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 of 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 these 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.

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

To effectively reach a decision, designers must somehow reason about the tradeoffs amongst a set of parameters. Such decisions are often made based on experience (i.e. this worked before, it should work again) or back of the envelope calculations. Exploration tools that can quickly survey the design space and report a variety of options are invaluable.

Resource allocation and scheduling are two fundamental high level synthesis problems. 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. In other words, a different resource allocation will likely produce a vastly different scheduling result.

On the other hand, 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 on the order that we should perform scheduling and allocation.

One possible 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 is 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 this relationship between these seemingly isolated points. Unfortunately, designers are 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 scheduling tools? When do we stop the exploration?

Moreover, due to the lack of connection amongst the traditional methods, there is a little information shared between time constrained and resource constrained solutions. This is unfortunate, as we are essentially throwing away potential solutions since solving one problem should offer more insight to the other problem. The reason for such a lack is that these methods don’t utilize the same framework and often employ different baseline models for solving the corresponding problems.

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.

The paper 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
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) [17], 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 [7, 11], genetic algorithms are implemented for design space exploration. In [6], the authors concentrate on providing alternative module bags for design space exploration by heuristically solving clique partitioning problems and using a force directed list scheduler. In the Voyager system [3], 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 [10] also find their applications in this domain. A survey on design space exploration methodologies can be found in [9] and [2].

Amongst the existing approaches, the most popular method is perhaps the force directed scheduling (FDS) algorithm [12], 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 handling. 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

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 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 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 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. 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, where both solutions have the same cost (20).

Studying the design space more carefully, reveals several key observations. First, the achievable deadlines are limited to the range \([t_{\text{asap}}, t_{\text{seq}}]\), where \( t_{\text{asap}} \) is the ASAP time for the application while \( t_{\text{seq}} \) 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 solution with a deadline beyond that of \( t_{\text{seq}} \) are not Pareto optimal. Furthermore, for each specific configuration we have the following lemma about the portion of the design space that it maps to.

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 \((t_{\text{min}}, c)\), where \( t_{\text{min}} \) is the resource constrained minimum scheduling time.

The proof of the lemma is straightforward as each feasible configuration has a minimum execution time \( t_{\text{min}} \) for the application, and obviously it can handle every deadline longer than \( t_{\text{min}} \). For example, in Figure 1(a), if the configuration \((m_1, a_1)\) has a cost \( c_1 \) and a 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 a bigger minimum scheduling time \( t_2 \). In this case, their feasible space overlaps after \((t_2, c_2)\).

As we discussed before, 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 finding the minimum area \( c \) amongst all the configurations that are capable of producing \( t \in [t_{\text{asap}}, t_{\text{seq}}] \). In other words, we can find these points 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 \( L \):

Lemma 3.2 Curve \( L \) is monotonically non-increasing as the deadline \( t \) increases.

Proof. Assume the lemma is false. Therefore, we will have two points \((c_1, t_1)\) and \((c_2, t_2)\) 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 the configurations with cost \( c \). Moreover, because the 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 3.3 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 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 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 \( t_1 \) 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 RCS 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 RCS and TCS 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 the RCS algorithm may not return the true optimal and could be worse than \( t_{\text{cur}} \).

By iteratively using the RCS and TCS algorithms, we can quickly step across the design space. Compared to using RCS or TCS alone, the above algorithm provides benefits in runtime by leveraging the horizontal segments in \( L \). To realize this, we perform exploration starting from the largest deadline \( t_{\text{asap}} \). 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 it provides a better opportunity to make large horizontal jumps. On the other hand, TCS
procedure DSE
output: curve $L$
1: interested time range $[t_{\text{min}}, t_{\text{max}}]$, where $t_{\text{min}} \geq t_{\text{asap}}$ and $t_{\text{max}} \leq t_{\text{seq}}$.
2: $L = \emptyset$
3: $t_{\text{cur}} = t_{\text{max}}$
4: while $t_{\text{cur}} \neq t_{\text{min}}$
5: perform TCS on $t_{\text{cur}}$ to obtain the optimal configurations $C_i$.
6: for configuration $C_i$ do
7: perform RCS to obtain the minimum time $t_{\text{rcs}}$
8: end for
9: $t_{\text{rcs}} = \min_i (t_{\text{rcs}})^*$ find the best rcs time $^*/$
10: $t_{\text{cur}} = \min (t_{\text{cur}}, t_{\text{rcs}}) - 1$
11: extend $L$ based on TCS and RCS results
12: end while
13: return $L$

Algorithm 1: Iterative Design Space Exploration Algorithm

algorithms take more time and provide poor solutions when the deadline is loose i.e the slack of the operations is large. By performing bigger jumps at the larger deadlines, we can gain big runtime savings.

In order for the above algorithm to work in practice, the TCS and RCS algorithms used in the process require special characteristics. First, they have to be fast which is generally requested for any design space exploration tool. Second, they have to be capable of providing close to optimal solutions, especially for the TCS algorithm. Otherwise, the conditions for Theorem 3.3 will not be satisfied and the generated curve $L$ will suffer in quality.

Moreover, notice that we will only enjoy the biggest jumps when we take the minimum RCS result amongst all the configurations that provide the minimum cost for the TCS problem. This is reflected in step 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 they have different deadline limits. Therefore a good TCS algorithm used in the proposed approach should be able provide multiple candidate solutions with the same minimum cost, if not all of them.

4. ANT COLONY OPTIMIZATIONS FOR TIME AND RESOURCE CONSTRAINED SCHEDULING

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 [12], various list scheduling heuristics, and the recently proposed ant colony optimization (ACO) based instruction scheduling algorithms [16]. 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:

- ACO-based scheduling algorithm generated better quality scheduling results that are close to the optimal with good stability for both the TCS and RCS problems [16].
- ACO-based scheduling method 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. As we have discussed previously, 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. [5]. It is a cooperative heuristic searching algorithm inspired by ethological studies on the behavior of ants. 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 the 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 will choose the shortest trails, which in turn creates an even larger amount of pheromone on the short trails, making these trails more attractive to the 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 the pheromone trails. One of the first problems which ACO was successfully applied was the Traveling Salesman Problem (TSP) [5], for which it gave competitive results compared to traditional methods. Researchers have since formulated ACO methods for a variety of traditional NP-hard problems. These problems include the maximum clique problem, the quadratic assignment problem, the
graph coloring problem, the shortest common super-sequence problem, and the multiple knapsack problem. ACO also has been applied to practical problems such as the vehicle routing problem, data mining, network routing problem and the system level task partitioning problem [4, 14, 15].

Wang et al [16] 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 their algorithms employ a collection of agents that collaborate to 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.

For example, in their TCS formulation, each instruction or DFG node \( op_i \) is associated with \( D \) pheromone trails \( \tau_{ij} \), where \( j = 1, \ldots, D \) and \( D \) is the specified deadline. These pheromone trails indicate the global favorableness of assigning the \( i \)-th operation at the \( j \)-th control step in order to minimize the resource cost with respect to the time constraint.

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 scheduling decision probabilistically. Once an instruction \( op_i \) is selected, the ants need to make a decision on which control step it should be assigned. This decision is also made probabilistically as illustrated in Equation (1).

\[
\begin{align*}
\rho_{hj} &= \begin{cases} 
\frac{\tau_{hj}(t)^{\alpha} \cdot \eta_{hj}^{\beta}}{\sum_l (\tau_{hl}(t)^{\alpha} \cdot \eta_{hl}^{\beta})} & \text{if } op_i \text{ can be scheduled at } l \text{ and } j \\
0 & \text{otherwise}
\end{cases} \\
&= \begin{cases} 
\frac{\tau_{hj}(t)^{\alpha} \cdot \eta_{hj}^{\beta}}{\sum_l (\tau_{hl}(t)^{\alpha} \cdot \eta_{hl}^{\beta})} & \text{if } op_i \text{ can be scheduled at } l \text{ and } j \\
0 & \text{otherwise}
\end{cases} 
\end{align*}
\]

Here \( j \) is the time step under consideration. The item \( \eta_{hj} \) is the local heuristic for scheduling operation \( op_i \) at control step \( j \), and \( \alpha \) and \( \beta \) are parameters to control the relative influence of the distributed global heuristic \( \tau_{hj} \) and local heuristic \( \eta_{hj} \). Assuming \( op_i \) is of type \( k \), \( \eta_{hj} \) to simply set to be the inverse of the distribution graph value. 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.

Once all the ants have found their scheduling solution, the pheromone trails are updated in a similar way as in classic ACO method reported in [5] by going through the evaporation and reinforcement processes. Evaporation is necessary for AS to effectively explore the solution space, while reinforcement ensures that the favorable instruction orderings receive a higher volume of pheromone and will have a better chance of being selected in the future iterations. The above process is repeated multiple times until an ending condition is reached. The best result found by the algorithm is reported.

In their study [16], the authors compared the proposed ACO-based algorithms with widely used FDS and list schedulers and reported better quality results with up to 19.5% area reductions on TCS problem and 14.7% reduction in time on RCS problem. Moreover, the algorithms scale well with regard to different applications and problem sizes. The runtime 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. 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 [8].

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. These samples focus mainly on frequently used numeric calculations performed by different applications, such as auto regression filter (the ARF benchmark) and discrete cosine transform (the COSINE1 and COSINE2 benchmarks).

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 SUIF [13] tools, and over 14,000 DFGs were extracted as preliminary candidates for our benchmark set. After careful study, 13 samples were selected from four MediaBench applications, 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 number of nodes/edges of the graph. The data, including related statistics, DFG graphs and source code for the all testing benchmarks, 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.
3. MMAS-D: use the iterative approach proposed in Algorithm 1 by switching between MMAS-based RCS and TCS.

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 [12] 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 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...
5.2 Quality Assessment

In our experiments, we first studied the effectiveness of the ACO approach for design space exploration. Two individual tests are carried out. One verifies its performance on TCS problem with a specific deadline, while the other tries to confirm its performance over the interested design space.

In the first tests, MMAS-based TCS is performed on the idctcol benchmark, an implementation of inverse discrete cosine transform, with deadline set to its ASAP time 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(a) 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, while the y-axis shows the number of solutions that iteration range produces at each specific cost.

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

To show the effectiveness of the algorithm over the whole design space, similar experiments are conducted across the range of interested deadlines. Figure 2(b) gives one example on the idctcol benchmark on deadlines from 19 to 32, where the x-axis is the deadline constraint and y-axis is the cost for scheduling results. The size of dots are proportional to the number of schedule results that the ants produce for the specific cost and deadline. It is easy to see that the focus area of algorithm adjusts pretty well 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 more often. This is evidenced by the movement of the dense area in the graph and the relatively invariant vertical spread.

**Table 1: Result Summary for Design Space Exploration**

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>Deadline</th>
<th>MMAS-TCS</th>
<th>MMAS-D</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL</td>
<td>11/8</td>
<td>(6 - 12)</td>
<td>-7.1%</td>
<td>-7.1%</td>
</tr>
<tr>
<td>hbsurf†</td>
<td>18/16</td>
<td>(11 - 22)</td>
<td>-9.9%</td>
<td>-13.2%</td>
</tr>
<tr>
<td>ARF</td>
<td>28/30</td>
<td>(11 - 22)</td>
<td>-12.4%</td>
<td>-18.6%</td>
</tr>
<tr>
<td>motionvectors†</td>
<td>32/29</td>
<td>(7 - 14)</td>
<td>-13.1%</td>
<td>-16.0%</td>
</tr>
<tr>
<td>EWF</td>
<td>34/47</td>
<td>(17 - 34)</td>
<td>-11.5%</td>
<td>-21.9%</td>
</tr>
<tr>
<td>FIR2</td>
<td>40/39</td>
<td>(12 - 24)</td>
<td>-16.8%</td>
<td>-22.8%</td>
</tr>
<tr>
<td>FIRI</td>
<td>44/43</td>
<td>(12 - 24)</td>
<td>-15.2%</td>
<td>-18.0%</td>
</tr>
<tr>
<td>h2v2smooth†</td>
<td>51/52</td>
<td>(17 - 34)</td>
<td>-19.3%</td>
<td>-20.5%</td>
</tr>
<tr>
<td>feedbackpoints†</td>
<td>53/50</td>
<td>(11 - 22)</td>
<td>-5.9%</td>
<td>-9.1%</td>
</tr>
<tr>
<td>collapsepyr†</td>
<td>56/73</td>
<td>(8 - 16)</td>
<td>-18.3%</td>
<td>-20.0%</td>
</tr>
<tr>
<td>COSINE1</td>
<td>66/76</td>
<td>(10 - 20)</td>
<td>-21.5%</td>
<td>-23.5%</td>
</tr>
<tr>
<td>COSINE2</td>
<td>82/91</td>
<td>(10 - 20)</td>
<td>-5.6%</td>
<td>-8.1%</td>
</tr>
<tr>
<td>wbmpheader†</td>
<td>106/88</td>
<td>(8 - 16)</td>
<td>-0.9%</td>
<td>-1.6%</td>
</tr>
<tr>
<td>interpolate†</td>
<td>108/104</td>
<td>(10 - 20)</td>
<td>-0.2%</td>
<td>-1.8%</td>
</tr>
<tr>
<td>matmul†</td>
<td>109/116</td>
<td>(11 - 22)</td>
<td>-3.7%</td>
<td>-5.6%</td>
</tr>
<tr>
<td>idctcol†</td>
<td>114/164</td>
<td>(19 - 38)</td>
<td>-30.7%</td>
<td>-32.0%</td>
</tr>
<tr>
<td>jpegidctifast†</td>
<td>122/162</td>
<td>(17 - 34)</td>
<td>-50.3%</td>
<td>-52.1%</td>
</tr>
<tr>
<td>jpegidctislow†</td>
<td>134/169</td>
<td>(16 - 32)</td>
<td>-31.4%</td>
<td>-34.6%</td>
</tr>
<tr>
<td>smoothcolor†</td>
<td>197/196</td>
<td>(15 - 30)</td>
<td>-7.3%</td>
<td>-8.6%</td>
</tr>
<tr>
<td>invertmatrix†</td>
<td>333/354</td>
<td>(15 - 30)</td>
<td>-11.2%</td>
<td>-11.9%</td>
</tr>
</tbody>
</table>

Table 1 shows the tested range. Benchmarks with † are extract from MediaBench. Saving is computed based on FDS results. No weight applied.

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 3, 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 savings is computed for every deadline used for each benchmark, 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.

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 3.3 and Figure 1(b). If we achieve an optimal solution at t1, with MMAS-D we automatically extend this optimality from t1 to t2, while an unperfect MMAS-TCS still have chance to provide worse quality solutions on deadlines between t1 and t2.

Figure 3: Design Space Exploration: MMAS-D vs. FDS

Figure 4: Timing Performance Comparison

All of the experiment results are obtained on a Linux box with a 2GHz CPU. Figure 4 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 larger benchmarks. The major execution time savings comes 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. Our algorithm provided superior cost/deadline tradeoff curve in the design space. We proved that it is possible to use the duality to help us effectively construct the optimal tradeoff curve, not only with a reduced computing time but also improved the results’ quality. We identified ACO-based scheduling algorithms to be used in our approach for its robustness, high quality results, reasonable execution time and the capability of providing multiple alternative result candidates. Combined with the MAX-MIN Ant System, our algorithms outperformed the popularly used force directed scheduling method with significant savings (average 17.3% savings on resource counts) and almost the same run time on comprehensive benchmarks constructed with classic and real-life samples. The algorithms also scaled well over different application and problem sizes.

7. REFERENCES

[1] citation removed to maintain author confidentiality for blind review.