Design Space Exploration for Data Path Synthesis

C.A.Mandal[1] *P.P.Chakrabarti*[2] *S.Ghose*[2]

[1] Department of Computer Science & Engineering
Jadavpur University, Calcutta 700032, INDIA
[2] Department of Computer Science & Engineering
Indian Institute of Technology , Kharagpur 721302, INDIA

Abstract

In this paper we examine the multi-criteria optimization involved in scheduling for data path synthesis (DPS). We present a method to find non-dominated schedules using a combination of restricted search and heuristic scheduling techniques. Our method supports design with architectural constraints such as the total number of functional units, buses, etc. The schedules produced have been taken to completion using GA-BIND [3] and the results are promising.

1 Introduction

Data path synthesis (DPS) involves first scheduling of operations, and then allocation and binding of abstract design entities to their physical counterparts. At the end of DPS we are required to find one or more "optimized" implementations for a design problem input to the synthesis system. We seek to optimize not only the area cost estimate of the data path but also its performance, measured as a function of the length of the schedule. This makes the synthesis problem a multi-criteria optimization problem.

A feature of most multi-criteria optimization problems is that the criteria are often non-commensurate and sometimes conflicting. It is therefore difficult to combine the criteria into a single cost function. We take the approach of representing the cost of a design as a tuple of costs of the individual objectives. This is similar to the approach taken in Stewart et al. [7]. *One cost tuple is said to be better than another distinct tuple if the cost of each criterion of the first tuple is no worse than the corresponding costs of the other tuple. A design whose cost tuple is better than that of another design is said to dominate that design. The global problem of optimization is to find the set of designs which are not dominated by any other designs.* The set of feasible designs satisfying the design parameters constitute the design space. Each design point in the design space corresponds to an estimate of hardware requirement and performance computed as a function of the schedule time. Thus an algorithm for DPS needs to consider techniques not only for scheduling and allocation but also for a systematic exploration of the design space to locate these non-dominated designs. The starting point of design space exploration often revolves round the basic scheduling problem. In our work, we do design space exploration (DSE) using a combination of controlled search, and heuristic scheduling techniques.

We employ a multi-objective search approach to perform design space exploration and scheduling. In our scheme we have a state space generation mechanism coupled with an estimator for obtaining various <hardware cost, performance> estimates. A controlled depth first branch and bound is used to determine the hardware cost estimate and produce a partial schedule for a given time constraint. This actually corresponds to a localized exact or near exact exploration of a region of the entire design space. In order to contain the combinatorial explosion, the computational effort to be spent on DSE can be controlled by certain parameters.

At the heart of the DSE mechanism is the controlled search based Resource Estimation and Partial Scheduling (REPS) algorithm. The basic DSE technique makes use of the REPS algorithm to estimate the hardware requirement, as tightly as possible, so that the design parameters are also satisfied. REPS also returns a partial or complete schedule depending on the situation. The design points obtained may be approximate (lower bounds) and the schedules may be partial in the sense that the degree of freedom of some operation may still be more than one. To meet this situation a local DSE mechanism has been developed to explore the neighborhood of such a design point to obtain one or more non-dominated design points for

0-8186-7755-4/96 $05.00 © 1996 IEEE

which feasible schedules will exist.

In the following we present details of our solution to the problem of design space exploration (DSE) to generate a set of schedules which will represent non-dominated designs. The inputs for design space exploration are explained in the next section. The estimates used by REPS for hardware cost and schedule time are discussed in section 3. REPS itself is presented in section 4 Then the overall DSE mechanism (which uses REPS) is explained in section 5. The experimental results are presented in section 6.

2 Inputs to DSE

1. Operation Precedences.
2. Design Parameters: Our design space exploration (DSE) scheme uses the following parameters.

   NFUS This indicates the number of sites where hardware operators will be clustered. However, FUs need not be formed during scheduling.

   NBUS This is the maximum number of logically distinct buses in the system. Presently, each communication path between units is abstracted as a bus. In our model of implementation two or more units are connected to each main bus. The connection may be switched or direct.

   NVREF This is the maximum number of distinct variable references permitted in any time step.

   Though the above parameters are independent they are well co-related.

3 Measures for DSE

We shall often have to consider a region of the design space and efficiently determine the design point or points in this region which will be feasible and worth retaining. In general we shall have to resort to scheduling to answer this question. However, scheduling could be computationally intensive. We, shall therefore rely on heuristic measures not only to aid scheduling but also to arrive at our decision as early as possible.

3.1 Estimation of Resources for Specific Operations

Given a DAG, we would like to estimate the number of each hardware operator for realizing each kind of operation to schedule the DAG in (say) n time steps. This estimate can be obtained as a lower bound. The method of determining the lower bound is similar to the techniques proposed in [6, 1] using the concept of *windows*.

3.2 Estimation on the Total Number of Operations per Time Step

This metric is required to ensure that the parameter *NFUS* is not violated. This metric is found using an extension of the method mentioned above, for the previous metric. Only, in this case, no distinction is made between the different types of operations, and all the operations occurring in a window are counted.

3.3 Estimation for Buses

The bus requirement is estimated by examining the transfers that take place in various windows. Each operand of an operation contributes to a transfer. Transfers also arise due to variable assignments. As usual we consider the transfers that will be restricted within the window under consideration and then compute the lower bound on the number of concurrent transfers. Common variables which form inputs to operations need to be handled carefully. For the purpose of computing a lower bound transfers arising from the same variable to operations which are neither ancestors or descendents of one another may be counted only once, otherwise they may be considered distinct.

3.4 Estimation for Variable Accesses

The number of distinct variable accesses is determined by examining the variable accesses that take place in various windows. Each input and output operand of an operation contributes to a variable access. As usual we consider the accesses that will be restricted within the window under consideration to compute the lower bound.

The above estimation methods are applicable to individual DAG's. For multiple DAG's these estimators need to be applied to each of those DAG's. The global l.b. is obtained by merging the individual l.b.'s.

4 Search Algorithm for Resource Estimation and Partial Scheduling

The resource estimation and partial scheduling algorithm uses the estimators described in the previous section to determine the resource cost for a given schedule time. It also returns a complete schedule if required. This requirement is controlled by a threshold. If the threshold W is set to one then it returns a complete schedule. If $W > 1$ then it returns a partial schedule in the sense that the degrees of freedoms (DOF) of all operations are suitably reduced but some may still have non-zero DOF. It is, however, ensured that the DOF of all operations will be less than W. Thus the REPS algorithm partitions the DAG, if necessary, into smaller DAG's, applies the estimator to these partitions and combines the estimates for the different partitions to arrive at the final estimate. The schedules of partitions are combined to return the partial schedule obtained.

4.1 The Search Scheme

The partitioning scheme requires splitting the n time steps, in which to schedule (a partition of) the DAG, if $n > W$, into bands each of at most W time steps. Each operation of the DAG is restricted to lie in only one of these bands. For operations whose, ASAP and ALAP times, t_a and t_o, lie within a band, nothing needs to be done. For other operations it is necessary to take a decision regarding the band where it should be restricted to be scheduled. A poor decision regarding the band where the operation should be placed could give rise to a high and sub-optimal resource cost estimate. A search must, therefore, be conducted on the DAG to take the right set of decisions. We have employed a depth first branch and bound scheme. The process of decomposition is done recursively till the size of none of the partitions of the current DAG are more than W. We now explain the search mechanism.

The memory requirement for storing the partial solutions is high. Thus we have chosen depth first search branch and bound (DFBB) since its memory requirement is minimal. In the search scheme the partitioned DAG's are treated like separate DAG's. If the number of time steps within which the DAG needs to be scheduled does not exceed W then no more repartitioning is done and the current estimates are accepted. Otherwise, it is split into two smaller DAG's. The splitting is done near about the middle so that the two sub-problems generated are of similar size. If there are one or more operations crossing the boundary then all the possibilities of distributing these operations need to be tested out. This is where the search comes in. We perform the search by explicit backtracking. In order to keep track of the moves a stack is used. For an operation that crosses the partition boundary there are three moves to be made: i) it has to be scheduled in the top half, ii) it has to be scheduled in the lower half and iii) its original freedom has to be restored. After making a move the ASAP and ALAP schedules and the resource estimates are recomputed. If this estimates exceed the estimates of the best design found so far, then the current move is rejected and backtracking is initiated. Move rejection followed by backtracking also takes place if the resource estimate is found to be infeasible with respect to the design parameters.

When partitioning is done, it becomes necessary to handle multiple basic blocks. A list is used to handle these b.b.'s. Attempt is made to ensure that the sizes of the b.b.'s in the list are near about the same. When the list becomes empty, it is assured that the sizes of all the (partitioned) b.b.'s is less than or equal to W. When this condition is satisfied no more partitioning needs to be done. If $W = 1$, then this is also the complete schedule and corresponds to a feasible design point. Otherwise, the design point found is an approximate one. If this point corresponds to a design with a better (lower) resource estimate than that of the best stored design then it replaces that design. The algorithm terminates with a failure if there exists a partition where the design parameters of *NFUS* and *NBUS* cannot be possibly satisfied.

The requirement for each resource is generated in the form of the a tuple $< m, w, j >$, where m is the number of units of that entity occurring in a window of size w in the b.b. j. Such tuples are generated for the maximum number of operations per time step, the bus requirement, the storage access point requirement and the requirement for hardware operator for each type of operation. $\lceil \frac{m}{w} \rceil$ is the l.b. on that resource. Tuples, instead of the resource requirement, are generated because this information is needed by the exploration heuristic used in the DSE tool (section 5.1).

The search mechanism explained above has one anomaly. The problem is that when REPS is being done with a relaxed time constraint then the search space turns out to be far larger than when REPS is being done with a tighter time constraint. This situation is addressed to by running an approximate scheduling algorithm on the current b.b. before going ahead to partition it into smaller b.b.'s. If the approximate scheduling algorithm terminates successfully then the current b.b. may be assumed to satisfy the l.b. on the resource estimate and need not be examined by the search mechanism any more.

4.2 Special Handling of Operations

REPS handles multi-cycle operations in the following manner. Suppose that the time frame of a multi-cycle operation of k time steps crosses the partition boundary set at time t. Up to k possibilities need to be examined. These are, initiating the operation at times earlier than time step $t - k$, initiating the operation at times $t - k, \ldots, t$, and at times later that t. Initiation of the operation at specific times is the additional overhead for handling multi-cycle operations. When more than a single operation crosses the partition boundary, partitioning is initiated with the operation requiring the least number of cycles for its execution. Handling of pipelined operations is as follows. Consider a p-stage pipelined implementation with a stage delay of d, of an operation of type x. The result of such an operation will be obtained $p-1$ time steps after initiation. Therefore, while scheduling the number of time steps to complete operations of type x should be taken a p. The l.b. is obtained as for a multicycle operation, the stage delay d being used in place of the number of time steps k of the multi-cycle operation.

5 Scheme for DSE

We now describe the overall scheme for design space exploration. At the heart of the DSE technique is the resource estimation and partial scheduling algorithm (REPS) which is repeatedly invoked with varying time constraints. The time constraints with which REPS is invoked is determined by the exploration heuristic described in section 5.1. With each invocation REPS either indicates that the time constraint is not feasible or it returns a design point and a schedule. When a new design point is obtained one of the three conditions will be true.

Dominated: If the point is dominated by existing design points then this design point is discarded.

Dominating : If the point dominates a set of the existing design points then all these are discarded and the new point is incorporated in the design space.

Non Dominated : If it neither dominates, nor is it dominated by other design points the it is simply incorporated in the design space.

If $W > 1$ then REPS will generally return a partial schedule and an approximate hardware requirement. In the latter case it is desirable to obtain the complete feasible schedules, which will be needed for performing subsequent allocation and binding. These schedules will have to be obtained using approximate scheduling algorithms.

5.1 Exploration Heuristic

The resource cost estimation scheme described above requires the number of time steps for each b.b. to be specified. To start with, the number of time steps for each DAG is set to its critical length, and then REPS is invoked. The resulting resource requirements are computed from the tuples, as explained above, and examined. It was mentioned that the requirement for each hardware resource or the requirement of FUs, buses, etc. are generated in the form of the a tuple $< m, w, j >$, where m is the number of units of that entity occurring in a window of size w in the b.b. j. In case any of the design parameters is violated a corrective action is taken as follows. Suppose that a design parameter X having the value v_X is violated, i.e. $\lceil \frac{m_X}{w_X} \rceil > v_X$. Consider the effect of adding $i, i > 0$, time steps to the DAG of the b.b. j_X. Now the earliest time of each operation o, $t_{a,o}$ remains unaltered, but $t_{l,o}$ goes up by i. Therefore, each operation previously restricted to lie in a window of size w will now lie in a window of size $w + i$. A minimal number of time steps $t_X > 0$ is added to w_X so that i.e. $\lceil \frac{m_X}{w_X + t_X} \rceil \leq v_X$. REPS is invoked after making the correction.

The DSE retains the set of mutually non-dominating design points that have been found. When a design point is found to be feasible it is compared with the stored design points. If is dominated by any point then it is not included in the set. If it dominates any point of the set then it replaces that point. Exploration continues with a new set of constraints, generated as follows. For each operation O whose requirement exceeds unity, we identify the DAG's where it is required maximally. In each of these DAG's we determine the time t by which the time constraint of that DAG should be relaxed so that the new requirement of the operator will be one less, i.e. $\lceil \frac{m_O}{w_O + t} \rceil = \lceil \frac{m_O}{w_O} \rceil - 1$. Let t_O be the maximum of all the times computed above. Let D_O be the DAG where this relaxation may be effected. Let O^* be the operation for which t_O has the minimum (non-zero) value of all the t_O's. Let D_{O^*} be the corresponding DAG.

We now relax the time constraint on the b.b. for D_{O^*} by t_{O^*} so that $\lceil \frac{m_{O^*}}{w_{O^*} + t_{O^*}} \rceil = \lceil \frac{m_{O^*}}{w_{O^*}} \rceil - 1$. This is the heuristic used to conduct the exploration of the design space. Exploration is terminated when the resource requirements of all the operations become unity.

5.2 Scheduling Schemes for Use with DSE

We have noted that the REPS generates ⟨*hardware cost, performance*⟩ estimates and a schedule for a given design input. For subsequent allocation and binding complete schedules are needed. Most of the existing scheduling algorithms, like FDLS [4], can be adapted to work with the partially scheduled DAG's generated by REPS. However, the performance of such modified heuristic algorithms may not match the performance of the original algorithm.

There is a second and more important aspect that needs to be addressed. It may be noted that the resource estimates are lower bounds and not exact estimates. It is, therefore, quite possible that for a time constraint and a set of hardware operators, as indicated by a design point, a feasible solution might not exist. Even if such a solution does exist, it might be missed out by the approximate scheduling algorithm. However, feasible solutions will be present in the neighborhood of a design point. We, therefore, resort to a systematic generation of schedules in the neighborhood of a design point reported by REPS and retained by the DSE mechanism as a non-dominated design point. Such a local exploration scheme should be capable of examining the neighborhood of a design point for feasible non-dominated solutions using approximate scheduling algorithms. The choice of polynomial time techniques here is emphasized, for otherwise an exact method could be used to obtain the schedule in the first place.

Thus after the first phase of DSE we have a set of design points. With each design point we also have the set of partitioned DAG's which had lead to its FU estimate component. At this juncture we complete the schedules of these partitioned DAG's using standard algorithms like FDLS [4] or the scheduling method proposed in [1]. The solutions obtained from this completion gives us upper bound (u.b.) estimates. If these match with the lower bound estimates obtained through DSE, we can terminate with accurate design points and schedules. On the other hand, if the u.b.'s and the l.b.'s differ, we explore around the estimated design point for feasible schedules leading to non-dominated <performance, FU requirement> design points. That is, we make limited search (in polynomial time) around the estimated design points obtained earlier.

6 Experimentation

The techniques proposed in this paper have been implemented and tested on some common examples like Facet [8], differential equation solver [5] and elliptic wave filter [2].

Tables 1,2 and 3 indicate the design points obtained after design space exploration of Facet, Diffeq. and Elliptic Wave Filter, respectively. All these designs are for single cycle implementations of the operations. The first two columns indicate design parameters. The design points obtained after design space exploration are indicated under "DSE", while the actual results obtained after allocation and binding are indicated alongside. While computing the costs of the FUs, the cost of each hardware operator is taken as follows: $\text{cost}(/) = 160, \text{cost}(\star) = 160, \text{cost}(+) = 20, \text{cost}(-) = 20, \text{cost}(<) = 10, \text{cost}(|) = 10$ and $\text{cost}(\&) = 10$. The allocation and binding has been done by the allocation and binding tool GABIND [3]. Each block of rows in a table indicates the design points obtained for a particular set of parameters. For Facet the design points obtained by DSE match the actual designs obtained after allocation and binding. This is also true for the elliptic wave filter example in table 3. For Diffeq. the actual implementation of the design points indicated in rows 2, 3 and 4 of table 2, require an additional adder in each case. For two FUs and seven time steps for Diffeq. the operations scheduled in three time steps were as follows: $< \star + >$; $< \star - >$; and $< + - >$. Therefore, although at most one $+$ and one $-$ are scheduled in any time step, it is not possible to have an FU configuration using two FUs where at least a $+$ or $-$ is not repeated. For the case with three FUs and seven time steps, however, such a problem did not exist. Yet an additional $+$ was used to keep the switch cost low. The implementation of Diffeq. using two FUs in seven time steps is especially nice, requiring only three switches. The design point indicated in row 2 is for designing with only two FUs, whereas there are four types of operations distributed over seven time steps. For the design point indicated in row 3 of table 2 the number of time steps is four, exactly equal to the length of the critical path in the data flow graph.

References

[1] A. Kumar, A. Kumar, and M. Balakrishnan. A novel integrated scheduling and allocation algorithm for data path synthesis. *Proceedings of VLSI Design '91, New Delhi*, pages 212–218, 1991.

[2] S Y Kung, H J Whitehouse, and T Kailath. *VLSI and Modern Signal Processing.* Prentice Hall, 1984.

[3] C A Mandal, P P Chakrabarti, and S Ghose. Allocation and binding for data path synthesis using a genetic approach. In *Proceedings of VLSI Design '96*, pages 122–125, 1996.

[4] P. G. Paulin and J. P. Knight. Algorithms for high-level synthesis. *IEEE Design & Test*, December 1989.

[5] Pierre G. Paulin. *High Level Synthesis of Digital Circuits Using Global Scheduling and Binding Algorithms.* PhD thesis, Carleton University, January 1988.

[6] C. V. Ramamoorthy, K. M. Chandy, and Mario J. Gonzalez. Optimal scheduling strategies in a multiprocessor system. *IEEE Transactions on Computer*, C-21(2):137–146, February 1972.

[7] B S Stewart and C C White. Multiobjective A*. *JACM*, 88(4):775–814, 1991.

[8] C. J. Tseng and D. P. Siewiorek. Automated synthesis of data paths in digital systems. *IEEE Transactions on Computer Aided Design*, CAD-5(3), July 1986.

		DSE			After Synthesis with GABIND		
num. of F.U.s	num. of bus	h/w opr. req.	F.U. cost	num. time steps	h/w opr. req.	F.U. cost	num. time steps
2	6	1⋆, 1/, 1+, 1-, 1&, 1—	380	5	1⋆, 1/, 1+, 1-, 1&, 1—	380	5
3	9	1⋆, 1/, 2+, 1-, 1&, 1—	400	4	1⋆, 1/, 2+, 1-, 1&, 1—	400	4
		1⋆, 1/, 1+, 1-, 1&, 1—	380	5	1⋆, 1/, 1+, 1-, 1&, 1—	380	5

Table 1: DSE results for Facet.

		DSE			After Synthesis with GABIND		
num. of F.U.s	num. of bus	h/w opr. req.	F.U. cost	num. time steps	h/w opr. req.	F.U. cost	num. time steps
2	6	2⋆, 1+, 1-, 1<	370	6	2⋆, 1+, 1-, 1<	370	6
		1⋆, 1+, 1-, 1<	210	7	1⋆, 2+, 1-, 1<	230	7
3	9	2⋆, 1+, 1-, 1<	370	4	2⋆, 2+, 1-, 1<	390	4
		1⋆, 1+, 1-, 1<	210	7	1⋆, 2+, 1-, 1<	230	7

Table 2: DSE results for Diffeq.

		DSE			After Synthesis with GABIND		
num. of F.U.s	num. of bus	h/w opr. req.	F.U. cost	num. time steps	h/w opr. req.	F.U. cost	num. time steps
3	9	2⋆, 3+	380	18	2⋆, 3+	380	18
		2⋆, 2+	360	19	2⋆, 2+	360	19
		1⋆, 2+	200	21	1⋆, 2+	200	21
		1⋆, 1+	180	27	1⋆, 1+	180	27

Table 3: DSE results for Elliptic Wave Filter.