Evaluating Power Consumption of Parameterized Cache and Bus Architectures in System-on-a-Chip Designs

Tony D. Givargis, Frank Vahid, and Jörg Henkel, *Senior Member, IEEE*

***Abstract*—Architectures with parameterizable cache and bus can support large tradeoffs between performance and power. We provide simulation data showing the large tradeoffs by such an architecture for several applications and demonstrating that the cache and bus should be configured simultaneously to find the optimal solutions. Furthermore, we describe analytical techniques for speeding up the cache/bus power and performance evaluation by several orders of magnitude over simulation, while maintaining sufficient accuracy with respect to simulation-based approaches.**

I. Introduction

THE GROWING demand for portable embedded computing devices is leading to new system-on-a-chip (SOC) architectures intended for embedded systems [10]. Such SOC architectures must be general enough to be used across several different applications in order to be economically viable, leading to recent attention to programmable silicon platforms [8], [15], [16], [18]. Different applications often have very different power and performance requirements (and a single application may have different requirements over time). Therefore, these platforms must be adaptable not only to each application but also to different power and performance requirements.

A typical SOC architecture will have one or more caches and buses [19], which consume a large percentage of system power, especially in deep-submicrometer technology. The cache can be made parameterizable by allowing configuration of its features like line size, associativity, and total size. The bus can be made parameterizable by configuring it to use different sized and spaced wires and by enabling or disabling encoding techniques. For example, an application that does not benefit from large associativity would consume less power if the cache were configured for smaller associativity. Although this would result in unused transistors, it would reduce power by reducing comparisons. The new-found abundance of transistors on-chip enables design of such parameterized components. In fact, some platforms come in the form of intellectual property, such as a Hardware Description Language (HDL), and thus can be synthesized to have only those transistors necessary for a selected configuration.

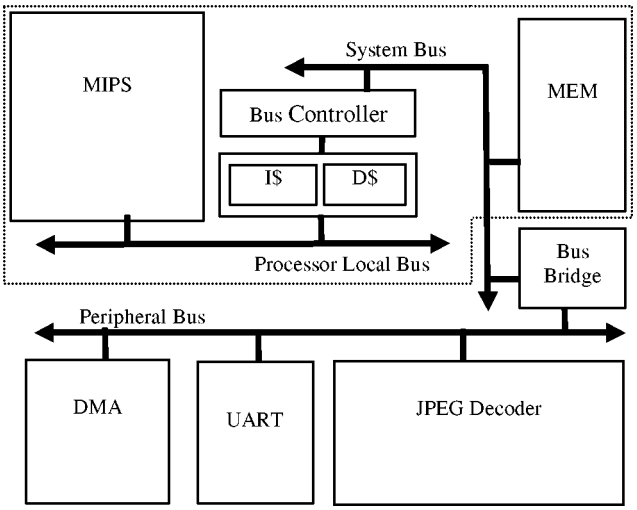

Fig. 1. Target architecture.

In this paper, we begin by describing our parameterized architecture. We summarize related work. We describe our experimental setup for evaluating the large power and performance tradeoffs possible by configuring our architecture, a setup based on accurate but time-consuming simulation. We then provide analytical techniques for speeding up the cache and bus evaluation by several orders of magnitude, while maintaining reasonable accuracy. We conclude that cache and bus parameters should be evaluated simultaneously, and that our analytical techniques are fast enough to permit extensive search for the optimal configuration for a given application and requirements.

II. Target Architecture

Fig. 1 shows the block diagram of our SOC target architecture. The whole system may feature a CPU, an instruction cache (I-cache), a data cache (D-cache), a main memory, peripheral units, and cores for diverse applications (like MPEG encoding, for example), as well as various buses to connect all these cores. Our investigations focus on the subsystem CPU, CPU-to-cache (processor local) bus, I/D-caches, cache-to-main-memory (system) bus, and the main memory. Our focus in this paper is on the enclosed area in Fig. 1, because it is the system

Manuscript received June 1, 2000; revised September 1, 2000. This work was supported by the National Science Foundation under Contracts CCR-9811164 and CCR-9876006 and by a Design Automation Conference Graduate Scholarship.

T. D. Givargis is with the Department of Computer Science and Engineering, University of California, Riverside, CA 92521 USA (e-mail: givargis@cs.ucr.edu).

F. Vahid is with the Department of Computer Science and Engineering, Univerity of California, Riverside, CA 92521 USA and the Center for Embedded Computer Systems, University of California, Irvine, CA 92717 USA (e-mail: vahid@cs.ucr.edu).

J. Henkel is with C&C Research Laboratories, NEC USA, Princeton, NJ 08540 USA (e-mail: henkel@ccrl.nj.nec.com).

Publisher Item Identifier S 1063-8210(01)03356-X.

part with heaviest traffic and thus accounts for the largest part of power consumption.

The I-cache and D-cache each have the following design parameters: cache size, line size, and associativity. Cache line is the size of a block of data that is loaded from main memory into the cache on a cache miss. Cache associativity is the number of slots that a particular cache line can be placed into once it is loaded from main memory. When cache associativity is set to one, a line of data will always map to a single cache line, when set to eight, a line of data can map to any of eight possible cache lines. The range of values considered are: cache sizes of (32 K, 16 K, 8 K, 4 K, 2K, 1K, 512, 256, or 128), line sizes of (8, 16, or 32), and associativity of (2, 4, or 8). Each bus has the following design parameters: number of data lines (32, 16, 8, or 4) with bus invert (on or off).

We assume that a parameterizable bus is in use such that the number of data wires in use can be selected to be one of 4, 8, 16, or 32. The result of such variation is that configuring the bus to use four wires will result in lowest coupling capacitance [1], since the spacing between the wires is increased. As a result, a bus with four wires will result in the lowest power per transfer, but more transfers and hence reduced performance. Conversely, using all 32 wires will result in the highest average capacitance, due to decrease in relative wire spacing. Note that when we say the size of the bus, we are referring to the number of wires that that bus is composed of and not the width of the wires. Instead, we assume that the routing area of the bus is constant for any size bus.

In our experiments, we focus on evaluating bus and cache parameters together in order to find the best tradeoff between power/performance/hardware effort.

III. Related Work

The related work important to our approach can be divided into three categories: work on system-level power optimization in general, architectural power optimization focusing on a single core (like a CPU, cache, main memory, etc.), and work on bus issues like power, performance, and size.

As for the first group, Dave *et al.* [2] introduce a codesign methodology that optimizes for power and performance at the task level. Their procedure for task allocation is based on an average power consumption and does not take into consideration data dependencies on the instruction level to estimate/optimize power. In addition, they do not take into consideration cache and bus effects in terms of power and performance. The system-level power optimization approach proposed by Hong *et al.* [6] has the same limitations regarding the addressing of architectural tradeoffs. Rather, they exploit the technique of variable voltage scaling in order to minimize power consumption.

At the architectural level for single system components (i.e., not considering any tradeoff between various system parts), high-performance microprocessors have been investigated and specific software synthesis algorithms have been derived to minimize power by Hsieh *et al.* [7]. Tiwari [17] investigated the power consumption at the instruction level for different CPU and digital signal-processing architectures and derived specific power optimizing compilation strategies.

An approach that is based on system level and takes into consideration the interdependencies between various system parts has been proposed by Li *et al.* [5]. Their target system features a CPU, a data cache, an instruction cache, and a main memory. The impact of buses is not accounted for. As a result, the evaluation will be less accurate in new technology, where routing capacitance will play a significant role in the system's power consumption. Fornaciari *et al.* [3] explore the impact of different bus encoding schemes for embedded systems in terms of power dissipation. Though they study power consumption for various cache sizes, they do not explore all relevant interdependencies (e.g., cache line size, associativity, main memory size, etc.). So, their approach basically assumes that most system parameters have already been determined, and they focus on finding the best bus encoding scheme.

A key contribution in reducing bus power has been the encoding of data, such as bus-invert [12]. Here, the Hamming distance of two consecutive data words is computed. If this distance is greater than half the word size, the inverted data are sent and a control signal is asserted to signal decoding of the data on the receiver side. Using bus-invert, a theoretical power savings of 25% average and 50% peak is obtainable. However, bus-invert coding may increase the bus cycle time, resulting in a performance penalty. Limited-weight codes [13] are generalized encoding schemes that, using more than one control signal, reduce average bit transitions on the bus, resulting in even lower power consumption.

Our approach is more comprehensive than approaches proposed so far. We take into consideration most of the relevant parts of a whole embedded system (CPU, instruction/data cache, main memory, and buses) and explore the interdependencies of different system parameters like cache sizes, associativity, main memory capacity, bus encoding schemes, and bus widths. Lastly, we consider, besides power, performance and area as well.

IV. Simulation-Based Analysis

In this section, we present a simulation-based approach to explore parameter optimization in SOC architectures. We consider a whole system consisting of a CPU, caches, a main memory, and interfaces between those cores, and we demonstrate the high impact of an adequate adaptation between core parameters and interface parameters in terms of power consumption. We will show that cache and bus configurations have a significant impact in this respect. In addition, we make the important observation that optimizing for performance does not imply that energy consumption is optimized as well. This is especially true in deep submicrometer technology.

A. Experimental Setup

For our experiments, we used the flow shown in Fig. 2, which allows us to estimate power consumption of the subsystem shown in Fig. 1, and estimate its performance for a given set of bus and cache parameters. After selecting a set of parameter values, the source code of the application is simulated to obtain a cache trace. This cache trace is then fed into a cache simulator, that is, a cache analyzer that outputs cache performance

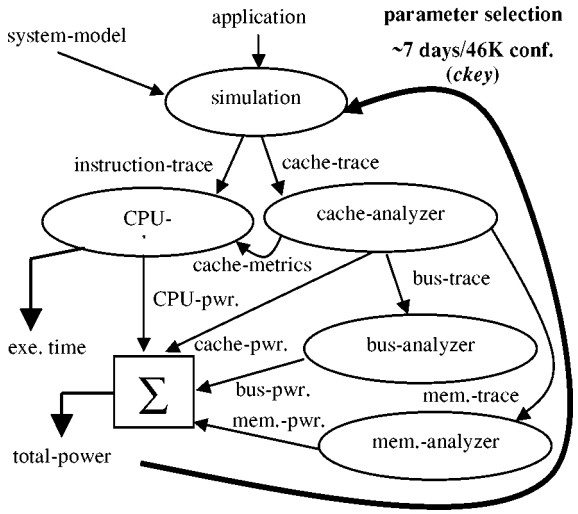

Fig. 2. Simulation-based performance analysis.

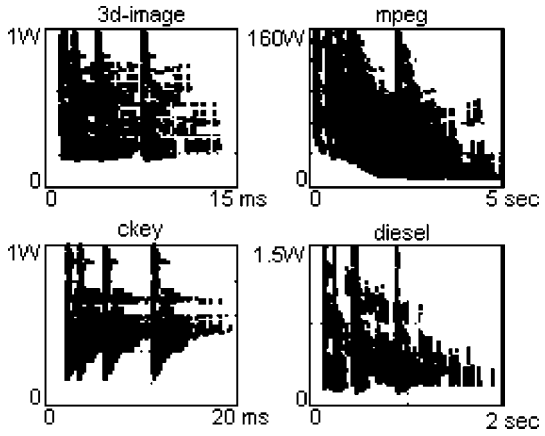

Fig. 3. Power versus execution time for older technology.

metrics, cache power consumption, and bus traces. The cache performance metrics, combined with the instruction trace generated by the simulator, is used in the CPU analyzer tool to compute CPU power consumption and system execution time. The bus trace is fed into the bus analyzer, where bus power consumption is computed and a memory trace is generated. Likewise, the memory trace is fed into the memory analyzer, where its power consumption is computed. Finally, the power consumption of all components is summed to obtain the total system power consumption. This process is iterated for each new set of parameter values.

To obtain CPU power, we have applied an instruction-based CPU power model. Here, each CPU instruction is characterized with respect to power. During a simulation, each executed instruction contributes to the total power consumption. In addition, cache misses and pipeline stalls are also accounted for, and their power consumption is added to the total CPU power. This approach is outlined in [14]. In [14], a cycle-accurate power simulation tool for an embedded system, using a strong ARM architecture as CPU, is introduced. The reported results are accurate within 5% compared to measurements conducted on a hardware board.

To obtain bus power, we have applied the standard power model $P = 0.5{*}V^2{*}C{*}f$, where f is the toggle frequency on the bus wires and C is the average capacitance of each wire. We have assumed that the routing area on the chip is constant. Thus, if less wires are routed, they will be spaced farther away, reducing the average wire capacitance. For this reason, we have modeled the capacitance of a bus with fewer wires with a lower capacitance.

For caches and memory, we used analytical power models. That means that those models resemble a parameterized structure of the underlying components. For example, a cache is decomposed into a decoding part, blocks, output, etc. A block is decomposed into lines, rows, and cells, and each of these basic components is associated with some relative capacitance. Model parameterization allows us to match our models accordingly when cache size, line size, and associativity are varied. The cache and memory models are driven by instruction traces and thus accumulate and finally output the energy consumption throughout the run of an application. Further details are found in [5].

Cache and memory area is obtained by using corresponding models as described in [20]. Area used by the bus-invert encoding is obtained by applying models described in [4]. Area used by the bus routing and CPU were excluded since these were not dependent on our parameter configurations.

For our experiments, we deployed four, mostly data-dominated, applications: an algorithm for computing three-dimensional (3-D) vectors of a motion picture *3d-image*, an MPEG-II encoder *mpeg*, a diesel engine control algorithm *diesel*, and a complex chroma-key algorithm *ckey*. (We in fact did a fifth example of an animation algorithm but omit results for presentation briefness since they matched the others.) The applications ranged in size from about 5 to 230 kB of C code.

B. Experiment Overview

For each of the four applications, we ran the cache simulation tool for all different cache configurations and obtained the cache, main memory, and CPU power consumption values as well as the CPU-to-cache and cache-to-main-memory bus traffic traces. We then input the bus traffic traces to a bus simulator [4] for all possible bus configurations. We thus obtained total power (cache plus main memory plus CPU plus bus) consumption for 46 656 possible configurations for each example.[1] Generating these power data for each of the four examples required roughly one week of computation time. We did this procedure for two distinct technologies: an older technology (0.8 μm), shown in Fig. 3, and a newer technology (0.18 μm), shown in Fig. 4. The wire-to-gate capacitance ratios for old and new technologies were three and 100, respectively, based on [20] and also supported by data in [11] showing rapidly increasing ratios. Each plot represents several tens of thousands of configurations, each configuration representing a point in the plot.[2] In addition,

[1]We obtain 46 656 as follows. For each of the two caches, we have nine possible cache sizes and three possible associativities/line sizes. For each of the two buses, we have four possible widths and two data encoding. As a result, we obtain $(9{*}3){*}(9{*}3){*}(4{*}2){*}(4{*}2) = 46\,656$.

[2]Note that due to the scale/resolution, many different design points appear as only one point in the plots.

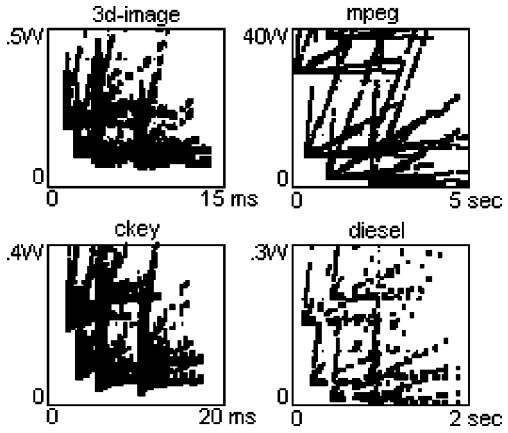

Fig. 4. Power versus execution for new technology.

many inferior data points have been cropped to further improve the presentation.

C. Power and Performance Tradeoff in Newer Technology

Fig. 3 provides plots of execution time (X-axis) and total power (Y-axis) for each example in the old technology. Total power includes CPU, cache, memory, and buses. Note that in three of the four examples, the configuration with the lowest execution time corresponded with the configuration having the lowest power consumption. Only in one example was there a real tradeoff between execution time and power.

Fig. 4 provides plots in the newer technology. Upon examination, we find that all four examples exhibit a tradeoff between execution time and power. The reason for this behavior is that the bus power is a significant component of total power in newer technology, and bus power is inversely related to execution time. In particular, fewer wires implies less wire capacitance and hence less power, but also implies more bus transfers and hence a higher execution time.

Each plot displays at least four distinct "regions." In the older technology, each region appears as a spike. In the newer technology, each region appears in an L-shape. Upon investigation of the data, we found that each region corresponds to a particular CPU-to-cache bus size of 32, 16, 8, or 4 (i.e., the four sizes we evaluated). The leftmost region corresponds to a CPU-to-cache bus of 32 bits (giving the smallest execution time), and the rightmost corresponds to 4 bits. The bottom-most point of each region corresponds to the optimal cache/bus configuration for that region based only on power and execution time (not size). The many points above and often to the right of the optimal correspond to different cache and cache-to-main-memory bus configurations. Within a region, there is no tradeoff between power and execution time—the lowest execution time configuration is also the lowest power configuration in any given region. Thus, we observe that the CPU-to-cache bus size is the major component in determining the system's power and execution time metrics. Moreover, for a given CPU-to-cache bus size, one can expect a local tradeoff among power and execution time. We conclude from these data that future automated search heuristics should begin by selecting a CPU-to-cache bus size that puts one in the appropriate region based on power and execution time constraints and/or cost function for a given application. Then, the automated search heuristics seeks the near-optimal power and execution time points in that region.

As an additional note, some plots show more than four regions, with a region X appearing directly above another region Y (actually all plots originally showed four pairs of regions before being cropped). These X and Y regions differ in that Y uses bus-invert for the CPU-to-cache bus while X does not. We see that bus-invert is crucial for low power in newer technology.

D. Size and Differing Cache/Bus Configurations

At the bottom of each region in the new technology, we see a very dense set of points. These points correspond to the optimal or near-optimal power and performance configurations for that region, each point representing a different cache configuration and cache-to-main-memory bus configuration. We obviously want to choose the configuration with the smallest size that is near the optimal within some tolerance.

We therefore examined all configurations within a couple percent of the optimal power, energy, and execution time of each region to find the minimum size configuration in this subregion. One might expect that the best cache configuration in one region would be the best in the other regions, for a single example. In some examples, this was indeed the case. But in other examples, the best configuration was different in different regions.

It must be noted that we consider both power and energy as important metrics in the design of a system. A system's power consumption dictates the design of heat-sink, power supply, and dc-to-dc converter (if needed) used in that system. Moreover, for systems that are designed for continuous operation, a configuration yielding lower power consumption that meets the system's timing constraints is preferred. Energy, on the other hand, is important for battery-driven systems that perform short tasks.

Table I illustrates the above-mentioned difference in configurations in different regions for two examples. The first three data rows correspond to the best configuration in the 32, 16-, and 8-bit CPU-to-cache bus regions for the *mpeg* example, and the second three rows for the *diesel* example. Notice that the best configurations for the first example involve variations in cache sizes, associativity, and line, and in one case bus invert is not used on the cache-to-memory bus. In the second example, associativity and line vary, as do the cache-to-memory bus size and the use of bus invert.

Slight changes in these parameters result in significant penalties. For example, in the *diesel* example, after examining the complete set of data (not shown), we observed that changing the cache line in the fifth row of the table from eight to 16 (to match that of the last row) resulted in a performance penalty of 28%. This is in part due to the fact that the diesel application had little spatial locality. The key conclusion from the above discussion is that there is no one best cache configuration independent of CPU-to-cache and cache-to-memory bus configuration, and vice-versa—the two must be sought simultaneously.

E. Tradeoff Between Design Constraints: Old Versus New Technology

The following experiments have been conducted to demonstrate the different behavior of the old and the new technology in terms of power consumption and area. In Fig. 5, the upper two

TABLE I
THE BEST CACHE CONFIGURATION DIFFERS FOR DIFFERENT CPU-CACHE BUS SIZE REGIONS

CPU-Cache Bus		Cache-Memory Bus		I$			D$			Exe. Time (sec)	Power (watt)	Energy	~Area (gate)
Width	Code	Width	Code	Size	Line	Assoc.	Size	Line	Assoc.				
MPEG													
32	Invert	32	Binary	16K	16	4	16K	16	4	.086	43.6	3.75 J	200K
16	Invert	32	Invert	16K	8	8	32K	8	8	.389	11.4	4.43 J	205K
8	Invert	32	Invert	32K	16	8	16K	8	8	.995	3.40	3.38 J	304K
Diesel													
32	Invert	32	Invert	1K	16	4	.5K	8	4	.002	.190	.380 mJ	16K
16	Invert	32	Invert	1K	16	4	.5K	8	4	.003	.070	.210 mJ	17K
8	Invert	4	Binary	1K	32	2	.5K	16	2	.005	.020	.100 mJ	18K

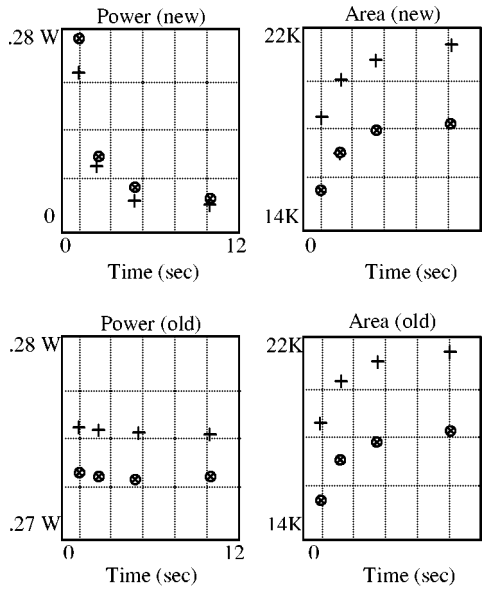

Fig. 5. Ckey's power, performance, and area tradeoffs using two specific configurations for new and old technologies.

plots show the power/performance tradeoffs (left) and area/performance tradeoffs (right) for the newer technology. The lower two plots show the same for the older technology. The configuration that yielded the "+" points has a larger data cache than the configuration yielding the "*" points. The points in each plot, from left to right, are obtained by varying the CPU-to-cache bus size from 32 down to 16, 8, and 4.

As we can see, the graphs show a strong power/performance tradeoff in dependency of the CPU-to-cache bus. Even if we choose half the cache size ("*"), the qualitative behavior is the same due to the CPU-to-cache bus dominance. For the older technology, however, a significant power/performance tradeoff with respect to bus sizes cannot be observed, i.e., all corresponding points are almost located on a horizontal line. This is because of the already mentioned lower wire/transistor capacitance relationship for older technology.

In terms of area/performance tradeoff, we again have a significant tradeoff for the new technology. But since area is the same (in terms of transistor counts), in this case the old technology shows the same tradeoff (in terms of the shape of the graph; the absolute numbers of the area are different due to different feature sizes, of course). Obviously, the question of whether or not there is a tradeoff depends on the technology as well as on the constraint that is of concern. These are key observations when performing design space explorations. Area and power obviously behave differently, either showing a tradeoff in conjunction with performance or not. A branch-and-bound technique, for example, could use this knowledge for a fast and efficient search of the design space.

F. Simultaneous Optimization of Buses and Caches

The implication of the above data is that cache and bus cannot be optimized independently; they must be optimized in some combined manner. Any approach that tries to separate their optimization may produce inferior results, especially when newer technology is deployed. For example, consider a straightforward heuristic that first optimizes cache without considering bus (assuming standard 32-bit buses with negligible capacitance), which essentially represents earlier cache power optimization work done for older technology, and that then optimizes the bus. While the cache configuration may represent the best power, execution time, and size in some regions, it may represent a rather inferior configuration in other regions. We applied this heuristic to the same two examples above. The heuristic achieves the best power and performance within each region but does very poorly in terms of size compared to the best sizes in Table I. For the first example, the sizes obtained for the three regions were 301 272, 304 472, and 306 072, representing size penalties of nearly 50% in the first two regions. The sizes for the second example were 26 272, 27 872, and 26 872, representing size penalties of 70%, 63%, and 52%, respectively. Thus, we see the need for new heuristics that simultaneously optimize cache and bus.

G. Summary of Simulation-Based Analysis

We can summarize our observations thus far as follows. First, in older technology, there is hardly a tradeoff between power and execution times, i.e., small execution times implies

low power consumption and a large execution time implies high power consumption. Second, newer technology does feature a real tradeoff, since bus power consumption becomes more significant due to a higher wire/gate capacitance ratio for smaller feature sizes. Third, the dominating source for power consumption in newer technology is the CPU-to-cache bus. The selection of this bus' size, therefore, is the major factor in making the tradeoff between a system's power and performance. Fourthly, for a given CPU-to-cache bus configuration, there is an optimal configuration of remaining cache/bus parameters that minimizes both power and performance. However, there is no optimum set of cache parameters across different CPU-to-cache bus configurations, and therefore the bus parameters of the CPU-to-cache bus parameters must be adapted to one another. Lastly, regarding size, for a given CPU-to-cache bus configuration, there is a near-optimal configuration of remaining cache/bus parameters that minimizes power, performance, and size. Again, this CPU-to-cache configuration differs for different CPU-to-cache bus configurations, and minor changes to the configuration can yield large penalties in performance or size.

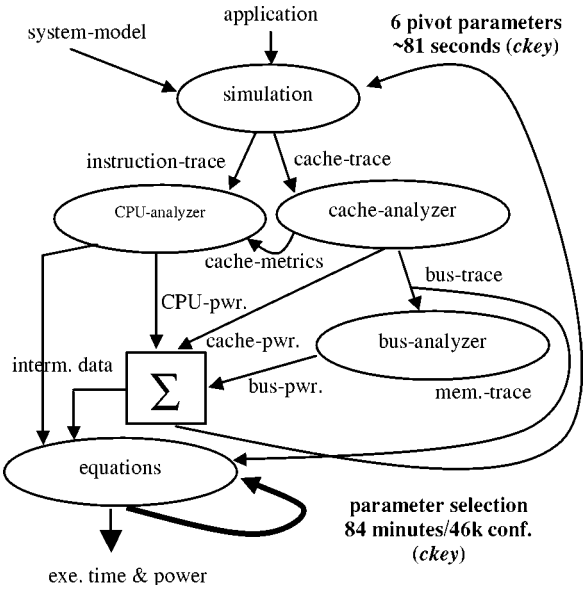

Fig. 6. Equation-based performance analysis.

V. Analytical Evaluation

Realizing that large tradeoffs are possible by configuring a system's parameters, but that system simulations are slow, we have devised a technique for fast evaluation of power consumed by the cache and bus subsystems of our target architecture. The technique uses a two-step approach of first collecting intermediate data about an application using a small number of simulations, and then using analytical equations to rapidly predict the performance and power consumption for each of thousands of possible configurations of system parameters.

Noticing that large tradeoffs are possible by configuring a cache for an application, but that cache simulations are slow, many researchers have focused on speeding up cache simulations. Kirovski *et al.* [9] reduces the *number* of trace-driven cache simulations necessary for exploring different cache configurations by establishing bounds and hence pruning numerous inferior configurations without having to simulate them. Wu and Wolf [22] order the search of different cache configurations such that, after each cache simulation, they can reduce the *size* of a given input trace by removing redundant information ("trace stripping"), thus speeding up subsequent simulations of other configurations. Our work differs in that we couple cache parameters with bus parameters (and possibly other parameters in the future), resulting in an enormous design space and thus seemingly excluding any approach based on repeated simulation. While one-pass cache-simulation [21] is a common technique, in which numerous cache configurations are evaluated simultaneously during one simulation, incorporating the myriad of other parameters that we wish to consider [bus, direct memory access (DMA), peripheral cores, etc.] into such an approach would likely become prohibitively complex.

Givargis and Vahid [4] have developed a set of formulas for rapidly estimating bit-switching activities on a bus with a given size and encoding scheme. They have also contributed formulas to estimate bit-switching activities used by the encoding/decoding logic. These formulas, combined with the capacitance estimation formulas by Chern *et al.* [1], can be used to perform a system-level exploration of bus size and encoding schemes for low power designs.

Our work is an improvement on previous work in that the long cache simulations can be replaced by the fast models in this paper to reduce the time to evaluate all cache/bus configurations for a given application from days/weeks to seconds/minutes.

A. Approach Overview

In our approach, we use a two-step technique, as shown in Fig. 6. The first step, *characterizing simulation*, involves simulating the application with typical input vectors once or a small number of times that is just enough to provide enough intermediate data to characterize the application for the second step. The second step, *parameter exploration*, uses heuristics to traverse the design space of possible parameter configurations, coupled with fast estimation equations that use the intermediate data to provide power, performance, and size values for a given configuration. These equations evaluate in constant time, so can deal with huge numbers of possible configurations.

We have chosen to focus initially on developing parameter optimization for a system's cache and bus subsystems because these typically consume a significant percentage of system power (we plan to soon extend our approach to also consider DMA). We have already developed an approach for buses [4] involving definition of the intermediate data (bus traffic); estimation equations for power, performance, and size as a function of bus parameters (size and encoding) and traffic; and an exhaustive search heuristic. We now describe the intermediate data and estimation equations necessary for cache parameter optimization, followed by a description of a method for coupling the cache methods with that previously developed for buses.

B. Cache Performance and Power

In this section, we discuss the technique that we have employed for rapidly evaluating cache metrics. We define the problem as follows. Given a trace of memory references (referred to hereafter as a *trace-file*), we are to compute the number of cache misses, denoted N, for all different caches. Two caches are different if they differ in their total cache size, line size (block size), or degree of associativity. We limit the values for each of these three parameters to a finite range of powers of two, say

$$S = \{\text{Min}, \ldots, 1\text{K}, 2\text{K}, 4\text{K}, 8\text{K}, 16\text{K}, \ldots, \text{Max}\}$$
$$L = \{\text{Min}, \ldots, 4, 8, 16, \ldots, \text{Max}\}$$
$$A = \{\text{Min}, \ldots, 4, 8, 16, \ldots, \text{Max}\}.$$

Note that for practical purposes, we only consider values that are powers of two for each of these parameters. Given a trace-file, we must define a function $f\colon (S \times L \times A) \rightarrow N$ to compute the number of cache misses for any cache configuration. We assume that with the aid of a cache simulator, we are able to compute the above function, for any value from the sets S, L, and A, in linear time with respect to the size of the trace-file.

Intuitively, our approach works as follows. We know that at low cache sizes, higher line size and associativity have a greater positive effect than they do at high cache sizes. For example, doubling the line size when cache size is 512 B may reduce cache miss rate by 30%; however, when the cache size is 8 K, it may not reduce the miss rate at all. Thus, we are interested in finding these improvement ratios at both low and high cache sizes, so that, by line fitting, the improvement ratio for any cache size can be estimated. This assumes a smooth design space between these points.

Our approach consists of three steps. First, we simulate the trace-file for some selected S, L, and A values and obtain the corresponding cache misses. Then we calculate a linear equation, using the least square approximation method. Last, we use our linear equations to compute N for all cache parameters. We first simulate the following points in our domain space:

$$f(S_{\min}, L_{\min}, A_{\min}) = N_1$$
$$f(S_{\max}, L_{\min}, A_{\min}) = N_2$$
$$f(S_{\min}, L_{\max}, A_{\min}) = N_3$$
$$f(S_{\min}, L_{\min}, A_{\max}) = N_4$$
$$f(S_{\max}, L_{\max}, A_{\min}) = N_5$$
$$f(S_{\max}, L_{\min}, A_{\max}) = N_6.$$

Then we compute the following ratios:

$$R_1 = N_1/N_3 \qquad R_2 = N_1/N_4 \qquad R_3 = N_2/N_5$$
$$R_4 = N_2/N_6.$$

Here, R_1/R_2 denotes the improvement we obtain by using maximum line size and associativity when cache size is at its minimum. Likewise, R_3/R_4 denote the positive improvement we obtain by using maximum line size and associativity when the cache size is at its maximum. Given these ratios, we estimate N for a given cache size, line size, and associativity as follows:

$$s = (S_i - S_{\min})/S_{\max}$$
$$l = (L_j - L_{\min})/L_{\max}$$
$$a = (A_k - A_{\min})/A_{\max}$$
$$t_1 = s(N_2 - N_1) + N_1$$
$$t_2 = l(R_3 - R_1) + R_1$$
$$t_3 = a(R_4 - R_2) + R_2$$
$$f(S_i, L_j, A_k) = t_1(1 - t_2 - t_3).$$

The first three equations, s, l, and a, normalize our parameters to be within a unit range. The next equation t_1 estimates cache misses using lowest line size and associativity by computing a linear line through the points N_1 and N_2. If more simulation data are available, the least square approximation is used to compute t_1. The next two equations t_2 and t_3 estimate the expected improvement gained from higher line size or associativity. The last equation combines the previous equations to estimate cache miss rate.

C. Combined Cache and Bus

In this section, we describe how to extend the cache data into bus data for simultaneous cache/bus design space exploration. The technique described in the previous section allows us to rapidly estimate the number of cache misses N for a given cache parameter setting. This number N is a measure of cache to main-memory bus traffic. Likewise, the total number of cache accesses, i.e., the size of the trace-file, is a measure of CPU to cache bus traffic. Given this traffic, and assuming data of random nature, we can use equations [4] to compute the bit-switching activity on the bus and use it, along with wire capacitance models, to compute power consumption of our system. Based on our own experiments and others referenced in [4], the random data assumption holds for data buses. (We have only considered analytical switching models for data buses and have relied on simulation for address buses.) In this paper, we consider varying the number of data bus wires, e.g., 16 or 32 bits, and data encoding, e.g., binary or bus-invert.

For our bus model, we assume that there are m, *n-bit* items transmitted per unit time on a bus of width k using binary encoding. Here, m denotes the traffic on the bus and is obtained by estimating cache misses, as described above. The following equation gives power consumption for the data bus:

$$P_{\text{bus}} = (C_{\text{bus}}) \left(\left\lceil \frac{n}{k} \right\rceil \text{transfer/item} \right) (k \text{ bit/transfer}) \cdot (m \text{ item/s}) \left(\tfrac{1}{2} \text{ transition/bit} \right)$$
$$= \frac{1}{2} (C_{\text{bus}}) \left\lceil \frac{n}{k} \right\rceil (k)(m) \text{ transition/s}.$$

In this equation, bus capacitance is calculated using models developed by Chern *et al.* [1]. Our equation is expanded to take into account bus-invert encoding. This method uses an extra

TABLE II
DESIGN SPACE CONFIGURATION PARAMETERS

Config.	Bus1	Bus2	I$ Size	D$ Size	Line	Assoc.
0	8/0	4/1	128	512	8	2
1	32/1	32/1	512	16K	8	2
2	32/0	16/0	8K	2K	8	2
3	4/0	8/1	32K	16K	8	2
4	16/0	32/1	512	4K	16	2
5	8/0	8/1	8K	512	16	2
6	8/1	16/0	4K	16K	32	2
7	8/1	4/0	1K	8K	8	4
8	16/0	16/0	1K	256	16	4
9	32/1	8/1	1K	1K	32	4

control line and extra circuit logic to compute the Hamming distance (bit transitions) between two consecutive data items. If the Hamming distance is greater than one-half the bus width, then the control line is asserted and the inverted data are sent over the bus [12]

$$PI_{\text{bus}} = (C_{\text{bus}})\left(\left\lceil \frac{n}{k} \right\rceil\right) \cdot \left[\frac{\binom{k+1}{1}}{2^k} \cdot 1 + \frac{\binom{k+1}{2}}{2^k} \cdot 2 + \cdots + \frac{\binom{k+1}{\left\lceil \frac{k}{2} \right\rceil}}{2^k} \cdot \left\lceil \frac{k}{2} \right\rceil \right] (m) \text{ transition/s.}$$

Given the traffic m on a bus, power can be quickly estimated using analytical models as described above. Likewise, similar analytical models can be applied to compute cache and memory power (and performance). These have been extensively modeled by [2].

D. Experimental Results

In order to verify our approach, we performed the following experiments. We explored power and performance for mapping the *diesel* and *ckey* to our system architecture and exploring three parameterized parts: cache, CPU-cache (Processor Local) bus, and cache-memory (System) bus. The cache parameters and their possible values were: cache size of 128, 256, 512, 1 K, 2 K, 4 K, 8 K, or 32 K; cache line of 8, 16, or 32; and associativity of 2, 4, or 8. The parameters for each bus were: data width of 4, 8, 16, and 32; and bus invert encoding either enabled or disabled.

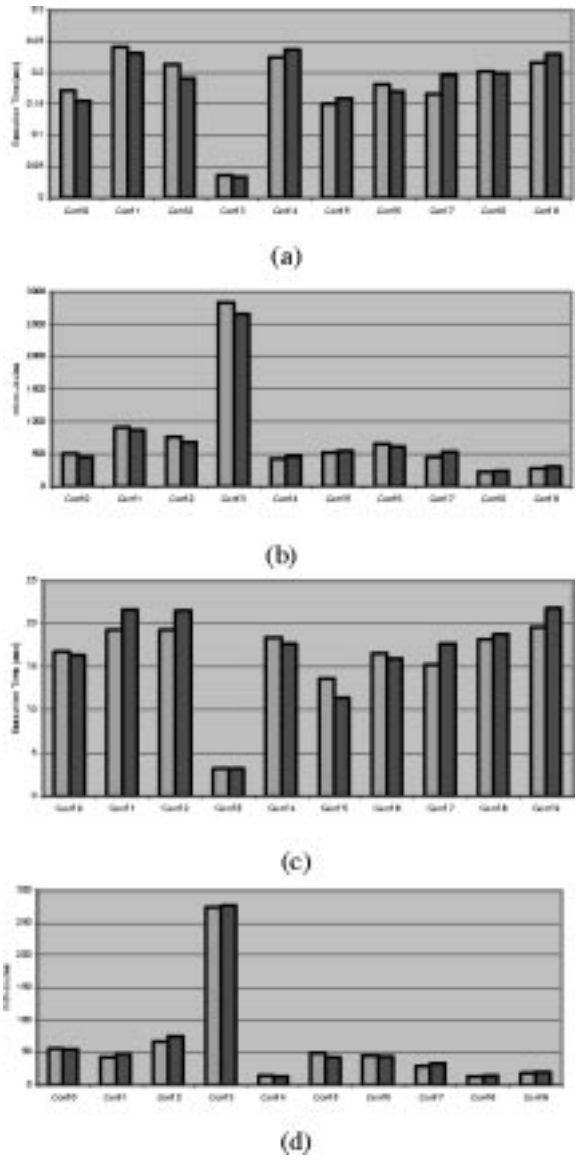

Fig. 7. Equation-based results: (a) diesel application's performance, (b) diesel application's energy, (c) ckey application's performance, (d) ckey application's energy. Light gray is actual measurement and dark gray denotes estimated measurements.

We compared these results with those obtained using simulation (described in the previous sections). For the fast estimation approach, we ran the system simulator only six times for the reference cache configurations described earlier. We then fed the power, performance, and hit-rate information from these simulations into our cache power/performance models, and next evaluated the models for all cache configurations. For each such configuration, we also obtained bus traffic data and these data into our bus model for all bus configurations. Obtaining these values for all possible cache/bus configurations required only 84 min, instead of seven days, a speedup of 120 times.

While we obtained data for all of the 45 568 possible cache/bus configurations, we present data for just a small subset of ten configurations in Table II. These configurations have been selected to reflect worst, average, and best case estimates. Fig. 7 provides performance and power respectively for *diesel* and *ckey* applications. The light-gray bars are actual

measurements and the dark-gray bars estimated measurements. While hundreds of times faster, our cache estimation approach resulted in an average error of only 2%, with the worst case being 18%, over the entire solution space of thousands of cache/bus configurations (and not just the ten configurations presented here). It should be noted that the CPU power consumption was about 44% and 65% (for *diesel* and *ckey* respectively) of total power consumption.

Perhaps even more important than the accuracy reported above is the *relative accuracy*, or fidelity, of the estimates. We see from the charts that our fast approach orders the various configurations the same as the simulation approach—thus, we have the ability to still pick the best parameter configuration, which is the most important aspect of the approach.

VI. Conclusion

We have shown that by designing parameterized caches and buses, and varying those parameters for a given application, we can obtain large tradeoffs in power and performance. We have also described analytical techniques that enable rapid exploration of the complete cache/bus configuration space, enabling selection of the optimal configuration for a given application. The results and analytical techniques can be applied not only to postfabrication parameters but also to prefabrication parameters that are configured and then used to synthesize an SOC architecture customized to a particular application and requirements. Future work includes further development of adaptive caches and buses, integration of such items with configuration-aware compilers, and extension of the techniques to parameterized microprocessors and peripheral cores. In addition, we would like to further investigate the impact of each of the parameters individually.

References

[1] J. H. Chern *et al.*, "Multilevel metal capacitance models for CAD design synthesis systems," *IEEE Electron Device Lett.*, vol. 13, Jan. 1992.
[2] B. P. Dave, G. Lakshminarayana, and N. K. Jha, "COSYN: Hardware–software co-synthesis of embedded systems," in *Proc. DAC'97*, 1997, pp. 703–708.
[3] W. Fornaciari, D. Sciuto, and C. Silvano, "Power estimation for architectural exploration of HW/SW communication on system-level buses," in *Proc. Int. Workshop Hardware/Software Codesign*, 1999, pp. 152–156.
[4] T. D. Givargis and F. Vahid, "Interface exploration for reduced power in core-based systems," in *Proc. Int. Symp. System Synthesis*, 1998, pp. 117–122.
[5] Y. Li and J. Henkel, "A framework for estimating and minimizing energy dissipation of embedded HW/SW systems," in *Proc. IEEE 35th Design Automation Conf. (DAC98)*, 1998, pp. 188–193.
[6] I. Hong, D. Kirovski, and M. Potkonjak, "Potential-driven statistical ordering of transformations," in *Proc. DAC'97*, 1997, pp. 347–352.
[7] C. T. Hsieh, M. Pedram, G. Mehta, and F. Rastgar, "Profile-driven program synthesis for evaluation of system power dissipation," in *Proc. IEEE 34th Design Automation Conf. (DAC97)*, 1997, pp. 576–581.
[8] "International Technology Roadmap for Semiconductors (ITRS),", 1999.
[9] D. Kirovski, C. Lee, M. Potkonjak, and W. Mangione-Smith, "Synthesis of power efficient systems-on-silicon,", ASP-DAC 1998.
[10] C. Kozyrakis and D. Patterson, "A new direction for computer architecture research," *IEEE Computer*, pp. 24–32, Nov. 1998.
[11] "Semiconductor Industry Association Roadmap,", 1997.
[12] M. R. Stan and W. P. Burleson, "Bus-invert coding for low power I/O," *IEEE Trans. VLSI*, vol. 3, Mar. 1995.
[13] ——, "Limited-weight codes for low power I/O," in *Proc. Int. Workshop Low Power Design*, Apr. 1994.
[14] T. Simunic, L. Benini, and G. De Micheli, "Cycle-accurate evaluation of energy consumption in embedded systems," in *Proc. Design Automation Conf.*, 1999, pp. 876–872.
[15] F. Vahid and T. D. Givargis, "The case for a configure-and-execute paradigm," in *Proc. Int. Workshop Hardware/Software Codesign*, 1999.
[16] J. van Meerbergen, A. Timmer, J. Leijten, F. Harmsze, and M. Strik, "Experiences with system level design for consumer ICs," in *Proc. VLSI'98*, pp. 17–22.
[17] V. Tiwari, "Logic and system design for low power consumption," Ph.D. dissertation, Princeton University, Nov. 1996.
[18] Velocity product information. VLSI Technology Inc.. [Online]. Available: http://www.vlsi.com/velocity
[19] Virtual Socket Interface Association. (1997) Architecture document. [Online]. Available: http://www.vsi.org
[20] N. H. E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*. Reading, MA: Addison Wesley, 1998.
[21] S. J. E. Wilton and N. P. Jouppi, "CACTI: An enhanced cache access and cycle time model," *IEEE J. Solid-State Circuits*, vol. 31, no. 5, pp. 677–688, 1996.
[22] Z. Wu and W. Wolf, "Iterative cache simulation of embedded CPU's with trace stripping," in *Proc. Int. Workshop Hardware/Software Codesign*, 1999, pp. 95–99.

Tony D. Givargis, photograph and biography not available at the time of publication.

Frank Vahid, photograph and biography not available at the time of publication.

Jörg Henkel (M'95–SM'01) received the Diploma and Ph.D. ("*summa cum laude*") degrees in electrical engineering from the Technical University of Braunschweig, Germany, in 1991 and 1996, respectively.

Since 1997, he has worked as a Research Staff Member at the Computer and Communication Research Laboratories, NEC, Princeton, NJ. In spring 2000, he was a Visiting Professor at the University of Notre Dame, IN. His interests include embedded system design, methodologies, architectures, and hardware/software codesign.

Dr. Henkel is the General Cochair of the 10th IEEE/ACM International Symposium on Hardware/Software Codesign CODES'02.