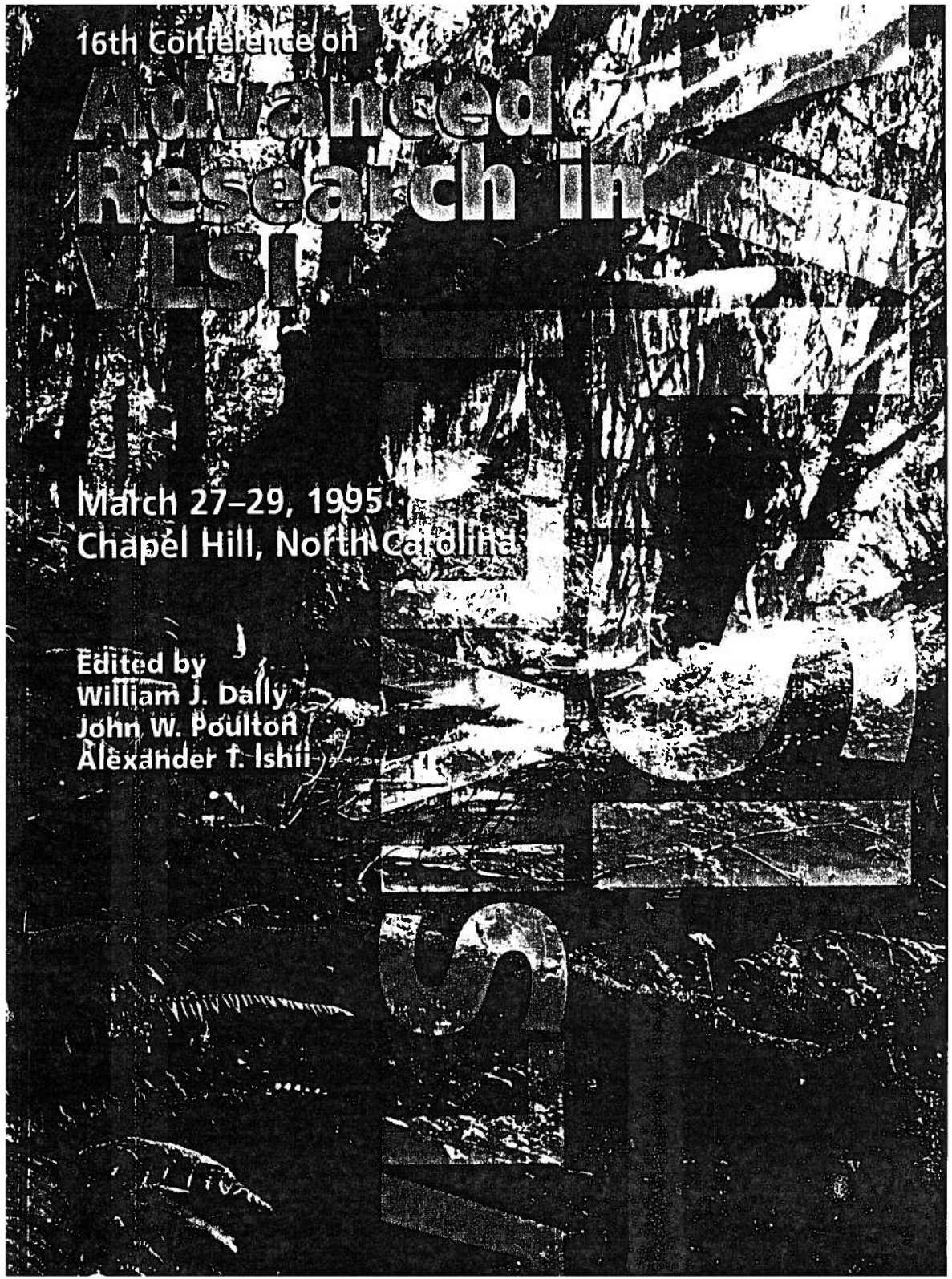
16th Conference on
Advanced
Research in
VLSI
March 27–29, 1995
Chapel Hill, North Carolina
Edited by
William J. Dally
John W. Poulton
Alexander T. Ishii

Systematic Objective-driven Computer Architecture Optimization

Timothy J. Stanley *
Trevor Mudge
1301 Beal Street
Advanced Computer Architecture Laboratory
University of Michigan
Ann Arbor, MI 48109

Abstract

Computer designers now have more transistors and architectural alternatives than at any time. Computer-aided design tools automate much of the physical design process. However, few tools have been developed to help the computer architect specify near-optimal microarchitectural configurations in the early design stages. Such tools are needed to systematically guide the early design specifications subject to multiple objectives such as cost, performance, and power consumption.

This paper illustrates an objective-driven microarchitectural design methodology that couples the specification design phase with an optimization technique. The design of a memory hierarchy with multiple performance objectives is used as a case study. This is a directed search problem with a high dimensionality. We show that the genetic algorithm, a global optimization technique based on the metaphor of natural selection and survival of the fittest, is an ideal candidate for such an objective-driven search in a high dimensional space. The paper concludes that search techniques such as genetic algorithms are necessary to systematically and efficiently drive architectural optimizations for multiple objectives such as dynamic power, and performance in the early, high-impact stages of the design process.

1: Introduction

Computer designers now have more transistors and microarchitectural alternatives than have been available at any time. [1] The hardware available to computer designers as measured by devices on a single chip has increased thirteen orders of magnitude since 1960 [18]. Architects have eagerly awaited the time when so many inexpensive hardware resources would be available for design [21]. However, this abundance has only served to emphasize the lack of systematic design procedures for optimally partitioning hardware resources among many competing architectural structures.

The continued advance of CAD tools has made the design process from the microarchitecture to chip layout highly automated - we can expect this to trend to continue. In contrast, the specification of the microarchitecture is neither automated nor driven by traditional optimization techniques. This is particularly unfortunate because early microarchitectural design decisions have a dramatic impact on final system performance.

A great deal of computer architecture work in the 1980s promoted a quantitative approach to computer architecture and design [12]. Much computer architecture research proposes a novel idea requiring additional hardware, quantifies the value of the idea with respect to a baseline design, and concludes that the idea has merit because the performance of the baseline improves. Questions

* This work supported by Advanced Research Projects Agency under ARPA-AASERT Contract Number DAAH04-93-G-0249

[1] In this paper, the words architecture and microarchitecture refer to the selection and organization of hardware structures such as local memory and functional units used in a chip implementation.

0-8186-7047-9/95 $04.00 © 1995 IEEE

Design Priority	Computer Design Example
Design Dimensions	system clock speed(s).
	multiple function units such as adders, multipliers, dividers, load/store units.
	multiple special purpose local (cache) memories having a wide variety of design parameters.
	any number of microarchitectures in which to interconnect these hardware structures.
	dozens of implementation technologies with different costs and performances.
Design Constraints	a hardware budget, e.g., a fixed number of transistors, or total chip real estate.
	an electrical static power budget.
	a manufacturing cost budget.
Design Objectives	maximizing some overall performance measurement, e.g., minimum cycles per instruction (CPI).
	minimizing dynamic power consumption.
	minimizing reference traffic to main memory.

Table 1. Some Computer Design Dimensions, Constraints, and Objectives.

A computer architecture design point is selected by combining multiple structures and parameters from the *design dimensions* to create a configuration that satisfies the *design constraints*. The performance of this design against multiple *design objectives* is determined using simulation techniques.

regarding the quality of the initial baseline design, or whether the additional hardware would best be applied in some other way remain unaddressed. From a more global perspective, then, the validity of the architectural idea cannot be accurately assessed.

Because changes to any aspect of an architecture can have unexpected ramifications for the rest of the architecture, it is difficult for the architect to find an optimal design point. The problem is illustrated in Table 1. The wide range of possible configurations under these design dimensions demand that an efficient and systematic method be used when iteratively optimizing the design of computer architectures.

This optimization method must be *efficient* because the sheer complexity of the design space, and the iterative nature of the design problem, do not permit frivolous exploration of regions that do not best satisfy the design objectives. The optimization approach must be *systematic* to free the architect from the considerable organizational burden of evaluating and plotting the results of many simulations and proposing the next set of designs to evaluate. In fact, the architect may not even have an efficient and systematic approach (heuristic, quantitative, or otherwise) to solve such a complex optimization problem. If this part of the architectural optimization process could be automated, the architect could more productively spend energy creatively investigating sources of lost performance on a smaller set of near-optimal designs and proposing novel solutions to improve performance for the next optimization iteration.

A high-level overview of a conventional computer design process is shown in Figure 1. The synthesis portion of the computer design process makes extensive use of automated computer-aided techniques to speed design time and improve design quality. However, the architectural specification and optimization portion of the design process remains non-systematic and human-intensive. The genetic algorithm, a global optimization technique useful in other fields of research [13] [10] [6] and based on the metaphor of natural selection and survival of the fittest, appears well-suited to the architectural specification problem for the following reasons:

1. it can be extended to consider multiple design objectives,
2. it widely explores large and complex design spaces in a systematic manner,
3. it is a knowledge-based search method,
4. it can be highly parallelized.

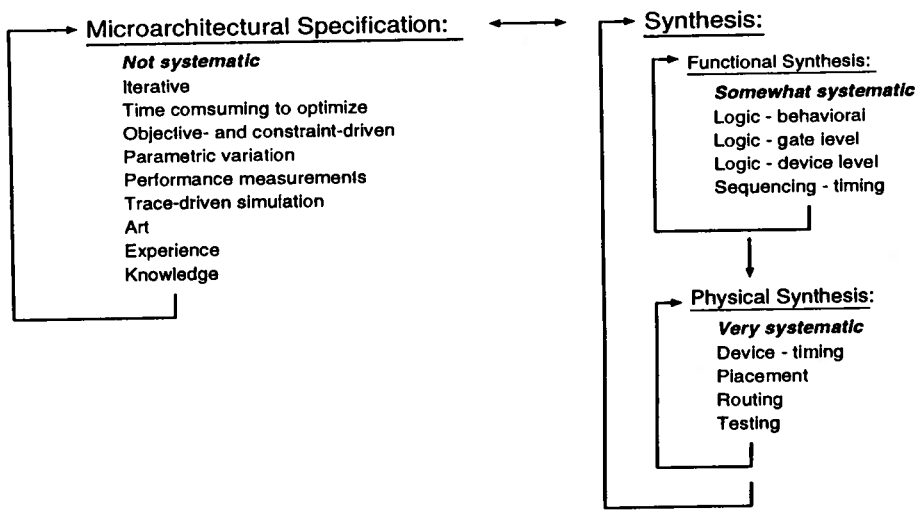

Figure 1. Computer-aided design for computer implementation.

All portions of the computer design process are iterative. Many parts are also NP-complete. Architectural specification and optimization are the only parts of the process lacking a systematic approach.

1.1: Overview of Paper

The remainder of this paper is organized as follows: Background work related to computer architecture optimization is discussed in Section 2. The problem of optimally partitioning hardware in computer design is reviewed. In Section 3, the genetic algorithm is described in the context of the computer architecture problem. The experimental method used to determine architectural system performance is described in Section 4. In Section 5, experimental results demonstrate the use of the genetic algorithm optimization on a high-dimensionality cache memory hierarchy optimization problem with multiple simultaneous objectives. Conclusions are drawn in Section 6. Based on the problem description and the results to date, the genetic algorithm is proposed and demonstrated as a systematic, efficient, multi-objective optimization method. Potential applications of this computer design approach are enumerated.

2: Background and Analysis

2.1: Complexity of the Computer Architecture Design Space

Consider a contemporary computer memory hierarchy implementing any number of the local memories enumerated in Table 2. The design dimensions for each local memory is selected from the list found in Table 3. The total amount of hardware expended on local memories can not exceed the hardware or power budget. Recently, researchers have considered the problem of optimizing the cache hierarchy when very large numbers of transistors are available [15] [7] as well as "unified" strategies to do so [3].

The similarity between the known NP-complete problem of *bin packing* and the combinatorial problem of partitioning a finite amount of hardware into a number of hardware structures to maximize some objective function is obvious [8]. The architectural optimization problem is composed of two parts. First is the NP-complete problem of enumerating all possible hardware partitions. Second is the problem of efficiently directing the search through these partitions toward an optimal solution based on a set of specified objectives. The exhaustive search methods often used to identify optimal hardware partitions for small design spaces, e.g. [20], are not possible given the size of the design space in Tables 1, 2, and 3. Rather, the architect must purposefully gravitate toward an optimum point in the design space to study performance.

Cache Type	Brief Description
Primary (L1)	First level of memory hierarchy. Usually on-chip in contemporary microprocessor implementations. Often split for instruction- and data-stream.
Write cache or buffer	Holds data writes. Decouples upper portion of memory hierarchy from lower portion of hierarchy. Can collate multiple writes to same line(s) [2].
Miss Cache, Victim cache	Holds most recent cache miss(es) of the preceding cache in the hierarchy. Conceptually offers an extra degree of associativity to preceding cache with ability to "move" this associativity to cache hot spots to reduce conflict misses [14].
Prefetch and stream buffers	Holds prefetched data.
Secondary (L2)	A larger and slower backup to the L1 caches. Could be implemented off-chip in SRAM, or any of the several new DRAM technologies, as well as on-chip [15].

Table 2. Partial taxonomy of local memories in microprocessor implementations.

This list is not intended to be inclusive, merely representative of the local memories included in the architecture studied in this paper. Local memories can be dedicated to the instruction-stream, the data-stream or unified to satisfy both streams. Further, all of the structures described above can be organized to form a "memory hierarchy" and could reside at virtually any position in this hierarchy.

2.2: Computer-aided Design for Architecture

Considerable computer-aided design (CAD) research continues in optimal *synthesis* of architectures (e.g. [9]), i.e., given an architectural specification (cache list and parameters, functional unit list and parameters, etc.), generate an optimally partitioned hardware representation. Far less research has been conducted on CAD techniques to optimally identify exactly what architectural structures should be used in the starting architectural *specification*.

Kumar and Davidson developed a hierarchical approach to computer system design and performance evaluation [17]. These researchers noticed that computer design methods were not systematic or efficient. They pruned the search space, and used a limited number of design parameters. Their method had problems distinguishing between locally versus globally optimal design points.

Conte [4] has partitioned the computer specification problem into two parts, memory hierarchy specification and functional unit specification. Exhaustive, single-pass simulation techniques were applied to the memory hierarchy partition. Simulated annealing runs of 150 simulations were applied to the functional unit partition. Performance interaction between the memory hierarchy and the functional units was not addressed. More recently, he has used similar techniques to investigate the impact of power and area on processor design [5].

3: The Genetic Algorithm [2]

The genetic algorithm (GA) [13] is a global search technique that has been successfully applied to many real-world optimization problems [10] [6] [1] and to similar NP-complete problem spaces in other disciplines [16]. The GA is based on the metaphor of natural selection and survival of the

[2] As applied to the computer architecture optimization problem.

Cache Parameter	Range of values
Block size (bytes). Also called line size.	As small as 4 bytes for on-chip caches to 1K+ bytes for off-chip caches.
Associativity (degree)	From 1 (direct-mapped) to the number of lines in the cache. Describes the number of cache locations where data belonging to an address could reside.
Size (bytes)	From 1 line to n lines. If fully associative, by increments of 1 line. If not fully associative, increments by multiples of 2. On-chip caches might be 4k bytes in size; off chip caches might be many megabytes.
Replacement Policy	A cache having associativity > 1 needs a replacement policy to select between blocks within a set. Popular replacement policies include: • random, • first-in first-out (FIFO), • least recently used (LRU), • not most recently used (NMRU), • not most recently replaced (NMRR).
Reference stream	Instruction, data, or unified. Write caches are special type of data caches.

Table 3. Partial taxonomy of local memory design parameters.

Several parameters affect the amount of hardware dedicated to the local memory and its performance, including size, associativity, and replacement policy. Not all parameters apply to every type of local memory.

When multiple caches (see Table 2) are being considered to build a memory hierarchy within a finite hardware constraint, the design space is simply too large to exhaustively explore.

fittest. At the most basic level, the data structures and process flow of the genetic algorithm are schematically shown in Figure 2. The advantage of the genetic algorithm over other methods is that it purposefully uses information from previous optimizations attempts to direct future attempts, much like a human architect. GAs converge upon a solution, rather than pruning the design space [17] or partitioning the design space [4]. At the same time, the genetic algorithm explores the search space widely and is regarded as a good global optimization technique on difficult multi-modal solution spaces.

A splice from a computer architecture chromosome string is shown in Figure 3. Each gene in the chromosome represents the value of one parameter in the architectural space. The GA uses a two types of genetic operators to create offspring from highly fit parents. Figure 3 schematically shows the creation of an offspring from two highly fit parents using the crossover operator. Mutation operators are probabilistically employed to constantly add diversity to the genetic material present in the population. For the experiments in this study, a population size of 100 chromosomes was used, 2 children were created per generation, the probability of crossover was 1.0, and the probability of mutation was 0.064. To permanently maintain genetic diversity, the population replacement policy prevents multiple duplicate chromosomes from simultaneously existing in the population.

Computer engineering environments are typically composed of many networked computers. The coarse-grained, explicit parallelism embodied in the population of the GA, and the small amount of communication required between population individuals makes it well suited for such networked parallelism. This strengthens the case for its use as a computer architecture optimization method over other less parallel alternatives.

3.1: Multiple Simultaneous Objectives

The GA used in the study considers the three following architectural objectives.

Hardware Budget. Mulder et. al. have developed an analytical model to compute the area of an on-chip memory structure [19] as a function of its size and organization (see Table 3). This work defines a technology-independent unit called the *register bit equivalent (RBE)*, as the amount

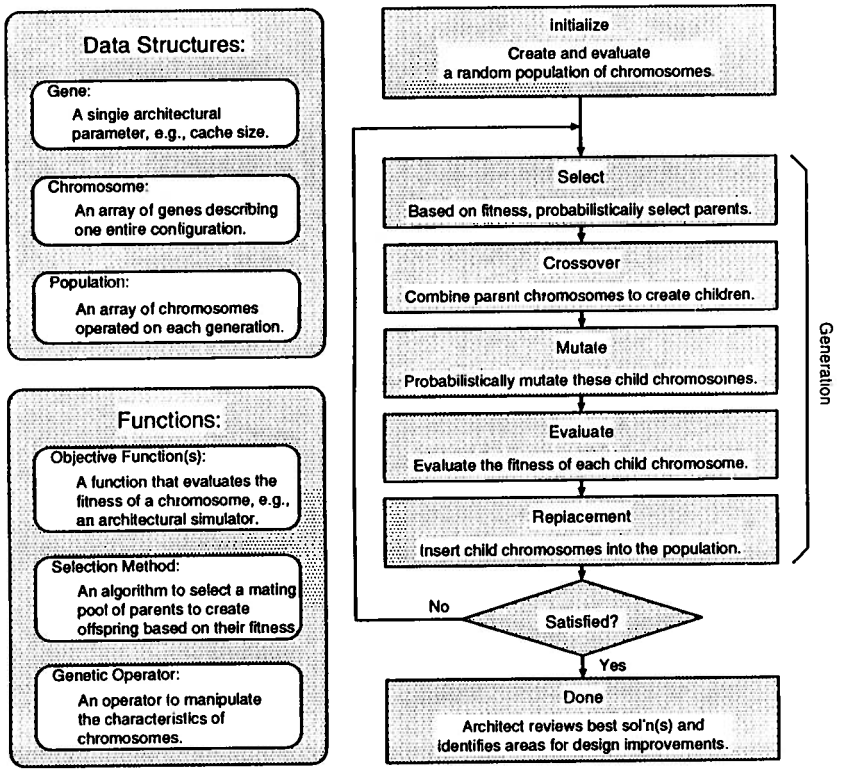

Figure 2. Genetic algorithm data structures and flow diagram.

The genetic algorithm works with a population of candidate solutions. During each generation, the GA combines the characteristics of the best candidates. In successive generations, good characteristics receive exponentially increasing reproductive opportunities leading to a convergence of the gene values. This is known as "reproduction with emphasis" [13]. As time proceeds, better candidate solutions are evolved and the fitness of the population increases.

of area consumed by 1 bit of a register file. Models are presented that express the area of an on-chip memory structure in RBEs as a function of its size, associativity, and bandwidth. We use this RBE model to compute the area consumed by the local memories as a function of their design parameters. We have extended this model to consider the additional hardware requirements of different cache replacement policies.

Using this model, an 8K byte direct-mapped cache with a line size of 32 bytes would require 49,839 RBEs; a 64K byte direct-mapped cache with a line size of 128 bytes would require 366,156 RBEs. For this study, we will assume that (3 * 128K) RBEs are available on-chip to implement the cache hierarchy described in Section 4.1.

Performance. One architectural performance measure is *cycles per instruction (CPI)*. Consider a benchmark composed of N instructions. We want to identify a design partition satisfying the hardware constraint, while minimizing the number of clock cycles required to execute those N instructions, and thus minimize CPI. The CPI for an architecture under consideration is obtained by simulating the architecture running a benchmark, counting the number of cycles consumed and instructions executed, and computing CPI directly.

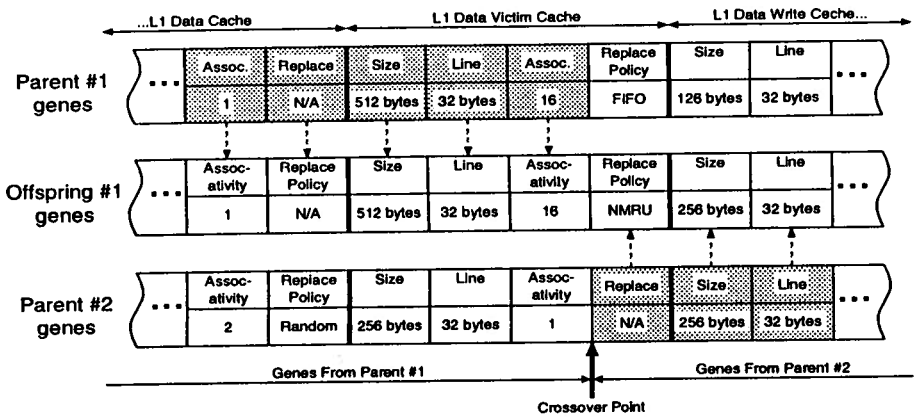

Figure 3. Schematic representation of the crossover operator.

To provide reproduction with emphasis, methods of combining two high-fitness parents to produce (hopefully more highly fit) offspring are employed. One or more locations on the gene string are randomly chosen at each reproductive opportunity. The offspring receives genes prior to the crossover point from the first parent, and genes occurring later in the string from the second parent. A second offspring (not shown) receives the genes not used in the first offspring.

Mechanisms must be used to ensure that the offspring gene string describes a legal configuration. In this example, parent #1 provides the offspring with L1 victim cache 16-way associativity, while parent #2 provides no replacement policy since its victim cache was direct-mapped. Associativity > 1 requires a replacement policy. This offspring must be supplied with randomly selected replacement policy to make it a legal configuration.

Power Factor. The amount of power consumed by a memory structure in a CMOS technology is a function of the number of times the transistors in that hardware structure are switched, and the size of the structure. If the transistors are switched every cycle, due to pre-charging issues, or organizational issues, then its power consumption is simply a function of its size. However, if the structure is designed so that its transistors switch only when an access occurs, then power consumption is a function of the structure's size *and* the number of accesses. In this study, we will assume that the on-chip memory structures have been designed to switch only when relevant references are made, rather than every cycle.

The percentage of overall hardware consumed by any specific local memory is computed by dividing its size (in RBEs) by the total amount of hardware dedicated to all memory structures in the design. The percentage of time that any specific local memory consumes power is computed by dividing the number of cycles it was accessed by the total cycles required to exercise the benchmark. A technology-independent power factor for each local memory is computed by dividing its RBE percentage by its switching percentage. This unit-less power factor will be between 0.0 and 1.0.

3.2: Pareto Optimality

Given multiple objective functions to compute chip real estate, performance, and power consumption of points in the design space, the identification of the better of several competing configurations depends on the relative priority or weight assigned to each objective. However, assigning weights to each objective is problematic - how do we determine what the relative weights should be? A formal method to rank such competing design points uses the concept of *Pareto optimality* [10].

Consider multiple competing points in the design space. A point is considered ***nondominated*** if

and only if there is no other point in the design space that better satisfies all of the objectives. In this specific problem we are *minimizing* real estate, CPI, and power. Mathematically, a vector of objectives **X** is partially less than vector **Y** (i.e., it better satisfies the minimization objectives and it *dominates* **Y**) when the following conditions hold:

$$(X <_p Y) \Leftrightarrow (\forall_i)(x_i \leq y_i) \wedge (\exists_i)(x_i < y_i) \quad (1)$$

This definition of Pareto optimality allows us to rigorously rank the quality of multiple competing design points while considering multiple objectives. First, the multiple objectives are assigned a priority order. The ranking procedure sorts all competing configurations into dominated and non-dominated groups based on their objectives. The non-dominated group is then further ranked by how well the first objective is met. If the first objective of two configurations are equal, the tie is broken by the second priority objective, and so on. These non-dominated ranked configurations are set aside in a ranked list. While the dominated group is not empty, the process is repeated. At each iteration, the ranked, non-dominated configurations are appended to the growing ranked list. When this procedure is done, the GA has a ranked list of Pareto-best to Pareto-worst design points and uses this list to make reproductive decisions during each generation.

4: Methodology

4.1: The Chip Architecture

The organization of 7 local memory structures to be optimized in this study are shown in the block diagram of Figure 4. There are 23 parameters to this cache simulation shown in Table 4. We want to systematically identify the value of each of these parameters in a near-optimal design. The instruction set is from the MIPS R3000, the pipeline is a single issue, 5-stage pipeline operating at 250 MHz, the secondary cache and main memory are modelled as Rambus RDRAM devices.

4.2: Architectural Simulator and Benchmarks

An address-trace driven simulator was written to evaluate the performance of the architecture in Figure 4 for three objectives specified in Section 3.1. Four benchmarks were chosen from the SPEC92 benchmark suite: the C compiler *gcc* and the logic minimization program *espresso* are integer benchmarks; the Monte Carlo simulation *doduc* and the highly vectorizable Navier Stokes equation program *hydro2* are floating point benchmarks. Traces from portions of these programs were used in a multiprogrammed mode with a quantum of 250,000 cycles. Over 174 million instructions were simulated per architectural configuration, accounting for over 227 million memory references.

5: Experiments and Results

5.1: Pareto Priority: Chip real estate, Performance, Power

The first experiment set the priority of the objectives as 1) meeting a hardware budget of (3 * 128K) RBEs, 2) minimizing CPI, 3) minimizing power. The Pareto-ranking of competing configurations is established using this priority as described in Section 3.2. Figure 5 plots the GA's progress at minimizing RBEs and CPI at the start of each generation. All configurations meeting the RBE budget are regarded as equally fit.

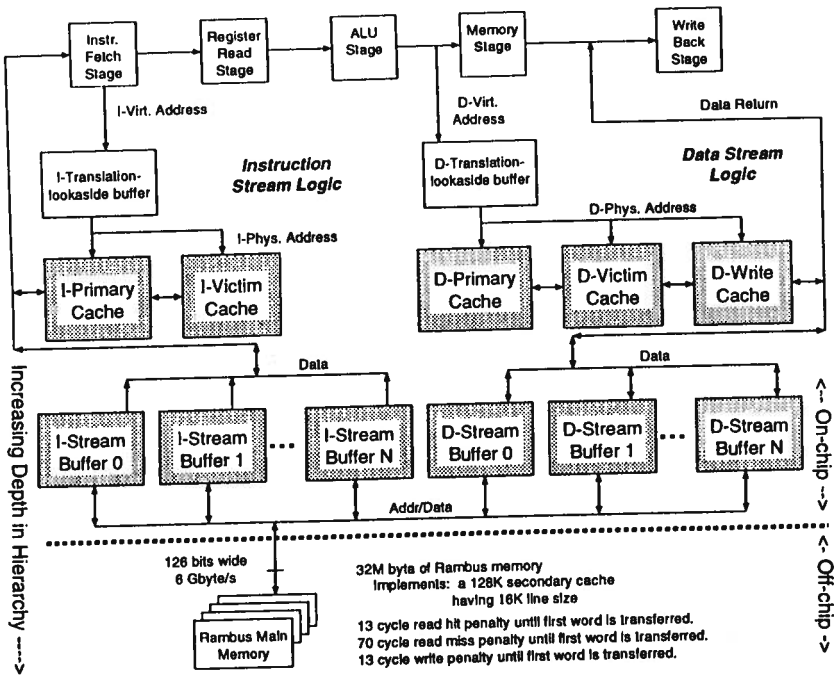

Figure 4. Several locality-specific caches assembled into a memory hierarchy.

On-chip local memories of interest are shown as shaded blocks. There are 5 caches, and 2 stream buffers. Stream buffers have a width (0..N above), i.e., the number of concurrent streams they can prefetch.

The instruction- and data-stream primary caches are accessed in 1 cycle. The victim caches are addressed every cycle their corresponding primary cache is addressed, however, there is an additional 1 cycle penalty when its data must be swapped with the primary cache. The data-stream write-cache is addressed with the data-stream primary cache and can provide read data in 1 cycle. However, write data that is not found in the write-cache must be fetched from the memory hierarchy before the store instruction can complete. Stream buffers are accessed only when caches higher in the hierarchy are unable to service the request, and add 1 additional cycle penalty if they have the data.

Parameter name	Minimum	Maximum	Increment
L1 I- and D-stream line size (bytes)	4	128	to next power of 2
Cache size (bytes)	128	64K	to next power of 2
Cache associativity (degrees of)	1	64	to next power of 2
Cache replacement policy	one of: random, FIFO, LRU, NMRU, NMRR		
Stream-buffer size (bytes)	128	64K	to next power of 2
Stream-buffer width	1	64	to next power of 2
Stream-buffer replacement policy	one of: random, FIFO, LRU, NMRU, NMRR		

Table 4. Local cache parameter ranges used.

The I- and D-stream line sizes are independent. All caches serving a stream use the same line size. The five replacement policies are enumerated in Table 3. This design space represents over $1.6 * 10^{19}$ configurations.

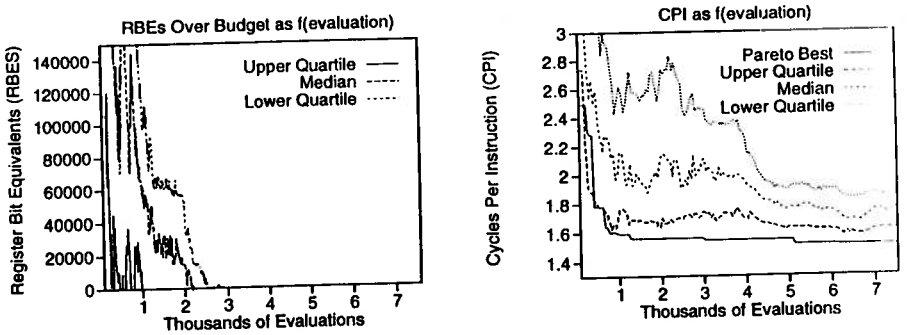

Figure 5. RBEs and CPI vs. number of evaluations.

As the GA optimizes the population, the number of RBEs used over the hardware budget converges to 0. As early as 309 evaluations, the entire upper quartile of best Pareto-performing configurations satisfy the hardware budget; by 2542 evaluations, the top 3 quartiles of the population satisfy the hardware budget.

CPI is minimized as the GA optimizes the entire population. As more evaluations are performed, the concentration of highly fit architectural configurations in the population that satisfy the hardware budget *and* provide low CPI steadily increases. Small reductions in CPI were observed until evaluation 7538.

Each evaluation represents one chromosome, i.e. one run of the simulator with a set of parameters to determine the configuration's objective fitness. A total of 7574 GA evaluations were performed. Because the GA aggressively explores high performance regions of the design space, it often considers the same configuration multiple times. In this experiment, 5635 unique architectural simulations were performed. Despite the focus of the search on specific regions of the design space, the GA used in this experiment implements a population replacement policy that does not allow multiple identical chromosomes to simultaneously exist in the population. This effectively maintains genetic diversity in the population and encourages sustained search.

The utility of the Pareto ranking is shown graphically in Figure 6. Within 670 evaluations, the GA evolves a mostly nondominated population according to the definition of Equation (1). These nondominated configurations drive the GA to explore the "Pareto frontier", i.e., areas of the design space where the multiple objectives are best satisfied simultaneously.

The architect must extract useful design information from this set of nondominated configurations as the GA proceeds. When 95 % of the values of a single gene (an architectural parameter) in the population have evolved to the same value, that gene is said to have converged. The entire population has converged when 95 % of the population's genetic material has converged according to this definition. The right graph of Figure 6 shows the convergence history of two interesting genes. The gene for the L1D-write cache size converged more rapidly than any other gene in the simulation. The architect should simply identify the converged value of that gene for design direction. However, the gene for the L1D-victim cache has converged less than the average of all genes and does not contain specific design information.

The architect must consider the distribution of L1D-victim cache sizes, as well as the Pareto best configuration, for design direction. The use of gene distribution as a design tool is illustrated in Figure 7. When the entire population is considered, two stable, competing, non-adjacent design space niches are seen. However, when only the upper quartile of the population is considered, the design decision is clarified. In this instance, the L1D-victim cache size has not definitively completed its evolution.

The architect should *not* use the mean gene value to direct design choices. In Figure 8, the evolution of the L1D-write cache and L1D-victim cache are shown. Notice that the population's mean value (using $\log_2$ of the cache size) for the L1D-victim cache during most of the simulation

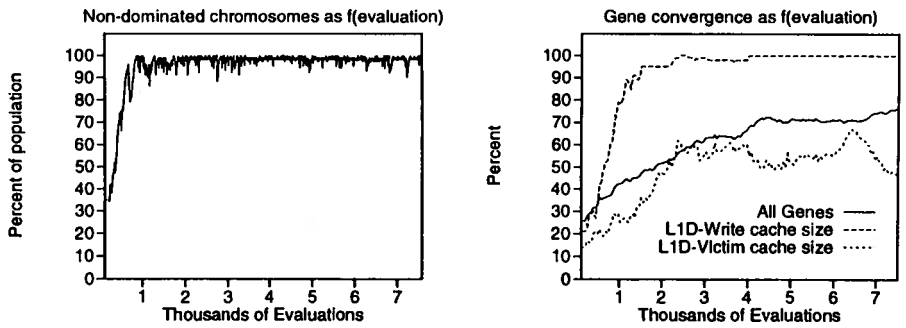

Figure 6. Characteristics of the population vs. number of evaluations.

As the GA rewards Pareto nondominated configurations with increased reproductive opportunities, eventually, most of the population becomes nondominated.

As the GA processes the genetic material of the population there is a convergence of architectural parameters. Some genes, such as the L1D-write cache, converge early to a single value. Other genes, such as the L1D-victim cache, converge at a rate slower than average. Graphs such as this showing the trajectory of each architectural parameter provide the architect with useful design information about near-optimal points in the design space.

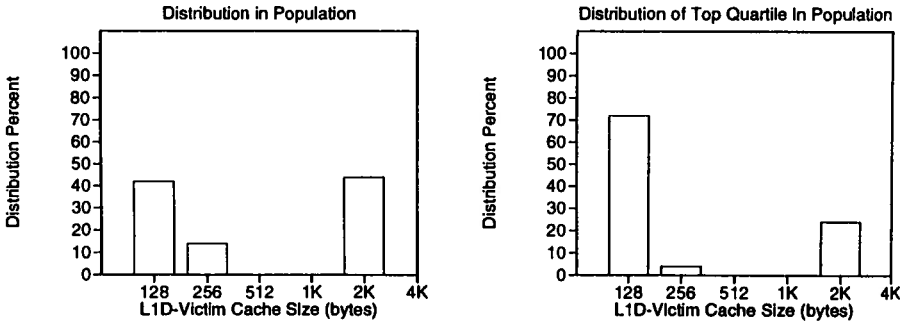

Figure 7. Distribution of L1D-victim cache size in population.

The final distribution of L1D-victim cache size is shown for the entire population on the left. Two stable niches in the design space are seen at 128-256 bytes, and at 2K bytes. When only the top performing quartile of the population is considered, the 128 byte niche is strongest.

would lead the architect to conclude that a 512 byte L1D-victim cache should be selected. However, the distributions show that no configuration in the final population has a 512 byte L1D-victim cache.

To illustrate the pitfall of prematurely using the Pareto-best configuration alone for design direction, consider that the L1D-victim cache size in the Pareto-best configuration had been fairly stable at 2K bytes since evaluation 2986. As shown by the gene distribution, the genetic material in the population reflects this history and genetic bias. The disruption of L1D-victim cache convergence at evaluation 6473 reflects the point in time when the gene value of 128 began to increase in distribution. It was not until evaluation 7223 that the Pareto-best configuration finally moved to the

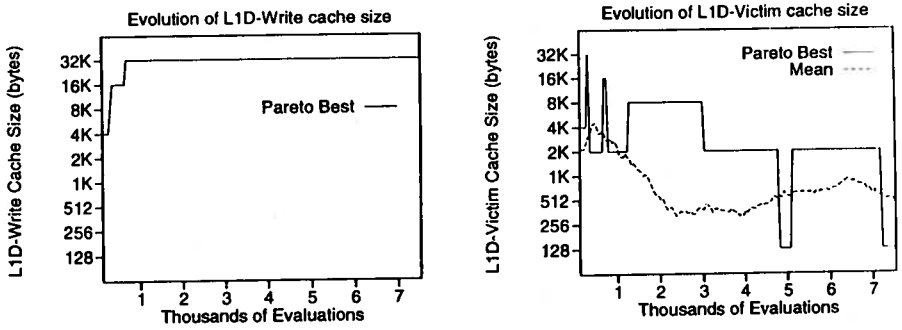

Figure 8. Evolution of two genes vs. number of evaluations.

The L1D-write cache rapidly and directly evolves to 32K bytes. The L1D-victim cache size evolution is much more complex. The value of the Pareto-best configuration does not track the mean value of the population.

128 byte L1D-victim cache. As a rule of thumb, if the gene value of the Pareto best configuration is not the most strongly expressed value in the population, the GA is still evolving the gene.

The behavior and design analysis of most genes falls somewhere between the simplicity of the rapidly evolving L1D-write cache, and the complexity of the slowly evolving L1D-victim cache. The final fitness and configuration achieved in this experiment are shown in Table 5. Using this information, the architect concludes that a CPI of 1.50 can be achieved within the specified hardware budget while having a power factor of 0.16. The top quartile configurations can be used as near-optimal baseline design points for further architectural work.

5.2: Pareto Priority: Chip real estate, Power, Performance

The second experiment set the priority of the objectives as 1) meeting the same hardware budget of Section 5.1, 2) meeting a power factor of 0.07 or less, 3) minimizing CPI. All configurations meeting the power factor budget are regarded as equally fit. Figure 9 plots the GA's progress at meeting the power factor budget and minimizing CPI at the start of each generation. The RBE graph, not shown, is similar to that shown in Figure 5.

As the GA proceeds, the population is increasingly composed of many configurations with lower CPI (upper quartile, right graph), but also an over-budget power factor (upper quartile, left graph). These configurations contribute by directing the search, but are not the Pareto-best. Because the GA does not evolve an entire population of configurations that satisfy the power factor budget, we conclude that the budget is very aggressive and that the density of satisfactory points in the design space is low.

At final evaluation of 5855, 4831 unique architectural simulations had been performed. Given these objective priorities and budgets, the architect can expect to achieve a performance of 1.81 CPI given a hardware budget of (3 * 128K RBEs) and a power factor budget of 0.07.

5.3: Analysis of Two Regions in Design Space

The final near-optimal design configurations found by the two experiments for the architecture described in Figure 4 are summarized in Table 5. It is the priority and specification of different objectives for the two experiments that directed the GA to systematically explore two different regions of the architectural design space.

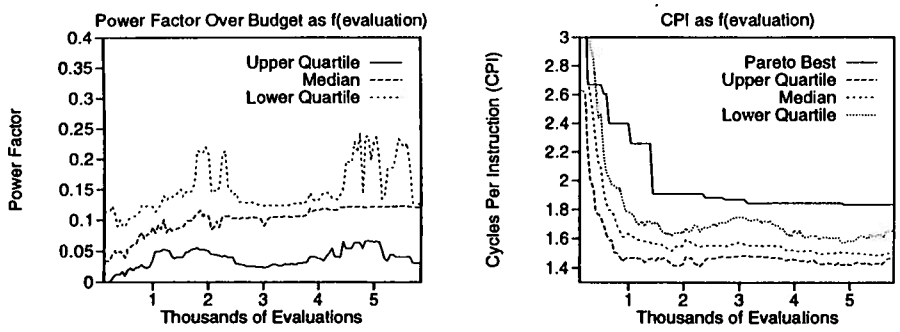

Figure 9. Mean power factor and CPI vs. number of evaluations.

As the GA proceeds, it is unable to evolve an entire population within the power factor budget (of 0.07). However, some chromosomes, including the Pareto-best, satisfy the power budget.

While the GA guides the architectural search in portions of the design space where hardware usage and power are near budget, it simultaneously optimizes for CPI using Pareto optimality. The Pareto-best configuration represents a systematic tradeoff of higher CPI, for a lower power factor.

The first observation is that the cache *sizes* evolved by the GA for the two experiment differ in only two instances. When the power budget is not specified, a 16K byte L1I-primary cache and a 512 byte L1I-stream buffer are preferred. When a power budget is specified, a 4K byte L1I-primary cache and a 2K byte L1I-stream buffer are preferred. We regard the cache sizes as the highest impact parameters in performance and area, and are encouraged to observe this level of GA consistency between two independent experiments.

A major architectural observation is that a very large and associative L1D-write cache is preferred in both experiments. This reflects the nature of the benchmark programs, and the interaction between the caches. The power of the large write cache is due to its write-back miss policy. Also, the high associativity of the write cache obviates the need for a L1D-victim cache. This architectural fact is confirmed as the L1D-victim cache ultimately evolves to a very small size. In consideration of these results, we question the utility of the L1D-write cache and victim caches in general when substantial amounts of on-chip memory are available. Our next experiment might be to eliminate the L1D-write and victim caches entirely, and simply provide a single write-back L1D-primary cache.

These experiments assumed that the clock speed of the chip was not limited by the access speed of any specific local memory structure. Future architectural work should consider impact on chip clock speed of large and associative caches such as the L1D-write cache and use time-per-instruction (TPI) as a performance measurement, rather than CPI.

6: Conclusions

In consideration of the rapid growth in available hardware and architectural ideas from which to construct a computer system, computer architects need better optimization tools and techniques to systematically direct the design space search based on the specified objectives. The genetic algorithm has been demonstrated as an effective, multi-objective, objective-based search technique using a multi-dimensional cache hierarchy as a case study.

Designers are not usually interested in a single near-optimal point in the design space. More often they are interested in characteristics of near-optimal configurations in a baseline architecture so that they can identify performance bottlenecks and propose new hardware structures to achieve

		Units	Experiment 1 Pareto Best	Experiment 1	Experiment 2 Pareto Best	Experiment 2
Objective		(Fitness)		Std. dev.		Std. dev.
RBEs over hardware budget		RBEs	0.00	0.000	0.00	18510
Performance		CPI	1.50	0.128	1.84	0 120
Power Factor		unit-less	0.16	0.033	0.00	0.068
Cache Parameter		(Gene values)		Converge %		Converge %
I-Stream	Line Size	bytes	64	100	64	98
Primary Cache	Size	bytes	16K	44	4K	52
	Associativity	degree	2	71	2	68
	Repl. policy	—	FIFO	55	NMRR	69
Victim Cache	Size	bytes	256	68	256	62
	Associativity	degree	2	67	2	69
	Repl. policy	—	Random	00	NMRU	31
Stream Buffer	Size	bytes	512	65	2K	50
	Width		1	83	2	51
	Repl. policy	—	NMRR	86	NMRU	70
D-Stream	Line Size	bytes	32	82	32	63
Primary Cache	Size	bytes	8K	57	8K	51
	Associativity	degree	4	83	2	54
	Repl. policy	—	Random	83	NMRR	97
Write Cache	Size	bytes	32K	100	32K	100
	Associativity	degree	8	98	8	98
	Repl. policy	—	FIFO	53	NMRR	61
Victim Cache	Size	bytes	128	44	128	77
	Associativity	degree	4	78	2	80
	Repl. policy	—	Random	53	NMRR	46
Stream Buffer	Size	bytes	4K	100	4K	57
	Width		16	99	16	66
	Repl. policy	—	FIFO	78	NMRU	63

Table 5. Tabulated objectives and cache parameters evolved for two experiments.

A low objective standard deviation indicates that the GA has sorted through the design space and retained many Pareto optimal configurations with similar performance. A high cache parameter convergence indicates the strength of the GAs recommendation for that parameter.

The power factor for the first experiment represents an absolute value; the power factor for the second experiment represents over budget power.

yet higher performance.

During the iterative, high-impact, early stages of the architectural design process, our approach systematically couples the specification and optimization phases of design. We conclude with some computer architecture design examples which can be approached only with a systematic, objective-driven approach such as ours.

Technology-Specific Optimization of Similar Microarchitectures. Various implementation technologies (GaAs vs. CMOS, SRAM vs. DRAM) have different costs and performances. This makes apples-to-apples comparisons difficult, since implementations in each technology need to be specifically optimized before comparisons can be made. A systematic objective-driven optimization tool is a large step toward permitting such comparisons.

Application-Specific Optimization of Similar Microarchitectures. Different applications have different architectural requirements. The embedded system market is particularly cost-driven and typically has a narrow, well-defined workload. Some vendors claim that they are capable of fabricating application-specific microprocessors for customers [11]. We envision a baseline core microprocessor design that is architecturally optimized (cache size selection, functional unit selection) on a rapid, application-specific basis using architectural CAD tools.

Power management. Low-power applications have become very important. All of the new DRAM

technologies provide low power modes of operation. However, each has dramatically different performance characteristics for normal and low-power modes of operation. Systematic optimization techniques are needed to compare the performance of these new technologies because each technology requires different optimal hardware support.

References

[1] E. R. Altman, V. K. Agarwal, and G. R. Gao. A Novel Methodology using Genetic Algorithms for the Design of Caches and Cache Replacement Policy. In *Proc. of the 5th Int. Conf. on Genetic Algorithms*, 1993.

[2] B.K. Bray and M.J. Flynn. Writes caches as an alternative to write buffers. Technical Report CSL-TR-91-470, Stanford University, April 1991.

[3] C. Chen and A. K. Somani. A unified architectural tradeoff methodology. In *Proc. of the 21st Ann. Int. Symp. on Computer Architecture*, 1994.

[4] T. M. Conte. *Systematic Computer Architecture Prototyping.* PhD thesis, University of Illinois, Urbana-Champaign IL USA, July 1992.

[5] T. M. Conte, K. N. P. Menezes, and S. W. Sathaye. The impact of power and area efficiency on superscalar processor design. Technical report, University of South Carolina, 1993.

[6] L. Davis. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, 1991.

[7] M. Farrens, G. Tyson, and A. R. Plezkun. A study of single-chip processor/cache organizations for large numbers of transistors. In *Proc. of the 21st Ann. Int. Symp. on Computer Architecture*, 1994.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, San Fransisco, CA, 1979.

[9] C. H. Gebotys. An Optimization Approach to the Synthesis of Multichip Architectures. *IEEE Transactions on Very Large Scale Integration Systems*, 2(1):11–20, March 1994.

[10] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, & Machine Learning.* Addison-Wesley, 1989.

[11] Linley Gwennap. ARM250 Integrates RISC System on a Chip. *MicroProcessor Report*, 6(14):22-24, October 1992.

[12] J.L. Hennessy and D.A. Patterson. *Computer Architecture, A Quantitative Approach.* Morgan Kaufman, San Mateo, California, 1990.

[13] J. H. Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, 1992. (First edition 1975, Ann Arbor: University of Michigan Press).

[14] N. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Proc. of the 17th Ann. Int. Symp. on Computer Architecture*, pages 364–373, New York NY (USA), May 1990. IEEE.

[15] N. P. Jouppi and S. J. E. Wilton. Tradeoffs in two-level on-chip caching. In *Proc. of the 21st Ann. Int. Symp. on Computer Architecture*, 1994.

[16] K. Juliff. A Multi-chromosone Genetic Algorithm for Pallet Loading. In *Proc. of the 5th Int. Conf. on Genetic Algorithms*, 1993.

[17] B. Kumar and E. S. Davidson. Computer system design using a hierarchical approach to performance evaluation. *Communications of the ACM*, 23(9):511–521, September 1980.

[18] Don Lindsay. The Limits of Chip Technology. *Microprocessor Report*, 7(1):21–24, January 25 1993.

[19] J. M. Mulder, N. T. Quach, and M. J. Flynn. An Area Model for On-Chip Memories and its Application. *IEEE Journal of Solid-state Circuits*, 26(2):98–105, February 1991.

[20] D. Nagle, R. Uhlig, T. Mudge, and S. Sechrest. Optimal allocation of on-chip memory for multiple-api operating systems. In *Proc. of the 21st Ann. Int. Symp. on Computer Architecture*, 1994.

[21] R.L. Sites. How to use 1000 registers. In *CalTech Conf. on VLSI*, 1979.

Invited Presentation:

Design of Low-Power Portable Systems

Speaker:
Anantha P. Chandrakasan
Massachusetts Institute of Technology