Exploring Time/Resource Trade-Offs by Solving Dual Scheduling Problems with the Ant Colony Optimization

GANG WANG
University of California, Santa Barbara
WENRUI GONG
Mentor Graphics
BRIAN DERENZI
University of Washington
and
RYAN KASTNER
University of California, Santa Barbara

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 of time- and resource-constrained scheduling problems. Our exploration automatically constructs a time/area tradeoff curve in a fast, effective manner. It is a general approach and can be combined with any high-quality scheduling algorithm. In our work, we use the max-min ant colony optimization technique to solve both time- and resource-constrained scheduling problems. Our algorithm provides significant solution-quality savings (average 17.3% reduction of resource counts) with similar runtime compared to using force-directed scheduling exhaustively at every time step. It also scales well across a comprehensive benchmark suite constructed with classic and real-life samples.

Categories and Subject Descriptors: J.6 [**Computer Applications**]: Engineering—*Computer-aided design (CAD)*

General Terms: Algorithms, Design, Theory

This work was supported in part by the National Science Foundation Grant CNS-0524771.
Authors' addresses: G. Wang (corresponding author), R. Kastner, Department of Electrical and Computer Engineering, University of California at Santa Barbara, Santa Barbara, CA 93106-9560; email: {wanggang, kastner}@engr.ucsb.edu; W. Gong, Mentor Graphics Corporation, 8005 SW. Boeckman Rd., Wilsonville, OR 97070; email: wenrui_gong@mentor.com; B. Derenzi, Department of Computer Science and Engineering, University of Washington, PO Box 352350, Seattle, WA 98195-2350; email: bderenzi@cs.washington.edu.

Additional Key Words and Phrases: Design space exploration, ant colony optimization, instruction scheduling, max-min ant system

ACM Reference Format:

Wang, G., Gong, W., Derenzi, B., and Kastner, R. 2007. Exploring time/resource trade-offs by solving dual scheduling problems with the ant colony optimization. ACM Trans. Des. Automat. Electron. Syst. 12, 4, Article 46 (September 2007), 20 pages. DOI = 10.1145/1278349.1278359 http:/doi.acm.org/10.1145/1278349.1278359.

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. Designers must also deal with increasingly strict time-to-market issues. Unfortunately, this does not afford them much time to make a decision.

Designers must be able to reason about the tradeoffs amongst a set of parameters. Such decisions are often made based on experience, that is, this worked before, it should work again. Exploration tools that can quickly survey the design space and report a variety of options are invaluable.

From an optimization point of view, design space exploration can be distilled to identifying a set of Pareto-optimal design points according to some objective function. These design points form a curve that provides the best tradeoffs for the variables in the objective function. Once the curve is constructed, the designer can make design decisions based on the relative merits of the various system configurations. Timing performance and the hardware cost are two common objectives in such a process.

Resource allocation and scheduling are two fundamental problems in constructing such Pareto-optimal curves for time/cost tradeoffs. The two problems are tightly interwoven. Resource-constrained scheduling takes as input an application modeled as data flow graph and a number of different types of resources. It outputs a start time for each of the operations such that the resource constraints are not violated, while attempting to minimize the application latency. Here allocation is performed before scheduling, and the schedule is obviously very dependent on the allocation; a different resource allocation will likely produce a vastly different scheduling result.

We could perform scheduling before allocation; this is the time-constrained scheduling problem. Here the inputs are a data flow graph and a time deadline (latency). The output is again a start time for each operation such that the latency is not violated, while attempting to minimize the number of resources that are needed. It is not clear as to which solution is better, nor is it clear as to the order in which we should perform scheduling and allocation.

One possible method of design space exploration is to vary the constraints to probe for solutions in a point-by-point manner. For instance, you can use some time-constrained algorithm iteratively, where each iteration has a different input latency. This will give you a number of solutions, and their various

resource allocations over a set of time points. Or you can run some resource-constrained algorithm iteratively. This will give you a latency for each of these area constraints.

An effective design space exploration strategy must understand and exploit the relationship between time- and resource-constrained problems. Unfortunately, designers are left with individual tools for tackling either problem. They are faced with questions like: Where do we start the design space exploration? What is the best way to utilize the scheduling tools? When do we stop the exploration?

Moreover, due to the lack of connection amongst the traditional methods, there is very little information shared between time-constrained and resource-constrained solutions. This is unfortunate, as we are throwing away potential solutions; solving one problem can offer more insight into the other.

In this article, we describe a design space exploration strategy for scheduling and resource allocation. The ant colony optimization (ACO) metaheuristic 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 solutions across the design space.

The article is organized as follows. We discuss related work in the next section. In Section 3, we present a design space exploration algorithm using duality between the time and resource scheduling problems. Section 4 discusses the ant colony optimizations that we use to search the design space. Experimental results for the new algorithms are presented and analyzed in Section 5. We conclude with Section 6.

2. RELATED WORK

Scheduling and resource allocation problems form the basis for design space exploration during high-level synthesis. These problems can be formulated as an integer linear program (ILP) [Wilken et al. 2000], however, it is typically impossible to solve large problem instances in this manner. Much research has been done to cleverly use heuristic approaches to address these problems.

In Dutta et al. [1992], the authors concentrate on providing alternative "module bags" for design space exploration by heuristically solving clique partitioning problems and using force-directed scheduling. Their work focuses more on those situations where the operations in the design can be executed on alternative resources. In the Voyager system [Chaudhuri et al. 1997], scheduling problems are solved by carefully bounding the design space using ILP, and good results are reported on small-sized benchmarks. Moreover, it reveals that clock selection can have an important impact on the final performance of the application. In Heijligers et al. [1995], Dick and Jha [1997], and Palesi and Givargis [2002], genetic algorithms are implemented for design space exploration. Simulated annealing [Madsen et al. 1997] has also been applied in this domain. A survey on design space exploration methodologies can be found in Lin [1997] and compared with McFarland et al. [1990].

Force directed scheduling (FDS) [Paulin and Knight 1987] is a popular scheduling algorithm. The original FDS algorithm is designed to solve the

time-constrained scheduling (TCS) problem, that is, to reduce the number of functional units used in the implementation with a given execution deadline. This objective is achieved by attempting to uniformly distribute the operations onto the available resource units. The distribution ensures that resource units allocated to perform operations in one control step are used efficiently in all other control steps, which leads to a high utilization rate. A"force" is used to measure the parallel usage of a resource type. Each force is computed based on the operation's mobility range, under the assumption that each operation op_i has a uniform probability of being scheduled into any of the control steps in this range. The algorithm proceeds iteratively by selecting the operation and time step with the minimal force. The authors also proposed a method called force-directed list scheduling (FDLS) to address the resource-constrained scheduling problem. Here, the priority function of the list scheduler is constructed using forces.

The FDS method is constructive, since the solution is computed without backtracking. Every decision is made deterministically in a greedy manner. If there are two potential assignments with the same cost, the FDS algorithm cannot accurately estimate the best choice. Moreover, FDS does not take into account future assignments of operators to the same control step. Consequently, it is possible that the resulting solution will not be optimal due to its greedy nature. FDS works well on small-sized problems, however, it often results in inferior solutions for more complex problems. This phenomenon is observed in our experiments, reported in Section 5.

In this article, we focus our attention on the basic design space exploration problem similar to the one treated in Paulin and Knight [1987], where the designers are faced with the task of mapping a well-defined application represented as a DFG onto a set of known resources where the compatibility between operations and resource types has been defined. Furthermore, clock selection has been determined in the form of execution cycles for the operations. The goal is to find the Pareto-optimal tradeoff amongst design implementations with regard to timing and resource costs. Our basic method can be extended to handle clock selection and the use of alternative resources. However, this is beyond the scope of this article.

3. EXPLORATION USING 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. Based on this, we have a 2D design space as illustrated in Figure 1(a), where the x-axis is the execution deadline and the y-axis is the aggregated hardware cost. Each point represents a specific tradeoff of the two parameters.

For a given application, the designer is given R types of computing resources (e.g., multipliers and adders) to map the application to the target device. We define a specific design as a *configuration*, which is simply the number of each specific resource type. In order to keep the discussion simple, in the rest of the article we assume there are only two resource types, namely M (multiply)

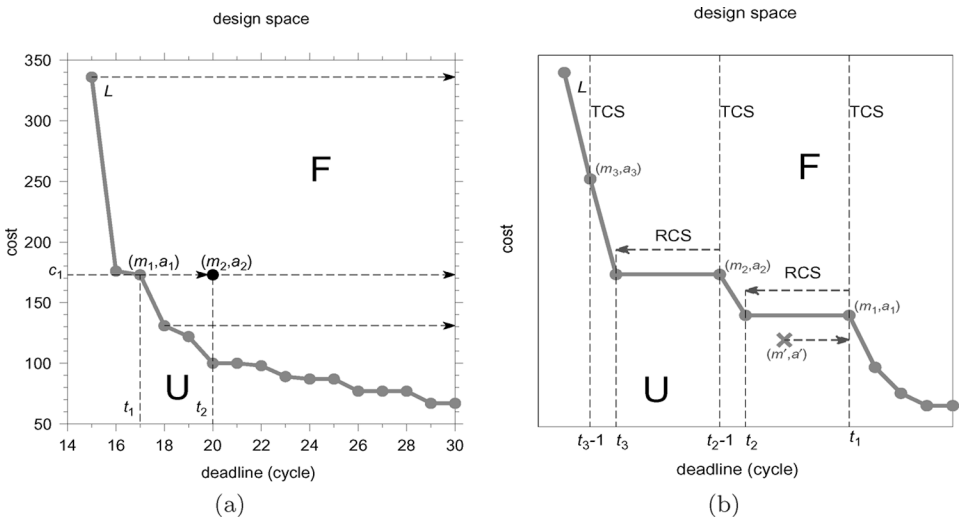

Fig. 1. Design space exploration using duality between schedule problems (curve l gives the optimal time/cost tradeoffs).

and A (add), 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. For example, assuming unit cost for all resources, it is possible that a configuration with 10 multipliers and 10 adders can achieve the same execution time as another configuration with 5 multipliers and 15 adders, as both solutions have the same cost (i.e., 20).

Studying the design space more carefully reveals several key observations. First, achievable deadlines are limited to the range $[t_{asap}, t_{seq}]$, where t_{asap} is the ASAP time for the application while t_{seq} is the sequential execution time when we have only one instance for each resource type. It is impossible to get a solution faster than the ASAP solution, and any with a deadline beyond that of t_{seq} are not Pareto-optimal. Furthermore, for each specific configuration we have the following lemma about the portion of the design space to which it maps.

LEMMA 3.1. *Let C be a feasible configuration with cost c for the target application. The configuration maps to a horizontal line in the design space starting at (t_{min}, c), where t_{min} is the resource-constrained minimum scheduling time.*

The proof of the lemma is straightforward, as each feasible configuration has a minimum execution time t_{min} for the application, and obviously can handle every deadline longer than t_{min}. For example, in Figure 1(a), if the configuration (m_1, a_1) has cost c_1 and minimum scheduling time t_1, the portion of design space that it maps to is indicated by the arrow next to it. Of course, it is possible for another configuration (m_2, a_2) to have the same cost but greater minimum scheduling time t_2. In this case, their feasible space overlaps beyond (t_2, c_1).

As we discussed before, the goal of design space exploration is to help the designer find the optimal tradeoff between time and area. Theoretically, this can be done by finding the minimum area c amongst all configurations that are capable of producing $t \in [t_{asap}, t_{seq}]$. In other words, we can find these points by performing time-constrained scheduling (TCS) on all t in the range of interest. 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(b), 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.

LEMMA 3.2. *Curve L is monotonically nonincreasing as the deadline t increases.*

PROOF. Assume the lemma is false. Therefore, we will have two points (t_1, c_1) and (t_2, c_2) on the curve L, where $t_1 < t_2$ and $c_1 < c_2$. This means we have a specific configuration C with cost c_1 that is capable of producing an execution time t_1 for the application. Since $t_1 < t_2$, and also from Lemma 3.1, we know that configuration C can produce t_2. This introduces a contradiction, since c_2, which is worse than c_1, is the minimum cost at t_2. □

Due to this lemma, we can use the dual solution of finding the tradeoff curve by identifying the minimum resource-constrained scheduling (RCS) time t amongst all configurations with cost c. Moreover, because of the monotonically nonincreasing 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 3.3. *If C is a configuration that provides the minimum cost at time t_1, then the resource-constrained scheduling result t_2 of C satisfies $t_2 \leq t_1$. More importantly, there is no configuration C' with a smaller cost that can produce an execution time within $[t_2, t_1]$.*

PROOF. The first part of the theorem is obvious. Therefore, we focus on the second. 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 based on Lemma 3.1. 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 preceding 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 heuristic RCS algorithm may not return the true optimal that could be worse than t_{cur}.

By iteratively using the RCS and TCS algorithms, we can quickly explore the design space. Our algorithm provides benefits in runtime and solution quality compared with using RCS or TCS alone. Our algorithm performs exploration starting from the largest deadline t_{max}. Under this case, the TCS result will provide a configuration with a small number of resources. RCS algorithms have a better chance to find the optimal solution when the resource number is small, therefore provide a better opportunity to make large horizontal jumps. On the other hand, TCS algorithms take more time and provide poor solutions when the deadline is unconstrained. We can gain significant runtime savings by trading off between RCS and TCS formulations.

The proposed framework is general and can be combined with any scheduling algorithm. We found that in order for it to work in practice, the TCS and RCS algorithms used in the process require special characteristics. First, they must be fast, which is generally requested for any design space exploration tool. More importantly, they must provide close to optimal solutions, especially for the TCS problem. Otherwise, the conditions for Theorem 3.3 will not be satisfied and the generated curve L will suffer significantly in quality. Moreover, notice that we enjoy the biggest jumps when we take the minimum RCS result amongst all configurations that provide the minimum cost for the TCS problem. This is reflected in steps 6–9 in Algorithm 1. For example, it is possible that both (m, a) and (m', a') provide the minimum cost at time t, but have different deadline limits. Therefore, a good TCS algorithm used in the proposed approach should be able to provide multiple, if not all, candidate solutions with the same minimum cost.

Algorithm 1. Iterative Design Space Exploration.

procedure DSE
output: curve L
1: interested time range $[t_{min}, t_{max}]$, where $t_{min} \geq t_{asap}$ and $t_{max} \leq t_{seq}$.
2: $L = \phi$
3: $t_{cur} = t_{max}$.
4: **while** $t_{cur} \geq t_{min}$ **do**
5: perform TCS on t_{cur} to obtain 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

4. ANT COLONY OPTIMIZATIONS FOR TIME- AND RESOURCE-CONSTRAINED SCHEDULING

In order to select 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 [Paulin and Knight 1987], various list scheduling heuristics, and the recently proposed ant colony optimization (ACO)-based instruction scheduling algorithms [Wang et al. 2005; Wang et al. to appear].

We found that ACO-based scheduling algorithms offer the following major benefits over FDS, several variants of list scheduling, and simulated annealing [Wang et al. to appear]: (1) ACO-based scheduling algorithms generate better-quality results that are close to optimal, with good stability for both the TCS and RCS problems; and (2) ACO-based methods provide reasonable runtime; (3) as a population-based method, the ACO-based TCS approach naturally provides multiple alternative solutions. As we have discussed, this feature provides potential benefits in the proposed DSE process, since we can select the largest jump provided by these candidates.

4.1 Ant Colony Optimization (ACO)

ACO is a cooperative heuristic searching algorithm inspired by ethological studies on the behavior of ants [Dorigo et al. 1996]. It was observed that ants—who lack sophisticated vision—manage to establish the optimal path between their colony and a food source within a very short period of time. This is done through indirect communication known as *stigmergy* via chemical substance, or *pheromone*, left by the ants on the paths. Each individual ant makes a decision on its direction biased on the "strength" of the pheromone trails that lie before it, where a higher amount of pheromone hints a better path. As an ant traverses a path, it reinforces that path with its own pheromone. A collective autocatalytic behavior emerges as more ants choose the shortest trails, which in turn creates an even larger amount of pheromone on the short trails, making such short trails more attractive to future ants. The ACO algorithm is inspired by this observation. It is a population-based approach where a collection of agents cooperate together to explore the search space. They communicate via a mechanism imitating pheromone trails.

One of the first problems to which ACO was successfully applied was the traveling salesman problem (TSP) [Dorigo et al. 1996], for which it gave competitive results compared with traditional methods. Researchers have since formulated ACO methods for a variety of traditional $\mathcal{NP}$-hard problems. These problems include the maximum clique problem, quadratic assignment problem, graph coloring problem, shortest common supersequence problem, and multiple knapsack problem. ACO also has been applied to practical problems such as the vehicle routing problem, data mining, network routing problem, and system-level task partitioning problem [Corne et al. 1999; Wang et al. 2004, 2003].

It was shown [Gutjahr 2002] that ACO converges to an optimal solution with probability of exactly one; however there is no constructive way to guarantee

this. Balancing exploration to achieve close to optimal results within manageable time remains an active research topic for ACO algorithms. The max-min ant system (MMAS) [Stützle and Hoos 2000] is a popularly used method to address this problem. MMAS is built upon the original ACO algorithm and improves it by providing dynamically evolving bounds on the pheromone trails so that the heuristic never strays too far from the best encountered solution. As a result, all possible paths will have a nontrivial probability of being selected; thus it encourages broader exploration of the search space while maintaining a good differential between alternative solutions. It was reported that MMAS was the best-performing ACO approach on a number of classic combinatory optimization tasks.

4.2 Time- and Resource-Constrained Scheduling Formulation

We formulated two MMAS-based algorithms for the TCS and RCS problems, respectively. These algorithms apply traditional heuristics within the ACO framework. The scheduling problems thus are formulated as searching problems and the algorithms employ a collection of agents that collaboratively explore the search space. A stochastic decision making strategy is proposed in order to combine global and local heuristics to effectively conduct this exploration. As the algorithm proceeds in finding better-quality solutions, dynamically computed local heuristics are utilized to better guide the searching process. The inputs for these algorithms are a DFG representation for the application, the resource types, and their counts for the RCS problem or the desired deadline for the TCS problem. The ant colony optimization (ACO) algorithm plays a core role in our work. Unfortunately, in the interests of space, we can only give a general introduction on the ACO formulation for the TCS problem. For a complete treatment of the algorithms, including detailed discussion on their implementation, applicability, complexity, extensibility, parameter selection, and performance, please refer to our previous publications [Wang et al. 2005; Wang et al. to appear].

In its ACO-based formulation, the TCS problem is solved with an iterative searching process. Each iteration consists of two stages. First, the ACO algorithm is applied where a collection of ants traverse the DFG to construct individual operation schedules with respect to the specified deadline, using global and local heuristics. Second, these scheduling results are evaluated using their resource costs. The associated heuristics are then adjusted based on the solutions found in the current iteration. The hope is that future iterations will benefit from this adjustment and come up with better schedules.

Each operation or DFG node op_i is associated with D pheromone trails τ_{ij}, where $j = 1, \ldots, D$ and D is the specified deadline. These pheromone trails indicate the global favorableness of assigning the ith operation at the jth control step in order to minimize the resource cost with respect to the time constraint. Initially, based on ASAP and ALAP results, τ_{ij} is set with some fixed value τ_0 if j is a valid control step for op_i; otherwise, it is set to be 0.

For each iteration, m ants are released and each ant individually starts to construct a schedule by picking an unscheduled instruction and determining its desired control step. However, unlike the deterministic approach used in the

FDS method, each ant picks up the next instruction for the scheduling decision probabilistically. Once an instruction op_h is selected, the ant needs to make decision on which control step it should be assigned. This decision is also made probabilistically, as illustrated in Eq. (1).

$$\mathsf{p}_{hj} = \begin{cases} \frac{\tau_{hj}(t)^{\alpha} \cdot \eta_{hj}^{\beta}}{\sum_l (\tau_{hl}^{\alpha}(t) \cdot \eta_{hl}^{\beta})} & \text{if } op_h \text{ can be scheduled at } l \text{ and } j \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Here j is the time step under consideration. The item η_{hj} is the local heuristic for scheduling operation op_h at control step j, and α and β are parameters to control the relative influence of the distributed global heuristic τ_{hj} and local heuristic η_{hj}. Assuming op_h is of type k, η_{hj} is simply set to be the inverse of the distribution graph value [Paulin and Knight 1987], which is computed based on the partial scheduling result and is an indication of the number of computing units of type k needed at control step j. In other words, an ant is more likely to make a decision that is globally considered "good" and also uses the fewest number of resources under the current partially scheduled result. We do not recursively compute the forces on successor and predecessor nodes. Thus selection is much faster. Furthermore, the time frames are updated to reflect the changed partial schedule. This guarantees that each ant will always construct a valid schedule.

Algorithm 2. MMAS for Timing-Constrained Scheduling.

procedure MaxMinAntSchedulingTCS(G,R)
input: DFG $G(V, E)$, resource set R
output: operation schedule
1: initialize parameter ρ, τ_{ij}, p_{best}, τ_{max}, τ_{min}
2: construct m ants
3: *BestSolution* $\leftarrow \phi$
4: **while** ending condition is not met **do**
5: for $i = 0$ **to** m **do**
6: $ant(i)$ constructs a valid schedule timing constrained $S_{current}$ as following:
7: $S_{current} \leftarrow \phi$
8: perform ASAP and ALAP
9: **while** exists unscheduled operation **do**
10: update time frame $[t_i^S, t_i^L]$ associated with each operation op_i and the distribution graphs q_k.
11: select one operation op_h among all unscheduled operations probabilistically
12: for $t_h^S \leq j \leq t_h^L$ **do**
13: set local heuristic $\eta_{hj} = 1/q_k(j)$ where op_h is of type k
14: **end for**
15: select time step l using η and τ as Equation (1).
16: $S_{current} = schedule(S_{current}, op_h, l)$
17: Update time frame and distribution graphs based on $S_{current}$
18: **end while**
19: **if** $S_{current}$ is better than that of *BestSolution* **then**
20: *BestSolution* $\leftarrow S_{current}$
21: **end if**
22: **end for**

23: update τ_{max} and τ_{min} using MMAS principle
24: **update η if needed**
25: update τ_{ij} based on Equation (2)
26: **end while**
27: return *BestSolution*

In the second stage of our algorithm, the ant's solutions are evaluated. The quality of the solution from ant h is judged by the total number of resources, namely, $Q_h = \sum_k r_k$. At the end of the iteration, the pheromone trail is updated according to the quality of individual schedules. Additionally, a certain amount of pheromone evaporates. More specifically, we have

$$\tau_{ij}(t) = \rho \cdot \tau_{ij}(t) + \sum_{h=1}^{m} \Delta\tau_{ij}^{h}(t) \qquad \text{where } 0 < \rho < 1. \tag{2}$$

Here ρ is the evaporation ratio, and

$$\Delta\tau_{ij}^{h} = \begin{cases} Q/Q_h & \text{if } op_i \text{ is scheduled at } j \text{ by ant } h \\ 0 & \text{otherwise} \end{cases}. \tag{3}$$

Q is a fixed constant to control the delivery rate of the pheromone. Two important operations are performed in the pheromone trail updating process. Evaporation is necessary for ACO to effectively explore the solution space, while reinforcement ensures that favorable operation orderings receive a higher volume of pheromone and will have a better chance of being selected in future iterations. The aforementioned process is repeated multiple times until an ending condition is reached. The best result found by the algorithm is reported.

Compared with the FDS method, the proposed algorithm differs in several respects. First, rather than using a one-time constructive approach based on greedy local decisions, the ACO method solves the problem in an evolutionary manner. By using simple local heuristics, it allows individual scheduling results to be generated in a faster manner. With a collection of such individual results and by embedding and adjusting the global heuristics associated with partial solutions, it tries to learn during the searching process. By adopting a stochastic decision making strategy considering both global experience and local heuristics, it tries to balance the efforts of exploration and exploitation in this process. Furthermore, it applies positive feedback to strengthen "good" partial solutions in order to speed-up the convergence. Of course, the negative effect is that it may fall into local minima, thus requires compensation measures such as the one introduced in MMAS. In our experiments, we implemented both the basic ACO and MMAS algorithms. The latter consistently achieves better scheduling results, especially for larger DFGs. A pseudocode implementation of our TCS algorithm using MMAS is shown as Algorithm 2, where the pheromone bounding step is indicated as step 23.

In our study [Wang et al. to appear], compared with widely-used FDS [Paulin and Knight 1987] and list scheduler with various priority heuristics, MMAS-based scheduling algorithms reported better-quality results with up to 19.5% area reductions on the TCS problem and 14.7% reduction in time on the RCS problem. Moreover, the algorithms scale well with regard to different

applications and problem sizes. The runtimes of these algorithms are also reasonably good, especially for the TCS formulation, which runs in the same scale as the FDS method.

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 named *ExpressDFG*. These benchmarks are taken from one of two sources: (1) popular benchmarks used in previous literature; and (2) real-life examples generated and selected from the MediaBench suite [Lee et al. 1997].

The benefit of having classic samples is that they provide a direct comparison between the results generated by our algorithm and those 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. These samples focus mainly on frequently used numeric calculations performed by different applications. However, these samples are typically small to medium in size, and 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:

—real-life testing cases from real-life applications;

—more up-to-date testing cases from modern applications;

—challenging samples for instruction scheduling algorithms with regard to the larger number of operations, higher level of parallelism, and data dependency; and

—a wide range of synthesis problems to test the algorithms' scalability.

For this purpose, 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 [Aigner et al. 2000] and Machine SUIF [Smith and Holloway 2002] tools, and over 14,000 DFGs were extracted as preliminary candidates for our benchmark set. After careful study, 13 DFG samples were selected from 4 MediaBench applications: JPEG, MPEG2, EPIC, and MESA.

Table I lists all 20 benchmarks that were included in our final benchmark set. Together with the names of the various functions from which the basic blocks originated are the number of nodes, number of edges, and instruction depth (assuming unit delay for every instruction) of the DFG. The data, including related statistics, DFG graphs, and source code for all testing benchmarks, is available online [ExpressDFG 2006].

We implemented four different design space exploration algorithms:

(1) FDS: Exhaustively step through the time range by performing time-constrained force-directed scheduling at each deadline.

Table I. ExpressDFG Benchmark Suite

Benchmark Name	# Nodes	# Edges	ID
HAL	11	8	4
horner_bezier†	18	16	8
ARF	28	30	8
motion_vectors†	32	29	6
EWF	34	47	14
FIR2	40	39	11
FIR1	44	43	11
h2v2_smooth_downsample†	51	52	16
feedback_points†	53	50	7
collapse_pyr†	56	73	7
COSINE1	66	76	8
COSINE2	82	91	8
write_bmp_header†	106	88	7
interpolate_aux†	108	104	8
matmul†	109	116	9
idctcol	114	164	16
jpeg_idct_ifast†	122	162	14
jpeg_fdct_islow†	134	169	13
smooth_color_z_triangle†	197	196	11
invert_matrix_general†	333	354	11

Benchmarks with † are extracted from MediaBench.

(2) MMAS-TCS: Step through the time range by performing only MMAS-based TCS scheduling at each deadline.
(3) MMAS-D: Use the iterative approach proposed in Algorithm 1 by switching between MMAS-based RCS and TCS.
(4) FDS-D: This is similar to the MMAS-D, except using FDS-based scheduling algorithms.

We implemented the MMAS-based TCS and RCS algorithms as described in Section 4. Since there is no widely distributed and recognized FDS implementation, we implemented our own. The implementation is based on Paulin and Knight [1987] and has all the applicable refinements proposed in the paper, including multicycle 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 that of reality, however, it is a close enough approximation and does not change the generality of the results. Other operation to 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 $[L_c, 2L_c]$, that is, from the critical path delay to $2x$ of this delay. This results in 263 testing cases in total, and 4 design space exploration experiments are carried out. For the FDS and MMAS-TCS algorithms, we run force-directed

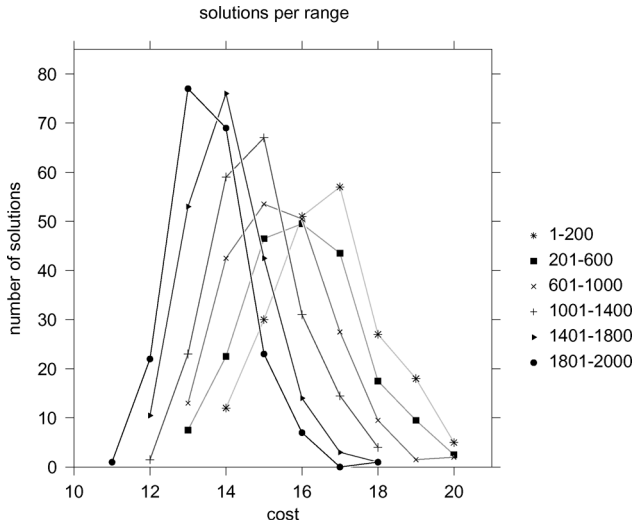

Fig. 2. Distribution of the TCS ACO solution quality on *idctcol* benchmark with a deadline set to its ASAP time. Each line shows a different phase of the algorithm execution where each point gives the number of solutions of a particular resource cost. The line "1–200" denotes the first 200 solutions found by the ACO algorithm, while the line "1801–2000" gives the last 200.

or MMAS-based time-constrained scheduling on every deadline and report the best schedule results together with the costs obtained. For the MMAS-D and FDS-D algorithms, we only run MMAS-based or FDS-based TCS on selected deadlines starting from $2L_c$ and make jumps based on RCS results on the configurations previously obtained by performing TCS.

5.2 Quality Assessment

We first studied the effectiveness of the ACO approach for design space exploration. Two individual tests are carried out: one to verify its performance on the TCS problem with a specific deadline, and the other tries to confirm its performance over the entire design space.

In the first tests, MMAS-based TCS is performed on the *idctcol* benchmark, an implementation of inverse discrete cosine transform, with the deadline set to its ASAP time of 19. We use 10 ants for each iteration, which provides 10 individual scheduling solutions. The total iteration limit is set to 200, which produces a total of 2000 scheduling results for this TCS problem. We want to examine the effectiveness of the algorithm, in other words, how does the quality of the solutions improve across iterations? Figure 2 shows this result by plotting the solution quality/frequency curves over time. Here each curve aggregates solutions found within certain iterations. For example, the curve labeled "1-200" diagrams the quality distribution for the first 200 scheduling results obtained in the first 20 iterations. The x-axis is the hardware cost for the schedule results, where we simply use resource number counts. The y-axis shows the number of solutions that the iteration range produces at each specific cost.

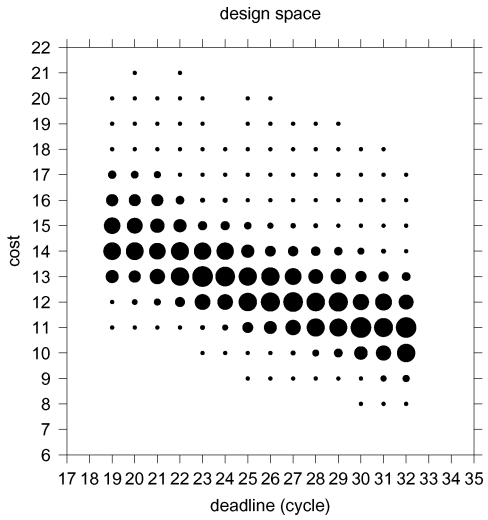

Fig. 3. Solution quality of the TCS ACO on the *idctcol* benchmark. We run the TCS ACO algorithm at each deadline ranging from its ASAP time of 19 to 32. The size of dot indicates the proportion of solutions with a specific resource cost found at each deadline.

From this graph, we can easily see that MMAS-based TCS is working. For example, comparing the initial (1–200) and final 200 solutions (1801–2000). In the initial 200 solutions, there are 5 with an area of 20, and the best have area of 14 (there are 12 such); by the last 200 solutions, there are 0 with an area of 20, 69 with an area of 14, and 1 with an area of 11. As the algorithm progresses, a positive trend emerges where the ants ignore the worst solutions and enforce the better ones.

To show the effectiveness of the algorithm over the whole design space, similar experiments are conducted across the range of deadlines of interest. Figure 3 gives one example of the *idctcol* benchmark on deadlines from 19 to 32, where the x-axis is the deadline constraint and y-axis the cost for scheduling results. The size of dots is proportional to the number of schedule results that the ants produce for a specific cost and deadline. It is easy to see that the focus area of the algorithm adjusts as the constraints change. Moreover, if we inspect each column more carefully, we can see that the algorithm effectively explores the "best" part of the design space. This is evidenced by the movement of dense area in the graph and the relatively invariant vertical spread.

We performed experiments on each benchmark using the four different design space exploration algorithms. First, time-constrained FDS scheduling is used at every deadline. The quality of results is used as the baseline for quality assessment. Then MMAS-TCS, MMAS-D, and FDS-D algorithms are executed; the difference is that MMAS-TCS steps through the design space in the same way as FDS, while MMAS-D, and FDS-D utilize the duality between TCS and RCS. Because of their randomized nature, MMAS-TCS and MMAS-D algorithms are executed five times in order to obtain enough statistics to evaluate their stability.

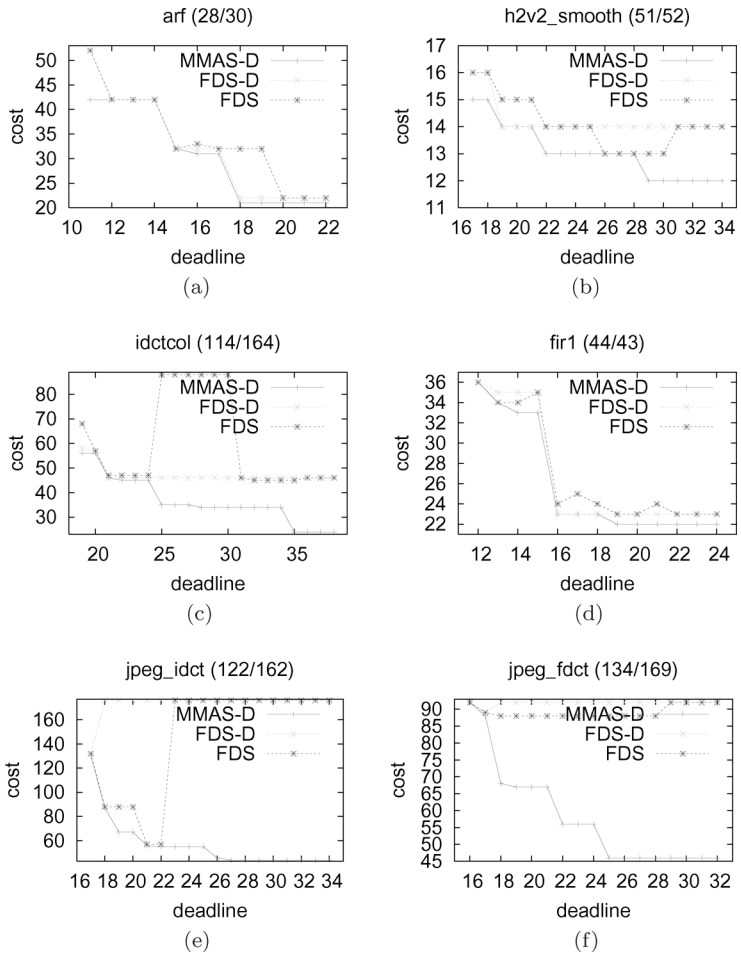

Fig. 4. Design space exploration results: MMAS-D, FDS-D, and FDS.

Detailed design space exploration results for six of the benchmarks are shown in Figure 4, where we compare the curves obtained by MMAS-D, FDS-D, and FDS algorithms. Table II summarizes the experiment results. For each benchmark we give the node/edge count, and the average resource savings of the FDS-D, MMAS-TCS, and MMAS-D algorithms compared with FDS. We report the savings in percentage of total resource counts (a negative result indicates a lower (better) resource cost). We weight the two resource types M and A equally, though we use different cost weights to bias alternative solutions (e.g., solution (3M, 4A) is more favorable than (4M, 3A) as resource M has a large cost weight). We could easily vary the relative costs and number of resources types. However, we feel this would introduce confusion caused by different weight choices. The percentage savings is computed for every deadline of every benchmark. The average for a certain benchmark is reported in Table II. It is easy to see that MMAS-TCS and MMAS-D both outperform the classic FDS method across the

Table II. Summary for Design Space Exploration Results

Name	Nodes/Edges	Deadline	FDS-D	MMAS-TCS	MMAS-D
HAL	11/8	(6–12)	14.3%	−7.1%	−7.1%
hbsurf	18/16	(11–22)	0.0%	−9.9%	−13.2%
ARF	28/30	(11–22)	−4.7%	−12.4%	−18.6%
motionvectors	32/29	(7–14)	−8.0%	−13.1%	−16.0%
EWF	34/47	(17–34)	−5.6%	−11.5%	−21.9%
FIR2	40/39	(12–24)	−4.1%	−16.8%	−22.8%
FIR1	44/43	(12–24)	−3.9%	−15.2%	−18.0%
h2v2_smooth	51/52	(17–34)	4.2%	−19.3%	−20.5%
feedbackpoints	53/50	(11–22)	−1.2%	−5.9%	−9.1%
collapsepyr	56/73	(8–16)	−5.3%	−18.3%	−20.0%
COSINE1	66/76	(10–20)	−3.1%	−21.5%	−23.5%
COSINE2	82/91	(10–20)	0.7%	−5.6%	−8.1%
wbmpheader	106/88	(8–16)	−2.4%	−0.9%	−1.6%
interpolate	108/104	(10–20)	−2.3%	−0.2%	−1.8%
matmul	109/116	(11–22)	−4.7%	−3.7%	−5.6%
idctcol	114/164	(19–38)	−11.2%	−30.7%	−32.0%
jpegidctifast	122/162	(17–34)	35.2%	−50.3%	−52.1%
jpegfdctislow	134/169	(16–32)	16.2%	−31.4%	−34.6%
smoothcolor	197/196	(15–30)	−4.6%	−7.3%	−8.6%
invertmatrix	333/354	(15–30)	0.0%	−11.2%	−11.9%
Total Avg.			0.48%	−16.4%	−17.3%

Each line gives the benchmark name, the tested time range, and the results of each design space exploration algorithm (FDS-D, MMAS-TCS, MMAS-D compared to the exhaustive FDS result). A negative result indicates a smaller resource allocation, which is desired.

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. On the other hand, FDS-D algorithms have a minor performance degradation compared with the FDS baseline. It outperforms FDS in 14 out of the 20 benchmarks, gives worse results on 4 samples, and shows no change on 2. Though it provides modestly better results over two testing samples (i.e., *wbmpheader* and *interpolate*) when compared to MMAS-D, overall MMAS-D produces a much better result. Finally, FDS-D is much less stable with regard to result quality. It seems to be more application dependent and yields bad results in certain cases (e.g., benchmark *jpegidictifast*).

It is interesting and initially surprising to observe that MMAS-D never had worse performance than MMAS-TCS. More careful inspection of the experiments reveals the reason: Using the duality between TCS and RCS not only reduces computation time, but can also improve the quality of the result. To understand this, recall Theorem 3.3 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 the MMAS-TCS algorithm can provide worse-quality solutions on deadlines between t_1 and t_2.

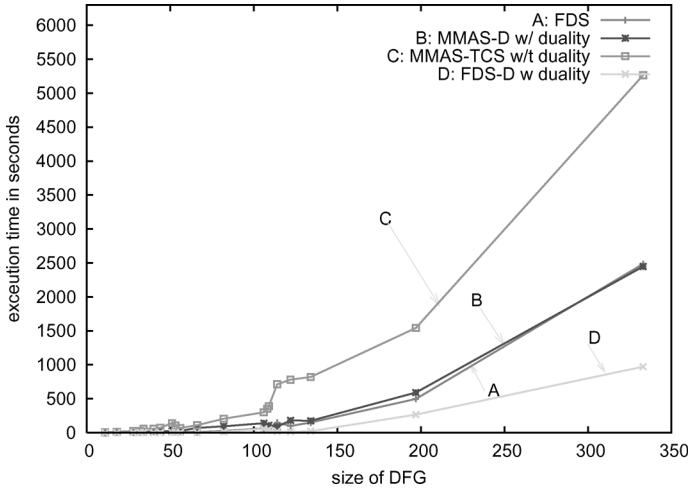

Fig. 5. Timing performance comparison.

This benefit is not specifically associated with MMAS scheduling algorithms, but rather, is also observed when other scheduling methods are used. For example, consider the curve generated by FDS-D in Figure 4(d). We can see that the configuration provided by TCS at deadline 24 can be pushed to deadline 16. FDS-D achieves better results over FDS at time stamps of 16, 17, 18, and 21. However, we will not always obtain this benefit. For the same curve, FDS-D actually suffers worse results at times 13 and 14. Extreme examples of this are shown in Figures 4(e) and 4(f), the two worst samples for FDS-D. It is easy to realize that if a generous TCS result is generated at a bigger deadline, the following RCS step is misled to provide a very small deadline result. The effect is that the algorithm provides a poor tradeoff curve.

In conclusion, the proposed duality-based design space exploration framework is a general approach and can be combined with any scheduling algorithm. However, the selection of such scheduling algorithms has a direct impact on the quality of the resulting tradeoff curves. This is not surprising in light of our discussion on Theorem 3.3 in Section 3.

Figure 5 diagrams the average execution time comparison for the four design space exploration approaches, ordered by size of benchmark. All of the experiment results use the same Linux box with a 2GHz CPU. It is easy to see that the all algorithms have similar runtime scale, where MMAS-TCS takes more time, while MMAS-D and FDS have very close runtimes, especially on larger benchmarks. The major execution time savings come from the fact that MMAS-D exploits the duality and only computes TCS on a selected number of deadlines. In over 263 testing cases, we find that on average MMAS-D skips about 44% of the 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 time- and resource-constrained scheduling problems and exploits their duality. Our algorithm provides a time/cost tradeoff curve in the design space in a systemic manner. We proved that it is possible to use the duality to help us effectively construct such a curve, while reducing the computing time and improving the quality of the result. The proposed method is general and can be combined with any high-quality scheduling algorithm. However, the underlying scheduling algorithm has direct impact on the quality of the tradeoffs curve. We showed that ACO-based scheduling algorithms are ideal due to their robustness, high performance, reasonable execution time, and the capability of providing multiple scheduling candidates. Our algorithms outperformed the popularly used force-directed scheduling method with significant savings (average 17.3% savings on resource counts) and almost the same runtime on comprehensive benchmarks constructed with classic and real-life samples. The algorithms also scaled well over different applications and problem sizes.

REFERENCES

AIGNER, G., DIWAN, A., HEINE, D. L., MOORE, M. S. L. D. L., MURPHY, B. R., AND SAPUNTZAKIS, C. 2000. *The Basic SUIF Programming Guide*. Computer Systems Laboratory, Stanford University Stanford, CA.

CHAUDHURI, S., BLYTHE, S. A., AND WALKER, R. A. 1997. 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.

CORNE, D., DORIGO, M., AND GLOVER, F., EDS. 1999. *New Ideas in Optimization*. McGraw Hill, London.

DICK, R. P. AND JHA, N. K. 1997. MOGAC: A multiobjective genetic algorithm for the co-synthesis of hardware-software embedded systems. In *Proceedings of the IEEE/ACM Conference on Computer Aided Design*, 522–529.

DORIGO, M., MANIEZZO, V., AND COLORNI, A. 1996. Ant system: Optimization by a colony of cooperating agents. *IEEE Trans. Syst. Man Cybern. Part-B 26*, 1 (Feb.), 29–41.

DUTTA, R., ROY, J., AND VEMURI, R. 1992. Distributed design-space exploration for high-level synthesis systems. In *Proceedings of the Design and Automation Conference (DAC)*. IEEE Computer Society Press, Los Alamitos, CA, 644–650.

EXPRESSDFG. 2006. ExpressDFG benchmark website. http://express.ece.ucsb.edu/benchmark/.

GUTJAHR, W. J. 2002. ACO algorithms with guaranteed convergence to the optimal solution. *Inf. Process. Lett. 82*, 3, 145–153.

HEIJLIGERS, M. J. M., CLUITMANS, L. J. M., AND JESS, J. A. G. 1995. High-Level synthesis scheduling and allocation using genetic algorithms. In *Proceedings of the EDA Technofair Design Automation Conference/Asia and South Pacific Design Automation Conference* (Makuhari, Massa, Chiba, Japan). Article 11.

LEE, C., POTKONJAK, M., AND MANGIONE-SMITH, W. H. 1997. MediaBench: A tool for evaluating and synthesizing multimedia and communicatons systems. In *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture*.

LIN, Y.-L. 1997. Recent developments in high-level synthesis. *ACM Trans. Des. Autom. Electron. Syst. 2*, 1, 2–21.

MADSEN, J., GRODE, J., KNUDSEN, P. V., PETERSEN, M. E., AND HAXTHAUSEN, A. 1997. LYCOS: The Lyngby co-synthesis system. *Des. Autom. Embedded Syst. 2*, 2 (Mar.), 125–63.

MCFARLAND, M., PARKER, A. C., AND CAMPOSANO, R. 1990. The high-level synthesis of digital systems. In *Proc. IEEE. 78*, 301–318.

PALESI, M. AND GIVARGIS, T. 2002. Multi-Objective design space exploration using geneticalgorithms. In *Proceedings of the 10th International Symposium on Hardware/Software Codesign*.

PAULIN, P. G. AND KNIGHT, J. P. 1987. Force-Directed scheduling in automatic data path synthesis. In *Proceedings of the 24th ACM/IEEE Conference on Design Automation Conference*.

SMITH, M. D. AND HOLLOWAY, G. 2002. *An Introduction to Machine SUIF and Its Portable Libraries for Analysis and Optimization*. Division of Engineering and Applied Sciences, Harvard University.

STÜTZLE, T. AND HOOS, H. H. 2000. Max-Min ant system. *Future Gen. Comput. Syst. 16*, 9 (Sept.), 889–914.

WANG, G., GONG, W., DERENZI, B., AND KASTNER, R. Ant colony optimizations for resource and timing constrained operation scheduling. *IEEE Trans. Comput.-Aided Des* (to appear).

WANG, G., GONG, W., AND KASTNER, R. 2003. A new approach for task level computational resourcebi-partitioning. In *Proceedings of the 15th International Conference on Parallel and Distributed Computingand Systems 1*, 1 (Nov.), 439–444.

WANG, G., GONG, W., AND KASTNER, R. 2004. System level partitioning for programmable platforms using the antcolony optimization. In *Proceedings of the 13th International Workshop on Logic and Synthesis (IWLS)*.

WANG, G., GONG, W., AND KASTNER, R. 2005. Instruction scheduling using Max-Min ant optimization. In *15th ACM Great Lakes Symposium on VLSI (GLSVLSI)*.

WILKEN, K., LIU, J., AND HEFFERNAN, M. 2000. Optimal instruction scheduling using integer programming. In *Proceedings of the ACM SIGPLAN Conference on Programming Language design and Implementation*.

Received June 2006; revised March 2007; accepted March 2007