MAELSTROM: Efficient Simulation-Based Synthesis for Custom Analog Cells

Michael Krasnicki, Rodney Phelps, Rob A. Rutenbar, L. Richard Carley
Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
{kraz, rodneyp, rutenbar, carley}@ece.cmu.edu

Abstract

Analog synthesis tools have failed to migrate into mainstream use primarily because of difficulties in reconciling the simplified models required for synthesis with the industrial-strength simulation environments required for validation. MAELSTROM is a new approach that synthesizes a circuit using the same simulation environment created to validate the circuit. We introduce a novel genetic/annealing optimizer, and leverage network parallelism to achieve efficient simulator-in-the-loop analog synthesis.

I. INTRODUCTION

Mixed-signal designs are increasing in number as a large fraction of new ICs require an interface to the external, continuous-valued world. The digital portion of these designs can be attacked with modern cell-based tools for synthesis, mapping, and physical design. The analog portion, however, is still routinely designed by hand. Although it is typically a small fraction of the overall design size (*e.g.*, 10,000 to 20,000 analog transistors), the analog partition in these designs is often the bottleneck because of the lack of automation tools.

The situation appears to be worsening as we head into the era of System-on-Chip (SoC) designs. To manage complexity and time-to-market, SoC designs require a high level of reuse, and cell-based techniques lend themselves well to a variety of strategies for capturing and reusing digital intellectual property (IP). But these digital strategies are inapplicable to analog designs, which rely for basic functionality on tight control of low-level device and circuit properties that vary from technology to technology. The analog portions of these systems are still designed by hand today. They are even routinely ported by hand as a given IC migrates from one fabrication process to another.

A significant amount of research has been devoted to cell-level analog synthesis, which we define as the task of sizing and biasing a device-level circuit with 10 to 50 devices. However, as noted in [1], previous approaches have failed to make the transition from research to practice. This is due primarily to the prohibitive effort needed to reconcile the simplified circuit models needed for synthesis with the "industrial-strength" models needed for validation in a production environment. In digital design, the bit-level, gate-level and block-level abstractions used in synthesis are faithful to the corresponding models used for simulation-based validation. This is not the case for analog synthesis.

Fig. 1 illustrates the basic architecture of most analog synthesis tools. An *optimization engine* visits candidate circuit designs and adjusts their parameters in an attempt to satisfy designer-specified performance goals. An *evaluation engine* quantifies the quality of each circuit candidate for the optimizer. Most research here focuses on trade-offs between the optimizer (which wants to visit many circuit candidates) and the evaluator (which must itself trade accuracy for speed to allow sufficiently vigorous search). Much of this work is really an attempt to evade a harsh truth--that analog circuits are difficult and time-consuming to evaluate properly. Even a small cell requires a mix of ac, dc and transient analyses to correctly validate. In modern design environments, there is enormous investment in simulators, device models, process characterization, and "cell sign-off" validation methodologies. Indeed, even the sequence of circuit analyses, models, and simulation test-jigs is treated as valuable IP. Given these facts, it is perhaps no surprise that analog synthesis strategies that rely on exotic, nonstandard, or fast-but-incomplete evaluation engines have fared poorly in real design environments. To trust a synthesis result, one must first trust the methods used to quantify the circuit's performance *during* synthesis. Most prior work fails here.

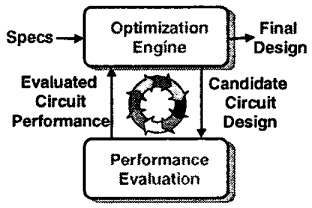

Fig. 1 Abstract Model of Analog Synthesis Tools.

Given the complexity of, investment in, and reliance on simulator-centric validation approaches for analog cells, we argue that for a synthesis strategy to have practical impact, it *must* use a simulator-based evaluation engine that is *identical* to that used to validate ordinary manual designs. This, however, poses significant challenges. For example, commercial circuit simulators are not designed to be invoked 50,000 times in the inner loop of a numerical optimizer. And, of course, the CPU time to visit and simulate this many solution candidates may be unacceptable.

In this paper we develop a new strategy to support efficient simulator-in-the-loop analog synthesis. The approach relies on three key ideas. First, we *encapsulate* commercial simulators so that their implementation idiosyncrasies are hidden from our search engine. Second, we use a novel combined *genetic/annealing optimization algorithm* that is robust in finding workable circuits, and avoids the starting-point dependency problems of gradient and other down-hill search methods. Third, we exploit *network-level workstation parallelism* to render the overall computation times tractable. Our new optimization algorithm was designed to support transparent distribution of *both* the search tasks and the circuit evaluation tasks across a network.

We have implemented these ideas in a tool called MAELSTROM. MAELSTROM has been successfully run on networks of 10 to 30 SUN or IBM UNIX workstations, and currently runs Cadence Design System's Spectre simulator [2] as its evaluation engine. In this paper we describe the basic algorithms underlying MAELSTROM, and present a set of experimental synthesis results that suggest that simulator-in-the-loop synthesis can be made both practical and efficient. The remainder of the paper is organized as follows. Section II briefly re-

DAC 99, New Orleans, Louisiana

views prior work. Section III gives a complete formulation of the synthesis problem. Section IV offers experimental results on circuits. Finally, Section V offers some concluding remarks.

II. Review Of Prior Approaches

Referring again to Fig. 1, we can broadly categorize previous work on analog synthesis by how it searches for solutions and how it evaluates each visited circuit candidate. See [3] for a more extensive survey.

Early work on synthesis used simple procedural techniques [4], rendering circuits as explicit scripts of equations whose direct evaluation completed a design. Although fast, these techniques proved to be difficult to update, and rather inaccurate. Numerical search has been used with equation-based evaluators [5], [6], [7], and even combinatorial search over different circuit topologies [8],[9], but equation-based approaches remain brittle in the face of technology changes. Hierarchical systems [10], [11], [12], [13] introduced compositional techniques to assemble equation-based subcircuits, but still faced the same update/accuracy difficulties. Some of these systems can manipulate circuit equations automatically to suit different steps of the synthesis task [6]. Qualitative and fuzzy reasoning techniques [14], [15] have been tried to capture designer expertise, but with limited success. Equation-based synthesis offers fast circuit evaluation, and is thus well suited to aggressive search over solution candidates. However, it is often prohibitively expensive to create these models--indeed, often more expensive than manually designing the circuit. Also, the simplifications required in these closed-form analytical circuit models necessarily limit their accuracy and completeness.

Symbolic analysis techniques, which have made significant strides of late[16],[17],[18],[7] offer an automated path to obtaining some of these design equations. These techniques automatically derive reduced-order symbolic models of the linear transfer function of a circuit. The resulting symbolic forms can be obtained fairly quickly, offer good accuracy, and can thus serve as evaluation engines, *e.g.*, [6]. However, they are strictly limited to linear performance specifications. Even a small analog cell may require a wide portfolio of dc, ac, and transient simulations to validate it. Symbolic analysis is a valuable but incomplete approach to circuit evaluation.

The synthesis systems most relevant to the ideas we develop in this paper are ASTRX/OBLX [1],[3] and the system from Seville [19]. In ASTRX/OBLX, we attacked the fundamental problem of tool usability with a compile-and-solve methodology. ASTRX starts from a SPICE deck describing an unsized circuit and desired performance specifications. ASTRX compiles this deck into a custom C program that implements a numerical cost function whose minimum corresponds to a good circuit solution for these constraints. OBLX uses simulated annealing [20] to solve this function for a minimum. This custom-generated cost code evaluates circuit performance via model-order reduction [21] for linear, small-signal analysis, and user-supplied equations for nonlinear specifications. ASTRX/OBLX was able to synthesize a wide variety of cells, but was still limited to essentially linear performance specifications. [19] similarly uses annealing for search, but actually runs a SPICE-class simulator in its annealer. However, this tool appears to employ a simulator customized for synthesis, only evaluates a few thousand circuit candidates in a typical synthesis run (in contrast, OBLX evaluates 10^4 to 10^5 solutions), and has only been demonstrated attacking problems with a small number of independent design variables.

Finally, we also note that there are several circuit *optimization* attacks that rely on simulator-based methods (*e.g.*, [22]). For circuit optimization we assume a good initial circuit solution, and seek to improve it. This can be accomplished with gradient and sensitivity techniques requiring a modest number of circuit evaluations. In contrast, in circuit *synthesis* we can assume nothing about our starting circuit (indeed, we usually have *no* initial solution). This scenario is much more difficult as a numerical problem, and requires a global search strategy to avoid being trapped in poor local minima that happen to lie near the starting point.

The problem with all these synthesis approaches is that they use circuit evaluation engines different from the simulators and simulation strategies that designers actually use to validate their circuits. These engines trade off accuracy and completeness of evaluation for speed. We argue that this is no longer an acceptable trade-off.

III. Synthesis Formulation

In this section, we present the full synthesis formulation of MAELSTROM. Our circuit synthesis strategy relies on three key ideas: simulator encapsulation, a novel combined genetic/annealing global optimizer, and scalable network parallelism. We describe these ideas below, beginning with a review of our basic synthesis-via-optimization formulation.

A. Basic Optimization Formulation

We use the basic synthesis formulation from OBLX [1], which we review here. We begin with a fixed circuit topology that we seek to size and bias. We approach circuit synthesis using a constrained optimization formulation, but solve it in an unconstrained fashion. We map the circuit design problem to the constrained optimization problem of (1), where x is the set of independent variables—geometries of semiconductor devices or values of passive circuit components—we wish to change to determine circuit performance; $f(x)$ is a set of objective functions that codify performance specifications the designer wishes to optimize, *e.g.* power or bandwidth; and $g(x)$ is a set of constraint functions that codify specifications that must be beyond a specific goal, *e.g.*, (gain > 60dB). Scalar weights, w_i, balance competing objectives.

$$\underset{x}{\text{minimize}} \sum_{i=1}^{k} w_i \cdot f_i(x) \quad \text{s.t.} \quad g(x) \le 0 \tag{1}$$

Formulation of the individual objective $f(x)$ and constraint $g(x)$ functions adapts ideas from [22]. The user is expected to provide a *good* value, and a *bad* value for each specification. These are used both to set constraint boundaries and to normalize the specification's range. For example, a single objective $f_i(x)$ is internally normalized as:

$$\hat{f}_i(x) = \frac{f_i(x) - good_i}{bad_i - good_i} \tag{2}$$

This normalization process provides a natural way for the designer to set the relative importance of competing specifications, and it provides a straightforward way to normalize the range of values that must be balanced in the cost function.

To support the genetic/annealing optimizer we shall introduce in Section IIIC, we perform the standard conversion of this constrained optimization problem to an unconstrained optimization problem with the use of additional scalar weights. As a result, the goal becomes minimization of a scalar cost function, $C(x)$, defined by (3).

$$C(x) = \sum_{i=1}^{k} w_i \hat{f}_i(x) + \sum_{j=1}^{l} w_j \hat{g}_j(x) \tag{3}$$

The key to this formulation is that the minimum of $C(x)$ corresponds to the circuit design that best matches the given specifications. Thus, the synthesis task becomes two more concrete tasks: evaluating $C(x)$ and searching for its minimum. Neither of these are simple. Our major contributions in this paper are an algorithm for global search that is efficient enough to allow use of commercial circuit simulators to evaluate $C(x)$, and a methodology for encapsulating simulators to hide unnecessary details from this search process. We treat the encapsulation methodology next.

B. Simulator Encapsulation for Simulation-Based Evaluation

Our overall goal is to be able to the use the simulation methods trusted by designers--but *during* analog cell synthesis. This means invoking a sequence of detailed circuit simulations for each evaluation of $C(x)$ during numerical search. Although different SPICE-class simulation engines share core mechanisms and offer similar input/output formats, they remain highly idiosyncratic in many features. In our experience, the mechanics of embedding a simulator inside a numerical optimizer are remarkably untidy. This is a real problem since we seek a strict separation of the circuit optimization and circuit evaluation engines, and would like ultimately to be able to "plug in" different simulators. We handle this problem using a technique we refer to as *simulator encapsulation.*

Simulator encapsulation hides the details of a particular simulator behind an insulating layer of software. This software "wrapper" renders the simulator an object with a set of methods, similar to standard object-oriented programming ideas. The simulator appears to the optimization engine as an object with methods to invoke a simulation, to change circuit parameters, to retrieve simulation results as a simple vector of numbers, and so forth. Clearly one major function of this encapsulation is to hide varying data formats from the optimizer; this engine need not concern itself with the details of how to invoke or interpret an ac, dc, or transient analysis in the simulator.

A more subtle function of encapsulation is to insulate the optimization engine from "unfriendly" behavior in the simulator. Most simulators are designed either for batch-oriented operation, or for interactive schematic-update-then-simulate operation. In the latter, the time scales are optimized for humans--overheads of a few seconds per simulation invocation are negligible. But inside a numerical optimizer that seeks to run perhaps 50,000 simulations, these overheads are magnified. Our ideal is a simulator which can be invoked once, and, remaining live, can interpret quickly a stream of requests to modify circuit values and resimulate. Few simulators approach this ideal. For example, some insist on rechecking a licence manager key (possibly located remotely on a network) for every new simulation request; others flush all internal state or drop myriad temporary files in the local file system. Of course, the maximally difficult behavior exhibited by a simulator is a crash, an occurrence far from rare even in commercial offerings. This is especially problematic in synthesis, since the optimization engine may often visit circuit candidates with highly non-physical parameter values, which occasionally cause simulator failure. Our encapsulation not only detects the crash but also restarts and reinitializes the simulator, all transparent to the optimizer. All these difficult behaviors can be hidden via appropriate encapsulation.

C. Combined Genetic/Annealing Optimization: PRSA

As in OBLX [1], we again favor global, stochastic search algorithms for the optimization engine because of their empirical robustness in the face of highly nonlinear, nonconvex cost functions. However, in OBLX we made an explicit trade-off to use a customized, highly tuned, very fast circuit evaluator to permit search over a large number of solution candidates. When we replace this custom evaluator with commercial circuit simulation, we are faced with a 10X to 100X increase in CPU time. The central question we address in this section is how to retain the virtues of global, stochastic search, but deal with the runtime implications of simulator-in-the-loop optimization.

Before we describe our new optimizer, it is worth justifying our choice of stochastic optimization. Given a good implementation of simulator encapsulation, we can replace the custom circuit evaluation used in OBLX with full, detailed simulation. We have rewritten the core annealing engine of OBLX in the form of a new, component-based optimization library called ANNEAL++ [23]. ANNEAL++ offers a range of annealing cooling schedules, move selection techniques, and dynamic updates on cost function weights, based on the ideas in [3]. As an experiment, we encapsulated the Cadence Spectre circuit simulator and used it with ANNEAL++ to resynthesize the custom folded-cascode opamp from [24]. The circuit has 32 devices and 27 designable variables; the circuit appears in Fig. 2, results appear in Table 1.

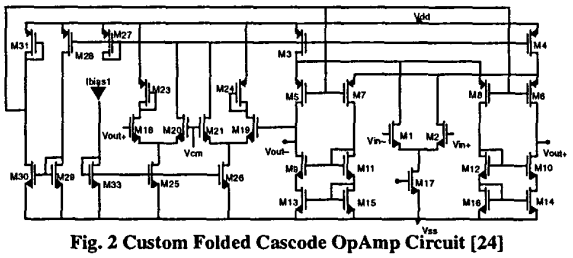

Fig. 2 Custom Folded Cascode OpAmp Circuit [24]

Table 1. Simple Synthesis Result for Circuit of Fig. 2, on a 55MHz IBM Power2

Attribute	Manual Design	Auto-Synthesis: Spec	Result
CLoad (pF)	1.25	1.25	
Vdd (V)	5	5	
DC Gain (dB)	71.2	≥71:	91
UGF (MHz)	47.8	≥48:	55
Phase Margin (deg)	77.4	≥77:	83
PSRR - Vss (dB)	92.6	≥93:	119
PSRR - Vdd (dB)	72.3	≥72:	92
Output Swing (V)	±1.4	±1.4:	±1.4
Settling Time (ns)	-	↓[a]:	47
Active Area ($10^3\mu^2$)	68.7	↓:	28
Circuits Evaluated			17,100
CPU (hours)			11

a. ↑ means maximize, while ↓ means minimize.

This rather straightforward synthesis strategy yields a surprisingly reasonable result, albeit somewhat slowly. Fig. 3 shows a set of sampled cross-sections from the cost-surface for this annealing-style synthesis formulation. At an intermediate point in the synthesis, we stopped the optimizer, and then iteratively stepped each independent variable over its range, while freezing all other variables. At each step point we evaluated the synthesis cost function using Spectre. Fig. 3 shows a few of these resulting cross-sections, suitably normalized for comparison. The mix of gently sloping plateaus and jagged obstacles is typical of these landscapes. Annealing style algorithms are a good choice here because of their hill-climbing abilities.

However, annealing algorithms have a reputation for slow execution because of the large number of solution candidates that must be visited. This is greatly exacerbated when we choose to fully simulate

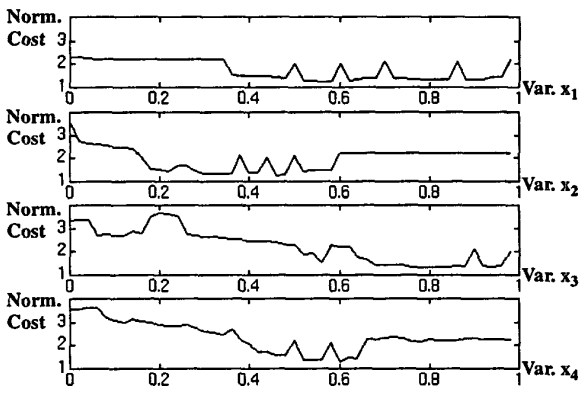

Fig. 3 Four 1-dimensional normalized cross-sections of the cost-surface for a typical simulation-based synthesis problem

each solution candidate. There are three broad avenues of solution here:

1. **Less search:** attempt to sample the cost function at fewer points. This is essentially the approach taken by [19], which uses an unusual, truncated annealing schedule with some of the character of a random multistart approach. However, in our experience, wider search always yields better solutions and a more robust tool.

2. **Parallel circuit evaluation:** each visited circuit candidate usually requires more than one circuit simulation to evaluate it. We can easily distribute these over a network to parallel workstations. Indeed, our implementation supports this simple parallelism. For example, if we resynthesize the opamp of Fig. 2, but distribute the 5 simulations required to evaluate each circuit across 3 IBM workstations, the 11 hour sequential time drops to 192 minutes. This is a useful form of parallelism to exploit, but it is strictly limited.

3. **Parallel circuit search:** what we really seek is a technique to allow multiple, concurrent points of the cost landscape to be searched in parallel, but synchronized in some manner that guarantees convergence to a final circuit or set of circuits of similar quality.

Unfortunately, annealing *per se* does not easily support parallel search. An annealing-based optimizer generates a serial stream of proposed circuit perturbations, and relies on statistics from previous circuits to adjust its control parameters. To parallelize search itself, an obvious set of methods to consider here are the genetic algorithms [25], whose population-based evolution models distribute over parallel machines more naturally. However, we do not wish to abandon the direct hill-climbing of annealing, which has empirically performed well in this task. Goldberg [26] suggests a solution here: *parallel recombinative simulated annealing* (PRSA)

PRSA, which has its roots in genetic algorithms, can be regarded as a strategy for synchronizing a population of annealers as they cooperatively search a cost surface. The idea is conceptually simple. Suppose in a serial annealer we would expect to visit 10,000 circuit candidates. To distribute this over 10 CPUs, we begin by creating 10 separate *PRSA-nodes*, each of which simply runs a standard annealing optimization (ANNEAL++ in our case) but with a schedule truncated to 10,000/10=1000 visited circuits. Obviously, the solution found by each of these 10 independent nodes will be very poor. To synchronize these nodes, we regard each annealer itself as one element of a larger population of evolving solutions, and allow annealers to exchange results among themselves. Thus, after generating a new candidate circuit solution, each annealer randomly communicates its result to a subset of the other PRSA-nodes. Each PRSA-node maintains a queue for these shared results, which represent samples of the cost surface visited by *other* annealers in the population. When generating a new circuit candidate, each annealer makes one of two choices:

1. **Perturbation:** the annealer can simply select its previously generated solution and *perturb* its element values. This is the traditional mechanism by which an annealer evolves a solution.

For all parallel PRSA nodes : P_i, (i = 1 to n)
(A) Set annealer temperature T = hot
(B) Generate random initial circuit solution $\underline{x}_{Pi}$.
(C) Repeat until equilibrium:
(C1) Send current circuit solution $\underline{x}_{Pi}$ to other randomly selected PRSA node
(C2) Receive migrants from other PRSA nodes
(C3) Apply perturbation or crossover to generate $\underline{x}_{Pi}^{new}$ from $\underline{x}_{Pi}$
(C4) Evaluate $\underline{x}_{Pi}^{new}$
(C5) $\Delta C = Cost(\underline{x}_{Pi}^{new}) - Cost(\underline{x}_{Pi})$
(C6) If $\Delta C < 0$
Replace $\underline{x}_{Pi}$ with $\underline{x}_{Pi}^{new}$ with probability 1.
(C7) Else
Replace $\underline{x}_{Pi}$ with $\underline{x}_{Pi}^{new}$ with probability $e^{-[(\Delta C)/T]}$
(D) If not frozen, lower T, goto (C)

Fig. 4 Pseudo-code for optimization in one PRSA-node.

2. **Recombination:** the annealer can *recombine* its previously generated solution with the solution on the top of its queue. This is the *crossover* (mating) operation from genetic algorithms, which randomly combines the features of two *parent* solutions into a single, new *offspring* solution.

Because circuit solution candidates are simply vectors of real numbers for us (e.g., MOSFET lengths and widths), crossover is simple to implement. We use a so-called *single-point* crossover scheme. Given two parent solutions $\underline{x} = [x_1, x_2, \ldots, x_n]$ and $\underline{y} = [y_1, y_2, \ldots, y_n]$, we combine by randomly selecting $r \in [1,n]$ and generate the offspring:

$$\underline{s} = [x_1, x_2, \ldots, x_r, y_{r+1}, y_{r+2}, \ldots, y_n] \quad (4)$$

Pseudo-code for the algorithm in each PRSA-node appears in Fig. 4.

In practice, we find that PRSA works extremely well to synchronize parallel annealers. In particular, good solutions found by one node quickly diffuse through the population, and drive annealers stuck in unpromising local minima toward better global solutions. Fig. 5. illustrates this synchronization effect by plotting the annealing cost value as a function of circuits visited in each of 10 parallel PRSA nodes during a sample circuit synthesis. Each PRSA-node visits roughly 2000 circuit candidates; the population of annealers visits 20,000, each evaluated via Spectre simulation. The curves demonstrate empirically how each annealing process is coordinated into searching for circuits of similar cost at similar times in the run.

Finally, we note that parallel circuit evaluation and parallel PRSA search are othogonal: we can do both. Each PRSA node can manage a set of independent evaluation nodes to perform the multiple simula-

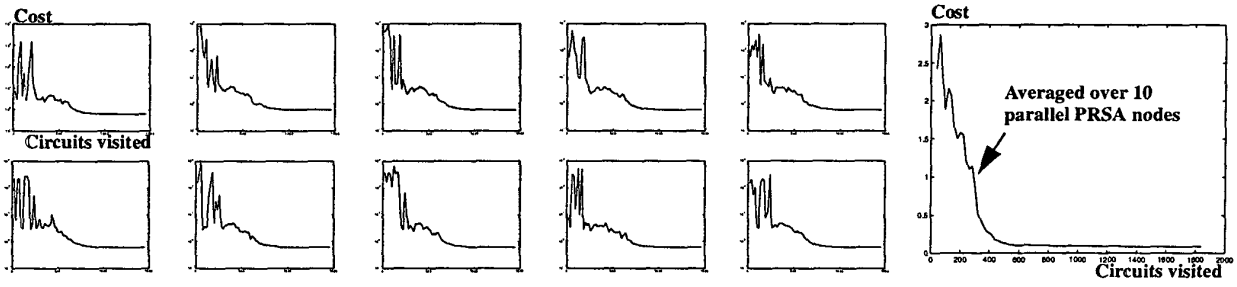

Fig. 5 Synchronized search behavior, cost versus circuits visited, for 10 parallel PRSA nodes.

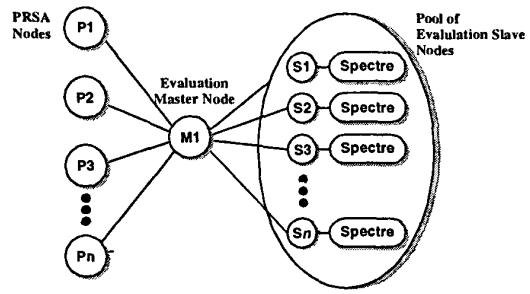

Fig. 6 Network architecture for MAELSTROM using DISTRIBUTEDPRSA

tions necessary to evaluate each solution candidate. We discuss this in the next section. We believe the capability to distribute both circuit evaluations and the optimization process itself is a significant contribution of this work.

D. Network Architecture: Distributed Search and Evaluation

Our implementation distributes all computation over a pool of workstations. At the lowest level, we manage concurrency and interprocessor communication using the publicly available PVM library [27]. We have implemented on top of this a general framework for optimization called DISTRIBUTEDPRSA. Fig. 6 shows a topological overview of DISTRIBUTEDPRSA. This library coordinates the interaction of the three concurrent tasks that comprise our synthesis tool:

1. **PRSA Node:** We use ANNEAL++ to implement a PRSA computational node, as discussed in the previous section. The DISTRIBUTEDPRSA library implements a mechanism that allows each PRSA node to send its current solution to another randomly selected PRSA node for use in crossover. In turn, each PRSA node keeps a small FIFO queue of recently received circuit solution candidates. This transfer of state information is a peer-to-peer transaction between the PRSA nodes and does not involve the evaluation master.

2. **Evaluation Master:** Each *evaluation master* schedules evaluation requests from some number of PRSA nodes across a pool of evaluation slaves. The cost calculation for each candidate circuit solution may require several Spectre simulation analyses. Each of these analyses can be performed in parallel on different machines. Thus, each evaluation master has one or more slaves for each analysis type. Currently, evaluation slaves are assigned to machines statically, based upon a configuration file. In the future, the evaluation master will dynamically reassign evaluation slaves across a pool of available workstations. The goal of this mechanism is to dynamically detect available processor time and to utilize it to expedite the synthesis process.

3. **Evaluation Slave:** An *evaluation slave* uses the simulator encapsulation library to perform one or more simulation analyses, *i.e.*, the slaves actually invoke the necessary circuit simulation tasks, with the encapsulation library serving as the interface to the simulator. If there are insufficient machines, one machine can be used to run multiple evaluation slaves.

IV. EXPERIMENTAL RESULTS

We have implemented these ideas in a tool called MAELSTROM, which currently runs on networks of SUN Solaris and IBM AIX nodes. In this section we present three results to demonstrate both the feasibility and efficiency of our synthesis strategy.

A. Custom Opamp Circuit

We have resynthesized the custom opamp [24] shown originally in Fig. 2, but now using the fully distributed version of MAELSTROM. Table 2 shows the desired specifications and the final synthesis results obtained with our tool. The optimization task had 27 independent variables that specified all device dimensions, capacitor sizes, and

Table 2. MAELSTROM Synthesis Result for Custom Opamp Circuit of Fig. 2

Attribute	Manual Design	Auto-Synthesis: Spec.	Result
CLoad (pF)	1.25	1.25	
Vdd (V)	5	5	
DC Gain (dB)	71.2	≥71:	110
UGF (MHz)	47.8	≥48:	70
Phase Margin (deg)	77.4	≥77:	84
PSRR - Vss (dB)	92.6	≥93:	131
PSRR - Vdd (dB)	72.3	≥72:	108
Output Swing (V)	± 1.4	± 1.4:	± 1.45
Settling Time (ns)	-	↓:	29
Active Area ($10^3\mu^2$)	68.7	↓:	23
Circuits Evaluated		70,000	
CPU Time (minutes)		219	

bias currents. Each of the variables had a broad (yet reasonable) range: all variables had a design range of at least one order of magnitude, many have ranges of two orders of magnitude. The process is 1.2μm CMOS. Note not only that we meet all specifications, but this result is significantly better than the earlier sequential synthesis shown in Table 1. The improved runtime is due to the large-scale parallelism; the improved solution is a result of allowing more search. The run in Table 1 searched only 17,000 circuits, we allowed this run to search 70,000 circuits.

This result was obtained in 219 minutes across 15 140Mhz SUN Ultra-1 workstations. The run consisted of 10 PRSA nodes, 1 evaluation master, and 15 evaluation slaves. (Note that physical CPUs actually share search, control, and evaluation tasks concurrently.) Each PRSA node examined approximately 7000 candidate solutions across the duration of the run. Evaluating each candidate solution required 5 separate Spectre circuit simulations.

B. Basic Folded Cascode Op-amp

Fig. 7 shows a basic fully differential folded cascode circuit, again to be sized in a 1.2μm CMOS process. This is illustrative of the sort

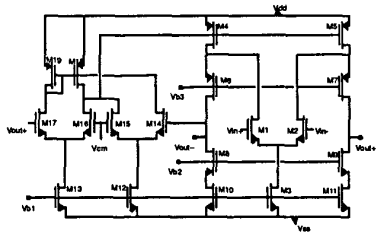

Fig. 7 Basic Folded Cascode Circuit

Table 3. MAELSTROM Result for Basic Folded Cascode Opamp Circuit in Fig. 7

Attribute	Auto-Synthesis: Spec.	Result
CLoad (pF)	1	
Vdd (V)	5	
DC Gain (dB)	≥70:	71.4
UGF (MHz)	≥10:	24.3
Phase Margin (deg)	≥60:	69
PSRR - Vss (dB)	≥40:	111
PSRR - Vdd (dB)	≥40:	132
Output Swing (V)	± 1.35:	± 1.37
Settling Time (ns)	≤100 :	50
Active Area ($10^3\mu^2$)	≤68 :	11
Circuits Evaluated	60,000	
CPU (minutes)	152	

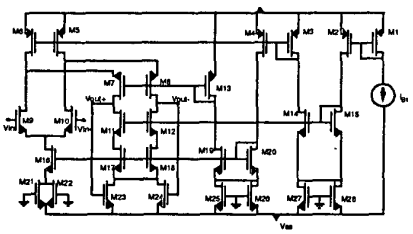

Fig. 8 Seville Benchmark Circuit

Table 4. MAELSTROM Result for Seville Benchmark Circuit of Fig. 8

Attribute	Auto-Synthesis: Spec.	Result
CLoad (pF)	1	
Vdd (V)	5	
DC Gain (dB)	≥ 70:	70
UGF (MHz)	≥30:	47
Phase Margin (deg)	≥ 60:	60
PSRR - Vss (dB)	≥ 40:	71
PSRR - Vdd (dB)	≥ 40:	94
Output Swing (V)	± 1.5:	± 1.5
Settling Time (ns)	≤ 80 :	68
Static Power (mW)	≤ 2.1 :	1
Active Area ($10^3\mu^2$)	≤ 68 :	38
Circuits Evaluated		70,000
CPU (minutes)		190

of routine redesign problems faced when common analog blocks are retargeted to new applications. Table 3 shows the desired specifications and the final synthesis result. This optimization task had 21 independent design variables and was again run on 30 Ultra-1 workstations with the same PRSA configuration.

C. Seville Benchmark Circuit

Fig. 8 shows the opamp benchmark circuit used in [19]. We have synthesized this result to the specifications from [19] in a 1.2μm process. (The specification for slew rate to exceed 70 V/μs was translated to a constraint of settling time below 80ns). This optimization task had 22 independent design variables, in contrast to the formulation in [19] which had 10. This represents the trade-off between up front manual design (to determine a subset of critical designable devices) versus simply allowing the optimization tool to search a larger solution space. The circuit meets all its specifications, and is comparable to the results from [19]. This synthesis was run on 18 SUN Ultra-1 workstations.

V. CONCLUSIONS

We described a new cell-level analog synthesis strategy that evaluated each proposed solution candidate using the same simulation methods relied on by designers to validate manual circuit designs. Our approach relies on three key ideas: simulator encapsulation to hide low-level details of specific simulators; a combined genetic/annealing algorithm for robust global search of the solution space; and network parallelism to render execution times short enough to make synthesis practical. MAELSTROM, a preliminary implementation of these ideas, has been run successfully on networks of up to 30 UNIX workstations, and can explore 10^4 to 10^5 circuit candidates in a few hours. Preliminary results suggest the approach is workable for many of the routine, cell-level, nominal sizing/biasing tasks that analog designer currently perform by hand.

Our current work focuses on tuning to support usage modes where designers seek only a "quick" approximate solution to explore the feasibility of a particular circuit topology, and support for evaluation across manufacturing corners [28].

Acknowledgment: This work was funded by the Semiconductor Research Corp. and by Rockwell and Texas Instruments. We thank Dave Guilleau of CMU, Emil Ochotta of Xilinx, Chris Wolff of Rockwell, Gary Richey of TI, and Al Dunlop and Mean-Sea Tsay of Lucent for valuable discussions about this work.

REFERENCES

[1] E. Ochotta, R.A. Rutenbar, L.R. Carley, "Synthesis of High-Performance Analog Circuits and ASTRX/OBLX," *IEEE Trans. CAD*, vol. 15, no. 3, March 1996.

[2] K.S. Kundert, *The Designer's Guide to SPICE & SPECTRE*, Kluwer Academic Publishers, Kluwer Academic Publishers, 1995.

[3] E. Ochotta, T. Mukherjee, R.A. Rutenbar, L.R. Carley, *Practical Synthesis of High-Performance Analog Circuits*, Kluwer Academic Publishers, 1998.

[4] M. Degrauwe *et al.*, "Towards an analog system design environment," *IEEE JSSC*, vol. sc-24, no. 3, June 1989.

[5] H.Y. Koh, C.H. Sequin, and P.R. Gray, "OPASYN: a compiler for MOS operational amplifiers," *IEEE Trans. CAD*, vol. 9, no. 2, Feb. 1990.

[6] G. Gielen, *et al.*, "Analog circuit design optimization based on symbolic simulation and simulated annealing," *IEEE JSSC*, vol. 25, June 1990.

[7] F. Leyn, W. Daems, G. Gielen, W. Sansen, "A Behavioral Signal Path Modeling Methodology for Qualitative Insight in and Efficient Sizing of CMOS Opamps," *Proc. ACM/IEEE ICCAD*, 1997.

[8] P. C. Maulik, L. R. Carley, and R. A. Rutenbar, "Integer Programming Based Topology Selection of Cell Level Analog Circuits," *IEEE Trans. CAD*, vol. 14, no. 4, April 1995.

[9] W. Kruiskamp and D. Leenaerts, "DARWIN: CMOS Opamp Synthesis by Means of a Genetic Algorithm," *Proc. 32nd ACM/IEEE DAC*, 1995.

[10] R. Harjani, R.A. Rutenbar and L.R. Carley, "OASYS: a framework for analog circuit synthesis," *IEEE Trans. CAD*, vol. 8, no. 12, Dec. 1989.

[11] B.J. Sheu, *et al.*, "A Knowledge-Based Approach to Analog IC Design," *IEEE Trans. Circuits and Systems*, CAS-35(2):256-258, 1988.

[12] E. Berkcan, *et al.*, "Analog Compilation Based on Successive Decompositions," *Proc. of the 25th IEEE DAC*, pp. 369-375, 1988.

[13] J. P. Harvey, *et al.*, "STAIC: An Interactive Framework for Synthesizing CMOS and BiCMOS Analog Circuits," *IEEE Trans. CAD*, Nov. 1992.

[14] C. Makris and C. Toumazou, "Analog IC Design Automation Part II--Automated CIrcuit Correction by Qualitative Reasoning," *IEEE Trans. CAD*, vol. 14, no. 2, Feb. 1995.

[15] A. Torralba, J. Chavez and L. Franquelo, "FASY: A Fuzzy-Logic Based Tool for Analog Synthesis," *IEEE Trans. CAD*, vol. 15, no. 7, July 996.

[16] G. Gielen, P. Wambacq, and W. Sansen, "Symbolic ANalysis Methods and Applications for Analog Circuits: A Tutorial Overview, " Proc. IEEE, vol. 82, no. 2, Feb., 1990.

[17] C.J. Shi, X. Tan, "Symbolic Analysis of Large Analog Circuits with Determinant Decision Diagrams," *Proc. ACM/IEEE ICCAD*, 1997.

[18] Q. Yu and C. Sechen, "A Unified Approach to the Approximate Symbolic Analysis of Large Analog Integrated Circuits," *IEEE Trans. Circuits and Sys.*, vol. 43, no. 8, August 1996.

[19] F. Medeiro, F.V. Fernandez, R. Dominguez-Castro and A. Rodriguez-Vasquez, " A Statistical Optimization Based Approach for Automated Sizing of Analog Cells," *Proc. ACM/IEEE ICCAD*, 1994.

[20] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, 13 May 183.

[21] L. T. Pillage and R.A. Rohrer, "Asymptotic Waveform Evaluation for Timing Analysis," IEEE Trans. CAD, vol. 9. no. 4, April 1990.

[22] W. Nye, *et al.*, "DELIGHT.SPICE: an optimization-based system for the design of integrated circuits," *IEEE Trans. CAD*, vol. 7, April 1988.

[23] M. Krasnicki, "Generalized Analog Circuit Synthesis," M.S. Thesis, Dept. of ECE, Carnegie Mellon, Dec. 1997.

[24] K. Nakamura and L.R. Carley, "A current-based positive-feedback technique for efficient cascode bootstrapping," *Proc. VLSI Circuits Symposium*, June 1991.

[25] J.H. Holland. *Adaptation in Nature and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[26] S. W. Mahfoud and D.E. Goldberg, "Parallel Recombinative Simulated Annealing: A Genetic Algorithm," *Parallel Computing*, vol. 21, 1995.

[27] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam. *PVM: Parallel Virtual Machine A User's Guide and Tutorial for Network Parallel Computing*. MIT Press, 1994.

[28] T. Mukherjee, L.R. Carley, R.A. Rutenbar, "Synthesis of Manufacturable Analog Circuits," *Proc. ACM/IEEE ICCAD*, 1994.