A Parallel Genetic Algorithm for Multiobjective Microprocessor Design

Timothy J. Stanley and Trevor Mudge
1301 Beal Street
Advanced Computer Architecture Laboratory
University of Michigan
Ann Arbor, MI 48109-2122

Abstract

The microprocessor chip designer must solve the problem of partitioning millions of transistors into an arbitrary number of hardware structures within a finite chip area toward achieving maximum performance. This combinative complexity is compounded by a lengthy performance evaluation of each proposed design. We present the application of a real-valued multiobjective genetic algorithm on an asynchronous parallel workstation network as a optimization approach well suited to this problem. By casting design budget constraints as multiple design objectives, the need for penalty functions is eliminated. A microprocessor cache memory design problem is optimized with the genetic algorithm.

1 Microprocessor Design Problem

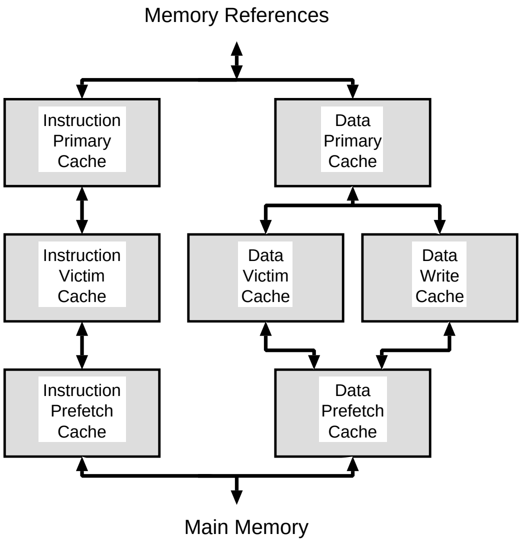
Microprocessor chip designers now have more transistors and design alternatives available to them than at any time in the past. The chip designer's selection of hardware structures from many alternatives (e.g., adders, multipliers, memories) must maximize microprocessor performance. The designer must solve the combinatorial design problem of partitioning millions of transistors into an arbitrary number of hardware structures within a finite chip area while achieving this goal. At the most basic level the microprocessor design problem is characterized by:

1. specifying finite chip size and power dissipation budget constraints,
2. specifying chip performance objectives,
3. selecting and interconnecting many hardware structures, each of which consumes area from a real-estate budget, and power from a power budget, into a chip *microarchitecture*,
4. organizing these structures to provide maximum performance within all design budgets,
5. modeling the microarchitecture in a high-level language and simulating its performance,
6. proposing modifications to the specifications and microarchitecture to overcome quantified performance shortcomings,
7. iterating this process until a balanced, near-optimal design is identified.

Performance interactions exist between interconnected hardware structures. Each microarchitectural design iteration alters these interactions, and requires an entire re-assessment of the utility of each hardware structure in the context of the new design. Questions regarding whether the hardware is best spent between multiple competing structures must be constantly re-evaluated, e.g., increasing the size of some on-chip hardware structure $\mathcal{M}$ means that less hardware is available for on-chip hardware structures $\mathcal{N}$, $\mathcal{O}$, and $\mathcal{P}$. The similarity between the microarchitectural design problem and NP-complete problems such as set partitioning or bin packing is apparent.

The combinative complexity faced by the designer is exacerbated by the computationally expensive performance evaluation of a proposed design. As technology improvements provide more transistors and chip area for design use, this complexity continues to grow. Ironically, the business climate demands increasingly shorter design cycles. The financial ramifications of improving time to market with a better price/performance design are substantial.

The genetic algorithm (**GA**) is regarded as a readily parallelized global design space search method well-suited to this problem. By casting design budget constraints as multiple design objectives, the need for



Cache Design Parameter	Minimum	Maximum	Increment
Line Size (bytes) (unit of transfer between levels of hierarchy)	4	128	to next power of 2
Cache size (bytes)	128	64K	to next power of 2
Cache associativity (degree)	1	64	to next power of 2
Cache replacement policy	Random, FIFO, LRU, NMRU, NMRR		

Replacement policies for removing old data from cache	
Random	
FIFO	First-in, first-out
LRU	Least recently used
NMRU	Not most recently used
NMRR	Not most recently replaced

The entire instruction hierarchy uses the same line size,
the entire data hierarchy uses the same line size.

The microarchitectural chromosome for this hierarchy is composed of 23 genes. Individual cache parameters are grouped together, and the hierarchy order is retained on the chromosome.

Figure 1: An On-chip Cache Hierarchy and Cache Parameters.

Caches are organized into memory hierarchies. Several design parameters affect the amount of hardware consumed by a cache, and its performance contribution. When multiple caches are being considered to build a memory hierarchy within a finite hardware constraint, the design space is too large to explore exhaustively.

penalty functions is eliminated. We present the application of a real-valued, multiobjective GA on a parallel heterogeneous workstation network. This ***G**enetic* ***A**lgorithm running on the* ***IN**ternet* is called ***GAIN***. A microprocessor on-chip memory design problem is optimized using GAIN.

2 Background

Computer architecture design has not traditionally been treated as a classic optimization problem. Much computer architecture work in the 1980s emphasized a quantitative approach to design [Hennessy and Patterson, 1990] over an "art and experience" approach. Early systematic attempts at computer architecture design pruned the search space and had difficulty distinguishing between global and local optima [Kumar and Davidson, 1980]. More recent work has partitioned the problem and used simulated annealing as a global optimizer, neglecting interaction between independently optimized hardware structures [Conte, 1992] [Conte et al., 1993]. Other researchers have applied the GA to the specific problem of selecting optimal replacement policies for on-chip cache memories [Altman et al., 1993].

Our early experiments with simple iterative hill climbing show that the computer architecture design space is multimodal, and is composed of many local optima with similar performance. Only a sustained global convergent search technique is appropriate for this problem. Lengthy objective function evaluation times demand a parallel approach. The explicit parallelism embodied in the GA population maps directly on to the coarse-grained parallelism found on a modern engineering workstation network. This suggests that the GA is particularly well-suited to this combinatorial engineering design problem.

2.1 Microprocessor Cache Memories

A predominant use of chip area in contemporary microprocessor design is for *cache* memory. A cache is a small fast memory structure holding the most recently referenced instructions or data. Its contents are managed by hardware and its presence is transparent to the programmer. Memory references exhibit a property called *locality of reference*. Data that is frequently accessed over a short period of time exhibits temporal locality; a new reference that is near the address of previously accessed data exhibits spatial locality. Because memory references reliably exhibit locality of reference, simple hardware controllers can capture the most recent references in a relatively small cache, and discard older data using a replacement policy.

Cache memories are hierarchically organized as shown in Figure 1, each caching some specialized subset of all memory accesses. Typically, the top of the hierarchy is composed of the fastest (and therefore smallest) memories, and the bottom of the hierarchy is composed of the largest (and therefore slowest) memories. The processor generates memory requests to the top level of the hierarchy. If the requested data is found at a particular level (a cache *hit*), it is provided at the speed of that hierarchy level. If the data is not found (a cache *miss*), successive levels of the hierarchy are queried until it is found and the data is provided at whatever speed the hierarchy level is capable.

Because locality of reference enables the hierarchy to statistically provide cache *hits* at the top of the hierarchy, the average memory hierarchy speed is approximated by the speed of the small fast top level. In modern microprocessors, performance is determined by the average memory access speed. [Hennessy and Jouppi, 1991]. The partition of finite chip area into a near-optimal cache hierarchy is critical.

The microprocessor memory hierarchy shown in Figure 1 and the details of its architecture are more extensively described in [Stanley and Mudge, 1995]. The 23 parameters used to describe this design space represent over 1.6×10^{19} possible configurations.

2.2 Design Constraints and Objectives

We focus our design effort on satisfying two design budget constraints, chip area and chip power dissipation, and one design objective, maximal performance.

The manufacturing cost of a microprocessor chip is directly proportional to its size, or area [Hennessy and Patterson, 1990]. As such, a design budget for this area/cost is specified early in the design. We quantify the hardware budget in *Register Bit Equivalents*, or RBEs [Mulder et al., 1991]. The RBE is a technology-independent unit-less measure of the area consumed by an on-chip memory as a function of its design parameters. Using this model, an 8K byte direct-mapped cache with a line size of 32 bytes requires 49,839 RBEs; a 64K byte direct-mapped cache with a line size of 128 bytes requires 366,156 RBEs.

The microprocessor power consumption budget is also specified in the early design stages. In contemporary VLSI CMOS technologies, the power consumption of a memory structure is a function of its size and the number of times it is referenced (which in turn is a function of its hierarchical position).

One microprocessor performance measure is *cycles per instruction* or CPI. Consider a benchmark program 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. CPI is obtained by simulating the architectural configuration, counting the number of cycles consumed and instructions executed, and computing CPI directly. At the same time, the number of accesses to each level of the cache hierarchy is counted so that the power usage can be evaluated.

A larger cache memory at any hierarchy location consumes more RBEs and power, however, it is likely to improve performance. It follows that a smaller cache memory consumes less design resources, but provides less performance.

3 Casting Constraints as Objectives

A typical approach to constrained optimization problems is the assignment of a penalty function [Powell and Skolnick, 1993] [Smith and Tate, 1993] to configurations with constraint violations. In our case, microprocessor designs exceeding the chip area or power budget constraints would be penalized.

We regard the assignment of a penalty function to over-budget configurations as undesirable. First, identifying the degree of penalization remains problematic. Second, we may have over-specified the constraints so that few satisfactory points exist in the design space. As design engineers, we remain interested in the best solution(s) found despite any over-specification. If slightly over-budget configurations provide significant performance improvements, we might change the engineering specifications. As such, we wish to avoid unconditionally discarding over-budget points. Finally, in our experience, the GA does not perform reliably when too much genetic material is discarded or adjusted due to constraint violation. To address the constraint problem, we observe that:

- the degree of constraint violation can be readily quantified,
- over-budget configurations represent possible, though less desirable, points in the design space,
- optimization techniques for engineering design problems such as ours must have the ability to simultaneously satisfy multiple objectives.

Therefore, we cast the design constraints into multiple design objectives as follows. Given:

- a design constraint $\mathcal{C}$,
- its value (our design budget) β,
- a function $f(x_1, x_2, ..., x_n)$ to compute amount the budget consumed where x_i are the design parameters,

we define a new function

$$f'(x_1, x_2, ..., x_n) = f(x_1, x_2, ..., x_n) - \beta \qquad (1)$$

to compute the degree to which the constraint is violated and a new minimizing objective function Θ to replace constraint $\mathcal{C}$

$$\Theta = \begin{cases} 0 & \text{if } f'(x_1, x_2, ..., x_n) < 0, \\ & \text{i.e., a satisfied constraint} \\ f'(x_1, x_2, ..., x_n) & \text{otherwise.} \end{cases} \tag{2}$$

Using objective Θ, configurations that are close to a constraint are regarded as better than those that are far away. Configurations that satisfy the constraint, are regarded as *equally* good. The GA's task is transformed from *penalizing* designs that *violate* the constraints to *evolving* designs that *satisfy* the constraints, using the new minimizing objective function Θ.

3.1 Multiple Objectives

Given multiple objective functions to compute chip area, performance, and power consumption, 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* [Goldberg, 1989] [Fonesca and Fleming, 1993].

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 objectives. In this problem we are *minimizing* chip area, CPI, and power. From [Goldberg, 1989], a vector of objectives $\mathbf{X}$ having length i is partially less than ($< p$) vector $\mathbf{Y}$, (i.e., it better satisfies the minimization objectives and it *dominates* $\mathbf{Y}$) when the following conditions are met:

$$(X < p\ Y) \Leftrightarrow (\forall_i)(x_i \leq y_i) \wedge (\exists_i)(x_i < y_i) \tag{3}$$

The application of Pareto optimality provides a partial ranking of competing multiobjective design points. We complete the ranking by extending Goldberg's nondominated sorting procedure as follows. First, the multiple objectives are assigned a priority order by the design engineer. The ranking procedure sorts all competing configurations into dominated and nondominated groups. The nondominated group is then further ranked by how well the first priority objective is met. If the first objective of two configurations is equal, the tie is broken by the second priority objective, and so on. These nondominated priority-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, nondominated configurations are appended to the growing ranked list until all configurations have been ranked.

3.2 The Implications

The implications of casting budget constraints as objectives whose value is the degree of constraint violation, and a priority-based Pareto-ranking extension are as follows:

1. The casting of constraints as additional minimizing objectives degenerates to the original objectives when the constraints of two competing individuals are both satisfied, for any number of constraints.
2. A slightly over budget, but otherwise nondominated, configuration will have its relative objective success properly rewarded with reproductive opportunity. Specifically, cross-breeding of over-budget configurations with at-budget configurations is permitted. As [Smith and Tate, 1993] have observed, optimal points in the design space tend to lie at the boundary of the feasible region, i.e., where a design budget is maximally consumed. We regard this extra searching of the design space on *both* sides of the region close to the constraint boundary as a desirable effect.

With this "constraint-as-objective" approach, we expand the degree to which the specification of constraints and objectives directs the design space search. For example, we can specify a power budget as a *constraint* that needs to be satisfied with a higher priority than performance. Or, we can specify a low-power *objective*, at lower priority than performance to systematically focus the search in regions where balanced CPI/power tradeoffs are found. It is the designer's specification of constraints and objectives that directs the GA to desirable regions of the design space.

4 GAIN: Parallel Network Evaluation

Considerable research on parallel GA models has been performed, e.g., [Gordon and Whitley, 1993]. However, our problem is that of simply distributing one and only one chromosome per workstation and collating results. Several interesting issues arise related to the asynchronous nature of a loosely-coupled parallel workstation network. Specifically, there is considerable variability in completion times for the evaluation function due to several factors including:

- The workstations are of different models, configurations, and therefore performance.

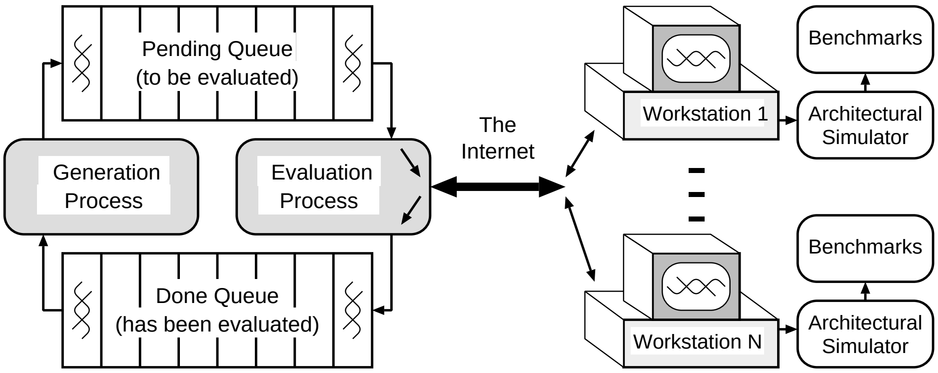


Figure 2: Decoupled Parallel GA Processes in *GAIN*.

To accommodate the asynchronous reality of the workstation network, *GAIN* is decomposed into generational and evaluation processes communicating through queues. The evaluation process controls the Internet workstation communication. Between 80 and 120 workstations have been used simultaneously per experiment to evaluate the microarchitectural objective functions.

- Proper network etiquette at the The University of Michigan requires that a remotely-submitted GA evaluation be automatically suspended when a user logs into the console of the publically shared workstation. The evaluation is restarted when the user leaves the console.
- Fatefully, the console user has also the power to terminate GA evaluations that have been submitted to that particular workstation [1].

A real-life parallel GA application must accommodate these vagaries; it is simply naive to expect unlimited predictable access to a costly distributed computing resource composed of hundreds of machines. Two extensions to the serial GA are needed - a method to handle the widely varying evaluation times, and a method to handle terminated chromosomes.

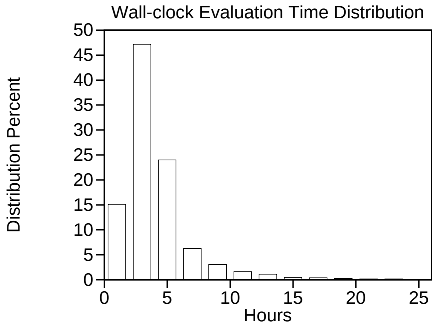
Synchronous and asynchronous solutions exist to the problem of variable objective function evaluation times [Zeigler and Kim, 1993]. A synchronous approach forces the GA to wait until all evaluations belonging to generation $\mathcal{G}$ complete before proceeding to generation $\mathcal{G}+1$. This is not practical for two reasons: first, generational synchronization *severely* curtails concurrency, and second, some chromosomes will never return from the network due to workstation console user terminations. Therefore, an asynchronous solution is chosen. The GA is allowed to proceed to generation $\mathcal{G}+1$ *before* the evaluations of generation $\mathcal{G}, \mathcal{G}-1, ..., \mathcal{G}-n$, are completed.

To accomplish this, the GA ***generation process*** is fully decoupled from the ***evaluation process*** as shown in Figure 2. These two cooperating processes communicate using queues. A new GA parameter, *Maximum Number of Pending Evaluations*, is needed by the generation process to specify the maximum number of unevaluated chromosomes that can be outstanding on the network at one time. When the number of unevaluated chromosomes exceeds *Maximum Number of Pending Evaluations*, the generation process sleeps. It wakes when the evaluation process returns evaluated chromosomes through the Done Queue.

The evaluation process communicates with individual workstations over the Internet. If an evaluation is terminated by a console user, it is resubmitted when the workstation becomes idle. After 5 failed resubmissions, GAIN assumes something is catastrophically wrong with the chromosome and moves on. To prevent deadlock from occurring between the generation and evaluation processes, the evaluation process informs the generation process when a chromosome has been terminated, so that the generation process can decrement its pending evaluation count.

The decoupled nature of GAIN and the variability in evaluation times lead to a new component of the GA's behavior that must be monitored. The *childhood* of a chromosome is defined as the number of chromosomes that have been created between its creation and its first opportunity to become a parent. This corresponds to its insertion into the Pending Queue and its removal from the Done Queue. We expect lengthy childhoods to adversely affect the stability and performance of the GA. The new parameter, *Maximum Number of Pending Evaluations*, effectively limits the childhood length as well as limiting the number of workstations simultaneously utilized by GAIN.

[1]The modern computer network is a cold, cruel, and random world with respect to survival of the fittest.



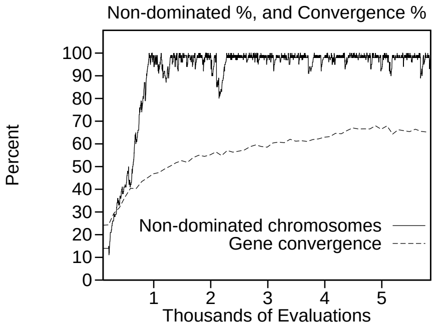


Figure 3: Evaluation Times and Population Characteristics.

The variance of the distribution of 13,429 evaluation completion times (wall clock) for the architectural objective function evaluation is large.

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

We expect that steady-state generational replacement with a small generation gap, rather than traditional full generational replacement, would stabilize the decoupled GA in the face of lengthy childhoods. This is confirmed by a limited number of experiments. For these experiments, GAIN generates two individuals per generation from a population size of 100. A final refinement of the GAIN generation process is that it does not proceed to generation $\mathcal{G}+1$ until it has retired *at least* a generation gap of chromosomes (2) from the Done Queue during generation $\mathcal{G}$.

GAIN is otherwise similar to GAs in the literature. A two-point crossover operator is used with a probability of 1.0. The probability of mutation is 0.06. Duplicate chromosomes are not allowed in the population. The weakest individuals in the population are chosen for replacement each generation.

5 Experiments and Results

At this point, we can experimentally evaluate whether GAIN's performance remains robust despite the decoupled asynchronous network complexities. The *Max Number of Pending Evaluations* and the population size were set to 100. The distribution of elapsed times for 13,429 objective function evaluations from several different experiments, running on a variety of workstation models, is shown in Figure 3. Priority workstation users terminated 132 chromosomes.

The objective priority was established as:

1. meeting a hardware budget of $(3 \times 128K)$ RBEs,
2. meeting a power factor budget of 0.07,
3. minimizing CPI.

The utility of Pareto-ranking is shown in the right graph of Figure 3. Within 670 evaluations, GAIN evolves a mostly nondominated population as defined by Equation (3). These nondominated configurations drive GAIN to explore the "Pareto frontier", i.e., areas of the design space where the multiple objectives are best satisfied simultaneously.

When 95 % of the values of a single gene (an architectural parameter) in the population have evolved to the same value, that gene has *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 3 shows the convergence history of the population's genetic material. The no-duplicate replacement policy maintains genetic diversity and convergence proceeds slowly and steadily. This graph indicates that GAIN's performance seems to have remained stable and robust after the asynchronous parallel extensions.

GAIN's progress at satisfying the chip area budget is shown in Figure 4. A total of 5855 evaluations were performed, 4831 of which were unique. Clearly, GAIN succeeds at concentrating its exploration in the area near the chip area budget specification.

As GAIN proceeds, it is unable to evolve an entire population within the power factor budget (of 0.07) as shown in Figure 5. However, some chromosomes, including the Pareto-best, satisfy the power budget. The graph showing power factor distribution frequency further indicates the low density of points in the design space that satisfy the power factor budget.

While GAIN searches regions of the design space where the design budgets are simultaneously satisfied, it also works to minimize CPI. Through time, the pop-

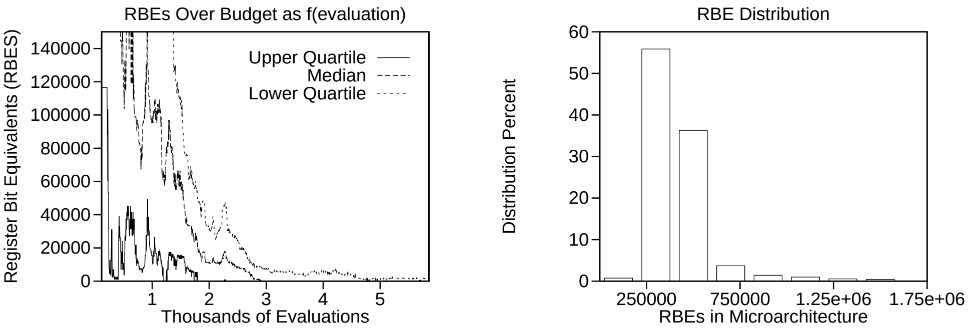


Figure 4: Constraint: Chip Area vs. Number of Evaluations and Distributions.

As GAIN optimizes the population, the number of RBEs used over the hardware budget converges to 0. As early as 1196 evaluations, the entire upper quartile of configurations satisfy the hardware budget; by 2907 evaluations, the top 2 quartiles of the population satisfy the hardware budget.

The right graph shows the distribution frequency of chip area usage for all evaluations. GAIN concentrates its search in the region where the RBE budget is specified.

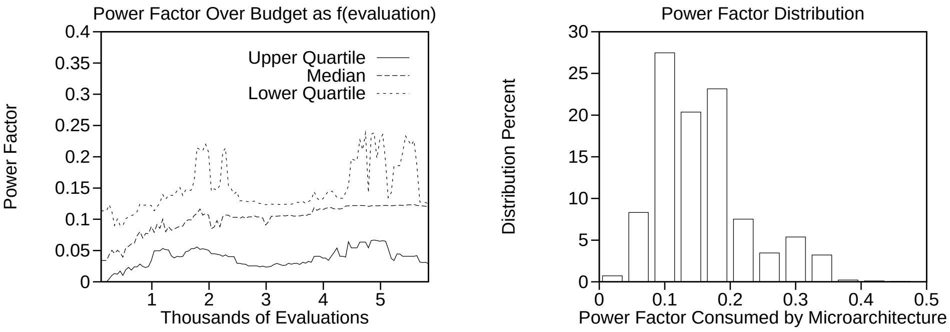


Figure 5: Constraint: Power vs. Number of Evaluations and Distributions.

As GAIN proceeds, it evaluates design points clustered near, but exceeding the power factor budget.

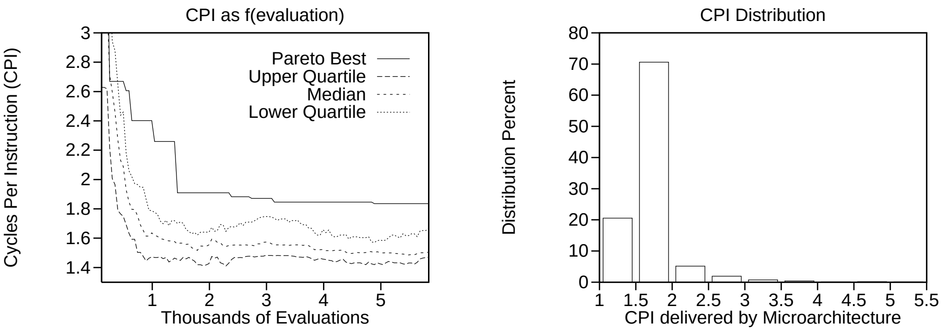


Figure 6: Objective: CPI vs. Number of Evaluations and Distributions.

While GAIN guides the architectural search to portions of the design space where hardware usage and power are at or near budget, it simultaneously optimizes for CPI using Pareto optimality. The Pareto-best configuration represents a systematic tradeoff of higher CPI, for a lower, constraint-satisfying power factor. The last Pareto-best configuration for this experiment was found at evaluation 4885.

---

ulation is increasingly composed of configurations with lower CPI as shown in Figure 6, but also with an over-budget power factor. These over-budget configurations contribute by directing the search, but are not the Pareto-best. Because GAIN does not evolve an entire population of configurations that satisfy the power factor budget, we conclude that the budget specification is very aggressive and that the density of satisfactory points in the design space is low.

The best Pareto-optimal point found however, was able to satisfy both budget constraints. Given the previously specified objective priorities, the architect can expect to achieve a performance of 1.84 CPI given a hardware budget of $(3 \times 128KRBEs)$ and a power factor budget of 0.07. For the next microprocessor design iteration, the designer would study this Pareto-optimal design point, and the top quartile of the Pareto-best configurations, to identify performance shortcomings present in near-optimal designs. The designer would use this information to propose further design improvements and repeat the optimization process until a satisfactory microprocessor design is realized.

6 Conclusions

We have presented a parallel asynchronous multiobjective genetic algorithm (called GAIN) and its application to microprocessor design optimization. Our results show that casting design constraints as multiple objectives effectively directs the search to designer-specified regions of the design space and eliminates the need for explicit penalty functions.

During the iterative, high-impact, early stages of the microprocessor design process, GAIN systematically couples the specification and optimization phases of design. From an architectural design point, we are encouraged with the results to date. Any technique to improve evaluation time, or reduce the number of evaluations required would improve the iterative design process further.

Our results show that the parallel asynchronous extensions to the GA have not adversely affected its robustness or ability to optimize our objective function. Because of the appeal of a distributed GA running on the Internet and its ability to rapidly explore complex design spaces, we plan to more thoroughly evaluate the performance of GAIN as a function of degree of workstation parallelism, generation gap, and time required to evaluate the objective function (childhood).

Acknowledgements

This research was supported by the Advanced Research Projects Agency under ARPA-AASERT Contract Number DAAH04-93-G-0249.

References

[Altman et al., 1993] Altman, E. R., Agarwal, V. K., and Gao, G. R. (1993). A novel methodology using genetic algorithms for the design of caches and cache replacement policy. In *Proc. of the* 5^{th} *Int. Conf. on Genetic Algorithms*.

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

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

[Fonesca and Fleming, 1993] Fonesca, C. M. and Fleming, P. J. (1993). Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In *Proc. of the* 5^{th} *Int. Conf. on Genetic Algorithms*.

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

[Gordon and Whitley, 1993] Gordon, V. and Whitley, D. (1993). Serial and parallel genetic algorithms as function optimizers. In *Proc. of the* 5^{th} *Int. Conf. on Genetic Algorithms*.

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

[Hennessy and Jouppi, 1991] Hennessy, J. L. and Jouppi, N. P. (1991). Computer technology and architecture: An evolving interaction. *IEEE Computer*, 24:18–29.

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

[Mulder et al., 1991] Mulder, J. M., Quach, N. T., and Flynn, M. J. (1991). An area model for on-chip memories and its application. *IEEE Journal of Solid-state Circuits*, 26(2):98–105.

[Powell and Skolnick, 1993] Powell, D. and Skolnick, M. M. (1993). Using genetic algorithms in engineering design optimization with non-linear constraints. In *Proc. of the* 5^{th} *Int. Conf. on Genetic Algorithms*.

[Smith and Tate, 1993] Smith, A. E. and Tate, D. M. (1993). Genetic optimization using a penalty function. In *Proc. of the* 5^{th} *Int. Conf. on Genetic Algorithms*.

[Stanley and Mudge, 1995] Stanley, T. J. and Mudge, T. (1995). Systematic objective-driven computer architecture optimization. In *Proc. of the* 16^{th} *Conf. on Advanced Research in VLSI*, pages 286–300.

[Zeigler and Kim, 1993] Zeigler, B. P. and Kim, J. (1993). Asynchronous genetic algorithms on parallel computers. In *Proc. of the* 5^{th} *Int. Conf. on Genetic Algorithms*.