System-Level Exploration for Pareto-Optimal Configurations in Parameterized System-on-a-Chip (December 2002)

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

***Abstract*—In this work, we provide a technique for efficiently exploring the power/performance design space of a parameterized system-on-chip (SOC) architecture to find all Pareto-optimal configurations. These Pareto-optimal configurations will represent the range of power and performance tradeoffs that are obtainable by adjusting parameter values for a fixed application that is mapped on the SOC architecture. Our approach extensively prunes the potentially large configuration space by taking advantage of parameter dependencies. We have successfully applied our technique to explore Pareto-optimal configurations of our SOC architecture for a number of applications.**

***Index Terms*—Design space exploration, low-power design, Pareto-optimal configurations, platform-based design, system-on-a-chip (SOC) design.**

I. Introduction

THE GROWING demand for portable embedded computing devices is leading to new system-on-a-chip (SOC) architectures intended for embedded systems. 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 parameterized SOC architectures. Different applications often have very different power and performance requirements. Therefore, these parameterized SOC architectures must be optimally configured to meet varied power and performance requirements of a large class of applications.

A typical SOC architecture will have a processor core, one or more caches, on-chip bus hierarchy, on-chip memory, and a large number of peripheral cores that provide application specific functionality such as multimedia and communication processing. Each of these SOC cores is likely to be parameterized, enabling a designer to tune a core's settings for a specific application that is to be mapped on the SOC architecture. For example, the on-chip buses may be configured to use bus-invert [1] coding for low power, or the caches may be configured to use a greater or lesser degree of associativity for increased performance [2], [3]. An assignment of a value to each of these parameters will impact the overall performance and power consumption of the SOC architecture. However, such impacts are highly dependent on the application running on the SOC. Therefore, a designer must have a method for finding a feasible set of parameter values, referred to as a configuration of the SOC, that meets the specification requirements. We outline an exploration approach that efficiently searches the entire configuration space and outputs Pareto-optimal configurations providing the designer with only the interesting configurations that result in a tradeoff between power and performance.

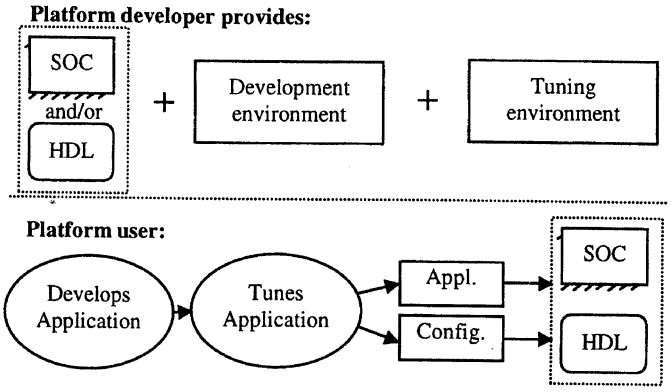


Fig. 1. SOC design flow.

Our exploration algorithm fits in the SOC design flow as follows. As depicted in Fig. 1, the SOC provider provides a parameterized architecture in hardware discription language (HDL) or configurable integrated circuit (IC) format along with all the traditional development tools such as compilers, debuggers, emulators, etc. In addition, the SOC provider provides a system-level model and a tuning environment. This tuning environment enables the SOC user to search the parameter space of the SOC and to find a configuration that meets power and performance requirements of the target application. This tuning application is the focus of this work.

The remainder of this paper is organized as follows. In Section II, we describe related work. In Section III, we state the parameterized SOC exploration problem and outline our approach for solving it. In Section IV, we give some experimental results. In Section V, we state our conclusions.

Manuscript received October 30, 2000; revised August 23, 2001. This work was supported in part by the National Science Foundation under CCR-9811164, CCR-9876006, NEC, and a Design Automation Conference Graduate Scholarship.

T. Givargis is with the Department of Information and Computer Science and member of the Center for Embedded Computer Systems, University of California, Irvine, CA 92697 USA (e-mail: givargis@ics.uci.edu).

F. Vahid is with the Department of Computer Science and Engineering, University of California, Riverside, CA 92521 USA, and also with the Center for Embedded Computer Systems, University of California at Irvine, CA 92697-3425 USA (e-mail: vahid@cs.ucr.edu).

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

Digital Object Identifier 10.1109/TVLSI.2002.807764

II. Previous Work

Much previous work has focused on power evaluation of SOC architectures at various levels of abstraction. Circuit-level approaches simulate the circuit at the transistor level while monitoring supply current [4], [5]. Logic-level or gate-level approaches simulate a gate-level design, and calculate power by considering switching activity of nodes in the design [6], [7], executing orders of magnitude faster than circuit-level approaches at the expense of some accuracy. Register-transfer level (RTL) power evaluation operates at an even higher-level of abstraction, modeling power consumption of more abstract circuit components, such as adders and multipliers etc. Simulation is performed at the RT-level and power is obtained by using these power models, also known as macro models. RTL power evaluation, in some publications such as [8], is shown to be accurate to within 5% of actual power consumption. Behavioral-level approaches seek to estimate power of a behavioral HDL description before a synthesized design is obtained. An abstract notion of physical capacitance and switching activity is used. Switching is estimated using entropy from circuit input to circuit output by quadratic or exponential degradation [9], [10].

Work has been done to evaluate power consumption of a particular type of core, like microprocessors. One approach, instruction-level power modeling is proposed by [11]. Given a program execution trace, energy is computed as the sum of the energy consumed by each instruction that is executed, circuit state energy consumed when a particular instruction is followed by another, and energy consumed by other effects such as stalls and cache misses. This approach is sped up in [12], by deriving a shorter program trace that results in equal power dissipation when compared to the original trace. In [13] a mathematical generic power model for 32-bit microprocessors is proposed. The approach classifies the instruction set into classes, like branches. The model has been applied to various 32-bit processors. Other researchers have focused on fast system-level models for cache, memory, and bus power consumption [14]–[16], consisting mostly of simulators coupled with equations that compute power consumption as a function of usage/traffic and core parameters. Further approaches aim at estimating the power consumption of whole embedded systems. In [17], 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. A trace-based approach deploying a mix of analytical models (for instruction cache, data cache and main memory) and instruction set simulators (ISS) is introduced in [18].

Other system-level approaches have been proposed for application-driven design of core-based systems [19]. Here, given a fixed application, heuristics are used to determine cache size and organization in order to minimize cache misses while also minimizing chip area. Likewise in [20], an analytical approach is provided for exploring the on-chip memory architecture given a fixed application.

A methodology, closely related to ours, named SPADE (system level performance analysis and design space exploration), is proposed in [21]. This work defines a general scheme for the design of programmable architectures, referred to as the Y-chart and shown in Fig. 2. Here, target applications are mapped onto the architecture and their performance is analyzed to obtain performance numbers. (The architecture, applications and performance numbers represent the dimensions along the Y shaped chart, hence the name Y-chart.) After analysis, the architecture or applications are tuned and the process is repeated until a desired system is obtained. In our work, we outline an approach to automate the exploration.

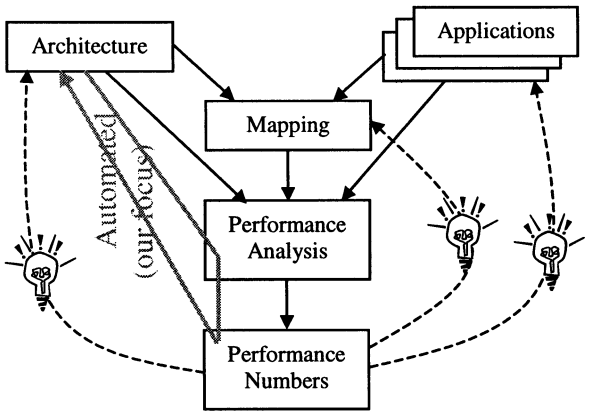


Fig. 2. Y-chart approach for the design of configurable architectures.

Previous work has focused on techniques that quickly and accurately simulate SOC architectures in order to obtain power and performance metrics. Our technique combines this work with an approach for efficiently exploring the configuration space of SOC architectures by pruning configurations that are guaranteed to be inferior to others already evaluated.

III. Approach Overview

We will next state the problem and outline our solution. Our solution will be given by first looking at an exhaustive method. Then we state the key observation that makes our approach more efficient, followed by our efficient solution.

A. Problem Formulation

We are given a SOC architecture composed of numerous interconnected parameterized computational, communication, and memory elements. Each of these parameters can be assigned a value from a finite set of values. A complete assignment of values to all the parameters is a configuration. We are also given a parameterized system-level model of the SOC that when executed can yield the power and performance of the SOC for a configuration. Such parameterized simulation models have been outlined in [17], [22], [23]. The problem is to efficiently compute, with the aid of a system-level model, the Pareto-optimal configurations, with respect to power and performance, for a fixed application executing on the SOC. In our problem, a configuration is Pareto-optimal if no other configuration has better power for a given performance.

The algorithms described in this paper can utilize any available power and performance measuring approach such as in-circuit emulation, gate-level simulation, RTL simulation or a system-level behavior approach. Hence, the exploration and

evaluation approaches are absolutely orthogonal. In this work a system-level model is used for evaluation [22], [23]. This system-level model can achieve simulation speeds that are four- to five-orders of magnitude faster than gate-level simulation, while maintaining performance/power estimation accuracy of 5% to 10% when compared to gate-level estimations. Here, for the CPU, cache and memory cores of the system an instruction-level power model is used that is based on [11]. For other cores, such as UART and CODECS, the core provider selects a set of appropriate instructions covering the possible actions of each core in the architecture. Then the provider performs gate-level power analysis to construct lookup tables for each instruction and creates a system-level core model that utilizes the lookup-tables for power evaluation through an executable specification. The core user connects the system-level peripheral and CPU core models, executes the system and thus obtains power and performance data. The simulation speed of this system is approximately 134 K processor instructions per second running on an 800 MHz Pentium PC.

B. An Exhaustive Solution

We start by outlining an exhaustive algorithm to solve the exploration problem. In this exhaustive algorithm, first power and performance are evaluated for all configurations. Then, configurations are sorted by nonincreasing execution time (i.e., higher performance). Then, in the sorted order, a walk through the space is performed while all configurations that result in power consumption above the minimum seen thus far are eliminated. The remaining configurations are Pareto-optimal. The algorithm is given below

```
Algorithm 1
list compute_Pareto_configurations(space
  s) {
  list all, pareto;
  float min_power = 1e100; /* infinity */
  for each configuration c in space s {
    simulate_SOC(c); all.push(c);
  }
  all.sort( /* key is execution time */ );
  while( !all.empty() ) {
    c = all.pop();
    if( c.power < min_power ) {
      min_power = c.power; pareto.push(c);
    }
  }
  return pareto;
}
```

The problem with this approach is that the configuration space is likely to be very large, making the approach impractical in many cases. The exhaustive approach is practical when applied to a small subset of the solution space consisting of one- or two-varying parameters while all others held constant. We have found that many parameters in an SOC platform have little interdependency among each other. Two parameters are interdependent if changing the value of one of them impacts the optimal parameter value of the other.

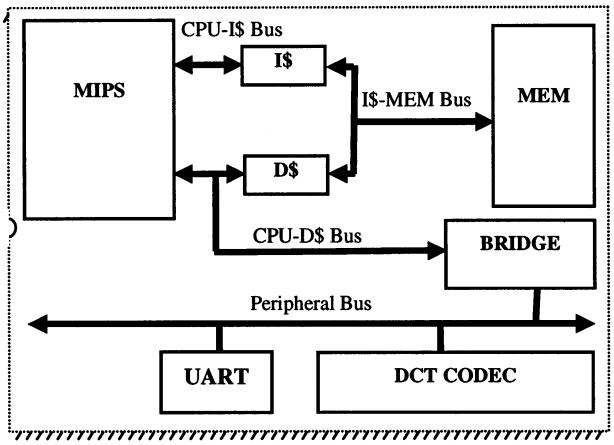
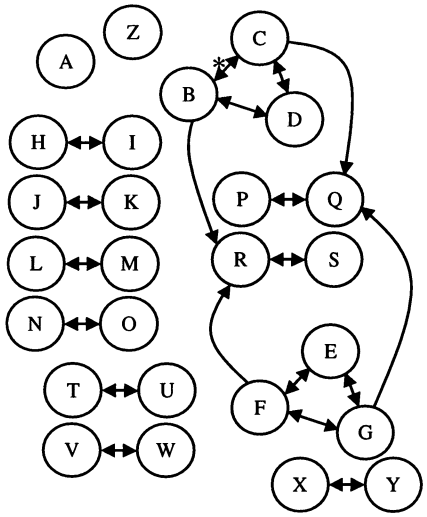


Fig. 3. Parameterized SOC platform.

C. Parameter Interdependency Model

We have used a directed graph model to capture the parameter interdependencies. Such a graph is constructed with its nodes representing parameters and edges representing interdependencies between parameters. Generally, a path from a node A to a node B indicates that the Pareto-optimal configurations of B should be calculated once the Pareto-optimal configurations of all the nodes from A to B, residing on the path, are calculated. During that calculation all other parameters not on the path are fixed to some arbitrary value. A path from a node A to a node B and back to A, which forms a cycle, indicates that the Pareto-optimal configurations of all the parameters on the cycle need to be calculated simultaneously. During that calculation all other parameters not on the path are fixed to some arbitrary value. The Pareto-optimal configurations of an isolated node is computed by setting all other parameters to some arbitrary value.

Fig. 3 shows the parameterized SOC architecture used in our experiments. The parameter description and interdependency graph of this architecture is depicted in Fig. 4. All interdependencies are manually determined. The cache size, associativity, and line parameters (B/C/D and E/F/G) are interdependent. The bus width and data encoding parameter pairs (H/I, J/K, L/M, N/O, P/Q, R/S, T/U, and V/W) are interdependent. The UARTs buffer parameters (X/Y) are interdependent. Since the cache configuration will affect the amount of data transferred to and from the main memory bus, the I/D$-memory-bus parameters are dependent on the cache parameters. To keep the graph from being cluttered, we have only shown a single edge from an arbitrary cache parameter to an arbitrary bus parameter (F to R for instance), since, by the transitive property, the bus parameters will be dependent on all the cache parameters. Finally, the MIPS voltage scale and DCT CODECs parameters are independent of the remaining parameters. Our parameterized SOC architecture is composed of a total of 26 parameters. The total design space is composed of 1014 configurations. The parameters of our SOC architecture are hardware parameters and assumed



Node	Core	Parameter		Node	Core	Parameter
A	MIPS	Voltage scale		L	CPU-D$-bus	Data bus width
B	I$	Total size		M		Data bus code
C		Line size		N		Addr bus width
D		Associativity		O		Addr bus code
E	D$	Total size		P	I/D$-Mem-bus	Data bus width
F		Line size		Q		Data bus code
G		Associativity		R		Addr bus width
H	CPU-I$-bus	Data bus width		S		Addr bus code
I		Data bus code		T	Peripheral-bus	Data bus width
J		Addr bus width		U		Data bus code
K		Addr bus code		V		Addr bus width
X	UART	Tx buffer size		W		Addr bus code
Y		Rx buffer size		Z	DCT CODEC	Pixel resolution

* Note that we have used double-arrowed lines in place of two single arrowed-lines for clarity.

Fig. 4. Target SOC interdependency graph and parameter descriptions.

```
Algorithm 2
list compute_Pareto_configurations_2(graph g) {
  list sub_graphs, pareto;
  sub_graphs = strongly_connected_components(g);
  // part 1
  for each sub-graph g in sub_graphs {
    pareto=compute_Pareto_configurations(g.space);
    eliminate configs. in g.space not in pareto;
  }
  // part 2
  while( !sub_graphs.size() != 1 ) {
    g1 = sub_graphs.pop_front();
    g2 = sub_graphs.pop_front();
    g = g1 union g2; sub_graphs.push_back(g);
    pareto=compute_Pareto_configurations(g.space);
    eliminate configs. in g.space not in pareto;
  }
  return pareto;
}
```

to be set prior to fabrication. However, the approach given in this paper is general and applicable to other types of parameters, such as post fabrication parameters, compiler assigned parameters, and software tunable parameters.

We assume that the designer of the SOC platform determines the interdependencies among the parameters. However, our current research is focused on developing automated approaches for computing such interdependencies. These automated approaches are characterized as either being conservative or nonconservative. The conservative approaches start assuming that all parameters are interdependent and then remove dependencies if its determined (through exhaustive simulation or sampling) that two parameters are clearly not interdependent. The nonconservative approaches start assuming that all parameters are independent and then introduce interdependencies if its determined that two parameters are likely dependent. In both cases, the characterization of parameter interdependencies is a one-time effort.

D. An Efficient Exploration Algorithm

Given an interdependency graph, our algorithm works as shown above in Algorithm 2.

The algorithm can be broken down into two phases. The first phase performs a local search for Pareto-optimal configurations. The second phase iteratively expands the local search to discover global Pareto-optimal configurations.

Part 1 of our algorithm performs clustering of interdependent nodes in the graph. This is the same problem as finding strongly connected components of a graph. This can be computed by performing two depth-first searches in linear time. In addition, if two clusters are connected (but not strongly) then they are topologically ordered. Here, each cluster represents a disjoint subspace of the overall configuration space. We use our exhaustive algorithm for calculating Pareto-optimal configurations for each of the clusters. Then, we restrict possible configurations of that cluster to the Pareto-optimal configurations only. This

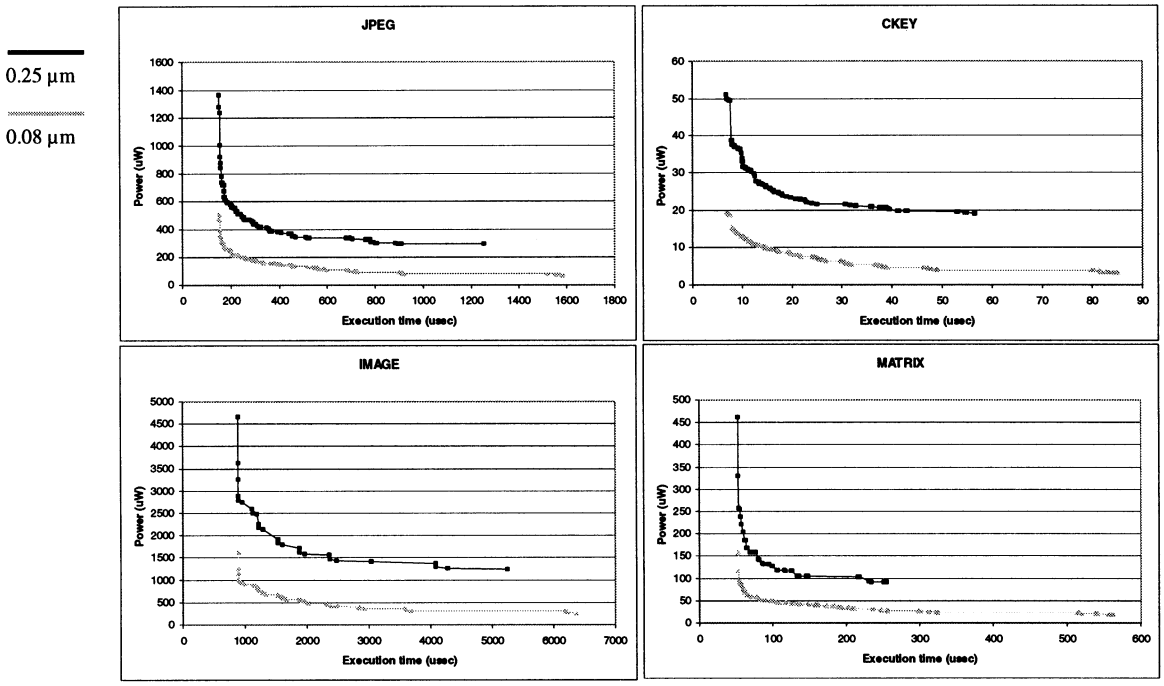


Fig. 5 Pareto-optimal configurations of *jpeg*, *ckey*, *image*, and *matrix* examples. Results shown for two technologies.

pruning is justified since if a configuration is not Pareto-optimal within a cluster, it cannot be part of a Pareto-optimal configuration for the entire configuration space. Conversely, if a configuration is Pareto-optimal within a cluster, it may or may not be Pareto-optimal given the entire configuration space, and thus must remain. Our exhaustive approach applied to clusters is usually feasible since these clusters represent only a small subspace of the total configuration space. Nevertheless, heuristics such as probabilistic exploration techniques or genetic algorithms can be used to search within a cluster when the exhaustive method is too slow.

Part 2 of our algorithm combines pairs of clusters into a single cluster and computes Pareto-optimal configurations within it. It does this by defining the configuration space of this new cluster to be the cross product of the Pareto-optimal configurations of the two merged clusters. This procedure is repeated until all the clusters have been merged and a single cluster remains. The Pareto-optimal configurations within this last cluster represent Pareto-optimal configurations of the entire configuration space. Part 2 of the algorithm combines clusters in no particular order. However, by combining cluster pairs that result in the smallest merged configuration space, the total number of configurations that are examined is sometimes minimized. However, such optimization does not change the time complexity of the algorithm.

The worse case time complexity of the algorithm is bounded by $O((K+\log(K)) \times 2^{N/K})$, where K denotes the number of initial strongly connected components (i.e., clusters) computed in part 1, and N denotes the number of parameters. Here, the $2^{N/K}$ factor[1] bounds the running time of the exhaustive computations of the Pareto-optimal points. The K in the first factor is a bound on the number of times that the first part of the algorithm iterates, while the $\log(K)$ is a bound on the number of times the second part of the algorithm iterates.[2] In the worst case, when $K = 1$ (all parameters are interdependent) the running time is exponential, namely 2^N. In the best case, when $K = N$ (all parameters are independent) the running time is linear, namely N. For most practical cases the running time will be closer to the best case since the factor $2^{N/K}$ will decrease very rapidly as K increases.

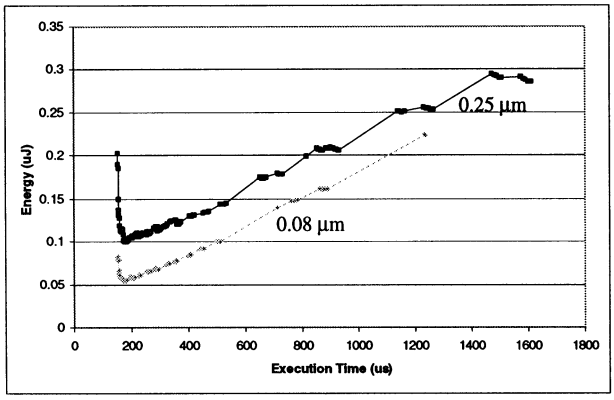


Fig. 6. Energy tradeoffs of the *jpeg* example.

IV. Experiments

As stated in the previous section, we have a parameterized simulation model of the SOC architecture shown in Fig. 4. We explored the configuration space for 4 application programs and

[1]For the purpose of time complexity analysis and presentation brevity, we assume each parameter can take on two values, however, in reality, parameters can be assigned larger number of values.

[2]The maximum number of nodes in a cluster is assumed to be N/K. This assumption holds in the worse and best case analysis. In any other case, the N/K ratio is the expected value but not the exact value.

TABLE I
SUMMARY OF RESULTS

Appl.	EXPL. TIME (.08/.25μm)	Configs. Visited (.08/.25μm)	Pareto configs. (.08/.25μm)	Exe-time tradeoff (.08/.25μm)	Power tradeoff (.08/.25μm)	Energy tradeoff (.08/.25μm)
jpeg	6.3/25 min	2672/10611	97/118	10/8.3 x	7.4/4.7 x	2.7/3.4 x
ckey	24/68 min	3522/10050	125/213	13/8.4 x	6.0/2.7 x	2.6/3.6 x
image	9.5/47 min	1927/9531	26/42	7.1/5.9 x	3.1/3.8 x	2.1/2.6 x
matrix	15/54 min	2535/9245	41/92	11/4.9 x	8.3/5.03 x	2.3/2.2 x

2 different technologies representing 8 different examples. Our applications are named jpeg, ckey, image, and matrix. The, jpeg application implements a JPEG compression algorithm using the on-chip DCT CODEC. The ckey application implements a complex chroma-key algorithm. The image application rotates an image by 90° and converts the image colors to grayscale. The matrix application performs a matrix invert operation on a large matrix.

For each of the four examples, we simulated both a version of the SOC that used power models for an older technology (0.25 μm) and a version that used power models for a newer technology (0.08 μm).

Among the eight runs, on the average, the time to explore and find Pareto- optimal configurations for any design was 34.5 min. On the average, our algorithm returned 93 Pareto-optimal configurations and simulated 6852 distinct configurations. Among the eight runs, the pruning ratio was 99.7%, meaning 997 out of 1000 configurations were pruned. Our results are summarized in Table I. The power/performance tradeoffs of the Pareto-optimal configurations for all 4 applications are presented in Fig. 5. The average performance tradeoff is 8.0 times. The average power tradeoff is 5.0 times. The average energy tradeoff is 2.9 times. We make the following further observations based on the Pareto-optimal data that we gathered:

- The Pareto-optimal configurations are highly dependent on the applications. A configuration that resulted in the lowest-power consumption while meeting some performance constraint for one application did not result in the lowest-power consumption in the other applications.
- For the same application, most pairs of configurations, one from an old technology and one from the new technology that resulted in the equal performance or power, were different. This means that an optimal configuration in the old technology may or may not be an optimal configuration in the new technology.
- With respect to energy, the optimal configurations were those that lay in middle of the power/performance tradeoff curves. The energy plot for the jpeg example is given in Fig. 6. The configurations ordering is identical to those depicted in Fig. 5. Here, the rate of decrease in power consumption starts to become smaller compared to the rate of increase in execution time, resulting in a net increase in energy consumption.

Our exploration of the four examples revealed all the configurations of interest to a designer. The Pareto-optimal configurations were obtained in reasonable amount of time. Pareto-optimal configurations differed for different applications as well as technologies. An efficient simulation and exploration tool is necessary to achieve the best performance when mapping an application to a parameterized architecture.

V. CONCLUSION

We have presented an approach for efficiently finding all Pareto-optimal configurations of parameterized SOC architectures. Our approach relies on our knowledge about the interdependencies among parameters of the SOC. We use a directed graph to capture this interdependency and give algorithms that search the configuration space, incrementally, and prune inferior configurations. Our experiments with several examples mapped onto our target SOC architecture demonstrate the feasibility of the approach.

REFERENCES

[1] M. R. Stan and W. P. Burleson, "Bus-invert coding for low power I/O," *IEEE Trans. VLSI Syst.*, vol. 3, Mar. 1995.
[2] A. Malik, B. Moyer, and D. Cermak, "A programmable unified cache architecture for embedded applications," in *Proc. Int. Conf. Compilers, Architecture, and Synthesis for Embedded Systems*, Grenoble, France, July 2000.
[3] D. H. Albonesi, "Selective cache ways: On-demand cache resource allocation," *MICRO*, 1999.
[4] S. M. Kang, "Accurate simulation of power dissipation in VLSI circuits," *IEEE J. Solid-State Circuits*, vol. SSC-21, Dec. 1986.
[5] G. Y Yacoub and W. H. Ku, "An accurate simulation technique for short-circuit power dissipation based on current component isolation," in *Proc. Int. Symp. Circuits and Systems*, Aug. 1989.
[6] R. Tjarnstorm, "Power dissipation estimate by switch level simulation," in *Proc. Int. Symp. Circuits and Systems*, Aug. 1989.
[7] T. H. Krodel, "PowerPlay—Fast dynamic power evaluation based on logic simulation," in *Proc. Int. Conf. Computer-Aided Design*, Sept. 1991.
[8] E. Macii and M. Pedram, "High-level power modeling, evaluation, and optimization," *IEEE Trans. Computer-Aided Design*, vol. 17, Nov. 1998.
[9] D. Marculescu, R. Marculescu, and M. Pedram, "Information theoretic measures for power analysis," *IEEE Trans. Computer-Aided Design*, vol. 15, June 1996.
[10] M. Nemani and F. Najm, "Toward a high level power evaluation capability," *IEEE Trans. Computer-Aided Design*, vol. 15, June 1996.
[11] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: A first step toward software power minimization," *IEEE Trans. VLSI Syst.*, vol. 2, Dec. 1994.
[12] C. T. Hsieh, M. Pedram, H. Mehta, and F. Rastgar, "Profile driven program synthesis for evaluation of system power dissipation," in *Design Automation Conf.*, June 1997.
[13] C. Barndolese, W. Fornaciari, F. Salice, and D. Sciuto, "Energy evaluation for 32-bit microprocessor," in *Proc. Int. Workshop Hardware/Software Co-Design*, May 2000.
[14] R. J. Evans and P. D. Franzon, "Energy consumption modeling and optimization for SRAMs," *IEEE J. Solid-State Circuits*, vol. 30, May 1995.
[15] T. Givargis and F. Vahid, "Interface exploration for reduced power in core-based systems," in *Proc. Int. Symp. System Synthesis*, Dec. 1998.
[16] T. Givargis, J. Henkel, and F. Vahid, "Interface and cache power exploration for core-based embedded system design," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1999.

[17] T. Simunic, L. Benini, and G. De Micheli, "Cycle-accurate evaluation of energy consumption in embedded systems," in *Design Automation Conf.*, June 1999.
[18] Y. Li and J. Henkel, "A framework for estimating and minimizing energy dissipation of embedded HW/SW systems," in *Design Automation Conf.*, June 1998.
[19] D. Kirovski, C. Lee, M. Potkonjak, and W. Mangione-Smith, "Application-driven synthesis of core-based systems," in *Int. Conf. Computer-Aided Design*, Nov. 1997.
[20] P. R. Panda, N. D. Dutt, and A. Nicolau, "Local memory exploration and optimization in embedded systems," *IEEE Trans. Computer-Aided Design*, vol. 18, Jan. 1999.
[21] P. Lieverse, P. van der Wolf, E. Deprettere, and K. Vissers, "A methodology for architecture exploration of heterogeneous signal processing systems," in *IEEE Workshop Signal Processing Systems*, Taipei, Oct. 1999.
[22] T. Givargis, F. Vahid, and J. Henkel, "Instruction-based system-level power evaluation of system-on-a-chip peripheral cores," in *Asia and South Pacific Design Automation Conf.*, Jan. 2000.
[23] UCR Dalton Group. The Platune Project. [Online]. Available: www.cs.ucr.edu/~dalton/Platune.

Tony Givargis received the B.S. degree in computer science and the Ph.D. degree from the University of California, Riverside in 1987 and 2001, respectively.

He is currently an Assistant Professor in the Department of Information and Computer Science and a member of the Center for Embedded Computer Systems, University of California. He is a coauthor of the textbook *Embedded System Design* (New York: Wiley, 2002). His research interests include platform based system design, real-time resource management of embedded computing systems, and design space exploration.

Frank Vahid received the B.S. degree in computer engineering from the University of Illinois and the M.S. and Ph.D. degrees from the University of California, Irvine in 1988, 1990, and 1994, respectively.

He is currently an Associate Professor in the Department of Computer Science and Engineering at the University of California, Riverside. He is also a Faculty Member at the Center for Embedded Computer Systems at UC Irvine.

Dr. Vahid received the Outstanding Teacher Award from the College of Engineering in 1997. He was program and general chair for the IEEE/ACM International Symposium on System Synthesis in 1996 and 1997, respectively, and for the IEEE/ACM International Workshop on Hardware/Software Codesign in 1999 and 2000. He received the best paper award from IEEE Transactions on VLSI in 2000. He is coauthor of the textbooks *Embedded System Design* (New York: Wiley, 2002) and *Specification and Design of Embedded Systems* (Englewood Cliffs, NJ: Prentice-Hall, 1994). His current research interests include architectures and design methods for low-power embedded systems with an emphasis on tuning SOC platforms to their executing programs.

Jörg Henkel (M'95–SM'01) is currently a Senior Research Staff Member at the Computers and Communications Research Laboratories, NEC USA, Princeton, NJ, where he is leading several projects in research and development of low-power SOC architectures and EDA tools.

Dr. Henkel is currently the General Co-Chair of the IEEE/ACM International Symposium on Hardware/Software Co-Design (2002) and served for the same event as a Program Co-Chair in 2001. In addition, he currently serves as a Program Co-Chair for the IEEE International Workshop on Rapid Systems Prototyping 2002. He is involved in the following program committees: IEEE/ACM International Symposium on Hardware/Software Co-Design (Codes), IEEE/ACM Design Automation and Test in Europe Conference (DATE), and IEEE International Symposium on Low-Power Electronics and Design (ISLPED).