Design Space Exploration using Time and Resource Duality with the Ant Colony Optimization

Gang Wang Wenrui Gong Brian DeRenzi Ryan Kastner
Department of Electrical and Computer Engineering
University of California
Santa Barbara, CA 93106-9560
{wanggang, gong, kastner}@ece.ucsb.edu

ABSTRACT

Design space exploration during high level synthesis is often conducted through ad-hoc probing of the solution space using some scheduling algorithm. This is not only time consuming but also very dependent on designer's experience. We propose a novel design exploration method that exploits the duality between the time and resource constrained scheduling problems. Our exploration automatically constructs a high quality time/area tradeoff curve in a fast, effective manner. It uses the MAX-MIN ant colony optimization to solve both the time and resource constrained scheduling problems. We switch between the time and resource constrained algorithms to quickly traverse the design space. Compared to using force directed scheduling exhaustively at every time step, our algorithm provides a significant solution quality savings (average 17.3% reduction of resource counts) with similar run time on a comprehensive benchmark suite constructed with classic and real-life samples. Our algorithms scale well over different applications and problem sizes.

Categories and Subject Descriptors

J.6 [**Computer-Aided Engineering**]: Computer-Aided Design (CAD)

General Terms

Algorithms, Design, Theory

Keywords

Design Space Exploration, Ant Colony Optimization, MAX-MIN Ant System, Instruction Scheduling Algorithms

1. INTRODUCTION

When building a digital system, designers are faced with a countless number of decisions. Ideally, they must deliver the smallest, fastest, lowest power device that can implement the application at hand. More often than not, these design parameters are contradictory. For example, making the device run faster often makes it larger and more power hungry. With increasingly strict time to market, exploration tools that can quickly survey the design space and report a variety of options are invaluable.

Resource and time constrained scheduling are two fundamental high level synthesis problems that are closely related with design space exploration. One possible space exploration method is to vary the constraints and probe for solutions in a point-by-point manner by solving either of these two problems. For instance, you can use some time constrained algorithm iteratively on different input latency. This will give you a number of solutions, and their various resource allocations over a set of time points. Similarly, performing resource constrained algorithm repetitively will provide a latency for each of the given area constraints.

In this case, the designers are often left with individual tools for tackling either time constrained or resource constrained problems. They must deal with questions such as: Where do we start in the design space? What is the best way to utilize the tools? When do we stop the exploration? Moreover, due to the lack of connection amongst the traditional methods, there is little information shared between time constrained and resource constrained solutions even though the two problems are tightly interwoven. This is unfortunate, as we are essentially throwing away potential solutions since solving one problem should offer more insight to the other problem. An effective design space exploration strategy must understand and exploit the relationship between these seemingly isolated points.

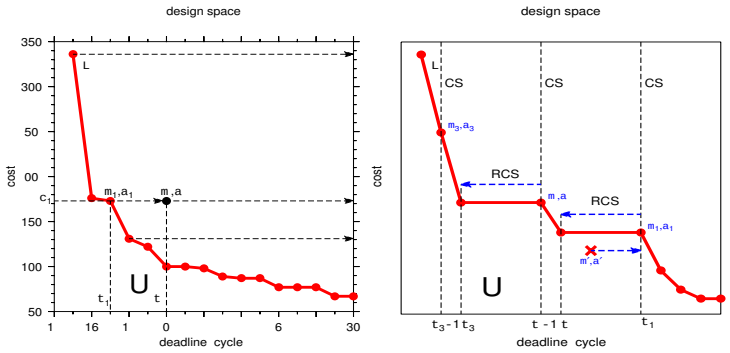
In this paper, we describe a design space exploration strategy for scheduling and resource allocation. The ant colony optimization (ACO) meta-heuristic lies at the core of our algorithm. We switch between timing and resource constrained ACO heuristics to efficiently traverse the search space. Our algorithms dynamically adjust to the input application and produce a set of high quality Pareto optimal solutions across the design space.

2. RELATED WORK

Design space exploration problems involving area cost and execution deadline tradeoffs are closely related with scheduling problems. Although these problems can be formulated with Integer Linear Programming(ILP), it is typically impossible to solve large problem instances in this manner. A lot of research work has been done to cleverly use heuristic approaches for addressing this problem. In [6, 10], genetic algorithms are implemented for design space exploration. In [5], the authors concentrate on providing alternative module bags by heuristically solving clique partitioning problems and using a force directed list scheduler. In the Voyager system [2], scheduling problems are solved by carefully bounding the design space using ILP, and good results are reported on small sized benchmarks. Other methods such as simulated annealing [9] also find their applications in this domain. A survey on design space exploration methodologies can be found in [8].

Amongst the existing approaches, the most popular method is perhaps the force directed scheduling (FDS) algorithm [11], where the parallel usage of a resource type (called force) is used as the heuristic. It is a deterministic constructive method. Though it is reported to work well on small sized problems, the algorithm lacks good lookahead function. When the input application gets more complex or the desired deadline is big, collision happens between forces, which leads to inferior solutions. This phenomena is observed in our experiments reported in Section 5.

 This work was supported by National Science Foundation Grant CNS-0524771.
DAC 2006, July 24–28, 2006, San Francisco, California, USA.



(a) (b)

Figure 1: Design Space Exploration Using Duality between Scheduling Problems

3. TIME AND RESOURCE CONSTRAINED DUALITY

We are concerned with the design problem of making tradeoffs between hardware cost and timing performance. This is still a commonly faced problem in practice, and other system metrics, such as power consumption, are closely related with them. Based on this, our design space can be viewed as a two dimensional space illustrated in Figure 1(a), where the x-axis is the execution deadline and the y-axis is the aggregated hardware cost. In this space, each point represents a specific tradeoff of the two parameters.

For a given application, the designer will be given R types of computing resources (e.g. multipliers and adders) to map the application onto. We define a specific design as a *configuration*, which is simply the number of instances for each resource type. In order to keep the discussion simple, in the rest of the paper we assume there are only two resource types M and A, though our algorithm is not limited to this constraint. Thus, each configuration can be specified by (m, a) where m is the number of resource M and a is the number of A. It is worth noticing that for each point in the design space shown in Figure 1(a), we might have multiple configurations that could realize it. Furthermore, for each specific configuration C with cost c, it covers a horizontal line in the design space starting at (t_{min}, c), where t_{min} is the resource constrained minimum scheduling time.

The goal of design space exploration is to help the designer find the optimal tradeoff between the time and area. Theoretically, this can be done by performing time constrained scheduling (TCS) on all t in the interested range. These points form a curve in the design space, as illustrated by curve L in Figure 1(a). This curve divides the design space into two parts, labeled with F and U respectively in Figure 1(a), where all the points in F are feasible to the given application while U contains all the unfeasible time/area pairs. More interestingly, we have the following attribute for curve L (Proof is not given due to space limit):

Lemma 3.1 *Curve L is monotonically non-increasing as the deadline t increases.*

Due to such a monotonically non-increasing property of curve L, there may exist horizontal segments along the curve. Based on our experience, horizontal segments appear frequently in practice. This motivates us to look into potential methods to exploit the duality between RCS and TCS to enhance the design space exploration process. First, we consider the following theorem:

Theorem 3.2 *If C is a configuration that provides the minimum cost c at time t_1, then the resource constrained scheduling result t_2 of C satisfies $t_2 \leqslant t_1$. More importantly, there is no configuration C' with a cost $c' < c$ that can produce a minimum execution time in the range of $[t_2, t_1]$.*

PROOF. The first part of the theorem is obvious. Therefore, we focus on the second part. Assuming there is a configuration C' that provides an execution time $t_3 \in [t_2, t_1]$, then C' must be able to produce t_1.Since C' has a smaller cost, this conflicts with the fact that C is the minimum cost solution (i.e. the TCS solution) at time t_1. Thus the statement is true. This is illustrated in Figure 1(b) with configuration (m_1, a_1) and (m', a'). □

This theorem provides a key insight for the design space exploration problem. It says that if we can find a configuration with optimal cost c at time t_1, we can move along the horizontal segment from (t_1, c) to (t_2, c) without losing optimality. Here t_2 is the RCS solution for the found configuration. This enables us to efficiently construct the curve L by iteratively using TCS and RCS algorithms and leveraging the fact that such horizontal segments do frequently occur in practice. Based on the above discussion, we propose a new space exploration algorithm as shown in Algorithm 1 that exploits the duality between RCS and TCS solutions. Notice the *min* function in step 10 is necessary since a practical RCS algorithm may not return the true optimal and could be worse than t_{cur}.

procedure DSE
output: curve L
1: interested time range $[t_{min}, t_{max}]$,
2: $L = \phi$
3: $t_{cur} = t_{max}$
4: **while** $t_{cur} \geqslant t_{min}$ **do**
5: perform TCS on t_{cur} to get the optimal configurations C_i.
6: **for** configuration C_i **do**
7: perform RCS to obtain the minimum time t^i_{rcs}
8: **end for**
9: $t_{rcs} = \min_i (t^i_{rcs})$ /* find the best rcs time */
10: $t_{cur} = \min (t_{cur}, t_{rcs}) - 1$
11: extend L based on TCS and RCS results
12: **end while**
13: return L

Algorithm 1: Iterative Design Space Exploration Algorithm

4. ANT COLONY OPTIMIZATIONS

In order to select the suitable TCS and RCS algorithms, we studied different scheduling approaches for the two problems, including the popularly used force directed scheduling (FDS) for the TCS problem [11], various list scheduling heuristics, and the recently

proposed ant colony optimization (ACO) based instruction scheduling algorithms [13]. We chose the ACO approach for our design space exploration algorithm. Compared with traditional methods such as FDS and list scheduling, the ACO-based scheduling algorithms offer the following major benefits:

- It generates better quality scheduling results that are close to the optimal with good stability for both the TCS and RCS problems [13].
- It provides reasonable runtime. It has the same complexity as FDS method for the TCS problem.
- More importantly, as a population based method, ACO-based approach naturally provides multiple alternative solutions. This is typically not available for traditional methods, especially for force directed TCS scheduling. This feature provides potential benefit in the iterative process for our algorithm since we can select the largest jump amongst these candidates.

Ant colony optimization was originally introduced by Dorigo *et al.* [4]. It is a population based approach inspired by ethological studies on the behavior of ants, in which a collection of agents cooperate together to explore the search space. They communicate via a mechanism imitating the pheromone trails, including autocatalytic feedback and evaporation. One of the first problems to which ACO was successfully applied was the Traveling Salesman Problem (TSP) [4], for which it gave competitive results compared to traditional methods. Researchers have since formulated ACO methods for a variety of traditional $\mathcal{NP}$-hard problems[3]. Effective algorithms have been constructed to solve time and resource constrained scheduling problems using the MAX-MIN Ant System(MMAS) [12], a variant of the original ACO approach. Space does not permit us to elaborate the algorithms here. Interested readers are encouraged to refer to [14, 13].

5. EXPERIMENTS AND ANALYSIS

5.1 Benchmarks and Setup

In order to test and evaluate our algorithms, we have constructed a comprehensive set of benchmarks. These benchmarks are taken from one of two sources: (1) Popular benchmarks used in previous literature; (2) Real-life examples generated and selected from the MediaBench suite [7].

The benefit of having classic samples is that they provide a direct comparison between results generated by our algorithm and results from previously published methods. This is especially helpful when some of the benchmarks have known optimal solutions. In our final testing benchmark set, seven samples widely used in instruction scheduling studies are included.

However, these samples are typically small to medium in size, and are considered somewhat old. To be representative, it is necessary to create a more comprehensive set with benchmarks of different sizes and complexities. Such benchmarks shall aim to provide challenging samples for instruction scheduling algorithms with regards to larger number of operations, higher level of parallelism and data dependency on more up-to-date testing cases from modern and real-life applications. They should also help us with a wider range of synthesis problems to test the algorithms' scalability.

With the above goals, we investigated the MediaBench suite, which contains a wide range of complete applications for image processing, communications and DSP applications. We analyzed these applications using the SUIF and Machine SUIFand over 14,000 DFGs were extracted as preliminary candidates for our benchmark set. After careful study, 13 samples were selected ranging from matrix operation (the *invertmatrix* benchmark) to imaging processing algorithm (the *jpeg idctifast* and *smoothcolor* benchmarks).

Table 1 lists all twenty benchmarks that were included in our final benchmark set. The "names" column gives the various functions where the basic blocks originated; the "size" column gives the the nodes/edges number pair. This benchmark set, including related statistics, DFG graphs and source code for the all testing samples, is available online [1].

We implemented three different design space exploration algorithms: 1) FDS: exhaustively step through the time range by performing time constrained force directed scheduling at each deadline;2) MMAS-TCS: step through the time range by performing only MMAS-based TCS scheduling at each deadline; and 3)MMAS-D: use the iterative approach proposed in Algorithm 1 by switching between MMAS-based RCS and TCS.

The MMAS-based TCS and RCS algorithms as similar to those described in Section [14]. Since there is no widely distributed and recognized FDS implementation, we implemented our own. The implementation is based on [11] and has all the applicable refinements proposed in the paper, including multi-cycle instruction support, resource preference control, and look-ahead using second order of displacement in force computation.

For all testing benchmarks, the operations are allocated on two types of computing resources, namely MUL and ALU, where MUL is capable of handling multiplication and division, while ALU is used for other operations such as addition and subtraction. Furthermore, we define the operations running on MUL to take two clock cycles and the ALU operations take one. This definitely is a simplified case from reality, however, it is a close enough approximation and does not change the generality of the results. Other resource mappings can easily be implemented within our framework.

With the assigned resource/operation mapping, ASAP is first performed to find the critical path delay L_c. We then set our predefined deadline range to be $[L_c, 2L_c]$, i.e. from the critical path delay to 2 times of this delay. This results in 263 testing cases in total. Three design space exploration experiments are carried out. For the FDS and MMAS-TCS algorithms, we run force-directed or MMAS-based time constrained scheduling on every deadline and report the best schedule results together with the costs obtained. For the MMAS-D algorithm, we only run MMAS-based TCS on selected deadlines starting from $2L_c$ and make jumps based on the MMAS RCS results on the TCS results obtained previously.

5.2 Quality Assessment

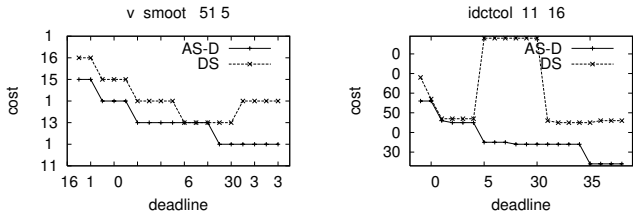
As discussed in Section 5.1, we perform three experiments on all the benchmark samples using different algorithms. First, time constrained FDS scheduling is used at every deadline. The quality of results is used as the baseline for quality performance assessment. Then MMAS-TCS and MMAS-D algorithms are executed; the difference is that MMAS-TCS steps through the design space in the same way as FDS while MMAS-D uses duality between TCS and RCS. Each of these two algorithms are executed five times in order to obtain enough statistics to evaluate their stability.

Detailed design space exploration results for some of benchmark samples are shown in Figure 2, where we compare the curves obtained by MMAS-D and FDS algorithms. Table 1 summarizes the experiment results. In each row, together with the benchmark name, we give the node/edge count, the average resource saving obtained MMAS-TCS and MMAS-D algorithms comparing with FDS. Though we do use different cost weights to bias alternative solutions (for example, solution (3M, 4A) is more favorable than (4M, 3A) as resource M has a large cost weight), we report the saving in percentage of total resource counts. We feel this is more objective and avoids confusion caused by different weight choices. The saving is computed for every deadline used for each bench-

mark, then the average for a certain benchmark is taken and reported in Table 1. It is easy to see that MMAS-TCS and MMAS-D both outperform the classic FDS method across the board with regard to solution quality, often with significant savings. Overall, MMAS-TCS achieves an average improvement of 16.4% while MMAS-D obtains a 17.3% improvement. Both algorithms scale well for different benchmarks and problem sizes. Moreover, by computing the standard deviation over the 5 different runs, the algorithms are shown to be very stable. For example, the average standard deviation on result quality for MMAS-TCS is only 0.104.

Name	Size	Deadline	MMAS-TCS	MMAS-D
HAL	11/8	(6 - 12)	-7.1%	-7.1%
hbsurf†	18/16	(11 - 22)	-9.9%	-13.2%
ARF	28/30	(11 - 22)	-12.4%	-18.6%
motionvectors†	32/29	(7 - 14)	-13.1%	-16.0%
EWF	34/47	(17 - 34)	-11.5%	-21.9%
FIR2	40/39	(12 - 24)	-16.8%	-22.8%
FIR1	44/43	(12 - 24)	-15.2%	-18.0%
h2v2_smooth†	51/52	(17 - 34)	-19.3%	-20.5%
feedbackpoints†	53/50	(11 - 22)	-5.9%	-9.1%
collapsepyr†	56/73	(8 - 16)	-18.3%	-20.0%
COSINE1	66/76	(10 - 20)	-21.5%	-23.5%
COSINE2	82/91	(10 - 20)	-5.6%	-8.1%
wbmpheader†	106/88	(8 - 16)	-0.9%	-1.6%
interpolate†	108/104	(10 - 20)	-0.2%	-1.8%
matmul†	109/116	(11 - 22)	-3.7%	-5.6%
idctcol†	114/164	(19 - 38)	-30.7%	-32.0%
jpegidctifast†	122/162	(17 - 34)	-50.3%	-52.1%
jpegfdctislow†	134/169	(16 - 32)	-31.4%	-34.6%
smoothcolor†	197/196	(15 - 30)	-7.3%	-8.6%
invertmatrix†	333/354	(15 - 30)	-11.2%	-11.9%
Total Avg.			-16.4%	-17.3%

Table 1: Result Summary for Design Space Exploration
Benchmarks with † are extract from MediaBench.
Saving is computed based on FDS results. No weight applied.



(a) (b)

Figure 2: Design Space Exploration: MMAS-D vs. FDS

It is interesting and initially surprising to observe that the MMAS-D always had better performance than MMAS-TCS method. More carefully inspection on the experiments reveals the reason: using the duality between TCS and RCS not only saves us computation but also improves the result quality. To understand this, we recall Theorem 3.2 and Figure 1(b). If we achieve an optimal solution at t_1, with MMAS-D we automatically extend this optimality from t_1 to t_2, while an unperfect MMAS-TCS still have chance to provide worse quality solutions on deadlines between t_1 and t_2.

All of the experiment results are obtained on a Linux box with a 2GHz CPU. Figure 3 diagrams the average execution time comparison for the three design space exploration approaches, ordered by the size of the benchmark. It is easy to see that the all the algorithms have similar run time scale, where MMAS-TCS takes more time, while MMAS-D and FDS have very close run times–especially on

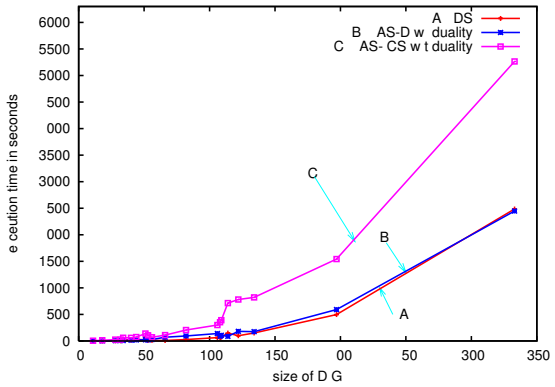


Figure 3: Timing Performance Comparison

larger benchmarks. The major execution time savings come from the fact that MMAS-D exploits the duality and only computes TCS on selected number of deadlines. Over 263 testing cases, we find on average MMAS-D skips about 44% deadlines with the help of RCS. The fact that MMAS-D achieves much better results than FDS with almost the same execution time makes it very attractive in practice.

6. CONCLUSION

We proposed a novel design space exploration method that bridges the time and resource constrained scheduling problems and exploits the duality between them.Combined with the MMAS, our algorithms outperformed the popularly used force directed scheduling method with significant savings (average 17.3% savings on resources) with almost the same run time on comprehensive benchmarks constructed with classic and real-life samples. The algorithms scaled well over different applications and problem sizes.

7. REFERENCES

[1] http://express.ece.ucsb.edu/benchmark/.

[2] S. Chaudhuri, S. A. Blythe, and R. A. Walker. A solution methodology for exact design space exploration in a three-dimensional design space. *IEEE Trans. Very Large Scale Integr. Syst.*, 5(1):69–81, 1997.

[3] D. Corne, M. Dorigo, and F. Glover, editors. *New Ideas in Optimization*. McGraw Hill, London, UK, 1999.

[4] M. Dorigo, V. Maniezzo, and A. Colorni. Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Transactions on Systems, Man and Cybernetics, Part-B*, 26(1):29–41, February 1996.

[5] R. Dutta, J. Roy, and R. Vemuri. Distributed design-space exploration for high-level synthesis systems. In *DAC '92*, pages 644–650, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

[6] M. J. M. Heijligers, L. J. M. Cluitmans, and J. A. G. Jess. High-level synthesis scheduling and allocation using genetic algorithms. page 11, 1995.

[7] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. MediaBench: a Tool for Evaluating and Synthesizing Multimedia and Communicatons Systems. In *Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture*, 1997.

[8] Y.-L. Lin. Recent developments in high-level synthesis. *ACM Trans. Des. Autom. Electron. Syst.*, 2(1):2–21, 1997.

[9] J. Madsen, J. Grode, P. V. Knudsen, M. E. Petersen, and A. Haxthausen. LYCOS: the Lyngby Co-Synthesis System. *Design Automation for Embedded Systems*, 2(2):125–63, March 1997.

[10] M. Palesi and T. Givargis. Multi-Objective Design Space Exploration Using GeneticAlgorithms. In *Proceedings of the Tenth International Symposium on Hardware/SoftwareCodesign*, 2002.

[11] P. G. Paulin and J. P. Knight. Force-directed scheduling in automatic data path synthesis. In *24th ACM/IEEE Conference Proceedings on Design Automation Conference*, 1987.

[12] T. Stützle and H. H. Hoos. MAX-MIN Ant System. *Future Generation Comput. Systems*, 16(9):889–914, September 2000.

[13] G. Wang, W. Gong, and R. Kastner. Instruction scheduling Using MAX-MIN Ant Optimization. In *GLSVLSI'2005*, April 2005.

[14] G. Wang, W. Gong, B. DeRenzi, and R. Kastner. Ant colony optimizations for resource and timing constrained instruction scheduling. *IEEE Transaction on Computer-Aided Design*, under review.