MPT-based branch-and-bound strategy for scheduling problem in high-level synthesis

P.-Y. Hsiao
G.-M. Wu
J.-Y. Su

Indexing terms: Scheduling problem, High-level synthesis, MPT

Abstract: A branch-and-bound algorithm based on a maximum possibility table (MPT) is proposed to solve scheduling problems in high-level synthesis. Six efficient priority rules are developed as bounding functions in the algorithm. Extensions for real-world constraints are also considered, including chained operations, multicycle operations, mutually exclusive operations and pipelined data paths. Experimental results indicate that the MPT-based algorithm returns competitive scheduling results at significant savings in execution time as compared with other methods.

1 Introduction

High-level syntheses are performed to transform algorithmically specified behaviours of digital systems into register-transfer level structures [1, 2]. There are three major phases in high-level synthesis. The first phase, called compilation, compiles the formal language into an internal representation. Control/data flow graphs (CDFGs) are the most frequently used representations. The second phase, called scheduling, assigns the operations into control steps according to certain constraints while minimising corresponding objective functions. The third phase, called allocation, tries to share hardware resources, such as functional units, storage and communication paths, while minimising costs as much as possible. However, the interrelationship between scheduling and allocation is very close. An optimal schedule is not guaranteed to provide an optimal synthesis result after subsequent allocation operations. For practical considerations, using many interleaved scheduling and allocation iterations with a fast heuristic algorithm, instead of searching for a single optimal solution, will obtain a more satisfactory synthesis result [3].

During the last decade, many systems using different scheduling techniques have been proposed. When resources are unconstrained, the simplest scheduling technique, as soon as possible (ASAP), and the complementary as late as possible (ALAP), as shown in Fig. 1, may be used along with the corresponding description of the computation set from the example of the HAL [4] system. Scheduling problems with limited resources have already been shown to be NP-hard and thus, for most systems, analytic methodology must depend on heuristic techniques. The class of algorithms including SLICER [5] and MAHA [6] is based on list scheduling techniques. List scheduling manages all operations in topological order, using the precedences dictated by data and control dependencies in the CDFGs. The force-directed scheduling (FDS) [7] algorithm is the primary scheme within the HAL system. In this algorithm, the cost of hardware resources can be reduced by conducting operations concurrently. The ASCAM [8] algorithm considers evaluated costs as major concerns in updating probability values. The FAMOS [3] algorithm is based on the scheme proposed by Kernighan and Lin to escape local minima traps [9]. It also includes various selection functions developed to define hardware resource costs. A global scheduling algorithm with code-motions has been proposed by Rim [16]. Instead of using heuristic algorithms to schedule operations, mathematical descriptions of scheduling objectives and constraints are derived from ALPS [10] and can be successfully translated into integer linear programming (ILP) formulations. Therefore, ALPS obtains solutions at the expense of execution time because ILP is, by nature, an exponential-time algorithm.

```
xl = x+ dx;
ul = u- (3*x*u*dx)- (3*y*dx);
yl = y+ u*dx;
c = xl < a;
```

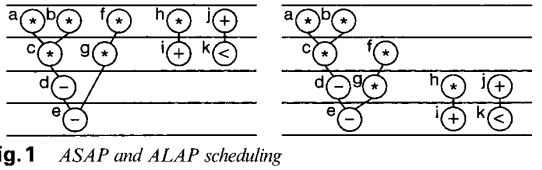

Fig. 1 *ASAP and ALAP scheduling*

In this paper, a branch-and-bound algorithm based on a maximum possibility table (MPT) [11] is proposed to solve scheduling problems in high-level synthesis. Six efficient priority rules are developed as bounding functions within our MPT-based algorithm.

2 Preprocess

For internal representations, we create maximum possibility tables (MPTs) instead of distribution graphs.

IEE Proceedings online no. 19982347

Paper first received 1st July 1996 and in final revised form 11th March 1997

The authors are with the Department of Computer and Information Science, National Chiao Tung University, Republic of China

MPTs indicate the maximum degrees of concurrence among similar operations in each control step. The content MPT is closely related to the mobility of operations. The following definition describes the maximum possible MPT value for each control step j, denoted as C-step.j.

Definition 2.1: Let *Poss(opn, j)* indicate whether an operation, denoted as *opn*, can be assigned in C-step.j or not. That is,

$$Poss(opn, j) == \begin{cases} 1 & \text{the mobility of } opn \text{ involves C-step.j} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Where *opn(type)* means the same type of operations, the maximum possible value of *opn(type)* for C-step.j can be derived from the following relationship:

$$MPT_{opn(type)}(j) = \sum_{opn(type)} Poss(opn, j) \quad (2)$$

Fig. 2 illustrates the mobility and MPTs derived from the ASAP and the ALAP schedulers. MPT* represents the MPT for multiplication operations, and for brevity, MPT+ corresponds to the MPT for addition, subtraction, and comparison. The MPT values indicate the maximum number of operations that can be active in each C-step. In addition, the zero in the MPTs indicates operations that must not be active in that C-step.

operation type	*							+	-	>			
node	a	c	b	f	g	h	MPT*	d	e	i	j	k	MPT*
C-step 1							4						1
C-step 2							4						3
C-step 3							2						4
C-step 4							0						3

Fig. 2 *MPT graph for Fig. 1*

Definition 2.2: Given a CDFG, the freedom of path *A* equals *n* if, and only if, the mobility of each operation within path *A* involves *n* C-steps.

For example, the freedom of path (*h*, *i*) in Fig. 1 equals 3.

Definition 2.3: The critical path of a given CDFG is defined as the single path of the CDFG with the longest delay time.

operation type	*					+	-	>	
node	b	f	g	h	MPT*	i	j	k	MPT*
C-step 1					3				1
C-step 2					4				3
C-step 3					3				3
C-step 4					2				3
C-step 5					0				2

Fig. 3 *MPT graph without critical path nodes*

For example, (*a*, *c*, *d*, *e*) or (*b*, *c*, *d*, *e*) can be considered to be critical paths, as shown in Fig. 1. Using the critical path (*a*, *c*, *d*, *e*) with four control steps, we now assume five control steps as the time constraint. This time constraint thus has one control step more than the critical path. MPT graphs without critical-path nodes can be constructed, as shown in Fig. 3.

Using these constructed MPTs, we first consider the maximal multiplier possibilities. C-step.2 has the maximal cost in MPT*. Therefore, whenever possible we must try to eliminate multiplication operations from C-step.2 when scheduling the critical path. The critical-path schedule for the five control-step time constraint is shown in Fig. 4.

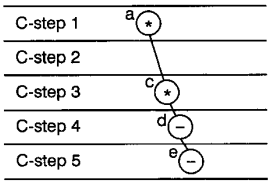

Fig. 4 *Critical path scheduling result*

When scheduling noncritical paths, we first partition the noncritical path nodes into several separate single paths according to their degrees of freedom. The nodes in paths (*b*), (*f*, *g*), (*h*, *i*) and (*j*, *k*) in Fig. 1 all have the same degrees of freedom.

Definition 2.4: For given CDFG paths *A* and *B*, a relation(~) exists if, and only if, an operation *i* in path *A* and an operation *j* in path *B* exist such that *j* accesses data from *i*. We denote this relation as *A*~*B*.

For example, we have (*b*)~(*a*, *c*, *d*, *e*) and (*f*, *g*)~(*a*, *c*, *d*, *e*) in Fig. 1. These paths are scheduled in the least-freedom first-served methodology. Specifically, when path *A* is scheduled, path *B* has the first scheduling priority if *B*~*A*.

Our algorithm was developed based on the branch-and-bound strategy. Every potential cost matrix, denoted as $M_{i,j}$, presents the latest assigned information after operation *i*(*opn.i*) has been assigned to C-step.*j*. The representation of the potential cost matrix is shown as follows:

$$M_{i,j} = \begin{bmatrix} m_{11} & m_{21} & \bullet & \bullet & m_{k1} \\ m_{12} & m_{22} & \bullet & \bullet & m_{k2} \\ \bullet & \bullet & \bullet & \bullet & \bullet \\ m_{1n} & m_{2n} & \bullet & \bullet & m_{kn} \\ M_{opn(1)} & M_{opn(2)} & \bullet & \bullet & M_{opn(k)} \end{bmatrix}$$

where *k* is the total number of types for all operations, *n* is the maximum number of control steps, m_{ij} indicates type *i* operations already assigned to C-step.*j* thus far and $M_{opn(i)}$ indicates the maximum number from m_{i1}, m_{i2}, ..., $m_{i,n}$.

For example, the corresponding potential cost matrix for the initial state of Fig. 4 is

$$M_{init} = \begin{bmatrix} + & * \\ 0 & 1 \\ 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

The value of $M_{i,j}$ presents the potential costs associated with the remaining operations in each C-step, providing very helpful referential information for the assignment of subsequent operations. Specifically, the value of the last row in $M_{i,j}$ indicates the actual costs consumed thus far. Therefore, $M_{i,j}$ should be updated after removing each assigned operation. However, there are

many possible alternative assignments for each operation, as the tree structure in Fig. 5 shows.

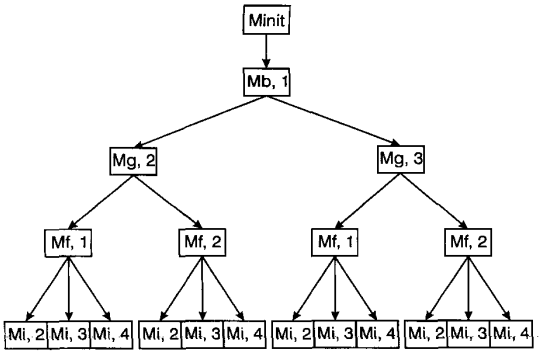

Fig.5 *Alternative assignments for each operation*

As shown in Fig. 5, M_{init} describes the distributions of operations after the critical path has been found. Associated with the CDFG shown in Fig. 1, where the time constraint is assumed to be four control steps, there is only one possible noncritical path assignment from M_{init} such as assigning *opn.b* to C-step.1. After *opn.b* has been manipulated, there are two available ways to assign the following *opn.g*, C-step.2 ($M_{g,2}$) or C-step.3 ($M_{g,3}$). After previous assignments have been made, there are two possible assignments for *opn.f* $M_{f,2}$ or $M_{f,1}$, for $M_{g,3}$ and $M_{g,2}$, respectively. Therefore, there are exponentially many possible paths to consider. To select the most efficient path, we propose a heuristic solution based on the following six priority rules and corresponding bounding functions to remove undesirable branches from the $M_{i,j}$ graph.

Table 1

Name of PR	Descriptive outline
Precedence constraint	(C-step.i) < (C-step.j) $\forall opn_i \rightarrow opn_j$, $S_i \le$ (C-step.i) $\le L_i$, $S_j \le$ (C-step.j) $\le L_j$
Lowest cost matrix among current candidates	min(COST($M_{i,j}$)),$S_i \le$ (C-step.i) $\le L_i$
Minimum value of MPT items	min(MPT(C-step.i)), $S_i \le$ (C-step.i) $\le L_i$
Lowest-cost ancestor matrix	min(COST(backtrack(1, $M_{i,j}$))), $S_i \le$ (C-step.i) $\le L_i$
Backtrack among ancestors until the MPT value can be determined	min(MPT(backtrack(> 1, $M_{i,j}$))), $S_i \le$ (C-step.i) $\le L_i$
Latest C-step	If candidates cannot be eliminated by R1 through R5, select matrix corresponding to latest C-step

where S_i, S_j: the C-step that opn.i or opn.j assigned to by ASAP.
L_i, L_j: the C-step that opn.i or opn.j assigned to by ALAP.
Min(): finding the minmum value.
COST(): evaluating the total cost.
MPT(): getting the element value of MPT.
Backtrack(no, $M_{i,j}$): Backtrack to no-th ancestor of $M_{i,j}$.

3 Priority rules and bound functions for scheduling

In this Section, we present detailed descriptions of six priority rules concerning the assignment of operations based on the derived representation from the previous Section. The corresponding bounding functions are also listed in Table 1. For convenience of presentation, ancestor matrices are represented as $M_{c,p}$, $M_{c,p+1}$, ..., $M_{c,q}$, and the operation mobility *opn.c* is between p and q. By sequentially applying these six priority rules (PR1 through PR6), we can guarantee that only one ancestor matrix, called the consulting matrix, should be referred. Using this consulting matrix, we can further construct the corresponding $M_{i,j}$ and update the MPT.

3.1 PR1: precedence constraint

Operation assignments must not violate data dependencies. That is, the associative relation between *opn.i* and *opn.j* in CDFG must be always preserved.

3.2 PR2: choose the lowest cost matrix from among current candidates

$M_{i,j}$ is made the consulting matrix if it has the lowest cost function among all candidate matrices, as determined by

$$COST(M_{i,j}) = M_{opn(1)} * cost_1 + M_{opn(2)} * cost_2 + \cdots + M_{opn(k)} * cost_k \quad (3)$$

where $cost_k$ is the area-associated cost of type k operation.

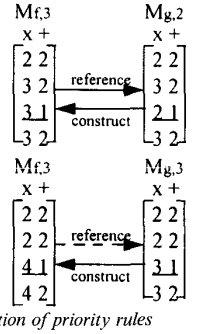

Fig.6 *Sample illustration of priority rules*

Fig. 6 illustrates construction $M_{f,3}$ by referring to $M_{g,2}$ or $M_{g,3}$, where *opn.f* is a multiplier needing to be assigned at C-step.3. Reference to $M_{g,2}$, shows that three multipliers and two adders will be consumed. However, reference to $M_{g,3}$, shows that four multipliers and two adders will be required. Precision demands selection of the former case.

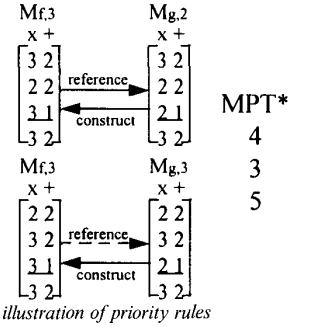

Fig.7 *Sample illustration of priority rules*

3.3 PR3: minimum value of MPT items

This rule selects the reference matrix with the minimum MPT value. This approach reduces the impact of unscheduled nodes as much as possible. In Fig. 7, we refer to $M_{g,2}$ and $M_{g,3}$ to construct $M_{f,3}$, with equivalent cost consumption. We can check the constructed

information representation MPT* to determine whether MPT*(3) = 5 is larger than MPT*(2) = 3; if so, we will select $M_{g,2}$ to construct $M_{f,3}$ to reduce the impact on nodes not yet scheduled.

3.4 PR4: lowest-cost ancestor matrix

This rule identifies the lowest-cost matrix from among ancestor matrices. As shown in Fig. 8, we refer to $M_{g,2}$ or $M_{g,3}$ to construct $M_{f,1}$. They have equivalent scheduled $M_{f,1}$ costs and MPT*. However, the cost consumption resulting from $M_{g,2}$ is lower than that resulting from $M_{g,3}$, therefore we will select $M_{g,2}$ to construct $M_{f,1}$.

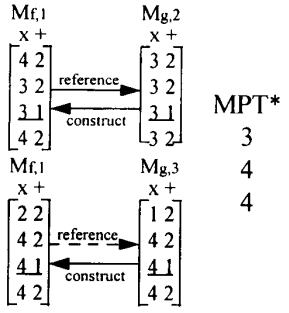

Fig. 8 *Sample illustration of priority rules*

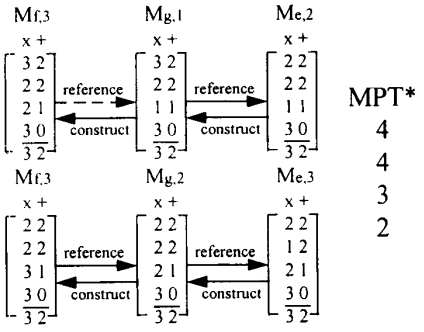

Fig. 9 *Sample illustration of priority rules*

3.5 PR5: backtrack among ancestors until the MPT value can be determined

This rule backtracks among ancestors of candidate matrices until PR3 is able to determine an appropriate selection. As shown in Fig. 9, we would like to construct $M_{f,3}$, but we cannot distinguish between $M_{g,1}$ and $M_{g,2}$ by applying PR1 through PR4. Hence, we backtrack among ancestor matrices to $M_{e,2}$ and $M_{e,3}$, where MPT(3) < MPT(2), after which the former can be abandoned.

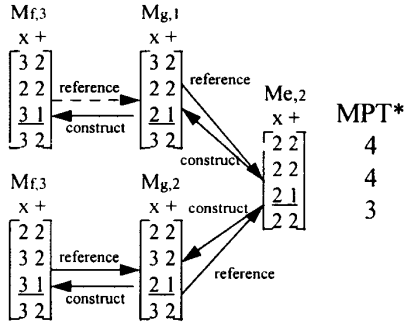

Fig. 10 *Sample illustration of priority rules*

3.6 PR6: latest C-step selection

If an appropriate selection cannot made using PR1 through PR5, the latest C-step is then considered. This rule is derived from the ALAP algorithm. As shown in Fig. 10, $M_{g,1}$ and $M_{g,2}$ have passed R1 through R5. We would then select $M_{g,2}$ to construct $M_{f,3}$ since its corresponding C-step occurred later than $M_{g,1}$.

A complete description of our algorithm is presented below. Note that the six priority rules must be checked in sequence.

Branch_Bound_SCHEDULE(){

Take the data flow graph, assignment delays and cost values of each operation;

Use ASAP and ALAP to determine the mobility of all operations;

Find the critical path;

Create MPTs for noncritical path nodes.

Schedule the critical path;

WHILE (there are nodes remain unassigned)

select a path A with the smallest degree of freedom;

SCHEDULE(A);

END

}

SCHEDULE(A)

BEGIN

select node i which is the leaf node of path A;

WHILE(i is not NULL)

FOR j = *mobility_start*(i) **TO** *mobility_end*(i) using PR1, PR2, ..., PR6 to select the consulting matrix;

IF (operation i is assigned to C-step j, all data dependencies are not violated)

THEN using consulting matrix to construct $M_{i,j}$ i = leaf node of path A; remove i from A; update the MPT;

END

WHILE (there are paths related to path A that are unscheduled) select one path B among them with the smallest degree of freedom;

SCHEDULE(B);

END

END

The result derived from our algorithm on the example from HAL is shown in Fig. 11. We assume that the available functional units are multipliers and ALUs (the ALU is capable of performing addition, subtraction and comparison), and that the multiplier costs are higher than the costs of the ALUs. It is not difficult to see that the critical path is (a, c, d, e). M_{init} is the initial potential cost matrix after the critical path has been scheduled. The initial MPT* = (3,3,2,0) and MPT$^+$ = (1,3,3,2); neither of these MPT graphs include the critical path. The arrows point out the matrices associated with the consulting direction. PR1, PR2, ..., PR6 along with the arrows are used as referential bases. When *opn.b* can only be assigned to C-step.1, we assign *opn.b* to C-step.1 and refer to M_{init} to construct Mb,1 with the first element in the second column increased by one

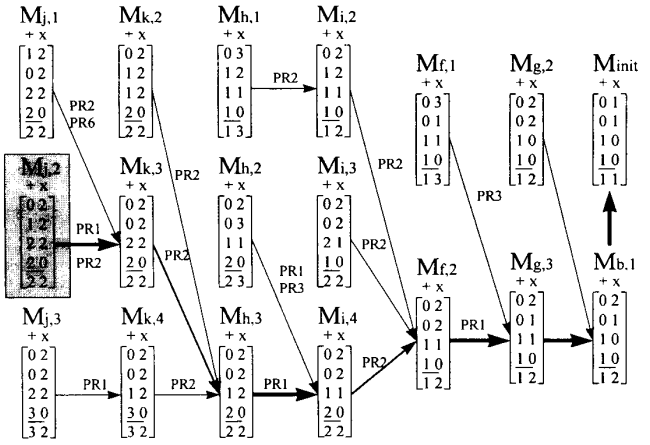

Fig. 11 *Procedures derived by our MPT-based algorithm for HAL example*

and the last element in the second column changed to 2. MPT* must be updated to (2,3,2,0). The *opn.f* can follow assignment to C-step.2 or C-step.1. Since it is assigned to C-step.2, it can refer by $M_{g,3}$ only, because according to PR1, *opn.g* and *opn.f* have data dependencies. As it is assigned to C-step.1, the candidate matrices $M_{g,2}$ and $M_{g,3}$ have the same costs as the candidate matrices. Now, MPT* = (2,2,1,0) and the third element is smaller than the second element. Therefore, $M_{f,1}$ can refer to $M_{g,3}$ according to PR3. Using the proposed methodology, the final solution is indicated by the shadowed block. We can backtrack indexes to obtain the optimal solution, as shown by the thick arrows; they are $M_{j,2}$, $M_{k,3}$, $M_{i,4}$, $M_{f,2}$, $M_{g,3}$, $M_{b,1}$. That is, two multipliers and two ALUs are required for the implementation.

For the computational complexity consideration of the proposed algorithm, the associated MPTs can be created in linear time and initialisation of the critical path also consumes linear time. Let S be the number of control steps and N be the total number of noncritical path nodes. Consider PR5 is backtracking behaviour, which may require $O(N)$. In the worst case, the entire time complexity will be bounded by $O(S^2N^2)$.

4 Extensions for real-world constraints

In previous Sections, we assumed that one clock cycle is required for each operation, but that is not always true in real situations. In this Section we must extend the previous algorithm to accommodate real world situations.

The conditional construct is similar to the 'if-then-else' or 'case' statement within the programming language. Inherently, it results in several mutually exclusive branches. Mutually exclusive operations can easily be handled by referring to them as only one operation if they use the same type of functional unit, even though they are assigned to the same control step.

Operations that require multiple control steps to execute are also considered. For nonpipelined implementation, when the operation is assigned, the functional unit cannot be shared by other operations until the assigned operation has been completed. Assume that $M_{i,j}$ indicates that operation i is necessarily assigned from C-step.j to C-step.$(j + d_i - 1)$, where d_i represents the propagation delay of operation i. Then, a new MPT, called MC-MPT, must be redefined as follows:

$$MC - MPT_{type(c),j} = \sum_{k=j}^{j+d_i-1} MPT_{type(c),k}$$

Therefore, PR3 and PR5 must be modified to choose the smallest MC-MPT. Specifically, the corresponding hardware use costs must be modified for each propagation C-step following assignment.

For functional units that provide chaining functions, we can chain several operations in one cycle if their total running time is less than the cycle time. We can further formulate a chaining problem as a multicycle problem by reducing the cycle time. The greatest common divisor of the functional unit's propagation delay time is selected as the new cycle time; a multicycle strategy can then be used to solve the chaining problem.

For pipelined data-path systems, execution of multiple tasks can be conducted concurrently. For a given latency L, operations assigned to C-step$(i + pL)$ (for $p = 0, 1, 2, ...$ and $i = 0, 1, 2, ..., L - 1$) cannot share a functional unit because they are executed simultaneously. Consequently, the MPT in PR3 and PR5 must be changed to the sum of MPTs (SMPT) over C-step $j + pL$ ($p = 0, 1, 2, ...$) as follows:

$$SMPT_{type(c),j} = \sum_{(r-j) \bmod L = 0, 1 \le r \le s} MPT_{type(c),r}$$

where

s denotes the number of C-steps

PR3 is then modified as follows:

$$SMPT_{type(c),j1} = \min\left(SMPT_{type(c),p+k}\right)$$
$$0 \le k \le pL$$

Table 2: Information on computers

Method	Computer	MIPS
MAHA [6]	VAX-11/750	1
FDS [7]	Xerox 1108 LISP machine	n/a
ALPS [10]	VAX-11/8800	12
FAMOS [3]	SUN 4/280	10
SEHWA [15]	VAX-11/750	1
Kung [12]	n/a	n/a
Komi [13]	SUN SPARC-2	25
Achatz [14]	SUN SPARC	12.5
MPT [ours]	SUN SPARC-IPC	14

Table 3: Experimental results for MAHA example

System	Cycles	Oper/cycle	Adds	Muls	CPU time (unnormalised)	CPU time (normalised)
MAHA	8	1	1	1	160s	2857
	4	3	2	3	160s	2857
FDS	8	1	1	1	50s	n/a
	4	2	2	2	25s	n/a
	3	3	3	3	35s	n/a
APLS	8	1	1	1	0.26s	55.7
	4	2	2	2	0.08s	17.1
	3	3	3	3	0.23s	49.3
FAMOS	8	1	1	1	0.033s	5.9
	4	2	2	2	0.05s	8.9
	3	3	3	3	0.033s	5.9
MPT (ours)	8	1	1	1	0.004s	1
	4	2	2	2	0.004s	1
	3	3	3	3	0.004s	1
	2	4	4	4	0.004s	1

5 Experimental results

Our MPT-based system has been implemented and tested in C language on a Sun SPARC-IPC. To verify the efficiency of the proposed algorithm, results from other approaches are described, and are compared with ours in Tables 3–6. For comparison with other methods, we have normalised the CPU time for each method using the MIPS information shown in Table 2. Note that the required resource results for ALPS in these Tables are optimal, since they are obtained using ILP.

The first example, taken from MAHA, contains chained operations involving fewer than eight cycles. The experimental results for this example are summarised in Table 3. The same required resource cost optimisations can be derived using FDS, ALPS, FAMOS and our MPT-based approach. However, a significant time saving is achieved by our MPT-based algorithm.

Table 4: Experimental results for FDS example

System	Cycles	Muls	ALUs	CPU time (unnormalised)	CPU time (normalised)
FDS	6	3	2	15s	n/a
	7	2	2	35s	n/a
APLS	6	3	2	0.17s	48.6
	7	2	2	0.28s	60
FAMOS	6	3	2	0.016s	3.8
	7	2	2	0.033s	5.9
MPT (ours)	6	3	2	0.003s	1
	7	3	1	0.004s	1
	8	2	2	0.007s	1
	9	2	1	0.009s	1
	14	1	1	0.21s	1

The second example is differential equation taken from FDS that contains multicycle operations and has a critical path six cycles long. In this example, multiplication was assumed to have a delay of two cycles, while addition was given a delay of half a cycle. The corresponding scheduling results for the second example are shown in Table 4; our MPT-based extension algorithm for multicycle operations derived in section IV was applied for resource scheduling. Considering resource costs, our MPT-based approach had one multiplier growth and one ALU decrement for the case involving seven cycles. However, a significant time saving was also achieved by our MPT-based algorithm.

Table 5: Experimental results for fifth order elliptic filter example

System	Cycles	Muls	ALUs	CPU time (unnormalised)	CPU time (normalised)
Kung	17	4	4	n/a	n/a
Komi	17	3	3	4.2s	883
	18	2	2	15.3s	2101
	21	2	1	22.2s	1723
Achatz	17	3	3	1s	99.2
	18	2	2	5s	343
	21	2	1	26s	1009
FDS	17	3	3	60s	n/a
	18	3	2	180s	n/a
	19	2	2	420s	n/a
	21	2	1	780s	n/a
APLS	17	3	3	0.26s	24.8
	18	2	2	3.1s	204
	21	2	1	34.5s	1285
FAMOS	17	3	3	0.067s	5.3
	18	2	2	0.101s	5.6
	21	2	1	0.783s	24.3
MPT (ours)	17	3	3	0.009s	1
	18	2	2	0.013s	1
	21	2	1	0.023s	1
	28	1	1	0.108s	1

The third example is a fifth-order elliptic digital filter [12] that contains 26 additions and eight multiplications. For this example, multiplication was assumed to take two cycles, and addition was assumed to have a one-cycle requirement. The critical path was 17 cycles long. The MPT-based solution obtained for 17 control steps is shown in Fig. 12; it requires three multipliers

Table 6: Experimental results for SEHWA example

System	Cycles	Adds	Muls	CPU time (unnormalised)	CPU time (normalised)
SEHWA (feasible)	7	6	3	n/a	n/a
SEHWA (exhaust)	6	5	3	10 min	2381
FDS	6	5	3	30s	n/a
ALPS	6	5	3	0.32s	15.2
FAMOS	6	5	3	0.05s	2
MPT (ours)	6	5	3	0.018s	1

and three ALUs. All scheduling results for the third example are shown in Table 5, along with two other ILP-based methods derived by H. Kmi *et al.* [13] and Achatz [14], respectively. Our MPT-based algorithm was able to obtain the optimal solutions for each critical-cycle consideration in significantly less execution time.

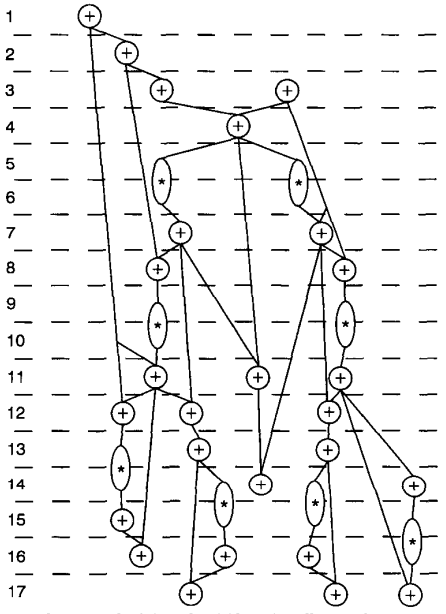

Fig. 12 *17 C-steps scheduling for fifth-order elliptic filter*

The fourth example is a pipelined 16-point digit FIR filter adopted from SEHWA [15]. For this example, multiplication was assumed to take 80ns and addition to take 40ns. Cycle time was 100ns. The latency for the pipelined data path was equal to three cycles. The corresponding scheduling results for this example are shown in Table 6. A significant time saving was achieved by our MPT-based algorithm, and an optimal solution was obtained.

Average normalised execution time can be expressed as geometric mean. The formula for the geometric mean is

$$\sqrt[n]{\prod_{i}^{n} execution\ time\ ratio_i}$$

where execution time $ratio_i$ is the execution time, normalised to the reference system, for the ith example of a total of n in the workload. Comparing the time performance of systems with geometric mean, our algorithm is 2875 times than MAHA, 103 times than APLS, 6 times than FAMOS, 1473 times than Komi, and 325 times than Achatz. The comparisons presented in these examples show the ALPS scheduling algorithm to be an optimal approach to using the ILP method. Moreover, our MPT-based algorithm can achieve near optimal scheduling of hardware resources in significantly shorter CPU times than any of the previous approaches we tested.

6 Conclusions

We create MPT graphs instead of the distribution graphs used in most previous scheduling systems. The MPT graphs indicate maximal degrees of concurrence among the similar operations in each control step. We preprocess the nodes among the critical path using a simple method that removes a subset of the total number of nodes from the computationally more expensive parts of the algorithm. The kernel of our method can be classified as a branch-and-bound algorithm. To improve the computation time, we propose six priority rules to serve as our bounding functions. Extensions for real-world constraints are also considered in our algorithm, including chained operations, multicycle operations, mutually exclusive operations, and pipelined data paths. Our algorithm guarantees to achieve near-optimal hardware resources usage in significantly shorter CPU times than other approaches tested.

7 References

1 MCFARLAND, M.C., PARKER, A.C., and CAMPOSANO, R.: 'The high-level synthesis of digital systems', *Proc. IEEE,* 1990, **78**, pp. 301–318

2 DE MICHELI, G.: 'Synthesis and optimization of digital circuits' (McGraw-Hill, 1994)

3 PARK, I.C., and KYUNG, C.M.: 'FAMOS: an efficient scheduling algorithm for high-level synthesis', *IEEE Trans.,* 1993, **CAD–12**, pp. 1437–1448

4 PAULIN, P.G., KNIGHT, J.P., and GIRCZYC, E.F.: 'HAL: A multi-paradigm approach to automatic data path synthesis'. Proceedings of 23rd *Design automation* conference, 1986, pp. 263–270

5 PANGRLE, B.M., and GAJSKI, D.D.: 'Slicer: a state synthesizer for intelligent silicon compilation'. Proceedings of IEEE international conference on *Computer design*, 1987

6 PARKER, A.C.: 'MAHA: A program for datapath synthesis'. Proceedings of 23rd *Design automation* conference, 1986, pp. 263–270

7 PAULIN, P.G., and KNIGHT, J.P.: 'Force-directed scheduling for the behavioural synthesis for ASIC's', *IEEE Trans.,* 1989, **CAD–8**, pp. 661–679

8 CIVERA, P., MASERA, G., PICCININI, G., and ZAMBONI, M.: 'Algorithms for operation scheduling in VLSI circuit design', *IEE Proc.,* 1993, **140**, pp. 339–346

9 KERNIGHAN, B.W., and LIN, S.: 'An efficient heuristic procedure for partitioning graph', *Bell Syst. Tech. J.,* 1970, **49**, (2), pp. 291–308

10 HWANG, C.-T., LEE, J.-H., and HSU, Y-C.: 'Formal approach to the scheduling program in high-level synthesis', *IEEE Trans.,* 1991, **CAD–10**, pp. 464–475

11 HSIAO, P.Y., WU, G.M., HO, M.H., and CHANG, C.J.: 'MPT_based scheduling for high-level synthesis'. International symposium on *VLSI technical systems and applications*, 1995, pp. 63–67
12 KUNG, S.Y., WHITEHOUSE, H.J., and KAILATH, T.: 'VLSI and modern signal processing' (Prentice Hall, Englewood Cliffs, NJ, 1985), pp. 258–264
13 KOMI, H., YAMADA, S., and FUKUNAGA, K.: 'A scheduling method by stepwise expansion in high-level synthesis'. Proceedings of IEEE conference on *Computer-aided design*, 1992, pp. 234–237
14 ACHATZ, H.: 'Extended 0/1 LP formulation for the scheduling problem in high-level synthesis'. Proceedings of EDAC-93, 1993, pp. 226–231
15 PARK, N., and PARKER, A.C.: 'SEHWA: a software package for synthesis of pipelines from behavioural specifications', *IEEE Trans.*, 1988, **CAD–7**, pp. 356–370
16 RIM, M., FANN, Y., and JAIN, R.: 'Global scheduling with code-motions for high-level synthesis applications', *IEEE Trans.*, 1995, **VLSI–3**, pp. 379–392