Distributed Adaptive Simulated Annealing for Synthesis Design Space Exploration

Sumit Gupta
Lubomir Bic

Technical Report #99-05
Department of Information and Computer Science
University of California, Irvine, CA 92697-3425

January 1999

Abstract

This work has attempted to exploit information sharing to improve the results of *Adaptive Simulated Annealing* [1] as an optimization algorithm of the high-level synthesis of testable data paths. We have used *Messengers* [3] as a coordination tool to run several parallel instances of the annealing algorithm on the same design with different probability arrays for the perturbations. When all these instances complete annealing, they exchange information about the *best* design among them which is given by a cost function [2] based on area, speed and testability costs of the digital design. This *best* design is then used as a starting point and several instances of annealing are run again in an attempt to further improve the design.

Contents

List of Figures

1 Introduction

An adaptive version of the well known Simulated Annealing algorithm is presented in [2] and its application to the combinatorial optimization problem arising in high-level synthesis of digital systems is described. A reward and penalty scheme has been implemented to learn the usefulness of the various perturbations applied to the design during annealing. The probabilities with which perturbations are applied are set to initial values and are, thereafter, "learned" as annealing progresses.

Experiments have shown us that the results of inherently random simulated annealing process produces varying results which depends on the initial probabilities with which the perturbations are applied, the design space of the particular design, the randomness of the *Metropolis criterion* used to accept or reject the new solution, among other factors. These experiments have motivated the present work, whereby we want to investigate whether the results of the Adaptive Annealing algorithm can be further improved by running the algorithm several times (in parallel) and exchanging information about the best design produced and perhaps the probability arrays of the perturbations.

In this report, we demonstrate how we have used *Messengers* [3] as a coordination tool to run several parallel instances of the annealing algorithm and use the best result of these parallel runs to re-run the annealing algorithm in an attempt to further improve the design.

2 Simulated Annealing

Simulated Annealing has been used as an optimization technique in a number of applications related to electronic design automation. For example, the popular TimberWolf placement and routing package is based on Simulated Annealing [8]. Annealing has also been applied to problems such as travelling salesperson [5], circuit partitioning [7], global routing [4], channel routing [4].

The conventional Annealing algorithm operates by starting with an initial solution to the combinatorial optimization problem and *improving* the

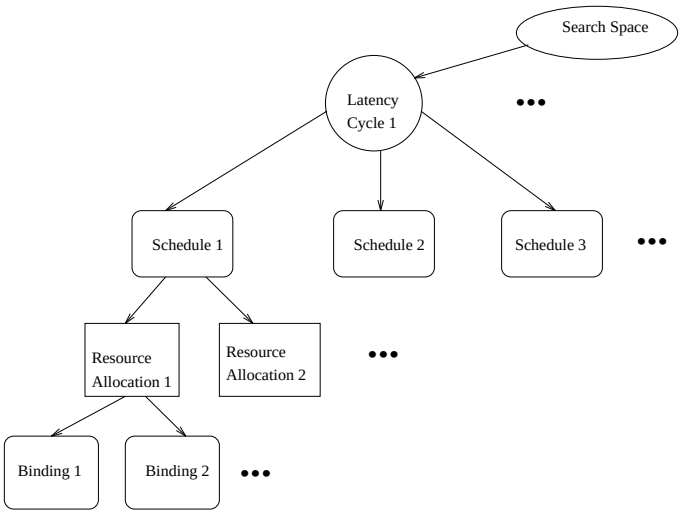

Figure 1: A Hierarchical View of the Search Space in the Data Path Synthesis Problem

solution through a series of *moves*. Unlike a greedy algorithm which only accepts system configurations resulting from better moves, annealing probabilistically accepts inferior moves.

Let S and S' denote the system configuration before and after the move. Let $cost(S)$ indicate the value of the objective function applied to system S. Then $\delta = cost(S') - cost(S)$ indicates the change in the level of the objective function. In a minimization problem, $\delta < 0$ indicates a better move. While annealing accepts better moves unconditionally, it accepts inferior moves with probability $e^{\frac{-\delta}{T}}$, where T is the *temperature* parameter [5]. Annealing begins with a high value of temperature and decreases the temperature by a factor $\alpha < 1$ at each stage until the temperature reaches a predefined lower limit. At each temperature, the algorithm makes a large number of moves until a convergence criterion is satisfied.

Figure 1 illustrates the search space in the data path synthesis problem. At the top of the hierarchy is the set of possible latency cycles for pipelined implementation. For any one of these latency cycles, there are several possible schedules. For every schedule, several resource allocations are possible, and for each resource allocation, there exist several bindings. A data path may

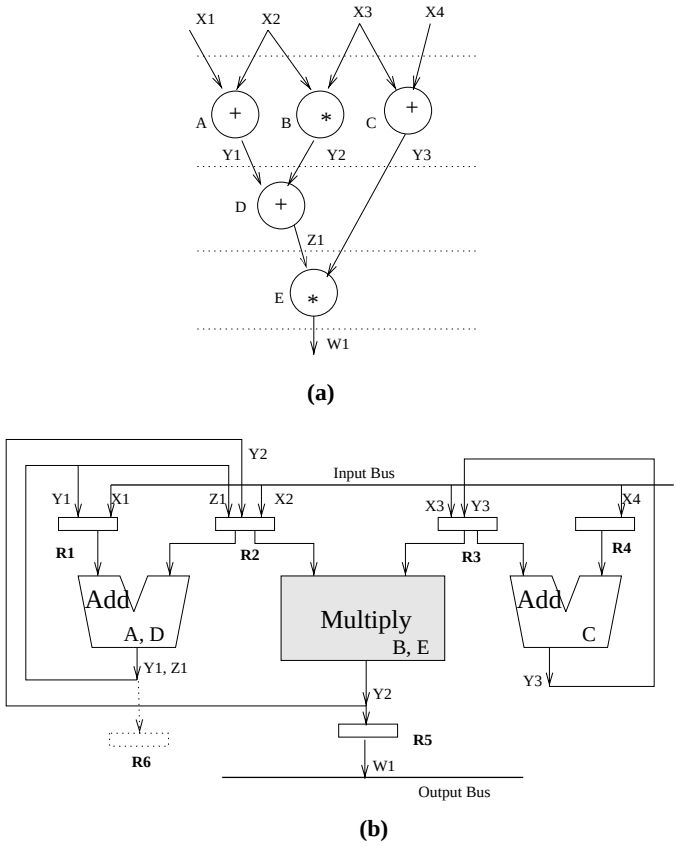

Figure 2: (a) A sample data flow graph (b) A data path corresponding to the sample data flow graph

be viewed as a path from the root of the tree in Figure 1 to a leaf node, where the nodes on the path correspond to the selection of values for the tuple $< Latencycycle, Schedule, ResourceAllocation, Binding >$. The search space of data paths is very large even for reasonably large data flow graphs. A transformation to the data path shifts the location of the corresponding path in the search tree. We illustrate a number of such transformations taking the example data flow graph in Figure 2(a).

3 Adaptive Simulated Annealing

An enhancement to the standard simulated annealing algorithm was presented in [2]. The enhanced algorithm, called *Adaptive Simulated Annealing*, is capable of "learning" [6] the best *move* which must be made at any stage in order to modify the current system configuration. The power of adaptive simulated annealing for the testability oriented data path synthesis problem has been illustrated.

In a Simulated Annealing algorithm, the selection of a transformation is a crucial decision which affects the properties of the resulting solution. An adaptive mechanism was developed for the selection of the perturbation operator based on the theory of learning automata [6] to "learn" to apply the appropriate perturb function. Initially, each of the K operators has equal probability $(1/K)$ of being selected. Depending on the outcome of the perturbation, the probability of selection of the various operators is updated according to the following rules. If the new solution resulting from the transformation i is an improved solution, we reward the operator i by increasing the probability of selection for operator i by ϵ. At the same time, the selection probability for the remaining operators is decreased by an amount of $\frac{\epsilon}{K-1}$. In a similar way, if a transformation i results in an inferior configuration, we penalize the operator i by reducing its selection probability by an amount ϵ and increasing the selection probability of the remaining operators by an amount of $\frac{\epsilon}{K-1}$. The amount ϵ is selected as follows.

$$\epsilon = \frac{\epsilon_0}{1 + e^{-\delta/T}} \tag{1}$$

where ϵ_0 is a constant smaller than 1 and close to 0. In the implementation presented, a value of $\epsilon_0 = 0.02$ was selected. Equation 1 ensures that the reward (or penalty) is proportional to the amount of improvement δ in the quality of the solution.

4 Distributed Computing: Messengers

Simulated annealing (and other similar optimization techniques such as Genetic Algorithms) are computationally very expensive even for small benchmark problems. Since high-level synthesis attempts to provide a low turnaround time (required in ASIC design), we need to look for techniques that speed up the execution of the optimization techniques in high-level synthesis.

We observed that running annealing with the learning algorithm several times even with the same initial probabilities led to very different results. Furthermore, since the synthesis search space depends on the design being considered, annealing results in widely different results when the initial probabilities of the perturbations.

We thus formulated a distributed version of the algorithm. The aim was two-fold:

- Run several instances of annealing and compare results
- Study the effect of different initial perturbation probabilities for the design under consideration

In the implementation described in this report, we anneal the same initial solution but with different initial transformation probabilities (i.e. probability arrays) on several logical nodes. After these nodes complete annealing, they communicate the best results that each of them have. The best result of all the nodes is then taken as the initial starting solution and annealing is run again on it on several nodes with different initial transformation probabilities.

The algorithm for the distributed Adaptive Simulated Annealing is given in Figure 3. This algorithm is implemented by starting with a parent messenger which reads the initial information about the design file name and the design parameters. It then spawns off P identical messengers each of which generate the initial data path and then run the simulated annealing algorithm with learning.

When annealing completes, the messengers return to the parent node with their best result and best cost. The parent node keeps the result with

```
procedure DistributedAnnealing()
begin
        Generate an initial data path $IDP$ using a latency cycle $C$,
            schedule $S$ and corresponding allocation and binding
        for $i$ = 1 to $P$ do in parallel
            Using different initial transformation probability arrays
                $PA_{i}$, apply the Simulated Annealing algorithm with
                learning to optimize the data path $IDP$ in area,
                performance and testability aspects
        endfor
        Find best cost and corresponding best data path $BDP$

        for $i$ = 1 to $P$ do in parallel
            Using different initial transformation probability arrays
                $PA_{i}$, apply the Simulated Annealing algorithm with
                learning to optimize the data path $BDP$ in area,
                performance and testability aspects
        endfor
end
```

Figure 3: Distributed Algorithm for Adaptive Simulated Annealing

the best cost among all the messengers. It then uses this result as the initial data path to repeat the process of spawning P identical messengers which run annealing on this data path. Once again, the messengers return to the parent with the best results of their annealing run. The parent then updates its best result, if one of the messengers returns a better result.

5 Results

The original Adaptive Simulated Annealing algorithm for testability-oriented synthesis was implemented on the *Sun Solaris* platform. The implementation requires about 10,000 lines of C code including an X-windows based graphical user interface.

We compared the relative performance of Adaptive Simulated Annealing with and without using the distributed paradigm for several benchmarks including the BiQuad filter, AR filter, and Elliptic Wave filter. This is shown in Table 1. The first row for each benchmark gives the results obtained

earlier without using *Messengers* and the second row gives the results using the distributed algorithm presented in sectiondistcomp.

Bench	Cost	Lat	# Mod	# Reg	# Mux	# Bilbo	Test Time
biquad	733	8	6	20	45	7	14
	745	8	6	20	48	6	12
arfilt	945	8	7	28	73	10	12
	917	8	6	28	75	6	10

Table 1: Results of Distributed Adaptive Simulated Annealing

The *cost* in the table is a compound measure of the three objective functions, namely, the silicon area, performance, and testability [9]. The area measure a refers to the area of the functional blocks such as adders and multipliers as well as the area of interconnect modules and registers. The pipeline latency ℓ is used as the performance parameter. The testability factor τ consists of the number of BILBOs [10] in the data path and the test application time. The cost function is given by

$$Cost = \beta_1 \cdot (a \times \ell) + \beta_2 \cdot \tau \tag{2}$$

where β_1 and β_2 are constants. We indicate in the tables the number of functional modules, the number of registers, the number of multiplexors, the number of BILBO registers, and the test application time in the final solution.

6 Conclusions

We have explored one of the ways of distributing the computation- expensive simulated annealing algorithm. Although previous work [4] has attempted to parallelize annealing, the novelty of this work arises from the fact that it runs co-ordinated distributed annealing threads and tries to exchange information among them so as to make annealing more effective. Also, the adaptive nature of the modified annealing algorithm used for this work, allows for a very natural venue for applying distributivity.

Due to its inherent random nature, the annealing algorithm is known to produce mixed results depending on which direction the hill climbing proceeds. The aim of this study is to identify metrics which can be tuned to direct annealing to produce better results. The methodology to do this, that has been proposed in this report, is to run several instances of annealing on the same design with different paramenters and then, "learn" a good set of parameters by monitoring the performance of these parallel runs.

In this report, we have demonstrated that *Messengers* can be efficiently used as a co-ordination tool for running and managing concurrent tasks and facilitate communication between them. *Messengers* provides a simpler and easier to use communication paradigm than previous generation distributed computing environments like PVM [11].

7 Future Work

Although this work has explored one aspect of parallelizing the adaptive simulated annealing algorithm, there are several other techniques that can be experimented with [1].

The experiments presented in this report need to be carried out with bigger designs and effect of the various parameters needs to be studied. In particular, further investigation needs to be done on the effect of the initial design, the initial probability array, the rate of cooling or annealing, the initial and final temperature and similar parameters of annealing and learning on the results of the annealing. The system developed here provides an ideal test bed to experiment with various design and annealing parameters.

References

[1] S. Gupta *An Intelligent tool for Automatic Synthesis of Testable Data paths*, B.Tech. Thesis, Dept. of Electrical Engg., Indian Institute of Technology, New Delhi, 1995

[2] C.P. Ravikumar, S. Gupta, A. Jajoo *Synthesis of Testable RTL Designs using Adaptive Simulated Annealing Algorithm*, Eleventh International Conference on VLSI Design, 1998, India,

[3] M. Fukuda, L. Bic, M. Dillencourt, F. Merchant, *Distributed Coordination with MESSENGERS*, Science of Computer Programming Journal, Special Issue on Coordination Models, Languages, and Applications, 31(2), July 1998

[4] P. Banerjee, M. Jones, and J. Sarjent. *Parallel Simulated Annealing algorithms for Cell Placement.* IEEE Transactions on Parallel and Distributed Systems, 1(1):91–105, January 1990.

[5] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. *Optimization by Simulated Annealing.* Science, 220(4598):671–680, May 1983.

[6] K.S. Narendra and M.A.L. Thathachar. *Principles of Learning Automata.* Printice Hall, 1989.

[7] C.P. Ravikumar. *Parallel Algorithms for VLSI Physical Design.* Ablex Publishing Corporation, 1996.

[8] C. Sechen and A. Sangiovanni-Vincentelli. *Timberwolf 3.2 : A new standard cell placement and global routing package.* In Proceedings of IEEE/ACM Design Automation Conference, pages 432–439, 1986.

[9] S.-P. Lin, C. Njinda, and M. Breuer. *Generating a Family of Testable Designs using the BILBO Methodology.* Journal of Electronic Testing: Theory and Applications, pages 71–89, 1993.

[10] B. Konemann, J. Moucha, and G. Zwiehoff. *Built-in logic block observation technique.* In Proceedings of IEEE Test Conference, pages 37–41, 1979.

[11] A. Geist and others. *Parallel Virtual Machine Version 3.1.* Oak Ridge National Laboratory, Oak Ridge, Tennessee, May 1993.