0	2 (2.7%)
I-Cache String Buffers	0	1	0	1(1.4%)
I-Cache # of Victims	0	0	0	0 (0.00%)
Total	**22**	**22**	**29**	**73 (100%)**

Our analysis also reveals unexpected insights from these irregular Pareto shapes and the negligible cache configurations. One most obvious result is revealed by AES benchmark in Figure 2(b) inside the performance cluster. Four out of the five optimal cache configurations inside the cluster are related to data cache write-back policy with different number of victims (0, 2, 4 and 8). A victim is a cache line that is evicted from the cache. If no victims are saved, all evicted lines must be read from memory again, when they are needed. Conventional wisdom states that by saving the most recent lines data can be fetched much faster, thus improving performance [11]. However, for this particular AES benchmark, our result clearly shows that not all victim policies contribute to significant improvement in performance. This evidence indicates that the conventional wisdom is not always accurate.

Additionally, in Figure 2(c), the example is not so obvious but there is a small evidence of counterintuitive insight. In the power cluster, the two optimal cache configurations are related to instruction cache size—64B and 256B respectively. In this case, changing from one I-Cache size to another does not yield significantly different power consumption. This is somewhat unexpected and contradicts studies relating cache size and system power consumption. For example, a study in [10] states that "increasing cache line sizes tends to consume more energy" and, for I-Cache size less than 4K, the smaller size tends to consume more power.

3) There exists cache configurations with certain properties that do not produce any Pareto optimal points

In our cache solution space, we found that cache configurations related to I-Cache victims (the last row in Figure 3 and Table II) do not produce any Pareto points. Also, cache configurations with I-Cache string buffer and D-Cache/I-Cache line length do not produce a significant number of Pareto points (second and third from the last row in Figure 3 and Table II). These cache properties are therefore not candidates for power-performance optimization and can be ignored completely. Our rankings in Table II and colored cells in Figure 3 can help us identify the "hot spots", the best candidates for power-performance optimization and the areas having less impact on the system's performance and power consumption. From our case study, the hot spots for cache configurations are the top three in Table II, in which they produce the most red cells in Figure 3. The table also shows that up to 50% of the total effort put into the cache based optimization can be cut (last 3 out of six cache properties in the table).

4) There exists at least one cache configuration that can fulfill each of the typical power-performance tradeoff requirements

As mentioned in Section II B, our analysis shows that there exist optimal cache configurations that can fulfil the three types of system requirements for a typical power-performance tradeoff—power-favored, performance-favored and balanced. As shown in Figure 2 and summarized in Table II, the Pareto points can be clustered into performance, power and balance clusters using the K-Means algorithm, the same way as the typical Pareto Optimal curve and clusters in Figure 1. For power-favored, performance-favored and balanced requirements, our results recommend the cache configurations related to instruction cache size (I-Cache Size), data cache size (D-Cache Size) and Write-Back policy respectively.

D. Threats to Validity

There are several validity threats to the design of this study. For threats to internal validity, our study is limited to the configurable cache system of the FPGA embedded system, one aspect of the energy-aware system. There are also other areas at different system layers having impact on the system power consumption and execution times. The cache configuration results might be different if configured differently along with other system parameters. In extending this work we should also include other areas or combinations of different areas and parameters to capture more data and detailed statistical analysis as well as allowing a broader coverage on the power-performance tradeoffs. Also, the configurable cache systems are only based on the configurable setting and property values available in the Atlys FPGA. For other embedded systems, there might be different tunable cache configurations.

For each hardware construction, we only change a single cache parameter value at a time while leaving the others the default values. Therefore, the power consumption and execution values are only based on the default cache configuration as provided by the manufacturer. This does not cover all combinations of available cache configurations. For more complete results, more hardware platforms can be built

with other combination of the available cache setting values and more data can be collected for the better statistical analysis. Also, the power consumption data of the system are only of the CPU and Memory. There are also two other power monitor rails of Video/Audio and Ethernet ports have been ignored. This might affect the result if we also include in the analysis.

During the data collection process, because we have 396 software projects to execute, we only execute each benchmark on each hardware platform three times. Also, each benchmark program is set to execute multiple iterations for each run to make sure each benchmark execute in about 10 seconds. Therefore, the measured numbers can contain some overhead from these loops that control the executions. If we increase the number of executions, the data reading and statistical data would be more accurate. Also, the Pareto analysis result will be more accurate if conducted on a larger solution space. Our solution space might not large enough. The front of our Pareto optimal values is therefore considered an approximation to the "true" front. Further runs of the system with more cache configurations may improve the front approximation. Our selection of using Pareto optimality and the post-Pareto methods for the analysis is for demonstration purpose only. There are many other methods that can also be used to solve the power-performance tradeoff problems.

For threats to external validity, we try to generalize the result of our study to all software application domains. We select the CHStone benchmark suite because it covers multiple domains of real-world software applications—media, cryptography, arithmetic and processor. However, processor domain only contains one benchmark which might not have enough coverage. And, the generalizability of the result is only for these four software domains. There is also a risk that the result might not reflect all domains of complex software applications, since all the benchmarks are small programs and specially designed for embedded systems.

VI. CONCLUSION

As a case study, our research demonstrates a detailed power-performance optimization process that can be used in developing efficient energy-aware systems. Because the process is tedious, it is crucial that developers and researchers understand the manual process and are aware that unexpected insights discovery is important and not easy to do in real systems. In the process, we also gather some basic background of how we can use Pareto optimality in power-performance tradeoffs; how power-performance requirements and the optimal solutions can be categorized; and how the data are collected and analyzed and used. Along with this, we provide some useful test results of FPGA cache configurations and demonstrate that the Pareto optimal cache configurations do exist in the CHStone benchmarks.

Our results suggest that some optimal configurations might not be as expected when analyzing the live power consumption data. We observe that the optimal configurations are sparse in the cache design space, are inconsistent across the benchmark and counterintuitive in many cases, making power-performance optimization processes hard to implement without analysis from actual data. Our results also show that even something very low-level like the cache system (that might not be captured in a power model) can impact the power-performance analysis significantly and unpredictably. These results motivate the need for tools and methodologies that operate directly on data gathered from the systems themselves.

REFERENCES

[1] J. Arora. *Introduction to optimum design*. Academic Press, 2004.

[2] Digilent Inc., "Atlys Board Reference Manual", 2012, http://www.digilentinc.com/Data/Products/ATLYS/Atlys_rm.pdf.

[3] Digilent Inc., "Atlys board support files for EDK BSB wizard. Supports EDK 13.2 - 14.2 for both AXI and PLB buses", 2012, http://www.digilentinc.com/Data/Products/ATLYS/Atlys_BSB_Support_v_3_6.zip.

[4] A. Gordon-Ross, F. Vahid, and N. D. Dutt. "Fast configurable-cache tuning with a unified second-level cache." In *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* pp. 80-91, 2009.

[5] Y. Hara., H. Tomiyama, S. Honda, H. Takada, and K. Ishii. "CHStone: A benchmark program suite for practical c-based high-level synthesis." In *2008 IEEE International Symposium on Circuits and Systems (ISCAS 2008),* pp. 1192-1195, IEEE, 2008.

[6] M. Harman, W. B. Langdon, Y. Jia, D. R. White, A. Arcuri, and J. A. Clark. "The GISMOE challenge: constructing the pareto program surface using genetic programming to find better programs (keynote paper)." In *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE 2012)*, pp.1-14, 3-7 Sept 2012.

[7] J. C. McCullough, A. Yuvraj, J. Chandrashekar, S. Kuppuswamy, A. C. Snoeren, and R. K. Gupta. "Evaluating the effectiveness of model-based power characterization." In *USENIX Annual Technical Conf.*, 2011.

[8] C. J. Petrie, T. A. Webster, and M. R. Cutkosky. *Using Pareto Optimality to Coordinate Distributed Agents.* AIEDAM Special Issue on Conflict Management Vol. 9, pp. 269-281, 1995.

[9] C. Sahin, F. Cayci, I.L.M Gutierrez, J. Clause, F. Kiamilev, L. Pollock, and K. Winbladh. "Initial explorations on design pattern energy usage." In *2012 First International Workshop on Green and Sustainable Software (GREENS)*, pp.55-61, IEEE, 2012.

[10] C. Su and A. M. Despain. "Cache design trade-offs for power and performance optimization: a case study." In *Proceedings of the 1995 International Symposium on Low Power Design*, ACM, 1995.

[11] Xilinx Inc., "Field Programmable Gate Array (FPGA)," 2013, http://www.xilinx.com/training/fpga/fpga-field-programmable-gate-array.htm.

[12] N. Lopez, O. Aguirre, J. F. Espiritu, and H. A. Taboada. "Using game theory as a post-Pareto analysis for renewable energy integration problems considering multiple objectives." In *Proceedings of the 41st International Conference on Computers & Industrial Engineering*, pp. 678-683, 2011.

[13] C. Zhang, F. Vahid, and W. Najjar. "A highly configurable cache architecture for embedded systems." In *Proceedings of 30th Annual International Symposium on Computer Architecture,* IEEE, 2003.

[14] F. Rezzi, L. Collamati, M. Costagliola, and M. Cutrupi. "Battery management in mobile devices." In *Frequency References on Power Management for SoC and Smart Wireless Interfaces*, pp.147-168, Springer International Publishing, 2014.

[15] R. Mittal, A. Kansal, and R. Chandra. "Empowering developers to estimate app energy consumption." In *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*, pp. 317-328, ACM, 2012.

[16] S. Rivoire , P. Ranganathan , and C. Kozyrakis, "A comparison of high-level full-system power models." In *Proceedings of the 2008 Conference on Power Aware Computing and Systems*, pp.3-3, 2008.

[17] J. Horn, N. Nafpliotis, and D. E. Goldberg. "A niched Pareto genetic algorithm for multiobjective optimization." In *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, vol.1, pp. 82-87, 1994.